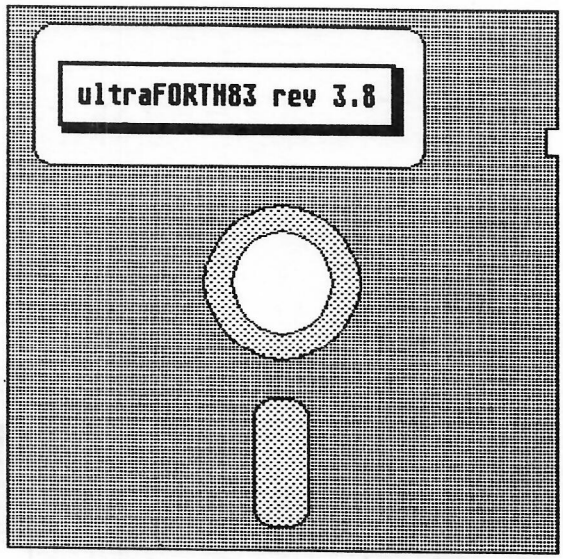


HANDBUCH

für Commodore

C16/116 (min. 32 KByte) Plus 4

C64 SX64 C128

Handbuch zum ultraFORTH83 rev 3.8
2. Auflage 1.11.1987

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

INHALTSVERZEICHNIS

Teil 1 - Erläuterungen

Teil 2 - Glossare

VERZEICHNIS DER ABBILDUNGEN

Prolog : über das ultraFORTH83

ultraFORTH83 ist eine Sprache, die in verschiedener Hinsicht ungewöhnlich ist. Einen ersten Eindruck vom ultraFORTH83 und von unserem Stolz darüber soll dieser Prolog vermitteln.

ultraFORTH83 braucht nicht geknackt oder geklaut zu werden. Im Gegenteil, wir hoffen, daß viele Leute das ultraFORTH83 möglichst schnell bekommen und ihrerseits weitergeben.

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

Wir haben ein Forth für verschiedene Rechner entwickelt. Das Ergebnis ist das ultraFORTH83, eines der besten Forthsysteme, das es gibt.
Nachdem Version 3.5 für den C64 erhältlich war, wurde es auf andere Rechner (Atari ST, Schneider CPC, CP/M-Computer, IBM PC und Kompatible) übertragen und weiter entwickelt.
Eine Anpassung an die anderen "kleinen" Commodore-Rechner wie C16, C116 und Plus4 wurde oft vermißt, denn eine andere einfache, billige und trotzdem gute Sprache für diese Rechner existiert anscheinend nicht. Tatsächlich gibt es bis jetzt (November '87) außer dem eingebauten BASIC nur noch drei Assembler (mit spezifischen Schwächen) und einen BASIC-Compiler. ultraFORTH83 kann diese preiswerten Rechner etwas attraktiver machen. Bei der Entwicklung wurde darauf geachtet, daß die im C16-System vorhandenen Funktionen zusammen mit ultraFORTH83 nutzbar bleiben.
Das Ergebnis:

- I/O-Routinen und Interrupt-Handling des Betriebssystems sind integriert
- Kernal/Betriebssystem sind unter ultraFORTH83 voll nutzbar
- Monitor kann vom ultraFORTH83 aus aufgerufen werden.
- Grafik (bisher nicht nutzbar, ev. möglich)
- Basic (bisher nicht nutzbar, ev. möglich)
- +4-Software (bisher nicht überprüft)

Während der Anpassungsarbeiten wurde auch die Version für den C64 gründlich überarbeitet und präsentiert sich nun deutlich verbessert. Stark verbessert wurden u.a. der Tracer und

SAVESYSTEM . Natürlich sollten die beiden Versionen so ähnlich wie irgend möglich sein, um die Portabilität und Wartbarkeit zu verbessern. Das Ergebnis haben Sie vielleicht schon bemerkt: es gibt nur einen Satz Disketten und nur einen Quelltext für beide Rechner. Rechnerspezifische Teile konnten in den gemeinsamen Quelltext integriert werden.

Das ultraFORTH83 enthält auf allen Rechnern : Multitasker, Heap (für namenlose Worte), Dekompiler, Assembler und Editor. An vielen Stellen des Systems wurden Zeiger und sog. deferred Worte benutzt, die eine einfache Umgestaltung des Systems für verschiedene Gerätekonfigurationen ermöglichen. Besonderes Augenmerk wurde auf einen extrem schnellen Blockpuffer-Mechanismus gerichtet, damit effiziente Massenspeichermanipulationen möglich werden.

Für den C64 gibt es außerdem Graphik, Sprites, Turtlegraphik und eine Menge Demoprogramme, für den C16 eine Kassettenschnittstelle mit Schnelllader.

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

> *Denn*: Wir behalten uns die kommerzielle Verwertung des ultraFORTH83 vor !

Mit diesem Handbuch ist die Unterstützung des ultraFORTH83 noch nicht zuende. Die VIERTE DIMENSION, Vereinszeitschrift derForth Gesellschaft e.V. c/o. Rainer Mertins
Antilopenstieg 6a
2000 HAMBURG 54
dient als Plattform.

Wenn euch das ultraFORTH83 gefällt, so schickt uns doch eine Spende von ca DM 20,- , denn die Entwicklung des ultraFORTH83 sowie des Handbuchs war teuer (allein einige hundert DM an Telefonkosten), und der Preis, den wir verlangen, deckt nur die Unkosten.

Schickt uns bitte auch Programme, die fertig oder halbfertig sind, Ideen zum ultraFORTH83, Artikel, die in der Presse erschienen sind (schreibt selbst welche !) , kurz: Schickt uns alles, was entfernt mit dem ultraFORTH83 zu tun hat.

Und natürlich : Anregungen, Ergänzungen und Fehler im Handbuch und ultraFORTH83

Für die Autoren des ultraFORTH83 :

Bernd Pennemann, Treitschkestr. 20, 1000 Berlin 41

Hardware-Anforderungen

Das ultraFORTH83 für C16/C64 läuft auf folgenden Rechnern:

C16, C116 mit 16 Kbyte	Gar nicht. Sorry ![1]
C16, C116 mit 32 Kbyte	Nur mit Diskettenlaufwerk
C16, C116 mit 64 Kbyte oder Plus 4	Mit Diskettenlaufwerk oder Kasettenrekorder[2].
C64, SX64	Mit Diskettenlaufwerk, Kassettenrekorder nicht ausprobiert[2].
C128	Im C64 Emulationsmodus mit Diskettenlaufwerk.

[1] Die Ursprungsversion des C16 mit 16kB ist nicht ultraFORTH83-fähig, da allein der FORTH-Kern den Speicher von $1000 bis $4B00 belegen würde.
Der Umbau dieser Rechner auf 64kB kostet im Selbstbau ca. 30DM, fertig gekauft ab 60DM und lohnt sich eigentlich immer. Eine ROM-Version, die mit den 16 KByte Speicher des C16 auskäme, befindet sich in der Überlegung. Ein Mäzen, der uns einen anständigen Arbeitslohn dafür bezahlt, könnte das ganze sehr beschleunigen.

[2] Die Benutzung eines Diskettenlaufwerkes ist allerdings insbesondere für das Arbeiten mit Forth sehr zu empfehlen.

Die ersten Schritte in ultraFORTH83

Diese Beispielssitzung soll die ersten Schritte bei der Benutzung des ultraFORTH83 etwas erleichtern.

Starten des Systems

Als erstes sollten Sie Ihren Rechner aus- und wieder einschalten, um ihn in einen definierten Ausgangszustand zu versetzen. Sollten Sie an einem C16 oder C116 sitzen und nun in der Einschaltmeldung etwas über so ca. '14000 bytes free' lesen, müssen wir Sie enttäuschen: Ihr Rechner ist zu klein. ultraFORTH83 benötigt mindestens 32 Kbyte. Nähere Informationen hierzu finden Sie im Kapitel über "Hardware-Anforderungen".
Als nächstes schalten Sie bitte Ihr Floppylaufwerk ein. Falls Sie kein Floppylaufwerk haben, so müssen Sie sich eins beschaffen, denn zumindest für die erste Sitzung brauchen Sie eins (wo wollen Sie denn sonst Ihre Disketten reinschieben, Sie Schlawiner!). Jene Leser und Leserinnen, die später mit Kassettenrekorder arbeiten möchten, sollten nach dieser Beispielssitzung nochmal die 'Kassetten-Beispielssitzung' durchexerzieren, bevor Sie sich auf den mühevollen Weg des Hinüberkopierens der Quelltexte von den Disketten machen.
Nun wird die erste Diskette eingeschoben. Achten Sie darauf, daß der Klebestreifen sicher auf der Schreibschutzkerbe klebt. Das Inhaltsverzeichnis der Diskette wird von BASIC aus mit:

```
LOAD "$",8
LIST
```

angezeigt. Sie bekommen nun mehrere Files gelistet, von denen sie das für Ihren Rechner bestimmte File laden sollten:

```
LOAD "<filename>",8
```

Hierbei sei gleich bemerkt, daß das Betriebsystem auf den anderen drei Diskettenseiten keine Files findet, weil ultraFORTH83 seine eigene Diskettenverwaltung benutzt.
Wenn Sie sich das geladene Programm mit **LIST** anzeigen lassen würden, erschiene nur eine einzige Zeile auf dem Bildschirm. Aber tun Sie's lieber nicht, denn **LIST** zerstört manche Programme. Geben Sie lieber **RUN** ein.

Es erscheint jetzt - evtl. nach einer Demonstration - die Einschaltmeldung "ultraFORTH83 rev 3.3". Drücken Sie ein paarmal <Return> ; das System quittiert jetzt jeden Tastendruck mit **ok** .

Den Befehlsumfang des ultraFORTH83 können Sie sich mit

```
words  <Return>
```

anzeigen lassen.

Erster Einstieg

Als nächstes wird ein Miniprogramm (in Forth *Wort* genannt) kompiliert. Dazu geben wir ein :

```
: test    ." hallo " ; <Return>
```

Achten Sie bitte darauf, die Zeile unverändert einzugeben. Insbesondere ist das Leerzeichen zwischen ." und **hallo** wichtig! Das System antwortet bei korrekter Eingabe mit **ok** . Ihr soeben kompiliertes Wort können Sie jetzt durch Eintippen seines Namens starten:

```
test <Return>
```

Es erscheint **hallo ok** auf dem Bildschirm. Als nächstes wollen wir unser Miniprogramm erweitern :

```
: test    BEGIN ." hallo" REPEAT ;
```

(Den Hinweis, daß <Return> zu drücken ist, lasse ich im folgenden weg!) Es erscheint der Hinweis **TEST exists** und dann **ok** . Beim Ausführen dieses Wortes **TEST** erscheinen nun unzählige **hallo** auf dem Bildschirm und alle Tastatureingaben werden ignoriert - das System befindet sich in einer Endlos-Schleife.
Das macht nichts: Am C64 werden jetzt die Tasten <run/stop> und <restore> gleichzeitig gedrückt (am besten richtig draufrumhämmern), bis das sinnlose **hallo** verschwunden ist.
Beim C16 wird stattdessen die <stop>-Taste zusammen mit dem <reset>-Schalter gedrückt. Es meldet sich der Monitor, von dem aus mit

```
G1014
```

ein Warmstart des ultraFORTH83 ausgeführt werden kann (siehe hierzu auch **RESTART** im Glossarteil des Handbuches). Nun sollte mit **WORDS** überprüft werden, ob die beiden **TEST** noch da sind.

Ein Inhaltsverzeichnis Ihrer Forth-Quelltextdisketten können Sie sich mit

```
1 list
```

ausgeben lassen. Wenn Sie danach eine andere Diskette einlegen, werden Sie feststellen, daß **1 LIST** immer noch das Inhaltsverzeichnis der ersten Diskette zeigt . Das liegt daran, daß die virtuelle Diskettenverwaltung des Forth (auch *Blockmechanismus* genannt) einige Blöcke im Speicher vorrätig hält, um die Diskettenzugriffe zu minimieren. Daher sollten Sie, um Kuddel-Muddel zu vermeiden, unbedingt vor (!) jedem Diskettenwechsel

```
flush
```

eingeben.

Probieren Sie nun mal

```
1 edit
```

Falls der Editor vorhanden ist, werden Sie zunächst aufgefordert, ihr Kürzel (auch *stamp* genannt) einzugeben. Haben Sie sich noch keines ausgedacht, so drücken Sie nur <Return>. Anschließend erscheint der Screen.

