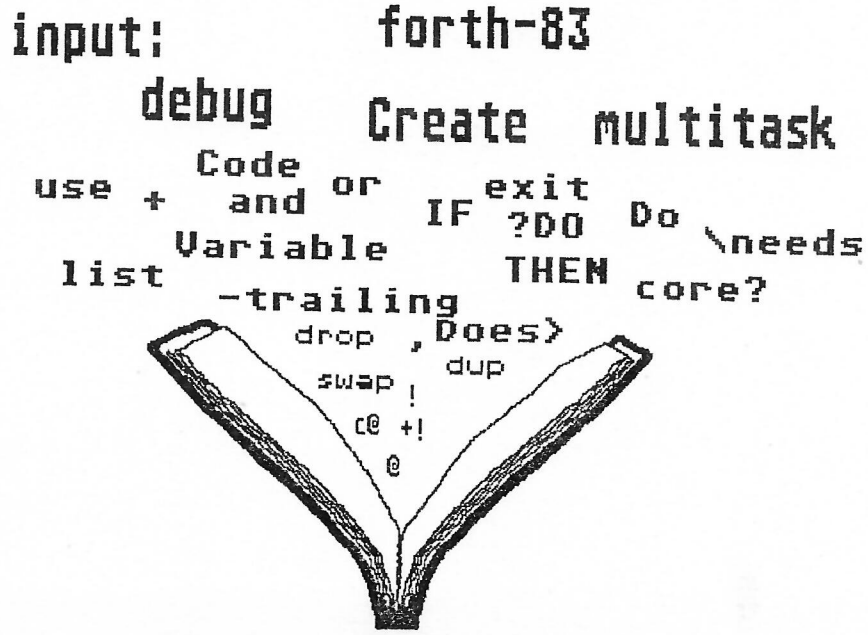
Glossar


input:
forth-83
debug
Create
multitask
Code
use
+
and
or
exit
IF
?DO
Do
\needs
Variable
list
THEN
core?
-trailing
drop
,
Does>
dup
swap
!
c@
+!
@

Notation

Die Worte des volksFORTH83 sind in Wortgruppen zusammengefasst. Die Worte jeder Wortgruppe sind alphabetisch sortiert. Die Wortgruppen sind nicht geordnet.

Worte werden in der in Kapitel 1 angegeben Schreibweise aufgeführt. 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 Ausführung des Wortes
nachher : Werte auf dem Stack nach Ausführung 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 spätere Ausführung des Wortes. Bei immediate Worten wird auch die Auswirkung des Wortes auf den Stack während der Kompilierung angegeben.

Worte werden ferner durch folgende Symbole gekennzeichnet:

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

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

83
: Dieses Wort wird im 83-Standard definiert und muß auf allen Standardsystemen äquivalent funktionieren.

U
: Kennzeichnet eine Uservariable.

Weicht die Aussprache eines Wortes von der natürlichen englischen Aussprache ab, so wird sie in Anführungszeichen angegeben. Gelegentlich folgt auch eine deutsche Übersetzung.

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

Stack-not.	Zahlentyp	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-abhängige Werte	na	nicht anwendbar

Arithmetik

***** (w1 w2 -- w2) 83 " times"
Der Wert w1 wird mit w2 multipliziert. w2 sind die niederwertigen 16 Bits des Produktes. Ein Überlauf wird nicht angezeigt.

***/** (n1 n2 n3 -- n4) 83 " 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 Divisior n3. Das Produkt von n1 mal n2 wird als 32-bit Zwischenergebnis dargestellt, um eine größere Genauigkeit gegenüber 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 " 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 größere Genauigkeit gegenüber dem sonst gleichwertigen Ausdruck n1 n2 * n3 /mod zu erhalten. Eine Fehlerbedingung besteht, falls der Divisior 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 Ausführungszeit verkürzt. Siehe auch CONSTANT.

/ (n1 n2 -- n3) 83 " 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 " divide-mod "
n3 ist der Rest und n4 der Quotient aus 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 Intervalls (-32768..32767) liegt.

0 (-- 0)
Siehe -1.

1 (-- 1)
Siehe -1.

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

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

2 (-- 2)
Siehe -1.

2* (w1 -- w2) " two-times "
w1 wird um ein Bit nach links geschoben und das ergibt w2. In das niederwertigste Bit wird eine Null geschrieben. Die Operation 2 * wirkt genauso.

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

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

2/ (n1 -- n2) 83 " two-divide "
n1 wird um ein Bit nach rechts verschoben und das ergibt n2. Das Vorzeichen wird berücksichtigt und bleibt unverändert. Die Operation 2 / wirkt genauso.

3 (-- 3)
Siehe -1.

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

4 (-- 4)
Siehe -1.

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

even (u1 -- u2)
u2 ist die nächstgrößere gerade Zahl zu u1.

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

min (n1 n2 -- n3) 83 " minimum "
n3 ist die Kleinere der beiden Werte n1 und n2. Benutzt die < Operation. Die kleinste Zahl für 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 Intervalls (-32768..32767) liegt.

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

u/mod (u1 u2 -- u3 u4) " 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 Werte (unsigned integer). Eine Fehlerbedingung besteht, wenn der Divisor Null ist.

umax (u1 u2 -- u3) " u-maximum "
u3 ist der Größere der beiden Werte u1 und u2. Benutzt die U> Operation. Die größte Zahl für u1 oder u2 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 für u1 oder u2 ist Null.

