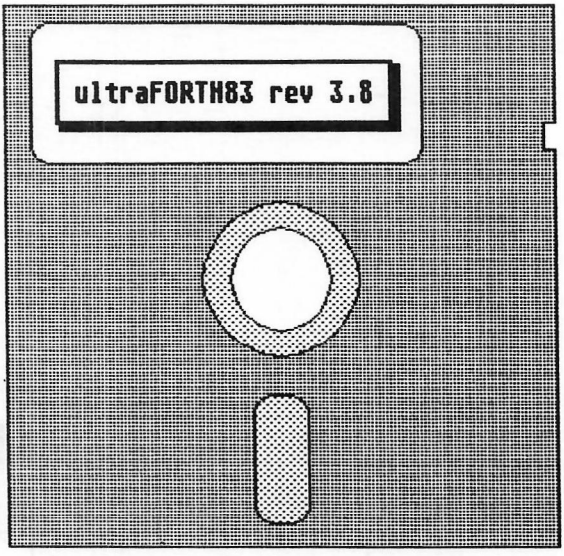


HANDBUCH

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.

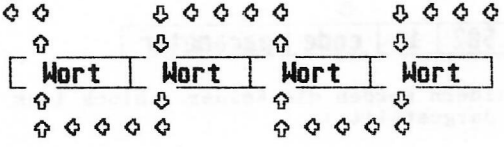
c) "count"
Die Länge des Namens dieses Wortes und drei Markierungsbits.

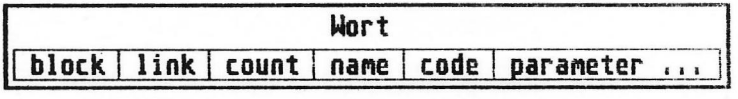
d) "name"
Der Name selbst.

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

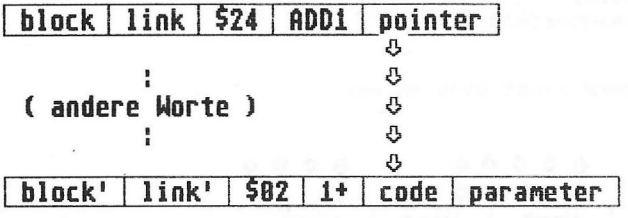
Ist das immediate-Bit gesetzt, so wird das entsprechende Wort im kompilierenden Zustand unmittelbar ausgeführt, und nicht ins Dictionary kompiliert (siehe auch IMMEDIATE im Glossary)

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 :

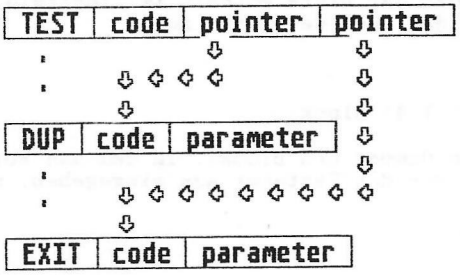
a) Typ "Constant"
Hier enthält das Paramterfeld des Wortes den Wert der Konstanten. Der dem Wort zugeordnete Code liest den Inhalt des Paramterfeldes aus und legt ihn auf den Stack.

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

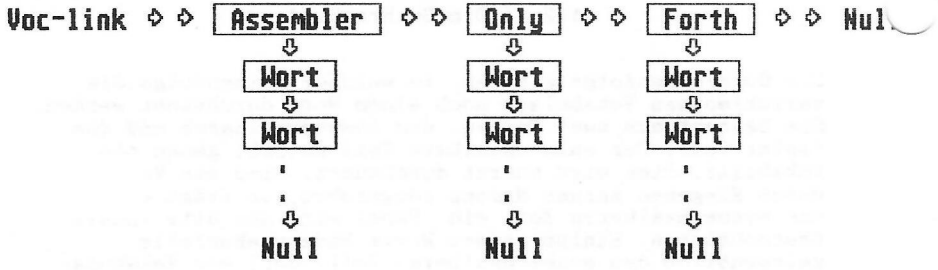
Die Suchreihenfolge gibt an, in welcher Reihenfolge die verschiedenen Vokabulare nach einem Wort durchsucht werden. Sie besteht aus zwei Teilen, dem auswechselbaren und dem festen Teil. Der auswechselbare Teil enthält genau ein Vokabular. Dies wird zuerst durchsucht. Wird ein Vokabular durch Eingeben 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:

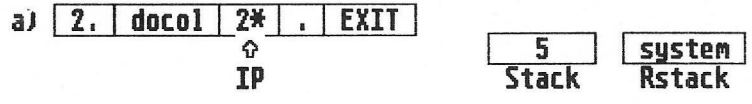
```
Variable IP
Variable W
: Next   IP @  @  W !
         2 IP +!
         W perform  ;
```

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.

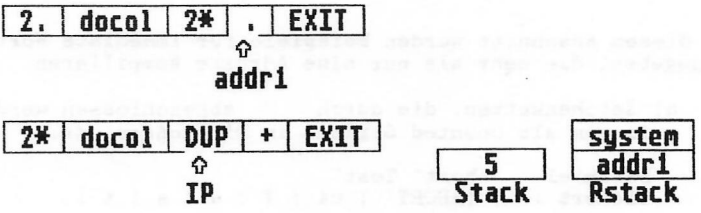
```
: 2*   dup + ;
: 2.   2* . ;
```

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



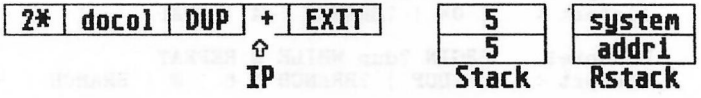
Nach der ersten Ausführung von NEXT bekommt man :

b)



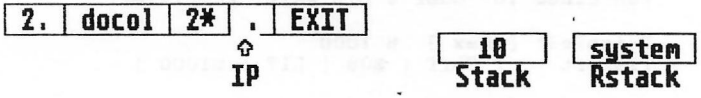
Nochmalige Ausführung von NEXT ergibt :
(DUP ist ein Wort vom Typ "Code")

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

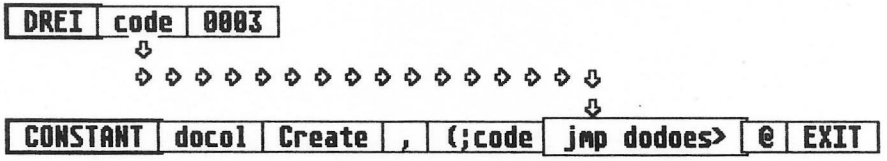
Die Struktur von Worten, die mit CREATE .. DOES> erzeugt wurden, soll anhand eines Beispiels erläutert werden.

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:

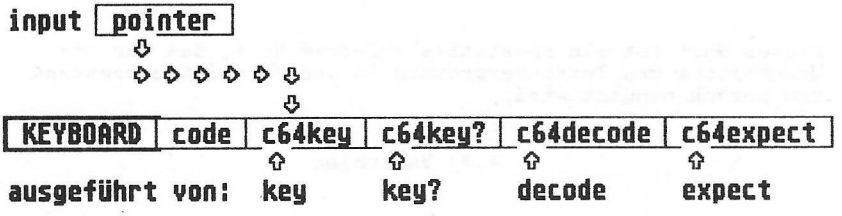
```
: key?   input @  2+  perform ;
```

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 positionieren, wenn noetig, neue Seite	at
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.

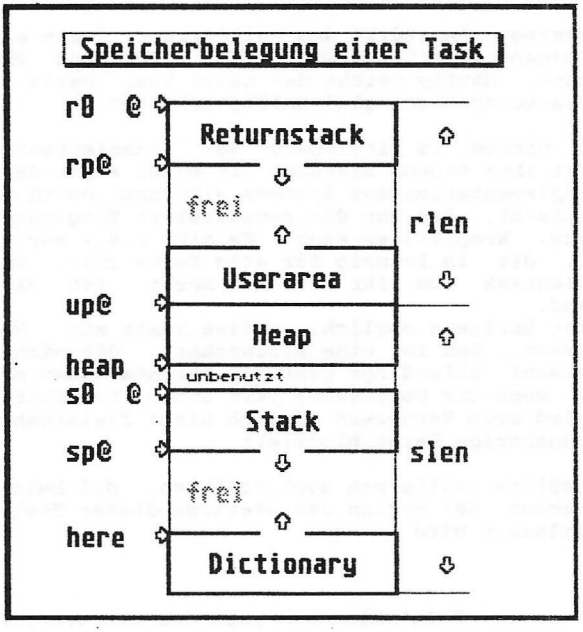
Das System unterstützt den Multitasker, indem es während vieler Ein/Ausgabeoperationen wie **KEY** , **TYPE** und **BLOCK** usw. **PAUSE** ausführt. Häufig reicht das schon aus, damit eine Task (z.B. der Druckerspooler) gleichmäßig arbeitet.

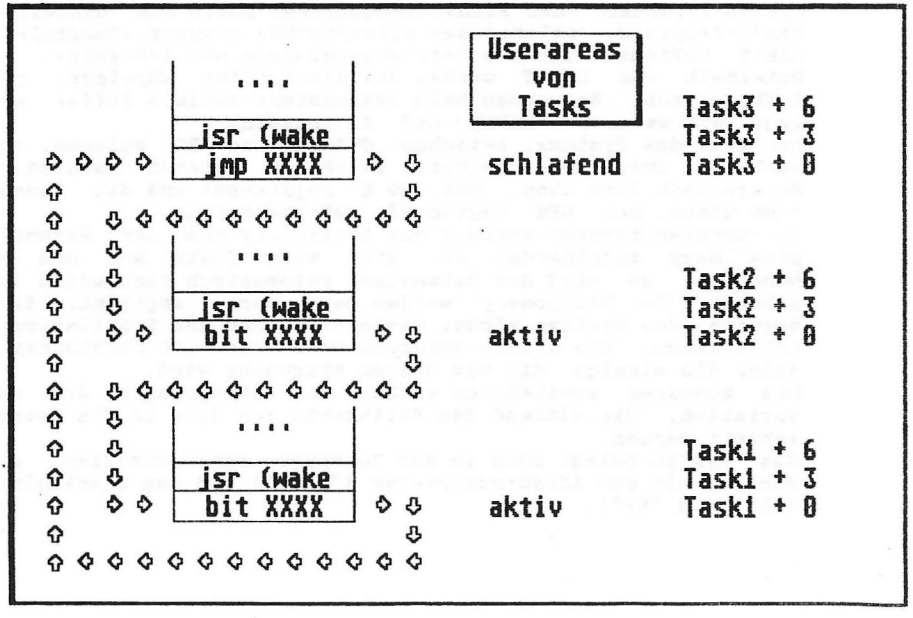
Tasks werden im Dictionary der Konsolentask erzeugt. Jede besitzt ihre eigene Userarea mit einer Kopie der Uservariablen. Die Implementation des Systems wird aber durch die Einchränkung vereinfacht, daß nur die Konsolentask Eingabetext interpretieren bzw. kompilieren kann. Es gibt z.B. nur eine Suchreihenfolge, 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 @
Returnstack
rp@
frei
rlen
Userarea
up@
Heap
heap
unbenutzt
s0 @
Stack
sp@
slen
frei
here
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

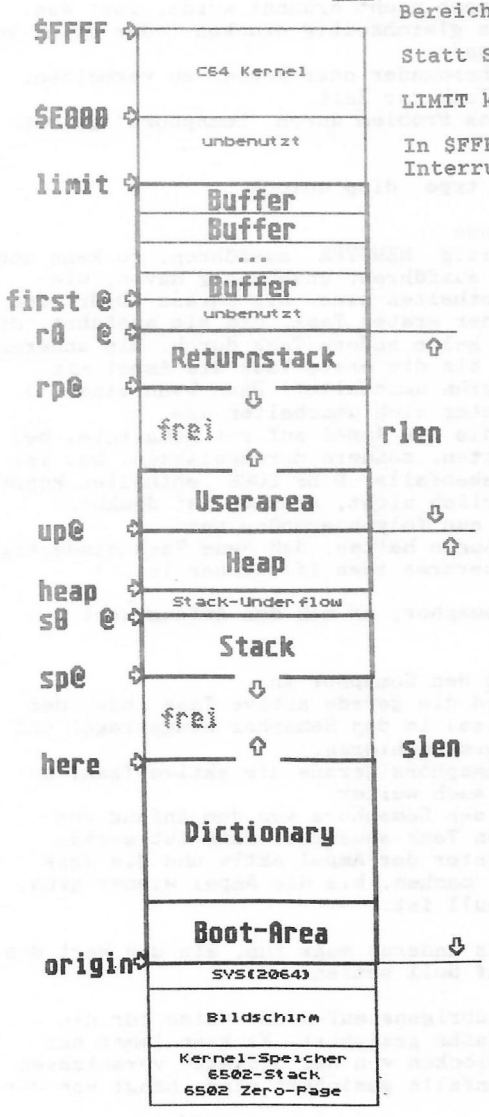
Links in der Graphik sind die Forthworte angegeben, die die entsprechenden Adressen liefern.

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

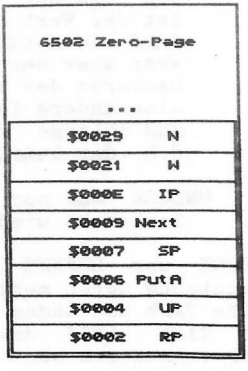
Statt SYS(2064) steht SYS(4112).

LIMIT kann bis zu $FD00 betragen.

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



C64 Memorymap



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
;
```

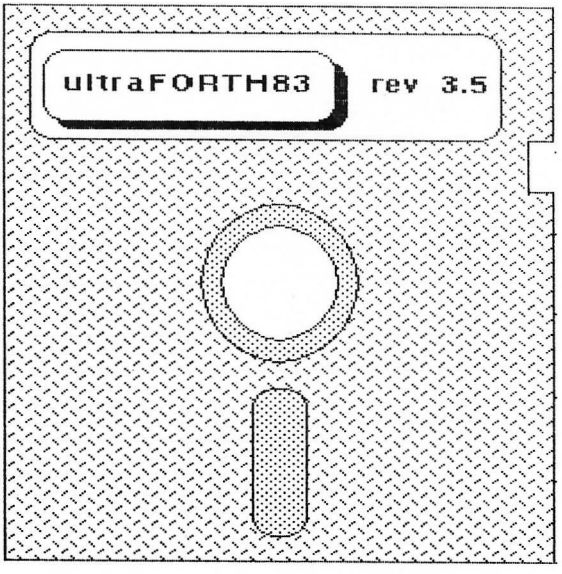
b/blk - bleibt unverändert.

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.

Glossare



Notation

Die Worte des ultraFORTH83 sind in Wortgruppen zusammengefasst. Die Worte jeder Sachgruppe sind alphabetisch sortiert. Die Wortgruppen sind nicht geordnet.

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

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

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

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

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

Worte werden ferner durch folgende Symbole gekennzeichnet:

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

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

83
: Dieses Wort wird im 83-Standard definiert und muss auf allen Standardsystemen aequivalent funktionieren.

U
: Kennzeichnet eine Uservariable. Wird aufgrund eines Missverstaendnisses nicht konsequent verwendet.

Weicht die Aussprache eines Wortes von der natuerlichen englischen Aussprache ab, so wird sie in Anfuehrungszeichen angegeben. Gelegentlich folgt auch eine deutsche Uebersetzung.

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

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

Arithmetik

* (w1 w2 -- w3) 83
"mal", engl."times"
Der Werte w1 wird mit w2 multipliziert. w3 sind die niederwertigen 16 Bits des Produktes. Ein Ueberlauf wird nicht angezeigt.

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

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

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

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

-1 (-- -1)
Oft benutzte Zahlenwerte wurden zu Konstanten gemacht. Definiert in der Form:
n CONSTANT n
Dadurch wird Speicherplatz eingespart und die Ausfuehrungszeit verkuerzt. Siehe auch CONSTANT .

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

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

0 (-- 0)
Siehe -1 .

1 (-- 1)
Siehe -1 .

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

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

2 (-- 2)
Siehe -1 .

2* (w1 -- w2)
"zwei-mal", eng.l"two-times"
w1 wird um ein Bit nach Links verschoben und das ergibt w2. In das niederwertigste Bit wird eine Null geschrieben. Die Operation 2 * wirkt genauso.

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

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

2/ (n1 -- n2) 83
"zwei-durch", engl."two-divide"
n1 wird um ein Bit nach rechts verschoben und das ergibt n2 . Das Vorzeichen wird beruecksichtigt und bleibt unveraendert. Die Operation 2 / wirkt genauso.

3 (-- 3)
Siehe -1 .

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

4 (-- 4)
Siehe -1 .

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

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

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

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

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

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

umax (u1 u2 -- u3)
"u-maximum"
u3 ist der Groessere der beiden Werte u1 und u2. Benutzt die u> Operation. Die groesste Zahl fuer n1 oder n2 ist 65535.

umin (u1 u2 -- u3)
"u-minimum"
u3 ist der Kleinere der beiden Werte u1 und u2. Benutzt die u< Operation. Die kleinste Zahl fuer n1 oder n2 ist 0.

Logik und Vergleiche

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

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

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

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

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

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

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

and (n1 n2 -- n3) 83
n1 wird mit n2 bitweise logisch UND verknuepft und das ergibt n3.

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

