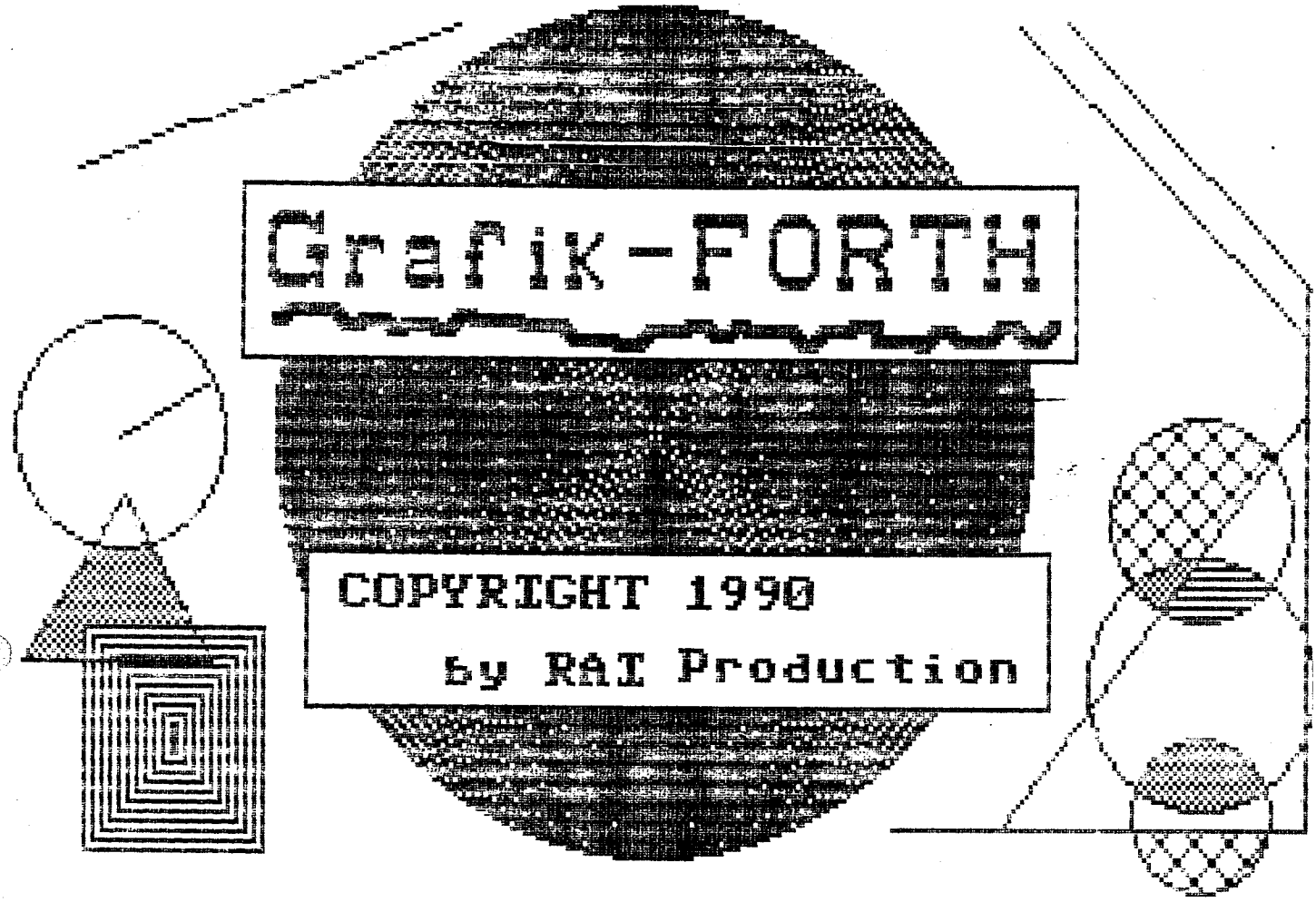


DIE ZUKUNFT

HAT

Testexemplar!
(blaue Tinte)

BEGONNEN!

Inhaltsverzeichnis

Grafik-FORTH

Nach viel harter Arbeit liegt Ihnen nun die, wie ich meine, beste Programmiersprache vor, um schnelle Grafik im hochauflösenden Modus zu programmieren. Grafik-Forth ist nicht nur eine Kombination von "fig-FORTH 1.4S" und TURBO-GRAPHICS-SYSTEM 256, sondern eine Weiterentwicklung, eine neue Programmiersprache.

Warum FORTH als Stammsprache?

Diese Frage werden sich bestimmt viele Leute stellen, die bisher noch nicht in FORTH programmiert haben. Ich habe es bewußt als Stammsprache gewählt, da es 10 bis 100 mal so schnell als BASIC ist und dabei weniger Speicherplatz als vergleichbare Assembler-Programme verbraucht! Ein FORTH-Programm braucht in der Regel nur einviertel des Speicherplatzes eines BASIC-Programms! Der größte Vorteil ist aber die leichte Erweiterbarkeit und die einfache Programmierung.

Der Editor

Der Editor ist ein Full-Screen-Editor. Man kann also bequem mit dem Cursor an die Stelle fahren, die man editieren will. Trotzdem hat der Editor viele Möglichkeiten zur Textmanipulation. Man kann nach Wörtern suchen, Abschnitte kopieren usw..

Sinn und Zweck dieses Handbuches

Der Sinn dieses Handbuches ist nicht dem Leser die Programmierung in der Sprache FORTH beizubringen. Der Hauptzweck ist dem Leser die Besonderheiten dieser FORTH-Version näher zu bringen. Denjenigen, die die Sprache FORTH lernen möchten, schlage ich das sehr gute Buch "Programmieren in FORTH" (327 Seiten) von Leo Brodie vor, welches im Carl Hanser-Verlag erschienen ist. In der Dokumentation von fig-FORTH 1.4S wird dieses Buch ebenfalls empfohlen. Die Bestellnummer lautet: 3-446-14070-0. Da dieses Buch 48,- DM kostet schlage ich Ihnen auch die Bücher "FORTH-Handbuch" (Hofacker Verlag, Bestellnummer 137) und "FORTH on the ATARI" (Hofacker Verlag, Bestellnummer 170, englisch!) vor. Das erste Buch hat früher einmal 49,- DM gekostet und kostet heute nur noch 5,- DM und hat 148 Seiten, das zweite kostet 9.80 DM und hat 118 Seiten. Da ich noch keines der beiden gelesen habe, kann ich nichts über deren Qualität sagen. Der Autor beider Bücher ist E. Flögel. Die letzten beiden genannten Bücher sind speziell für fig-FORTH (Grafik-FORTH ist eine fig-FORTH Version), während "Programmieren in FORTH" mehr auf FORTH-79 abgestimmt ist. Wer weiterführende Literatur sucht, dem schlage ich das Buch "In FORTH denken" von Leo Brodie vor, welches Hanser Verlag erschienen ist (ISBN 3-446-14334-3, 48,- DM, 299 Seiten). Alle Angaben

ohne Gewähr!

Ich bitte Sie die COPYRIGHT-Rechte zu beachten. Ein wesentlicher Grund warum sowenig neue Software für die 8-Bitter rauskommt ist, daß soviel raubkopiert wird!

Ich danke besonders dem Softwarehaus Sailer und hoffe, daß alle viel Freude und Erfolg mit dieser Programmiersprache haben werden.

Rainer Hansen

1. Die Grafik-FORTH-Diskette

Grafik-FORTH laden

Legen Sie die Grafik-FORTH-Diskette mit Seite 1 nach oben in die Diskettenstation 1 ein, und schalten Sie den Computer an. Während des Ladens ist der Bildschirm zur Beschleunigen des Ladevorgangs "abgeschaltet". Nach dem Laden wird das Titelbild ausgegeben, woraufhin man die START-Taste drUckt; nun können Sie Worte eingeben. Auf der 2. Seite befindet sich eine gekürzte Version von Seite 1, die den Vorteil hat, daß mehr Speicherplatz frei bleibt. Zusätzlich befinden sich auf der Diskette Beispielprogramme, wie z.B. das Listing vom Editor. Auf der fig-FORTH-Diskette befindet sich das Public Domain fig-FORTH 1.4S, und auf der letzten Diskette befindet sich DOS 3 mit dem Programm BEREICH.CMD, sowie auf der RUckseite Grafik-FORTH Teil 3.

Diskettenformat

FORTH arbeitet normalerweise mit einem speziellen Diskettenformat. Die Diskette wird in Bereiche von einem KByte (1024 Bytes) unterteilt. Ein solcher Bereich wird Block genannt. Die Blocknummern können zwischen 0 und bis zu 89 (Single-Density), 129 (Medium-Density), 178 (Double-Density) oder 358 (Double-Sided/ Double-Density) liegen. Wenn Sie Blöcke listen wollen, geben Sie einfach "Blocknummer LIST <RETURN>" ein. Dieses Blockformat hat viele Vorteile, so lassen sich auf einer FORTH-Diskette bei Medium-Density (DOS 2.5-Format) 3 KByte mehr Daten speichern als bei einer DOS 2.5-Diskette. Durch das Aufteilen in Blöcke wird außerdem der Programmierer zum Programmieren von kürzeren Definitionen ermuntert. Nebenbei gesagt, laufen Lesen und Schreiben schneller als bei DOS 2.5 ab. Durch den Befehl DOUBLE DENSITY wird es möglich mit Double-Density Disks zu arbeiten, wenn die Floppy ebenfalls auf Double-Density eingestellt ist. Durch den Percom-Block kann man die Floppy entsprechend einstellen. Wie das Konfigurieren geschieht wird später besprochen.

Wenn Blöcke von der Diskette gelesen werden, werden sie in einem der beiden Blockpuffer gespeichert. Durch den Befehl BLOCK wird ein Block, wenn er sich noch nicht in einem Blockpuffer befindet, in einen geladen, und die Adresse des ersten Bytes vom Block wird auf den Stack gelegt. Falls vorher in dem Puffer ein Block gespeichert war, wird er vorher abgespeichert, wenn er als geändert markiert war.

Inhaltsverzeichnis

Das Inhaltsverzeichnis wird durch s d INDEX angezeigt, wobei s die Anfangs- und d die Endblocknummer ist. Genaugenommen wird jeweils die erste Zeile des jeweiligen Blocks ausgegeben.

Percom-Block

Durch den 12 Bytes langen Percom-Block kann man die Floppy softwaregesteuert konfigurieren. Die einzelnen Bytes haben folgende Bedeutung:

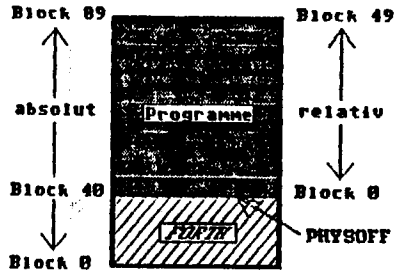
		SD	ED	DD
Byte 0:	Anzahl der Spuren	040	040	040
Byte 1:	Schrittrate			
Byte 2:	Sektoren/Spur (H)	000	000	000
Byte 3:	Sektoren/Spur (L)	018	026	018
Byte 4:	Seitenzahl-1			
Byte 5:	Density	000	004	004
Byte 6:	Bytes/Sektor (H)	000	000	001
Byte 7:	Bytes/Sektor (L)	128	128	000
Byte 8:	Station ON LINE			
Byte 9:	Übertragungsrate			
Byte 10:	Reserviert			
Byte 11:	Reserviert			

Floppy konfigurieren

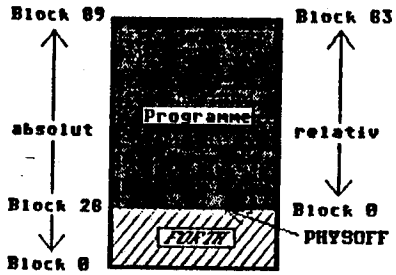
Den Percom-Block gibt es z.B. bei "aufgerUsteten" 1050ern oder bei der neuen XF551 (zweiseitiges Laufwerk). Die 1050 "erkennt" nach dem Einlegen einer Diskette automatisch das Format, während man bei der XF 551 das Format nur durch die softwaremäßige Konfigurierung des Percom-Blocks verändern kann. Als erstes läd man, soweit noch nicht vorhanden, die Worte fUr die Benutzung des Percom-Blocks von Block 24 (Seite 1), durch 24 LOAD. Danach gibt man READ PERCOM PERC DUMP <RETURN> ein, und die 12 Percombytes werden ausgegeben. Durch b ! PERCOM! ändert man die entsprechenden Bytes (z.B. 4 5 PERCOM! speichert in Byte 5 den Wert 4). Zum Schluß schickt man alles mit WRITE PERCOM zur Floppy zurUck. Die Befehle SINGLE DENSITY" und "DOUBLE DENSITY" stellen den Computer auf die richtige Sektorlänge ein. Dieses geschieht durch das verändern des Wertes von der Benutzervariablen OFFSET. FUr den Computer auf Medium-Density einzustellen benutzt man ebenfalls den Befehl "SINGLE DENSITY", da die Sektorlänge von Single-Density gleich der von Medium-Density ist.

Damit man nicht beim Speichern versehentlich bei einer Kopie von der Originaldisk Teile von FORTH Uberschreibt, gibt es die Benutzervariable PHYSOFF. In ihr ist die erste Blocknummer enthalten, die gelesen oder beschrieben werden kann. Die Nummer, die man z.B. bei LIST angibt, ist immer eine relative dazu. Bei Grafik-FORTH ist PHYSOFF auf 40 (Teil 1 + 2, bei Teil 3 ist PHYSOFF 0) gesetzt, was bedeutet, wenn Sie 0 LIST eingeben, eigentlich Block 40 der Diskette listen.

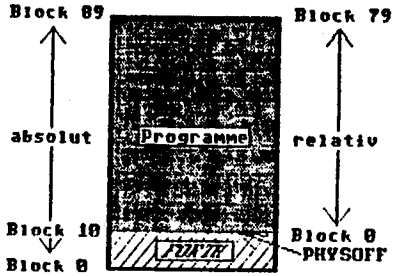
Disketten-Aufteilung
bei Grafik-FORTH



Disketten-Aufteilung
bei fig-FORTH 1.4S Teil 1



Disketten-Aufteilung
bei fig-FORTH 1.4S Teil 2



Der Wert von PHYSOFF spielt nur bei Single-Density eine Rolle, während er bei Double-Density keinen Einfluß hat.