Logik und Vergleiche

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

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

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

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

< (n1 n2 -- flag) 83 " 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 " equals "
Wenn w1 gleich w2 ist, ist flag wahr.

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

and (w1 w2 -- w3) 83
w1 wird mit w2 bitweise logisch UND verknüpft und das ergibt w3.

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

```
key Ascii a case? IF ... exit THEN
    Ascii b case? IF ... exit THEN
      ..
drop
```

Entspricht dem Ausdruck OVER = DUP IF NIP THEN .

false (-- 0)
Hinterläßt Null als Zeichen für logisch-falsch auf dem Stack.

not (w1 -- w2) 83
Jedes Bit von w1 wird einzeln invertiert und das ergibt w2.

or (w1 w2 -- w3) 83
w1 wird mit w2 logisch ODER verknüpft und das ergibt w3.

true (-- -1)
Hinterläßt -1 als Zeichen für logisch wahr auf dem Stack.

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

U> (u1 u2 -- flag) 83 " u-greater-than "
Wenn u1 größer als u2 ist, ist flag wahr. Ansonsten gilt das gleiche wie für U< .

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

xor (w1 w2 -- w3) 83 " x-or "
w1 wird mit w2 bitweise logisch EXKLUSIV ODER verknüpft und das ergibt w3.

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 überschrieben.

+! (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 überschrieben.

2! (32b adr --) 83 " two-fetch "
32b werden in den Speicher ab Adresse adr geschrieben.

2@ (adr -- 32b) 83 " two-fetch "
Von der Adresse adr wird der Wert 32b aus dem Speicher geholt.

@ (adr -- 16b) 83 " fetch "
Von der Adresse adr wird der Wert 16b aus dem Speicher geholt. Siehe auch ! .

c! (16b adr --) 83 " c-store "
Von 16b werden die niederwertigsten 8 Bit 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 Adresse 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-up "
Beginnend bei adr1 werden u Bytes zur Adresse adr2 kopiert. Zuerst wird das Byte von adr1-plus-u-minus-1 nach adr2-plus-u-minus-1 kopiert und dann absteigend fortgefahren. Wenn u Null ist, wird nichts kopiert. Das Wort wird benutzt, um Speicherinhalte auf höhere Adressen zu verschieben, wenn die Speicherbereiche sich überlappen.

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

ctoggle (8b adr --) 83 " c-toggle "
Für jedes gesetzte Bit in 8b wird im Byte mit der Adresse adr das entsprechende Bit invertiert (d.h. ein zuvor gesetztes Bit ist danach gelöscht und ein gelösch-

tes Bit ist danach gesetzt). Für alle gelöschten Bits in 8b bleiben die entsprechenden Bits im Byte mit der Adresse adr unverändert. Der Ausdruck DUP C@ ROT XOR SWAP C! wirkt genauso.

erase (adr u --)
Von der Adresse adr an werden u Bytes im Speicher mit $00 überschrieben. Hat u den Wert Null, passiert nichts.

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

move (adr1 adr2 u --)
Beginnend bei adr1 werden u Bytes nach adr2 kopiert. Dabei ist es ohne Bedeutung, ob überlappende Speicherbereiche aufwärts oder abwärts kopiert werden, weil MOVE die passende Routine datzu auswählt. Hat u den Wert Null, passiert nichts. Siehe auch CMOVE und CMOVE>.

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

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

pad (-- adr) 83
adr ist die Startadresse einer "scratch area". In diesem Speicherbereich können Daten für Zwischenrechnungen abgelegt werden. Wenn die nächste verfügbare Stelle für das Dictionary verändert wird, ändert sich auch die Startadresse von PAD. Die vorherige Startadresse von PAD geht ebenso wie die Daten dort verloren.

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 Länge als "counted string" abzuspeichern. adr2 darf gleich, größer und auch kleiner als adr1 sein.

32-Bit-Worte

d* (d1 d2 -- d3) " d-times "
d1 multipliziert mit d2 ergibt d3.

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

d- (d1 d2 -- d3) " d-minus"
d2 minus d1 ergibt d3.

d0= (d -- flag) 83 " d-zero-equals "
Wenn d gleich Null ist, ist flag wahr.

d< (d1 d2 -- flag) 83 " d-less-than "
Wenn d1 kleiner als d2 ist, ist flag wahr.

d= (d1 d2 -- flag) " d-equal"
Wenn d1 gleich d2 ist, ist flag wahr.

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-times "
Der Wert von n1 wird mit n2 multiplizert und d ist das doppelt genaue Produkt.

m/mod (d n1 -- n2 n3) " m-divide-mod "
n2 ist der Rest und n3 der Quotient aus der Division der doppelt genauen Zahl d durch den Divisor n1. Der Rest n2 hat dasselbe Vorzeichen wie n1 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-divide-mod "
u2 ist der Rest und ud2 der doppelt genaue Quotient aus der Divison der doppelt genauen Zahl ud1 durch den Divisor u1. Die Zahlen u sind vorzeichenlose 16-Bit Werte (unsigned integer). Eine Fehlerbedingung besteht, wenn der Divisor Null ist.

um* (u1 u2 -- ud) 83 " u-m-times "
Die Werte u1 und u2 werden mulitpliziert und das ergibt das doppelt genaue Produkt ud. UM* ist die anderen multiplizierenden Worten zugrundeliegende Routine.