```
KEY ASCII A CASE? IF ... EXIT THEN
    ASCII B CASE? IF ... EXIT THEN
    ..
    DROP
```

Entspricht dem Ausdruck OVER = DUP IF NIP THEN.

false (-- 0)
Hinterlaesst Null als Zeichen fuer logisch-falsch auf dem Stack.

not (n1 -- n2) 83
Jedes Bit von n1 wird einzeln invertiert und das ergibt n2.

or (n1 n2 -- n3) 83
n1 wird mit n2 bitweise logisch ODER verknuepft und das ergibt n3.

true (-- -1)
Hinterlaesst -1 als Zeichen fuer logisch-wahr auf dem Stack.

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

u> (u1 u2 -- flag)
"u-groesser-als", engl."u-greater-than"
Wenn u1 groesser als u2 ist, ist flag wahr, sonst gilt das gleiche wie fuer U< .

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

xor (n1 n2 -- n3) 83
"x-or"
n1 wird mit n2 bitweise logisch EXCLUSIV ODER verknuepft und ergibt n3.

Speicheroperationen

! (16b adr --) 83
"store"
16b werden in den Speicher auf die Adresse adr geschrieben.
In 8-bitweise adressierten Speichern werden die zwei Bytes adr und adr+1 mit dem Wert 16b ueberschrieben. Der 6502 verlangt, das vom 16b Wert die niederwertigen 8b nach adr und die hoeerwertigen 8b nach adr+1 geschrieben werden (lowbyte-highbyte-order).

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

@ (adr -- 16b) 83
"fetch"
Von der Adresse adr wird der Wert 16b aus dem Speicher geholt. Dabei werden beim 6502 die niederwertigen 8b von adr und die hoeerwertigen 8b von adr+1 geholt und als 16b auf den Stack gelegt.

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

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

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

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

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

ctoggle (8b adr --)
"c-toggle"
Fuer jedes gesetzte Bit in 8b wird im Byte mit der Adresse adr das entsprechende Bit invertiert (dh. ein zuvor gesetztes Bit ist danach geloescht und ein zuvor geloeschtes Bit ist danach gesetzt). Fuer alle geloeschten Bits in n bleiben die entsprechenden Bits im Byte mit der Adresse adr unveraendert. Der Ausdruck DUP C@ ROT XOR SWAP C! wirkt genauso.

erase (adr u --)
Von adr an werden u Bytes des Speichers mit $00 ueberschrieben. Hat u den Wert Null, passiert nichts.

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

move (adr1 adr2 u --)
Beginnend bei adr1 werden u Bytes nach adr2 bewegt. Dabei ist es ohne Bedeutung, ob ueberlappende Speicherbereiche aufwaerts oder abwaerts kopiert werden, weil MOVE die passende Routine dazu auswaehlt. Hat u den Wert Null, passiert nichts. Siehe auch CMOVE und CMOVE> .

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

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

pad (-- adr) 83
adr ist die Startaddresse einer "scratch area". In diesen Speicherbereich koennen Daten fuer Zwischenrechnungen abgelegt werden. Wenn die naechste verfuegbare Stelle fuer das Dictionary veraendert wird, aendert sich auch die Startadresse von pad. Die vorherige Startadresse von pad geht ebenso wie die Daten dort verloren. pad erstreckt sich bis zum obersten Wert des Datenstack.

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

32-Bit-Worte

dO= (d -- flag) 83
"d-null-gleich", engl."d-zero-equals"
Wenn d gleich Null ist, wird flag wahr.

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

d< (d1 d2 -- flag) 83
"d-kleiner-als", engl."d-less-than"
Wenn d1 kleiner als d2 ist, wird flag wahr. Benutzt die < Operation, jedoch fuer 32bit Vergleiche.

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

dnegate

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

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

m* (n1 n2 -- d)
"m-mal", engl."m-times"
Der Wert n1 wird mit n2 multipliziert. d ist das doppeltgenaue vorzeichenrichtige Produkt.

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

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

um* (u1 u2 -- ud) 83
"u-m-mal", engl."u-m-times"
Die Werte u1 und u2 werden multipliziert und als doppeltgenauer Wert d abgelegt. UM* ist die anderen multiplizierenden Worten zugrundeliegende Routine. Die Zahlen u sind vorzeichenlose Zahlen.

um/mod (ud u1 -- u2 u3) 83

"u-m-durch-mod", engl."u-m-divide-mod"

u2 ist der Rest und u3 der Quotient aus der Division von ud durch den Divisor u1. Die Zahlen u sind vorzeichenlose Zahlen. Eine Fehlerbedingung besteht, wenn der Divisor Null ist oder der Quotient ausserhalb des Intervalls (0 .. 65535) liegt. UM/MOD ist die allen anderen dividierenden Worten zugrundeliegende Routine. Die Fehlermeldung "division overflow" wird ausgegeben, wenn der Divisor Null ist.

Stack

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

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

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

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

2drop (32b --) 83
"two-drop"
Der Wert 32b wird fallengelassen, dh. die beiden obersten Werte werden vom Stack entfernt.

2swap (32b1 32b2 -- 32b2 32b1) 83
"zwei-swap", engl."two-swap"
Die beiden obersten 32b Werte werden vertauscht.

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

clearstack (-- empty)
Loescht den Datastack, indem die Stack-Startadresse aus der Uservariablen S0 in den Datastackzeiger (stackpointer) geschrieben wird.

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

drop (16b --) 83
Der Wert 16b wird fallengelassen, dh. der oberste Wert wird vom Stack entfernt. Siehe auch 2DROP.

dup (16b -- 16b 16b) 83
Der Wert 16b wird verdoppelt. Siehe auch 2DUP.

nip (16b1 16b2 -- 16b2)
Der zweite Wert wird vom Stack entfernt. 16b2 ist der oberste und 16b1 der zweite Wert auf dem Stack.

over (16b1 16b2 -- 16b1 16b2 16b1) 83
Der Wert 16b1 wird ueber 16b2 herueber kopiert.

pick (16bn..16b0 +n -- 16bn..16b0 16bn) 83
Der +nte Wert im Stack wird oben auf den Stack kopiert. Dabei wird +n selbst nicht mitgezaehlt. 0 PICK wirkt wie DUP. 1 PICK wirkt wie OVER.

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

rot (16b1 16b2 16b3 -- 16b2 16b3 16b1) 83
Die drei obersten 16b Werte werden rotiert, sodass der unterste zum obersten wird.

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

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

sp! (adr --)
"s-p-store"
Setzt den Datastackzeiger (engl."stack pointer") auf die Adresse adr. Der oberste Wert im Datastack ist nun der, welcher in der Speicherstelle bei adr steht.

sp@ (-- adr)
"s-p-fetch"
Holt die Adresse adr aus dem Datastackzeiger. Der oberste Wert im Stack stand in der Speicherstelle bei adr, bevor SP@ ausgefuehrt wurde. Der Ausdruck SP@ @ wirkt wie DUP.

under (16b1 16b2 -- 16b2 16b1 16b2)
Eine Kopie des obersten Wertes auf dem Stack wird unter den zweiten Wert eingefuegt. Der Ausdruck SWAP OVER wirkt genauso.

Returnstack

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

push (adr --)
Der Inhalt aus der Adresse adr wird auf dem Returnstack bis zum naechsten EXIT verwahrt und sodann nach adr zurueck geschrieben. Dies ermoeglicht die lokale Verwendung von Variablen innerhalb einer :-Definition. Wird zB. benutzt in der Form:
: WORT ... BASE PUSH HEX ... ;
Hier wird innerhalb von WORT in der Zahlenbasis HEX gearbeitet, zB. um ein Speicherbereich auszugeben (Hex-Dump). Das Wort ; fuehrt EXIT aus. Nachdem WORT ausgefuehrt worden ist, besteht die gleiche Zahlenbasis wie vorher.

r> (-- 16b) C,83
"r-von", engl."r-from"
Der Wert 16b wird von dem Returnstack geholt. Siehe auch >R .

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

r0 (-- adr)
adr ist die Adresse einer Uservariablen, in der die Startadresse des Returnstack steht.

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

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

rdrop (--) C
"r-drop"
Der Wert 16b wird vom Returnstack fallengelassen, dh. der oberste Wert wird vom Returnstack entfernt. Der Datenstack wird nicht veraendert.

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

Strings

" (-- adr) C,I
(text (--) compiling
"string"
Liest den Text bis zum naechsten " und legt ihn als Counted String im Dictionary ab. Kann nur bei der Kompilation verwendet werden. Zur Laufzeit wird Startadresse des Counted String auf den Stack gelegt. Benutzt in der Form:
: <name> ... " <text>" ... ;

(+d1 -- +d2) 83
(1), engl."sharp";
Der Rest von +d1 geteilt durch den Wert in BASE wird in ein ASCII Zeichen umgewandelt und dem Ausgabestring in Richtung absteigender Addressen hinzugefuegt. +d2 ist der Quotient und verbleibt auf dem Stack zur weiteren Bearbeitung. Ueblicherweise benutzt zwischen <# und #> .
((1) Bisher gibt es keine verbindliche deutsche Aussprache. Das Zeichen wird in der Notenschrift verwendet und dort in engl."sharp" und in deutsch "kreutz" ausgesprochen. # wird auch oft statt Nr. verwendet; zB. SCR#)

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

#s (+d -- 0 0) 83
"sharp-s"
+d wird umgewandelt bis der Quotient zu Null geworden ist. Siehe auch # . Dabei wird jedes Zwischenergebnis in ein Ascii-Zeichen umgewandelt und dem String fuer die strukturierte Zahlenausgabe angefuegt. Wenn die Zahl von vorn herein den Wert Null hatte, wird eine einzelne Null in den String gegeben. Wird ueblicherweise benutzt zwischen <# und #> .

/string (adr1 n1 n2 -- adr2 n3)
n2 ist ein Index, welcher in den String weist, der im Speicher bei der Adresse adr1 beginnt. Der String hat die Laenge n1. n3 ist gleich n1-minus-n2, wenn n2 kleiner als n1 ist, und ad2 ist dann adr1-plus-n2. n3 ist gleich Null, wenn n2 groesser oder gleich n1 ist, und adr2 ist dann adr1-plus-n1. Der Vergleich benutzt die Operation U< . Wird zB. benutzt, um die vorderen n Zeichen eines String abzuschneiden.

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

accumulate (+d1 adr char -- +d2 adr)
Dient der Umwandlung von Ziffern in Zahlen. Multipliziert die Zahl +d1 mit BASE, um sie eine Stelle in der aktuellen Zahlenbasis nach links zu ruecken, und addiert den Zahlenwert von char dazu. char stellt eine Ziffer dar. adr wird nicht veraendert. Wird zB. in CONVERT benutzt. +n muss eine in der Zahlenbasis BASE gueltige Ziffer darstellen.

capital (char1 -- char2)
Die Zeichen im Bereich von a bis z werden in die Grossbuchstaben A bis Z umgewandelt. Andere Zeichen werden nicht veraendert.

capitalize (adr -- adr)
Wandelt alle Klein-Buchstaben im Counted String bei der Adresse adr in Gross-Buchstaben um. adr wird nicht veraendert. Siehe auch CAPITAL.

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

digit? (char -- digit true)
oder (char -- false)
Prueft mit BASE ob das Zeichen char eine gueltige Ziffer ist. Ist diese wahr, wird der Zahlenwert der Ziffer und TRUE auf dem Stack hinterlegt. Ist char keine gueltige Ziffer, wird FALSE hinterlegt.

hold (char --) 83
Das Zeichen char wird in den bildhaften Ausgabestring eingefuegt. Ueblicherweise benutzt zwischen <# und #> .

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

number (adr -- d)
Wandelt den Counted String mit der Adresse adr in die Zahl d um. Die Umwandlung erfolgt entsprechend der Zahlenbasis in BASE. Eine Fehlerbedingung besteht, wenn die Ziffern des String nicht in eine Zahl verwandelt werden koennen.

number? (adr -- d Ø>)
oder (adr -- n Ø<)
oder (adr -- adr false)

Wandelt den Counted String mit der Adresse adr in die Zahl n um. Die Umwandlung erfolgt entsprechend der Zahlenbasis in BASE oder wird vom ersten Zeichen im String bestimmt. Enthaelt der String zwischen den Ziffern auch Ascii-Zeichen fuer Komma oder Punkt, so wird er als doppelt-genaue Zahl d interpretiert. Sonst wird der String in eine einfach-genaue Zahl n umgewandelt. Wenn die Ziffern des String nicht in eine Zahl verwandelt werden koennen, bleibt die Adresse des String erhalten und der Wert fuer logisch-falsch wird auf den Stack gelegt. Die Zeichen, die zur Bestimmung der Zahlenbasis dem Ziffernstring vorangestellt werden koennen, sind:

% (Basis 2 "binaer")
& (Basis 10 "decimal")
$ (Basis 16 "hexadecimal")
h (Basis 16 "hexadecimal")

Der Wert in BASE wird dadurch nicht veraendert.

scan (adr1 n1 char -- adr2 n2)

Der String mit der Laenge n1, der im Speicher ab Adresse adr1 steht, wird nach dem Zeichen char durchsucht. adr2 ist die Adresse, bei der das Zeichen char gefunden wurde. n2 ist die Laenge des verbleibenden String. Wird char nicht gefunden, ist adr2 die Adresse des ersten Zeichen hinter dem String und n2 ist Null.

sign (n --) 83

Wenn n negativ ist, wird ein Minuszeichen in den bildhaften Ausgabestring eingefuegt. Wird ueblicherweise benutzt zwischen <# und #> .

skip (adr1 n1 char -- adr2 n2)

Der String mit der Laenge n1, der im Speicher ab Adresse adr1 steht, wird nach dem ersten Zeichen, das verschieden von char ist, durchsucht. adr2 ist die Adresse dieses Zeichens. n2 ist die Laenge des verbleibenden String. Besteht der gesammte String aus den Zeichen char, ist adr2 die Adresse des ersten Zeichen hinter dem String und n2 ist Null.

Datentypen

(-- sys) 83 "colon"
ein definierendes Wort, das in der Form:
: <name> ... ;
benutzt wird. Es erzeugt die Wortdefinition fuer <name> im Kompilations- Vokabular und schaltet den Kompiler an. Das erste Vokabular in der Suchreihenfolge wird durch das Kompilations- Vokabular ersetzt. Das Kompilations- Vokabular wird nicht geaendert. Der Quelltext wird anschliessend kompiliert. <name> wird eine "colon-definition" oder eine ":-Definition" genannt. Die neue Wortdefinition fuer <name> kann solange nicht im Dictionary gefunden werden, bis das zugehoerige ; oder ;CODE erfolgreich ausgefuehrt wurde.

Eine Fehlerbedingung liegt vor, wenn ein Wort nicht gefunden und nicht in eine Zahl gewandelt werden kann oder wenn, waehrend der Kompilation vom Massenspeicher, der Quelltext erschoepft ist, bevor ; oder ;CODE erreicht werden. sys wird vom zugehoerigen ; abgebaut.

(--) 83 I C "semi-colon"
(sys --) compiling
beendet die Kompilation einer :-Definition; macht den Namen <name> dieser :-Definition im Dictionary auffindbar, schaltet den Interpreter ein und kompiliert ein EXIT. Die Stackparameter sys, die von : hinterlassen wurden, werden geprueft und abgebaut. Siehe : und EXIT .

Alias (cfa --)
ein definierendes Wort, das typisch in der Form:
' <namex> Alias <name>
benutzt wird. ALIAS erzeugt einen Kopf fuer <name> im Dictionary. Wird <name> aufgerufen, so verhaelt es sich wie <namex>. Insbesondere wird beim Kompilieren nicht <namex>, sondern <name> ins Dictionary eingetragen. Im Unterschied zu
: <name> <namex> ;
ist es mit ALIAS auch moeglich, Worte, die den Returnstack beeinflussen (vergleiche >R oder R>), mit anderem Namen zu definieren. Ausser dem neuen Kopf fuer <name> wird kein zusaetzlicher Speicherplatz verbraucht.

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

Defer (--)
ein definierendes Wort, das in der Form:
Defer <name>
benutzt wird. Erzeugt den Kopf fuer ein neues Wort <name> im Dictionary, haelt 2 Bytes in dessen Parameterfeld frei und speichert dort die Kompilationsadresse einer Fehlerroutine. Wird <name> nun ausgefuehrt, so wird die Ausfuehrung mit einer Fehlermeldung abgebrochen. Man kann dem Wort <name> zu jedem Zeitpunkt eine andere Funktion zuweisen, siehe IS . <name> laesst sich jedoch schon kompilieren, ohne dass es eine sinnvolle Aktion ausfuehrt. Damit ist die Kompilation erst spaeter definierter Worte indirekt moeglich. Andererseits ist die Veraenderung des Verhaltens von <name> fuer spezielle Zwecke moeglich.

Deferred Worte im System sind:
R/W , 'COLD , 'RESTART , 'ABORT , 'QUIT ,
NOTFOUND , .STATUS und DISKERR .
Ein spezielles Deferred Wort ist
>INTERPRET.

Input: (--) "input-colon"
ein definierendes Wort, benutzt in der Form:
Input: <name>
newKEY newKEY? newDECODE newEXPECT [
INPUT: erzeugt einen Kopf fuer <name> im Dictionary und kompiliert einen Vektor von Zeigern auf Worte, die fuer die Eingabe von Zeichen zustaendig sind. Wird <name> ausgefuehrt, so wird ein Zeiger auf das Parameterfeld von <name> in die Uservariable INPUT geschrieben. Alle Eingaben werden jetzt ueber die neuen Eingabeworte abgewickelt. Die Reihenfolge der Worte nach INPUT: <name> bis zur [muss eingehalten werden.

Im System ist das mit INPUT: definierte Wort KEYBOARD enthalten, wenn der Editor geladen ist, stehen ausserdem EDIBOARD und DIGITS zur Verfuegung.

Is (cfa --)
ein Wort, mit dem das Verhalten eines Deferred Wortes veraendert werden kann. IS wird in der Form:
' <namex> Is <name>
benutzt. Wenn <name> kein Deferred Wort ist, so wird eine Fehlerbehandlung eingeleitet, sonst wird <name> die Ausfuehrung von <namex> zugewiesen. Siehe DEFER.

Output: (--) "output-colon"
ein definierendes Wort, benutzt in der Form:
Output: <name>
newEMIT newCR newTYPE newDEL newPAGE newAT
newAT? [
OUTPUT: erzeugt einen Kopf fuer <name> im Dictionary und kompiliert einen Vektor aus Zeigern auf Worte, die fuer die Ausgabe von Zeichen verantwortlich sind. Wird <name> ausgefuehrt, so wird ein Zeiger auf das Parameterfeld von <name> in die Uservariable OUTPUT geschrieben. Alle Ausgaben werden jetzt ueber die neuen Ausgabeworte abgewickelt. Die Reihenfolge der Worte nach OUTPUT: <name> bis zur [muss eingehalten werden.

Im System ist das mit OUTPUT: definierte Wort DISPLAY enthalten.

User (--) 83
ein definierendes Wort, benutzt in der Form:
User <name>
USER erzeugt einen Kopf fuer <name> und haelt 2 Byte in der Userarea frei. Siehe UALLOT . Diese 2 Byte werden fuer den Inhalt der USERvariablen benutzt und werden nicht initialisiert. Im Parameterfeld der USERvariablen im Dictionary wird nur ein Byte- Offset zum Beginn der Userarea abgelegt. Wird <name> spaeter ausgefuehrt, so wird die Adresse des Wertes der USERvariablen in der Userarea auf den Stack gegeben.

Variable (--) 83
ein definierendes Wort, benutzt in der Form:
Variable <name>
VARIABLE erzeugt einen Kopf fuer <name> und haelt 2 Byte in seinem Parameterfeld frei. Siehe ALLOT. Dies Parameterfeld wird fuer den Inhalt der VARIABLEn benutzt und wird nicht initialisiert. Wird <name> spaeter ausgefuehrt, so wird die Adresse des Parameterfeldes auf den Stack gegeben.

Vocabulary (--) 83
ein definierendes Wort, das in der Form:
Vocabulary <name>
benutzt wird. VOCABULARY erzeugt einen Kopf fuer <name>, das den Anfang einer neuen Liste von Worten bildet. Spaetere Ausfuehrung von <name> ersetzt das erste Vokabular in der Suchreihenfolge durch <name>. Wird <name> das Kompilations Vokabular, so werden neue Definitionen in die Liste von <name> gelinkt. Vergleiche DEFINITIONS .

Dictionary - Worte

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Vokabular - Worte

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

Assembler (--)
Ein Vokabular, das Prozessor-spezifische Worte enthaelt, die fuer Code-Definitionen benoetigt werden.

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

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

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

Forth (--) 83
Das urspruengliche Vokabular.

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

Only (--)
Loescht die Suchreihenfolge vollstaendig und ersetzt sie durch das Vokabular ONLY im festen und auswechselbaren Teil der Suchreihenfolge. ONLY enthaelt nur wenige Worte, die fuer die Erzeugung einer Suchreihenfolge benoetigt werden.

Onlyforth (--)
entspricht der haeufig benoetigten Sequenz
ONLY FORTH ALSO DEFINITIONS.

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

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

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

voc-link (-- addr)
Eine Uservariable. Sie enthaelt den Anfang einer Liste mit allen Vokabularen. Diese Liste wird u.a. fuer FORGET benoetigt.

vp (-- addr) "v-p"
Eine Variable, die das Ende der Suchreihenfolge markiert. Sie enthaelt ausserdem Informationen ueber die Laenge der Suchreihenfolge.

Heap - Worte

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

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

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

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

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

Kontrollstrukturen

+LOOP (n --) 83,I,C "plus-loop"
(sys --) compiling
n wird zum Loopindex addiert. Falls durch die Addition die Grenze zwischen limit-1 und limit ueberschritten wurde, so wird die Schleife beendet und die Loop-Parameter werden entfernt. Wurde die Schleife nicht beendet, so wird sie hinter dem korrespondierenden DO bzw. ?DO fortgesetzt.

?DO (w1 w2 --) 83,I,C "question-do"
(-- sys) compiling
Wird in der folgenden Art benutzt:
?DO ... LOOP bzw. ?DO ... +LOOP
Beginnt eine Schleife. Der Schleifenindex beginnt mit w2, limit ist w1. Details ueber die Beendigung von Schleifen: s. +LOOP. Ist w2=w1 , so wird der Schleifenrumpf ueberhaupt nicht durchlaufen.

?exit (flag --) "question-exit"
Fuehrt EXIT aus, falls das flag wahr ist. Ist das flag falsch, so geschieht nichts.

BEGIN (--) 83,I,C
(sys ---) compiling
Wird in der folgenden Art benutzt:
BEGIN (...flag WHILE) ... flag UNTIL
oder: BEGIN (...flag WHILE) ... REPEAT
BEGIN markiert den Anfang einer Schleife. Der ()-Ausdruck ist optional und kann beliebig oft auftreten. Die Schleife wird wiederholt, bis das flag bei UNTIL wahr oder oder das flag bei WHILE falsch ist. REPEAT setzt die Schleife immer fort.

bounds (start count -- limit start)
Dient dazu, ein Intervall, das durch Anfangswert start und Laenge count gegeben ist, in ein Intervall umzurechnen, das durch Anfangswert start und Endwert+1 limit beschrieben wird.

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

DO (w1 w2 --) 83,I,C
(sys --) compiling
Entspricht ?DO , jedoch wird der Schleifenrumpf mindestens einmal durchlaufen. Ist w1=w2 , so wird der Schleiferumpf 65536-mal durchlaufen.

ELSE (--) 83,I,C
(sys1 -- sys2)compiling
Wird in der folgenden Art benutzt:
flag IF ... ELSE ... THEN
ELSE wird unmittelbar nach dem Wahr-Teil , der auf IF folgt, ausgefuehrt. ELSE setzt die Ausfuehrung unmittelbar hinter THEN fort.

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

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

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

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

LEAVE (--) 83,C
Setzt die Ausfuehrung des Programmes hinter dem naechsten LOOP oder +LOOP fort, wobei die zugehoehrige Schleife beendet wird. Mehr als ein LEAVE pro Schleife ist moeglich, ferner kann LEAVE zwischen anderen Kontrollstrukturen auftreten. Der Forth83-Standard schreibt abweichend vom ultraFORTH vor, dass LEAVE ein immediate Wort ist.

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

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

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

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

UNTIL (flag --) 83,I,C
(sys --) compiling
Wird in der folgenden Art benutzt:
BEGIN (... flag WHILE) ... flag UNTIL
Markiert das Ende einer Schleife, deren Abbruch durch flag herbeigefuehrt wird. Ist das flag vor UNTIL wahr, so wird die Schleife beendet, ist es falsch, so wird die Schleife unmittelbar hinter BEGIN fortgesetzt.

WHILE (flag --) 83,I,C
(sys1 -- sys2)compiling
Wird in der folgenden Art benutzt:
BEGIN .. flag WHILE .. REPEAT
oder: BEGIN .. flag WHILE .. flag UNTIL
Ist das flag vor WHILE wahr, so wird die Ausfuehrung der Schleife bis UNTIL oder REPEAT fortgesetzt, ist es falsch, so wird die Schleife beendet und das Programm hinter UNTIL bzw. REPEAT fortgesetzt. Es koennen mehre WHILE in einer Schleife verwendet werden.

Compiler - Worte

," (--) "comma-quote"
Speichert einen counted String im Dictionary ab HERE. Dabei wird die Laenge des Strings in dessen erstem Byte, das nicht zur Laenge hinzugezaehlt wird, vermerkt.

Ascii (-- char) I
(--) compiling
Wird in der folgenden Art benutzt:
Ascii ccc
wobei ccc durch ein Leerzeichen beendet wird. char ist der Wert des ersten Zeichens von ccc im benutzten Zeichensatz (gewoehnlich ASCII). Falls sich das System im kompilierenden Zustand befindet, so wird char als Konstante compiliert. Wird die :-definition spaeter ausgefuehrt, so liegt char auf dem Stack.

compile (--) 83,C
Typischerweise in der folgenden Art benutzt:
: <name> ... compile <namex> ... ;
Wird <name> ausgefuehrt, so wird die Kompilationsadresse von <namex> zum Dictionary hinzugefuegt und nicht ausgefuehrt. Typisch ist <name> immediate und <namex> nicht immediate.

Does> (-- addr) 83,I,C "does"
(--) compiling
Definiert das Verhalten des Wortes, das durch ein definierendes Wort erzeugt wurde. Wird in der folgenden Art benutzt:
: <namex> ... <create> ... Does> ... ;
und spaeter:
<namex> <name>
wobei <create> CREATE oder ein anderes Wort ist, das CREATE ausfuehrt.

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

immediate (--) 83
Markiert das zuletzt definierte Wort als "immediate", d.h. dieses Wort wird auch im kompilierenden Zustand ausgefuehrt.

Literal (-- 16b) 83,I,C
(16b --) compiling
Typisch in der folgenden Art benutzt:
[16b] Literal
Kompiliert ein systemabhaengiges Wort, so dass bei Ausuehrung 16b auf den Stack gebracht wird.

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

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

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

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

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

Interpreter - Worte

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

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

+thru (n1 n2 --) "plus-thru"
LOAD die Bloecke hintereinander, die n1..n2 vom gegenwaertigen Block entfernt sind.
Beispiel : 1 2 +thru laedt die naechsten beiden Bloecke.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fehlerbehandlung

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

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

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

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

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

diskerr (--)
Ein deferred Wort, das normalerweise mit (DISKERR vorbesetzt ist. DISKERR wird aufgerufen, wenn ein Fehler beim Massenspeicherzugriff auftrat. (DISKERR gibt dann die Meldung "error ! r to retry" aus. Wird anschliessend r gedrueckt, so wiederholt das System den Massenspeicherzugriff, der zum Fehler fuehrte. Wird eine andere Taste gedrueckt, so wird der Zugriff abgebrochen und die Meldung "aborted" ausgegeben. In diesem Fall wird die interne Verteilung der Blockpuffer nicht geaendert.

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

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

warning (-- adr)
adr ist die Adresse einer Variablen. Ist der Inhalt der Variablen null, so wird die Warnung "exists" ausgegeben, wenn ein Wort redefiniert wird. Ist der Wert nicht null, so wird die Warnung unterdrueckt.

Sonstiges

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

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

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

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

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

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

bye (--)
Dieses Wort fuehrt FLUSH und EMPTY aus. Anschliessend wird der Monitor des Rechners angesprungen oder eine andere implementationsabhaengige Funktion ausgefuehrt.

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

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

noop (--)
Tut gar nichts.

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

restart (--)
Bewirkt den Warmstart des Systems. Dieses Wort wird beim C64 durch Druecken von RUN/STOP - RESTORE aufgerufen. Es setzt 'QUIT , ERRORHANDLER und 'ABORT auf ihre normalen Werte und fuehrt ABORT aus.

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

Massenspeicher

>drive (block #drv -- block') "to-drive"
block' ist die Nummer des Blocks block auf dem Laufwerk #drv, bezogen auf das aktuelle Laufwerk (Vergleiche OFFSET und DRIVE) . Beispiel:
5 20 >drive block
holt den Block mit der Nummer 20 vom Laufwerk 5, egal welches Laufwerk gerade das aktuelle ist.

all-buffers (--)
Belegt den gesamten Speicherbereich zwischen dem Ende des Returnstacks und LIMIT mit Blockpuffern.

allotbuffer (--)
Fuegt der Liste der Blockpuffer noch einen weiteren hinzu, falls oberhalb vom Ende des Retunstacks dafuer noch Platz ist. FIRST wird entsprechend geaendert. Vergleiche ALL-BUFFERS

b/blk (-- &1024) "bytes pro block"
&1024 ist die Anzahl der Bytes in einem Block.

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

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

block (u -- addr) 83
addr ist die Adresse des ersten Bytes des Blocks u in dessen Blockpuffer. Der Block u stammt aus dem File in FILE . Falls der Block in diesem Puffer UPDATEd und nicht der Block u ist, dann wird er auf den Massenspeicher zurueckuebertragen, bevor der Blockpuffer an den Block u vergeben wird. Befindet sich der Block u nicht in einem Blockpuffer, so wird er vom Massenspeicher in einen an ihn vergebenen Blockpuffer (s.o.) geladen. Eine Fehlerbedingung liegt vor, falls u keine Nummer fuer einen vorhandenen Block ist. Nur Daten im letzten Blockpuffer, der von BLOCK oder BUFFER vergeben wurde, sind gueltig. Alle anderen Blockpuffer duerfen nicht mehr als gueltig angenommen werden. Der Inhalt eines Blockpuffers sollte nur veraendert werden, wenn die Aenderungen auch auf den Massenspeicher uebertragen werden sollen.