Ist der Editor nicht vorhanden, so suchen Sie sich aus den Forth-Inhaltsverzeichnissen den Editor-Loadscreen heraus. Sie können ihn dann mit

```
<screen-Nummer> load
```

laden, wobei statt <screen-Nummer> die im Inhaltsverzeichnis angegebene Nummer eingesetzt wird. Ihre Diskettenstation sollte dann ca. 5 Minuten laufen, während die Nummern der Screens ausgegeben werden, die gerade geladen werden. Sie haben also genug Zeit, den folgenden Text in Ruhe zu lesen. Starten Sie bitte anschließend den Editor, wie es oben beschrieben wurde. Sollte die Diskettenstation jedoch gleich wieder zur Ruhe gekommen und **ok** ausgegeben worden sein, so haben sie mit großer Wahrscheinlichkeit die von **LIST** erzeugte Zeilennummer statt der screen-Nummer genommen. (Zumindest ist das bei mir immer so.) Sie haben also einen falschen Screen geladen. Hoffentlich nichts schlimmes.

Nun sollten Sie das Kapitel über den Editor lesen. Falls Sie verzweifelt versuchen, ihn zu verlassen, dürfen Sie <run/stop> drücken.
Laden Sie nun nach Belieben Dinge, die Ihnen wichtig erscheinen und schauen Sie sich mit dem Editor die Quelltextscreens an.

Vor allem sollten Sie sich die sogenannten Loadscreens ansehen. Sie enthalten in der Regel einige wichtige Definitionen, wie z.B. Vokabulare. Ferner laden sie alle Screens, die zur Applikation gehören. Man kann also dem Loadscreen ansehen, welche Teile der Diskette zu dieser Applikation gehören. Am Ende des Screens sind oft mit \\ (beim C16/64: zwei Pfundzeichen) wegkommentierte Forthworte wie **SAVE** zu finden. **SAVE** bewirkt, daß das geladene Programm gegen Löschen geschützt ist. Der Aufruf schadet nicht und außerdem schützt er vor "schwer erkennbaren Ursachen von System-Crashs".

Das Laden einiger Programme wird mit Meldungen der Form ?! CODE ?! abgebrochen. Sie bringt zum Ausdruck, daß vor dem Laden der Anwendung erst der Assembler geladen werden muß.

Das ultraFORTH83 wird mit **BYE** verlassen (siehe Glossarteil). Die Version für den C16 landet dabei im Monitor; ebenso wie durch Drücken der <Run/Stop> und <Reset>-Taste. Die "normale" Benutzung des Monitors verursacht keine Probleme bei einem anschließenden Warmstart des ultraFORTH83. Bei alleinigem Drücken von <Run/Stop> ohne <Reset> gelangt man in das BASIC, von dem aus mit **SYS 4116** ein ultraFORTH83-Warmstart ausgeführt werden kann. Das BASIC sollten Sie tunlichst nicht weiter benutzen, da es dieselben Speicherbereiche wie das ultraFORTH83 benutzt und

der Rechner daher bald abstürzt.
Die Version für den C64 landet nach Eingabe von **BYE** sofort im BASIC, von dem aus man mit **SYS 2068** wieder in das ultraFORTH83 gelangt.

Die nächsten Schritte ...

Anpassen der Speicherbelegung

Es sei für C16/C116/Plus4-Benutzer mit 64 Kbyte RAM erwähnt, daß ihre ultraFORTH83-Version nur 32 Kbyte Speicher benutzt. Das prüfen Sie mit

```
limit u.
```

(Bitte den Punkt nicht vergessen). Wenn 32768 ausgegeben wird, werden nur 32 KByte benutzt. Sie ändern das (wenn gewünscht) mit

```
$fd00 ' limit >body !  cold
```

Für die weitere Veränderung der Speicherbelegung finden sie einen Screen 'relocate the System' im Lieferumfang, dem sie auch seine Benutzung entnehmen. Eine Karte der Speicherbelegung des ultraFORTH83 finden Sie auf Seite 49.
Den genannten Screen benötigen Sie auch, wenn Sie beim Laden einer Applikation auf die Meldung "Dictionary full" stoßen. Dann ist der freie Speicher zwischen **here** und **sp@** erschöpft, in dem das System neu definierte Worte ablegt. Sie müssen, um jetzt weiterarbeiten zu können, die zuletzt definierten Worte vergessen. Erscheint nach Drücken der <Return>-Taste die Meldung "still full" , so ist der Speicher immer noch zu voll. Mit dem Wort **BUFFERS** , das Sie auf den Quelltextdisketten finden, können Sie die Zahl der Blockpuffer (unterhalb **LIMIT**) verringern und damit gleichzeitig den freien Platz oberhalb des Dictionary vergrößern. Mit weniger als ca. 3 Puffern wird das Editieren von Quelltexten jedoch sehr zeitraubend.
Im nächsten Abschnitt wird beschrieben, wie Sie das so veränderte System auf der Diskette speichern.

Erstellen eines eigenen Arbeitssystems

Im folgenden beschreibe ich, wie Sie ein eigenes Arbeitssystem (oder als Spezialfall eine Stand-Alone-Anwendung) erzeugen können. Damit ist ein System gemeint, das die Programme und Hilfsmittel enthält, die Sie oft benötigen. Im Prinzip können Sie diese natürlich nach Bedarf zu dem Kern hinzu kompilieren, aber schneller geht es natürlich, wenn schon (fast) alles da ist. Bei der Entwicklung von umfangreichen Anwendungen ist es außerordentlich zeitsparend, wenn man ein System zur Verfügung hat, das bereits die Teile der Applikation enthält, die fehlerfrei sind. Übrigens erfolgen alle Ein-/Ausgaben DEZIMAL. Wem das nicht gefällt: **HEX** eingeben !

Die meisten der für die Erzeugung eines Arbeitssystems notwen-

digen Schritte kennen Sie schon:

1.) Booten des nackten Forth-Kerns (FileName:'___ultraFORTH__')
2.) Prüfen der Speicherbelegung. Die Differenz zwischen
```
here u.       und
s0 @ u.
```
sollte mindestens ca. 1 kByte betragen. Wenn Sie alle Quellen aller Diskettenseiten laden wollen, sollten Sie schon mit 5-6 kByte rechnen.
3.) ggf. Ändern der Speicherbelegung.(s. relocate-screen).
4.) Laden der gewünschten Quellen. Da Sie als Anfänger/in vermutlich noch nicht so recht wissen, was Sie eigentlich brauchen, hier ein paar Beispiele: Editor, Assembler, Tracer und evtl. die Kassettenversion.
Falls Sie den Assembler wirklich dauerhaft im System haben wollen, sollten Sie nicht den 'transienten' Assembler laden. Dieser verschwindet nämlich beim nächsten **SAVE** wieder.
5.) Auf jeden Fall müssen Sie **SAVESYSTEM** laden.
6.) Falls Sie möchten, daß Ihr zu erstellendes System bei jedem Laden gleich irgendetwas ausführt (z.B. unzählige Male **hallo** ausdruckt), dann ist es jetzt nötig, **'COLD** oder **'RESTART** umzudefinieren. Das nennt sich dann "Stand-Alone-Anwendung".
7.) Aufruf von **SAVESYSTEM**

Als Beispiel wird hier ein Stand-Alone-System vorgestellt, das permanent **hallo** druckt:

1.) neu Booten: **LOAD "<filename>",8**
RUN
2.) und 3.) entfällt da unsere Anwendung kaum Speicher braucht.
4.) Die 'Quelle' geben wir von Hand ein:
```
: test    BEGIN ." hallo" REPEAT ;
```
5.) Nun **SAVESYSTEM** laden.
6.) Damit **TEST** sofort nach dem Laden ausgeführt wird, müssen wir folgendes eingeben :
```
' test Is 'cold
```
7.) **savesystem hallo-system**

Das File hallo-system können Sie jetzt immer laden, wenn Sie mal wieder jemand brauchen, der **hallo** zu Ihnen sagt. Sonst kann es leider nichts.

Umgang mit den Disketten

Nun sollten Sie - vermutlich haben Sie bereits einige Systemabstürze hinter sich - Sicherheitskopien Ihrer Disketten (*Back-Ups)* erstellen. Falls Sie stolze(r) Besitzer(in) von zwei Diskettenlaufwerken sind, laden Sie dazu am besten das Wort **COPY2DISK** . Es braucht allerdings ca. eine halbe Stunde pro Diskettenseite. Alle anderen kramen in ihren Diskettenkästen nach dem schnellsten Kopierprogramm. Aber Achtung : Nicht alle Kopierprogramme kopieren wirklich die ganze Diskette. Es funktionieren z.B. 'Quickcopy' und 'sd.backup.xx', das bei neueren Laufwerken von Commodore mitgeliefert wird. Grundsätzlich gilt: Wenn das Programm irgendwelche Filenamen wissen will, ist es schlecht. Wenn es was von 'allocated blocks' oder 'whole Disk' erzählt, ist es gut. (Wenn es was von 'wholy' oder 'hole' Disk spricht, ist sein Englisch schlecht oder es ist gar kein Kopierprogramm.) Wenn Sie nicht für jede Diskettenseite mindestens 4 Diskettenwechsel machen müssen, ist es entweder ein Superprogramm oder es taugt nichts.
Wenn Sie neue Disketten für das Abspeichern von (Ihren eigenen) Forth-Screens vorbereiten möchten, sollten Sie das Wort **FORMATDISK** (im Lieferumfang enthalten) benutzen. Dadurch ist Ihre Diskette vor dem Überschreiben mit Files geschützt. So vorbereitete Disketten dürfen nicht mit dem Validate-Befehl des Diskettenlaufwerkes aufgeräumt werden.

Zur Druckerbenutzung auf dem C16

Bisher habe ich nur den Treiber VC1526 getestet. Er funktioniert mit meinem Citizen-"100 DM"-Drucker, soweit im Zusammenhang mit diesem Drucker von "funktionieren" die Rede sein kann. Da die meisten mir bekannten C16-Besitzer diesen Drucker verwenden, seien hier ein paar Hinweise zur Verringerung der erheblichen Störeinstrahlung gegeben:

- Druckerkabel dick mit Aluminiumfolie umwickeln
- Diskettenlaufwerk bei Druckerbenutzung ausschalten und umgekehrt.

Andere Druckertreiber, die den Userport des C64 benutzen, müßten angepaßt werden. Ich habe nichts angepaßt, da ich die entsprechenden Drucker nicht besitze. Wenn jemand was anpaßt : Bitte eine Diskette mit den Quellen schicken -- wird in Zukunft von uns ~~verbe~~eitet.

ultraFORTH83 für Kassettenrekorder

Die Kassetten-Version ist für C16/C116 (mit 64kB) und Plus4 entwickelt worden, da Käufer dieser Billigrechner oft vor dem Kauf des vergleichsweise teuren Diskettenlaufwerks zurückschrecken. Trotzdem sei noch einmal darauf hingewiesen, daß ein Diskettenlaufwerk für ultraFORTH83 dringend zu empfehlen ist, da einige (viel Speicher beanspruchende) Anwendungsfälle mit Kassettenrekorder gar nicht zu bearbeiten sind und da vor allem kleine Fehleingaben leicht zum Zerstören der selbst geschriebenen - in der Kassetten-Version speicherresidenten - Quellen führt. Also bitte wenigstens öfter abspeichern! Da Kassettenrekorder ebenso wie Peripheriegeräte bestimmter Firmen fürchterlich langsam sind (einer bekannten Heimcomputerfirma wird nachgesagt, daß man mit ihren Geräten kein Programm geladen kriegt, bevor es hoffnungslos veraltet ist) haben wir uns aus Benutzer(innen)freundlichkeit von folgenden Ideen leiten lassen:

- Ein Kassettenschnellader muß her! Supertape
- Wer schon 10 Minuten warten muß, soll wenigstens spazieren gehen können, statt alle 17 Sekunden eine Taste drücken zu müssen! Der Ladevorgang kann ebenso wie der Sicherungsvorgang an einem Stück erfolgen. Diese Automatisierung erfordert einige Fehler-Prüfungen, die hoffentlich alle funktionieren.
- Die (notgedrungen) speicherresidenten Quelltexte sollen auch einen normalen Systemabsturz überleben!
- Die wertvollen Quelltexte sollen auch vor Bedienungsfehlern sicher sein! Entgegen der üblichen Forth-Philosophie werden viele Bedienungsfehler abgefangen.

Die Kassetten-Version besteht aus 4 Teilen:

- Dem ultraFORTH Kern
- Einer Ramdisk, die im Speicher ein Laufwerk simuliert.
- Einer Kassettenschnittstelle
- Dem Programm Supertape, einem Kassettenschnelllader, der mit 3600 baud ca.10 mal so schnell ist wie die Commodoreroutine. Es wird von der Zeitschrift "c't" für alle gängigen Rechner angeboten. Supertape ist bisher leider nur für C16-kompatible Rechner angepaßt. Wir bedanken uns beim Heise-Verlag für die freundliche Genehmigung, Supertape weiterzuverbreiten.