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

Stack

-roll (16bn .. 16b1 16b0 +n -- 16b0 16bn .. 16b1)
" minus-roll "
Das oberste Glied eier Kette von +n Werten wird an die n-te Position gerollt. Dabei wird +n selbst nicht mitgezählt. 2 -ROLL wirkt wie -ROT. 0 -ROLL verändert 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 (--) " dot-s "
Gibt alle Werte, die auf dem Stack liegen aus, ohne den Stack zu verändern. Oft benutzt, um neue Worte auszutesten. Die Ausgabe der Werte erfolgt von links nach rechts, der oberste Stackwert zuerst !

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

2drop (32b --) 83 " two-drop "
Der Wert 32b wird vom Stack entfernt.

2over (32b1 32b2 -- 32b1 32b2 31b1) " two-over "
Der Wert 32b1 wird über den Wert 32b2 herüber kopiert.

2swap (32b1 32b2 -- 32b2 32b1) 83 " two-swap "
Die beiden obersten 32-Bit Werte 32b1 und 32b2 werden vertauscht.

?dup (16b -- 16b 16b) 83 " question-dup "
oder (0 -- 0)
Nur wenn der Wert 16b von Null verschieden ist, wird er verdoppelt.

clearstack (-- empty)
Löscht den Datenstack. Alle Werte, die sich vorher auf dem Stack befanden, sind verloren.

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

drop (16b --) 83
Der Wert 16b wird vom Stack entfernt.

dup (16b -- 16b 16b) 83
Der Wert 16b wird dupliziert.

nip (16b1 16b2 -- 16b2)
Der Wert 16b1, der unter 16b2 auf dem Stack liegt, wird vom Stack entfernt.

over (16b1 16b2 -- 16b1 16b2 16b1) 83
Der Wert 16b1 wird über 16b2 herüberkopiert.

pick (16bn..16b0 +n -- 16bn..16b0 16bn) 83
Der +n-te Wert auf dem Stack wird nach oben auf den Stack kopiert. Dabei wird +n selbst nicht mitgezählt. 0 PICK wirkt wie DUP, 1 PICK wie OVER.

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

rot (16b1 16b2 16b3 -- 16b2 16b3 16b1) 83
Die drei obersten Werte auf dem Stack werden rotiert, sodaß der unterste zum obersten wird.

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

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

sp! (adr --) " s-p-store "
Setzt den Stackzeiger (stack pointer) auf die Adresse adr. Der oberste Wert auf dem Stack ist dann der, welcher in der Adresse adr steht.

sp@ (-- adr) " s-p-fetch "
Holt die Adresse adr aus dem Stackzeiger. Der oberste Wert im Stack stand in der Speicherstelle bei adr, bevor SP@ ausgeführt wurde.

under (16b1 16b2 -- 16b2 16b1 16b2)
Eine Kopie des obersten Wertes auf dem Stack wird unter dem zweiten Wert eingefügt.

Returnstack

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

push (adr --)
Der Inhalt aus der Adresse adr wird auf den Returnstack bis zum nächsten EXIT oder ; verwahrt und sodann nach adr zurück geschrieben. Dies ermöglicht die lokale Verwendung von Variablen innerhalb einer :-Definition. Wird z.B. benutzt in der Form :

```
: hex.  ( n -- ) base push hex . ;
```

Hier wird innerhalb von HEX. in der Zahlenbasis HEX gearbeitet, um eine Zahl auszugeben. Nachdem HEX. ausgeführt worden ist, besteht die gleiche Zahlenbasis wie vorher, durch HEX wird sie also nur innerhalb von HEX. verändert.

r> (-- 16b) C,83 " r-from "
Der Wert 16b wird vom Returnstack geholt. Vergleiche R>.

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

r0 (-- adr) U " r-zero "
adr ist die Adresse einer Uservariablen, in der die Startadresse des Returnstacks steht.

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

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

rdrop (--) C " r-drop "
Der oberste Wert wird vom Returnstack entfernt. Der Datenstack wird nicht verändert. Entspricht der Operation R> DROP .

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

Strings

Siehe auch im Anhang : "Strings"

" (-- adr) C,I " string "
(--) (compiling)
Liest den Text bis zum nächsten " und legt ihn als counted string im Dictionary ab. Kann nur während der Kompilation verwendet werden. Zur Laufzeit wird die Startadresse des counted string auf den Stack gelegt. Wird in der folgenden Form benutzt :
: <name> ... " <text>" ... ;
Das Leerzeichen, das auf das erste Anführungszeichen folgt, ist nicht Bestandteil des Strings.

(+d1 -- +d2) 83 " sharp "
Der Rest von +d1 geteilt durch den Wert in BASE wird in ein Ascii-Zeichen umgewandelt und dem Ausgabestring in Richtung absteigender Adressen hinzugefügt. +d2 ist der Quotient und verbleibt auf dem Stack zur weiteren Bearbeitung. Üblicherweise zwischen <# und #> benutzt.