buffer (u -- adr) 83
Vergibt einen Blockpuffer an den Block u. addr ist die Adresse des ersten Bytes des Blocks in seinem Puffer. Dieses Wort entspricht BLOCK , jedoch mit der Ausnahme, das, wenn der Block noch nicht im Speicher ist, er nicht vom Massenspeicher geholt wird. Daher ist der Inhalt dieses Blockpuffers nicht festgelegt.

convey (blk1 blk2 to.blk --)
Die Bloecke im Ursprungsbereich von blk1 bis blk2 inclusive werden in den Zielbereich ab Block to.blk verschoben. Die Bereiche duerfen sich ueberlappen. Ist blk2 kleiner als blk1, so wird "nein" ausgegeben und die Ausfuehrung abgebrochen.

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

core? (blk file -- addr/false)"core-question"
Sofern sich der Block blk aus dem File file im Speicher befindet, ist addr die Adresse des ersten Bytes des Blocks in seinem Blockpuffer. Befindet sich der Block nicht im Speicher, so wird false als Ergebnis geliefert. Vergleiche BUFFER und FILE

drive (#drv --)
Selektiert das Laufwerk #drv als aktuelles. Dann liefert 0 BLOCK die Adresse des ersten Blocks auf diesem Laufwerk. Vergleiche >DRIVE und OFFSET.

drv? (block -- #drv) "drive-question"
#drv ist das Laufwerk, auf dem sich der Block block befindet. Vergleiche OFFSET >DRIVE und DRIVE.

empty-buffers (--)
Loescht den Inhalt aller Blockpuffer. UPDATEd Blockpuffer werden nicht auf den Massenspeicher zurueckgeschrieben.

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

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

flush (--) 83
Fuehrt SAVE-BUFFERS aus und loescht anschliessend alle Blockpuffer. Vergleiche EMPTY-BUFFERS

freebuffer (--)
Der Blockpuffer, auf den FIRST zeigt, wird, falls UPDATEd, auf den Massenspeicher gebracht und von der Liste der Blockpuffer entfernt. FIRST wird entsprechend veraendert. Der Speicherbereich, in dem sich dieser Puffer befand, steht damit zur Verfuegung. Gibt es nur noch einen Blockpuffer, so geschieht nichts.

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

offset (-- addr)
addr ist die Adresse einer Uservariablen, die einen Offset enthaelt. Dieser Offset wird zu der Blocknummer addiert, die sich bei Aufruf von BLOCK BUFFER usw. auf dem Stack befindet. Die Summe ergibt die Nummer des Blocks, der vom Massenspeicher geholt wird.

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

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

save-buffers (--) 83
Der Inhalt aller Blockpuffer, die als UPDATEd gekennzeichnet sind, wird in ihre korrespondierenden Massenspeicherbloecke zurueckgeschrieben. Alle Blockpuffer werden als unveraendert gekennzeichnet, bleiben aber an ihre Bloecke vergeben.

update (--) 83
Der zuletzt mit BLOCK BUFFER usw. zugegriffene Block wird als veraendert gekennzeichnet. Bloecke mit solcher Kennzeichnung werden auf den Massenspeicher zurueckgeschrieben, wenn ihr Blockpuffer fuer einen anderen Block benoetigt oder wenn SAVE-BUFFERS ausgefuehrt wird. Vergleiche PREV

C64-spezifische Worte

(bload (-- flag)
laedt ein File von einem externen Geraet. Ein Fehler beim Laden wird durch flag <> FALSE angezeigt. Die File- Parameter muessen in der Zeropage abgelegt sein. Siehe (BSAVE .

(bsave (-- flag)
speichert ein File auf ein externes Geraet. Ist flag = FALSE , so war das Abspeichern erfolgreich, sonst ist flag <> FALSE. (BLOAD setzt voraus, dass die File- Parameter in der Zeropage abgelegt sind:

ADR	Typ	Bedeutung
0AE	Wort	Endadresse des Files +1
0B7	Byte	Laenge des Filenamens
0BA	Byte	Device Nummer
0BB	Wort	Adresse des Filenamens
0C1	Wort	Anfangsadresse des Files

Vergleiche (BLOAD .

1541r/w (adr blk file r/wf -- flag) "15-41-r-w"
liest oder schreibt einen Block Daten ($400 (&1024) Bytes) vom Diskettenblock blk nach adr bzw. von adr auf den Diskettenblock blk. Ist r/wf = FALSE , so wird auf die Diskette geschrieben, ist r/wf <> FALSE , so wird von der Diskette gelesen. 1541R/W ermittelt die zu lesenden bzw. zu beschreibenden Sektoren, fuehrt das Lesen bzw. Schreiben aus und prueft auf alle moeglichen Fehler. Vergleiche R/W .

file ist die Nummer des Files, dem der zu uebertragende Block zugeordnet ist. Zur Zeit ist ausschliesslich der Direktzugriff auf die Diskette realisiert. Deshalb wird eine Fehlerbehandlung eingeleitet, wenn file <> 0 ist.

?device (dev# --) "question-device"
prueft, ob das Geraet mit der Nummer dev# am seriellen Bus anwesend ist. Meldet sich das Geraet nicht, so wird eine Fehlerbehandlung eingeleitet. Ein Fehler liegt vor, wenn dev# nicht die Bedingung 4 =< dev# =< $0F (&15) erfuellt. Typische dev# sind 8 fuer das Diskettenlaufwerk und 4 fuer den Drucker.

bus! (8b --) "bus-store"
gibt 8b ueber den seriellen Bus aus. Ein Fehler liegt vor, wenn die Ausgabe nicht vorbereitet wurde. (Z.B. durch BUSOUT oder BUSOPEN).

bus@ (-- 8b) "bus-fetch"
holt 8b vom seriellen Bus. Ein Fehler liegt vor, wenn die Eingabe nicht vorbereitet wurde. (Z.B. durch BUSIN).

busclose (dev# 2nd --)
beendet die Ausgabe an oder die Eingabe vom Geraet mit der Nummer dev# in bzw. aus einer mit BUSOPEN eroeffnete Datei. BUSCLOSE braucht nicht mit BUSOFF abgeschlossen werden. Weiteres Verhalten siehe BUSOUT .

busin (dev# 2nd --)
bereitet die Eingabe ueber den seriellen Bus vom Geraet mit der Nummer dev# vor. Verhaelt sich sonst wie BUSOUT , siehe dort.

businput (adr u --)
holt u Zeichen vom seriellen Bus und legt sie im Speicher ab adr ab. Ein PAUSE wird ausgefuehrt. Ein Fehler liegt vor, wenn die Eingabe nicht vorbereitet wurde. (Z.B. durch BUSIN).

busoff (--)
gibt den seriellen Bus und den Semaphor I/O frei. Vergleiche BUSIN , BUSOUT , BUSOPEN , BUSCLOSE und I/O , LOCK , UNLOCK .

busopen (dev# 2nd --)
bereitet die Ausgabe ueber den seriellen Bus auf das Geraet mit der Nummer dev# vor und teilt dem Geraet mit, dass alle nachfolgenden Zeichen bis zum naechsten BUSOFF als Dateiname aufzufassen sind. Dient zur Vorbereitung von Ein- oder Ausgaben von bzw. in Dateien auf Diskette. Weiteres Verhalten siehe BUSOUT .

busout (dev# 2nd --)
bereitet die Ausgabe ueber den seriellen Bus auf das Geraet mit der Nummer dev# vor. Meldet sich das Geraet nicht, so wird eine Fehlerbehandlung eingeleitet, sonst bekommt das Geraet den Wert 2nd zugesandt. Zulaessige Werte sind 4 =< dev# =< $0F (&15) und 0 =< 2nd =< $1F (&31) anderenfalls liegt ein Fehler vor. Der Semaphor I/O wird gesperrt und damit der serielle Bus vor dem Zugriff durch andere Tasks geschuetzt (Vergleiche LOCK , I/O). Vergleiche BUSOFF , BUSIN , BUSOPEN , BUSCLOSE .

bustype (adr u --)
gibt u Zeichen, die ab adr im Speicher stehen, ueber den seriellen Bus aus. Ein PAUSE wird ausgefuehrt. Ein Fehler liegt vor, wenn die Ausgabe nicht vorbereitet wurde. (Z.B. durch BUSOUT oder BUSOPEN).

c64at (row col --) "c-64-at"
positioniert den Cursor in die Zeile row und die Spalte col. Ein Fehler liegt vor, wenn row > $18 (&24) oder col > $27 (&39) ist. Vergleiche AT .

c64at? (-- row col) "c-64-at-question"
ermittelt die aktuelle Cursorposition und legt die Nummer der Zeile row und die Nummer der Spalte col nacheinander auf den Stack. Vergleiche AT? .

c64cr (--) "c-64-c-r"
ein Wagenruecklauf wird auf die Console gegeben. Vergleiche CR . Es wird immer die Console angesprochen. Ein PAUSE wird ausgefuehrt.

c64decode (adr len0 key -- adr len1) "c-64-decode"
wertet key aus. Ist key weder #BS noch #CR , so wird key in der Speicherstelle adr + len0 abgelegt, das Zeichen als Echo zum Ausgabegeraet gesandt und len0 inkrementiert. Im Falle von #BS wird das letzte Echo geloescht und len0 dekrementiert, bei #CR wird len0 nicht veraendert und in die Variable SPAN kopiert. Vergleiche DECODE . Sollen andere Zeichen (z. B. Cursortasten) ausgewertet werden, so ist ein USERdecode zu schreiben und in die Input-Struktur einzuhaengen. Vergleiche INPUT: und EDIDECODE .

c64del (--) "c-64-del"
ueberschreibt das Zeichen links vom Cursor mit einem SPACE und positioniert den Cursor darauf. Vergleiche DEL . Siehe C64CR fuer weiteres Verhalten.

c64emit (8b --) "c-64-emit"
gibt 8b auf die Console aus. Alle nicht druckbaren (Steuer-) Zeichen werden durch einen Punkt ersetzt. Es wird immer die Console angesprochen. Ein PAUSE wird ausgefuehrt. Vergleiche EMIT und OUTPUT: .

c64expect (adr len --) "c-64-expect"
erwartet len Zeichen vom Eingabegeraet, die ab adr im Speicher abgelegt werden. Ein Echo der Zeichen wird ausgegeben. CR beendet die Eingabe vorzeitig. Ein abschliessendes Leerzeichen wird immer ausgegeben. Die Laenge der empfangenen Zeichenkette wird in der Variablen SPAN uebergeben. Vergleiche EXPECT . Siehe auch EDIEXPECT .

c64init (--) "c-64-init"
initialisiert den C64. Neben der normalen Reset-Initialisierung wird das basic-ROM abgeschaltet, Rahmen-, Bildschirm- und Zeichenfarbe aus INK-POT gesetzt, Repeat fuer alle Tasten eingeschaltet und der Modus Gross/Kleinschrift gewaehlt. C64INIT wird typisch durch COLD oder RESTART aufgerufen.

c64key (-- 8b) "c-64-key"
wartet auf einen Tastendruck. Waehrend der Wartezeit wird PAUSE ausgefuehrt. Der cbm-Code des Tastendrucks wird auf den Stack gelegt. Steuerzeichen werden nicht ausgewertet, sondern unveraendert abgeliefert. Vergleiche KEY .

c64key? (-- f) "c-64-key-question"
prueft, ob eine Taste gedrueckt wurde und hinterlaesst dann f = TRUE , wurde keine Taste gedrueckt, so ist f = FALSE . Vergleiche KEY? .

c64page (--) "c-64-page"
loescht den Bildschirm und positioniert den Cursor in die linke obere Ecke. Vergleiche PAGE . Siehe C64CR fuer weiteres Verhalten.

c64type (adr len --) "c-64-type"
gibt den String, der im Speicher bei adr beginnt und die Laenge len hat, auf die Console aus. Wirkt immer auf die Console. Alle nicht druckbaren (Steuer-) Zeichen werden durch einen Punkt ersetzt. Ein PAUSE wird ausgefuehrt. Vergleiche TYPE , OUTPUT: und C64EMIT .

con! (8b --) "con-store"
gibt 8b auf die CONsole aus. Es wird immer der Bildschirm angesprochen. Alle Steuerzeichen werden zur Console gesandt. Ein PAUSE wird ausgefuehrt.

curoff (--)
schaltet den Cursor aus.

curon (--)
schaltet den Cursor ein.

derror? (-- f) "derror-question"
prueft den Fehlerkanal des aktuellen Diskettenlaufwerks. Ist die Fehlernummer kleiner als $0A (&10) (das heisst, kein Fehler), so ist f = FALSE . Liegt ein Fehler vor, so wird f = TRUE und die ganze Fehlermeldung in der Form NN,Fehlertext,TT,SS ausgegeben. Dabei ist NN die Fehler-Nummer, TT die Track- uns SS die Sektor-Nummer, bei der der Fehler auftrat (alle Zahlen in decimal).

diskclose (--) "disk-close"
schliesst den mit DISKOPEN eroeffneten Diskettenkanal wieder. DISKCLOSE ist zum Abschluss von DISKOPEN vorgesehen. Vergleiche READSECTOR und WRITESECTOR .

diskopen (-- f) "disk-open"
oeffnet den Disketten-Kanal $0D (&13) fuer nachfolgende Lese- oder Schreib-Vorgaenge im direkten Zugriff auf die Diskette. DISKOPEN sollte die Aus- und Eingabe mit READSECTOR und WRITESECTOR vorbereiten. Ist das Oeffnen des Disketten-Kanals erfolgreich, so ist f = FALSE , sonst ist f = TRUE .

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

findex (from to --)
liest, unter Umgehung des BLOCK-Mechanismus, die ersten Zeilen der Blocks from bis to einschliesslich nach PAD und gibt sie aus. Der Bereich PAD bis PAD $100 + wird von FINDEX benutzt. FINDEX kann mit einer beliebigen Taste angehalten und mit RUN/STOP abgebrochen werden. Vergleiche dazu STOP? . Die Block-Buffer werden nicht veraendert. Vergleiche auch READSECTOR .

getkey (-- 8b)
holt den cbm-Code des jeweils naechsten Tastendrucks und legt ihn auf den Stack. War keine Taste gedrueckt, so ist 8b = FALSE . (Vergleiche KEY?) GETKEY liest direkt aus dem Tastaturpuffer.

i/o (-- semaphoradr) "i-o"
ein Semaphor (Ampel) (eine spezielle Variable). I/O schuetzt den seriellen Bus vor dem Zugriff durch andere Prozesse, wenn er von einem benutzt wird. Ist der Inhalt von I/O FALSE , so ist der Weg zum seriellen Bus fuer alle Prozesse freigegeben. Alle im System enthaltenen Routinen sind abgesichert. Der Benutzer muss bei eigenen Routinen selbst fuer eine Absicherung sorgen. Vergleiche LOCK , UNLOCK und die Beschreibung des Taskers.

index (from to --)
liest die BLOCKs from bis to einschliesslich und gibt deren erste Zeilen aus. INDEX kann mit einer beliebigen Taste angehalten und mit RUN/STOP abgebrochen werden. Vergleiche dazu STOP? . Die ersten Zeilen von Screens enthalten typisch Kommentare, die den Inhalt charakterisieren.

ink-pot (-- adr)
ein Byte-Feld von 3 mal 4 Byte Laenge, in denen fuer 3 verschiedene Zwecke (Einschalt-, Editor- und User-Farben) die Rahmen-, Bildschirm-, Zeichen-Farbe und ein Dummy-Byte abgelegt sind. Zur Farbaenderung ist die gewuenschte Farb-Nummer (0-&15) in das entsprechende Byte zu schreiben.

keyboard (--)
ein mit INPUT: definiertes Wort, das als Eingabegeraet die Tastatur setzt. Die Worte KEY , KEY? , DECODE und EXPECT beziehen sich auf die Tastatur. Steuerzeichen werden nicht ausgewertet. Siehe C64KEY , C64KEY? , C64DECODE und C64EXPECT. Vergleiche INPUT: , STANDARDI/O und EDIBOARD .

printable? (8b -- 8b f) "printable"
f ist TRUE , wenn 8b ein druckbares Zeichen ist, sonst ist f = FALSE .

readsector (adr tra# sec# -- f)
liest alle $100 (&256) Bytes des Sektors sec# von Spur tra# der Diskette im aktuellen Laufwerk ueber den seriellen Bus und legt sie ab adr im Speicher ab. War das Lesen erfolgreich, so ist f = FALSE , sonst ist f <> FALSE und die Fehlermeldung des Laufwerks wird ausgegeben. Im Fehlerfall wird der Speicherinhalt ab adr nicht veraendert. Der Disketten-Kanal $0D (&13) muss fuer READSECTOR offen sein (vergleiche DISKOPEN).

writesector (adr tra# sec# -- f)
schreibt $100 (&256) Bytes, die ab adr im Speicher stehen, ueber den seriellen Bus in den Sektor sec# der Spur tra# auf der Diskette im aktuellen Laufwerk. War das Schreiben erfolgreich, so ist f = FALSE , sonst ist f <> FALSE und die Fehlermeldung des Laufwerks wird ausgegeben. Der Disketten-Kanal $0D (&13) muss fuer WRITESECTOR offen sein (vergleiche DISKOPEN).

Multitasking Worte

's (Tadr -- usradr) "tick-s"
wird benutzt in der Form:
... <taskname> 's <uname> ...
liest den Namen einer Uservariablen <uname> und hinterlaesst die Adresse usradr dieser Uservariable in der durch Tadr gekennzeichneten Task. Typisch wird Tadr durch Nennung von <taskname> erzeugt. Eine Fehlerbedingung liegt vor, wenn <uname> nicht der Name einer Uservariablen ist. Vergleiche USER und TASK . 'S ist fuer die Veraenderung des Inhalts von User-Variablen einer Task durch eine andere Task vorgesehen.

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

lock (semadr --)
der Semaphor, dessen Adresse auf dem Stack liegt, wird von der Task, die LOCK ausfuehrt, blockiert. Dazu prueft LOCK den Inhalt von semadr. Zeigt der Inhalt an, dass eine andere Task den Semaphor blockiert hat, so wird PAUSE ausgefuehrt, bis der Semaphor freigegeben ist. Ist der Semaphor freigegeben, so schreibt LOCK ein Kennzeichen der Task, die LOCK ausfuehrt, in den Semaphor und sperrt ihn damit fuer alle anderen Tasks. Den Code zwischen semadr LOCK ... und ... semadr UNLOCK kann also immer nur eine Task ausfuehren. Vergleiche UNLOCK und die Beschreibung des Taskers.

multitask (--)
schaltet das Multitasking ein. Das Wort PAUSE ist nach Ausfuehrung von MULTITASK keine NOOP-Funktion mehr, sondern gibt die Kontrolle ueber den Prozessor an eine andere Task weiter.

pass (n0 ... nr-1 Tadr r --)
aktiviert die Task, die durch Tadr gekennzeichnet ist, und weckt sie auf. Tadr wird typisch durch Nennung eines Task-Namens erzeugt. r gibt die Anzahl der Parameter n0 bis nr-1 an, die vom Stack der PASS ausfuehrenden Task auf den Stack der durch Tadr gekennzeichneten Task uebergeben werden. Die Parameter n0 bis nr-1 stehen der durch Tadr gekennzeichneten Task zur weiteren Verarbeitung zur Verfuegung. Vergleiche STOP , ACTIVATE , PAUSE , UP@ und UP! .

pause (--)
eine NOOP-Funktion, wenn der Singletask-Betrieb eingeschaltet ist; bewirkt jedoch, bei aktiviertem Multitasking, dass die Task, die PAUSE ausfuehrt, die Kontrolle ueber den Prozessor an eine andere Task abgibt. Existiert nur eine Task, oder schlafen alle anderen Tasks, so wird die Kontrolle unverzueglich an die Task zurueckgegeben, die PAUSE ausfuehrte. Ist mindestens eine andere Task aktiv, so wird die Kontrolle des Prozessors von dieser uebernommen und erst bei erneuter Ausfuehrung von PAUSE oder STOP an eine andere Task weitergegeben. Da die Tasks zyklisch miteinander verkettet sind, erhaelt die Task, die zuerst PAUSE ausfuehrte, irgendwann die Kontrolle zurueck. Eine Fehlerbedingung liegt vor, wenn eine Task weder PAUSE noch STOP ausfuehrt. Vergleiche STOP , MULTITASK und SINGLETASK .

rendezvous (semadr --)
gibt den Semaphor mit der Adresse semadr frei und fuehrt PAUSE aus, um anderen Tasks den Zugriff auf das durch diesen Semaphor geschuetzte Geraet zu ermoeglichen. Anschliessend wird LOCK ausgefuehrt, um das Geraet zurueck zu erhalten. Vergleiche LOCK und UNLOCK .

singletask (--)
schaltet das Multitasking aus. Das Wort PAUSE ist nach Ausfuehrung von SINGLETASK eine NOOP-Funktion. Eine Fehlerbedingung liegt vor, wenn eine Hintergrundtask SINGLETASK ohne anschliessendes MULTITASK ausfuehrt, da die MAIN- oder TERMINAL-TASK dann nicht mehr die Kontrolle ueber den Prozessor bekommt. Vergleiche UP@ und UP! .

sleep (Tadr --)
bringt die Task, die durch Tadr gekennzeichnet ist, zum Schlafen. Tadr wird typisch durch Nennung eines Task-Namens erzeugt. SLEEP hat den gleichen Effekt, wie die Ausfuehrung von STOP durch die Task selbst. Der Unterschied ist, dass STOP in der Regel ein Endpunkt in der Bearbeitung ist, SLEEP trifft die Task zu einem nicht vorhersehbaren Zeitpunkt, so dass die laufende Arbeit der Task unterbrochen wird. Vergleiche WAKE .

stop (--)
bewirkt, dass die Task, die STOP ausfuehrt, sich schlafen legt. Der Inhalt des IP (Interpretive Pointer), des RP (Returnstack Pointer) und des SP (Stack Pointer) werden gesichert, dann wird die Kontrolle ueber den Prozessor an die naechste Task abgegeben. Diese Aktionen werden ebenfalls von PAUSE ausgefuehrt (siehe dort), der Unterschied zu

PAUSE ist, dass die Task bei STOP inaktiv hinterlassen wird, bei PAUSE dagegen aktiv. Vergleiche auch ACTIVATE , PASS , WAKE , SLEEP , UP@ und UP! .

Task (rlen slen --)
wird benutzt in der Form:
rlen slen Task <cccc>
Task ist ein definierendes Wort, das eine Task - den Arbeitsbereich fuer ein weiteres Programm, das gleichzeitig zu den schon laufenden Programmen ablaufen soll - einrichtet. Die Task erhaelt den Namen cccc, hat einen Stack von der Groesse slen und einen Returnstack von rlen Bytes. Im Stack-Bereich liegen das task-eigene Dictionary (einschliesslich PAD), das in Richtung zu hoeheren Adressen waechst und der Stack, der zu niedrigeren Adressen hin waechst. Im Returnstack-Bereich befinden sich die task-eigene Userarea (waechst zu hoeheren Adressen), und der Returnstack (wird gegen kleinere Adressen groesser). Eine Task ist damit eine verkleinertes Abbild des gesamten ultraFORTH-Systems.

Die Ausfuehrung von cccc in einer beliebigen Task hinterlaesst die gleiche Adresse, die die Task cccc selbst mit UP@ erzeugt. Diese Adresse wird von 'S , ACTIVATE , LOCK , PASS , SLEEP , TASK und WAKE benutzt.

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

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

up@ (-- Tadr) "u-p-fetch"
liefert die Adresse Tadr des ersten Bytes der Userarea der Task, die UP@ ausfuehrt. Tadr ist die Adresse, die jede Task kennzeichnet. Vergleiche dazu 'S , ACTIVATE , LOCK , PASS , SLEEP , TASK und WAKE . In der Userarea sind Variablen und andere Datenstrukturen hinterlegt, die jede Task fuer sich haben muss. Vergleiche UP! .

up! (adr --) "u-p-store"
richtet den UP (User Pointer) auf adr. Vergleiche UP@ .

wake (Tadr --)
weckt die Task, die durch Tadr gekennzeichnet ist, auf. Tadr wird typisch durch Nennung eines Task-Namens erzeugt. Die Task fuehrt ihren Code dort weiter aus, wo sie durch SLEEP angehalten wurde oder wo sie sich selbst durch STOP beendet hat (Vorsicht!) . Vergleiche SLEEP, STOP, ACTIVATE und PASS .

Input und Output Worte

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

#cr (-- n) "number-c-r"
eine Konstante, die den Wert liefert, den man durch KEY erhaelt, wenn die Return-Taste gedrueckt wird.

#tib (-- adr) 83 "number-t-i-b"
eine Variable, die die Laenge des aktuellen Textes im Text-Eingabe-Puffer haelt. Vergleiche TIB .

-trailing (adr +n0-- adr +n1) 83 "dash-trailing"
adr ist die Anfangsadresse und +n0 die Laenge eines Strings. -TRAILING veraendert +n0 so zu +n1, dass eventuell abschliessende Leerzeichen nicht mehr in der neuen Stringlaenge +n1 enthalten sind. Der String selbst bleibt unangetastet. Ist +n0 = 0, so ist auch +n1 = 0. Besteht der ganze String aus Leerzeichen, so ist +n1 = 0.

. (n --) 83 "dot"
druckt n vorzeichenbehaftet aus.

." (--) 83 I C "dot-quote"
(--) compiling
wird in :-Definitionen in der Form verwendet:
: <name> ... ." cccc" ... ;
Der String cccc wird bis zum abschliessenden " so compiliert, dass bei Ausfuehrung von <name> der String cccc ausgedruckt wird. Das Leerzeichen nach ." und das abschliessende " sind Pflicht und gehoeren nicht zum String.

.((--) 83 I "dot-paren"
(--) compiling
wird in der Form:
... .(cccc) ...
benutzt und druckt den String cccc bis zur abschliessenden Klammer sofort aus. Das Leerzeichen nach .(und die schliessende Klammer sind Pflicht und gehoeren nicht zum String.

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

>tib (-- adr) 83 "to-tib"
adr ist die Adresse eines Zeigers auf den Text-Eingabe-Puffer. Siehe TIB .

?cr (--) "question-c-r"
prueft, ob in der aktuellen Zeile mehr als C/L - $0A (&10) Zeichen ausgegeben wurden und fuehrt dann CR aus.

at (row col --)
positioniert die Schreibstelle eines Ausgabegeraetes in die Zeile row und die Spalte col. AT ist eines der ueber OUTPUT vektorisierten Worte.

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

base (-- adr) 83 U
adr ist die Adresse einer Uservariablen, die die Zahlenbasis enthaelt, die zur Wandlung von Zahlen-Ein- und Ausgaben benutzt wird.

bl (-- 16b) "b-l"
16b ist der ASCII-Wert fuer das Leerzeichen.

c/l (-- +n) "characters-per-line"
+n ist die Anzahl der Zeichen pro Bildschirm-Zeile.

col (-- u)
u ist die Spalte in der die Schreibstelle eines Ausgabe-Geraetes sich gerade befindet. Vergleiche ROW und AT? .

cr (--) 83 "c-r"
bewirkt, dass die Schreibstelle eines Ausgabe-Geraetes an den Anfang der naechsten Zeile verlegt wird. Eines der OUTPUT-Worte.

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

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

decimal (--)
stellt BASE auf $0A (&10) ein. Alle Zahlenein- und Ausgaben erfolgen im dezimalen Zahlensystem.

decode (adr +n0 key -- adr +n1)
wertet key aus. Typisch werden normale druckbare ASCII-Zeichen in die Speicherstelle adr + +n0 uebertragen, als Echo zum Ausgabegeraet gesandt und +n0 inkrementiert. Andere Zeichen (#BS, #CR, Steuercodes) koennen andere Aktionen zur Folge haben. Eines der ueber INPUT vektorisierten Worte. Vergleiche C64DECODE . Wird von EXPECT benutzt.

del (--)
loescht das letzte ausgesandte Zeichen. Eins der OUTPUT-Worte. Bei Druckern ist die korrekte Funktion nicht garantiert.

emit (16b --) 83
die unteren 7 Bit (commodore - Benutzer: Achtung: die unteren 8 Bit) werden ausgegeben. Ist das Zeichen nicht druckbar, (insbesondere alle Steuercodes) so wird stattdessen ein "." ausgegeben. Eines des OUTPUT-Worte.

expect (adr +n --) 83
empfaengt Zeichen und speichert sie im Speicher. Die Uebertragung beginnt bei adr und setzt sich zu hoeheren Adressen fort, bis ein Return erkannt oder +n Zeichen uebertragen sind. Ein Return wird nicht mit abgespeichert. Wenn +n = 0 ist, so werden keine Zeichen uebertragen. Alle empfangenen Zeichen werden als Echo, statt des Return wird ein Leerzeichen ausgegeben. Vergleiche SPAN . Eines der ueber INPUT vektorisierten Worte.

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

input (-- adr) U
adr ist die Adresse einer Uservariablen, die einen Zeiger auf ein Feld von (zur Zeit) 4 Kompilations-Adressen enthaelt, die fuer ein Eingabe-Geraet die Funktionen KEY KEY? DECODE und EXPECT realisieren. Vergleiche die gesonderte Beschreibung der INPUT- und OUTPUT-Struktur.

key (-- 16b) 83
empfaengt ein Zeichen von einem Eingabe-Geraet. Die niederwertigen 7 Bit (commodore 8 Bit) enthalten den ASCII- (commodore-) Code des zuletzt empfangenen Zeichens. Alle gueltigen ASCII- (commodore-) Codes koennen empfangen werden. Steuerzeichen werden nicht ausgewertet, sondern so, wie sie

sind, abgeliefert. Es wird kein Echo ausgesandt. KEY wartet, bis tatsaechlich ein Zeichen empfangen wurde. Eines der INPUT-Worte.

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

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

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

output (-- adr) U
adr ist die Adresse einer Uservariablen, die einen Zeiger auf ein Feld von (zur Zeit) 7 Kompilations-Adressen enthaelt, die fuer ein Ausgabe-Geraet die Funktionen EMIT , CR , TYPE , DEL , PAGE , AT und AT? realisieren. Vergleiche die gesonderte Beschreibung der INPUT- und OUTPUT-Struktur.

page (--)
bewirkt, dass die Schreibstelle eines Ausgabegeraetes auf eine leere neue Seite bewegt wird. Vergleiche C64PAGE. Eines der OUTPUT-Worte.

query (--) 83
Zeichen werden von einem Eingabe-Geraet geholt und in den Text-Eingabe-Puffer, der bei TIB beginnt, uebertragen. Die Uebertragung endet beim Emfang von Return oder wenn die Laenge des Text-Eingabe-Puffers erreicht ist. Die Werte von >IN und BLK werden auf 0 gesetzt und SPAN wird nach #TIB kopiert. Um Text aus dem Puffer zu lesen, kann WORD benutzt werden. Siehe EXPECT und "Quelltext".

row (-- n)
n ist die Zeile, in der die Schreibstelle eines Ausgabe-Geraetes sich gerade befindet. Vergleiche COL und AT? .

space (--) 83
sendet ein Leerzeichen an das Ausgabe-Geraet.

spaces (+n --) 83
sendet +n Leerzeichen an ein Ausgabe-Geraet. Ist +n = 0, so wird nichts ausgesandt.

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

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

stop? (-- flag) "stop-question"
steht vom Eingabe-Geraet ein Zeichen zur Verfuegung, so wird es geholt. Ist es #CR (commodore RUN/STOP bzw. Ctrl-C), so ist flag = TRUE, sonst wird auf das naechste Zeichen gewartet. Ist dieses jetzt = #CR (RUN/STOP) so wird STOP? mit TRUE verlassen, sonst mit FALSE .

tib (-- adr) 83 "tib"
liefert die Adresse des Text-Eingabe-Puffers. Er wird benutzt, um die Zeichen vom Quelltext des aktiven Eingabe-Geraetes zu halten. Er kann mindestens $50 (&80) Zeichen aufnehmen. Siehe >TIB .

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

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

u.r (u +n --) "u-dot-r"
druckt die Zahl u in einem +n langen Feld ohne Vorzeichen rechtsbuendig aus. Reicht +n nicht zur Darstellung der Zahl aus, so wird ueber den rechten Rand hinaus ausgegeben. Die Zahl u wird in jedem Fall vollstaendig dargestellt.

Ergänzungen / Berichtigungen des Glossars

Im folgenden werden Änderungen und Ergänzungen des Glossars zusammengefaßt. Hierbei wurden außer Worten des Forth-Kerns auch einige andere - im Lieferumfang enthaltene - Worte mitaufgenommen, um die Übersicht zu verbessern.
Vergleiche hierzu auch "Änderungen seit rev. 3.5"

Nachtrag

Create (--) 83
Ein definierendes Wort, das in der Form
Create <name>
benutzt wird. Es erzeugt einen Kopf für <name>. Die nächste freie Stelle im Dictionary (vergleiche **HERE** und **DP**) ist nach einem **CREATE** <name> das erste Byte des Parameterfeldes von <name>. Wenn <name> ausgeführt wird, legt es die Adresse seines Parameterfeldes auf den Stack. **CREATE** reserviert keinen Speicherplatz im Parameterfeld von <name>.
Das Verhalten von <name> kann mit **DOES>** verändert werden.

eate: (--)
Ein definierendes Wort, das in der Form
Create: <name> .. ;
benutzt wird. Es wirkt wie : mit der Ausnahme, daß bei Ausführung von <name> die Parameterfeldadresse von <name> auf den Stack gebracht, das Wort jedoch nicht ausgeführt wird.

exit (--) 83,C
Wird in einer :-Definition benutzt. Bei Ausführung von **EXIT** wird in das diese :-Definition aufrufende Wort zurückgekehrt. Eine Fehlerbedingung besteht, wenn das oberste Element des Returnstacks keine Rückkehradresse enthält. **EXIT** darf nicht innerhalb von **DO .. LOOP** verwendet werden.

order (--)
Bei Aufruf wird die aktuelle Suchreihenfolge (s. "Definition der Begriffe") ausgegeben. Anschließend wird das Vokabular ausgegeben, in das neue Worte eingetragen werden.

C64- und C16-spezifische Worte

(drv (--adr)
adr ist die Adresse einer Variablen, die das aktuelle Laufwerk enthält. Der Inhalt ist i.a. gleich 0. Wenn Diskettenzugriffe nicht über den Blockmechanismus, sondern z.B. über **READSECTOR** bzw. **WRITESECTOR** gehen, sollte **(DRV** gesetzt werden.

(64 (--) I
(16 (--) I
Diese Worte ermöglichen es maschinenspezifische Teile in einer gemeinsamen Quelle zu vereinigen. Typische Benutzung:

```
    ...WorteFürAlleMaschinen...
(64 ......WorteNurFürC64..... C)
(16 ......WorteNurFürC16..... C)
    ...WorteFürAlleMaschinen...
```

Beim C64 bewirkt **(64** nichts, **(16** ignoriert alles bis **C)** . Beim C16 bewirkt **(16** nichts, **(64** ignoriert alles bis **C)**.

C) (--) I
Tut nichts und zwar sofort. Vgl. **NOOP (16 (64.**

c64fkeys (--)
Dieses Wort gibt es nur auf dem C16. Es stellt die Funktionstastenbelegung des C64 her.

Tools

cpush (adr u --)
Es werden analog **PUSH** u bytes ab inclusive adr gesichert. Beim nächsten **EXIT** oder **UNNEST** werden sie zurückgespeichert.

debug (--)
Benutzt in der Form:
debug <name>
Hierbei ist <name> ein ausführbares Wort. Aktiviert den Tracer, so daß das Wort <name> schrittweise ausgeführt wird.

endloop (--)
Deaktiviert den Tracer für das angezeigte Wort; er bleibt jedoch für alle folgenden Worte aktiv. Ist das angezeigte Wort (**LOOP** oder (**+LOOP** oder ein von **REPEAT** oder **UNTIL** kompilierter Sprung, so kann damit das (wiederholte) Tracen der Schleife unterdrückt werden.

nest (--)
Weist den Tracer an, das angezeigte Wort ebenfalls zu tracen.

trace' (--)
Benutzt in der Form:
Trace' <name>
Wirkt wie **DEBUG** <name> , zusätzlich wird jedoch das zu tracende Wort anschließend ausgeführt.

unnest (--)
Weist den Tracer an, das getracete Wort zuende auszuführen und erst ab dem aufrufenden Wort wieder zu tracen.

unbug (--)
Deaktiviert den Tracer. Die Ausführung wird fortgesetzt.

Kassettenversion

\IF (--) I
Benutzt in der Form
\IF <name> <worte> ..
Wenn <name> in der aktuellen Suchreihenfolge gefunden wird, werden die folgenden Worte bis zum Zeilenwechsel ausgeführt, sonst geschieht nichts. Gegenstück zu **\NEEDS**

(rd (--adr)
adr ist die Adresse einer Variablen, die die Anfangsadresse der aktuellen Ramdisk enthält bzw. 0 wenn explizit keine Ramdisk existiert. Zum Format der Ramdisk vgl. die Shadows des Quelltextes. Vgl. auch **RD**

.rd (--)
Druckt zentrale Informationen über die Ramdisk aus.

7>c (8b--7b) "seven-to-char
Zur Rück-Umwandlung von mit C>7 erzeugten 7-bit-Buchstaben in Buchstaben. Funktioniert nur für den von ultraFORTH benutzten Zeichensatz sicher.

autoload (--adr)
adr ist die Adresse einer Variablen. Ist sie ungleich 0, so wird beim nächsten **TAPEINIT** eine Ramdisk geladen. Wird i.a. nur vor **SAVESYSTEM** benutzt.

binary (u--u)
u ist die BlockNummer eines Blockes, der nicht komprimiert werden soll. Dies ist für binäre Ramdisk-Blöcke erforderlich, da sie sonst verändert würden. Der Block belegt ab Deklaration ca. 1024 Bytes.

bload (adr1 adr3 8b-- adr2)
Ab adr1 wird ein File mit dem 8b langen Namen, der ab adr3 im Speicher ahgelegt ist, vom mit device gesetzten Gerät gelesen. Diverse Fehlerbedingungen werden behandelt. adr2 ist die Endadresse des geladenen Files plus 1 . Beim C16/C64 hat der Benutzer bei allen Lade-Operationen Sorge zu tragen, daß genug Platz vorhanden ist, sonst folgen undefinierte Resultate. (System-Absturz)

bsave (adr1 adr2 adr3 8b--)
Der Bereich von adr1 bis exklusive adr2 wird mit dem 8b langen Namen, der ab adr3 im Speicher abgelegt ist, auf das mit device gesetzte Gerät geschrieben. Diverse Fehlerbedingungen werden behandelt.

c>7 (8b--7b) "char-to-seven"
Zur Umwandlung von Buchstaben in 7-bit-Buchstaben. Die Zurückverwandlung mit 7>C funktioniert nur für den von ultraFORTH benutzten Zeichensatz sicher.

cload (adr1 adr3 8b u1--adr2 u2)
Lädt ab adr1 das File mit dem 8b langen Namen, der bei adr3 steht von dem Gerät der Nummer u1. u1 enthält beim C16/C64 im unteren Byte die Geräte-Nummer und im oberen Byte die Sekundäradress (i.a.=0). adr2 ist die Endadresse+1 des geladenen Files. u2 ist eine Fehlerbeschreibung, die zum Aufruf von **DERR?** benutzt werden sollte. Beim C16/C64 hat der Benutzer bei allen Lade-Operationen Sorge zu tragen, daß genug Platz vorhanden ist. Sonst folgen undefinierte Resultate. (System-Absturz)

commodore (--)
Setzt den Kassettenrekorder im Commodore-Format als aktuelles Ausgabegerät. Vgl.: **DEVICE SUPERTAPE FLOPPY**

compress (adr1 adr2 u1--u2)
Zum Komprimieren von Forth-Quelltexten. Beginnend bei adr1 werden u1 Bytes zur adr2 komprimiert. Die Länge des komprimierten Bereichs ist u2 . Sie beträgt ca. 30-50% der ursprünglichen Länge. Vgl.: **EXPAND**

csave (adr1 adr2 adr3 8b u1--u2)
Sichert den Speicherbereich von adr1 bis exklusive adr2 unter dem 8b langen Namen, der bei adr3 steht auf das Gerät der Nummer u1. u1 enthält beim C16/C64 im unteren Byte die Geräte-Nummer und im oberen Byte die Sekundäradresse (i.a.=0). u2 ist eine Fehlerbeschreibung, die zum Aufruf von **DERR?** benutzt werden sollte.

derr? (u--flag)
Wird nach **CLOAD** und **CSAVE** benutzt. Falls ein Fehler aufgetreten ist, gibt es eine u entsprechende Fehlermeldung aus. flag ist true, wenn ein Fehler aufgetreten ist, sonst false.

device (--adr)
adr ist die Adresse einer Variablen, die im niederwertigen Byte die Geräte-Nummer des aktuellen Gerätes enthält und im höherwertigen Byte die Sekundäradresse (i.a.=0). Vgl.: COMMODORE SUPERTAPE FLOPPY

expand (adr1 adr2 u1--u2)
Zum Expandieren von komprimierten Forth-Quelltexten. Beginnend bei adr1 werden u1 Bytes zur adr2 expandiert. Die Länge des expandierten Bereichs wird in u2 zurückgegeben. Vgl.: **COMPRESS**

floppy (--)
Setzt das Diskettenlaufwerk als aktuelles Ausgabegerät. Vgl.: **DEVICE COMMODORE SUPERTAPE**

id" (--)
Benutzt in der Form
id" ccc"
Setzt den Namen der aktuellen Ramdisk auf RD.ccc

loadramdisk (--)
Es wird eine neue Ramdisk eingerichtet und vom aktuellen Gerät geladen. Beim C16/C64 hat der Benutzer bei allen Lade-Operationen Sorge zu tragen, daß genug Platz vorhanden ist. Sonst folgen undefinierte Resultate. (System-Absturz)

memtop (-- adr)
adr ist die erste Adresse oberhalb des verfügbaren RAM-Bereiches. Sie ist systemabhängig.

n" (--adr 8b)
Ist identisch der Sequenz
Ascii " parse
Der Quelltext bis inklusive dem nächsten " wird nicht ausgeführt, sondern als Zeichenkette verstanden. Sie beginnt bei der Adresse adr und ist 8b Bytes lang.

ramdisk (--)
Ein Vokabular, in sich Worte der Ramdisk befinden.

ramR/W (adr1 u adr2 flag--flag)
R/W wird bei Benutzung der Ramdisk auf **RAMR/W** gesetzt. Dadurch wird bei Blockzugriffen auf Drive 0 weiterhin auf ein Diskettenlaufwerk zugegriffen, Blockzugriffe auf Drive 1 und folgende werden jedoch auf die aktuelle Ramdisk umgeleitet. Zur Benutzung der Ramdisk vgl.: **DRIVE >DRIVE DRV? COPY CONVEY**

rd (--adr)
adr ist die Anfangsadresse der aktuellen Ramdisk. Wenn keine gültige Ramdisk eingerichtet ist, wird eine Fehlerbehandlung eingeleitet. Zum Format der Ramdisk vgl. die Shadows des Quelltext. Vgl. auch (RD

rdcheck (--)
Druckt Informationen über die Ramdisk aus und prüft die zentralen Zeiger.

rddel (--)
Löscht alle Blöcke der aktuellen Ramdisk.

rdnew (adr1 adr2--)
Richtet eine neue Ramdisk von adr1 bis maximal adr2-1 ein und setzt sie als die aktuelle. Eine Fehlerbedingung liegt vor, wenn der Speicherbereich von adr1 bis adr2 bereits anders belegt ist.

rduse (adr--)
Erklärt die Ramdisk ab adr zur aktuellen. Eine Fehlerbedingung liegt vor, wenn adr nicht die Anfangsadresse einer Ramdisk ist.

restore" (--)
Wirkt wie **ABORT"** , mit der Ausnahme, daß vorher mit **STORE** gesicherte Bereiche zurückgespeichert werden.

saveramdisk (--)
Die aktuelle Ramdisk wird auf das aktuelle Gerät gesichert. Eine Fehlerbedingung liegt vor, wenn keine Ramdisk eingerichtet war.

store (adr --)
Wirkt wie **PUSH**, mit der Ausnahme, daß bei einem anschließenden **RESTORE"** der gesicherte Bereich zurückgespeichert wird.

supertape (--)
Setzt den Kassettenrekorder im Supertape-Format als aktuelles Ausgabegerät. Supertape ist eine Schnell-Lade-Routine, die von der Zeitschrift "c't" für alle gängigen Mikro-Rechner angeboten wird. Wir danken dem Heise-Verlag für die freundliche Genehmigung, es in ultraFORTH83 integrieren und weiterverbreiten zu dürfen. Vgl.: **DEVICE COMMODORE FLOPPY**

tapeinit (--)
Initialisiert die Kassettenversion. Wenn autoload ungleich 0 ist, wird eine Ramdisk geladen. Wird im allgemeinen als **'RESTART** installiert.

Massenspeicher-Utilities

2disk1551 (--)
Sendet beim C16 einen Befehl über den Bus, der ein anwesendes Diskettenlaufwerk 1551 auf Unit 9 umstellt. Alle nicht gemeinten Laufwerke sind vorher auszuschalten.

bamallocate (--)
Kennzeichnet alle Sektoren einer Diskette als belegt.

copy2disk (--)
Kopiert eine Diskette von einem Laufwerk auf ein anderes. Die Directory wird mitkopiert, daher auch für Files zu verwenden.

copydisk (u1 u2 u3 --)
Kopiert die Blöcke u1 bis u2 einer Diskette ab Block u3 auf die andere Diskette.

formatdisk (--)
Formatiert eine Diskette für die Benutzung unter ultraFORTH83. Beim C16/C64 benutzt in der Form:

```
formatdisk [<name>,[<id>]]
```

Neue Disketten müssen mit ,[<id>] formatiert werden.
Beispiel:

```
formatdisk hallo,xx
```

savesystem (--)
Benutzt in der Form

```
savesystem <name>
```

Das ultraFORTH-System wird bis **HERE** auf einen externen Massenspeicher gesichert. Vorher wird es in den Zustand versetzt, in dem es beim nächsten Kaltstart sein soll. Insbesondere werden die aktuellen Parameter von Uservariablen in den User-Kaltstart-Bereich kopiert. Nicht gesichert werden Blockpuffer. S.a. "Erstellen eines eigenen Arbeitssystems".


Definition der Begriffe
Definition der Begriffe
Definition der Begriffe
FORTH-Gesellschaft eV
FORTH-Gesellschaft eV
Definition der Begriffe
Definition der Begriffe
Begriffe
ultraFORTH83
Definition der Begriffe
Definition der Begriffe
Definition der Begriffe

Entscheidungskriterien

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

1. Korrekte Funktion - bekannte Einschraenkungen,Eindeutigkeit
2. Transportabilitaet - wiederholbare Ergebnisse, wenn Programme zwischen Standardsystemen portiert werden
3. Einfachheit
4. Klare, eindeutige Namen - die Benutzung beschreibender statt funktionaler Namen, zB [COMPILE] statt 'c, und ALLOT statt dp+!
5. Allgemeinheit
6. Ausfuehrungsgeschwindigkeit
7. Kompaktheit
8. Kompilationsgeschwindigkeit
9. Historische Kontinuitaet
10. Aussprechbarkeit
11. Verstaendlichkeit - es muss einfach gelehrt werden koennen

Adresse, Byte (address, byte)
Adresse, Kompilation (address, compilation)
Adresse, Natuerliche (address, native machine)
Adresse, Parameterfeld (address, parameter field) ""apf""
anzeigen (display)
Arithmetik, 2er-komplement (arithmetic, two's complement)
Block (block)
Blockpuffer (block buffer)
Byte (byte)
Kompilation (compilation)
Definition (Definition)
Dictionary (Woerterbuch)
Division, floored (division, floored)
Empfangen (receive)
Falsch (false)
Fehlerbedingung (error condition)
Flag (logischer Wert)
Floor, arithmetic
Glossar (glossary)
Interpreter, Adressen (interpreter, address)
Interpreter, Text (interpreter, text)
Kontrollstrukturen (structure, control)
laden (load)
Massenspeicher (mass storage)
Programm (program)
Quelltext (input stream)
Rekursion (recursion)
Screen (Bildschirm)
Suchreihenfolge (search order)
stack, data (Datenstapel)
stack, return (Ruecksprungstapel)
String, counted (abgezaehlte Zeichenkette)
String, Text (Zeichenkette)
Userarea (Benutzerbereich)
Uservariable (Benutzervariable)
Vokabular (vocabulary)
Vokabular, Kompilation (vocabulary, compilation)
Wahr (true)
Wort, Definierendes (defining word)
Wort, immediate (immediate word)
Wortdefinition (word definition)
Wortname (word name)
Zahl (number)
Zahlenausgabe, bildhaft (pictured numeric output)
Zahlenausgabe, freiformatiert (free field format)
Zahlentypen (number types)
Zahlenumwandlung (number conversion)
Zeichen (character)
Zustand (mode)

Definition der Begriffe

Es werden im allgemeinen die amerikanischen Begriffe beibehalten, es sei denn, der Begriff ist bereits gelauefig. Wird ein deutscher Begriff verwendet, so wird in Klammern der engl. Originalbegriff beigefuegt; wird der Originalbegriff beibehalten, so wird in Klammern eine moeglichst treffende Uebersetzung angegeben.

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

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

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

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

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

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

Block (block)
: Die 1024byte Daten des Massenspeichers, auf die ueber Blocknummern im Bereich <0...Anzahl_existenter_Bloecke-1> zugegriffen wird. Die exakte Anzahl der Bytes, die je Zugriff auf den Massenspeicher uebertragen werden, und die Uebersetzung von Blocknummern in die zugehoerige Adresse des Laufwerks und des physikalischen Satzes, sind rechnerabhaengig.
Siehe: "Blockpuffer" und "Massenspeicher"

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

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

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

Definition (Definition)
Siehe: "Wortdefinition"

Dictionary (Woerterbuch)
Eine Struktur von Wortdefinitionen, die im Hauptspeicher des Rechners angelegt ist. Sie ist erweiterbar und waechst in Richtung hoeherer Speicheradressen. Eintraege sind in Vokabularen organisiert, so dass die Benutzung von Synonymen moeglich ist, d.h. gleiche Namen koennen, in verschiedenen Vokabularen enthalten, vollkommen verschiedene Funktionen ausloesen.
Siehe: "Suchreihenfolge"

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

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

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

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

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

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

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

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

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

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

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

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

Massenspeicher (mass storage)
Speicher, der ausserhalb des durch FORTH adressierbaren Bereiches liegen kann. Auf den Massenspeicher wird in Form von 1024byte grossen Bloecken zugegriffen. Auf einen Block kann innerhalb des Forth-Adressbereichs in einem Blockpuffer zugegriffen werden. Wenn ein Block als veraendert (UPDATE) gekennzeichnet ist, wird er

letztendlich wieder auf den Massenspeicher zurueckgeschrieben.

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

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

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

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

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

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

stack, return (Ruecksprungstapel)
Eine "Zuletzt-rein, Zuerst-raus" Struktur, die hauptsaechlich Adressen von Wortdefinitionen enthaelt, deren Ausfuehrung durch den Adressinterpreter noch nicht beendet ist. Wenn eine Wortdefinition eine andere Wortdefinition aufruft, so wird die Ruecksprungadresse auf dem Returnstack abgelegt.
Der Returnstack kann zeitweise auch fuer die Ablage anderer Daten benutzt werden.

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

String, Text (Zeichenkette)
Eine Hintereinanderfolge von 8bit Daten, die im Speicher durch ihre niedrigste Adresse und ihre Laenge in Bytes charakterisiert ist. Strings enthalten normalerweise ASCII-Zeichen. Wenn der Begriff "String" alleine oder in Verbindung mit anderen Begriffen benutzt wird, so sind Textstrings gemeint.

Userarea (Benutzerbereich)
Ein Gebiet im Speicher, das zum Ablegen der Uservariablen benutzt wird und fuer jeden einzelnen Prozess/Benutzer getrennt vorhanden ist.

Uservariable (Benutzervariable)
Eine Variable, deren Datenbereich sich in der Userarea befindet. Einige Systemvariablen werden in der Userarea gehalten, so dass die Worte, die diese benutzen, fuer mehrere Prozesse/Benutzer gleichzeitig verwendbar sind.

Vokabular (vocabulary)
Eine geordnete Liste von Wortdefinitionen. Vokabulare werden vorteilhaft benutzt, um Worte voneinander zu unterscheiden, die gleiche Namen haben (Synonyme). In einem Vokabular koennen mehrere Definitionen mit dem gleichen Namen existieren; diesen Vorgang nennt man redefinieren. Wird das Vokabular nach einem Namen durchsucht, so wird die juengste Redefinition gefunden.

Vokabular, Kompilation (vocabulary, compilation)
Das Vokabular, in das neue Wortdefinitionen eingetragen werden.

Wahr (true)
Ein Wert, der nicht Null ist, wird als "wahr" interpretiert. Wahrwerte, die von Standard-FORTH-Worten errechnet werden, sind 16bit Zahlen, bei denen alle 16 Stellen auf "1" gesetzt sind, so dass diese zum Maskieren benutzt werden koennen.

Wort, Definierendes (defining word)
Ein Wort, das bei Ausfuehrung einen neuen Dictionary-Eintrag im Kompilationsvokabular erzeugt. Der Name des neuen Wortes wird dem Quelltext entnommen. Wenn der Quelltext erschoepft ist, bevor der neue Name erzeugt wurde, so wird die Fehlermeldung "ungueltiger Name" ausgegeben.
Beispiele von definierenden Worten sind:
: CONSTANT CREATE

Wort, immediate (immediate word)
Ein Wort, das ausgefuehrt wird, wenn es waehrend der Kompilation oder Interpretation aufgefunden wird. Immediate Worte behandeln Sondersituationen waehrend der Kompilation.
Siehe z.B. IF LITERAL ." usw.

Wortdefinition (word definition)
Eine mit einem Namen versehene, ausfuehrbare FORTH-Prozedur, die ins Dictionary kompiliert wurde. Sie kann durch Maschinencode, als eine Folge von Kompilationsadressen oder durch sonstige kompilierte Worte spezifiziert sein. Wenn der Begriff "Wort" ohne Zusatz benutzt wird, so ist im allgemeinen eine Wortdefinition gemeint.

Wortname (word name)
Der Name einer Wortdefinition. Wortnamen sind maximal 31 Zeichen lang und enthalten kein Leerzeichen. Haben zwei Definitionen verschiedene Namen innerhalb desselben Vokabulars, so sind sie eindeutig auffindbar, wenn das Vokabular durchsucht wird.
Siehe: "Vokabular"

Zahl (number)
Wenn Werte innerhalb eines groesseren Feldes existieren, so sind die hoeherwertigen Bits auf Null gesetzt. 16bit Zahlen sind im Speicher so abgelegt, dass sie in zwei benachbarten Byteadressen enthalten sind. Die Bytereihenfolge ist rechnerabhaengig. Doppeltgenaue Zahlen (32bit) werden auf dem Stack so abgelegt, dass die hoeherwertigen 16bit (mit dem Vorzeichenbit) oben liegen. Die Adresse der niederwertigen 16bit ist um zwei groesser als die Adresse der hoeherwertigen 16bit, wenn die Zahl im Speicher abgelegt ist.
Siehe: "Arithmetik, 2er-komplement" und "Zahlentypen"

Zahlenausgabe, bildhaft (pictured numeric output)
Durch die Benutzung elementarer Worte fuer die Zahlenausgabe (z.B. <# # #s #>) werden Zahlenwerte in Textstrings umgewandelt. Diese Definitionen werden in einer Folge benutzt, die ein symbolisches Bild des gewuenschten Ausgabeformates darstellen. Die Umwandlung schreitet von der niedrigstwertigen zur hoechstwertigen Ziffer fort und die umgewandelten Zeichen werden von hoeheren gegen niedrigere Speicheradressen abgelegt.

Zahlenausgabe, freiformatiert (free field format)
Zahlen werden in Abhaengigkeit von BASE umgewandelt und ohne fuehrende Nullen, aber mit einem folgenden Leerzeichen, angezeigt. Die Anzahl von Stellen, die angezeigt werden, ist die Minimalanzahl von Stellen - mindestens eine - die notwendig sind, um die Zahl eindeutig darzustellen.
Siehe: "Zahlenumwandlung"

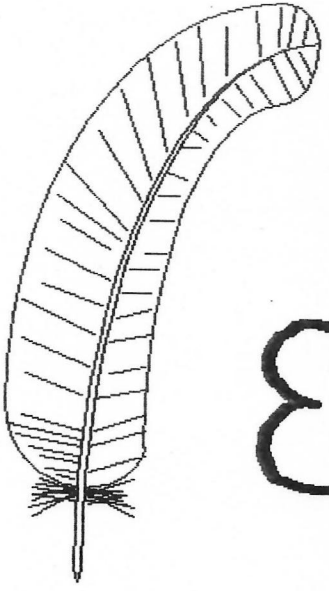
Zahlentypen (number types)
Alle Zahlentypen bestehen aus einer spezifischen Anzahl von Bits. Zahlen mit oder ohne Vorzeichen bestehen aus bewerteten Bits. Bewertete Bits innerhalb einer Zahl haben den Zahlenwert einer Zweierpotenz, wobei das am weitesten rechts stehende Bit (das niedrigstwertige) einen Wert von zwei hoch null hat. Diese Bewertung setzt sich bis zum am weitesten links stehenden Bit fort, wobei sich die Bewertung fuer jedes Bit um eine Zweierpotenz erhoeht. Fuer eine Zahl ohne Vorzeichen ist das am weitesten links stehende Bit in diese Bewertung eingeschlossen, so dass fuer eine solche 16bit Zahl das ganz linke Bit den Wert 32.768 hat. Fuer Zahlen mit Vorzeichen wird die Bewertung des ganz linken Bits negiert, so dass es bei einer 16bit Zahl den Wert -32.768 hat. Diese Art der Wertung fuer Zahlen mit Vorzeichen wird 2er-komplementdarstellung genannt.
Nicht spezifizierte, bewertete Zahlen sind mit oder ohne Vorzeichen; der Programmkontext bestimmt, wie die Zahl zu interpretieren ist.

Zahlenumwandlung (number conversion)
Zahlen werden intern als Binaerzahlen gefuehrt und extern durch graphische Zeichen des ASCII Zeichensatzes dargestellt. Die Umwandlung zwischen der internen und externen Form wird unter Beachtung des Wertes von BASE durchgefuehrt, um die Ziffern einer Zahl zu bestimmen. Eine Ziffer liegt im Bereich von Null bis BASE-1. Die Ziffer mit dem Wert Null wird durch das ASCII-zeichen "0" (Position 3/0, dezimalwert 48) dargestellt. Diese Zifferndarstellung geht den ASCII-code weiter aufwaerts bis zum Zeichen "9", das dem dezimalen Wert neun entspricht. Werte, die jenseits von neun liegen, werden durch die ASCII-zeichen beginnend mit "A", entsprechend dem Wert zehn usw. bis zum ASCII-zeichen "~", entsprechend einundsiebzig, dargestellt.Bei einer negativen Zahl wird das ASCII-zeichen "-" den Ziffern vorangestellt. Bei der Zahleneingabe kann der aktuelle Wert von BASE fuer die gerade umzuwandelnde Zahl dadurch umgangen werden, dass den Ziffern ein "Zahlenbasisprefix" vorangestellt wird. Dabei wird durch das Zeichen "%" die Basis voruebergehend

auf den Wert zwei gesetzt, durch "&" auf den Wert zehn und durch "$" oder "h" auf den Wert sechzehn. Bei negativen Zahlen folgt das Zahlenbasisprefix dem Minuszeichen. Enthaelt die Zahl ein Komma oder einen Punkt, so wird sie als 32bit Zahl umgewandelt.

Zeichen (character)
Eine 7bit Zahl, deren Bedeutung durch den ASCII-Standard festgelegt ist. Wenn es in einem groesseren Feld gespeichert ist, so sind die hoeherwertigen Bits auf Null gesetzt.

Zustand (mode)
Der Textinterpreter kann sich in zwei moeglichen Zustaenden befinden: dem interpretierenden oder dem kompilierenden Zustand. Die Variable STATUS wird vom System entsprechend gesetzt, und zwar enthaelt sie bei der Interpretation eine False-flag, bei der Kompilation eine True-flag.
Siehe: "Interpreter, Text" und "Kompilation"

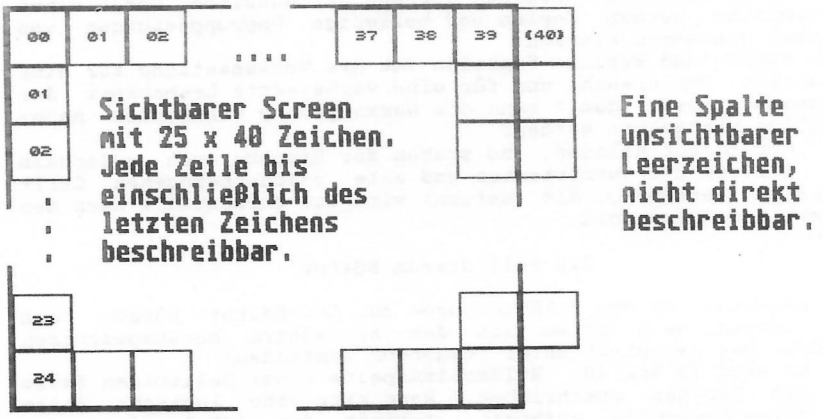


Der Editor

C64 Full Screen Editor

Allgemeines

Der Bildschirm des C64 bedingt eine Abweichung vom FORTH-Standard- Format der Screens mit 16 Zeilen zu 64 Zeichen, wenn man Rollen zur Seite oder das haessliche 1 2/5 -Zeilen-Format vermeiden will. Ein Screen wird in unserem Editor mit 24 Zeilen zu 41 Zeichen und 1 Zeile mit 40 Zeichen (zusammen 1024 Zeichen) dargestellt. Dabei sind 25 Zeilen mit 40 Zeichen sichtbar und auch voll beschreibbar, eine Spalte von Leerzeichen befindet sich rechts vom Bildschirm:



Der Bildschirm scrollt weder aufwaerts noch seitlich. Es existiert kein "Quote- Modus".

Im Editor haben verschiedene Tasten besondere Bedeutung, insbesonders sind die Funktionstasten belegt, diverse Tasten wirken, wenn sie zusammen mit CTRL betaetigt werden.

Die Auswirkung der Funktionstasten wird von oben (F1/F2) nach unten (F7/F8) geringer. Siehe die Beschreibung der Funktionstasten. Loeschfunktionen sind aus Gruenden der Sicherheit beidhaendig einzugeben (F2, F4, F6). Die CURSORtasten, INSerT, DELete und HOME verhalten sich wie gewohnt.

Der Editor bearbeitet zunaechst nur einen besonderen Buffer und veraendert den Disk- Buffer nicht. Erst auf Kommando wird der Disk- Buffer updated. Damit koennen grobe Editier-Fehler keinen grossen Schaden anrichten.

Der Editor verhindert, dass versehentlich Text verloren geht, indem er Funktionen nicht ausfuehrt, wenn dadurch Zeichen nach unten oder zur Seite aus dem Bildschirm geschoben wuerden.

Der Editor unterstützt das "Shadow-Konzept". Zu jedem Quelltext-Screen existiert ein Kommentar-Screen, so daß viel Platz für Kommentare zur Verfügung steht. Bei Benutzung dieser Screens wird die Lesbarkeit von Forthprogrammen wesentlich erhöht (obwohl ein guter FORTH-Stil selbstdokumentierend ist). Auf Tastendruck stellt der Editor den Kommentar-Screen zur Verfügung, der somit "deckungsleich" angefertigt werden kann. Das Druckerinterface druckt Quelltext und Kommentar, falls gewünscht, nebeneinander aus.
Zum "Umgraben" umfangreicher Quelltexte ist der Wechsel zwischen zwei beliebig wählbaren Screens oft nützlich. Durch einen Tastendruck werden Kopien und beliebige Umgruppierungen von Zeichen besonders einfach.
Eine Suche- und Ersetze-Funktion ist die Voraussetzung für eine effektive Fehlersuche und für eine verbesserte Lesbarkeit der Programme, denn damit kann die Namensgebung von Worten nachträglich verbessert werden.
Ist der Editor geladen, so stehen zur Eingabe auch außerhalb des Editors die Cursortasten und alle zeichenbezogenen Ctrl-Codes zur Verfügung. Mit <Return> wird die Cursorzeile nach den Änderungen übernommen.

C16 Full Screen Editor

Im Gegensatz zu den Erläuterungen zum C64-Editor können beim C16 Zeilen nach unten aus dem Bildschirm herausgeschoben werden. Das geschieht unter folgenden Umständen:

- Es wird in die 40. Bildschirmspalte einer beliebigen Zeile ein Zeichen geschrieben. Wenn hier eine logische Zeile Bildschirmzeile aufhört, schiebt die I/O-Routine des Betriebssystems eine neue Zeile ein.
- Es wird <ESC> <I> oder eine andere <ESC>-Kombination eingegeben, die eine Zeile aus dem Bildschirm schiebt.

Die C16-ESCAPE-Tasten-Funktionen, wie sie z.B. aus dem BASIC bekannt sind, können benutzt werden, führen aber zusammen mit unseren Editor-Funktionen zu sonderbaren Reaktionen auf dem Bildschirm.

von FORTH in den Editor

edit (n --)
Einstieg in den Editor Screen # n. Der Screen wird, wenn noetig, von der Diskette geholt. War das System frisch geladen, so wird vor dem Einstieg die Signatur des Benutzers erfragt. Diese Signatur kann in die rechte obere Ecke des Screens kopiert werden. Siehe GETSTAMP, "print stamp$" und "updated exit". Typisch enthaelt die Signatur das Tagesdatum und ein Programmierer- Kuerzel:
04nov85re) oder We 18. April 85)
Man befindet sich nun im Editor- Modus, den man nur mit "cancel", "updated exit", "flushed exit" oder "load exit" wieder verlassen kann.

l (n --) "list"
Einstieg in den Editor. Siehe EDIT. Unterschied zu EDIT: Der Cursor wird bei L HOME positioniert, bei EDIT nicht.

r (--) "re-list"
Einstieg in den Editor. Der zuletzt editierte Screen wird aufgerufen, mit dem Cursor dort, wo er bei Verlassen des Editors war. Bei Fehlern waehrend LOAD von der Diskette, die der Interpreter/Compiler erkennt, werden die Variablen SCR und R# mit der Fehlerstelle versorgt. Ein R zeigt dann den Screen, in dem der Fehler auftrat, der Cursor steht 2 Zeichen hinter dem bemaengelten Wort.

+l (n --) "plus-list"
Einstieg in den Editor. Zur aktuellen Screen-Nr. wird n addiert und der entsprechende Screen zum Editieren geholt. Das Gegenstueck des Editors zu +LOAD und +THRU .

view (--)
wird benutzt in der Form:
view <name>
Sucht das Wort <name> im Dictionary und ruft den zugehoerigen Quelltext- Screen zum Editieren auf. Setzt allerdings voraus, dass die richtige Diskette im Laufwerk ist. Vergleiche L und EDIT

Verlassen des Editors

CTRL c "cancel"

Der Editor wird verlassen. Dabei wird der Arbeits-Buffer- Inhalt weggeworfen und der Disk- Buffer nicht angetastet.

STOP

Der Editor wird verlassen wie mit "cancel".

CTRL x "updated-exit"

Der Editor wird verlassen. Arbeits- und Disk- Buffer werden miteinander verglichen, bei Abweichungen wird

1. die Signatur, sofern eingegeben, in die rechte obere Ecke kopiert, und
2. der Arbeits- in den Disk- Buffer kopiert, einschliesslich der nicht sichtbaren Spalte von Leerzeichen rechts der Zeilen 0-23.

Wird keine Veraenderung festgestellt, so wird der Editor wie bei "cancel" verlassen.

CTRL f "flushed-exit"

Der Editor wird verlassen wie bei "updated exit", anschliessend wird SAVE-BUFFERS ausgefuehrt, alle UPDATEd Screens werden auf die Diskette zurueckgeschrieben, sie bleiben jedoch in den Disk- Buffern. Siehe SAVE-BUFFERS und FLUSH .

CTRL l "load-exit"

Der Editor wird verlassen wie bei "flushed exit", dann wird der editierte Screen ab der aktuellen Cursorposition (!) geladen. Das Laden laesst sich optisch verfolgen. Siehe SHOWLOAD .

Die gewohnten Editiertasten

CuRSoR right

CuRSoR left

CuRSoR down

CuRSoR up

DELete "backspace"

INSerT

HOME

Verhalten sich wie gewohnt.

SHIFT HOME "to-end"
Der Cursor wird hinter das letzte Zeichen auf dem Bildschirm bewegt.

RETURN
Der Cursor wird an den Anfang der naechsten Zeile bewegt, der Insert- Modus wird geloescht, die logische Verbindung von Zeilen wird aufgehoben.

SHIFT RETURN
Siehe "RETURN".

Die Funktionstasten

F1 "insert-line"
Der Screen wird ab und einschliesslich der Cursorzeile um eine Zeile nach unten geschoben, die Cursorzeile wird mit Leerzeichen gefuellt.

F2 "delete-line"
Die Cursorzeile wird weggeworfen, der Rest des Screens um eine Zeile hochgezogen, die letzte Zeile wird mit Leerzeichen gefuellt. Um ganze Screens zu loeschen, muss man F2 benutzen. Erscheint dies zu muehsam, so definiere man sich:

```
: wipe ( --) scr @ block  b/blk bl fill ;
```

WIPE loescht den zuletzt editierten Screen.

F3 "pull-line"
Nach "insert line" wird die oberste Zeile vom Zeilen- Stack in die Cursorzeile geholt.

F4 "push-line"
Die Cursorzeile wird auf den Zeilen- Stack transportiert, der Rest des Screens um eine Zeile hochgezogen und die letzte Zeile geloescht. Vergleiche "delete line". Der Zeilen- Stack kann viele Zeilen aufnehmen, abhaengig vom Dictionary- Space oberhalb PAD bis unterhalb SP@ . Die Zeilen sind dort sicher, bis das naechste Mal compiliert wird. Auch FLUSH mit anschliessendem Diskettenwechsel beruehrt den Zeilen- Stack nicht, so dass kleine bis mittlere Kopierarbeiten ueber diesen Stack vorgenommen werden koennen. Vergleiche "copy line" und "copy char".

F5 "pull-char"
Nach einem "INSerT" wird das oberste Zeichen vom Zeichen- Stack unter die Cursorposition geholt.

F6 "push-char"
Das Zeichen unter dem Cursor wird auf den Zeichen-Stack transportiert, dann wird ein "delete char" ausgefuehrt. Der Zeichen- Stack kann 80 Zeichen aufnehmen. Die Zeichen sind dort bis zum naechsten Compilieren sicher. Vergleiche "copy char".

F7 "+tab"
Der Cursor wird um 10 Zeichen nach rechts bewegt.

F8 "-tab"
Der Cursor wird um 5 Zeichen nach links bewegt.

Andere Editor Funktionen

CTRL @ (für C64) "copy-char"
CTRL & (für C16)

Das Zeichen unter dem Cursor wird auf den Zeichenstack kopiert, der Cursor nach rechts bewegt. Vergleiche "push char" und "pull char" .

CTRL ↑ "copy-line"

Die Cursorzeile wird auf den Zeilenstack kopiert, der Cursor abwaerts bewegt. Vergleiche "push line" und "pull line".

CTRL e "erase-line"

Die Cursorzeile wird geloescht. Es wird nichts verschoben.

CTRL d "delete-char"

Das Zeichen unter dem Cursor wird geloescht, der Cursor bleibt, wo er ist, die Zeichen rechts vom Cursor werden nach links gezogen. Die gleiche Funktion liesse sich mit "CuRSoR right" und "backspace" erreichen. Vergleiche "push char".

CTRL i "insert-on"

Der Insert- Modus wird eingeschaltet, jedes in diesem Modus eingegebene Zeichen bewirkt ein vorangehendes "insert". Der neue Text wird also in den vorhandenen eingefuegt. Vergleiche "insert".

CTRL o "overwrite-on"

Der Insert- Modus wird wieder abgeschaltet.

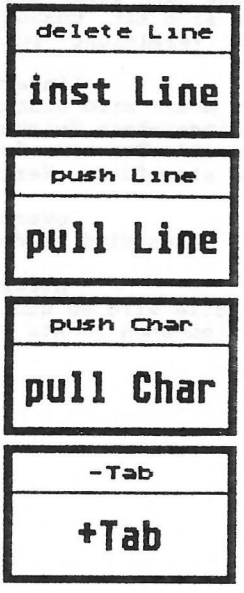
CTRL r "clear-to-right"

Die Cursorzeile wird ab und einschliesslich der Cursorposition nach rechts geloescht.

Ein bisschen Komfort(h)

CTRL $ "print-stamp-string"
Die Benutzersignatur wird in die rechte obere Ecke des Screens kopiert. Vergleiche "getstamp".

CTRL # "show-screen-number"
Die Screen- Nummer wird in der obersten Zeile eingeblendet. Jeder folgende Tastendruck loescht die Anzeige wieder.



Blaettern durch den Text

CTRL n "next-screen"
Der naechste Screen wird zum Editieren bereitgestellt. Der gerade bearbeitete Screen wird wie bei "updated exit" updated.

CTRL b "back-screen"
Der vorhergehende Screen wird zum Editieren geholt. Siehe "next screen".

CTRL w "shadow-screen"
Es wird vom Original-Screen in den Shadow-Screen, oder von dort zurueck, gewechselt. Siehe "next screen".

CTRL a "alter-screen"
Wechselt vom aktuellen in einen alternativen Screen und wieder zurueck. Nach dem Kaltstart auf Screen # 1 eingestellt. Siehe "next screen".

Suchen und Ersetzen

CTRL ' "search"

Es wird nach einem String gesucht. Zunaechst werden der letzte zu durchsuchende Screen, der Such- und der Ersatz- String abgefragt. <RETURN> laesst die Felder unveraendert, alle anderen Eingaben werden uebernommen. Die Suche beginnt im aktuellen Screen ab Cursorposition. Ist der Zielscreen vor dem Startscreen, so werden die Screens in absteigender Reihenfolge, sonst in aufsteigender Reihenfolge, durchsucht. Innerhalb der Screens wird immer von oben nach unten gesucht. Das Suchen kann jederzeit mit "STOP" abgebrochen werden.
Wird der Such- String gefunden, so haelt der Cursor dahinter. "STOP" bricht auch jetzt die weitere Suche ab, die Taste <r> ersetzt den Such- durch den Ersatz- String, alle anderen Tasten bewirken das Weitersuchen.

Andere Worte des Editors

Editor (--)
Ein Vocabulary, in das Worte compiliert sind, die mit dem Editor zu tun haben. Siehe VOCABULARY .

digits (--)
ein mit INPUT: definiertes Wort, das die Tastatur als Eingabegeraet setzt. Zeichen, die keine Ziffern darstellen (siehe BASE), werden nicht angenommen. DIGITS benutzt DIGDECODE . Siehe INPUT: , KEYBOARD und EDIBOARD .

digdecode (adr len0 key -- adr len1) "digit-decode"
wertet key aus. Ist key weder #BS noch #CR , dann wird geprueft, ob key eine gueltige Ziffer repraesentiert. Siehe BASE . Wenn nicht, bleibt len0 unveraendert und key geht verloren, sonst wird key in der Speicherstelle adr + len0 abgelegt, das Zeichen als Echo zum Ausgabegeraet gesandt und len0 inkrementiert. Im Falle von #BS wird das letzte Echo geloescht und len0 dekrementiert, bei #CR wird len0 nicht veraendert und in die Variable SPAN kopiert. DIGDECODE wird von DIGITS benutzt. Vergleiche INPUT: , DECODE , C64DECODE und EDIDECODE .

ediboard (--)
ein mit INPUT: definiertes Wort, das als Eingabegeraet die Tastatur setzt. Cursortasten und alle im Editor definierten, auf Zeichen bezogene, CTRL-Codes werden nach EDIBOARD ausgefuehrt. EDIBOARD ist, wenn der Editor geladen ist, der Standard-Input. Siehe STANDARDI/O . EDIBOARD benutzt EDIEXPECT und EDIDECODE . Vergleiche INPUT: , KEYBOARD und DIGITS .

ediexpect (adr len --)
richtet alle EDITOR- Puffer ein und erwartet dann len Zeichen vom Eingabegeraet, die ab adr im Speicher abgelegt werden. Ein Echo der Zeichen wird ausgegeben. CR beendet die Eingabe vorzeitig. Ein abschliessendes Leerzeichen wird immer ausgegeben. Die Laenge der empfangenen Zeichenkette wird in der Variablen SPAN uebergeben. Vergleiche EXPECT . Siehe auch EDIEXPECT .

edidecode (adr len0 key -- adr len1)
wertet key aus. Ist key = #CR , so wird die Zeile, in der der Cursor steht, komplett im Speicher ab adr aufwaerts abgelegt, len1 auf die aktuelle Laenge eingestellt und diese Laenge ausserdem in SPAN uebergeben. Ist key ein zeichenbezogener

CTRL-Code, so wird die zugehoerige Aktion ausgefuehrt. Ist key ein normales druckbares Zeichen, so wird es auf das Ausgabegeraet ausgegeben. len wird in diesen Faellen nicht veraendert. Vergleiche DECODE und INPUT: .

(pad (-- adr)
adr ist die Adresse einer Variablen, die die Adresse von PAD haelt. Weichen PAD und der Inhalt von (PAD ab, so werden die Editor- Buffer fuer Zeilen- und Zeichen- Stack neu initialisiert.

(search (text tlen buf blen -- adr tf // ff)
text ist die Adresse eines Textes der Laenge tlen. (SEARCH sucht diesen Text in einem Puffer, der bei buf beginnt und blen Zeichen lang ist. Wird der Text im Puffer gefunden, so wird die Adresse adr des Textes im Puffer und ein TRUE tf uebergeben, sonst nur ein FALSE ff.

getstamp (--)
fragt die Benutzer- Signatur ab. Vergleiche STAMP$ und "print-stamp-string".

stamp$ (-- adr)
adr ist die Adresse einer Datenstruktur, die die Benutzer- Signatur enthaelt. Vergleiche GETSTAMP und "print-stamp-string".

shadow (-- adr)
adr ist die Adresse einer Variablen, die den Abstand zwischen Original- und Shadow- Screen enthaelt.

v (-- +n)
wird in der folgenden Form benutzt:

```
v <name>
```

V sucht <name> im Dictionary und hinterlaesst die Nummer +n des Screens, von dem <name> compiliert wurde. Ist +n = 0 so wurde <name> vom Terminal compiliert. Vergleiche VIEW .

Besondere LOAD Worte

(load (blk +n --) "paren-load"
laedt den Block blk nicht von Anfang an, sondern ab dem +n 'sten Zeichen. Ist +n =0, so macht (LOAD dasselbe wie LOAD . Vergleiche LOAD .

showload (blk +n --)
laedt den Block blk nicht von Anfang an, sondern ab dem +n 'sten Zeichen. Beim Laden werden die Screens gelistet und eine Marke hinter den geladenen Namen gesetzt. Das Laden kann so optisch verfolgt werden. Ein "load exit" benutzt SHOWLOAD . Vergleiche (LOAD und LOAD .

Spezielle C64 Worte

ultraFORTH83 (--)
ein Wort ohne Funktion. Siehe NOOP .

.blk (--) "print-block"
druckt die Block-Nummer aus, die gerade geladen wird. Ist der Inhalt von BLK = 0, so wird nichts gedruckt. .BLK wird typisch in der Form:
' .blk Is .status
verwendet. .BLK wird nun von .STATUS ausgefuehrt. Nach dem Laden des Editors ist dies voreingestellt. Eigene Worte koennen .STATUS jederzeit zugewiesen werden.

cbm>scr (8b0 -- 8b1) "commodore-to-screen"
ein Zeichen wird von commodore- Code 8b0 in den commodore- screen- Code 8b1 gwandelt.

FORTH-Gesellschaft (--)
ein Wort ohne Funktion. Siehe NOOP .

rvsoff (--) "reverse-off"
schaltet die inverse Darstellung von Zeichen auf dem Bildschirm ab.

rvson (--) "reverse-on"
schaltet die inverse Darstellung von Zeichen auf dem Bildschirm an.

scr>cbm (8b0 -- 8b1) "screen-to-commodore"
ein Zeichen wird von commodore- screen- Code 8b0 in den commodore- Code 8b1 gwandelt. Siehe ASCII .

unlink (--)
hebt die logische Verbindung aller physikalischen Bildschirmzeilen auf. Ein, fuer normale Menschen (nicht-commodore-Benutzer), voellig unverstaendliches und unnuetzes Wort; leider notwendig.

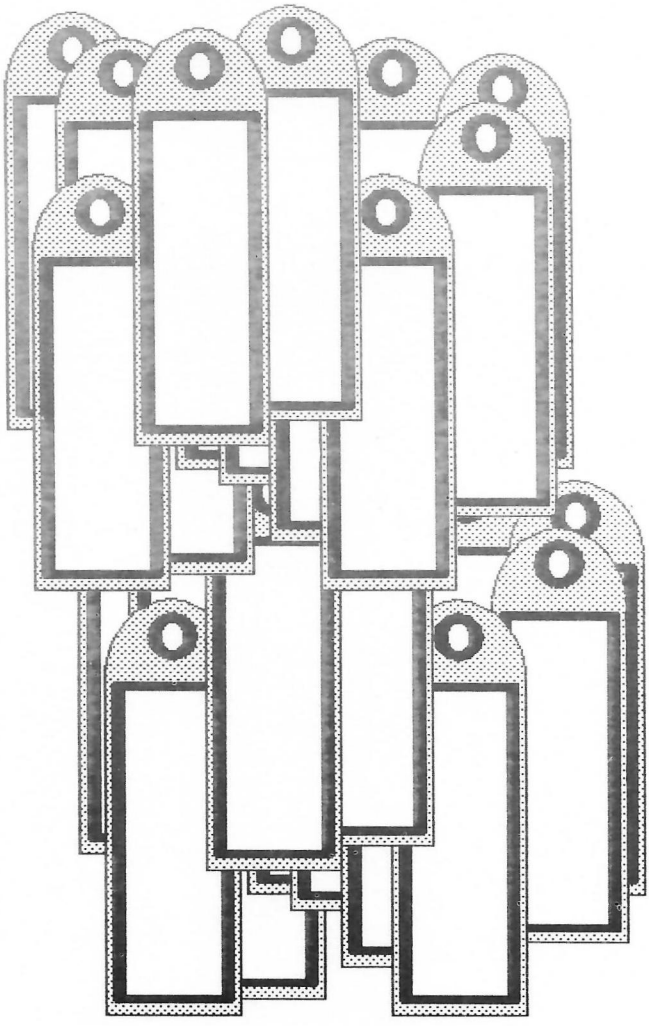
32-Bit Worte

2! (32b adr --) "two-store"
32b werden im Speicher bei adr abgelegt. Siehe 2VARIABLE und "Definition der Begriffe, Zahl (number)".

2@ (adr -- 32b) "two-fetch"
Von der Adresse adr werden 32b auf den Stack geholt. Siehe 2VARIABLE und "Definition der Begriffe, Zahl (number)".

2Constant (32b --) "two-constant"
(-- 32b) compiling
wird so benutzt:
32b 2Constant <name>
2CONSTANT erzeugt eine 32-Bit Konstante mit dem Namen <name>. 32b wird compiliert. Bei Ausfuehrung von <name> wird 32b auf dem Stack hinterlassen. Siehe "Definition der Begriffe, Zahl (number)".

2Variable (--) "two-variable"
(-- adr) compiling
wird in der Form:
2Variable <name>
benutzt. 2VARIABLE erzeugt eine 32-Bit Variable mit dem Namen <name>. Der Inhalt der Variablen wird nicht initialisiert. Bei der spaeteren Ausfuehrung von <name> wird die Adresse adr der Variablen hinterlassen. Mit 2! koennen 32b-Werte in der Variablen abgelegt und mit 2@ wieder entnommen werden. Vergleiche 2CONSTANT , 2! , 2@ , VARIABLE , ! und @ . Siehe auch "Definition der Begriffe, Zahl (number)".



Graphic Glossary

Allgemeines

Das Graphic-Paket für ultraFORTH ermöglicht die einfache Nutzung der Graphicmöglichkeiten des C64 von FORTH aus. Alle Routinen wurden auf maximale Geschwindigkeit ausgelegt, daher sind die grundlegenden Worte wie 'plot', 'line' etc. in Maschinencode geschrieben. Das Paket gliedert sich in drei Teile:

1. Hires-Graphic
2. Sprites
3. Turtle Graphic

Die Turtle-Graphic enthält alle vom LOGO her bekannten Befehle, ebenso sind die in LOGO gebräuchlichen Abkürzungen implementiert. Dies soll es vor allem dem Anfänger ermöglichen, sich spielerisch in FORTH einzuarbeiten. Ein geteilter Bildschirm (Window) ermöglicht auch das interaktive Arbeiten mit den Graphic-Befehlen, d.h. man kann die Wirkung jedes Befehls unmittelbar am Bildschirm beobachten.
Die Graphic 'verbiegt' den IRQ-Vector des Betriebssystems. Sie ist deshalb mit Routinen unverträglich, die ihrerseits den IRQ-Vector für eigene Zwecke verstellen. Dies dürfte jedoch nur in den seltensten Fällen vorkommen und zu Problemen führen.

Speicherbelegung

Der Hires-Bildschirm ist in 200 Reihen zu je 320 Punkten aufgeteilt. Der Punkt (0/0) liegt - wie in Koordinatensystemen üblich - unten (!) links, der Punkt (319/199) oben rechts. Eine Prüfung auf die Gültigkeit der eingegebenen Koordinaten findet aus Geschwindigkeitsgründen nicht statt. Allerdings werden Punkte außerhalb der Bitmap nicht gezeichnet, um ein versehentliches überschreiben des Arbeitsspeichers zu verhindern.

Eine Hires-Bitmap belegt 8k RAM. Dazu müssen Bildschirm-, Farbram, Zeichensatz und Sprites im gleichen 16k-Bereich liegen, da sie der VIC-Chip anders nicht darstellen kann. Aus diesem Grund wird für die Graphic der obere 16k-Bereich ($C000-$FFFF) eingeschaltet. Dieser Bereich ist folgendermaßen aufgeteilt:

```
C000 Videoram für Text
C400 Farbram für Hires-Screen
C800 Bereich für Spritedaten
D000 frei
D800 Charactersatz im RAM (!)
E000 Hires Bitmap
```

ACHTUNG : Auf dem C16 wurde bisher keine Graphik fertiggestellt. Wer sich daran versuchen möchte, fordere bitte die ersten Quelltexte von Claus Vogt an.

Ein- und Ausschalten der Graphic

Die Graphic benutzt ein eigenes Vocabulary, das für den Benutzer jedoch nicht aufrufbar ist, um ein versehentliches Aufrufen der Graphic-Worte zu verhindern, wenn der Graphic-Modus nicht eingeschaltet ist.

graphic (-)

Der Graphic-Bildschirm wird eingeschaltet, die Worte des Graphic-Vocabularys werden verfügbar, die Taste F1 erhält eine Umschaltfunktion (s.u.)
Sollen neue Worte ins Graphic-Vocabulary compiliert werden, lautet die Befehlsfolge
graphic also definitions.

nographic (-)

Der Graphic-Modus wird abgeschaltet, der Bildschirm liegt wieder im normalen Bildschirmbereich etc.

Innerhalb der Graphic gibt es drei Modi:

text (-)

Der gesamte Bildschirm ist im Textmodus. Insbesondere sind auch alle Editorfunktionen ausführbar. Dieses Wort wird beim Aufruf von graphic ausgeführt.

hires (-)

Der Bildschirm ist im Hires-Modus. Befehle können eingegeben und ausgführt werden, die Worte sind allerdings nicht sichtbar.

window (n -)

Der Bildschirm wird in zwei Teile geteilt. Die oberen n Zeilen werden im Hires-Modus dargestellt, der untere Teil im Text-Modus. In diesem Modus ist interaktives Arbeiten mit den Graphic-Worten besonders einfach, weil gleichzeitig die Worte und ihre Wirkung sichtbar sind.

Commodore-Taste

schaltet zwischen text, graphic und window um. der Aufruf von graphic stellt 20 Zeilen für den Hires-Bereich und 5 Zeilen für den Text-Bereich ein.

Farbeinstellungen

Die Commodore üblichen Farbcodes können durch sinnvolle Abkürzungen ersetzt werden:

blk	schwarz	(Ø)	wht	weiß	(1)
red	rot	(2)	cyn	cyan	(3)
pur	purpur	(4)	grn	grün	(5)
blu	blau	(6)	yel	gelb	(7)
ora	orange	(8)	brn	braun	(9)
lre	hellrot	(1Ø)	gr1	grau1	(11)
gr2	grau2	(12)	lgr	hellgrün	(13)
lbl	hellblau	(14)	gr3	grau3	(15)

lrscreen (-) Abkürzung cs

löscht Hires-Bildschirm

border (color -)

setzt die Rahmenfarbe für den Textmodus. Die Einstellung bleibt auch beim Rücksprung in den Normalmodus erhalten.
Beispiel: yel border

screen (color -)

setzt die Hintergrundfarbe für den Textmodus. Die Einstellung bleibt auch beim Rücksprung in den Normalmodus erhalten.
Beispiel: blu screen

background (color -) Abkürzung bg

setzt die Hintergrundfarbe für den Hiresmodus.

pencolor (color -) Abkürzung pc

setzt die Zeichenfarbe für den Hiresmodus.

colors (background foreground -)

ist eine Zusammenfassung der Worte background und pencolor, setzt gleichzeitig Zeichen- und Hintergrundfarbe für den Hiresmodus.
Beispiel: yel blu colors

Worte für Hires-Graphic

plot (x y -)

setzt einen Punkt an den Koordinaten (x/y).

unplot (x y -)

löscht einen Punkt mit den Koordinaten (x/y).

flip (x y -)

setzt einen Punkt an den Koordinaten (x/y), wenn er gelösch war, und löscht ihn, wenn er gesetzt war.

line (x1 y1 xØ yØ -)

zeichnet eine Gerade von (xØ/yØ) nach (x1/y1). Dieses Wort besteht aus einer eigenen Maschinenroutine und ist damit erheblich schneller als eine Schleife mit plot.

drawto (x1 y1 -)

zeichnet eine Gerade vom zuletzt gezeichneten Punkt zu den Koordinaten (x1/y1). Der letzte Punkt kann sowohl mit line als auch mit plot, unplot etc. gezeichnet worden sein. Seine Koordinaten liegen in den Variablen xpoint und ypoint (s.u.).
Beispiel: 1Ø 1Ø 15Ø 1Ø line
15Ø 15Ø drawto
1Ø 15Ø drawto
1Ø 1Ø drawto
zeichnet ein Quadrat.

flipline (x1 y1 xØ yØ -)

ähnlich wie line, doch werden Punkte wie bei flip umgeschaltet. Insbesondere kann ein zweites Ausführen von flipline mit denselben Koordinaten den alten Zustand auf der Bildschirm exakt wiederherstellen.

pointx (- adr)
Eine Variable, die die zuletzt gezeichnete X-Koordinate anzeigt. Jedes der oben aufgeführten Worte aktualisiert xpoint

pointy (- adr)

Eine Variable, die die zuletzt gezeichnete Y-Koordinate enthält. Wird wie pointx aktualisiert.

Worte für Sprites

Der VIC-Chip kann gleichzeitig bis zu 8 Sprites mit den Nummern (spr#) Ø - 7 darstellen. Im dafür vorgesehenen Buffer können die Daten von 32 Sprites abgelegt werden. Jedes Sprite benötigt 63 Bytes; der Buffer ist daher in Bereiche zu je 64 Bytes unterteilt, die über eine Nummer (mem#) zwischen Ø und 31 angesprochen werden können. Durch geschickte Zuordnung der Buffernummern zu den Spritenummern eröffnen sich vielfältige Möglichkeiten, da diese Zuordnung frei wählbar und natürlich auch innerhalb eines Programms änderbar ist. Sprites werden immer sowohl im Textmodus als auch im Hiresmodus dargestellt.

getform (adr mem# -)

holt sich die Daten eines Sprite von Adresse adr in den Spritebuffer mem#.
Beispiel: 3Ø block Ø getform
holt die Daten vom Diskettenblock 3Ø in den Spritebuffer Ø.

formsprite (mem# spr# -)

ordnet die Spritedaten aus Buffer mem# dem Sprite spr# zu.
Beispiel: 3 4 formsprite
Sprite 4 wird mit den Daten aus Buffer 3 dargestellt.

setsprite (mem# y x color spr# -)

setzt ein Sprite, dessen Daten im Spritebuffer unter der Nummer mem# liegen, als Sprite spr#. Es erscheint an den Koordinaten (x/y) in der Farbe color. Für x und y gelten die bei Commodore üblichen Werte, d.h. (Ø/Ø) liegt links oben im nichtsichtbaren Bereich des Bildschirms.
Beispiel: Ø 1ØØ 1ØØ blk 1 setsprite
Dieses Wort faßt mehrere andere (getform, colored etc.) zusammen, um die Programmierung zu vereinfachen. Als Nachteil muß die Übergabe von 5 Parametern in Kauf nehmen.

setbit (3b adr f1 -)

setzt oder löscht in Abhängigkeit von f1 das Bit Nummer 3b im Byte adr. Dieses Wort wird in einigen Spriteworten benutzt, um gezielt die Werte in den Spriteregistern beeinflussen zu können.

set (3b adr -)

setzt Bit Nummer 3b im Byte adr.

reset (3b adr -)

löscht Bit Nummer 3b im Byte adr.

xmove (x spr# -)

bewegt Sprite spr# an die horizontale Position x.

ymove (y spr# -)

bewegt Sprite spr# an die vertikale Position y.

move (y x spr# -)

Zusammenfassung der Worte xmove und ymove.

Die folgenden Befehle beeinflussen die Eigenschaften eines Sprites

high (spr# -)
vergrößert Sprite spr# in y-Richtung.

low (spr# -)

verkleinert Sprite spr# in y-Richtung.

wide (spr# -)

vergrößert Sprite spr# in x-Richtung.

slim (spr# -)

verkleinert Sprite spr# in x-Richtung.

big (spr# -)

vergrößert Sprite spr# in x- und y-Richtung.

small (spr# -)

verkleinert Sprite spr# in x- und y-Richtung.

behind (spr# -)

Sprite spr# erscheint hinter dem Text.

infront (spr# -)

Sprite spr# erscheint vor dem Text.

colored (spr# color -)

setzt Farbe color für Sprite spr#.

3colored (- adr)

übergibt die Adresse des Multicolorregisters auf dem Stack.
Mit der Sequenz
5 3colored set
wird der Multicolormodus für Sprite 5 eingeschaltet.

sprcolors (color color -)

setzt die Farben für den Multicolormodus.

sprite (- adr)

übergibt die Adresse des Sprite-Anzeigeregisters auf dem Stack. Mit der Sequenz
4 sprite reset
wird Sprite 4 gelöscht.

Turtle Graphic

Alle von LOGO her bekannten Worte sind mit gleicher Syntax - allerdings in der Forth-typischen präfix-Notation implementiert. Allerdings wurde auf die Darstellung der Turtle und damit auf die Befehle **showturtle** und **hideturtle** verzichtet. Einige Worte sind bereits bei der Hires-Graphic besprochen worden:

pencolor (pc)
background (bg)
clrscreen (cs)

Andere Worte haben lediglich zusätzliche Namen erhalten. Allerdings sind beide Namen in allen Bereichen der Graphic aufrufbar:

fullscreen ist gleichbedeutend mit **hires**.
splitscreen ist gleichbedeutend mit **window**.

heading (- deg)

übergibt die derzeitige Richtung der Turtle auf dem Stack. Ø° bedeutet dabei 'nach rechts', 9Ø° bedeutet 'nach oben' usw.

setheading (deg -) Abkürzung **seth**

setzt die Richtung der Turtle auf deg Grad. Richtungen s.o.

right (deg -) Abkürzung **rt**

dreht die Turtle um **deg** Grad nach rechts. Bei Überschreitung von 36Ø° wird 'um die Uhr' gerechnet.

left (deg -) Abkürzung **lt**

dreht die Turtle um **deg** Grad nach links. Bei Unterschreitung von Ø° wird 'um die Uhr' gerechnet.

forward (n -) Abkürzung **fd**

bewegt die Turtle um n Schritte in der eingestellten Richtung nach vorn. Bei Überschreitungen des Bildschirmrandes werden die Punkte zwar nicht mehr gezeichnet, aber die Position der Turtle (xcor,ycor) trotzdem beeinflußt. Der Benutzer muß selbst auf die Einhaltung der Grenzen achten !

back (n -) Abkürzung **bk**

bewegt die Turtle um n Schritte in der angegebenen Richtung zurück. Bei Bereichsüberschreitungen gilt das oben Gesagte.

xcor (- x)

übergibt die augenblickliche X-Koordinate der Turtle auf dem Stack.

ycor (- y)

übergibt die augenblickliche Y-Koordinate der Turtle auf dem Stack.

setx (x -)

setzt die Turtle auf die X-Koordinate x.

sety (y -)

setzt die Turtle auf die Y-Koordinate y.

setxy (x y -)

Zusammenfassung der Befehle **setx** und **sety**.

pendown (-) **Abkürzung pd**

Die Turtle zeichnet bei ihren Bewegungen.

penup (-) **Abkürzung pu**

Die Turtle zeichnet nicht bei ihren Bewegungen.

home (-)

Die Turtle wird in die Mitte des Bildschirms (170,100) positioniert mit Richtung nach oben (90°). Der Schreibstift wird eingeschaltet (**pendown**).

draw (-)

Der Bildschirm wird gelöscht und in ein Text- und ein Graphicfenster unterteilt, die Turtle auf Homeposition gesetzt.

nodraw (-)

schaltet in den Textmodus und löscht den Bildschirm.

turtlestate (- pen bg pc) **Abkürzung ts**

liefert auf dem Stack den augenblicklichen Zustand der Turtle in der Reihenfolge Schreibstift an (True) oder aus (False), Hintergrundfarbe, Schreibfarbe.

Der 6502-Assembler

Im folgenden werden die Konzepte des 6502-Assemblers für das ultraFORTH83 dargestellt. Es wird kein vollständiges Glossar angegeben, da die Mnemonics des Assemblers allen Programmierern vertraut sein dürften. Eine genaue Darstellung der Funktionsweise findet Sie in den FORTH DIMENSIONS, Vol III,5 p. 143ff . Im folgenden wird eine kurze Zusammenfassung angegeben sowie Änderungen gegenüber dem Original dargestellt.

Die Funktionsweise des Adressinterpreters sowie der Routine NEXT wird in Kapitel 2 dargestellt.
Der 6502-Assembler gestattet strukturierte Programmierung. Die Strukturelemente sind analog zu den Kontrollstrukturen des Forth aufgebaut, tragen jedoch andere Namen, um die Verwechselungsgefahr zu verringern und die Übersichtlichkeit zu erhöhen.
Ein Beispiel:

```
cc  ?[ <ausdruck1> ][ <ausdruck2> ]?
```

cc steht für "condition code" . <ausdruck1> wird ausgeführt, wenn cc zutrifft, andernfalls <ausdruck2> . Der Teil :

```
][ <ausdruck2>
```

kann auch weggelassen werden. Das Analogon in Forth ist IF ... ELSE ... THEN .
Beachten Sie bitte, daß vor ?[immer (!) ein condition code stehen muß. Außerdem findet keine Prüfung auf korrekte Verschachtelung der Kontrollstrukturen statt.
Weitere Kontrollstrukturen sind:

```
[[ <ausdruck1>  cc  ?[[  <ausdruck2>  ]]?
[[ <ausdruck1>  cc  ?]
[[ <ausdruck1> ]]
```

Die analogen Ausdrücke in Forth wären :