Für Hinweise, die zur Verbesserung der Benutzung oder zur Aufdeckung von Fehlern dienen, wird zwar keine Belohnung ausgesetzt, aber mein Dank wird Euch auf ewig verfolgen. Da ich grade beim Danksagen sind, möchte ich mich nochmal bei Doerte bedanken, die mich mit mit intensiven Gesprächen immer wieder in die schnöde Realität zurückholte und trotz aller Eifersucht den Rechner bisher nicht aus dem Fenster warf.

Eine Beispielssitzung

Sie befinden sich bereits in Forth und haben die Kassettenversion geladen. Ansonsten laden Sie es erstmal von Ihrer Diskette. Als erstes müssen wir einen Teil des Speichers für die Ramdisk räumen. Er muß mindestens ca. 500 bytes lang sein, empfehlenswert sind jedoch etwa 16 Kbyte. Anschließend richten wir dort eine Ramdisk ein. Alle Blockzugriffe auf Drive 1 und folgende gehen dann in die Ramdisk.
Beispiel (für den C16) :

```
$c000 ' limit >body ! cold      $C000 bis $FD00 werden ge-
                                räumt (15 Kbyte)
limit memtop rdnew               .. eine Ramdisk erzeugt
1 drive                         .. auf Drive 1 geschaltet
1 list                          .. und ein Screen gelistet.
supertape                       Aktuelles Gerät setzen
```

Für den C64 heißen die entsprechenden Adressen z.B. $9000 und $C000 statt $C000 und $FD00. Außerdem muß man **COMMODORE** statt **SUPERTAPE** schreiben. Falls später einmal das System abstürzt, können Sie es einfach neu laden, wobei die Ramdisk bestehen bleibt. Voraussetzung ist natürlich, daß sie nicht zerschossen wurde und das neu geladene System sie auch nicht zerstört. Letzteres passiert z.B., wenn beim neuen System **LIMIT** zu groß ist.
Vergleiche hierzu auch: **RDUSE 'COLD 'RESTART**

Wenn der Editor geladen ist, geben Sie ein:

```
2 edit
```

Sie können jetzt z.B. den folgenden Text eingeben wobei Commodore-Benutzer/innen statt '\' ein Pfundzeichen eingeben:

```
\ Mein erster LadeScreen

savesystem erster Versuch"
id" meine kleine Ramdisk"
saveramdisk
autoload on
savesystem Mit autoload"
saveramdisk
savesystem Mit autoload"
autoload off
saveramdisk
```

Wenn Sie nun mit <home> in die linke obere Ecke des Bildschirms gehen und <ctrl> und <L> drücken, wird der beschriebene Screen auf die Ramdisk zurückgeschrieben und geladen. Nun werden ca. 5 Minuten lang Daten auf dem Rekorder gesichert, wobei nach Ausführung jeder der obigen Zeilen der Bildschirm kurz aufleuchtet.

Sie haben (wenn alles glatt gegangen ist) auf ihrer Kassette:

```
erster Ver          - ein Forthsystem (z.B. für Systemabstürze)
RD.meine kleine     - Die Ramdisk mit dem geschriebenen Screen
Mit autolo          - Ein Forthsystem mit Autoload
RD.meine kleine     - Die Ramdisk mit dem geschriebenen Screen
Mit autolo          - Ein Forthsystem mit Autoload
RD.meine kleine     - Die Ramdisk mit dem geschriebenen Screen
```

Hierzu sei bemerkt:

- Da Filenamen nur eine begrenzte Länge haben dürfen, fehlt bei langen Namen der Rest.
- Der Name einer Ramdisk beginnt immer mit "RD.".
- Es ist natürlich nicht nötig alles zweimal abzuspeichern, aber sicherer.
- Die Forthsysteme bestehen (falls Supertape benutzt wird) im Grunde genommen aus drei einzelnen Files (s.u.).
- Die beiden identischen Systeme "Mit autoload" werden nach dem Laden während des Kaltstarts die unmittelbar folgende Ramdisk laden, sofern **TAPEINIT** als **'RESTART** installiert ist. Dies prüfen Sie mit

```
' 'restart >body @ >name .name
```

  und erzwingen es mit

```
' tapeinit Is 'restart
```

Informationen über die aktuelle Ramdisk erhalten Sie mit: **.RD** und **RDCHECK**

Wenn Sie Quelltexte von Diskette auf Kassette kopieren möchten, benutzen Sie **COPY** und **CONVEY** (im Glossar nachlesen!!). Leider passen nicht sonderlich viele Quelltextscreens in eine Ramdisk. Pro Diskettenseite müssen Sie schon mit 2-6 Ramdisks à 20 kB rechnen.
Die Kassettenversion ist neu und noch wenig getestet. Richten Sie sich daher bitte auf mindestens 2-3 Tage ein, an denen Sie ein Diskettenlaufwerk zur Verfügung haben sollten, um die Bedienung erlernen und kopieren zu können. Teilen Sie uns bitte (möglichst schriftlich) mit, welche Probleme Sie haben. Bevor Sie völlig verzweifeln, rufen Sie mich lieber an (Claus Vogt 030 - 216 89 38), aber bitte nicht nachts um 3 Uhr...

Die Ramdisk

Bei Ausführung von **TAPEINIT** wird das deferred Wort **R/W** mit der Schreib-Leseroutine für die Ramdisks **RAMR/W** neu besetzt. Daher müssen für die Benutzung der Ramdisk keine speziellen Worte benutzt werden, denn Zugriffe auf Blöcke, die sich auf Drive 1, 2, 3 etc. befinden, werden automatisch auf die Ramdisk umgeleitet.
Einige Worte der Ramdisk sind im Vokabular **RAMDISK** .
Beim Lese-Zugriff auf einen nicht vorhandenen Block der Ramdisk wird ein leerer Block geliefert.

Die Ramdisk benutzt zur Ablage der Blöcke im Speicher ein komprimierendes Format: aufeinanderfolgende Leerzeichen werden in ein Zeichen zusammengezogen, das wie folgt berechnet wird : ($7F+AnzahlDerLeerzeichen).
In diesem Format können keine Bytes, die größer als $7F sind, abgelegt werden. Daher funktioniert die Ramdisk nur mit normalen Quelltexten.
Es kann zu Probleme führen, wenn Bytes, die größer als $80 sind, in den Ramdisk-Bereich geschrieben werden. Dann werden beim nächsten Zugriff auf den beschädigten Block zuviele Bytes an das System zurück übergeben. Dieses reagiert beim nächsten FLUSH i.a. mit der Ausgabe **no File** und beim nächsten **EMPTY-BUFFERS** mit Systemzusammenbruch. Ist man auf die Meldung **no File** gestossen, so sollte **COLD** eingeben werden; die Ramdisk bleibt dabei unverändert.
Zur Korrektur von Fehlern, muß man das Speicherformat der Ramdisk kennen. Es ist auf den Shadows der Quelltexte beschrieben. Da Fehlerkorrektur sehr mühsam ist, sollte die Ramdisk lieber oft genug gesichert werden.
Blöcke, die nicht nur normalen Text, sondern auch Graphikzeichen oder binäre Daten enthalten, müssen mit **BINARY** deklariert werden. Sie belegen dann immer 1 kByte im Speicher.

Die Ramdisk ist voll, wenn weniger als 1 kByte frei ist.

Die Kassettenschnittstelle

Das aktuelle Gerät für alle Load/Save-Operationen kann mit den folgenden Worten umgeschaltet werden :
COMMODORE **SUPERTAPE** (nur C16) **FLOPPY**

Speichern:

SAVESYSTEM sichert ultraFORTH-Bereich
SAVERAMDISK sichert Ramdisk

Laden:

LOADRAMDISK lädt Ramdisk

Ein mit **SAVESYSTEM** im **COMMODORE**-Format oder auf **FLOPPY** gesichertes System wird wie gewohnt geladen.

Ein mit **SAVESYSTEM** im **SUPERTAPE** -Format gesichertes System wird wie folgt geladen:

- Rechner aus- und wieder einschalten.
- MONITOR aufrufen.
- Mit dem Monitor-Befehl L wird Supertape geladen. Während der Meldung PRESS PLAY ON TAPE können schon mal bis zu 8 Zeichen im Voraus eingetippt werden. (z.B. um dann Kaffee trinken zu gehen.)
- Den (im Filenamen versteckten) Monitorbefehl **G####** abschicken (z.B. mit <cursor-up> <cursor-up> <return>). Es werden zwei weitere Teile des Systems geladen. Falls ein Ladefehler aufgetreten ist, wird ein Monitor-Break erzeugt. Ansonsten wird das ultraFORTH83 automatisch gestartet.

Es besteht die Möglichkeit nach Laden des Systems weitere Prozesse automatisch auszulösen. Vgl.: **AUTOLOAD** **'RESTART** **'COLD**

Änderungen seit rev. 3.5

(BLOAD (BSAVE	wurden aus dem System entfernt.
(64 (16 C)	sind hinzugekommen.
CREATE:	ist hinzugekommen.
INPUT: OUTPUT:	wurden geändert, so daß sie jetzt mit ; statt [abgeschlossen werden. Damit ist die Syntax natürlicher geworden und eine zusätzliche Fehlerprüfung möglich.
C64FKEYS	ist für die C16-Version hinzugekommen.
FREEBUFFERS	wurde korrigiert.
(DRV	wurde sichtbar gemacht.
UNNEST	wird statt bisher EXIT von ; kompiliert. Diese Änderung wurde im Zusammenhang mit dem neuen Tracer notwendig. Der automatische Dekompiler wurde ebenfalls entsprechend geändert.
LIST	kann jetzt mit jeder Taste angehalten und mit <run/stop> abgebrochen werden.
'COLD 'RESTART	wurden freigeräumt. Bisher zeigten diese deferred Worte auf Code, der systemabhängige Initialisierungen durchführte.
COLD RESTART	wurden überarbeitet.
CAPITALIZE	wurde so geändert, daß beide Commodore-Zeichensätze vom System korrekt verarbeitet werden.
L	im manuellen Dekompiler wurde in
K	umbenannt, um Verwechselungen mit dem Wort L des Editors vorzubeugen.
SAVESYSTEM	wurde vereinfacht. Es benutzt jetzt nur den seriellen Bus und nicht mehr das Betriebssystem.

Zur Benutzung mehrerer Laufwerke ergaben sich Änderungen in INDEX und im Editor.

Der Tracer wurde auf ein neues Konzept umgestellt und dadurch in der Bedienung deutlich einfacher.

Eine Kassettenversion ist zum Lieferumfang hinzugekommen.

Das System kommt jetzt in DEZIMAL hoch!

Schwer erkennbare Ursachen von System-Crashs

Oft hängen System-Crashs mit Referenzen auf nicht mehr existierende Worte zusammen. Beispiel:

```
: .blk    blk @ ?dup IF . space THEN ;
' .blk Is .status
cold
```

Das deferred Wort **.STATUS** referenziert jetzt **.BLK** . Nach **COLD** (oder **EMPTY** oder **FORGET .BLK**) ist **.BLK** gelöscht, die Referenz existiert weiterhin.

Bei beiden Beispielen arbeitet das System erstmal wie gewohnt weiter. Erst viel später (nach überschreiben des ehemaligen Speicherbereichs von **.BLK** führt der Aufruf von **.STATUS** zu undefinierten Reaktionen (i.a. System-Crashs).

Nach Belegung eines deferred Wortes **SAVE** eingeben oder aufpassen.

Was tun bei Fehlern ?

Wenn Ihr denkt, daß Ihr einen Fehler im ultraFORTH gefunden habt, dann tut bitte folgendes:

- Überprüft, ob die Ursache vielleicht in "Schwer erkennbare Ursachen von Systemcrashs" beschrieben ist.
- Fahrt Euer System nochmal ganz neu hoch, bis Ihr die minimale Konfiguration erreicht habt, bei der der Fehler noch auftritt. Schreibt auf, was Ihr ladet.
- Hängt alle Zusatzgeräte ab, die gerade nicht benötigt werden.
- Schickt eine genaue Beschreibung an die Autoren des ultraFORTH83. Ggf. über die Forth-Gesellschaft in Hamburg.
- Schreibt dazu, welchen Rechner Ihr habt, (ggf. Zusatzausrüstung) und welche Zusatzgeräte angeschlossen waren.
- Wenn die Programme, die den Fehler verursachten, länger als ein oder zwei Screens sind, kopiert sie auf eine Diskette und schickt sie mit.
- Beschreibt den Ladeprozeß der Fehler-Konfiguration so genau wie möglich (am besten mit einem Lade-Screen, der alles automatisch hochfährt).
- Schreibt dazu, wo Ihr den Fehler vermutet.