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

#s (+d -- 0 0) 83 " sharp-s "
+d wird mit # umgewandelt, bis der Quotient zu Null geworden ist. Dabei wird jedes Zwischenergebnis in ein Ascii-Zeichen umgewandelt und dem String für die strukturierte Zahlenausgabe angefügt. Wenn +d von vornherein den Wert Null hatte, wird eine einzelne Null in den String gegeben. Wird üblichwerweise zwischen <# und #> benutzt.

/string (adr1 n1 n2 -- adr2 n3) " cut-string "
n2 ist ein Index, welcher in den String weist, der im Speicher bei der Adresse adr1 beginnt. Der String hat die Länge n1. adr2 ist die Adresse und n3 die Länge des rechten Teilstrings, der bei Teilung des ursprünglichen Strings vor dessen n2-ten Element entsteht.

<# (--) 83 " less-sharp "
Leitet die strukturierte Zahlenausgabe ein. Um eine doppelt genaue Zahl in einen 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 rücken, und addiert den Zahlenwert von char. char stellt eine Ziffer dar. adr wird nicht verändert. wird z.B. in CONVERT benutzt. char muß eine in der Zahlenbasis gültige Ziffer darstellen.

capital (cha1 -- char2)
Die Zeichen im Berich a bis z werden in die Großbuchstaben A bis Z umgewandelt. Andere Zeichen werden nicht verändert.

capitalize (adr -- adr)
Wandelt alle Kleinbuchstaben im counted string bei der Adresse adr in Gr0ßbuchstaben um. adr wird nicht verändert. Siehe auch CAPITAL .

convert (+d1 adr1 -- +d2 adr2) 83
Wandelt den Ascii-Text ab adr1+1 in eine Zahl entsprechend der Zahlenbasis BASE um. Der enstehende 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)
Prüft, ob das Zeichen char eine gültige Ziffer entsprechend der aktuellen Zahlenbasis in BASE ist. Ist das der Fall, so wird der Zahlenwert der Ziffer und TRUE auf den Stack gelegt. Ist char keine gültige Ziffer, wird FALSE hinterlegt.

hold (char --) 83
Das Zeichen char wird in den Ausgabestring für die Zahlenausgabe eingefügt. Üblicherweise zwischen <# und #> benutzt.

nullstring? (adr -- adr false)
oder (adr -- true)
Prüft, ob der counted string bei der Adresse adr ein String der Länge 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 bei der Adresse adr in eine Zahl d um. Die Umwandlung erfolgt entsprechend der Zahlenbasis in BASE. Eine Fehlerbedingung besteht, wenn die Ziffern des Strings nicht in eine Zahl verwandelt werden können. Durch Angabe eines Präfix (siehe NUMBER?) kann die Basis für diese Zahl modifiziert werden.

number? (adr -- d 0>)
oder (adr -- n 0<)
oder (adr -- adr false)
Wandelt den counted string bei der Adresse adr in eine Zahl n um. Die Umwandlung erfolgt entsprechend der Zahlenbasis in BASE oder wird vom ersten Zeichen im String bestimmt. Enthält der String zwischen den Ziffer auch die Asciizeichen für Punkt oder Komma, so wird er als doppelt genaue Zahl interpretiert und 0> gibt die Zahl der Ziffern hinter dem Punkt einschließlich an. Sonst wird der String in eine einfach genaue Zahl n umgewandelt und eine Zahl kleiner als Null hinterlassen. Wenn die Ziffern des Strings nicht in eine Zahl umgewandelt werden können, bleibt die Adresse des String erhalten und FALSE wird auf den Stack gelegt. Die Zeichen, die zur Bestimmung der Zahlenbasis dem Ziffernstring vorangestellt werden können, sind :

```
%       ( Basis 2   "binär")
&       ( Basis 10  "dezimal")
$       ( Basis 16  "hexadezimal")
h       ( Basis 16  "hexadezimal")
```

Der Wert in BASE wird dadurch nicht verändert.

scan (adr1 n1 char -- adr2 n2)
Der String mit der Länge 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 Länge des verbleibenden Strings. Wird char nicht gefunden, so ist adr2 die Adrese des ersten Bytes hinter dem String und n2 ist Null.

sign (n --) 83
Wenn n negativ ist, wird ein Minuszeichen in den Ausgabestring für die Zahlenausgabe eingefügt. Wird üblicherweise zwischen <# und #> benutzt.

skip (adr1 n1 car -- adr2 n2)
Der String mit der Länge 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 und n2 die Länge des verbleibenden Strings. Besteht der gesamte String aus dem Zeichen char so ist adr2 die Adresse des Bytes 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 für <name> im Kompilations-Vokabular und schaltet den Kompiler an. Das erste Vokabular der Suchreihenfolge (das "transient" Vokabular) wird durch das Kompilations-Vokabular ersetzt (ACHTUNG!). Das Kompilations-Vokabular wird nicht geändert. Der Quelltext wird anschließend kompiliert. <name> wird als "colon-definition" oder ":-Definition" bezeichnet. Die neue Wortdefinition für <name> kann nicht im Dictionary gefunden werden, bis das zugehörige ; oder ;CODE erfolgreich ausgeführt wurde. RECURSIVE macht <name> jedoch sofort auffindbar. Vergleiche HIDE und REVEAL .
Eine Fehlerbehandlung wird eingeleitet, wenn ein Wort während der Kompilation nicht gefunden bzw. nicht in eine Zahl (siehe auch BASE) gewandelt werden kann.
Der auf dem Stack hinterlassene Wert sys dient der Kompiler-Sicherheit und wird durch ; bzw. ;CODE abgebaut.