DOS 2 und FORTH

Was machen, wenn man Programme, die unter DOS 2 abgespeichert sind, in FORTH laden will? Die Lösung für diese Frage heißt DOS 3! Dieses viel geschmähte DOS benutzt das gleiche Blockformat wie FORTH (1024 Bytes/Block) und im MenÜ gibt es den Punkt "Access DOS 2". Mit diesem Programm kann man DOS 2-Files in das DOS 3-Format bringen. Nachdem man dieses getan hat, geht man ins MenÜ und wählt die Funktion "X user-defined". Nun legt man die DOS 3-Diskette, soweit nicht schon vorhanden, in das Laufwerk und drÜckt <RETURN>. Es wird gefragt ob man BEREICH laden will und man drÜckt <Y><RETURN>, woraufhin das Programm geladen wird. Mit ihm kann man sich die Nummern der Blöcke anzeigen lassen, die das entsprechende Programm belegt. Der Dateiname wird ohne "D:" aber mit Extender eingegeben! Diese Nummern hält man sich am besten auf einem StÜck Papier fest und man kann dann von FORTH aus die entsprechenden Blöcke laden (FORTH-Diskette einlegen und <BREAK>-Taste drÜcken. Im DOS 3-MenÜ den Punkt "Go at hex :addr" auswählen und E477 als Adresse eingeben. Schon fÜhrt der Computer einen sogenannten Kaltstart durch und Grafik-FORTH wird geladen).

Mit BLOCK kann man in FORTH die entsprechenden Blöcke laden und man muß sie nur noch an die richtige Adresse kopieren und starten. Achtung: Die Blocknummern, die ausgegeben werden, gelten fÜr PHYSOFF=3 (3 PHYSOFF !), und die ersten 6 Bytes eines Maschinensprachefiles sind keine Programmdaten, sondern sind zwei Bytes mit dem Wert 255, die Anfangsadresse und die Endadresse. Bei zusammengesetzten Files kann dieser Fileheader mehrmals vorkommen.

Autostart

Es gibt bei DOS 2 eine AUTORUN.SYS-Datei, die automatisch geladen wird, wenn der Computer angeschaltet wird. Bei Grafik-FORTH gibt es die Variable 'AUTO, die die Codefeldadresse (Cfa) eines Wortes enthält. Gibt man z.B. AUTO @ EXECUTE ein, so wird das Wort, dessen Cfa in AUTO gespeichert ist, ausgefÜhrt. Im Normalfall zeigt AUTO auf das Wort CR (WagenrÜcklauf). Das Wort auf das AUTO zeigt wird ausgefÜhrt, wenn folgendes eingetreten ist:

a. der Computer eingeschaltet wurde
b. die RESET-Taste wurde gedrÜckt
c. ABORT oder RESET aufgerufen wurden
d. die BREAK-Taste gedrÜckt wurde
e. WARNING hat einen negativen Wert und ein Fehler ist aufgetreten

Jedesmal wenn also "Grafik-FORTH" ausgegeben wird, wird auch das Wort, auf das AUTO zeigt, ausgeführt. Man kann z.B. die Cfa von einem anderen Wort in AUTO speichern, was dann bei den oben genannten Fällen ausgeführt wird. Ein solches Wort fängt am besten mit dem Wort CR an. Hier nun ein Beispiel, was das ganze demonstriert:

Angenommen Sie wollen, daß bei jedem der oben genannten Fälle das Wort HALLO ausgeführt wird, welches Ihren Namen ausgibt. HALLO ist wie folgt definiert:

```
: HALLO    CR ." <Ihr Name>" CR ;
```

Sie speichern einfach nur die Cfa von HALLO in AUTO:

```
' HALLO CFA  AUTO !
```

Falls Sie HALLO z.B. nur nach dem Booten ausführen lassen wollen, so definieren Sie ein Wort, welches nach der Ausführung die Cfa von CR in AUTO speichert, so daß bei weiteren Aufrufen von RESET nun nur ein CR ausgegeben wird. Hier ein Beispiel:

```
: HALLO'   HALLO   ' CR CFA  AUTO ! ;   ' HALLO' CFA  AUTO !
```

Wo ist was gespeichert?

Auf der ersten Grafik-FORTH-Diskettenseite befinden sich folgende Programme:

- Block 0: Inhaltsverzeichnis
- Blöcke 2-11: Fehlermeldungen
- Blöcke 12-14: Auf diesen Blöcken befinden sich die "Systemworte". MAKEBOOT schreibt die momentan im Speicher befindliche FORTH Version auf eine Diskette. Wenn man z.B. gewisse Wörter immer wieder von der Diskette laden muß, ist es praktischer sie einmal zu laden, danach 12 LOAD einzugeben, um MAKEBOOT zu laden, und dann mit MAKEBOOT alles auf Diskette zu speichern. Diese neue FORTH Version können Sie dann einfach durch das Booten der Diskette laden. FORMAT (Drives --) formatiert, wie der Name schon sagt, eine Diskette. Der Quelltext für dieses Wort steht in Block 14. Man kann zwischen drei Formaten auswählen:
  - Single Density:
    + 128 in 725 speichern (DECIMAL 128 725 ! (Voreinstellung))
    + 02100 an die entsprechende Stelle im Quelltext schreiben (Voreinstellung)
  - Enhanced Density:
    + 128 in 725 speichern (DECIMAL 128 725 ! (Voreinstellung))
    + 02200 an die entsprechende Stelle im Quelltext schreiben
  - Double Density:
    + 256 in 725 speichern (DECIMAL 256 725 !)
    + 02100 an die entsprechende Stelle im Quelltext schreiben (Voreinstellung)

Wenn Ihnen die Bildschirmfarben nicht gefallen, können Sie andere Werte in die entsprechenden Speicherstellen speichern:

- Schrifthelligkeit: Speicherstelle 709 (0-15)
- Hintergrundfarbe: Speicherstelle 710 (0-255)
- Rahmenfarbe: Speicherstelle 712 (0-255).

Falls noch nicht geschehen mit 12 LOAD die Systemworte laden! Nun geben Sie SETSYS ein, womit die Farben und die Bildschirmränder "notiert" werden:

SETSYS

Erklaerung!

Die Mitwirkenden:

Interpreter

SETSYS

SETSYS

SETSYS

Wenn der 

die Buchstaben SETSYS findet, weckt er  .

 sucht nach

den Werten fuer die Bildschirm-raender und die Farben.

Wenn er sie gefunden hat, schreibt er sie ins Systembuch.

* Blöcke 15-21: Ein Decompiler, der mit 16 LOAD geladen werden kann. Durch DEC Name kann ein Wort decompiliert werden. Durch "odr DIS" kann Maschinencode ab der Adresse adr disassembliert werden. Wenn man entweder eine Taste (nicht <CONTROL>+<1>) drückt oder der Computer auf ein JMP, RTS oder RTI stößt, so wird das dekompelieren beendet. #DIS dient zum disassemblieren von einer bestimmten Anzahl von Befehlen ab einer bestimmten Adresse (z.B. ' U/ 4 #DIS).

* Block 24: Befehle für den Percom-Block, die sich alle auf Laufwerk 1 beziehen!

* Blöcke 27-29: Befehle für die Ein- und Ausgabe. Mit LPOPEN wird Kanal 7 für den Drucker geöffnet. Mit IOCB wird der Kanal gewählt, auf den sich alle folgenden Befehle beziehen (aktueller Kanal). CLOSE schließt den aktuellen Kanal und OPEN (Operation Hilfscode Gerät --) öffnet ihn. Als Geräte kann man entweder E: (Editor), S: (Bildschirm - nur Ausgabe), K: (Tastatur - nur Eingabe), C: (Kassette) und P: (Drucker - nur Ausgabe) wählen. Mit PUT (b --) kann man ein Byte an ein Gerät senden und mit GET (-- b) eins empfangen. Mit BPUT (a # Akkuwert --) kann man Speicherbereiche senden und mit BGET (a # --) empfangen. LYPE (a # --) ist das TYPE-Kommando für den Drucker. Mit CRLP wird an den Drucker der Befehl Zeilenende geschickt. LINELP (Zeile# SCR# --) gibt eine Zeile auf einem Drucker aus. LISTLP (Block# --) listet einen Block auf einem Drucker und LPINDEX (s d --) gibt den Index auf dem Drucker aus. FONT wechselt den internationalen Zeichensatz (Font) auf einem Epson-Drucker. .CLP (n --) gibt eine Zahl zweistellig und .LP (n --) eine Zahl beliebigstellig auf einem Drucker aus.

* Blöcke 30-31: Eine Demo, welche ein Balkendiagramm zeichnet. Mit GRAFIK EIN DEMO kann sie nach dem Laden mit 31 LOAD aufgerufen werden.

* Blöcke 33-34: Auf diesen Blöcken befinden sich verschiedene Stringoperatoren.
  - Mit STRING kann man einen String mit einer Länge von max. 255 Zeichen erzeugen, und mit S! kann man eine Zeichenkette in einen String speichern. Die Parameter, die ein solcher String auf dem Stack hinterläßt, sind die Stringanfangsadresse und die Stringlänge. Diese Parameter können direkt für TYPE verwendet werden:
    + 30 STRING EINGABE
    + $ Wie geht es Dir?$ EINGABE S!
    + EINGABE TYPE
  - S+ addiert zwei Strings und speichert das Ergebnis ab PAD ab:
    + z.B. EINGABE HALLO S+
    + mit PAD COUNT TYPE kann man diesen Text ausgeben
  - Mit LEN kann man die Länge einer Zeichenkette bestimmen und mit MLEN die maximal zulässige Länge:
    + EINGABE LEN.
    + EINGABE MLEN .

- Das Wort "LEFT begrenzt die Zeichenkette auf n Zeichen vom linken Rand aus, während "MID eine Teilkette ab Zeichen n1 mit einer Länge * bestimmt. "RIGHT begrenzt eine Zeichenkette auf n Zeichen vom rechten Rand aus:
  + EINGABE 3 "LEFT TYPE
  + EINGABE 4 8 "MID TYPE
  + EINGABE 5 "RIGHT TYPE
- Um einen Text in einen String zu speichern gibt es zwei Wörter. " begrenzt einen Text innerhalb einer Doppelpunktfunktion und $ hat die gleiche Funktion aber außerhalb einer Doppelpunktfunktion, wobei zu beachten ist, daß nach dem ersten " oder $ ein Leerzeichen folgen muß, damit der Interpreter es als Wort erkennen kann. Genauer gesagt, hängt die Verwendung eines der beiden Wörter von der Zustandsvariable STATE ab.
  + : TEST " Dies ist ein Beispiel" EINGABE S! ;
  + $ Hallo$ EINGABE S!
- COMPARE vergleicht zwei Zeichenketten und hinterläßt ein Flag, daß wahr ist, wenn beide Zeichenketten gleich sind. Da nur eine Längenangabe gebraucht wird, muß man die Längenangabe eines Strings mit DROP löschen!
  + EINGABE DROP NAME COMPARE .

* Blöcke 36-40: Drei verschiedene Sortieralgorithmen befinden sich auf diesen Blöcken. Mit 36 LOAD werden sie geladen, und man kann einen z.B. mit BUBBLE SORT aufrufen. Mit DEMO kann diese Demonstration wiedergestartet werden.
* Blöcke 42-43: Blöcke so auf einem Drucker auszugeben, wie sie am Bildschirm aussehen, war bis dato immer eine große Schwierigkeit. Mit diesem Utility wird das ganze nun zum Kinderspiel. Man gibt einfach Block# LISTP (z.B. 10 LISTP) ein und schon wird der entsprechende Block auf einem Epson-kompatiblen Drucker mit allen Sonderzeichen ausgegeben. Die Worte von den Blöcken 27, 28 und 29 müssen vorher geladen worden sein (27 LOAD 28 LOAD 29 LOAD) und es muß nach dem Laden LPOPEN eingegeben worden sein.
* Blöcke 45-47: Wer mit großen Zahlen (größer als 65535) rechnen muß, wird sich über diese Routinen freuen. Auf diesen Blöcken befinden sich u.a. Worte zum Umgang mit 32-Bit Zahlen. Als Erkennungszeichen muß eine solche Zahl einen Punkt enthalten (z.B. 342220.45).
  - D+ addiert und D- subtrahiert zwei 32-Bit Zahlen:
    + 200.000 300.000 D+ D.
    + 345.23 123.79 D- D.
  - Mit 2CONSTANT wird eine doppeltlange Konstante und mit 2VARIABLE eine doppeltlange Variable erzeugt:
    + 1000000.00 CONSTANT SCHULDEN
    + 134.89 2VARIABLE EINNAHMEN
  - D< vergleicht zwei Zahlen und hinterläßt ein Flag, daß wahr ist, wenn d1<d2. D0= testet, ob eine Zahl gleich Null ist und D=, ob zwei Zahlen gleich sind.:
    + FRANKFURT-MEXIKO LUXEMBURG-MEXIKO D< .

    + ATARI COMMODORE D= (Nein!) .
    + RENDITE D0= .
  - 2! speichert und 2@ holt eine 32-Bit Zahl:
    + 0.0 EINAHMEN 2!
    + : TEST SCHULDEN EINNAHMEN 2@ D< IF ." Irgend etwas stimmt nicht!" THEN ;
  - 2DROP nimmt eine 32-Bit Zahl vom Stack, 2SWAP vertauscht die beiden obersten Zahlen, 2DUP dupliziert die oberste Zahl, 2OVER kopiert die zweitoberste Zahl nach oben und 2ROT rotiert die drittoberste nach oben:
    + 1111.111 2222.2222 2OVER D. D. D.
    + 1111.111 2222.2222 2SWAP D. D.
    + 1111.111 2222.2222 3333.3333 2ROT D. D. D.
    + 400.000 300.000 2SWAP 2DROP D.
  - DMAX läßt von 2 Zahlen die größte auf dem Stack, während DMIN die kleinste auf dem Stack läßt:
    + HAUS HOF DMAX D.
    + FRANKFURT-MEXIKO LUXEMBURG-MEXIKO DMIN D.

Auf Block 45 befinden sich die Worte PICK und ROLL. PICK kopiert eine Zahl (16 Bit) nach oben, während ROLL eine nach oben rotiert. Diese Worte funktionieren - im Gegensatz zu denen bei fig-FORTH 1.4S! Mit dem Wort QUIT', welches auf Block 47 definiert wird, kann man Stackbewegungen verfolgen.

Folgende Programme befinden sich auf der zweiten Seite:

* Block 0: Inhaltsverzeichnis
* Blöcke 2-11: Fehlermeldungen
* Blöcke 12-14: siehe Grafik-FORTH Teil 1
* Block 15: Daten für den Decompiler
* Blöcke 18-36: Der Editor befindet sich auf diesen Blöcken. Man kann den Editor sehr leicht erweitern, indem man ein entsprechendes neues Wort schreibt und dem Wort auf Block 30, Zeile D, eine Taste zuweist. Man schreibt zuerst den Tastaturcode der Taste, der nichts mit dem Internencode oder dem ASCII-Code zu tun hat!, Man erhält den entsprechenden Wert durch:
  - : .CODE BEGIN 764 C@ DUP . 12 = UNTIL ;
    .CODE gibt die Tastaturcodes der gedrückten Tasten solange aus, bis <RETURN> gedrückt wird.