Wir tun unser bestes, aber versprechen können wir nichts.

Errata

Leider enthält ein Teil des Handbuches einige Fehler, die erst bei einer vollstandigen Überarbeitung entfernt werden können:

S.28 Statt
": Input: keyboard c64key c64key? c64decode c64expect ["
muß es
"Input: keyboard c64key c64key? c64decode c64expect ;"
heißen. Das gilt analog für die Seiten 29, 36, 89 und 90.

S.63 Die Anschrift ist veraltet.

S.81 In der Erklärung von UM/MOD ist "division overflow" durch "/0" zu ersetzen.

S.111 Die Worte (BLOAD und (BSAVE wurden entfernt.

Alle Stellen im Handbuch, wo C64 steht, sind durch C64,C16,C116 und Plus 4 zu ersetzen.

Alle Stellen im Handbuch, wo serieller Bus steht, sind durch serieller Bus (bzw.beim C16,C116 und Plus4 ggf. der parallele Floppybus) zu ersetzen.

1) Dictionarystruktur des ultraFORTH83

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

1.1) Struktur der Worte

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

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

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

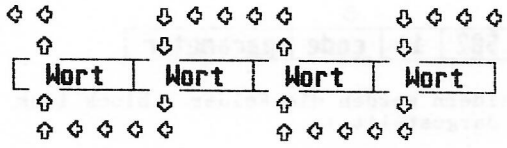
c) "count"
Die Länge des Namens dieses Wortes und drei Markierungsbits.

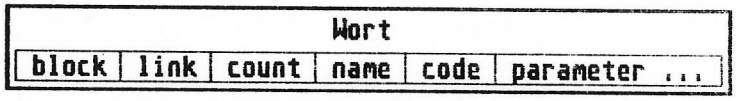
d) "name"
Der Name selbst.

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

f) "parameter"
Das Parameterfeld.

Das Dictionary sieht dann so aus :





1.1.1) Countfeld

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

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

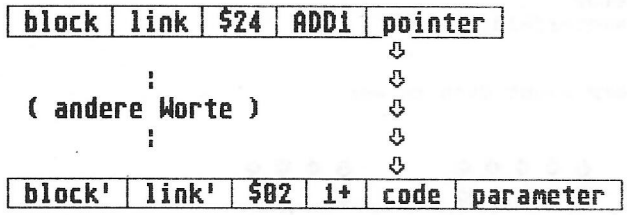
Ist das immediate-Bit gesetzt, so wird das entsprechende Wort im kompilierenden Zustand unmittelbar ausgeführt, und nicht ins Dictionary kompiliert (siehe auch IMMEDIATE im Glossary)

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

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

Beispiel: ' 1+ Alias add1

und ergibt folgende Struktur im Speicher (Dictionary) :



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

1.1.2) Name

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

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

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

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

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

1.1.3) Link

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

1.1.4) Block

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

1.1.5) Code

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

1.1.6) Parameter

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

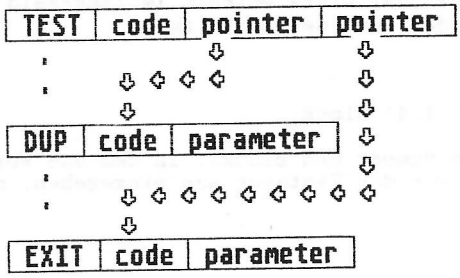
a) Typ "Constant"
Hier enthält das Paramterfeld des Wortes den Wert der Konstanten. Der dem Wort zugeordnete Code liest den Inhalt des Paramterfeldes aus und legt ihn auf den Stack.

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

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

Beispiel : : test dup ;

ergibt:



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

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

1.2) Vokabular-Struktur

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

1.2.1) Die Suchreihenfolge

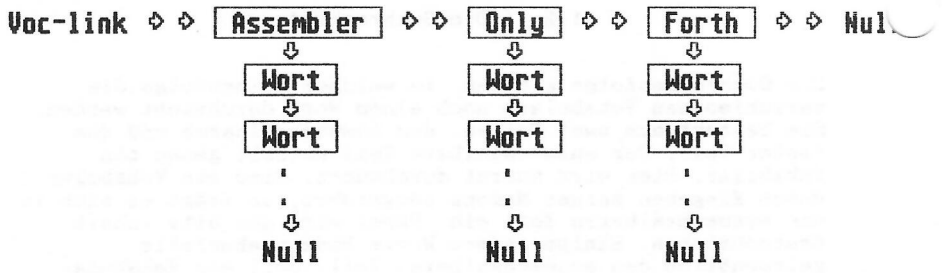
Die Suchreihenfolge gibt an, in welcher Reihenfolge die verschiedenen Vokabulare nach einem Wort durchsucht werden. Sie besteht aus zwei Teilen, dem auswechselbaren und dem festen Teil. Der auswechselbare Teil enthält genau ein Vokabular. Dies wird zuerst durchsucht. Wird ein Vokabular durch Eingeben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

Beispiele :

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

1.2.2) Struktur des Dictionaries

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



2) Die Ausführung von Forth-Worten

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

2.1) Aufbau des Adressinterpreters beim 6502

Der Adressinterpreter besteht aus den folgenden Registern :
IP W SP RP

Beim 6502 werden alle Register durch je zwei Zeropage-Speicherzellen simuliert.

a) IP ist der Instruktionszeiger (englisch : Instruction Pointer). Er zeigt auf die nächste auszuführende Instruktion. Das ist beim ultraFORTH83 die Speicherzelle, die die Kompilationsadresse des nächsten auszuführenden Wortes enthält.

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

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

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

2.2) Die Funktion des Adressinterpreters

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

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

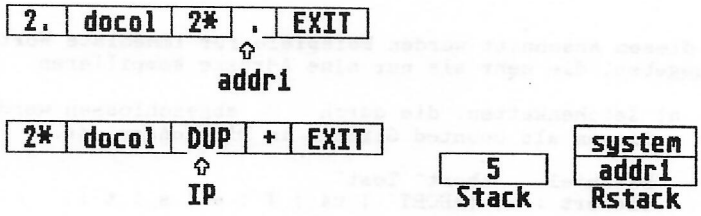
Tatsächlich ist NEXT jedoch eine Maschinencoderoutine, weil dadurch die Ausführungszeit von Forth-Worten erheblich kürzer wird. NEXT ist somit die Einsprungadresse einer Routine, die diejenige Instruktion ausführt, auf die das Register IP zeigt. Ein Wort wird ausgeführt, indem der Code, auf den die Kompilationsadresse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

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

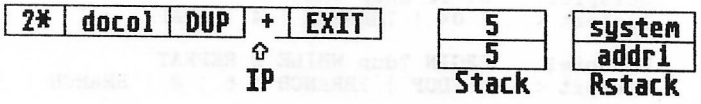
```
: 2*   dup + ;
: 2.   2* . ;
```

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

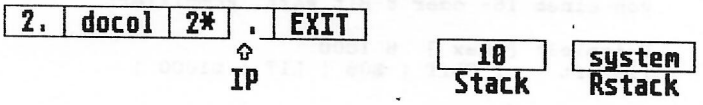
Nach der ersten Ausführung von NEXT bekommt man :

b) 

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

c) 

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

d) 

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

2.3) Verschiedene Immediate Worte

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

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

```
Beispiel:   abort" Test"
liefert :   ¦ (ABORT" ¦ 04 ¦ T ¦ e ¦ s ¦ t ¦
```

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

```
Beispiel:   0< IF swap THEN -
liefert :   ¦ 0< ¦ ?BRANCH ¦ 4 ¦ SWAP ¦ - ¦

Beispiel:   BEGIN ?dup WHILE @ REPEAT
liefert :   ¦ ?DUP ¦ ?BRANCH ¦ 8 ¦ @ ¦ BRANCH ¦ -10 ¦
```

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

d) Zahlen werden in das Wort LIT oder CLIT , gefolgt von einem 16- oder 8-Bit Wert, kompiliert.

```
Beispiel: [ hex ]  8 1000
liefert :   ¦ CLIT ¦ $08 ¦ LIT ¦ $1000 ¦  .
```

3) Die Does>-Struktur

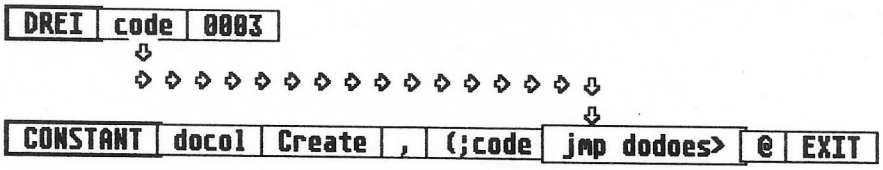
Die Struktur von Worten, die mit CREATE .. DOES> erzeugt wurden, soll anhand eines Beispiels erläutert werden.

Das Beispielprogramm lautet :

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

Der erzeugte Code sieht folgendermaßen aus:

Der jmp-Befehl tritt bei 6502-Systemen auf und wird bei anderen Rechnern sinngemäß ersetzt.



Das Wort (;CODE wurde durch DOES> erzeugt. Es setzt das Codefeld des durch CREATE erzeugten Wortes DREI und beendet dann die Ausführung von CONSTANT .
Das Codefeld von DREI zeigt anschließend auf jmp dodoes> . Wird DREI später aufgerufen , so wird der zugeordnete Code ausgeführt. Das ist in diesem Fall jmp dodoes> . dodoes> legt nun die Adresse des Parameterfeldes von DREI auf den Stack und führt die auf jmp dodoes> folgende Sequenz von Worten aus. (Man beachte die Ähnlichkeit zu :-Definitionen). Diese Sequenz besteht aus @ EXIT . Der Inhalt des Parameterfeldes von DREI wird damit auf den Stack gebracht. Der Aufruf von DREI liefert also tatsächlich 0003.

Statt des jmp dodoes> und der Sequenz von Forthworten kann sich hinter (;CODE auch ausschließlich Maschinencode befinden. Das ist der Fall, wenn wir das Wort ;CODE statt DOES> benutzt hätten. Wird diese Maschinencodesequenz später ausgeführt, so zeigt das W-Register des Adressinterpreters auf das Codefeld des Wortes, dem dieser Code zugeordnet ist. Erhöht man also das W-Register um 2 , so zeigt es auf das Parameterfeld des gerade auszuführenden Wortes.

4) Vektoren und Deferred Worte

Das ultraFORTH83 besitzt eine Reihe von Strukturen, die dazu dienen, das Verhalten des Systems zu ändern.

4.1) deferred Worte

Sie werden durch das Wort DEFER erzeugt. Im System sind bereits folgende deferred Worte vorhanden:
R/W 'COLD 'RESTART 'ABORT 'QUIT NOTFOUND .STATUS DISKERR
Um sie zu ändern, benutzt man die Phrase:
' <name> Is <namex>
Hierbei ist <namex> ein durch DEFER erzeugtes Wort und <name> der Name des Wortes, das in Zukunft bei Aufruf von <namex> ausgeführt wird.
Wird <namex> ausgeführt bevor es mit IS initialisiert wurde, so erscheint die Meldung "Crash" auf dem Bildschirm.

Anwendungsmöglichkeiten von deferred Worten

Durch Ändern von R/W kann man andere Floppies oder eine RAM-Disk betreiben. Es ist auch leicht möglich, Teile einer Disk gegen überschreiben zu schützen.
Durch Ändern von NOTFOUND kann man z.B. Worte, die nicht im Dictionary gefunden wurden, anschließend in einer Disk-Directory suchen lassen oder automatisch als Vorwärtsreferenz vermerken.
Ändert man 'COLD , so kann man die Einschaltmeldung des ultraFORTH83 unterdrücken und stattdessen ein Anwenderprogramm starten, ohne daß Eingaben von der Tastatur aus erforderlich sind ("Turnkey-Applikationen").
.STATUS schließlich wird vor Laden eines Blocks ausgeführt. Man kann sich damit anzeigen lassen, welchen Block einer längeren Sequenz das System gerade lädt und z.B. wieviel freier Speicher noch zur Verfügung steht.

4.2) >interpret

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

4.3) Variablen

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

```
ERRORHANDLER PERFORM
```

Zuweisen und Auslesen der Variablen geschieht mit @ und ! .
Der Inhalt von ERRORHANDLER wird ausgeführt, wenn das System ABORT" oder ERROR" ausführt und das von diesen Worten verbrauchte Flag wahr ist.

