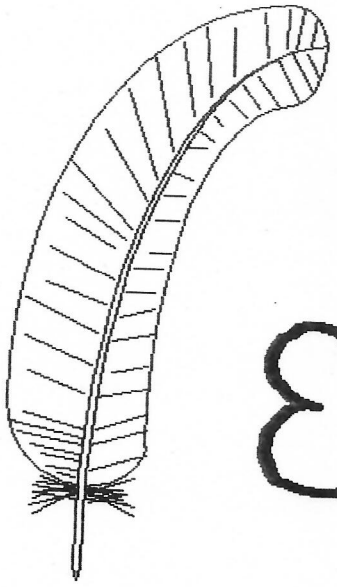
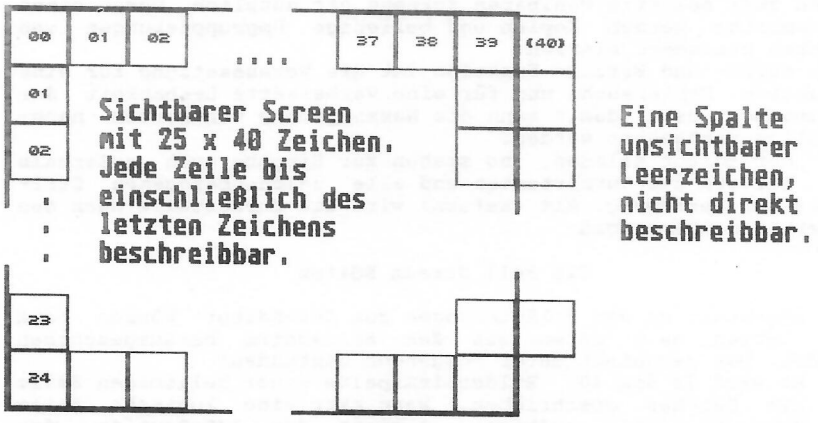


Der Editor

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.

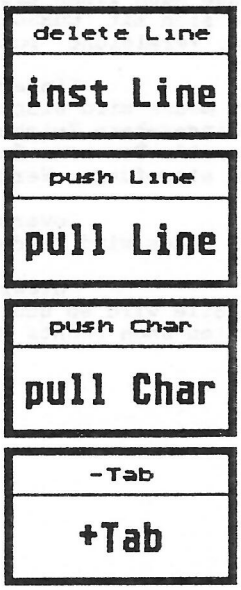
CTRL r "clear-to-right"

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.

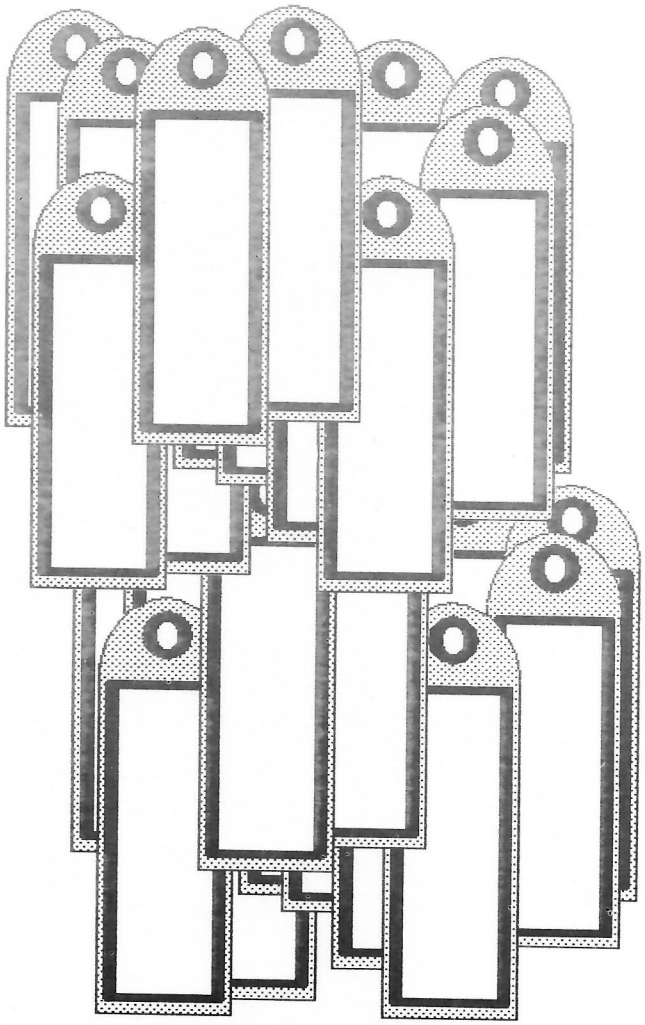
32-Bit Worte

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 x0 y0 -)

zeichnet eine Gerade von (x0/y0) 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: 10 10 150 10 line
150 150 drawto
10 150 drawto
10 10 drawto
zeichnet ein Quadrat.