  Danach schreibt man C, um den Wert abzuspeichern und IST Name-des-Wortes-im-Überschreibmodues Name-des-Wortes-im-Einfügemodus.
* Blöcke 39-41: Sprites und Muster komfortabler zu erstellen ist der Zweck dieser Worte. BIN schaltet auf die binäre Zahlenbasis um, und S, speichert eine "Spritezeile" ab. Auf den Blöcken 40 und 41 finden Sie Beispiele dazu. OBJEKT liefert die nötigen Parameter zur Speicherung eines Sprites und Muster das gleiche für Füllmuster. Die Nummer vor OBJEKT oder MUSTER ist die entsprechende Sprite- oder Füllmusternummer.

  Während des Kopierens der entsprechenden Speicherbereiche an die gewünschten Stellen, muß das Betriebssytem

mittels RAM ausgeschaltet sein, da die Adressen sich im RAM-Bereich hinter dem ROM befinden.

- *Blöcke 42-43: Die Aufrufadressen der Grafikbefehle im Kern* werden hier als Konstanten definiert. Diese Konstanten sind z.B. praktisch, wenn *man in Maschinensprache Worte schreiben will, die die Grafikroutinen benutzen.*
- *Blöcke 45-46: Für Leute, die nicht andauernd an RAM und ROM* denken wollen und nicht die Supergeschwindigkeit brauchen, sind diese Worte gedacht. Nun können Sie ohne das Schreiben von RAM und ROM arbeiten.
- *Blöcke 48-49: Turtle-Grafik ist mit den Worten auf diesen Blöcken möglich.* Es befindet sich auch eine Demo dazu dabei, wobei jeder Winkel mit Sinus und Cosinus berechnet wird. TPLOT gibt einen Punkt aus, dessen Koordinaten in X und Y gespeichert sind. SIN und COS berrechnen den Wert eines Winkels mal 10000. TURTLE schaltet die Turtlegrafik ein und GOTO (x y --) "geht zu einem Punkt". TURN dreht die Zeichenrichtung um n Grad nach links. DRAW zeichnet eine Linie mit einem Winkel von 'Winkel' Grad und einer Länge von n Punkten.

Folgende Programme finden Sie auf der <u>dritten Seite</u> (0 PHYSOFF ! nicht vergessen!):

- Block 0: Inhaltsverzeichnis
- Blöcke 2-11: Fehlermeldungen
- Block 12: Nach dem Laden von Block 18 wird das 1050 TURBO unterstützt.
- Block 15: Daten für den Decompiler
- Block 18: Der "Sieb des Eratosthenes" ist ein sogennanter Benchmark. Es werden die ersten 1899 Primzahlen berechnet, und die dafür verwendete Zeit kann man dann mit anderen Sprachen vergleichen (z.B. BASIC).
- Blöcke 21-29: Eine Sprite-Demo, die die Geschwindigkeit der Grafikroutinen zeigt.
- Blöcke 30-32: Eine Füllmusterdemonstration, die alle Füllmuster anzeigt. Mit HCOPY kann man diese dann auf dem Drucker ausgeben.
- Blöcke 33-35: Wörter zum Laden und Speichern von Bildern
- Blöcke 36-45, 46-55, 56-65, 66-75, 76-85: jeweils ein Bild im Grafik-FORTH Format

2. Eine kleine Einführung in FORTH

Wie im Vorwort schon gesagt wurde, soll diese kurze Einführung nur die wichtigsten Eigenschaften von FORTH kurz ansprechen. Wie ich ebenfalls schon erwähnt habe, schlage ich Ihnen das Buch "Programmieren in FORTH" von Leo Brodie vor. Ich habe selbst diese faszinierende Sprache anhand dieses Werkes gelernt.

Das FORTH-Lexikon

FORTH wird in Worten und Zahlen ausgedrückt, die jeweils durch Leerschritte voneinander getrennt sind:

```
3 SCHRITTE VOR  STOP
```

Kommandos können entweder direkt über die Tastatur eingegeben oder von Diskette mit LOAD geladen werden. Alle Wörter, ob sie nun Systembestandteil sind oder vom Benutzer definiert wurden, haben ihren Platz im FORTH-Lexikon. Definitionswörter sind zum Hinzufügen neuer Wörter in das Lexikon. Ein solches Definitionswort ist der : (man nennt ihn Doppelpunkt oder englisch "Colon"). Man verwendet ihn zur Definition eines neuen Wortes, das aus vorher definierten Wörtern gebildet wird. Man könnte ein neues Wort BEISPIEL wie folgt definieren:

```
: BEISPIEL  3 SCHRITTE VOR  STOP ;
```

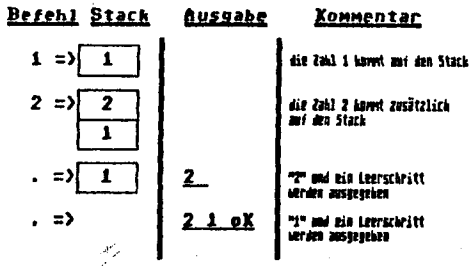
Der Stack

Bei FORTH werden Zahlen auf einem sogenannten Stack abgelegt. "Stack" ist ein englisches Wort und heißt übersetzt "Stapel". Auf diesem "Stapel" werden alle Zahlen vor und nach ihrer Bearbeitung "gestapelt". Die zuletzt eingegebene Zahl "liegt oben drauf".

Diese Funktionsweise läßt sich am besten an einem Beispiel erklären. Geben Sie dazu folgendes ein:

```
1 2 . . <RETRUN> 2 1 ok
```

Sehen Sie hier, was im Einzelnem passiert:



Dieses Prinzip, bei dem das biblische Wort von den ersten, die die letzten sein werden, realisiert wurde, heißt "LIFO" (Abk. für "Last In, First Out").

Wenn man ein Wort aufruft, das z.B. eine Zahl als Eingabe erwartet, dann holt sich das Wort diese vom Stack. Eine Zahl ist normalerweise ein 16-Bit Wert und belegt somit 2 Bytes = eine Zelle. Diese Methode hat viele Vorteile, wie Sie selbst später sehen werden.

Datenstrukturen

In FORTH "gibt es von Haus aus" zwei Datenstrukturen. Es gibt die Konstanten und die Variablen. Konstanten wird einmal ein Wert zugewiesen, der sich dann nicht mehr ändert (der Wert bleibt konstant). CONSTANT ist ein Definitionswort, das wie folgt gebraucht wird:

```
20 CONSTANT ZWANZIG
```

Das Definitionswort Variable erzeugt einen reservierten Speicherplatz für sich ändernde Daten. Es wird wie folgt benutzt:

```
0 VARIABLE ZAEHLER
```

Das Wort @ (sprich "hole" oder "fetch") holt den Inhalt einer angegebenen Speicherzelle. ZAEHLER @ holt den Inhalt der Variable ZAEHLER auf den Stack.

Das Gegenstück dazu ist das ! (sprich "speichern" oder "store"), welches Werte in angegebene Speicherplätze speichert. 10 ZAEHLER ! speichert z.B. den Wert 10 in die Variable ZAEHLER.

Steuerstrukturen

In FORTH gibt es alle Steuerstrukturen, die man für eine strukturierte Programmierung ohne GOTO benötigt.

Die Syntax der IF-THEN-Konstruktion:

```
... ( Flagge) IF LAUFEN THEN SPRINGEN ...
```

Die "Flagge" ist ein Wert auf dem Stack, der vom IF verbraucht wird. Ein Wert ungleich Null bringt den Code nach IF (im Beispiel LAUFEN) zur Ausführung. Das Wort THEN markiert das Ende des Konditionalausdrucks; danach wird das Wort SPRINGEN ausgeführt. Ein Wert gleich Null ist Auslöser dafür, daß der Code zwischen IF und THEN übersprungen wird; es wird direkt SPRINGEN ausgeführt.

Mit dem Wort ELSE kann man im Falle einer Flagge gleich Null einen alternativen Ausdruck ausführen:

```
... ( Flagge) IF LAUFEN ELSE GEHEN THEN SPRINGEN ...
```

FORTH verfügt über verschiedene Schleifenformen:

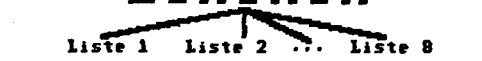
```
* ( bis+1) ( von) DO ... LOOP ...
* ( bis+n) ( von) DO ... n +LOOP ...
* ... BEGIN ... ( Flagge) UNTIL ...
* ... BEGIN ... ( Flagge) WHILE ... REPEAT ...
```

Wie sieht ein Lexikoneintrag aus?

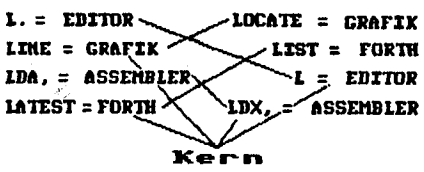
Dieser Abschnitt ist eher für fortgeschrittene Programmierer gedacht. Sie müssen also nicht unbedingt weiterlesen, um mit diesem System arbeiten zu können.

Das FORTH-Lexikon ist eine verkettete Liste von Namen. Es gibt 8 Listen an denen neue Wörter angehängt werden. Nach dem Anfangsbuchstaben wird entschieden, an welche Liste ein Wort angehängt wird. Zusätzlich gibt es noch die Vokabulare (ASSEMBLER, EDITOR, GRAFIK und FORTH), die bestimmen, wie die Wörter einer Liste untereinander verbunden sind:

Lexikon



Liste 5



Jedes Wort in FORTH hat den folgenden Kopf:

a) Verkettungsfeld (LFA) mit einer Länge von 2 Bytes
b) Namensfeld (NFA) mit einer Länge von bis zu 32 Bytes
c) Code Pointer Feld (CFA) mit einer Länge von 2 Bytes
d) Paramterfeld mit einer variablen Länge

zu a) Das Verkettungsfeld verbindet die Wörter eines Vokabulars innerhalb einer Liste untereinander.

zu b) Das erste Byte des Namenfeldes enthält die Anzahl der Zeichen des Namens. Die nächsten Bytes entsprechen dem ASCII-Code des Namens (im Normalfall). Durch die Uservariable WIDTH wird festgelegt, wieviel Zeichen eines Wortes im Höchstfall gespeichert werden. Ein Name kann also länger sein als diese Zahl, aber es werden nur maximal die WIDTH @ ersten Zeichen eines Wortes in ASCII-Form gespeichert, während das Längenbyte immer die "echte Länge" enthält. Um das ganze ein bischen deutlicher zu machen hier ein paar Beispiele:

* der Name lautet "GERADENGLEICHUNG":
  - der Name hat eine Länge von 16 Zeichen
  - WIDTH hat irgendwann den Wert 31 (Voreinstellung) erhalten
  ==> das Längenbyte hat den Wert 16 und 16 Zeichen werden gespeichert
* der Name lautet "VORLESUNG"
  - der Name hat eine Länge von 9 Zeichen
  - WIDTH war auf den Wert 5 eingstellt
  ==> das Längenbyte hat den Wert 9 und es werden die 5 ersten Zeichen gespeichert
* der Name lautet "IST"
  - der Name hat eine Länge von 3 Zeichen
  - WIDTH hat den Wert 5 irgendwann bekommen
  ==> das Längenbyte hat den Wert 3 und es werden 3 Zeichen gespeichert

Das erste Bit des Längenbytes (Bitwert 128) ist immer eins, damit der Anfang des Namenfeldes gefunden werden kann. Das zweite Bit (Bitwert 64) zeigt an, ob ein Wort ein Immediate-Wort ist. Dies besagt, daß der Interpreter es bei der Kompilation nicht kompiliert sondern direkt ausführt. IMMEDIATE verändert dieses

Bit. Das dritte Bit (Bitwert 32) bestimmt, ob ein Wort für der Interpreter auffindbar ist. Das Wort SMUDGE verändert dieses Bit. Die restlichen Bits sind nur für die Namenslänge von Bedeutung.

Das erste Bit (Bitwert 128) des letzten Bytes der abgespeicherten Namens hat den Wert 1, was zur Namensbegrenzung gebraucht wird. Wenn Sie also Beispielsweise den Namen HALLO haben, so ist nicht HALLO, sondern HALLO (untestrichen steht für Invers) gespeichert.

zu c) Die im CFA enthaltene Adresse unterscheidet eine Variable von einer Konstanten oder einer Doppelpunktfunktion. Sie entscheidet, was ein Wort nach seinem Aufruf tut. So legt z.B eine Variable ihre Adresse während der Laufzeit auf den Stack eine Konstante legt einen Wert auf den Stack und eine Doppelpunktfunktion ruft nacheinander alle Wörter auf, aus denen sie besteht.

zu d) Nach dem Code Pointer kommt das Parameterfeld. Es besteht aus den Daten für die in der CFA enthaltenen Adresse. So besteht die PFA einer Variablen aus einer Zelle, die den Wert der Variable enthält oder bei einer Konstanten aus einer Zelle, die den Wert der Konstanten enthält. Bei einer Doppelpunktfunktion besteht das Parameterfeld aus den Aufrufadressen (CFAs) ihrer Wörter.

3. Editor

Der Editor ist ein sogenannter Full-Screen Editor. Damit ist gewährleistet, daß man den Text bequem editieren kann. Der Bildschirm ist in zwei Teile aufgeteilt. Der obere Teil besteht aus Statusmeldungen und dem zu editierenden Text, der untere Teil aus 4 Zeilen zur Eingabe von Befehlen.

Da der Atari Computer normalerweise nur 40, anstatt der in FORTH üblichen 64, Zeichen/Zeile darstellen kann, habe ich zu einem Trick gegriffen. Auf einmal können Sie zwar weiterhin nur 40 Zeichen/Zeile sehen, aber durch Verschieben des Bildschirms können Sie die vollen 64 Zeichen nacheinander sehen. Der Bildschirm wird automatisch verschoben, so daß man sich auf das reine Bearbeiten des Textes konzentrieren kann. Mit EDITOR wird auf das Editor-Vokabular umgeschaltet, wodurch man die Editor-Wörter erst benutzen kann. Damit man nicht versehentlich wichtige Daten zerstören kann, schreibt man die ganze Zeit in einen Pseudo-Block. Erst wenn man UE eingibt werden die Daten in den richtigen Block übernommen.