4.4) Vektoren

Das ultraFORTH83 benutzt die indirekten Vektoren INPUT und OUTPUT. Die Funktionsweise der sich daraus ergebenden Strukturen soll am Bespiel von INPUT verdeutlicht werden. INPUT ist eine Uservariable, die auf einen Vektor zeigt, in dem wiederum vier Kompilationsadressen abgelegt sind. Jedes der vier Inputworte KEY KEY? DECODE und EXPECT führt eine der Kompilationsadressen aus. Kompiliert wird solch ein Vektor in der folgenden Form:

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

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

Das Beispiel KEY? soll dieses Prinzip verdeutlichen:

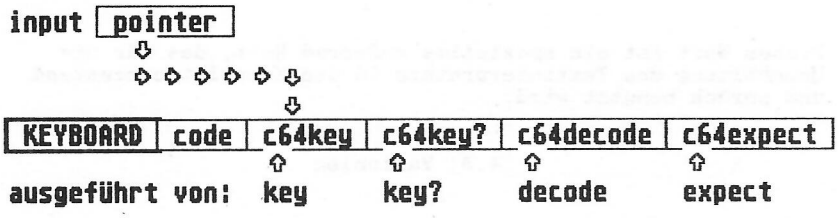
```
: key?   input @  2+  perform ;
```

Tatsächlich wurde KEY? im System ebenso wie die anderen Inputworte durch das nicht mehr sichtbare definierende Wort IN: erzeugt.
Analog verhält es sich mit OUTPUT und den Outputworten EMIT CR TYPE DEL PAGE AT und AT? . Outputvektoren werden mit OUTPUT: genauso wie die Inputvektoren erzeugt.
Mit der Input/Output-Vektorisierung kann man z.B. mit einem Schlag die Eingabe von der Tastatur auf ein Modem umschalten.

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

```
: Input: keyboard  c64key c64key? c64decode c64expect [
keyboard
```

ergibt :



Die Druckeranpassung

Alle Ausgabebefehle (EMIT, TYPE, SPACE, etc) sind im ultraFORTH83 vektorisiert, d. h. bei ihrem Aufruf wird die Codeadresse eines zugehoerigen Befehls aus einer Tabelle entnommen und ausgefuehrt. Im System enthalten ist eine Tabelle mit Namen DISPLAY , die fuer die Ausgabe auf das Bildschirm-Terminal sorgt. Dieses Verfahren bietet entscheidende Vorteile:

- Mit einer neuen Tabelle koennen alle Ausgaben auf ein anderes Geraet (z. B. einen Drucker) geleitet werden, ohne die Ausgabebefehle selbst aendern zu muessen.
- Mit einem Wort (DISPLAY , PRINTER) kann das gesamte Ausgabeverhalten geaendert werden. Gibt man z. B. ein:

```
PRINTER  1 LIST  DISPLAY
```

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

Eine neue Tabelle wird mit dem Wort OUTPUT: erzeugt.

```
: OUTPUT:   CREATE ]   DOES>   OUTPUT !  ;
```

OUTPUT: erwartet eine Liste von Ausgabeworten, die mit [abgeschlossen werden muss. Beispiel:

```
OUTPUT: >PRINTER
PEMIT PCR PTYPE PDEL PPAGE PAT PAT? [
```

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

Der folgende Quaeltext enthaelt die Anpassung fuer einen epson-Drucker RX80 am C64, und zwar sowohl mit Interface (ueber den seriellen Bus) als auch ohne. Im letzten Fall wird eine centronics-Schnittstelle am Userport erzeugt. Diese Anpassung laeuft ohne Aenderungen mit allen uns bekannten Druckern mit centronics-Schnittstelle, lediglich die Befehle fuer die Steuercodes muessen gegebenenfalls angepasst werden.

Im Loadscreen (#40) muessen die Worte (s und (u wegkommentiert werden, je nachdem, ob man ueber den seriellen Bus oder den Userport drucken will. Zusaetzlich zu den reinen Ausgaberoutinen (Screens 45 und 46) sind eine Reihe nuetzlicher Worte enthalten, mit denen die Druckersteuerung sehr komfortabel vorgenommen werden kann. Das Wort PTHRU ermoeglicht den Ausdruck von FORTH- Quelltexten in komprimierter Form (6 Screens pro Seite), DOKUMENT erlaubt die uebersichtliche Darstellung von Screen und zugehoerigem Shadow nebeneinander.

40

```
0    \ printer loadscreen          27jul85re)
1
2    Onlyforth hex
3
4    Vocabulary Print
5    Print definitions also
6
7    Create Prter 2 allot  ( Semaphor)
8    Prter unlock
9
A      : ) ; immediate
B      : (u ; immediate  \ for user-port
C      : (s  [compile] ( ; immediate
D    \ : (s ; immediate  \ for serial bus
E    \ : (u  [compile] ( ; immediate
F
10   (s  1 +load )
11
12   02 0A +thru
13
14   Onlyforth
15
16   clear
17
18
```

41

```
0    \ Buffer for the ugly SerBus  28jul85re)
1
2    100 ¦ Constant buflen
3
4    ¦ Variable Prbuf  buflen allot Prbuf off
5
6    ¦ : >buf  ( char --)
7     Prbuf count + c!  1 Prbuf +! ;
8
9    ¦ : full?  ( -- f)   Prbuf c@ buflen = ;
A
B    ¦ : .buf  ( --)
C     Prbuf count -trailing
D     4 0 busout bustype busoff Prbuf off ;
E
F    : p!  ( char --)
10    pause  >r
11    r@ 0C ( Formfeed  ) =
12    IF  r> >buf .buf exit  THEN
13    r@ 0A ( Linefeed  ) =
14    r@ 0D ( CarReturn ) = or  full? or
15    IF  .buf  THEN  r> >buf ;
16
17
18
```

40

```
setzt order auf FORTH FORTH ONLY   FORTH


fuer multitasking


Centronics-Schnittstelle ueber User-Port
(s Text bis ) wird ueberlesen
serielle Schnittstelle (wegkommentiert)
(u Text bis ) wird ueberlesen

lade den naechsten Screen nur fuer
     seriellen Bus


unbrauchbare Koepfe weg
```

41

```
Beim seriellen Bus ist die Ausgabe jedes
einzelnen Zeichens zu langsam


Buffer fuer Zeichen zum Drucker

ein Zeichen zum Buffer hinzufuegen


Buffer voll?

Buffer ausdrucken und leeren


Hauptausgaberoutine fuer seriellen Bus
Zeichen merken
ist es ein Formfeed?
  ja, Buffer ausdrucken incl. Formfeed
ist es ein Linefeed?
oder ein CR oder ist der Buffer voll?
  ja, Buffer ausdrucken, CR/LF merken
```

42

```
0    \ p! ctrl: ESC esc:            28jul85re)
1
2    (u
3    : p!  \ char --
4     0DD01 c!  0DD00 dup c@ 2dup
5     4 or swap c!  0FB and swap c!
6      BEGIN  pause  0DD0D c@ 10 and  UNTIL ;
7    )
8
9    ¦ : ctrl:  ( 8b --)   Create c,
A      does>  ( --)   c@ p! ;
B
C      07 ctrl: BEL     ¦  7F ctrl: DEL
D    ¦ 0D ctrl: CRET    ¦  1B ctrl: ESC
E      0A ctrl: LF         0C ctrl: FF
F
10   ¦ : esc:   ( 8b --)   Create c,
11     does>  ( --)   ESC c@ p! ;
12
13     30 esc: 1/8"          31 esc: 1/10"
14     32 esc: 1/6"
15     54 esc: suoff
16     4E esc: +jump         4F esc: -jump
17
18
```

43

```
0    \ printer controls             28jul85re)
1
2    ¦ : ESC2   ESC  p! p! ;
3
4      : gorlitz  ( 8b --)   BL ESC2 ;
5
6    ¦ : ESC"!"  ( 8b --)   21 ESC2 ;
7
8    ¦ Variable Modus  Modus off
9
A      : on:  ( 8b --)  Create c,
B      does>  ( --)
C      c@ Modus c@ or dup Modus c! ESC"!" ;
D
E    ¦ : off:  ( 8b --)   Create 0FF xor c,
F      does>  ( --)
10     c@ Modus c@ and dup Modus c! ESC"!" ;
11
12     10 on: +dark      10 off: -dark
13     20 on: +wide      20 off: -wide
14     40 on: +cursiv    40 off: -cursiv
15     80 on: +under     80 off: -under
16   ¦  1 on: (12cpi
17   ¦  4 on: (17cpi      5 off: 10cpi
18
```

42

```
Hauptausgaberoutine fuer Centronics
Zeichen auf Port   , Strobe-Flanke
       ausgeben
wartet bis Busy-Signal zurueckgenommen
wird

gibt Steuerzeichen an Drucker


Steuerzeichen fuer den Drucker
in hexadezimaler Darstellung
gegebenenfalls anpassen !

gibt Escape-Sequenzen an Drucker


Zeilenabstand in Zoll

Superscript und Subscript ausschalten
Perforation ueberspringen ein/aus
```

43

```
Escape + 2 Zeichen

 nur fuer Goerlitz-Interface

spezieller Epson-Steuermodus

  Kopie des Drucker-Steuer-Registers

schaltet Bit in Steuer-Register ein


schaltet Bit in Steuer-Register aus


Diese Steuercodes muessen fuer andere
Drucker mit Hilfe von ctrl:, esc: und
ESC2 umgeschrieben werden

Zeichenbreite in characters per inch
eventuell durch Elite, Pica und Compress
ersetzen
```

44

```
0    \ printer controls              28jul85re)
1
2    : 12cpi    10cpi (12cpi ;
3    : 17cpi    10cpi (17cpi ;
4    : super    0 53 ESC2 ;
5    : sub      1 53 ESC2 ;
6    : lines  ( #lines --)    43 ESC2 ;
7    : "long  ( inches --)    0 lines p! ;
8    : american    0 52 ESC2 ;
9    : german      2 52 ESC2 ;
A
B    : prinit
C    (s  Ascii x gorlitz  Ascii b gorlitz
D        Ascii e gorlitz  Ascii t gorlitz
E        Ascii z gorlitz  Ascii l gorlitz )
F    (u  0FF DD03 c!
10       0DD02 dup c@  4 or swap c! ) ;
11
12   ¦ Variable >ascii  >ascii on
13
14   : normal    >ascii on
15     Modus off  10cpi  american  suoff
16     1/6"  0C "long  CRET ;
17
18
```

45

```
0    \ Epson printer interface      08sep85re)
1
2    ¦ : c>a  ( 8b0 -- 8b1)
3     >ascii @ IF
4     dup 41 5B uwithin IF  20 or  exit THEN
5     dup C1 DB uwithin IF  7F and exit THEN
6     dup DC E0 uwithin IF 0A0 xor THEN
7     THEN ;
8
9    ¦ Variable pcol  pcol off
A    ¦ Variable prow  prow off
B
C    ¦ : pemit  c>a p!  1 pcol +! ;
D    ¦ : pcr   CRET LF  1 prow +!  0 pcol ! ;
E    ¦ : pdel   DEL  -1 pcol +! ;
F    ¦ : ppage  FF  0 prow !  0 pcol ! ;
10   ¦ : pat    ( zeile spalte -- )
11     over   prow @ < IF  ppage  THEN
12     swap prow @ - 0 ?DO pcr LOOP
13     dup  pcol < IF  CRET  pcol off  THEN
14     pcol @ - spaces ;
15   ¦ : pat?   prow @  pcol @ ;
16   ¦ : ptype  ( adr count --)  dup pcol +!
17    bounds ?DO  I c@ c>a p!  LOOP ;
18
```

44

```
gegebenenfalls aendern


Aufruf z.B.mit 66 lines
Aufruf z.B mit 11 "long
Zeichensaetze, beliebig erweieterbar


Initialisierung ...
 . fuer Goerlitz-Interface


 . fuer Centronics:  Port B auf Ausgabe
                     PA2 auf Ausgabe fuer Strobe

Flag fuer Zeichen-Umwandlung

schaltet Drucker mit Standardwerten ein
```

45

```
wandelt Commodore's Special-Ascii in
ordinaeres ASCII



Routinen zur Druckerausgabe        Befehl

ein Zeichen auf Drucker            emit
CR und LF auf Drucker              cr
ein Zeichen loeschen (?!)          del
Formfeed ausgeben                  page
Drucker auf zeile und spalte       at
positionieren, wenn noetig,
neue Seite


Position feststellen               at?
Zeichenkette ausgeben              type
```