```
BEGIN  <ausdruck1>  WHILE  <ausdruck2>  REPEAT
BEGIN  <ausdruck1>  UNTIL
BEGIN  <ausdruck1>  REPEAT
```

Auch hier darf bei den Assemblerworten cc nicht weggelassen werden. Außerdem ist nur genau ein ?[[zwischen [[und]]? zulässig. Beachten Sie bitte auch den Unterschied zwischen]] und]]? !

Als condition code sind zulässig :

```
0=  0<>  0<  0>=  CS  CC  VS  VC
```

Sie können den Prozessor-Flags Z N C und V zugeordnet werden. Im ersten Beispiel wird also <ausdruck1> ausgeführt, wenn cc durch 0= ersetzt wird und das Z-Flag gesetzt ist. Jeden der condition codes kann man durch ein folgendes NOT erweitern, also z.B.:

```
0= NOT ?[  0 # lda  ]?
```

Neben allen anderen Opcodes mit ihren Adressierungsarten gibt es auch die Sprünge BCC BCS usw. Sie sind nur in der Adressierungsart Absolut zulässig, d.h. auf dem Stack befindet sich die Adresse des Sprungzieles. Liegt diese Adresse außerhalb des möglichen Bereiches, so wird die Fehlermeldung "out of range" ausgegeben.

Für die anderen Opcodes sind, je nach Befehl, die folgenden Adressierungsarten zulässig:

```
.A  #  ,X  ,Y  X)  )Y  )
```

Die Adressierungsart Absolut wird verwendet, wenn keine andere angegeben wurde. Wird mit einem Mnemonic eine nicht erlaubte Adressierungsart verwendet, so wird die Fehlermeldung "invalid" ausgegeben.

Beispiele für die Verwendung des 6502-Assemblers (zur Erläuterung wird die herkömmliche Notation hinzugefügt) :

```
.a rol              rol a
1 # ldy             ldy #1
data ,X sta         sta data,x
$6 x) adc           adc ($6,x)
vector )y lda       lda (vector),y
vector ) jmp         jmp (vector)
```

Zusätzlich enthält das System noch mehrere Macros, die alle nur Absolut adressieren können:

winc - Inkrementiert einen 16-Bit-Zeiger um 1. wdec dekrementiert analog.

2inc - Inkrementiert einen 16-Bit-Zeiger um 2. 2dec dekrementiert analog.

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

Ein Beispiel für die Verwendung von ;C: ist:

```
... 0< ?[  ;c: ." Fehler" ;  Assembler  ]?  ...
```

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

Schließlich gibt es noch die Worte >LABEL und LABEL . >LABEL erzeugt ein Label im Heap, wobei es den Wert des Labels vom Stack nimmt. LABEL erzeugt ein Label mit dem Wert von HERE. Beispiel:

```
Label schleife    dex
                  schleife bne
```

Ein Codewort muß letztendlich immer auf NEXT JMP führen, damit der Adressinterpreter weiter arbeitet. Im folgenden Glossar werden Konstanten angegeben, auf die gesprungen werden kann und die Werte auf den Stack bringen bzw. von ihm entfernen. Wichtig ist insbesondere die Routine SETUP. Sie kopiert die Anzahl von Werten, die im Akkumulator angegeben wird, in den Speicherbereich ab N.

Für den Zugriff auf den Stack wird, soweit das möglich ist, die Benutzung der Worte SETUP und PUSH ... empfohlen. Das reicht allerdings häufig nicht aus. In diesem Fall kann man die Werte auf dem Stack folgendermaßen zugreifen:

```
SP x) lda     \ Das untere Byte des ersten Wertes
SP )y lda     \ Das obere Byte des ersten Wertes
```

sowie durch Setzen des Y-Registers auch die zweiten, dritten etc. Werte. Beachten Sie bitte, das in NEXT verlangt wird, daß das X-Register den Inhalt $00 und das Y-Register den Inhalt $01 hat. Das wurde im obigen Beispiel ausgenutzt. Beispiele für Assemblercode in Forth, den wir als gut empfinden, sind unter Anderem die Worte FILL und -TRAILING . Wollen Sie Assembler programmieren, so sollten Sie sich diese Worte und noch einige andere ansehen.

Beim Assemblerprogrammieren muß beachtet werden, daß ultraFORTH das ROM abschaltet. Daher müssen Lesezugriffe ins ROM etwas anders organisiert werden.
Beispiel für den C16:

```
  ffd2 jsr                    springt eine RAM-Routine an.
  ff3e sta  ffd2 jsr  ff3f sta  springt eine ROM-Routine an.
```

Sie funktioniert nur, wenn sie im unteren RAM-Bereich (<$8000) steht. Sonst folgen undefinierte Reaktionen.
Beim C64 ist eine Bankumschaltung nur für Lesezugriffe in das BASIC-ROM erforderlich. Hierbei ist zusätzlich zu beachten, daß eine Bankumschaltung mit SEI vorbereitet werden muß, da andernfalls der periodische Tastaturinterrupt zu einem Absturz führen würde.
Auf dem C16 ist kein SEI erforderlich, da im RAM der Vektor $FFFE auf eine eigene Interruptroutine zeigt (sie benötigt ca. 1 Promille der Rechenzeit). Aus dem gleichen Grund führt eine BRK - Instruktion zwar weiterhin in den Monitor, allerdings mit falschem Registerdump, da der Monitor auf dem Stack die Daten der Interruptroutine statt der Register vorfindet.

Assembler

PushA -- addr
Eine Konstante, die die Adresse einer Maschinencode-Sequenz enthält, die den Inhalt des Akku vorzeichenbehaftet auf den Datenstack legt und dann zu NEXT springt. Wird als letzter Sprungbefehl in Code-Worten benutzt.

PushØA -- addr
Eine Konstante, die die Adresse einer Maschinencode-Sequenz enthält, die den Inhalt des Akku auf das Low-Byte des Datenstacks ablegt. Das High-Byte wird grundsätzlich auf Ø gesetzt. Anschliessend wird zu NEXT gesprungen. Wird als letzter Sprungbefehl in Code-Worten benutzt.

Push -- addr
Eine Konstante, die die Adresse einer Maschinencode-Sequenz enthält, die den Inhalt des Akku als High-Byte auf den Datenstack legt. Das Low-Byte wird vom prozessorstack geholt und muß vorher dort abgelegt worden sein. Anschliessend wird zu NEXT gesprungen. Wird als letzter Sprungbefehl in Code-Worten benutzt.

Next -- addr
Eine Konstante, die die Adresse von NEXT auf den Datenstack legt. Wird als letzter Befehl in Code-Worten benutzt.

xyNext -- addr
Wie NEXT jedoch werden vorher das X-Register mit Ø und das Y-Register mit 1 geladen. Das System erwartet grundsätzlich bei Aufruf von NEXT diese Werte in den Registern. Ansonsten reagiert es mit Absturz.

Puta -- addr
Eine Konstante, die die Adresse einer Maschinencode-Sequenz enthält, die den Akku als Low-Byte auf den Datenstack ablegt. Das High-Byte wird nicht verändert, ebenso wird im Gegensatz zu PUSH kein Platz auf dem Datenstack geschaffen. Anschliessend wird NEXT durchlaufen. Wird als letzter Befehl in Code-Worten benutzt.

Pop -- addr
Eine Konstante, die die Adresse einer Maschinencode-Sequenz enthält, die das oberste Element vom Datenstack entfernt. Anschliessend wird zu NEXT gesprungen. Wird als letzter Sprungbefehl in Code-Worten benutzt.

Poptwo -- addr
Eine Konstante, die die Adresse einer Maschinencode-Sequenz enthält, die die obersten beiden Elemente vom Datenstack entfernt. Anschliessend wird zu NEXT gesprungen. Wird als letzter Sprungbefehl in Code-Worten benutzt.

RP -- addr
Eine Konstante, die die Adresse des Returnstackpointers enthält.

UP -- addr
Eine Konstante, die die Adresse des Userpointers, also des Offsets zu ORIGIN enthält.

SP -- addr
Eine Konstante, die die Adresse des Datenstackpointers enthält.

IP -- addr
Eine Konstante, die die Adresse des Instruktionspointers der Forth-Maschine enthält. Dieser zeigt auf das jeweils nächste abzuarbeitende Wort.

W -- addr
Eine Konstante, die die Adresse des Wort-Pointers der Forth-Maschine enthält. Dieser zeigt auf das jeweils gerade bearbeitete Wort.

N -- addr
Eine Konstante, die die Adresse eines Speicherbereichs in der Zeropage enthält, der dem Anwender zur Verfügung steht.

setup -- addr
Eine Konstante, die die Adresse einer Maschinencode-Sequenz enthält, die n Elemente vom Datenstack abbaut und bei N ablegt. Die Anzahl n muß im Akku stehen, wenn SETUP als Subroutine angesprungen wird. Das oberste Element des Datenstacks liegt Bei N , das zweite bei N+2 etc. Zum Schluß werden X- und Y-Register auf Ø bzw. 1 gesetzt.

wcmp (addr1 addr2 --)
Dieses Assemblermakro assembliert eine Sequenz, die bei Ausführung den Inhalt des Wortes an addr1 mit dem Inhalt des Wortes an addr2 vergleicht. Anschließend ist das Carry-Flag high, wenn der Inhalt von addr1 größer oder gleich dem Inhalt von addr2 ist.
Es werden der Akku sowie die Statusregisterflags C Z O N verändert.

ram (--)
Makro. Schaltet bei Ausführung auf eine andere Speicherbank. Die genaue Wirkungsweise ist maschinenabhängig.

rom (--)
Makro. Schaltet bei Ausführung auf eine andere Speicherbank. Die genaue Wirkungsweise ist maschinenabhängig.

sys (addr--)
Makro. Schaltet bei Ausführung auf eine Speicherbank mit Systemroutinen und führt jsr aus. Die genaue Wirkungsweise ist maschinenabhängig.

Abweichungen des ultraFORTH83 von
Programmieren in Forth

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

Kapitel 1 - Grundlagen

3 -) Das ultraFORTH System befindet sich nach dem Einschalten im Hexadezimal-Modus. Daher muß man, um die folgenden Beispiele des Buches eingeben zu können, erst DECIMAL eintippen. Danach werden alle folgenden Zahlen als Dezimalzahlen interpretiert. Außerdem dürfen alle Worte auch klein geschrieben werden. Das Beispiel lautet dann:

```
decimal 15 spaces
```

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

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

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

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

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

17 -) Das ultraFORTH gibt bei Stackleerlauf irgendeine Zahl aus, nicht unbedingt eine Null ! Nebenbei bemerkt: Das ultraFORTH ist das erste System auf dem C64, für den der Satz mit dem großen Stack wirklich zutrifft, denn dort können sich nun typisch mehr als tausend Werte befinden.

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

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

Kapitel 2 - Wie man in Forth rechnet

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

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

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

-) 'S heißt im ultraFORTH SP@ .

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

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

-) Im ultraFORTH gibt es das Wort 2OVER nicht. Benutze:

```
: 2over   3 pick  3 pick  ;
```

Kapitel 3 - Der Editor und externe Textspeicherung

Beim ultraFORTH auf dem C64 wird ein leistungsfähiger Bildschirmeditor mitgeliefert, dessen Bedienung sich von dem im Buch benutzten Zeileneditor stark unterscheidet. Wir haben jedoch den im Buch verwendeten Editor zum System dazugepackt. Er wurde geringfügig modifiziert.

46 -) Das ultraFORTH für den C64 stellt auf einem Diskettenlaufwerk 170 Blöcke zur Verfügung. Um den Bildschirm besser ausnutzen zu können, wurden sie in 24 Zeilen mit je 41 Zeichen und eine Zeile mit 40 Zeichen aufgeteilt. Bei den Zeilen mit 41 Zeichen ist das 41. Zeichen nicht sichtbar.

48 -) Bevor man Editor-Worte benutzen kann, muß man erst EDITOR eingeben.

55 -) Das Wort F gibt nicht NONE aus, sondern positioniert nur auf das erste Zeichen in der ersten Zeile, wenn der Text nicht gefunden wurde.

56 -) Mit .I kann man sich den I-Puffer und mit .F den Suchpuffer ansehen.

60 -) Das ultraFORTH besitzt sowohl die Worte FLUSH als auch SAVE-BUFFERS . Beim Diskettenwechsel sollten sie FLUSH verwenden.

-) Das FLUSH nach COPY ist unnötig.

61 -) S gibt nur die Zeilennummer aus, und zwar vor dem Inhalt der Zeile, in der der Text gefunden wurde.

-) M ist zu gefährlich. Der Editor sollte keine fremden Screens ruinieren. Benutzen Sie das Wort G oder BRING .

66 -) DEPTH ist vorhanden, 'S heißt SP@ . Wegen der Abweichungen beim Wort DO und der Stackadressen muß 4 - weggelassen werden.

Kapitel 4 - Entscheidungen

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

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

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

Kapitel 5 - Die Philosophie der Festkomma-Arithmetik

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

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

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

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

Kapitel 6 - Schleifenstrukturen

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

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

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

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

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

Kapitel 7 - Zahlen, Zahlen, Zahlen

125 -) Erinnern Sie sich bitte daran, daß EMIT beim ultraFORTH auf dem C64 keinen ASCII Code verarbeitet. Außerdem ist es Systemabhängig, ob EMIT Steuerzeichen ausgeben kann.

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

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

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

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

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

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

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

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

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

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

Kapitel 8 - Variablen, Konstanten und Felder

158 -) 2VARIABLE 2@ und 2! sind nicht im System enthalten :

```
: 2@   dup 2+ @  swap @ ;
: 2!   rot over 2+ !   ! ;
: 2Variable   Variable  2 allot ;
: 2Constant   Create , , Does> 2@ ;
```

Kapitel 9 - Forth intern

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

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

die dem 83-Standard entsprechen, folgendermaßen:
38 ' GRENZE >BODY ! .

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

180 -) Das ultraFORTH erkennt 32-Bit Zahlen bereits serienmäßig, daher gibt es kein Wort 'NUMBER . Wollen Sie eigene Zahlenformate erkennen (etwa Floating Point Zahlen) , so können Sie dafür das deferred Wort NOTFOUND benutzen.

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

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

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

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

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

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

-) Der Bereich der Benutzervariablen liegt woanders.

191 -) Zusätzliche Definitionen : Beim ultraFORTH ist die Bibliothek anders organisiert. Ein Inhaltsverzeichnis der Disketten finden sie häufig auf Block 1 .

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

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

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

202 -) LOCATE heißt im ultraFORTH VIEW .

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

Kapitel 10 - Ein- und Ausgabeoperationen

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

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

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

-) Fußnote 4 trifft für das ultraFORTH nicht zu.

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

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

215 -) Beim C64 sind Zeilen nur 41 Zeichen lang. Verwenden Sie bitte statt 64 die Konstante C/L . Sie gibt die Länge der Zeilen in einem System an.

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

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

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

220 -) Auch für KEY gilt das für EMIT gesagte: Der Zeichensatz beim C64 ist nicht ASCII.

-) Statt der Systemabhängigen Zahl 32 sollte besser die Konstante BL verwendet werden !

223 -) TEXT ist nicht vorhandern.

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

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

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

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

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

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

Kapitel 11 - Wie man Compiler erweitert...

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

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

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

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

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

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

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

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

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

-) Fußnote 4 trifft für das ultraFORTH nicht zu.

Kapitel 12 - Drei Beispiele

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

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

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

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

Abweichungen des ultraFORTH83 von
'FORTH TOOLS'
von A. Anderson & M. Tracy

p.15 CLEAR macht im ultraFORTH83 etwas anderes als in FORTH TOOLS. Benutze statt CLEAR das Wort CLEARSTACK oder definiere
```
' clearstack Alias clear
```

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

p.34 2OVER 2ROT fehlen. Benutze
```
: 2over   3 pick  3 pick ;
: 2rot    5 roll 5 roll ;
```

p.41 Es gibt noch keine Files.

p.42 Es gibt noch kein Wort DICTIONARY

p.45 THRU druckt keine Punkte aus.

p.46 Der Editor funktioniert anders. Ausserdem enthaelt eine Zeile beim C-64 nur 40 Zeichen und nicht 64.

p.64 ultraFORTH83 enthaelt kein Wort ?. Benutze
```
: ?   @ . ;
```

p.73 Die Worte 2! 2@ 2VARIABLE und 2CONSTANT fehlen. Benutze
```
: 2Variable     Variable 2 allot ;
: 2Constant     Create , , Does> 2@ ;
: 2!            rot over 2+ !   ! ;
: 2@            dup 2+ @  swap @ ;
```

p.99 AGAIN gibt es nicht. Benutze stattdessen REPEAT oder definiere
```
' REPEAT  Alias  AGAIN  immediate restrict
```

p.103 Benutze ' extend Alias s>d

p.107 DU< fehlt.

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

p.118 CPACK entspricht dem Wort PLACE.

p.125 Benutze
```
: String   Create dup , 0 c, allot  Does> 1+ count ;
```

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

p.141 Das Wort IS aus dem ultraFORTH83 kann innerhalb von Definitionen benutzt werden.

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

p.149 Benutze
```
: >link      >name 2- ;
: link>      2+ name> ;
: n>link     2- ;
: l>name     2+ ;
```

p.166 STOP entspricht \\

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

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

Targetcompiler-Worte

Das ultraFORTH83-System wurde mit einem sog. Targetcopiler erzeugt. Dieser Targetcompiler compiliert ähnlichen Quelltext wie das ultraFORTH83-System. Es gibt jedoch Unterschiede. Insbesondere sind im Quelltext des ultraFORTH83 Worte enthalten, die der Steuerung des Targetcompilers dienen, also nicht im Quelltext definiert werden. Wenn Sie sich also über eine Stelle des Quelltextes sehr wundern, sollten Sie in der folgenden Liste die fragwürdigen Worte suchen.

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

Folgende Worte im Quelltext werden vom Targetcompiler benutzt:

origin! Target >here nonrelocate Host Transient Tudp Tvoc-link move-threads .unresolved

sowie eine Anzahl weiterer Worte.

Die mitgelieferte Quelle des Systems gilt für drei Versionen:

Die C64-Version, die C16-Version für 64 KByte (mit Interrupt-Handling) und eine leicht abgemagerte C16-Version, die nur 32 KByte nutzen kann und dafür keine INTERRUPT-Umleitung braucht.

Zur Kennzeichnung dieser Vielfalt dienen Worte, die von manchen Rechnern als Kommentare interpretiert werden, auf anderen aber Code erzeugen (Achtung : Die im Quelltext des ultraFORTH83-Kerns verwendete Syntax unterscheidet sich von der in den anderen Quelltexten verwendeten !).
Im Einzelnen sind das folgende Worte: (die Definitionen gelten, wenn Code für den C16+ erzeugt werden soll.)

```
für alle Commodores      : (c                ; immediate
für C64                  : (c64  [compile] ( ; immediate
für C16/C116 & Plus4     : (c16              ; immediate
für C16/C116 & Plus4-64k : (c16+             ; immediate
für C16/C116 & Plus4-32k : (c16- [compile] ( ; immediate
die Schließende Klammer  : )                 ; immediate
```

Alles, was bis zur nächsten schließenden Klammer auf das Wort (C64 folgt, wird also auf einem C16 als Kommentar verstanden. Auf einem C64 werden die obigen Definitionen natürlich so geändert, daß alles, was auf die Worter (C16 (C16+ und (C16- folgt, als Kommentar aufgefaßt wird.

Vor der Targetcompilierung müssen sie ggf. umdefiniert werden (s. Quelldiskette). Bei der Arbeit mit diesen Worten muß man allerdings höllisch aufpassen. Beispiele für Fehler sind:

```
1.)    (c : xx  ( Kommentar )  .... ; )
2.)    CODE xx  (c16 ..... )  (c64 ..... )  next jmp  end-code
3.)    (c CODE yy  .... SP )Y lda .... )
```

Am häßlichsten ist 2.), weil es beim Ausführen ohne jede Meldung zu einem todsicheren System-Zusammenbruch führt. Daher wird im Quelltext in einem solchen Fall eher:

```
    (c16 ...  ( )
```

geschrieben. Wer die Beispiele nicht versteht, bedenke, daß die Worte (C usw. nach der nächsten (!) schließenden Klammer suchen und daß)Y sowie) Worte des Assemblers sind.
Die Worte, die im laufenden System eine ähnliche Funktion ausführen, sind (16 (64 und C) (siehe Glossar). Sie sind so abgesichert, daß die genannten Fehlerquellen wegfallen.

Meldungen des ultraFORTH83

Das ultraFORTH83 gibt verschiedene Fehlermeldungen und Warnungen aus. Zum Teil wird dabei das zuletzt eingegebene Wort wiederholt. Diese Meldungen sind im folgenden nach Gruppen getrennt, aufgelistet:

Kernel

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

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

C) missing
Das Ende der Eingabezeile oder des Screens wurde erreicht, bevor das zu **(64** oder **(16** gehörende **C)** gefunden wurde.

compiling
Das System befindet sich im compilierenden Zustand und erwartet die Eingabe einer Zeile.

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

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

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

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

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

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

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

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

ok
Das System befindet sich im interpretierenden Zustand und erwartet die Eingabe einer Zeile.

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

no file
Auf Ihrem System sind keine Files benutzbar. Die Variable **FILE** muß daher den Wert Null haben.

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

no device
Das angewählte Gerät konnte über den seriellen Bus nicht angesprochen werden.

protected
Es wurde versucht, ein Wort zu vergessen, das mit **SAVE** geschützt wurde. Benutzen Sie die Phrase: '
' <name> >name 4 - (forget save
wobei <name> der Name des zu vergessenden Wortes ist.

read error ! r to retry
Bei einem Lesezugriff auf die Diskette trat ein Fehler auf. Durch Drücken der Taste <R> wird ein erneuter Leseversuch gestartet. Bei Drücken einer anderen Taste wird die Meldung "aborted" ausgegeben und **ABORT** ausgeführt.

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

still full
Nach der Meldung DICTIONARY FULL wurde noch kein Platz im Dictionary geschaffen. Holen Sie das unverzüglich nach, indem Sie einige Worte mit **FORGET** vergessen !

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

ultraFORTH83 rev 3.8
Dies ist die Einschaltmeldung des ultraFORTH83.

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

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

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

write error ! r to retry
Siehe " read error ..."

¦<name>
Der Name des ausgegebenen Wortes befindet sich auf dem Heap und ist nach einem **CLEAR** nicht mehr sichtbar.

RELOCATION

returnstack ?
Der Speicherplatz, der nach Ausführung von **RELOCATE** für den Returnstack übrig bliebe, ist zu klein.

stack ?
Der Speicherplatz, der nach Ausführung von **BUFFERS** oder **RELOCATE** für das Dictionary übrig bliebe, ist zu klein.

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

ASSEMBLER

invalid
Die Kombination von Opcode und Adressierungsart ist nicht zulässig.

not here
Es wird in einem Speicherbereich assembliert, in dem das Makro **ROM** nicht angewendet werden darf.

out of range
Es wurde versucht, mit einem Branch außerhalb des Bereiches von -128..+127 Bytes zu springen.

EDITOR

from keyboard
Das Wort, das **VIEW** auffinden soll, wurde von der Tastatur aus eingegeben. Es kann daher nicht auf dem Massenspeicher gefunden werden.

your stamp :
Sie werden aufgefordet, ihr Namenskürzel einzugeben. Denken Sie sich eines aus; es kann z.B. aus Ihren Initialien und dem Datum bestehen.

KASSETTENVERSION

ChSumErr
Beim Screentausch über den Userport trat ein Prüfsummenfehler auf.

error ### : <meldung> load/save
Eine I/O-Operation konnte nicht durchgeführt werden, weil der Fehler <meldung> auftrat.

no ramdisk
Es existiert keine Ramdisk. Wenn Sie eine Ramdisk benutzen wollen, müssen Sie sie zuerst einrichten (s. "ultraFORTH83 für Kassettenrecorder").

ramdisk full
Die Ramdisk ist voll. Löschen Sie Screens oder Sichern Sie die Ramdisk und legen Sie eine neue an (s. **RDDEL** **EMPTY-BUFFERS**).

range!
Die bei Ausführung von **RDNEW** auf dem Stack befindlichen Werte sind nicht plausibel.

STxxx
Das ist die Einschaltmeldung des Supertape.

SyncErr
Analog ChSumErr

terminated
Analog ChSumErr

VERSCHIEDENES

can't be DEBUGged
Der Tracer kann das zu tracende Wort nicht verarbeiten Evtl. ist es in Maschinencode geschrieben.

save-error
SAVESYSTEM gibt diese Meldung aus.

Task error
Eine Task hat **ABORT"** oder **ERROR"** ausgeführt, was normalerweise einen Systemabsturz versursacht.

trace dump is
Siehe Kap. 7.3 des Handbuches

INDEX der im Handbuch erklärten Forthworte

.blk	162	C64-Special	(--)
.name	91	Dictionary	(addr --)
.r	121	In/Output	(n +n --)
.rd	129	Kassetten	(--)
.s	82	Stack	(? -- ?)
.status	106	Sonstiges	(--)
/	73	Arithmetik	(n1 n2 -- n3)
/mod	73	Arithmetik	(n1 n2 -- n3 n4)
/string	85	Strings	(addr1 n1 n2 -- addr2 n3)
0	74	Arithmetik	(-- 0)
0<	76	Logik/Vergl.	(n -- flag)
0<>	76	Logik/Vergl.	(n -- flag)
0=	76	Logik/Vergl.	(w -- flag)
0>	76	Logik/Vergl.	(n -- flag)
1	74	Arithmetik	(-- 1)
1+	74	Arithmetik	(w1 -- w2)
1-	74	Arithmetik	(w1 -- w2)
1541r/w	111	C64-special	(addr block file n -- flag)
2	74	Arithmetik	(-- 2)
2!	163	32-Bit-Worte	(32b addr --)
2*	74	Arithmetik	(w1 -- w2)
2+	74	Arithmetik	(w1 -- w2)
2-	74	Arithmetik	(w1 -- w2)
2/	74	Arithmetik	(n1 -- n2)
2@	163	32-Bit-Worte	(addr -- 32b)
2Constant	163	32-Bit-Worte	(32b --) compiling
2Constant	163	32-Bit-Worte	(-- 32b)
2disk1551	133	MaSpUtil	(--)
2drop	82	Stack	(32b --)
2dup	82	Stack	(32b -- 32b 32b)
2swap	82	Stack	(32b1 32b2 -- 32b2 32b1)
2Variable	163	32-Bit-Worte	(--) compiling
2Variable	163	32-Bit-Worte	(-- addr)
3	74	Arithmetik	(-- 3)
3+	74	Arithmetik	(w1 -- w2)
4	74	Arithmetik	(-- 4)
7>c	129	Kassetten	(8b -- 7b)
:	88	Datentypen	(-- sys)
;	88	Datentypen	(--)
;	88	Datentypen	(sys --) compiling
<	76	Logik/Vergl.	(n1 n2 -- flag)
<#	85	Strings	(--)
=	76	Logik/Vergl.	(w1 w2 -- flag)
>	76	Logik/Vergl.	(n1 n2 -- flag)
>body	92	Dictionary	(addr1 -- addr2)
>drive	108	Massensp.	(block drv# -- block')
>in	101	Interpreter	(-- addr)
>interpret	101	Interpreter	(--)
>name	92	Dictionary	(addr1 -- addr2)
>r	84	Returnstack	(16b --)
>tib	122	In/Output	(-- addr)
?cr	122	In/Output	(--)
?device	111	C64-special	(dev# --)
?DO	96	Kontroll	(w1 w2 --)
?DO	96	Kontroll	(-- sys) compiling
?dup	82	Stack	(0 -- 0)
?dup	82	Stack	(16b -- 16b 16b)
?exit	96	Kontroll	(flag --)
?head	95	Heap	(-- addr)

?pairs	104	Fehler	(n1 n2 --)
?stack	104	Fehler	(--)
@	78	Speicher	(addr -- 16b)
[	100	Compiler	(--)
[	100	Compiler	(--) compiling
[']	100	Compiler	(--) compiling
[']	100	Compiler	(-- addr)
[compile]	100	Compiler	(--)
[compile]	100	Compiler	(--) compiling
\	103	Interpreter	(--)
\	103	Interpreter	(--) compiling
\\	103	Interpreter	(--)
\\	103	Interpreter	(--) compiling
\IF	129	Kassetten	(--)
\needs	103	Interpreter	(--)
]	103	Interpreter	(--)
]	103	Interpreter	(--) compiling
abort	104	Fehler	(--)
Abort"	104	Fehler	(--) compiling
Abort"	104	Fehler	(flag -- ?)
abs	74	Arithmetik	(n -- u)
accumulate	86	Strings	(+d1 addr char -- +d2 addr)
activate	117	Multitasking	(Taddr --)
Alias	88	Datentypen	(cfa --)
all-buffers	108	Massensp.	(--)
allot	91	Dictionary	(w --)
allotbuffer	108	Massensp.	(--)
also	93	Vocabulary	(--)
and	76	Logik/Vergl.	(n1 n2 -- n3)
Ascii	99	Compiler	(--) compiling
Ascii	99	Compiler	(-- char)
Assembler	93	Vocabulary	(--)
at	122	In/Output	(row col --)
at?	122	In/Output	(-- row col)
autoload	129	Kassetten	(-- addr)
B	62	Dekompiler	(addr --)
b/blk	108	Massensp.	(-- &1024)
b/buf	108	Massensp.	(-- n)
bamallocate	133	MaSpUtil	(--)
base	122	In/Output	(-- addr)
BEGIN	96	Kontroll	(--)
BEGIN	96	Kontroll	(sys --) compiling
binary	129	Kassetten	(u -- u)
bl	122	In/Output	(-- n)
blk	101	Interpreter	(-- addr)
blk/drv	108	Massensp.	(-- n)
bload	129	Kassetten	(addr1 addr3 8b -- addr2)
block	108	Massensp.	(u -- addr)
bounds	96	Kontroll	(start count -- limit start)
bsave	129	Kassetten	(a1 a2 a3 8b --)
buffer	108	Massensp.	(u -- addr)
bus!	112	C64-special	(8b --)
bus@	112	C64-special	(-- 8b)
busclose	112	C64-special	(dev# 2nd --)
busin	112	C64-special	(dev# 2nd --)
businput	112	C64-special	(addr u --)
busoff	112	C64-special	(--)

busopen	112	C64-special	(dev# 2nd --)
busout	112	C64-special	(dev# 2nd --)
bustype	113	C64-special	(addr u --)
bye	106	Sonstiges	(--)
C	62	Dekompiler	(addr --)
c!	78	Speicher	(16b addr --)
C)	127	C64-special	(--)
C)	127	C64-special	(--) compiling
c,	91	Dictionary	(16b --)
c/l	122	In/Output	(-- +n)
c64at	113	C64-special	(row col --)
c64at?	113	C64-special	(-- row col)
c64cr	113	C64-special	(--)
c64decode	113	C64-special	(addr len0 key -- addr len1)
c64del	113	C64-special	(--)
c64emit	113	C64-special	(8b --)
c64expect	113	C64-special	(addr len --)
c64fkeys	127	C64-special	(--)
c64init	114	C64-special	(--)
c64key	114	C64-special	(-- 8b)
c64key?	114	C64-special	(-- flag)
c64page	114	C64-special	(--)
c64type	114	C64-special	(addr len --)
c>7	129	Kassetten	(8b -- 7b)
c@	78	Speicher	(addr -- 8b)
capital	86	Strings	(char1 -- char2)
capitalize	86	Strings	(addr -- addr)
case?	76	Logik/Vergl.	(16b1 16b2 -- 16b1 false)
case?	76	Logik/Vergl.	(16b 16b -- true)
cbm>scr	162	C64-Special	(--)
clear	91	Dictionary	(--)
clearstack	82	Stack	(? -- empty)
cload	130	Kassetten	(addr1 addr3 8b u1 -- addr2 u2)
cmove	78	Speicher	(from to u --)
cmove>	78	Speicher	(from to u --)
col	122	In/Output	(-- u)
cold	106	Sonstiges	(--)
commodore	130	Kassetten	(--)
compess	130	Kassetten	(addr1 addr3 u1 -- u2)
compile	99	Compiler	(--)
con!	114	C64-special	(8b --)
Constant	88	Datentypen	(16b --)
context	93	Vocabulary	(-- addr)
convert	86	Strings	(+d1 addr1 -- +d2 addr2)
convey	109	Massensp.	(blk1 blk2 to.blk --)
copy	109	Massensp.	(u1 u2 --)
copy2disk	133	MaSpUtil	(--)
copydisk	133	MaSpUtil	(u1 u2 u3 --)
core?	109	Massensp.	(blk file -- addr)
core?	109	Massensp.	(blk file -- false)
count	78	Speicher	(addr -- addr+1 len)
cpush	128	Tools	(addr u --)
cr	122	In/Output	(--)
Create	126	Datentypen	(--) compiling
Create	126	Datentypen	(-- pfa)
Create:	126	Datentypen	(--) compiling
Create:	126	Datentypen	(-- pfa)
csave	130	Kassetten	(a1 a2 a3 8b u1 -- u2)

ctoggle	78	Speicher	(8b addr --)
curoff	114	C64-special	(--)
curon	114	C64-special	(--)
current	93	Vocabulary	(-- addr)
D	62	Dekompiler	(addr --)
d+	80	32-Bit-Worte	(d1 d2 -- d3)
d.	122	In/Output	(d --)
d.r	122	In/Output	(d +n --)
d0=	80	32-Bit-Worte	(d -- flag)
d<	80	32-Bit-Worte	(d1 d2 -- flag)
dabs	80	32-Bit-Worte	(d -- ud)
debug	128	Tools	(--)
decimal	123	In/Output	(--)
decode	123	In/Output	(addr +n0 key -- addr +n1)
Defer	89	Datentypen	(--)
definitions	93	Vocabulary	(--)
del	123	In/Output	(--)
depth	82	Stack	(-- n)
derr?	130	Kassetten	(u -- flag)
derror?	114	C64-special	(-- flag)
device	130	Kassetten	(-- addr)
digidecode	159	Editor	(adr len0 key -- adr len1)
digit?	86	Strings	(char -- digit true)
digit?	86	Strings	(char -- false)
digits	159	Editor	(--)
diskclose	115	C64-special	(--)
diskerr	104	Fehler	(--)
diskopen	115	C64-special	(-- flag)
display	115	C64-special	(--)
dnegate	80	32-Bit-Worte	(d1 -- d2)
DO	96	Kontroll	(w1 w2 --)
DO	96	Kontroll	(-- sys) compiling
Does>	99	Compiler	(--) compiling
Does>	99	Compiler	(-- addr)
dp	91	Dictionary	(-- addr)
drive	109	Massensp.	(drv# --)
drop	82	Stack	(16b --)
drv?	109	Massensp.	(block -- drv#)
dup	83	Stack	(16b -- 16b 16b)
ediboard	159	Editor	(--)
edidecode	159	Editor	(adr len0 key -- adr len1)
ediexpect	159	Editor	(adr len --)
edit	151	Editor	(n --)
Editor	159	Editor	(--)
ELSE	96	Kontroll	(--)
ELSE	96	Kontroll	(sys1 -- sys2) compiling
emit	123	In/Output	(16b --)
empty	91	Dictionary	(--)
empty-buffers	109	Massensp.	(--)
end-trace	107	Sonstiges	(--)
endloop	128	Tools	(--)
erase	79	Speicher	(adr u --)
Error"	105	Fehler	(flag --)
Error"	105	Fehler	(--) compiling
errorhandler	105	Fehler	(-- addr)
execute	97	Kontroll	(addr --)
exit	126	Kontroll	(--)

expand	130	Kassetten	(addr1 addr2 u1 -- u2)
expect	123	In/Output	(addr +n --)
extend	80	32-Bit-Worte	(n -- d)
false	76	Logik/Vergl.	(-- 0)
file	109	Massensp.	(-- addr)
fill	79	Speicher	(adr u 8b --)
find	101	Interpreter	(addr1 -- addr2 n)
findex	115	C64-special	(from to --)
first	109	Massensp.	(-- addr)
floppy	130	Kassetten	(--)
flush	109	Massensp.	(--)
forget	91	Dictionary	(--)
formatdisk	133	MaSpUtil	(--)
Forth	93	Vocabulary	(--)
forth-83	93	Vocabulary	(--)
FORTH-Gesellsc	162	C64-Special	(8b0 -- 8b1)
freebuffer	109	Massensp.	(--)
getkey	115	C64-special	(-- 8b)
getstamp	160	Editor	(--)
hallot	95	Heap	(n --)
heap	95	Heap	(-- addr)
heap?	95	Heap	(addr -- flag)
here	92	Dictionary	(-- addr)
hex	123	In/Output	(--)
hide	92	Dictionary	(--)
hold	86	Strings	(char --)
I	97	Kontroll	(-- w)
i/o	115	C64-special	(-- semaphoraddr)
id"	131	Kassetten	(--)
IF	97	Kontroll	(flag --)
IF	97	Kontroll	(-- sys) compiling
immediate	99	Compiler	(--)
index	115	C64-special	(from to --)
ink-pot	116	C64-special	(-- addr)
input	123	In/Output	(-- addr)
Input:	89	Datentypen	(--)
interpret	102	Interpreter	(--)
IP	179	Assembler	(-- addr)
Is	89	Datentypen	(cfa --)
J	97	Kontroll	(-- w)
K	62	Dekompiler	(addr --)
key	123	In/Output	(-- 16b)
key?	124	In/Output	(-- flag)
keyboard	116	C64-special	(--)
l	151	Editor	(n --)
l/s	124	In/Output	(-- +n)
last	92	Dictionary	(-- addr)
LEAVE	97	Kontroll	(--)
limit	110	Massensp.	(-- addr)
list	124	In/Output	(u --)
Literal	99	Compiler	(16b --) compiling
Literal	99	Compiler	(-- 16b)

load	102	Interpreter	(n --)
loadramdisk	131	Kassetten	(--)
lock	117	Multitasking	(semaddr --)
LOOP	97	Kontroll	(--)
LOOP	97	Kontroll	(sys --) compiling
m*	80	32-Bit-Worte	(n1 n2 -- d)
m/mod	80	32-Bit-Worte	(d n1 -- n2 n3)
max	74	Arithmetik	(n1 n2 -- n3)
memtop	131	Kassetten	(-- adr)
min	75	Arithmetik	(n1 n2 -- n3)
mod	75	Arithmetik	(n1 n2 -- n3)
move	79	Speicher	(addr1 addr2 u --)
multitask	117	Multitasking	(--)
N	62	Dekompiler	(addr --)
N	179	Assembler	(-- addr)
n"	131	Kassetten	(-- addr 8b)
name	102	Interpreter	(-- addr)
name>	92	Dictionary	(addr1 -- addr2)
negate	75	Arithmetik	(n1 -- n2)
nest	128	Tools	(--)
Next	178	Assembler	(-- addr)
nip	83	Stack	(16b1 16b2 -- 16b2)
noop	107	Sonstiges	(--)
not	76	Logik/Vergl.	(n1 -- n2)
notfound	102	Interpreter	(addr --)
nullstring?	86	Strings	(addr -- addr false)
nullstring?	86	Strings	(addr -- true)
number	86	Strings	(addr -- d)
number?	87	Strings	(addr -- addr false)
number?	87	Strings	(addr -- d 0>)
number?	87	Strings	(addr -- n 0<)
off	79	Speicher	(addr --)
offset	110	Massensp.	(-- addr)
on	79	Speicher	(addr --)
Only	93	Vocabulary	(--)
Onlyforth	93	Vocabulary	(--)
or	76	Logik/Vergl.	(n1 n2 -- n3)
order	126	Vokabular	(--)
origin	92	Dictionary	(-- addr)
output	124	In/Output	(-- addr)
Output:	90	Datentypen	(--)
over	83	Stack	(16b1 16b2 -- 16b1 16b2 16b1)
pad	79	Speicher	(-- addr)
page	124	In/Output	(--)
parse	102	Interpreter	(char -- addr +n)
pass	117	Multitasking	(n0..nr-1 Taddr r --)
pause	118	Multitasking	(--)
perform	97	Kontroll	(addr --)
pick	83	Stack	(16bn..16b0 +n -- 16bn..16b0 16bn)
place	79	Speicher	(addr1 +n addr2 --)
Pop	178	Assembler	(-- addr)
Poptwo	178	Assembler	(-- addr)
prev	110	Massensp.	(-- addr)
printable?	116	C64-special	(8b -- 8b flag)
push	84	Returnstack	(addr --)

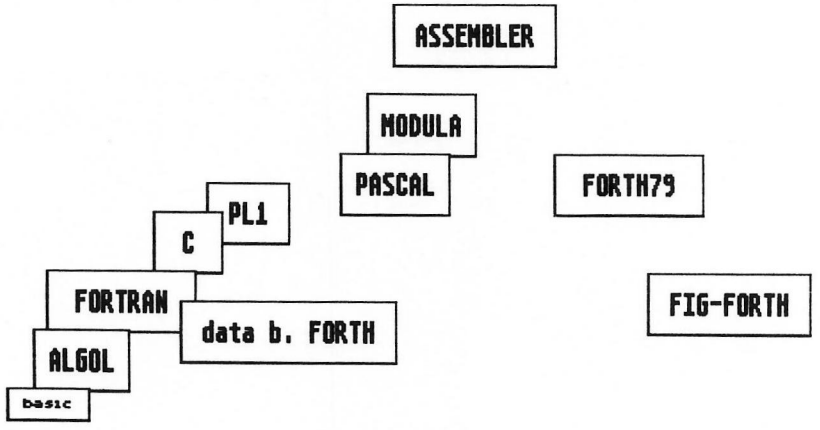
```
Push            178    Assembler                    ( -- addr )
PushOA          178    Assembler                    ( -- addr )
PushA           178    Assembler                    ( -- addr )
Puta            178    Assembler                    ( -- addr )

query           124     In/Output                   ( -- )
quit            102   Interpreter                   ( -- )

r               151       Editor                    ( -- )
r#              107     Sonstiges                   ( -- )
r/w             110     Massensp. ( addr block file n -- flag )
r0               84   Returnstack                   ( -- addr )
r>               84   Returnstack                   ( -- 16b )
r@               84   Returnstack                   ( -- 16b )
ram             179     Assembler                   ( -- )
ramdisk         131     Kassetten                   ( -- )
ramR/W          131     Kassetten( addr1 u addr2 flag -- flag )
rd              131     Kassetten                   ( -- addr )
rdcheck         131     Kassetten                   ( -- )
rddel           131     Kassetten                   ( -- )
rdepth           84   Returnstack                   ( -- n )
rdnew           131     Kassetten       ( addr1 addr2 -- )
rdrop            84   Returnstack                   ( -- )
rduse           131     Kassetten             ( addr -- )
readsector      116   C64-special    ( addr tra# sec# -- flag )
recursive       100      Compiler                   ( -- )
recursive       100      Compiler                   ( -- ) compiling
rendezvous      118 Multitasking           ( semaddr -- )
REPEAT           97      Kontroll                   ( -- )
REPEAT           97      Kontroll               ( sys -- ) compiling
restart         107     Sonstiges                   ( -- )
restore"        131     Kassetten                   ( -- )
restrict        100      Compiler                   ( -- )
reveal           92    Dictionary                   ( -- )
roll             83         Stack ( 16bn 16bm..16b0 + -- 16bm..16b0 16bn )
rom             179     Assembler                   ( -- )
rot              83         Stack    ( 16b1 16b2 16b3 -- 16b2 16b3 16b1 )
row             124     In/Output                   ( -- n )
RP              179     Assembler                   ( -- addr )
rp!              84   Returnstack             ( addr -- )
rp@              84   Returnstack                   ( -- addr )
rvsoff          162   C64-Special                   ( -- )
rvson           162   C64-Special                   ( -- )

S                62    Dekompiler             ( addr -- )
s0               83         Stack                  ( -- addr )
save             92    Dictionary                   ( -- )
save-buffers    110     Massensp.                   ( -- )
saveramdisk     132     Kassetten                   ( -- )
savesystem      133      MaSpUtil                   ( -- )
scan             87       Strings     ( addr1 n1 char -- addr2 n2 )
scr             107     Sonstiges                   ( -- addr )
scr>cbm         162   C64-Special               ( 8b0 -- 8b1 )
seal             93    Vocabulary                   ( -- )
setup           179     Assembler                   ( -- addr )
shadow          160        Editor                   ( -- adr )
showload        161       Editor           ( blk +n -- )
sign             87       Strings               ( n -- )
singletask      118 Multitasking                   ( -- )
```

skip	87	Strings	(addr1 n1 char -- addr2 n2)
sleep	118	Multitasking	(Taddr --)
source	102	Interpreter	(-- addr +n)
SP	179	Assembler	(-- addr)
sp!	83	Stack	(addr --)
sp@	83	Stack	(-- addr)
space	124	In/Output	(--)
spaces	124	In/Output	(+n --)
span	125	In/Output	(-- addr)
stamp$	160	Editor	(-- adr)
standardi/o	125	In/Output	(--)
state	102	Interpreter	(-- addr)
stop	118	Multitasking	(--)
stop?	125	In/Output	(-- flag)
store	132	Kassetten	(addr --)
supertape	132	Kassetten	(--)
swap	83	Stack	(16b1 16b2 -- 16b2 16b1)
sys	179	Assembler	(addr --)
tapeinit	132	Kassetten	(--)
Task	119	Multitasking	(rlen slen --)
tasks	119	Multitasking	(--)
THEN	97	Kontroll	(--)
THEN	97	Kontroll	(sys --) compiling
thru	103	Interpreter	(n1 n2 --)
tib	125	In/Output	(-- addr)
toss	94	Vocabulary	(--)
trace'	128	Tools	(--)
true	77	Logik/Vergl.	(-- -1)
type	125	In/Output	(addr +n --)
u.	125	In/Output	(u --)
u.r	125	In/Output	(u +n --)
u/mod	75	Arithmetik	(u1 u2 -- u3 u4)
u<	77	Logik/Vergl.	(u1 u2 -- flag)
u>	77	Logik/Vergl.	(u1 u2 -- flag)
uallot	92	Dictionary	(n1 -- n2)
ud/mod	80	32-Bit-Worte	(ud1 u1 -- u2 ud2)
udp	92	Dictionary	(-- addr)
um*	80	32-Bit-Worte	(u1 u2 -- ud)
um/mod	81	32-Bit-Worte	(ud u1 -- u2 u3)
umax	75	Arithmetik	(u1 u2 -- u3)
umin	75	Arithmetik	(u1 u2 -- u3)
unbug	128	Tools	(--)
under	83	Stack	(16b1 16b2 -- 16b2 16b1 16b2)
unlink	162	C64-Special	(--)
unlock	119	Multitasking	(semaddr --)
unnest	128	Tools	(--)
UNTIL	98	Kontroll	(flag --)
UNTIL	98	Kontroll	(sys --) compiling
UP	179	Assembler	(-- addr)
up!	120	Multitasking	(addr --)
up@	119	Multitasking	(-- Taddr)
update	110	Massensp.	(--)
User	90	Datentypen	(--)
uwithin	77	Logik/Vergl.	(n u1 u2 -- flag)
v	160	Editor	(-- +n)
Variable	90	Datentypen	(--)

view	151	Editor	(--)	
voc-link	94	Vocabulary	(-- addr)	
Vocabulary	90	Datentypen	(--)	
vp	94	Vocabulary	(-- addr)	
wake	120	Multitasking	(Taddr --)	
warning	105	Fehler	(-- addr)	
wcmp	179	Assembler	(addr1 addr2 --)	
WHILE	98	Kontroll	(flag --)	
WHILE	98	Kontroll	(sys1 -- sys2)	compiling
word	103	Interpreter	(char -- addr)	
words	94	Vocabulary	(--)	
writesector	116	C64-special	(addr tra# sec# -- flag)	
xor	77	Logik/Vergl.	(n1 n2 -- n3)	
xyNext	178	Assembler	(-- addr)	
!	95	Heap	(--)	

Index Graphikworte für C64

reset	171	Sprites	(3b adr --)
right	173	Turtle	(deg --)
rt	173	Turtle	(deg --)
screen	169	Graphik	(color --)
set	171	Sprites	(3b adr --)
setbit	171	Sprites	(3b adr flag --)
seth	173	Turtle	(deg --)
setheading	173	Turtle	(deg --)
setsprite	171	Sprites	(mem# y x color spr --)
setx	174	Turtle	(x --)
setxy	174	Turtle	(x y --)
sety	174	Turtle	(y --)
slim	172	Sprites	(spr# --)
small	172	Sprites	(spr# --)
sprcolors	172	Sprites	(color color --)
sprite	172	Sprites	(-- adr)
text	168	Graphik	(--)
ts	174	Turtle	(-- pen bg pc)
turtlestate	174	Turtle	(-- pen bg pc)
unplot	170	Graphik	(x y --)
wht	169	Graphik	(-- color)
wide	172	Sprites	(spr# --)
window	168	Graphik	(n --)
xcor	174	Turtle	(-- x)
xmove	171	Sprites	(x spr# --)
ycor	174	Turtle	(-- y)
yel	169	Graphik	(-- color)
ymove	171	Sprites	(y spr# --)


ASSEMBLER
MODULA
PASCAL
FORTH79
PL1
C
FORTRAN
data b. FORTH
FIG-FORTH
ALGOL
basic