; (--) 83 I C "semi-colon"
(sys --) compiling
beendet die Kompilation einer :-Definition; macht den Namen dieser :-Definition im Dictionary auffindbar, schaltet den Kompiler aus, den Interpreter ein und kompiliert ein UNNEST (Siehe auch EXIT). Der Stackparameter sys, der in der Regel von : hinterlassen wurde, wird geprüft und abgebaut. Eine Fehlerbehandlung wird eingeleitet, wenn sys nicht dem erwarteten Wert entspricht.

2Constant (32b --) 83
ein definierendes Wort, das in der Form:
32b 2Constant <name>
benutzt wird. Erzeugt einen Kopf für <name> und legt 32b in dessen Parameterfeld so ab, daß bei Ausführung von <name> 32b wieder auf den Stack gelegt wird.

2Variable (--) 83
ein definierendes Wort, das in der Form:
2Variable <name>
benutzt wird. Erzeugt einen Kopf für <name> und hält 4 Byte in seinem Parameterfeld frei, die den Inhalt der 2VARIABLEN aufnehmen. Die 2VARIABLE wird nicht initialisiert. Wenn <name> ausgeführt wird, wird die Adresse des Parameterfelds auf den Stack gelegt. Siehe VARIABLE .

Alias (cfa --)
ein definierendes Wort, das typisch in der Form:
' <oldname> Alias <newname>
benutzt wird. ALIAS erzeugt einen Kopf für <newname> im Dictionary. Wird <newname> aufgerufen, so verhält es sich wie <oldname>. Insbesondere wird beim Kompilieren nicht <newname>, sondern <oldname> im Dictionary ein-

getragen. Im Unterschied zu
: <newname> <oldname> ;
ist es mit ALIAS möglich, Worte, die den Returnstack beeinflussen (z.B. >R oder R>), mit anderem Namen zu definieren. Außer dem neuen Kopf für <newname> wird kein zusaetzlicher Speicherplatz verbraucht.
Gegenwärtig wird bei Ausführung von >NAME aus einer CFA in der Regel der letzte mit ALIAS erzeugte Name gefunden.

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

Create (--) 83
ein definierendes Wort, das in der Form:
Create <name>
benutzt wird. Erzeugt einen Kopf für <name>. Die nächste freie Stelle im Dictionary (vergleiche HERE und DP) ist nach einem CREATE <name> das erste Byte des Parameterfelds von <name>. Wenn <name> ausgeführt wird, legt es die Adresse seines Parameterfelds auf den Stack. CREATE reserviert keinen Speicherplatz im Parameterfeld von <name>.

Defer (--)
ein definierendes Wort, das in der Form:
Defer <name>
benutzt wird. Erzeugt den Kopf für ein neues Wort <name> im Dictionary, hält 2 Byte in dessen Parameterfeld frei und speichert dort zunächst die Kompilationsadresse einer Fehlerroutine. Wird <name> nun ausgeführt, so wird eine Fehlerbehandlung eingeleitet. Man kann dem Wort <name> jedoch zu jeder Zeit eine andere Funktion zuweisen mit der Sequenz:
' <action> Is <name>
Nach dieser Zuweisung verhält sich <name> wie <action>. Mit diesem Mechanismus kann man zwei Probleme elegant lösen: Einerseits läßt sich <name> bereits kompilieren, bevor ihm eine sinnvolle Aktion zugewiesen wurde. Damit ist die Kompilation erst später definierter Worte (Vorwärts-Referenzen) indirekt möglich. Andererseits ist die Veränderung des Verhaltens von <name> für spezielle Zwecke auch nachträglich möglich, ohne neu kompilieren zu müssen.

Deferred Worte im System sind:
R/W , 'COLD , 'RESTART , 'ABORT , 'QUIT , NOTFOUND , .STATUS und DISKERR .
Diese Worte sind DEFERred, damit ihr Verhalten für die Anwendung geändert werden kann. 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 für <name> im Dictionary und kompiliert einen Satz von Zeigern auf Worte, die für die Eingabe von Zeichen zuständig sind. Wird <name> ausgeführt, so wird ein Zeiger auf das Parameterfeld von <name> in die Uservariable INPUT geschrieben. Alle Eingaben werden jetzt über die neuen Eingabewörte abgewickelt. Die Reihenfolge der Worte nach INPUT: <name> bis zum semi-colon muß eingehalten werden: (..key ..key? ..decode ..expect).

Im System ist das mit INPUT: definierte Wort KEYBOARD enthalten, nach dessen Ausführung alle Eingaben von der Tastatur geholt werden und ein einfacher Zeilen-Editor wirksam ist. Siehe DECODE und EXPECT .

Is (cfa --)

ein Wort, mit dem das Verhalten eines deferred Wortes verändert werden kann. IS wird in der Form:

```
' <action> Is <name>
```

benutzt. Wenn <name> kein deferred Wort ist, wird eine Fehlerbehandlung eingeleitet, sonst verhält sich <name> anschließend wie <action>. 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 für <name> im Dictionary und kompiliert einen Satz von Zeigern auf Worte, die für die Ausgabe von Zeichen zuständig sind. Wird <name> ausgeführt, so wird ein Zeiger auf das Parameterfeld von <name> in die Uservariable OUTPUT geschrieben. Alle Ausgaben werden jetzt über die neuen Ausgabeworte abgewickelt. Die Reihenfolge der Worte nach OUTPUT: <name> bis zum semi-colon muß eingehalten werden: (..emit ..cr ..type ..del ..page ..at ..at?).