46

```
0    \ print  pl                    28jul85re)
1
2    ¦ Output: >printer
3     pemit pcr ptype pdel ppage pat pat? [
4
5
6
7    : bemit   dup  c64emit  pemit ;
8    : bcr          c64cr    pcr   ;
9    : btype   2dup c64type  ptype ;
A    : bdel         c64del   pdel  ;
B    : bpage        c64page  ppage ;
C    : bat     2dup c64at    pat   ;
D
E    ¦ Output: >both
F     bemit bcr btype bdel bpage bat pat? [
10
11   Forth definitions
12
13   : Printer
14     normal  (u prinit )  >printer ;
15   : Both
16     normal  >both ;
17
18
```

47

```
0    \ 2scr's nscr's thru      ks  28jul85re)
1
2    Forth definitions
3
4    ¦ : 2scr's  ( blk1 blk2 --)
5     cr LF  17cpi  +wide +dark  15 spaces
6     over 3 .r  13 spaces dup 3 .r
7     -dark -wide cr  b/blk 0 DO
8      cr I c/l /  15 .r  4 spaces
9      over block I +  C/L 1- type  5 spaces
A      dup  block I +  C/L 1- -trailing type
B     C/L +LOOP  2drop  cr ;
C
D    ¦ : nscr's  ( blk1 n -- blk2)   2dup
E     bounds DO I  over I + 2scr's LOOP + ;
F
10   : pthru  ( from to --)
11    Prter lock  Output push Printer  1/8"
12    1+ over - 1+ -2 and 6 /mod
13    ?dup IF swap >r
14    0 DO 3 nscr's 2+ 1+ page LOOP  r> THEN
15    ?dup IF 1+ 2/ nscr's page THEN drop
16    Prter unlock ;
17
18
```

46

erzeugt die Ausgabetabelle >printer

Routinen fuer Drucker
und Bildschirm gleichzeitig (both)

Ausgabe erfolgt zuerst auf Bildschirm
(Routinen von DISPLAY)
dann auf Drucker
(Routinen von >PRINTER)

erzeugt die Ausgabetabelle >both

Worte sind von Forth aus zugaenglich

legt Ausgabe auf Drucker

legt Ausgabe auf Drucker und Bildschirm

47

gibt 2 Screens nebeneinander aus
Screennummer in Fettschrift und 17cpi

formatierte Ausgabe der beiden Screens

gibt die Screens so aus:

```
1   3
2   4
```

gibt die Screens von from bis to aus
Ausgabegeraet merken und Printer ein
errechnet Druckposition der einzelnen
Screens und gibt sie nach obigem Muster
aus

48

```
0     \ Printing with shadows        28jul85re)
1
2     Forth definitions
3
4     ¦ : 2scr's  ( blk1 blk2 --)
5      cr LF  17cpi  +wide +dark  15 spaces
6      dup  3 .r
7      -dark -wide cr  b/blk 0 DO
8       cr I c/l /  15 .r  4 spaces
9       dup  block I +  C/L 1- type  5 spaces
A       over block I +  C/L 1- -trailing type
B      C/L +LOOP  2drop  cr ;
C
D     ¦ : nscr's  ( blk1 n -- blk2)
E      0 DO dup [ Editor ]  shadow @   2dup
F      u> IF negate THEN
10     + over 2scr's 1+ LOOP ;
11
12    : dokument  ( from to --)
13     Prter lock  Output push  Printer
14     1/8"  1+ over - 3 /mod
15     ?dup IF swap >r
16     0 DO 3 nscr's page LOOP  r> THEN
17     ?dup IF nscr's page THEN drop
18     Prter unlock ;
```

49

```
0     \ 2scr's nscr's thru       ks  28jul85re).
1
2     Forth definitions   40 ¦ Constant C/L
3
4     ¦ : 2scr's  ( blk1 blk2 --)
5      pcr LF LF 10cpi +dark  012 spaces
6      over 3 .r  020 spaces dup 3 .r
7      cr 17cpi -dark
8      010 C/L * 0 DO cr over block I + C/L
9      6 spaces type 2 spaces
A      dup block I + C/L -trailing type
B      C/L  +LOOP  2drop cr ;
C
D     ¦ : nscr's ( blk1 n -- blk2)   under 0
E      DO 2dup dup rot + 2scr's 1+ LOOP nip ;
F
10    : 64pthru  ( from to --)
11     Prter lock  >ascii push  >ascii off
12     Output push  Printer
13     1/6" 1+ over - 1+ -2  and 6 /mod
14     ?dup IF swap >r
15     0 DO 3 nscr's 2+ 1+ page LOOP  r> THEN
16     ?dup IF 1+ 2/ nscr's page THEN drop
17     Prter unlock  ;
18
```

48

wie 2scr's (mit Shadow)

wie nscr's (mit Shadow)

screen Shadow
scr+1 Sh+1

wie pthru (mit Shadow)

49

Dasselbe nochmal fuer Standard-Forth
Screens mit 16 Zeilen zu 64 Zeichen

Siehe oben

Wie pthru fuer Standard-Screens

4A

```
0    \ pfindex                          11nov85re)
1
2    Onlyforth Print also
3
4    : pfindex  ( from to --)
5     Prter lock  Printer  0C "long
6     +jump  findex  cr page  -jump
7     Prter unlock  display  ;
8
9
A
B
C
D
E
F
10
11
12
13
14
15
16
17
18
```

4B

```
0    \ Printspool                       28jul85re)
1
2    \needs tasks  .( Tasker?!) abort
3
4    100 100 Task Printspool
5
6    : spool  ( from to --)
7     Printspool 2 pass
8
9     pthru
A     stop ;
B
C    : endspool  ( --)
D     Printspool activate
E     stop ;
F
10
11
12
13
14
15
16
17
18
```

4A

Ein schnelles Index auf den Drucker
12" Papierlaenge
Perforation ueberspringen

4B

Drucken im Untergrund

Der Tasker wird gebraucht

Der Arbeitsbereich der Task wird erzeugt

Hintergrund-Druck ein
von/bis werden an die Task gegeben
beim naechsten PAUSE fuehrt die
Task pthru aus und legt sich dann
schlafen.

Hintergrund-Druck abbrechen
die Task wird nur aktiviert,
damit sie sich sofort wieder schlafen
legt.

5) Der Heap

Eine der ungewöhnlichen und fortschrittlichen Konzepte des ultraFORTH83 besteht in der Möglichkeit, Namen von Worten zu entfernen, ohne den Rumpf zu vernichten.
Das ist insbesondere während der Kompilation nützlich, denn Namen von Worten, deren Benutzung von der Tastatur aus nicht sinnvoll wäre, tauchen am Ende der Kompilation auch nicht mehr im Dictionary auf. Man kann dem Quelltext sofort ansehen, ob ein Wort für den Gebrauch außerhalb des Programmes bestimmt ist oder nicht.
Die Namen, die entfernt wurden, verbrauchen natürlich keinen Speicherplatz mehr. Damit wird die Verwendung von mehr und längeren Namen und dadurch die Lesbarkeit gefördert.
Namen, die später eliminiert werden sollen, werden durch das Wort ¦ gekennzeichnet. Das Wort ¦ muß unmittelbar vor dem Wort stehen, das den zu eliminierenden Namen erzeugt. Der so erzeugte Name wird in einem Speicherbereich abgelegt, der Heap heißt. Der Heap kann später mit dem Wort CLEAR gelöscht werden. Dann sind natürlich auch alle Namen, die sich im Heap befanden, verschwunden.

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

ergibt :

Es werden weitere Worte definiert und dann CLEAR ausgeführt

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

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

Für C64-Benutzer ist zu beachten, daß das Zeichen ¦ mit der Tastenkombination " CBM - " erreicht wird und wie ein halber punktierter Cursor aussieht.

6) Der Multitasker

Das ultraFORTH83 besitzt einen recht einfachen, aber leistungsfähigen Multitasker. Er ermöglicht die Konstruktion von Druckerspoolern, Uhren, Zählern und anderen einfachen Tasks. Als Beispiel soll gezeigt werden, wie man einen einfachen Druckerspooler konstruiert. Dieser Spooler ist in verbesserter Form auch im Quelltext des Multitaskers enthalten, hier wird er aus didaktischen Gründen möglichst simpel gehalten.

6.1) Anwendungsbeispiel: Ein Kochrezept

Das Programm für einen Druckerspooler lautet:

```
$80 $100  Task background
: spool    background activate   1 100 pthru stop ;
multitask   spool
```

Normalerweise würde **PTHRU** den Rechner "lahmlegen", bis die Screens von 1 bis 100 ausgedruckt worden sind. Bei Aufruf von **SPOOL** ist das nicht so; der Rechner kann sofort weitere Eingaben verarbeiten. Damit alles richtig funktioniert, muß **PTHRU** allerdings einige Voraussetzungen erfüllen, die dieses Kapitel erklären will.
Das Wort **TASK** ist ein definierendes Wort, das eine Task erzeugt. Die Task besitzt übrigens Userarea, Stack, Returnstack und Dictionary unabhängig von der sog. Konsolen- oder Main-Task (siehe Bild). Im Beispiel ist $80 die Länge des reservierten Speicherbereichs für Returnstack und Userarea ("rlen" im Bild), $100 die Länge für Stack und Dictionary ("slen" im Bild), jeweils in Bytes. Der Name der Task ist in diesem Fall **BACKGROUND** . Die neue Task tut nichts, bis sie aufgeweckt wird. Das geschieht durch das Wort **SPOOL** .
MULTITASK sagt dem Rechner, daß in Zukunft womöglich noch andere Tasks außer der Konsolentask auszuführen sind. Es schaltet also den Taskwechsler ein.
Bei Ausführen von **SINGLETASK** wird der Taskwechsler abgeschaltet. Dann wird nur noch die gerade aktive Task ausgeführt.

Bei Ausführung von **SPOOL** geschieht nun folgendes:

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

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

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

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

```
1 100 newspool
```

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

6.2) Implementation

Der Unterschied dieses Multitaskers zu herkömmlichen liegt in seiner kooperativen Struktur begründet. Damit ist gemeint, daß jede Task explizit die Kontrolle über den Rechner und die Ein/Ausgabegeräte aufgeben und damit für andere Tasks verfügbar machen muß. Jede Task kann aber selbst "wählen", wann das geschieht. Es ist klar, daß das oft genug geschehen muß, damit alle Tasks ihre Aufgaben wahrnehmen können.
Die Kontrolle über den Rechner wird durch das Wort **PAUSE** aufgegeben. **PAUSE** führt den Code aus, der den gegenwärtigen Zustand der gerade aktiven Task rettet und die Kontrolle des Rechners an den Taskwechsler übergibt. Der Zustand einer Task besteht aus den Werten von Interpreterpointer (IP) , Returnstackpointer (RP) und des Stackpointers (SP).
Der Taskwechsler besteht aus einer geschlossenen Schleife (siehe Bild). Jede Task enthält einen Maschinencodesprung auf die nächste Task ("jmp XXXX" im Bild), gefolgt von der Aufweckprozedur ("jsr (wake" im Bild). Unter der Adresse, auf die der Sprungbefehl zielt, befinden sich die entsprechenden Instruktionen der nächsten Task. Ist diese Task gestoppt, so wird dort ebenfalls ein Maschinencodesprung zur nächsten Task ausgeführt. Ist die Task dagegen aktiv, so ist der Sprung durch den (nichts bewirkenden) BIT-Befehl ersetzt worden. Darauf folgt der Aufruf der Aufweckprozedur. Diese Prozedur lädt den Zustand der Task (bestehend aus SP , RP und IP) und setzt den Userpointer (UP), so daß er auf diese Task zeigt.
SINGLETASK ändert nun **PAUSE** so, daß überhaupt kein Taskwechsel stattfindet, wenn **PAUSE** aufgerufen wird. Das ist in dem Fall sinnvoll, wenn nur eine Task existiert, nämlich die Konsolentask, die beim Kaltstart des Systems "erzeugt" wurde. Dann würde **PAUSE** unnötig Zeit damit verbrauchen, einen Taskwechsel auszuführen, der sowieso wieder auf dieselbe Task führt. **STOP** entspricht **PAUSE** , jedoch mit dem Unterschied, daß die leere Anweisung durch einen Sprungbefehl ersetzt wird.