Editor-Worte

Es werden nicht alle Worte besprochen, die das Editor-Vokabular enthält, sondern nur die, die direkt für den Benutzter gedacht sind und in den Kommandozeilen eingegeben werden können.

a. ED ==> man kann einen Block editieren
b. FH = FLUSH ==> speichert alle "geänderten" Blöcke ab
c. UE ==> UEbernimmt einen Block als geändert in den Blockpuffer
d. B ==> geht einen Block zurück
e. L ==> listet den aktuellen Block
f. L. ==> listet einen Block im Blockfenster
g. N ==> der Nächste Block wird zum aktuellen
h. W ==> Wechselt zum Block im anderen Blockpuffer
i. DATUM= ==> man kann den Stempel eingeben
j. S" ==> man gibt den Anfangsblock, Endblock, S"_, den zu suchenden Text, der mit einem Anführungszeichen begrenzt wird, ein
   Beispiel: 20 30 S" Hallo"
   Nun wird Hallo auf den Blöcken 20 bis 30 gesucht. Wenn Hallo gefunden wurde, wird Hallo + Blocknummer + Zeilennummer + Textstelle ausgegeben.
k. WIPE ==> der aktuelle Block wird gelöscht und als geändert markiert

Tastatureditor

Es stehen folgende Tastaturbefehle nach der Eingabe von ED zur Verfügung:
a. <CONTROL>+Taste drücken
b. spezielle Kombinationen

zu a.
* + ==> links
* * ==> rechts
* - ==> hoch
* = ==> runter
* H ==> Home (springt an den Blockanfang)
* Q ==> springt an den linken Rand einer Zeile
* I ==> schaltet den Einfügemodus ein
* V ==> Von (Anfang eines auszuschneidenden Bereiches)
* B ==> Bis (Ende eines auszuschneidenden Bereiches)
* S ==> Setzen (setzt den ausgeschnittenen Bereich ab der Cursorposition ein)
* D ==> gibt den Stempel aus
* <DELETE-BACKSPACE> ==> löscht ein Zeichen rechts vom Cursor, alle nachfolgenden Zeichen der Zeile rücken nach
* ; ==> geschweifte Klammer-Auf
* < ==> geschweifte Klammer-Zu
* 3 ==> springt aus dem Texteditor in die Kommandozeile
* > ==> fügt ein Leerzeichen in den Text ein

zu b.
* <CAPS> ==> schaltet zwischen Groß- und Kleinschreibung um
* <TAB> ==> springt um 3 Zeichen vorwärts
* <RETURN> ==> springt an den Anfang der nächsten Zeile
* <SHIFT>+<CONRTOL>+W ==> Wechseln (nun kann man ein Zeichen schreiben, das sonst nicht benutzbar ist, da der Editor es sonst als Kommando versteht)
* <INVERS> ==> schaltet zwischen Normal- und Inversdarstellung um
* <DELETE-BACKSPACE> ==> löscht ein Zeichen links vom Cursor und springt dorthin

Besonderheiten im Einfügemodus:
* <TAB> ==> fügt 3 Leerzeichen ein
* <DELETE-BACKSPACE> ==> alle Zeichen rechts vom Cursor "rutschen" nach
* <CONTROL>+I ==> der Einfügemodus wird verlassen

Das Zeichen am Ende einer Zeile springt, wenn ein Zeichen eingefügt wird, nicht in die nächste Zeile sondern wird gelöscht.

Statuszeile

Die Statuszeile hat folgendes Format:
SCREEN SCR* Modus X=X-Position

* SCR* ==> die Nummer des Blockes, der gerade angezeigt wird
* Modus ==> N steht für Normal=Uberschreibmodus und I für Insert=Einfügemodus
* X-Position ==> gibt die momentane Cursorspalte an

Zeilennummern

Die Zeilennummern am äußersten linken Rand sind in hexadezimaler Basis angegeben. Das hat den Grund, daß es keinen Platz für zwei Stellen gibt. Da alle Zeilennummern zusammen ein Player sind, bleiben sie auch beim Verschieben des Textbildschirms sichtbar.

Stempel

Der Stempel besteht aus einer 3 Zeichen langen Abkürzung für den Programmierernamen und dem Datum.
Beispiel: DATUM= RAI 09.08.1989

4. Grafikbefehle

Das besondere an Grafik-FORTH ist die superschnelle Grafik. Durch völlig betriebssystemunabhänige und tabellenbenutzende Befehle ist dies erst möglich geworden.

Bedeutung von RAM und ROM

Da diese Routinen, aufgrund der zahlreichen Tabellen, viel Speicherplatz belegen, habe ich sie "hinter das Betriebssystem-ROM gelegt". Damit man nun die einzelnen Befehle ansprechen kann, muß also zuerst das Betriebssystem mittels RAM ausgeschaltet werden und nach der Durchführung der Grafikbefehle wieder mit ROM eingeschaltet werden. Zwischen RAM und ROM dürfen alle Befehle aufgerufen werden, die keine Betriebssystemroutinen benutzen (z.B. +,-,*,/,=,@,C!). Befehle die Betriebssystemroutinen benutzen sind z.B.:

* .
* D.
* U.
* .R
* D.R
* EMIT
* TYPE
* LIST
* INDEX
* .LINE
* KEY
* BLOCK
* FLUSH
* EXPECT
* SPACE
* SPACES
* CR
* LOAD
* WIPE
* (LINE)
* QUIT
* HCOPY
* ."

Die einzelnen Grafikbefehle

GRAFIK

Durch das Wort GRAFIK wird das GRAFIK-Vokabular zum Context-Vokabular, wodurch man die Grafikbefehle erst benutzen kann. Durch FORTH wird das FORTH-Vokabular wieder zum Context-Vokabular. Durch GRAFIK DEFINITIONS wird GRAFIK zum aktuellen Vokabular und alle Wörter, die danach definiert werden, werden dort

angehängt. Durch FORTH DEFINITIONS kann man FORTH wieder zum aktuellen Vokabular machen.
Beispiele:
* GRAFIK EIN
* GRAFIK DEFINITIONS : TEST ... ;

EIN

Durch den Befehl EIN wird der Grafikbildschirm eingeschaltet (RAM ... SETDL ... ROM). Der obere Teil besteht aus 192 Grafikzeilen und der untere Teil aus 4 Textzeilen. Es handelt sich also praktisch um einen GRAPHICS-8+16-Bildschirm mit zusätzlichen 4 Textzeilen. Durch das Wort wird ebenfalls der gesamte Grafikbildschirm gelöscht (RAM ... CLEAR RCL ... ROM) und die oberste y-Koordinate erhält den Wert 0 (RAM ... 0 WINDOW ... ROM). Man benutzt diesen Befehl meistens im Zusammenhang mit dem Wort GRAFIK (GRAFIK EIN), um vom normalen Textmodus und FORTH-Vokabular zur Grafikprogrammmierung zu wechseln.
Beispiel: GRAFIK EIN RAM 10 20 PLOT ROM

AUS

AUS wirkt wie der BASIC-Befehl GRAPHICS 0. Der Textmodus wird also eingeschaltet. Weder das aktuelle Vokabular noch CONTEXT werden dadurch geändert.
Beispiel: GRAFIK AUS

CLEAR

CLEAR löscht die Spalten 0-255 des Grafikbildteils.
Beispiel: RAM CLEAR 2 3 100 200 LINE ROM

RCL

RCL löscht den rechten Bildschirmbereich, also die Spalten 256-319.
Beispiel: RAM RCL ROM

COLOR (Farbe)

Mit COLOR kann eine von 3 "Farben" gewählt werden. Farbe 0 bedeutet, daß ein Punkt, der verändert werden soll, gelöscht wird (entspricht COLOR 0 in BASIC). Bei Farbe 1 hingegen wird der Punkt gesetzt (COLOR 1 in BASIC), und bei Farbe 2 wird ein Punkt gesetzt, wenn er vorher gelöscht war und gelöscht, wenn er vorher gesetzt war (exklusives Oder). Die Befehle PLOT, LINE, CIRCLE, TEXT, DRAWTO, BOX und SPRITE sind von der mit COLOR gewählten "Farbe" abhänig.

Beispiel: RAM 1 COLOR 100 34 2DUP PLOT PLOT ROM

PLOT (x y)

PLOT verändert den Punkt x,y. Die Parameter dürfen die Werte 0-255 haben.
Beispiel: RAM 67 89 PLOT 70 80 PLOT ROM

LINE (x1 y1 x2 y2)

LINE zieht eine Linie zwischen den Punkten x1,y1 und x2,y2. Dabei wird immer von "oben nach unten gezeichnet", wodurch erst Turtle-Grafik möglich wird. Nach dem Zeichnen befinden sich die Koordinaten des Punktes x2,y2 in den Speicherstellen 90 und 91. Die Parameter dürfen Werte von 0-255 haben.
Beispiel: RAM 0 1 46 43 LINE 50 51 100 123 LINE ROM

CIRCLE (x y x-Radius y-Radius)

CIRCLE zeichnet eine Ellipse mit dem Mittelpunkt x,y und den Radien x-Radius und y-Radius. Ist x-Radius=y-Radius, so wird ein Kreis gezeichnet. x,y dürfen die Werte 0-255 haben und x- und y-Radius 0-126. Sind der Mittelpunkt und die Radien so gewählt, daß die Ellipse über den Rand des Bildschirms hinausgeht, so wird am gegenüberliegenden Rand weitergezeichnet!
Beispiel: RAM 127 DUP 10 DUP CIRCLE ROM

FILL (x y Muster#)

FILL füllt eine begrenzte Fläche vom Punkt x,y aus mit dem Muster Muster#. Dabei wird von der Mitte nach oben und unten und von der Mitte nach links und rechts die Fläche abgetastet und gefüllt. Es kann, bei komplizierten Flächen und einer ungünstigen Wahl des Punktes x,y, nötig sein die Fläche mehrmals von verschiedenen Punkten aus zu füllen. x und y können Werte von 0-255 haben und Muster# einen Wert von 0-31. FILL ist nicht mit dem FILL-Befehl des FORTH-Vokabulars zu verwechseln!
Beispiel: RAM 100 120 21 GRAFIK FILL ROM

SPRITE (x y Sprite#)

Sprites sind Grafikobjekte mit einer Größe von 16x16 Punkten, die an beliebigen Stellen angezeigt werden können. Wenn "Farbe" 3 gewählt wurde, kann das Sprite durch einen nochmaligen Aufruf an der gleichen Stelle wieder gelöscht werden. Wenn Sie verhindern wollen, daß ein Sprite flackert, wenn es gesetzt oder gelöscht wird, müssen Sie VCOUNT (54283) abfragen. Es enthält die Bildschirmzeile, die gerade generiert wird, geteilt durch 2. Für

PAL-Systeme (bei uns) kann VCOUNT einen Wert zwischen 0 und 155 haben. Nach meinen eigenen Erfahrungen ist der Elektronenstrahl außerhalb des Grafikfensters, wenn VCOUNT einen Wert >=100 hat. Durch das folgende Wort kann man verhindern, daß ein Sprite flackert:

```
CODE SPRITE'  ( x y Sprite# -- )
   XSAVE STX, BEGIN, VCOUNT LDA, 100 # CMP, >= UNTIL,
   HEX 0E70C ( SPRITE-Handleradresse) JMP, DECIMAL C;
```

Der Nachteil des Wortes SPRITE' ist, daß es recht allgemein gehalten ist und somit den Programmablauf möglicherweise unnötig verzögert. Es könnte ja sein, daß das Sprite nur in einem kleinen Teil des Bildschirms angezeigt werden kann und somit der Vergleichswert 100 für die Abfrage von VCOUNT viel kleiner sein könnte, was das ganze Programm beschleunigen würde. Ich empfehle Ihnen deshalb ein bischen mit VCOUNT zu experimentieren.

x,y sind die Üblichen Koordinaten, und Sprite# kann einen Wert von 0-15 haben.

Beispiel:

```
GRAFIK DEFINITIONS
: T2   RAM  78 24 0 SPRITE  100 0 DO LOOP  78 24 0 SPRITE ;
```

TEXT (x y a # Modus#)

Mit TEXT läßt sich ein Text, der ab der Adresse a mit der Länge # abgespeichert ist, ab der Position x,y im Modus Modus# ausgeben. Dabei wird ein ab Adresse $F000 gespeicherter Zeichensatz benutzt. Es können alle Zeichen des Zeichensatzes ausgegeben werden (also auch inverse Zeichen). Modus# gibt den Textmodus an:

- Modus#=0 ==> 40 Zeichen Normalschrift
- Modus#=64 ==> 40 Zeichen doppelthoch
- Modus#=128 ==> 80 Zeichen Schmalschrift
- Modus#=192 ==> 80 Zeichen doppelthoch

x,y ist die linke obere Ecke des Textstreifens. y kann beliebig zwischen 0 und 239 (bei doppelthoher Schrift) bzw. 247 gewählt werden. x kann beliebig zwischen 0 und 311 gewählt werden, doch die Textausgabe beginnt stets an einer Vielfachen von 8 Positionen. x wird intern in x=int(x/8)*8 umgewandelt. Intern sind damit also nur 40 Positionen möglich.

Beispiel: RAM 8 10 0 BLOCK 10 64 TEXT ROM

CROSS (x y)

CROSS legt Über den Bildschrim ein Fadenkreuz. x,y gibt dabei den Schnittpunkt des Fadenkreuzes an, das aus zwei auf sich senkrecht stehenden Linien besteht. x,y haben den Üblichen Wertebereich.

Beispiel: RAM 20 45 CROSS ROM

CUR (x1 y1 v -- x2 y2)

Mit CUR können x- und y-Koordinaten mit der Tastatur oder dem Joystick verändert werden. Die Schrittweite wird mit v festgelegt, und x,y geben die momentane Position an. Durch Bewegen des Joysticks in Port 1 werden die Koordinaten entsprechend verändert. Wenn man über die Tastatur steuert, muß man beachten, daß die Steuerung wie folgt verläuft:

```
Q  W  E

A  S  D

Z  X  C
```

Nach der Ausführung von CUR liegen die neuen Koordinaten auf dem Stack. x,y,v können Werte von 0-255 haben.

Beispiel: RAM 40 50 STEP C@ CUR ROM . .

CSPEED (-- v)

Mit CSPEED kann man die Schrittweite v über die Tastatur abfragen. Die Tasten 1 bis 0 stehen für die Werte 1-10.

Beispiel: RAM CSPEED STEP C! ROM

HCOPY

Mit HCOPY läßt sich eine Druckroutine aufrufen, die den gesamten Grafikbereich auf einem Epson-Drucker oder kompatiblen ausdruckt (320*256 Punkte!). Der Ausdruck erfolgt dabei mit einer Dichte von 72 Punkten pro Zoll, und das Betriebssystem-ROM muß eingeschaltet ein!

Beispiel: HCOPY GRAFIK AUS

WINDOW (y-Top)

Mit diesem Befehl wird der sichtbare Grafikbereich geändert. y-Top gibt dabei die y-Koordinate der obersten Zeile an. Bei der Verschiebung wird nicht gescrollt, sondern der ANTIC umprogrammiert. y-Top kann Werte von 0 bis 64 haben.

Beispiel: RAM 0 DUP 255 DUP LINE 45 WINDOW ROM

(SIN) (Winkel -- Sinuswert)

(SIN) schlägt in einer Tabelle nach dem entsprechenden Sinuswert für Winkel nach. Winkel darf nur zwischen 0 und 180 liegen, und Sinuswert gibt den eigentlichen Sinuswert*10000 an! 90 (SIN) ergibt also 10000 und nicht 1. Der Grund dafür ist, daß FORTH nur mit Festkommazahlen rechnet.