Im System ist das mit OUTPUT: definierte Wort DISPLAY enthalten, nach dessen Ausführung alle Ausgaben auf den Bildschirm geleitet werden. Vergleiche auch das Wort PRINT aus dem Printer-Interface.

User (--) 83

ein definierendes Wort, benutzt in der Form:

```
User <name>
```

USER erzeugt einen Kopf für <name> im Dictionary und hält 2 Byte in der Userarea frei. Siehe UALLOT . Diese 2 Byte werden für den Inhalt der Uservariablen benutzt und werden nicht initialisiert. Im Parameterfeld der Uservariablen im Dictionary wird nur ein Offset zum Beginn der Userarea abgelegt. Wird <name> ausgeführt, so wird die Adresse des Wertes der Uservariablen in der Userarea auf den Stack gegeben.

Uservariablen werden statt normaler Variablen z.B. dann benutzt, wenn der Einsatz des Multitaskers geplant ist und mindestens eine Task die Variable unbeeinflußt von anderen Tasks benötigt. (Jede Task hat ihre eigene Userarea).

Variable (--) 83
ein definierendes Wort, benutzt in der Form:
Variable <name>
VARIABLE erzeugt einen Kopf für <name> im Dictionary und hält 2 Byte in seinem Parameterfeld frei. Siehe ALLOT . Dies Parameterfeld wird für den Inhalt der Variablen benutzt, wird jedoch nicht initialisiert. Wird <name> ausgeführt, so wird die Adresse des Parameterfeldes von <name> auf den Stack gelegt. Mit @ und ! kann der Wert von <name> gelesen und geschrieben werden. Siehe auch +! .

Vocabulary (--) 83
ein definierendes Wort, das in der Form:
Vocabulary <name>
benutzt wird. VOCABULARY erzeugt einen Kopf für <name>, das den Anfang einer neuen Liste von Worten bildet. Wird <name> ausgeführt, so werden bei der Suche im Dictionary zuerst die Worte in der Liste von <name> berücksichtigt.

Wird das VOCABULARY <name> durch die Sequenz:
<name> DEFINITIONS
zum Kompilations-Vokabular, so werden neue Wort-Definitionen in die Liste von <name> gehängt. Vergleiche auch CONTEXT , CURRENT , ALSO , TOSS , ONLY , FORTH , ONLYFORTH .

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 für 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.

align (--)
rundet HERE auf den nächsten geraden Wert auf. Ist HERE gerade, so geschieht nichts.

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

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

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

custom-remove (dic symb -- dic symb)
Ein deferred Wort, daß von FORGET , CLEAR usw. aufgerufen wird. dic ist die untere Grenze des Dictonaryteils, der vergessen wird und symb die obere. Gewöhnlich zeigt symb in den Heap. Dieses Wort kann dazu benutzt werden, eigene Datenstrukturen, die Zeiger enthalten, bei FORGET korrekt abzuabrbeiten. Es wird vom Fileinterface verwendet, daher darf es nicht einfach überschrieben werden. Man kann es z.B. in folgender Form benutzen :

```
: <name>  [ ' custom-remove >body @ , ]
          <liststart> @ remove ;
' <name> Is custom-remove
```

Auf diese Weise stellt man sicher, daß das Wort, das vorher in CUSTOM-REMOVE eingetragen war, weiterhin ausgeführt wird. Siehe auch REMOVE

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

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

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 geschützten Bereich, so wird ebenfalls eine Fehlerbehandlung eingeleitet. Es wurden Vorkehrungen getroffen, die es ermöglichen, aktive Tasks und Vokabulare, die in der Suchreihenfolge auftreten, zu vergessen.

here (-- addr) 83
addr ist die Adresse des nächsten 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.

remove (dic sym thread -- dic sym)
Dies ist ein Wort, das zusammen mit CUSTOM-REMOVE verwendet wird. dic ist die untere Grenze des Dictionarybereiches, der vergessen werden soll und sym die obere. Typisch zeigt sym in den Heap. thread ist der Anfang einer Kette von Zeigern, die durch einen Zeiger mit dem Wert Null abgeschlossen wird. Wird REMOVE dann ausgeführt, so werden alle Zeiger (durch Umhängen der übrigen Zeiger) aus der Liste entfernt, die in dem zu vergessenden Dictionarybereich liegen. Dadurch ist es möglich, FORGET und ähnliche Worte auf Datenstrukturen anzuwenden.

reveal (--)
Trägt 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 später COLD ausgeführt, so befindet sich das System im gleichen Speicherzustand wie bei Ausführung 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 möglich, daß es mehrere addr2 für ein addr1 gibt. In diesem Fall ist nicht definiert, welche ausgewählt 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 rücken. Ein Fehler liegt vor, falls der feste Teil sechs Vokabulare enthält.

Assembler (--)
Ein Vokabular, das Prozessor-spezifische Worte enthält, die für Code-Definitionen benötigt werden.

context (-- addr)
addr ist die Adresse des auswechselbaren Teils der Suchreihenfolge. Sie enthält 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 eingefügt werden.