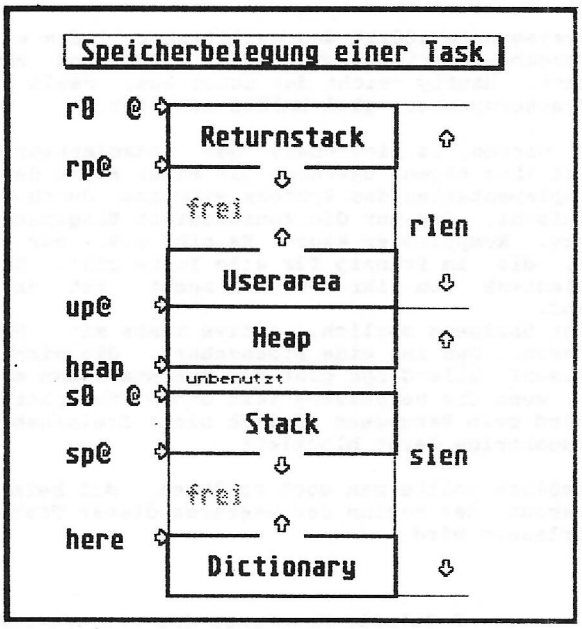
Das System unterstützt den Multitasker, indem es während vieler Ein/Ausgabeoperationen wie **KEY** , **TYPE** und **BLOCK** usw. **PAUSE** ausführt. Häufig reicht das schon aus, damit eine Task (z.B. der Druckerspooler) gleichmäßig arbeitet.

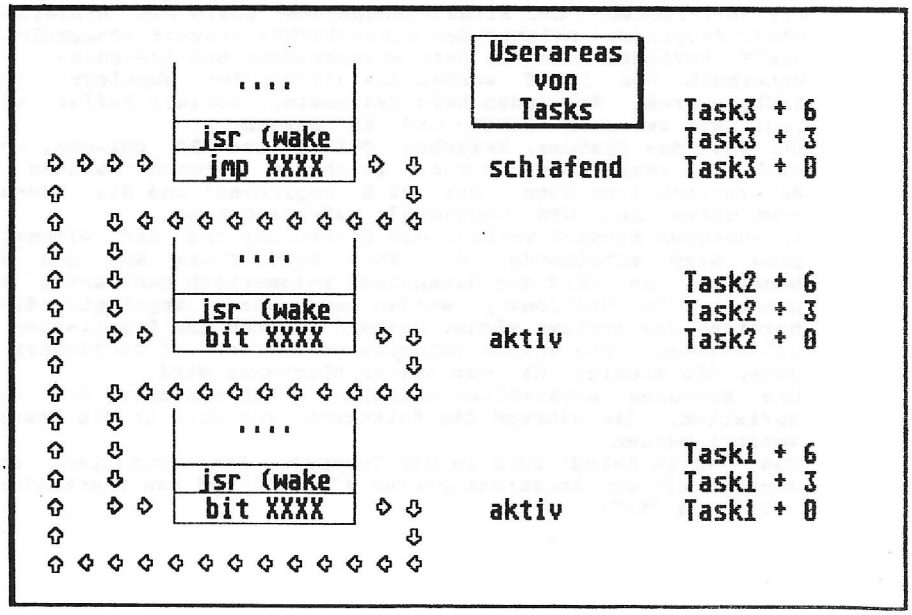
Tasks werden im Dictionary der Konsolentask erzeugt. Jede besitzt ihre eigene Userarea mit einer Kopie der Uservariablen. Die Implementation des Systems wird aber durch die Einchränkung vereinfacht, daß nur die Konsolentask Eingabetext interpretieren bzw. kompilieren kann. Es gibt z.B. nur eine Suchreihenfolge,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

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

6.2a) Die Memorymap des ultraFORTH83

Die Memorymap des ultraFORTH83 finden Sie auf S.49.
Der vom ultraFORTH83 benutzte Speicherbereich reicht von **ORIGIN** bis **LIMIT** . Unterhalb von **ORIGIN** befinden sich die Systemvariablen, der Bildschirmspeicher sowie ein einzeiliges BASIC-Programm, welches das ultraFORTH83 startet. Oberhalb von **LIMIT** befinden sich das Betriebssystemrom und I/O-Ports.
Unterhalb von **LIMIT** werden die Blockpuffer abgelegt; jeder 1 Kbyte groß. Es werden beim Systemstart soviele Puffer angelegt, wie zwischen **LIMIT** und **R0** passen.
Der Rest des Systems, zwischen **ORIGIN** und **R0** gelegen, teilt sich in zwei Bereiche auf. Im oberen Bereich wachsen der Returnstack (von oben, bei **R0 @** beginnend) und die Userarea (von unten, bei **UP@** beginnend) aufeinander zu.
Im unteren Bereich wachsen das Dictionary und der Datenstack plus Heap aufeinander zu. Wird mehr Platz auf dem Heap benötigt, so wird der Datenstack automatisch nach unten verschoben. Im Dictionary werden neue Worte abgelegt; dieser Bereich des Systems wächst deshalb während der Kompilation am schnellsten. Die Grenze zwischen Datenstack und Dictionary ist daher die einzige, die vom System überwacht wird.
Die Bootarea schließlich enthält die Anfangswerte der Uservariablen, die während des Kaltstarts von dort in die Userarea kopiert werden.
Das System belegt auch in der Zeropage Speicherzellen, unter anderem mit dem Adreßinterpreter ("NEXT") und den Stackpointern ("RP" und "SP").


Speicherbelegung einer Task
r0 @
rp@
up@
heap
s0 @
sp@
here
Returnstack
frei
rlen
Userarea
Heap
unbenutzt
Stack
slen
frei
Dictionary


Userareas von Tasks
....
jsr (wake
jmp XXXX
schlafend
Task3 + 6
Task3 + 3
Task3 + 0
....
jsr (wake
bit XXXX
aktiv
Task2 + 6
Task2 + 3
Task2 + 0
....
jsr (wake
bit XXXX
aktiv
Task1 + 6
Task1 + 3
Task1 + 0

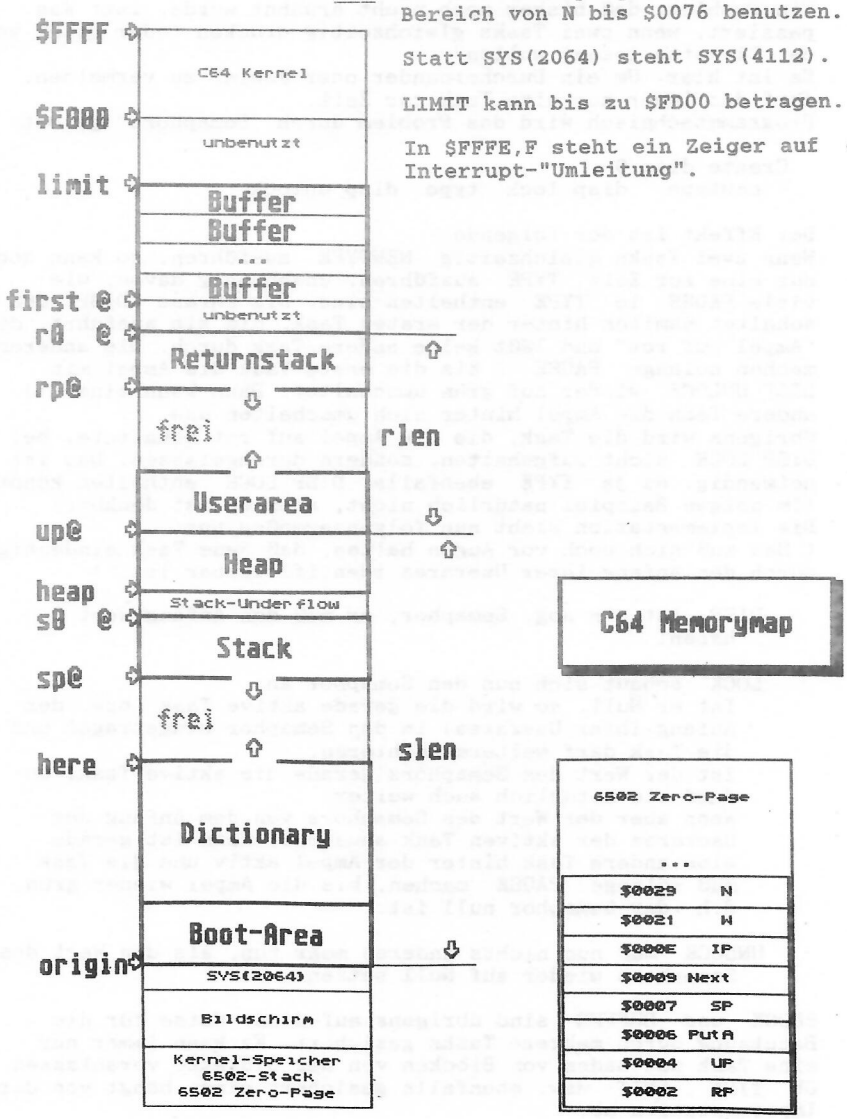
Links in der Graphik sind die Forthworte angegeben, die die entsprechenden Adressen liefern.

C16 Der Assemblerprogrammierer kann den Bereich von N bis $0076 benutzen.

Statt SYS(2064) steht SYS(4112).

LIMIT kann bis zu $FD00 betragen.

In $FFFE,F steht ein Zeiger auf die Interrupt-"Umleitung".



6.3) Semaphore und "Lock"

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

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

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

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

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

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

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

6.4) Eine Bemerkung bzgl. BLOCK und anderer Dinge

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

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

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

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

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

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

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

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

ist RICHTIG, denn PAD ist für jeden Task verschieden.

In der Version 3.8 wurde das Wort **LIST** dahingehend geändert, daß es, ähnlich wie **INDEX** , durch Drücken von Tasten angehalten und abgebrochen werden kann. Das bringt gewisse Probleme mit sich, wenn dieses Wort in einer Task (z.B. einem Druckerspooler) benutzt wird. In diesem Fall versucht sowohl die Konsolen-Task als auch die das Wort **LIST** enthaltende Task Eingaben von der Tastatur zu lesen. Tippt man einen Text ein, so werden die einzelnen Zeichen zufällig auf die beiden Tasks verteilt. Daher sollte jede Task (natürlich mit Ausnahme der Konsolen-Task) ihren Eingabevektor so definieren, daß Sie keine Eingaben erhalten kann.
Im folgenden wird gezeigt, wie ein Eingabevektor beschaffen ist, der keine Eingaben erlaubt (siehe auch S. 28):

```
: halt    ." Task gestoppt!" stop ;     \ legt die Task völlig
                                        \ still

Input: nul-Input    halt false  drop    halt ;

\ statt :           key  key?   decode  expect
```

Für **KEY** könnte man auch immer ein bestimmtes (ungefährliches) Zeichen zurückliefern. Die Entscheidung sollte aber beim Programmieren getroffen werden - wer weiß schon welches Zeichen immer und überall ungefährlich ist. Bei **EXPECT** läßt sich schwerlich eine bessere Lösung finden, da von **EXPECT** die Variable **SPAN** verändert werden soll und es unangenehm werden könnte, wenn zwei Tasks gleichzeitig an der gleichen Variablen rumpfuschen. Jede Task, die jetzt am Anfang **NUL-INPUT** ausführt, bekommt keine Eingaben und wird schlimmstenfalls stillgelegt.

Noch ein Hinweis : Eine Task sollte auf keinen Fall **ABORT"** oder ein ähnliches Wort ausführen. In diesem Fall würde sich diese Task wie die Konsolen-Task benehmen, was einen Systemabsturz zur Folge hat. Selbst wenn diese Fehlermöglichkeit abgefangen und die Task angehalten wird, bleibt noch das Problem, daß Semaphore noch auf diese Task "gelockt" sein können und damit die Benutzung bestimmter Teile des Forth blockiert bleibt!

7) Debugging - Techniken

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

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

7.1) Voraussetzungen für die Fehlersuche

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

-) suggestive und prägnante Namen für Worte

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

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

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

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

7.2) Der Tracer

Angenommen, Sie wollen folgendes Wort auf Fehler untersuchen: (bitte tippen Sie ein, alle nötigen Eingaben sind fett und alle Ausgaben des ultraFORTH83 sind unterstrichen dargestellt.)

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

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

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

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

```
teststring count .s 27 30160  ok
-trailing .s 27 30160  ok
```

und stellen zu Ihrem Erstaunen fest, daß **-TRAILING** kein einziges Leerzeichen abgeschnitten hat. (Spätestens jetzt sollten Sie den Tracer laden. Prüfen Sie, ob es das Wort **DEBUG** im **FORTH** Vokabular gibt, dann ist der Tracer schon vorhanden.
Mit dem Tracer können Sie Worte schrittweise testen. Seine Bedienung wird anhand des soeben definierten Wortes demonstriert.