flipline (x1 y1 x0 y0 -)

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

Wort	S.	Gruppe	Stack vorher	Stack nachher
!	78	Speicher	(16b addr	--)
"	85	Strings		(--) compiling
"	85	Strings		(-- addr)
#	85	Strings		(-- addr)
#>	85	Strings	(32b	-- addr +n)
#bs	121	In/Output		(-- n)
#cr	121	In/Output		(-- n)
#s	85	Strings	(+d	-- 0 0)
#tib	121	In/Output		(-- addr)
'	91	Dictionary		(-- addr)
abort	106	Sonstiges		(--)
'cold	106	Sonstiges		(--)
'quit	106	Sonstiges		(--)
'restart	106	Sonstiges		(--)
's	117	Multitasking	(Taddr	-- usraddr)
(	101	Interpreter		(--)
(	101	Interpreter		(--) compiling
(16	127	C64-special		(--)
(16	127	C64-special		(--) compiling
(64	127	C64-special		(--)
(64	127	C64-special		(--) compiling
(drv	127	C64-special		(-- addr)
(error	104	Fehler	(string	--)
(forget	91	Dictionary	(addr	--)
(load	161	Editor	(blk +n	--)
(pad	160	Editor		(-- adr)
(quit	106	Sonstiges		(--)
(rd	129	Kassetten		(-- addr)
(search	160	Editor	(text tlen buf blen	-- adr tf // ff)
*	73	Arithmetik	(w1 w2	-- w3)
***ultraFORTH8	162	C64-Special		(--)
*/	73	Arithmetik	(n1 n2 n3	-- n4)
*/mod	73	Arithmetik	(n1 n2 n3	-- n4 n5)
+	73	Arithmetik	(w1 w2	-- w3)
+!	78	Speicher	(w1 adr	--)
+l	151	Editor	(n	--)
+load	101	Interpreter	(n	--)
+LOOP	96	Kontroll	(n	--)
+LOOP	96	Kontroll	(sys	--) compiling
+thru	101	Interpreter	(n1 n2	--)
,	91	Dictionary	(16b	--)
,"	99	Compiler		(--)
-	73	Arithmetik	(w1 w2	-- w3)
-->	101	Interpreter		(--)
-->	101	Interpreter		(--) compiling
-1	73	Arithmetik		(-- -1)
-roll	82	Stack	(16bn..16b1 16b0 +n	-- 16b0 16bn..16b1)
-rot	82	Stack	(16b1 16b2 16b3	-- 16b3 16b1 16b2)
-trailing	121	In/Output	(addr +n0	-- addr +n1)
.	121	In/Output	(n	--)
."	121	In/Output		(--)
."	121	In/Output		(--) compiling
.(	121	In/Output		(--)
.(	121	In/Output		(--) compiling

.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 --)

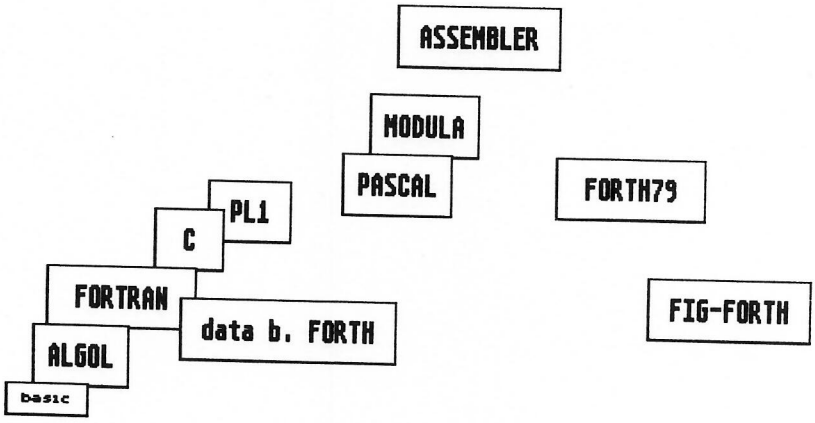
Push	178	Assembler	(-- addr)
PushOA	178	Assembler	(-- addr)
PushA	178	Assembler	(-- addr)
Puta	178	Assembler	(-- addr)
query	124	In/Output	(--)
quit	102	Interpreter	(--)
r	151	Editor	(--)
r#	107	Sonstiges	(--)
r/w	110	Massensp.	(addr block file n -- flag)
r0	84	Returnstack	(-- addr)
r>	84	Returnstack	(-- 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