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

Forth (--) 83
Das ursprüngliche Vokabular.

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

Only (--)
Löscht die Suchreihenfolge vollständig und ersetzt sie durch das Vokabular ONLY im festen und auswechselbaren Teil der Suchreihenfolge. ONLY enthält nur wenige Worte, die für die Erzeugung einer Suchreihenfolge benötigt werden.

Onlyforth (--)
entspricht der häufig benötigten Sequenz
ONLY FORTH ALSO DEFINITIONS.

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

toss (--)
Entfernt das erste Vokabular des festen Teils der Suchreihenfolge. Insofern ist es das Gegenstück 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 enthält den Anfang einer Liste mit allen Vokabularen. Diese Liste wird u.a. für FORGET benötigt.

vp (-- addr) "v-p"
Eine Variable, die das Ende der Suchreihenfolge markiert. Sie enthält ausserdem Informationen über die Länge der Suchreihenfolge.

Heap - Worte

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

halign (--) "h-align"
HEAP wird auf den nächsten geradzahligen Wert abgerundet. Ist HEAP gerade, geschieht nichts.

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 geändert.

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

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

Kontrollstrukturen

+LOOP (n --) (sys --) 83,I,C "plus-loop" compiling

n wird zum Loopindex addiert. Falls durch die Addition die Grenze zwischen limit-1 und limit überschritten 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 --) (-- sys) 83,I,C "question-do" compiling

Wird in der folgenden Art benutzt:

```
?DO ... LOOP   bzw.  ?DO ... +LOOP
```

Beginnt eine Schleife. Der Schleifenindex beginnt mit w2, limit ist w1. Details über die Beendigung von Schleifen: s. +LOOP. Ist w2=w1 , so wird der Schleifenrumpf überhaupt nicht durchlaufen.

?exit (flag --) "question-exit"

Führt EXIT aus, falls das flag wahr ist. Ist das flag falsch, so geschieht nichts.

BEGIN (--) (sys ---) 83,I,C 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 Länge count gegeben ist, in ein Intervall umzurechnen, das durch Anfangswert start und Endwert+1 limit beschrieben wird.

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

DO (w1 w2 --) (sys --) 83,I,C compiling

Entspricht ?DO , jedoch wird der Schleifenrumpf mindestens einmal durchlaufen. Ist w1=w2 , so wird der Schleiferumpf 65536-mal durchlaufen.

ELSE (--) (sys1 -- sys2) 83,I,C compiling

Wird in der folgenden Art benutzt:

```
flag IF ... ELSE ... THEN
```

ELSE wird unmittelbar nach dem Wahr-Teil , der auf IF folgt, ausgeführt. ELSE setzt die Ausführung unmittelbar hinter THEN fort.

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

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 ausgeführt 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 äusseren Schleife auf den Stack zu holen.

LEAVE (--) 83,C
Setzt die Ausführung des Programmes hinter dem nächsten LOOP oder +LOOP fort, wobei die zugehöhrige Schleife beendet wird. Mehr als ein LEAVE pro Schleife ist möglich, ferner kann LEAVE zwischen anderen Kontroll-strukturen auftreten. Der Forth83-Standard schreibt abweichend vom volksFORTH vor, daß LEAVE ein immediate Wort ist.

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

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

REPEAT (--) 83,I,C
(-- sys) compiling
Wird in der folgenden Form benutzt:
BEGIN (.. WHILE) .. REPEAT
REPEAT setzt die Ausführung 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 herbeigeführt 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 Ausführung der Schleife bis UNTIL oder REPEAT fortgesetzt, ist es falsch, so wird die Schleife beendet und das Programm hinter UNTIL bzw. REPEAT fortgesetzt. Es können mehre WHILE in einer Schleife verwendet werden.

Compiler - Worte

," (--) "comma-quote"
Speichert einen counted String im Dictionary ab HERE. Dabei wird die Länge des Strings in dessen erstem Byte, das nicht zur Länge hinzugezählt 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 (gewöhnlich ASCII). Falls sich das System im kompilierenden Zustand befindet, so wird char als Konstante compiliert. Wird die :-definition später ausgeführt, so liegt char auf dem Stack.

compile (--) 83,C
Typischerweise in der folgenden Art benutzt:

```
: <name> ... compile <namex> ... ;
```

Wird <name> ausgeführt, so wird die Kompilationsadresse von <namex> zum Dictionary hinzugefügt und nicht ausgeführt. 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 später:

```
<namex> <name>
```

wobei <create> CREATE oder ein anderes Wort ist, das CREATE ausführt.

Zeigt das Ende des Wort-erzeugenden Teils des definierenden Wortes an. Beginnt die Kompilation des Codes, der ausgeführt 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 ausgeführt.

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

Literal (-- 16b) 83,I,C
(16b --) compiling
Typisch in der folgenden Art benutzt:

```
[ 16b ] Literal
```

Kompiliert ein systemabhängiges Wort, so daß bei Ausführung 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 für 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 ausgeführt. 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 später ausgeführt 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 möglich.

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 höher ist, als die Nummer des gegenwärtig interpretierten Blockes.

+thru (n1 n2 --) "plus-thru"
LOAD die Blöcke hintereinander, die n1..n2 vom gegenwärtigen Block entfernt sind.
Beispiel : 1 2 +thru lädt die nächsten beiden Blöcke.