Sie geben ein:

```
debug -trailing  ok
```

Es geschieht zunächst nichts. **DEBUG** hat nur den Tracer "scharf gemacht". Sobald **-TRAILING** ausgeführt werden soll, wird der Tracer aktiv.

Nun fahren Sie fort:

```
teststring count .s 27 30160  ok
-trailing
```

und erhalten folgendes Bild:

```
30071  5609 2DUP                        27 30160
```

27 30160 ist der Stackinhalt, wie er von **TESTSTRING COUNT** geliefert wurde, nämlich Adresse und Länge des Strings **TESTSTRING** [1].

(Für Fortgeschrittene : 5609 ist die Kompilationsadresse von **2DUP** , 30071 die Position von 2DUP in **-TRAILING** .)

---

[1] Der genaue Wert dieser Zahlen ist systemabhängig.

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

```
debug -trailing   teststring count .s 27 30160  ok
-trailing
30071  5609  2DUP          27 30160
30073  6819  BOUNDS        27 30160 27 30160
30075  6780  (?DO          30160 30187 27 30160
30079  5609  2DUP          27 30160
30081  5623  +             27 30160 27 30160
30083  5991  1-            30187 27 30160
30085  5066  C@            30186 27 30160
30087 13375  BL            32 27 30160
30089  6505  =             32 32 27 30160
30091  7044  ?BRANCH       FFFF 27 30160
30095  7556  LEAVE         27 30160
30103  4956  UNNEST        27 30160  ok
```

Sehen wir uns die Ausgabe nun etwas genauer an. Bei den ersten beiden Zeilen wächst der Wert ganz links immer um 2. Es ist der Inhalt des Instructionpointers (IP), der immer auf die nächste auszuführende Adresse zeigt. Der Inhalt dieser Adresse ist jeweils eine Kompilationsadresse (5609 bei **2DUP** usw.). Jede Kompilationsadresse benötigt zwei Bytes, daher muß der IP immer um 2 erhöht werden.

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

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

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

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

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

Der ultraFORTH83-Tracer gestattet es, jederzeit Befehle einzugeben, die vor dem Abarbeiten des nächsten Trace-Kommandos ausgeführt werden. Damit kann man z. B. Stack-Werte verändern oder das Tracen abbrechen. Ändern Sie probehalber beim nächsten Trace-Lauf von **-TRAILING** das **TRUE**-Flag ($FFFF) auf dem Stack vor der Ausführung von **?BRANCH** durch Eingabe von NOT auf 0 und verfolgen Sie den weiteren Trace-Lauf. Sie werden bemerken, daß die **LOOP** ein zweites Mal durchlaufen wird.

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

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

```
30071  5609 2DUP                    27 30160
30073  6819 BOUNDS                  27 30160
  6821  5364 OVER                   27 30160 27 30160
  6823  5623 +                      30160 27 30160 27 30160
  6825  5259 SWAP                   30187 30160 27 30160
  6827  4956 UNNEST                 30160 30187 27 30160
30075  6780 (?DO                    30160 30187 27 30160
    ...
```

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

Wollen Sie das Tracen eines Wortes beenden, ohne die Ausführung des Wortes abzubrechen, so benutzen Sie **UNNEST** .

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

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

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

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

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

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

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

Für die sehr neugierigen Forth'ler will ich noch Hinweise zur Funktionsweise des Tracers geben.

Der Tracer modifiziert den Adressinterpreter (siehe Kap. 2) so, daß er bei jedem ausgeführten Wort überprüft, ob während der Ausführung dieses Wortes die weiter oben gezeigte Tabelle ausgegeben werden soll oder nicht. Die einfachste Möglichkeit, einfach immer die Daten auszudrucken, ist ausgesprochen unpraktisch, denn der Programmierer wird damit förmlich erschlagen. Der "alte" Tracer (ultraFORTH83 rev. 3.5) gab die Daten nur aus, wenn die Höhe des Returnstacks einem vorgegebenen Wert entsprach. In der Regel war das nur in einem Wort der Fall, schaffte aber Probleme, wenn Worte getraced wurden, die die Höhe des Returnstacks änderten.
Der Tracer, der ab Version 3.8 enthalten ist, verwendet statt dessen den IP. Zeigt der Interpreter in ein Intervall, daß z.B. mit Hilfe des Wortes **DEBUG** gesetzt werden kann, wird die Tracer-Information ausgegeben. Dadurch ist der Tracer wesentlich bedienungsfreundlicher; die Implementation war für uns jedoch etwas knifflig...

7.3) Stacksicherheit

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

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

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

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

```
<name> stack full
```

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

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

Sie bekommen dann bei Eingabe von

```
1 0 /
```

ungefähr folgenden Ausdruck :

```
trace dump is:
    7871 ???
    7928 UM/MOD
       0 ???
    8052 M/MOD
    8065 /MOD
   12312 EXECUTE
   13076 INTERPRET
   13199 'QUIT
/ divison overflow
```

'QUIT **INTERPRET** und **EXECUTE** rühren vom Textinterpreter her. Interessant wird es bei **/MOD** . Wir wissen nämlich, daß **/MOD** von **/** aufgerufen wird. **/MOD** ruft nun wieder **M/MOD** auf, in **M/MOD** gehts weiter nach **UM/MOD** . Dieses Wort ist in Code geschrieben, bei einem Fehler ruft es aber **DIVOVL** auf. Dieses Wort ist nicht sichtbar, daher erscheinen drei Fragezeichen. Dieses Wort "verursachte" den Fehler, indem es **ABORT"** aufrief.

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

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

Es wird folgendermaßen benutzt:

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

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

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

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

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

7.4) Aufrufgeschichte

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

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

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

Beispiel :

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


RECHNE
  /
  /   RECHNE  division overflow
```

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

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

Ausgabe der Werte auf dem Stack bei Aufruf eines Wortes

Die Möglichkeit, Teile eines Tracelaufs komfortabel zu unterdrücken.

7.5) Der Dekompiler

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

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

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

Dennoch gibt es natürlich auch im ultraFORTH83 Dekompiler, und zwar sowohl einen vollautomatischen, der durch **SEE <name>** gestartet wird, als auch einen einfachen, von Hand zu bedienenden, der im folgenden näher beschrieben wird. Dieser mamuelle Dekompiler erscheint uns wichtiger; er wird benutzt, wenn der erzeugte Code von selbstgeschriebenen Worten untersucht werden soll. In solchen Fällen ist ein Dekompiler sinnvoll; der automatische versagt jedoch meistens, da er nur Strukturen dekompilieren kann, für die er programmiert wurde.
Der manuelle Dekompiler befindet sich, wie der Tracer, im Vokabular TOOLS. Folgende Worte stehen zur Verfügung.

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

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

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

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

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

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

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

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

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

```
' test >body
```

Jetzt können Sie TEST nach folgendem Muster dekompilieren

```
n 30198:   6154  CLIT  ok
c 30200:     12   . ok
n 30201:   6505  =  ok
n 30203:   7044  ?BRANCH  ok
b 30205:     27  30232 ok
n 30207:  16151  CR  ok
n 30209:   9391  ."  ok
s 30211:     20   Die Zahl ist zwolf  ! ok
n 30232:   4956  UNNEST
drop
```

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

Probieren Sie dieses Beispiel auch mit dem Tracer aus:

```
20 trace' test
```

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

8) Anpassung des ultra-/ volksFORTH an andere Rechner

Die Hardware-Voraussetzungen für die Anpassung an einen Rechner sind mindestens 24, besser noch 32 kBytes RAM.

Das ultraFORTH83 besteht aus zwei Teilen:

a) Einem Kern, der weitgehend rechnerunabhängig ist, aber prozessorspezifische Teile enthält.
Dieser Kern enthält Stackmanipulatoren, Compiler usw. . Eine Teilmenge der Worte ist in Maschinensprache geschrieben. Ihre Größe ist von Rechner zu Rechner verschieden. Ist das ultraFORTH83 bereits auf einem Rechner (z.Zt. nur 6502) implementiert, so muß dieser Teil nur marginal geändert werden.

b) Einem systemabhängigen Teil, der (weitgehend) in Forth selbst geschrieben werden kann.
Er enthält Worte für Terminal- und Massenspeichersteuerung. Anhand des C64 wird weiter unten erklärt, welche Funktionen mindestens ausgeführt werden müssen.

Vorgehensweise :

-) Falls die Teile a) und b) geändert werden müssen, so benötigen Sie einen Assembler, in Forth geschrieben, und die Minimalversion (minimaler Anteil von Maschinencode am Gesamtsystem) sowie Zugriff auf einen Rechner, auf dem das ultraFORTH läuft (C64). Schreiben Sie mir, falls Sie so etwas planen :
Bernd Pennemann / Eekboomkoppel 17 / 2000 Hamburg 62

-) Falls nur der Teil b) geändert werden muß, sollten Sie als "Eichnormal" und Kommunikationsmittel Zugriff auf einen C64 mit ultraFORTH haben. Sie können sich, wenn Sie gute Kenntnisse in Forth haben, selbst das ultraFORTH des C64 für Ihren Rechner umstricken. Für Fragen stehe ich natürlich zur Verfügung.

Einfacher, sicherer, aber langwieriger ist folgendes Verfahren :
Sie erstellen anhand der C64-Quelltexte den rechnerabhängigen Teil des ultraFORTH für Ihren Rechner. Den Quelltext schicken Sie mir per C64-Diskette (oder Modem). Ich sende Ihnen ein übersetztes und ggf. korrigiertes Forth-System auf derselben Diskette zurück. Dieses System spielen Sie dann vom C64 auf Ihren Rechner über, testen und beseitigen Fehler. Daraus gewinnen Sie einen verbesserten Quelltext, den Sie mir schicken usw. Nach einigen Iterationen dürften Sie ein sicher arbeitendes System besitzen.

Warum so kompliziert ?
Ihr ultraFORTH wird mit einem "Metacompiler" erzeugt. Das geschieht auf dem C64. Dieser Metacompiler ist ein

ziemlich kompliziertes Programm, daß man, um es korrekt bedienen zu können, am besten selbst schreibt. Außerdem verschenke ich solche Programme nicht besonders gern.

Folgende Worte sind im Quelltext für das ultraFORTH auf dem C64 an andere Rechner anzupassen:

C64KEY - liest ein Zeichen von der Tastatur ein. Bit 8 ist Null. Es wird gewartet, bis eine Taste gedrückt wurde.

C64KEY? - Liefert wahr (-1 = $FFFF) , falls eine Taste gedrückt wurde. Das Zeichen wird nicht gelesen, d.h. ein nachfolgendes C64KEY liefert den Wert der gedrückten Taste.

C64DECODE - kann unverändert übernommen werden.

C64EXPECT - kann unverändert übernommen werden.

C64EMIT - gibt ein Zeichen auf dem Bildschirm aus.

C64CR - löst den Wagenrücklauf auf dem Bildschirm aus.

C64DEL - kann anfangs durch NOOP ersetzt werden. Löscht das zuletzt gedruckte Zeichen und rückt den Cursor noch links.

C64PAGE - löscht den Bildschirm und positioniert den Cursor links oben in der Ecke.

C64AT? - liefert Zeile und Spalte der Cursorposition. Position 0 0 ist links oben in der Ecke.

c64at - Positioniert den Cursor. Argumente sind Zeile und Spalte.

C64TYPE - Kann durch folgenden Text ersetzt werden :

```
: c64type ( adr len -- )  bounds  ?DO  I c@ emit  LOOP
;
```

b/blk - bleibt unverändert.

DRV? - liefert die Nummer des aktuellen Diskettenlaufwerks. Kann durch eine Konstante ersetzt werden.

DRVINIT - Initialisiert die Laufwerke nach dem Kaltstart.
Meistens durch NOOP ersetzt.

1541R/W - liest bzw. schreibt einen 1024 Bytes langen Speicherbereich. Parameter sind die Adresse des Speicherbereiches, ein Flag, das angibt, ob gelesen oder geschrieben werden soll und die Blocknummer. Die Blocknummer ist eine Zahl, die angibt, der wievielte der 1024 Bytes langen Blöcke bearbeitet werden soll. Die ganze Diskette wird dabei in solche Blöcke eingeteilt.
Um später Files bearbeiten zu können, gibt es auch einen noch unbenutzten File-Parameter.
Dieses Wort kann, wenn die anderen Worte funktionieren, interaktiv von der Tastatur aus eingegeben werden. Dafür ist es sinnvoll, eine Diskette im Speicher zu simulieren. Den betreffenden Speicherbereich, in dem die Simulation stattfindet, kann man dann mit dem Betriebssystem des Rechners auf Diskette speichern.