LOCATE (x y -- Status)

Mit LOCATE wird abgefragt, ob der Punkt x,y gesetzt ist. Wenn der Punkt nicht gesetzt ist, ist Status gleich Null und sonst hat Status den Bit-Wert (128,64,32,16,8,4,2,1) des Punktes. x,y können Werte von 0-255 haben.
Beispiel: RAM 10 13 LOCATE ROM .

SETDL

Mit SETDL wird die Display-List des Grafikbildschirms "gesetzt".
Beispiel: RAM SETDL CLEAR ROM

DRAWTO (x2 y2)

DRAWTO zeichnet eine Linie zu einem vorherigen Punkt (Koordinaten in den Speicherstellen 90,91). FÜr Assemblerprogrammierer: Es gibt keine Grafikroutine eigens fÜr DRAWTO. Im Handler wird DRAWTO durch PLOT JSR, LINE JSR, simuliert. Wenn Sie einen Befehl in Maschinencode schreiben und eine DRAWTO-Routine benötigen, simulieren Sie sie durch einen Sprung zur PLOT- und danach zur LINE-Routine, wie es im Handler ebenfalls gemacht wird.
Beispiel: RAM 10 23 PLOT 45 78 DRAWTO ROM

BOX (xo yo xu yu)

Mit diesem Befehl wird ein Rechteck mit dem linken oberen Punkt xo,yo und dem unteren rechten xu,yu gezeichnet. xo,yo,xu und yu haben einen Wertebereich von 0-255.
Beispiel: RAM 40 50 90 120 BOX ROM

Tabelle der Grafikroutinen

Befehl	Betriebssystem	Handleradr.	Routineadr.	Parameter	belegte Speicherstellen (Zeropage)
CLEAR	aus	$E721	$DD00	--	242,243
RCL	aus	$E715	$DD2D	--	92,93
PLOT	aus	$E700	$DD03	x1 y1	242,243
LINE	aus	$E703	$DD06	x1 y1 x2 y2	114,115,118,119,126,240-245
CIRCLE	aus	$E706	$DD09	x y rx ry	126,127,242-245
FILL	aus	$E709	$DD0C	x1 y1 Muster*	92,93,97,119,120,126,127,242,243
SPRITE	aus	$E70C	$DD0F	x1 y1 Sprite*	126,127,246,247
TEXT	aus	$E70F	$DD24	tx ty a * Modus*	91-93,96,97,114,115,118-120,126,127,240,245
CROSS	aus	$E712	$DD18	x3 y3	91,96,97
CUR	aus	$E724	$DD1B	x1 y1 v -- xn yn	92
CSPEED	aus	$E727	$DD27	-- v	
HCOPY	an	' HCOPY	$9521	--	92,93,96,97,240,241
WINDOW	aus	$E71B	$DD30	y-Top	93
LOCATE	aus	$E71E	$DD12	x1 y1 -- Status	242,243
SETDL	aus	$E718	$DD2A	--	
DRAWTO	aus	$E72A	--	x2 y2	siehe LINE
BOX	aus	$E72D	$DD15	x3 y3 x4 y4	92,93,116,117 + LINE
COLOR	aus	$E733	$DD1E	Farbe	

Die Parameter stehen fÜr folgende Adressen (dezimal):
* x1=xn=90
* y1=yn=91
* x2=rx=y-Top=Sprite*=Muster*=Farbe=96
* y2=ry=97
* a=240
* *=119
* Modus*=120
* v=$FFF1
* Status=x=126
* y=127
* tx=91
* ty=y3=93
* x3=92
* x4=116
* y4=117

Wenn man einen Grafikbefehl in Maschinensprache aufrufen will, benutzt man am besten die Routineadresse. Bevor man eine Routine aufruft, speichert man die Parameter in die entsprechenden Adressen und ruft sie dann via JSR auf. Wenn man die Zahlen auf dem Stack hat und die Handlerroutinen benutzen will, muß man mittels JMP dorthin springen. Von der Routine wird dann am Ende direkt nach NEXT gesprungen. Hier nun ein Beispiel:

```
CODE TEST ... 45 * LDA, 90 ( x1) STA, 50 * LDA, 91 ( y1) STA,
   HEX 0DD03 JSR, DECIMAL ... NEXT JMP, C;
```

Wenn Sie eine solche Routine aufrufen, müssen Sie vorher mittels RAM das Betriebssystem aus- und es nachher mit ROM wieder einschalten.

Mehr Uber Sprites

Sprites haben eine Größe von 16 * 16 Punkten. Die Spritetabelle liegt im Speicher von $EA00 (59904) bis $EBFF (60415). Jedes der 16 Sprites besteht aus 32 Bytes, 16 * 16 Bits. Man kann die Sprites mittels der in Block 39 (Grafik-FORTH Teil 2) befindlichen Befehle erstellen. Immer zwei Bytes bei einem Sprite bilden eine Zeile, die man mit S, abspeichern kann. Man kann nicht den Befehl , benutzen, da dieser immer zuerst das Low- und dann das Highbyte abspeichert. OBJEKT berechnet aus der Spritenummer deren Anfangsadresse und legt diese zusammen mit der Länge auf den Stack. Vor dem Kopieren mit CMOVE in die Spritetabelle ist zu beachten, daß vorher mit RAM das Betriebssystem ausgeschaltet wird (ROM später nicht vergessen!).

Mehr Uber Füllmuster

Es gibt 32 verschiedene Füllmuster, die alle eine Größe von 8 * 8 Punkten haben. Jedes Füllmuster beleget 8 Bytes (8 * 8 Bits). Die Mustertabelle befindet sich ab $EC00 (60416) im Speicher und belegt 640 Bytes. Jede Füllmusterzeile entspricht einem Byte und kann deshalb mit C, abgespeichert werden. MUSTER (Block 39) wandelt eine Musternummer in die entsprechende Adresse und Länge (8) um. Da die Mustertabelle ebenfalls im Speicherbereich hinter dem ROM liegt, muß man vor dem Kopieren mit RAM das RAM anschalten (ROM später nicht vergessen!).

Bilder laden oder speichern

Mit den Wörten auf den Blöcken 33-35 ist es möglich Bilder zu speichern und zu laden. Durch Block# GET BILD kann man ein Bild laden, welches ab Block Block# abgespeichert ist. Block# PUT BILD speichert ein Bild ab Block Block# ab.

In der Variablen #ZEILEN speichert man die Anzahl der Zeilen, die ein Bild haben soll. Für normale GRAPHICS 8-Bilder (z.B. DESIGN-MASTER-Bilder) speichert man also den Wert 192 in #ZEILEN und für Grafik-FORTH-Bilder den Wert 256 für 256 Zeilen. Wenn man Bilder von DOS 2-Disketten Übernehmen will, wandelt man sie mit Hilfe des "Access DOS 2"-MenUpunktes im DOS 3-MenU um. Man sollte umgewandelte DOS 2-Bilddateien grundsätzlich auf eine Diskette speichern, die nur für Bilder gebraucht wird. Damit ist sichergestellt, daß alle Blöcke eines Bilder direkt hintereinander liegen und es somit keine Schwierigkeiten bei ihrem laden gibt.

Danach wählt man dann den Punkt "X-user-defined" im HauptmenU. Dort gibt man zuerst ein <RETURN> und dann auf die Frage ob BEREICH geladen werden soll <Y>+<RETURN> ein. Nachdem BEREICH

geladen worden ist, legt man die Diskette mit der eben erzeugten Bilddatei ein und drUckt die RETURN-Taste (<RETURN>). Nach der Eingabe des Dateinamens und der Beantwortung der Frage, ob man die Blocknummern auf einem Drucker ausgedruckt haben will, werden die entsprechenden Blocknummern ausgegeben.

Als vorletzten Schritt läd man Grafik-FORTH und den Block 48 auf Seite 1 (48 LOAD), legt Grafik-FORTH Teil 3 ein und gibt 0 PHYSOFF ! 33 LOAD ein, wodurch die Wörter zum Laden und Speichern von Bildern geladen werden.

Zum Schluß gibt man 3 PHYSOFF ! (wegen DOS 3) GRAFIK EIN 192 #ZEILEN ! Block# GET BILD ein. Mit 0 PHYSOFF ! 256 #ZEILEN ! Block# PUT BILD kann man dann das Bild auf den Block Block# einer FORTH-Datendisk abspeichern.

5. Assembler

FORTH-Programme sind im Vergleich zu BASIC sehr schnell. Es gibt aber immer wieder Fälle bei denen es sinnvoll ist den Computer in Maschinensprache zu programmieren. Dabei ist bemerkenswert, daß FORTH-Programme z.T. erheblich kürzer sind als vergleichbare Assemblerprogramme.

In dieser fig-FORTH-Version ist ein sehr guter Makroassembler integriert. Er besitzt folgende Eigenschaften:

- vom Benutzer können Makros definiert werden
- Zahlen können in einer beliebigen Zahlenbasis angegeben werden
- in den Ausdrücken kann voll auf vorhandene Worte zurückgegriffen werden
- zur Programmierung werden strukturierte Schleifen mit bedingten Anweisungen verwendet
- es gibt eine Fehlerprüfung
- es sind keine Marken zugelassen
- der Assembler ist in FORTH geschrieben und belegt ca. 1300 Bytes
- es können "High-Level"-Worte in die Routinen eingebaut werden

Während der Assemblierung enthält die Uservariable CONTEXT die Wörterbuchadresse des Assemblers. Wörter werden zuerst im Assemblervokabular gesucht, so daß es möglich ist, daß, wenn Namen im Assemblerwortschatz und im FORTH-Wortschatz gleich sind, das Assemblerwort aufgerufen wird.

Während der Asssemblierung einer CODE-Definition werden die Worte in ausführbaren Maschinencode übersetzt. Dabei werden für die Verzweigungen die nötigen Adressberechnungen durchgeführt. Die bedingten Anweisungen (z.B. IF, UNTIL, BEGIN, ...) erzeugen dann den entsprechenden Maschinencode.

Die meisten Assemblerworte sind mit einem "," abgeschlossen. Diese Schreibweise hat drei Gründe:

- Das Komma zeigt den logischen Abschluß einer Anweisung an und entspricht somit einer Zelle in Assemblerschreibweise.
- In FORTH speichert das Komma eine Zahl ins Wörterbuch. Somit wird durch das Komma verdeutlicht, daß diese Anweisung im Wörterbuch gespeichert wird.
- Das Komma unterscheidet bestimmte Befehlcodes von möglichen Hexzahlen, wie ADC.

Der Assembler führt mehrere Tests auf Fehler in der Eingabe durch:

a. Alle bedingten Anweisungen müssen richtig geschachtelt und gepaart sein.
b. Die Adressierungsarten und Operanden müssen bei den Befehlscodes erlaubt sein.
c. Die Anzahl der Stackargumente darf nicht durch eine CODE-Definition nach außen hin verändert werden.

Diese Tests werden durch Überwachung des Stacks (in CSP) und durch Bitmasken für die Adressierungsarten durchgeführt. Wird ein Fehler gefunden, so wird 0; nicht durchgeführt und die Definition ist nicht benutzbar. Ihr Name wird zwar bei VLIST mitgelistet, doch der Interpreter kann sie nicht finden. Um den Namen zu löschen gibt man SMUDGE FORGET Name ein. SMUDGE ändert ein Bit, so daß das Wort vom Interpreter gefunden werden kann und mit FORGET gelöscht werden kann. Die Fehlermeldung "DEFINITION NOT FINISHED" erscheint, wenn der Wert der USER-Variablen CSP am Ende der Definition vom Wert des Stackpointers abweicht, sprich wenn vor und nach der Definition unterschiedlich viele Zahlen auf dem Stack liegen:

a. zuwenige Zahlen: 5 CODE HALLO 2# # LDA, (<- bezieht sich auf eine Zahl, die außerhalb der Definition von HALLO auf den Stack gelegt wurde -> falsch!) HERE STA, NEXT JMP, C; --> Fehlermeldung DEFINITION NOT FINISHED
b. zuviele Zahlen: CODE HALLO HERE LDA, PAD STA, 5 (<- zuviel) NEXT JMP, C; --> Fehlermeldung DEFINITION NOT FINISHED

Die Fehlermeldung "CONDITIONALS NOT PAIRED" wird ausgegeben, wenn ein Fehler bei der Verschachtelung aufgetreten ist. Ist bei einem Befehlscode die angegebene Adressierung nicht möglich, so erscheint die Meldung "HAS INCORRECT ADRESS MODE".

Adressierungsarten

Für die einzelnen Adressierungsarten sind folgende Schreibweisen festgelegt:

Symbol	Modus	Operand
.A	Akkumulator	
#	unmittelbar	nur 8 Bit
,X	mit X indiziert	Zero-Page/absolut
,Y	mit Y indiziert	Zero-Page/absolut
X)	indiziert indirekt X	nur Zero-Page
)Y	indirekt indiziert Y	nur Zero-Page
)	indirekt	nur absolut
keins	Speicher	Zero-Page/absolut

Es folgen einige Beispiele in FORTH-Assembler und normaler Assemblerschreibweise:

```
.A ROL,          ROL A oder ROL
1 # LDY,         LDY #1
DATA ,X STA,     STA DATA,X
DATA ,Y CMP,     CMP DATA,Y
6 X) ADC,        ADC (6,X)
POINT )Y STA,    STA (POINT),Y
VEKTOR ) JMP,    JMP (VEKTOR)
```

Stack

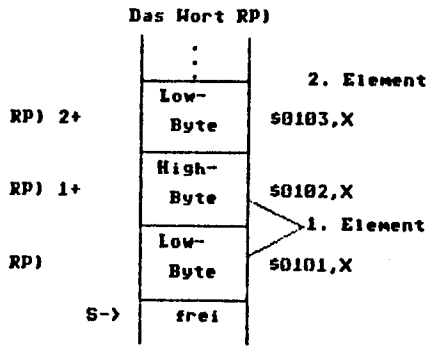
Der Stack wird in der Zero-Page angelegt und wächst in Richtung niedrigerer Adressen. Die Anfangsadresse ist 234. Das X-Register wird als Stapelzeiger verwendet. Ein Element des Stacks ist ein 16-Bit Wert und belegt somit 2 Bytes. Die Befehlsfolge INX, INX, entspricht dem Wort DROP. Um auf die erste oder zweite Zahl auf dem Stack zugreifen zu können, sind im Assembler die Makros BOT und SEC vereinbart.

```
BOT LDA, <=> 0 ,X LDA,
SEC LDY, <=> 2 ,X LDY,
```

Mit BOT LDA, BOT 1+ LDY, werden die niederwertigen 8 Bit des obersten Elements auf dem Stack im Akkumulator und die höherwertigen 8 Bit im Y-Register gespeichert.

Returnstack

Der FORTH-Returnstack befindet sich in Page 1. Er startet bei 511 und wächst nach unten. Das CPU-Register S zeigt auf den nächsten freien Speicherplatz unterhalb des benutzten Teils des Returnstacks.



Als Beispiel hier das Wort "R", das den "obersten" Wert des Returnstacks liest:

```
CODE R  ( -- n )   XSAVE STX, TSX, RP) LDA,
   RP) 1+ LDY, PHA, TYA, XSAVE LDX, PUSH JMP, C;
```


FORTH-Register

Verschiedene FORTH-Register sind nur auf Assemblerebene benutzbar.

IP (-- a) (assemblieren)
: Während der Laufzeit enthält IP die Adresse des nächsten Wortes, das von NEXT aufgerufen wird.

W (-- cfa) (assemblieren)
: Adresse der Codefeldadresse des Wortes, das soeben ausgeführt wird.

UP (-- a) (assemblieren)
: Enthält die Anfangsadresse der Benutzervariablen.

Prozessor-Register

* Das Y-Register ist null. Es kann frei genutzt werden.
* Das X-Register enthält das Lowbyte des "obersten" Stackelements relativ zur Adresse 0.
* Der CPU-Stackpointer S zeigt ein Byte unterhalb des Lowbytes des "obersten" Returnstackelementes.
* Der Akkumulator darf frei genutzt werden.
* Der Prozessor ist im Binärmodus und muß in diesem zurückkehren.

XSAVE

XSAVE ist ein ein Bytebuffer in der Zero-Page (Adresse 255) für die zeitweilige Speicherung des X-Registers.

N-Bereich

N ist die Adresse 240 in der Zero-Page. Von N-1 bis N+7 ist ein ungenutzter Speicherbereich, der dem Benuzter zur freien Verfügung steht. Da viele FORTH-Routinen diesen Speicherbereich benutzen, sollte N nur innerhalb eines einzelnen Wortes benutzt werden.

SETUP

Für das Kopieren von Zahlen vom Stack nach N gibt es die spezielle Routine "SETUP". Man läd zuerst den Akkumulator mit 1, 2,3 oder 4, je nachdem wieviel Werte man vom Stack in den N-Bereich kopieren will, und springt dann mittels JSR, nach SETUP.
Beispiel: 3 # LDA, SETUP JSR,

Steuerstrukturen

Der Assembler benötigt keine Marken. Diese werden durch zwei Besonderheiten ersetzt. Jedes in FORTH definierte Wort kann jederzeit in einer CODE-Definition verwendet werden. Zweitens wird die Assemblierung durch bedingte Verzweigungen (BEGIN, ... cc UNTIL, und cc IF, ... ELSE, ... THEN,) gesteuert. Die Abkürzung cc steht für Condition Code. Dieser Code steuert die Verzweigung.

Das folgende Beispiel zeigt die Programmierung einer Warteschleife:

```
CODE DEMO   128 # LDA, N STA,  BEGIN, N DEC,
   0= UNTIL,  NEXT JMP, C;
```

Die Hilfszelle N wird in einer "BEGIN, ... UNTIL,"-Schleife dekrementiert. Als Bedingung für den Schleifenabbruch wird die Abfrage 0= verwendet. Wenn diese Bedingung erfüllt ist, wird mittels JMP zum nächsten Wort gesprungen.

Es gibt folgende Bedingungen (Condition Codes) für die Verzweigungen:

```
0=.....Verzweigung, wenn das Z-Flag des Prozessors
                    gleich 1 ist
0<.....Verzweigung, wenn das N-Flag des Prozessors
                    gleich 1 ist
CS.....Verzweigung, wenn das CARRY-Bit gesetzt
                    ist.
>=.....gleich CS aber, nur korrekt nach CMP,
```

Durch NOT wird das Gegenteil abgefragt:

* PORTA LDA, 0= IF, <a> THEN, (wenn PORTA gleich null ist wird <a> ausgeführt)
* PORTA LDA, 0= NOT IF, <a> THEN, (wenn PORTA ungleich null ist wird <a> ausgeführt)

Wenn man ein "BEGIN <a> cc WHILE <b> REPEAT"-Schleife in Assembler programmieren will muß man zu einem Trick greifen. Man schreibt: HERE (oder BEGIN, DROP) <a> cc IF, <b> ROT JMP, THEN,

FORTH-Wörter in Maschinensprache

Es gibt einen speziellen Befehl um High-Level-Wörter in Routinen einzubauen: EXEC, (pfa --). Wie Sie schon dem Stackkommentar entnehmen können verlangt EXEC, eine Adresse auf dem Stack. Diese Adresse ist die "PFA" des aufzurufenden Wortes und wird mit Hilfe des Tick (') ermittelt. ' VLIST EXEC, bindet z.B. das Wort "VLIST" in eine Routine ein.

Wohin am Ende eines Wortes springen?

Es gibt mehrere Möglichkeiten wohin man springen kann:

a. NEXT
b. PUT
c. PUSH
d. POP
e. POPTWO

zu a. NEXT JMP, ruft das nächste Wort auf.

zu b. Mit PUT JMP, wird das oberste Stackelement von einem 16-Bit Wert überschrieben (z.B. CODE DEMO 2000 LSB # LDA, PHA, 2000 MSB # LDA, PUT JMP, C;). Das niederwertige Byte wird über den Returnstack und das höherwertige über den Akkumulator an PUT übergeben.

zu c. Mit den gleichen Bedingungen kann eine 16-Bit Zahl auch an PUSH übergeben werden. PUSH JMP, legt eine 16-Bit Zahl auf dem Stack ab. Anstatt mit PUSH, kann man auch mit folgendem Programm Daten auf dem Stapel ablegen:

```
CODE DEMO   DEX, DEX, 2000 LSB # LDA, BOT STA,
              2000 MSB # LDA, BOT 1+ STA, NEXT JMP, C;
```

zu d. Ein Sprung zu POP erniedrigt den Stackzeiger um eine 16-BIT Zahl.

zu e. Ein Sprung zu POPTWO erniedrigt den Stackzeiger um zwei 16-Bit Zahlen.

Alle Sprünge springen zum Wort NEXT.

Wörter des Assemblervokabulars

```
ASSEMBLER  ( -- )
   setzt CONTEXT auf ASSEMBLER
BEGIN,  ( -- Anfangsadr-der-Schleife 1)   (assemblieren)
CPU  ( Operand -- )
   Ein Definitionswort, das Befehle ohne Adressierung erzeugt.
C;  ( -- )
   beendet eine CODE-Definition
ELSE,  ( a1 2 -- a2 2)   (assemblieren)
   Der Programmteil hinter ELSE, wird während der Laufzeit
   ausgeführt, wenn cc vor IF, unwahr ist.
IF,  ( cc -- a 2) (assemblieren)
   Während des Assemblierens wird der cc gespeichert und die
   Adresse auf den Stack gelegt, die von ELSE, oder THEN,
   weiterverwendet wird. Die 2 dient zur Fehlererkennung.
INDEX  ( -- a) (assemblieren)
   eine Tabelle, die die erlaubten Bitmuster enthält
M/CPU  ( Bitmuster Opcode -- )
   Ein Definitionswort des Assemblers. Es erzeugt Operation Codes
   mit mehrfachen Möglichkeiten der Adressierung.
```

MEM (--)
   setzt MODE auf die absolute Adressierung
MODE (-- a)
   Variable, welche die augenblickliche Adressierungsart
   enthält.
THEN, (a 2 --) (assemblieren)
   Während des Assemblierens wird die Adresse a für die
   Verzweigungsberechnung benötigt. Die 2 dient zur
   Fehlererkennung.
UNTIL, (a 1 cc --) (assemblieren)
   Während des Assemblierens wird ein bedingter Sprung, der von
   cc abhängt, zur Adresse a assembliert. Die Eins dient zur
   Fehlererkennung.
UP (-- Anfangsadr-der-Benutzer-Variablen)
UPMODE (a ? -- a ?)
   stellt den Adressmodus aufgrund der Operandenlänge und des
   Opcodetyps ein.
W (-- a)
   Während der Laufzeit enthält a die CFA des Wortes, das gerade
   ausgeführt wird.

Was ist ein Operand und was ein Opcode?

Der Maschinensprachebefehl an sich ist der Opcode (z.B. LDA,) oder Operation Code.
Der Operand ist praktisch das Parameter dazu. Bei 40 LDA, ist 40 der Operand und LDA, der Opcode.

6. Fehlerbehandlung

Bei Grafik-FORTH gibt es 3 Möglichkeiten den Computer auf einen Fehler reagieren zu lassen. Die zentrale Rolle spielt dabei die Uservariable WARNING. Wenn WARNING den Wert 0 hat, wird eine Fehlernummer ausgegeben, wenn es einen positiven Wert hat, wird ein Text ausgegeben und wenn es einen negativen Wert hat, wird ABORT ausgeführt (wie bei Druck auf die BREAK-Taste). In den Blöcken 2-11 stehen die Texte, die beim Auftreten eines bestimmten Fehlers ausgegeben werden:

1. Block 2: Meldungen 128 bis 143
2. Block 3: Meldungen 144 bis 159

3. Block 4: Meldungen 1 bis 15
4. Block 5: Meldungen 16 bis 31
5. Block 6: Meldungen 32 bis 47
6. Block 7: Meldungen 48 bis 63
7. Block 8: Meldungen 64 bis 79
8. Block 9: Meldungen 80 bis 95
9. Block 10: Meldungen 96 bis 111
10. Block 11: Meldungen 112 bis 127

Während die Meldungen 128 bis 159 Betriebssystemmeldungen sind, hängen die anderen mit FORTH zusammen. Sie werden immer vom aktuellen Laufwerk geladen! Da es öfter vorkommen kann, daß das Laden der Fehlermeldungen von Disk sich als ungünstig herausstellt (z.B. beim Editieren von Blöcken), rate ich Ihnen den Wert von WARNING ("von Haus aus" 0) nicht zu verändern.

7. DOER/MAKE

DOER (in Block 48/Seite : 1 definiert) ist ein Definitionswort, welches einen Eintrag erzeugt, dessen Parameterfeld aus einer Zelle besteht. Diese Zelle enthält die Vektorfeldadresse und zeigt auf das Wort NOTHING (deutsch: nichts).

Während der Ausführung eines mit DOER definierten Wortes, wird die Vektoradresse auf den Returnstack gelegt. Die FORTH-Ausführung wird dann mit dieser Adresse fortgesetzt; dabei wird also die Vektorfunktion ausgeführt. Dieser Trick gelingt aber nur mit Doppelpunktfunktionen.

z.B: DOER SHOW

: TEST1 MAKE SHOW 2 . 3 . ;

TEST1 <RETURN> --> SHOW zeigt auf den Code "2 . 3 . ;".

Wenn nun SHOW aufgerufen wird, wird der Code ausgeführt und eine 2 und eine 3 werden ausgegeben.

: TEST2 MAKE SHOW 2 . ;AND 3 . ;

TEST2 funktioniert anders als TEST1. Nach dem Aufruf von TEST2 zeigt SHOW auf den Code "2 . ;" und eine 3 wurde ausgegeben! Nach dem Aufruf von SHOW wird also nur noch eine 2 ausgegeben! ;AND ermöglicht also ein Wort so zu definieren, daß bei seinem Aufruf sowohl Worte mit MAKE näher bestimmt, als auch Worte ausgeführt werden. Mit Hilfe von ;AND können auch innerhalb eines Wortes mehrere Worte mit MAKE näher bestimmt werden. Die DOER/MAKE-Konstruktion läßt sich auch außerhalb von Doppelpunktwörtern verwenden, wenn anstatt des Semikolons (";") "M;" benutzt wird.

z.B. MAKE SHOW 2 . 3 . M;

Auf Block 49 befinden sich drei Beispielanwendungen für die DOER/MAKE-Konstruktion. Beim ersten Beispiel wird mit Hilfe der Rekursion der größte gemeinsame Teiler zweier Zahlen ermittelt. "SAGE WAS" demonstriert die Verschachtelung von MAKE-Anweisungen und das letzte Beispiel zeigt eine Vorwärtsreferenz. Es gibt sehr viele Anwendungen, die sich sehr elegant mit der DOER/MAKE-Konstruktion programmieren lassen.

Befehle:

* DOER: definiert ein Vektorwort
* MAKE:
  - in einer Definition:
    + : Definitions-Name MAKE DOER-Name FORTH-Code ;
  - bei Direktausführung:
    + MAKE DOER-Name FORTH-Code M;
* NOTHING: ein Wort, das nichts tut
* ;AND: ermöglicht die Fortsetzung des Codes eines Wortes nach MAKE
* UNDO: Gebrauch: UNDO DOER-Name; DOER-Name macht nichts und dient sicherzustellen, daß DOER-Name ausführbar ist

8. Befehlsübersicht

Abkürzungen für den Stack-Kommentar

n (number)	einfach-lange Zahl mit Vorzeichen
d (double)	doppelt-lange Zahl mit Vorzeichen
u (unsigned)	einfach-lange Zahl ohne Vorzeichen
ud (unsigned double)	doppelt-lange Zahl ohne Vorzeichen
t (triple)	dreifach-lange Zahl
q (quadruple)	vierfach-lange Zahl
c (character)	8-Bit Zeichenwert
b (byte)	8-Bit Byte
?	Bool'sche Flagge, oder
t= (true)	wahr
f= (false)	falsch
a (address)	Adresse
cfa	Codefeldadresse
pfa	Parameterfeldadresse
'	(vor einem Wort) Adresse von
s d	(als Paar) Source Destination oder "von - nach"
lo	(=unteres) Limit
hi	(=oberes) Limit
u	Count
o	Offset oder Abstand
i	Index
m	Maske
Name	Zeichenfolge, von einem Leerzeichen begrenzt
x	"unwichtig" (Datenstruktur-Notation)

Befehl — Stack
Erklärung

Stack

2DROP (d --)
entfernt das oberste Zahlenpaar des Stacks

2DUP (d -- d d)
verdoppelt (dupliziert) das oberste Zahlenpaar

2OVER (d1 d2 -- d1 d2 d1)
kopiert das zweite Zahlenpaar nach oben

2ROT (d1 d2 d3 -- d2 d3 d1)
rotiert das dritte Zahlenpaar nach oben

2SWAP (d1 d2 -- d2 d1)
vertauscht die beiden obersten Zahlenpaare

>R (n --)
legt die oberste Zahl vom Stack auf den Returnstack

-DUP (n -- n n) oder (0 -- 0)
dupliziert n nur, wenn n ungleich null ist

OVER (n1 n2 -- n1 n2 n1)
kopiert die zweite Zahl nach oben

```
R                       ( -- n)
   kopiert die oberste Zahl vom Returnstack auf den Stack
R>                      ( -- n)
   legt die oberste Zahl des Returnstacks auf den Stack
ROT                     ( n1 n2 n3 -- n2 n3 n1)
   rotiert die drittoberste Zahl nach oben
SWAP                    ( n1 n2 -- n2 n1)
   vertauscht die beiden obersten Zahlen
```

Vergleichsbefehle

```
0<                      ( n -- ?)
   ? ist wahr, wenn n<0
0=                      ( n -- ?)
   ? ist wahr, wenn n=0
<                       ( n1 n2 -- ?)
   ? ist wahr, wenn n1<n2
=                       ( n1 n2 -- ?)
   ? ist wahr, wenn n1=n2
>                       ( n1 n2 -- ?)
   ? ist wahr, wenn n1>n2
```

Speicher

```
!                       ( n a -- )
   speichert n in der Adresse a
+!                      ( n a -- )
   der Inhalt der Adresse a wird um n erhöht
@                       ( a -- n)
   ersetzt die Adresse durch ihren Inhalt
BLANKS                  ( a # -- )
   füllt # Bytes ab Adresse a mit Leerzeichen
C!                      ( b a -- )
   speichert ein Byte in die Adresse a
C@                      ( a -- b)
   liest den Byte-Inhalt der Adresse a
CMOVE                   ( s d # -- )
   kopiert einen # langen Speicherbereich von Adresse s nach d,
   das erste Byte wird zuerst kopiert
CMOVE>                  ( s d # -- )
   wie CMOVE, aber das letzte Byte wird zuerst kopiert
ERASE                   ( a # -- )
   füllt # Bytes ab a mit 0
FILL                    ( a # b -- )
   füllt # Bytes ab a mit b's
TOGGLE                  ( a b -- )
   der Byte-Inhalt von a wird XOR mit b genommen und in a
   gespeichert
```


Arithmetische Funktionen

```
*                       ( n1 n2 -- Produkt)
   mutlipliziert n1 mit n2
*/                      ( n1 n2 n3 -- n4)
   n4=(n1*n2)/n3, das Zwischenergebnis ist 32-Bit lang
*/MOD                   (n1 n2 n3 -- Rest Quotient)
   wie */, aber zusätzlich ist auch der Rest auf dem Stack
+                       ( n1 n2 -- Summe)
   addiert die Zahlen n1 und n2
-                       ( n1 n2 -- Differenz)
   subtrahiert n2 von n1
/                       ( n1 n2 -- Quotient)
   n1/n2
/MOD                    ( n1 n2 -- Rest Quotient)
   Quotient und Rest der Division
1+                      ( n -- n+1)
   addiert 1
1-                      ( n -- n-1)
   subtrahiert 1
2*                      ( n -- 2*n)
   multipliziert mit 2 (arithmetic left shift)
2+                      ( n -- n+2)
   addiert 2
2-                      ( n -- n-2)
   zieht 2 ab
2/                      ( n -- n/2)
   dividiert durch 2 (arithmetic right shift)
ABS                     ( n -- abs(n))
   hinterläßt den Absolutwert einer Zahl (Betrag)
D+                      ( d1 d2 -- d-Summe)
   addiert zwei 32-Bit Zahlen
D-                      ( d1 d2 -- d-Differenz)
   subtrahiert zwei 32-Bit Zahlen
DABS                    ( d -- abs(d))
   berechnet den Absolutwert einer 32-Bit Zahl
DMINUS                  ( d -- -d)
   wechselt das Vorzeichen einer 32-Bit Zahl
M*                      ( n1 n2 -- d-Produkt)
   multipliziert 16-Bit Zahlen, das Produkt ist 32 Bit lang
M/MOD                   ( d n -- d-Rest d-Quotient)
   dividiert eine 32-Bit durch eine 16-Bit Zahl
MAX                     ( n1 n2 -- max)
   läßt die größte der beiden Zahlen auf dem Stack
MIN                     ( n1 n2 -- min)
   läßt die kleinste der beiden Zahlen auf dem Stack
MOD                     ( n1 n2 -- Rest)
   hinterläßt den Rest von n1/n2
MINUS                   ( n -- -n)
   wechselt das Vorzeichen
S->D                    ( n -- d)
   wandelt eine 16-Bit in eine 32-Bit Zahl um
```

U* (u1 u2 -- ud-Produkt)
multipliziert zwei vorzeichenlose 16-Bit Zahlen, das Ergebnis ist 32-Bit lang
U/ (ud u1 -- u-Rest u-Ergebnis)
wie M/MOD, aber alle Zahlen werden vorzeichenlos behandelt

Umwandlung

(ud -- ud)
wandelt eine Zahlenstelle in ein ASCII-Zeichen um
#> (ud -- a #)
beendet die Zahlenformatierung
#S (ud -- 0 0)
wandelt die restlichen stellen in ASCII-Zeichen um
<# (ud -- ud)
Beginn der Zahlenformatierung
HOLD (c --)
fügt das ASCII-Zeichen c in die Zeichenkette ein
SIGN (n ud --)
wird unmittelbar von #> gebraucht, der gebildeten Zeichenkette wird ein "-" vorangesetzt, wenn n negativ ist

Ein- und Ausgabe

. (n --)
gibt n aus
." xxx" (--)
schreibt die Zeichenkette xxx, die zweiten Anführungszeichen bezeichnen das Ende der Zeichenkette
.R (n1 n2 --)
gibt n1 rechtsbündig in einem n2 stelligen Feld aus
CR (--)
setzt den Cursor an den Anfang der nächsten Zeile
COUNT (a -- a+1 #)
ändert die Adresse einer Zeichenkette, deren Länge in ihrem ersten Byte gespeichert ist, und fügt das Längenbyte hinzu, so daß die beiden Parameter als Argument für TYPE, CMOVE, etc. dienen können
D. (d --)
gibt eine 32-Bit Zahl aus
D.R (d n --)
schreibt eine 32-Bit Zahl mit Vorzeichen rechtsbündig in ein Feld der Größe n
EMIT (c --)
schreibt das ASCII-Zeichen c
EXPECT (a # --)
erwartet die Eingabe von # Zeichen (oder einem RETURN) und speichert ihren Code ab a im Speicher
KEY (-- c)
wartet auf eine Eingabe und legt deren Code auf den Stack

KEY? (-- ?)
? ist wahr, wenn eine Taste gedrückt wurde
SPACE (--)
gibt ein Leerzeichen aus
SPACES (n --)
gibt n Leerzeichen aus
?TERMINAL (-- ?)
siehe KEY?
TYPE (a # --)
gibt # Zeichen aus, die ab a gespeichert sind
U. (u --)
gibt u als vorzeichenlose Zahl aus
WORD (c --)
liest eine Zeichenkette aus dem Eingabepuffer, bis es den ASCII-Code c findet, die Kette wird im Speicher bei HERE abgelegt, wobei das erste Byte die Länge der Zeichenkette enthält

Kontrollstrukturen

BEGIN ... AGAIN (--)
endlose Schleife
BEGIN ... UNTIL UNTIL: (? --)
die Schleife wird solange durchlaufen, bis ? wahr ist
BEGIN XXX WHILE: (? --)
WHILE YYY REPEAT
XXX wird immer ausgeführt, YYY nur, wenn ? wahr ist, die Schleife wird beendet wenn ? falsch ist
DO ... LOOP DO: (Grenze Index --), LOOP: (--)
Schleifen struktur mit festgelegten Indexgrenzen bei jedem Durchlauf wird der Index um 1 erhöht
DO ... +LOOP DO: (Grenze Index --), +LOOP: (n --)
wie DO ... LOOP, aber n wird zum Index addiert
I (-- i)
kopiert den Schleifenindex auf den Stapel (oberste Zahl auf dem Returnstack)
IF XXX THEN IF: (? --)
XXX wird ausgeführt, wenn ? wahr ist
IF XXX IF: (? --)
ELSE YYY ELSE: (--)
THEN
XXX wird ausgeführt, wenn ? wahr ist, sonst YYY
J (-- i)
kopiert den Schleifenindex der nächst äußeren Schleife auf den Stack (drittoberste Zahl auf dem Returnstack)
LEAVE (--)
verläßt die Schleife beim nächsten LOOP oder +LOOP

Compiler-Worte und Definitionsworte

' \ Name (-- pfa)
liefert die Parameterfeldadresse des Wortes Name
(XXX) (--)
veranlaßt den Textintepreter, den Text XXX bis zum Abschlußzeichen ")" zu übergehen
, (n --)
speichert n in die nächste verfügbare Speicherzelle des Lexikons
: \ Name (--)
Beginn der Definition von Name
; (--)
Ende der Definition
;CODE (--)
steht innerhalb der Definition von Definitionswörtern und markiert das Ende des Compilierzeitverhalten und den Beginn des Laufzeitverhaltens, die Laufzeitanweisungen werden in FORTH-Assembler geschrieben
<BUILDS (--)
erzeugt einen Lexikoneintrag (nur Kopf und Code Pointer)
ALLOT (n --)
vergrößert das Parameterfeld des zuletzt definierten Wortes um n Bytes
C, (b --)
speichert b in die nächste verfügbare Speicherstelle
C; (--)
Ende der CODE-Definition
CODE \ Name (--)
beginne Assembler-Definition
COMPILE \ Name (--)
kompeliere die cfa von Name in die Quelldefinition, während der Laufzeit wird Name ausgeführt
CONSTANT \ Name (n --)
erzeugt eine Konstante Name mit dem Wert n
CREATE \ Name (--)
trägt Name ins Lexikon ein, mit HERE als cfa, Bit 8 der Namenlänge ist gesetzt
DOES> Laufzeit: (-- a)
wie ;CODE, aber die Laufzeitanweisungen werden in FORTH-Hochsprache geschrieben, zur Laufzeit wird die pfa des definierten Wortes auf dem Stack gelegt
LITERAL Compilierzeit: (n --)
Laufzeit: (-- n)
wird nur innerhalb einer Doppelpunkt-Definition verwendet, während der Compilierzeit wird eine Zahl als "Literal" ins Lexikon eingetragen, während der Laufzeit wird diese Zahl auf den Stack gelegt
TIB (-- a)
Anfang des Eingabepuffers
VARIABLE Compilierzeit: \ Name (n --)
Laufzeit: (-- a)
erzeugt eine Variable Name mit dem Wert n

VOCABULARY \ Name (--)
erzeugt das Vokabular Name
[(--)
schaltet den Compiler aus
[COMPILE] \ Name (--)
Name wird in die Definition kompiliert, auch wenn es ein IMMEDIATE-Wort ist!
\ (--)
Rest der Zeile ist Kommentar
] (--)
schaltet den Compiler an

Disk I/O

<CONTROL>+<,> (--)
Ende eines Blocks
--> (--)
läd den nächsten Block
.LINE (Zeilen# Block# --)
gibt Zeile Zeilen# von Block Block# auf dem Bildschirm aus
;S (--)
Ende eines Blocks
B/BUF (-- n)
Konstante, Blockgröße in Byte
B/SCR (-- n)
Konstante, Blöcke pro Screen
BLOCK (u -- a)
Block u in Buffer bei a sichern
BUFFER (u -- a)
sucht den nächsten Block-Puffer aus und schreibt u in dessen Statuszelle ohne Block u zu laden
DR0 (--)
Diskettenlaufwerk 1 wird aktuelles und PHYSOFF erhält den Wert 40
DR1 (--)
Diskettenlaufwerk 2 wird aktuelles und PHYSOFF erhält den Wert 0
EMPTY-BUFFERS (--)
löscht die Blockpuffer
FLUSH (--)
speichert die als geändert markierten Blöcke ab
INDEX (s d --)
listet die erste Zeile von den Blöcken s bis d
LIST (u --)
listet Block u
LOAD (u --)
läd Block u
SAVE-BUFFERS (--)
siehe FLUSH
UPDATE (--)
markiert einen Block als geändert

Uservariablen

BASE (-- a)
enthält die aktuelle Zahlenbasis

BLK (-- a)
legt die Blocknummer des Blocks auf den Stack, der gerade kompiliert wird, bei der Ausführung von Befehlen, die per Tastatur eingeben worden sind, ist BLK null

CONTEXT (-- a)
zeigt auf das Vokabular, das durchsucht wird

CURRENT (-- a)
zeigt auf das Vokabular, an welches neue Definitionen angehängt werden

DP (-- a)
enthält den Wert, den HERE auf den Stack legt

IN (-- a)
zeigt auf die aktuelle Position im Eingabetext

INPT (-- a)
enthält das zuletzt eingegebene Zeichen

OFFSET (-- a)
enthält eine Zahl, die von BLOCK zu der ausgewählten Blocknummer dazuaddiert wird, bevor dieser geladen wird

PHYSOFF (-- a)
spielt nur bei Single-Density eine Rolle, enthält den "Abstand" zum Diskettenanfang in Blöcken, wird von RESET (DRO) auf 40 gesetzt und erhält jedesmal, wenn die BREAK-Taste oder wann immmer "Grafik-FORTH" ausgegeben wird, den Wert 40

STATE (-- a)
gibt den Systemstatus an (ob kompiliert (192) oder ausgeführt (0) wird)

WARNING (-- a)
enthält ein Flag, das anzeigt, was bei einem Fehler gemacht werden soll

Betriebssystem

?ERROR (Fehlernummer ? --)
wenn ? wahr ist, wird die Fehlernummer oder der entsprechende Text ausgegeben oder ABORT wird durchgeführt

?STACK (-- ?)
? ist wahr, wenn zu viele Werte auf dem Stack liegen

ABORT (--)
bricht alles ab und der Stack wird geleert, PHYSOFF erhält den Wert 40!!!

ASSEMBLER (--)
macht ASSEMBLER zum CONTEXT-Vokabular

COLD (--)
System in Startbedingung zurücksetzen (wird bei jedem Start (Warm- (=<RESET> drücken) und Kaltstart(=einschalten des Computers)) durchgeführt.

DECIMAL (--)
Zahlenbais: Dezimal

DEFINITIONS (--)
das CONTEXT-Vokabular wird ebenfalls zum CURRENT-Vokabular

EDITOR (--)
macht EDITOR zum CONTEXT-Vokabular

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

EXIT (--)
CREATE EXIT ' ;S , SMUDGE

FORTH (--)
macht FORTH zum CONTEXT-Vokabular

GRAFIK (--)
macht GRAFIK zum CONTEXT-Vokabular

HERE (-- a)
legt die Adresse des nächsten freien Platzes im Lexikon auf den Stack

HEX (--)
Zahlenbasis: Hexadezimal

INTERPRET (--)
interpretiert den Eingabetext (wenn BLK=0, dann Tastatureingabe, sonst BLK=Blocknummer des zu interpretierenden Textes) ab der Stelle, auf die IN zeigt

PAD (-- a)
hinterlegt die Anfangsadresse des Textpuffers

QUIT (--)
bewirkt die Rückkehr in die äußere FORTH-Schleife, beide Stacks werden geleert und es wird eine neue Eingabe erwartet, es wird kein "oK" ausgegeben

RAM (--)
schaltet das Betriebssytem-ROM aus und der Bereich von hexadezimal C000 bis FFFF wird RAM, man kann nun alle Grafikbefehle bis auf HCOPY benutzen

RESET (--)
siehe ABORT

INSTALL (--)
installiert die Grafikroutinen

ROM (--)
schaltet das Betriebsystem-ROM wieder ein und somit wird der Bereich von hexadezimal C000 bis FFFF ROM, gleichzeitig kann man nicht mehr die Grafikbefehle bis auf HCOPY benutzen

9. Stilkonventionen

Die kopierten Seiten sind aus dem Buch "In FORTH denken", welches im Carl Hanser Verlag erschienen ist. Die Bestellnummer dieses sehr guten Buches finden Sie im Vorwort.

Anhang E
Zusammenfassung der Stilkonventionen

Unter Voraussetzung der Quellenangabe kann der Inhalt dieses Anhangs ohne Einschränkung reproduziert und weitergegeben werden.

Regeln zu Leerschritten und zum Einrücken

1 Leerschritt zwischen Doppelpunkt und Namen,

2 Leerschritte zwischen Namen und Kommentar*,

2 Leerschritte oder ein Wagenrücklauf und Zeilenvorschub nach dem Kommentar und vor der Definition*,

3 Leerschritte zwischen Namen und Definition, wenn kein Kommentar gegeben wird,

3 Leerschritte einrücken bei jeder Folgezeile (oder ein Vielfaches von 3 bei verschachteltem Einrücken),

1 Leerschritt zwischen Worten/Zahlen in einem Ausdruck,

2 oder 3 Leerschritte zwischen Ausdrücken,

1 Leerschritt zwischen dem letzten Wort und dem Semikolon,

1 Leerschritt zwischen dem Semikolon und IMMEDIATE (wenn es aufgerufen wird)

Keine Leerzeilen zwischen Definitionen, außer wenn es darum geht, besondere Definitionengruppen optisch voneinander zu trennen.

Abkürzungen für den Stack-Kommentar

n	(number)	einfach-lange Zahl mit Vorzeichen
d	(double)	doppelt-lange Zahl mit Vorzeichen
u	(unsigned)	einfach-lange Zahl ohne Vorzeichen
ud	(unsigned double)	doppelt-lange Zahl ohne Vorzeichen
t	(triple)	dreifach-lange Zahl
q	(quadruple)	vierfach-lange Zahl
c	(character)	7-Bit Zeichenwert
b	(byte)	8-Bit Byte
?		Bool'sche Flagge, oder
t=	(true)	wahr
f=	(false)	falsch

* Eine oft gefundene Alternative ist die, 1 Leerschritt zwischen Namen und Kommentar und 3 Leerschritte zwischen Kommentar und Definition zu setzen. Eine großzügigere Technik benutzt 3 Leerschritte vor und nach dem Kommentar. Wofür Sie sich auch entscheiden mögen, bleiben Sie konsistent.

a oder adr	Adresse
cfa	Codefeldadresse
pfa	Parameterfeldadresse
'	(als Präfix) Adresse von
s d	(als Paar) Source Destination oder „von - nach"
lo hilo	für lower (= unteres) Limit
hi	für high (= oberes) Limit (inklusive)|≠ Count
o	Offset oder Abstand
i	Index
m	Maske
x	„unwichtig" (Datenstruktur-Notation)

Ein „Offset" ist ein in absoluten Einheiten ausgedrückter Unterschied, beispielsweise in „Bytes".

Ein „Index" ist ein in logischen Einheiten ausgedrückter Unterschied, beispielsweise „Elemente" oder „Records".

Festlegungen zum Eingabetext-Kommentar

c	Einzelzeichen mit abschließendem Leerzeichen
Name	Zeichensequenz mit abschließendem Leerzeichen
Text	Zeichensequenz, abgeschlossen durch ein bestimmtes Zeichen - kein Leerzeichen.

Beenden Sie einen „Text" mit dem tatsächlich erforderlichen Begrenzer, z.B. Text" oder Text) .

Musterbeispiele für guten Kommentarstil

Sie finden nachstehend zwei Quelltextblöcke als Musterbeispiele guten Kommentarstils.

```
Block # 127
 0 \ Formatierer          Datenstrukturen -- s.2          06.04.85
 1  6 CONSTANT KOPFZEILE   \ #Zeilen Textanfang
 2 55 CONSTANT ENDZEILE    \ #Zeilen Textende
 3
 4 CREATE TEXTANFANG  82 ALLOT
 5 \       ( 1links ! 1rechts ! 80textanfang )
 6 CREATE TEXTENDE  82 ALLOT
 7 \       ( 1links ! 1rechts ! 80textende )
 8
 9 VARIABLE WAAGRECHT   \ aktuelle horizontale Position des Formatierers
10 VARIABLE SENKRECHT   \ aktuelle vertikale Position des Formatierers
11 VARIABLE LINKS       \ aktueller prinzipieller linker Rand
12 VARIABLE WALL        \ aktueller prinzipieller rechter Rand
13 VARIABLE WALL-WAR    \ WALL bei Beginn Formatierung der akt. Zeile
14
15
```

```
Block # 127
 0 \ Formatierer        Positionierung -- S.1          06.06.85
 1 : SKIP  ( n) WAAGRECHT +! ;
 2 : NEULINKS  \ linken Rand neu einrichten
 3   LINKS @ PERMANENT @ @ +  TEMPORAER @ + WAAGRECHT ! ;
 4 : \ZEILE   \ neue Zeile anfangen
 5    EINGANG  CR'  1 SENKRECHT +!  NEULINKS  WALL @ WALL-WAR ! ;
 6 : OBEN?  ( -- T=oben)  OBRAND  SENKRECHT @ = ;
 7 : >OBRAND  \ von der Zeile an den oberen Rand gehen
 8    0 SENKRECHT !  BEGIN \ZEILE OBEN? UNTIL ;
 9
10
11
12
13
14
15
```

Konventionen der Namensgebung

Bedeutung	*Form*	*Beispiel*
Arithmetik		
Integerzahl 1	1Name	1+
Integerzahl 2	2Name	2*
Annahme relativer Eingabeparameter	+Name	+DRAW
Annahme skalierter Eingabeparameter	*Name	*DRAW
Compilation		
Start des „Hochsprachen" Codes	Name:	CASE:
Ende des „Hochsprachen" Codes	;Name	;CODE
Etwas in das Lexikon legen	Name,	C,
Kommt in der Compilationszeit zur Ausführung	[Name]	[COMPILE]
kleine Abweichungen	Name'	CR'
Interne Form oder Primitivwort	(Name) oder <Name>	(TYPE) <TYPE>
Laufzeitteil im Compilationswort:		
Systeme ohne „Folding"	Kleinschreibung	if
Systeme mit „Folding"	(NAME)	(IF)
Definitionswort	:name	:FARBE
Blocknummer, an der ein Overlay beginnt	NamING	DISKING
Datenstrukturen		
Tabelle oder Array	Namen	MITGLIEDER
Totale Anzahl der Elemente	#Name	#MITGLIEDER
aktuelle Elementnummer (Variable)	Name#	MITGLIEDS#
Setzen des aktuellen Elements	(n) Name	13MITGLIED
Vorrücken zum nächsten Element	+Name	+MITGLIED

Bedeutung	*Form*	*Beispiel*
Offsetgröße zum Element vom Strukturanfang	Name+,	DATUM+
Größe (in Bytes pro) von (Kurzform für BYTES/Name)	/Name	/MITGLIED
Indexzeiger	>Name	>IN
Umwandeln der Strukturadresse in die Elementadresse	>Name	>BODY
Dateienindex	(Name)	(PERSONAL)
Dateizeiger	-Name	-BERUF
Initialisieren der Struktur	0Name	0RECORD
Richtung, Umwandlung		
rückwärts	Name<	SLIDE<
vorwärts	Name>	CMOVE>
von	<Name	<TAPE
nach	>Name	>TAPE
umwandeln in	Name>Name	PFENNIGE>MARK
nach unten	\Name	\ZEILE
nach oben	/Name	/ZEILE
öffnen	|Name	|DATEI
schließen	|Name	|DATEI
Logik, Kontrolle		
Zurückbringen einer Bool'schen Flagge	Name?	KURZ?
Zurückbringen einer reversen Bool'schen Flagge	-Name?	-KURZ?
Adresse eines Bool'schen Wertes	'Name?	'KURZ?
Beinhaltet konditionelle Operation	?Name	?DUP (auch DUP)
Aktivieren ?	+Name	+CLOCK
oder: fehlendes Symbol	Name	BLINKING
Stilllegen, Deaktivieren	-Name	-CLOCK-BLINKING
Speicher		
Sichern des Wertes von	@Name	@CURSOR
Zurückbringen des Wertes von	!Name	!CURSOR
Speichern in	Name!	SEKUNDEN!
Holen von	Name@	INDEX@
Buffername	:Name	:INSERT
Adresse von Name	'Name	'S
Zeigeradresse auf Name	'Name	'TYPE
Austausch, insbes. Bytes	>Name<	>MOVE<

Bedeutung	*Form*	*Beispiel*
Numerische Typen		
Bytelänge	*CName*	*C@*
Größe von 2 Zellen, 2er-Komplement	DName	D+
Mischoperator 16- und 32-Bit	MName	M*
Größe von 3 Zellen	TName	T*
Größe von 4 Zellen	*QName*	*Q**
Encoding ohne Vorzeichen	*UName*	*U.*
Ausdrucken des Elements	.Name	.S
numerisches Drucken	Name.	D. , U.
rechtsbündiges Drucken	Name.R	U.R
Menge		
„pro"	*/Name*	*/SEITE*
Reihenfolge		
Start	<Name	<#
Ende	Name>	#>
Text		
Es folgt ein String mit abschließenden Anführungszeichen	Name"	ABORT" Text"
Text- oder Stringoperator (ähnlich dem $-Präfix in BASIC)	"Name	"COMPARE
Superstring Array	"Name"	"FARBEN"

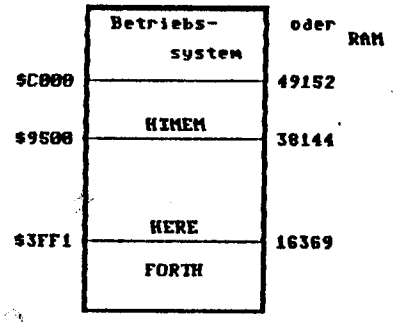
Wie man die Symbole ausspricht

! Speichern oder englisch „store"
@ Holen oder englisch „fetch"
# Nummer oder „Sharp"
$ Dollar
% Prozent
^ ein "Dach" oder englisch "Caret"
&. . ampersand?
* Sternchen, Asterisk oder englisch "star"
(Runde Klammer-Auf
) Runde Klammer-Zu
- Bindestrich oder englisch "dash"
+ Plus
= Gleich
{ } Geschweifte Klammern
[ ] Eckige Klammern
" Anführungszeichen
' als Präfix: Tick (deutsch: Hükchen), als Suffix: prime
~ Tilde
| Barzeichen
\ "Backslash" (für "nach unten", "weglassen")

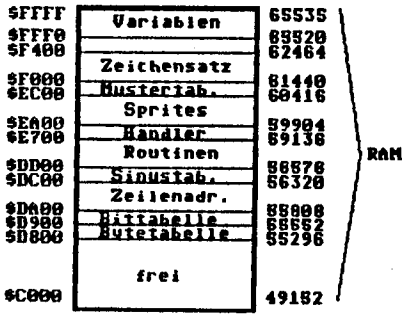
/ "Slash" oder "Schrägstrich" (für "nach oben")
< *kleiner als (left dart).*
> größer als (right dart)
? Fragezeichen (manchmal auch englisch "query")
, Komma
. *Punkt oder englisch "Dot"*

10. Speicherpläne

Speicheraufteilung



Bereich von $C000 bis $FFFF



11. Stackzeichnung

Falls Sie eine verzwickte Situation auf dem Stack lösen müssen, ist es praktisch dies mit Hilfe einer Stackzeichnung zu tun. Auf der nächsten Seite finden Sie ein Beispiel für eine Stackzeichnung und auf der übernächsten Seite ist ein leeres Formular. Sie können dieses fotokopieren und dann neue Wörter mit dessen Hilfe erstellen.

Wortname: 'ZEILE	Programmierer: RAI	Datum: 07.08.90
Operationen:	Stackeffekte:	Returnstack:
/// // / / / / / ///// / / / / /	-- a	
ZEILEN# c	Zeile	
RAM	Zeile	
DUP	Zeile Zeile	
SCRHI +	Zeile a1	
c@ 256*	Zeile 'Zeile-hi	
SWAP	'Zeile-hi Zeile	
SCRLO +	'Zeile-hi a2	
c@	'Zeile-hi 'Zeile-lo	
+	'Zeile	
ROM ;	a='Zeile	

Wortname:	Programmierer:	Datum:
Operationen:	Stackeffekte:	Returnstack:

Editor-Referenzliste

a. <CONTROL>+Taste drücken
b. andere Tastaturkommandos

zu a.
* + -> links
* * -> rechts
* - -> hoch
* = -> runter
* H -> Home (springt an den Blockanfang)
* Q -> springt an den linken Rand einer Zeile
* I -> schaltet den Einfügemodus an
* V -> Von (Anfang eines auszuschneidenden Bereiches)
* B -> Bis (Ende eines auszuschneidenden Bereiches)
* S -> Setzen (setzt den ausgeschnittenen Bereich ab der Cursorposition ein)
* D -> gibt den Stempel aus
* <DELETE-BACKSPACE> -> löscht ein Zeichen rechts vom Cursor
* ; -> geschweifte Klammer-Auf
* < -> geschweifte Klammer-Zu
* 3 -> springt aus dem Texteditor in die Kommandozeile
* > -> fügt ein Leerzeichen in den Text ein

zu b.
* <CAPS> -> schaltet zwischen Groß- und Kleinschreibung um
* <TAB> -> springt um 3 Zeichen vor
* <RETURN> -> springt an den Anfang der nächsten Zeile
* <SHIFT>+<CONTROL>+W -> wechseln <=> Tastaturkommandos aus
* <INVERS> -> schaltet zwischen Normal- und Inversdarstellung um
* <DELETE-BACKSPACE> -> löscht ein Zeichen links vom Cursor und springt dorthin

Besonderheiten im Einfügemodus:
* <TAB> -> fügt 3 Leerzeichen ein
* <DELETE-BACKSPACE> -> alle Zeichen rechts vom Cursor "rutschen" nach
* <CONTROL>+I -> der Einfügemodus wird verlassen

Stempel

Beispiel: DATUM= RAI 09.08.1989

Editorbefehle

* ED -> Editieren eines Blockes
* FH -> speichert alle "geänderten" Blöcke ab
* UE -> übernimmt einen Block als geändert in den Blockpuffer
* B -> geht einen Block zurück
* L. -> n L. listet den Block n im Blockfenster
* L -> listet den aktuellen Block
* N -> der nächste Block wird zum aktuellen
* W -> wechselt zum Block im anderen Blockpuffer
* S" -> s d S" <TEXT>" sucht <TEXT> und gibt die Fundstellen aus
* WIPE -> der aktuelle Block wird gelöscht und als geändert markiert