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

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

>interpret (--) "to-interpret"
Ein deferred Wort, das die gegenwärtige Interpretationsroutine aufruft, ohne eine Rückkehradresse 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 enthält. 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 enthält 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 erfüllt 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 gegenwärtigen 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 zurückgebracht. s.a. BLK >IN BLOCK

loadfile (-- addr)
Addr ist die Adresse einer Variablen, die auf das Forth-File zeigt, das gerade geladen wird. Diese Variable wird bei Aufruf von LOAD , THRU usw. auf das aktuelle File gesetzt.

name (-- addr)
Holt den nächsten String, der durch Leerzeichen eingeschlossen wird, aus dem Quelltext, wandelt ihn in Grossbuchstaben um und hinterlässt 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 String-eingabeformate 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 Länge +n des nächsten Strings im Quelltext, der durch den Delimiter char abgeschlossen wird. +n ist Null, falls der Quelltext erschöpft oder das erste Zeichen char ist. >IN wird verändert.

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

source (-- addr +n)
liefert Anfang addr und maximale Länge +n des Quelltextes. Ist BLK Null, beziehen sich Anfang und Länge 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 gegenwärtigen Zustand enthält. Der Wert Null zeigt den interpretierenden Zustand an, ein von Null verschiedener Wert den kompilierenden Zustand.

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

word (char -- addr)83
erzeugt einen counted String durch Lesen von Zeichen vom Quelltext, bis dieser erschöpft ist oder der Delimiter char auftritt. Der Quelltext wird nicht zerstört. Führende Delimiter werden ignoriert. Der gesamte String wird im Speicher beginnend ab Adresse addr als eine Sequenz von Bytes abgelegt. Das erste Byte enthält die Länge des Strings (0..255). Der String wird durch ein Leerzeichen beendet, das nicht in der Längenangabe enthalten ist. Ist der String länger als 255 Zeichen, so ist die Länge undefiniert. War der Quelltext schon erschöpft, als WORD aufgerufen wurde, so wird ein String der Länge Null erzeugt.
Wird der Delimiter nicht im Quelltext gefunden, so ist der Wert von >IN die Länge des Quelltextes. Wird der Delimiter gefunden, so wird >IN so verändert, dass >IN das Zeichen hinter dem Delimiter indiziert. #TIB wird nicht verändert.
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
Lädt 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" ausgeführt. 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 überprüfen.

?stack (--) "question-stack"
Prüft, ob der Stack über- oder leerläuft. Der Stack läuft leer, falls der Stackpointer auf eine Adresse oberhalb von S0 zeigt. In diesem Fall wird die Fehlermeldung "stack empty" ausgegeben. Der Stack läuft über, 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, führt 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" später ausgeführt, so geschieht nichts, wenn flag falsch ist. Ist flag wahr, so wird der Stack geleert und der Inhalt von ERRORHANDLER ausgeführt. Beachten Sie bitte, daß im Gegensatz zu ABORT kein END-TRACE ausgeführt wird.

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 ausgeführt, 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 unterdrückt. Kurioserweise ist werden die Warnungen also durch WARNING ON abgeschaltet.

Sonstiges

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

'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 Geräte 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 Kommandoinhterpreter (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 Geräte 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, führt 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 ausgeführt wird. Es ist mit NOOP vorbesetzt und kann dazu benutzt werden, Informationen über den Systemzustand oder den Quelltext auszugeben.

bye (--)
Dieses Wort führt FLUSH und EMPTY aus. Anschliessend wird der Monitor des Rechners angesprungen oder eine andere implementationsabhängige Funktion ausgeführt.

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

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

makeview (-- 16b)
Ein deferred Wort, dem mit IS ein Wort zugewiesen wurde. Dieses Wort erzeugt aus dem gerade kompilierten Block eine 16b Zahl, die von Create in das Viewfeld eingetragen wird. Das deferred Wort wird benötigt, um das Fileinterface rausschmeißen zu können.

next-link (-- addr)
addr ist die Adresse einer Uservariablen, die auf die Liste aller Nextroutinen zeigt. Der Assembler erzeugt bei Verwendung des Wortes NEXT Code, für den ein Zeiger in diese Liste eingetragen wird. Die so entstandene Liste wird vom Tracer benutzt, um alle im System vorhandenen Kopien des Codes für NEXT zu modifizieren.

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 enthält. Vergleiche (ERROR und SCR.

restart (--)
Bewirkt den Warmstart des Systems. Es setzt 'QUIT , ERRORHANDLER und 'ABORT auf ihre normalen Werte und führt ABORT aus.

scr (-- adr) 83 "s-c-r"
adr ist die Adresse einer Variablen, die die Nummer des gerade editierten Screens enthält. 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) . block' kann positiv oder negativ sein. Beispiel:

```
23 1 >drive block
```

holt den Block mit der Nummer 21 vom Laufwerk 1, egal welches Laufwerk gerade das aktuelle ist.

all-buffers (--)
Belegt den gesamten Speicherbereich von LIMIT abwärts bis zum oberen Ende des Returnstacks mit Blockpuffern. Siehe ALLOTBUFFER .

allotbuffer (--)
Fügt der Liste der Blockpuffer noch einen weiteren hinzu, falls oberhalb vom Ende des Retunstacks dafür noch Platz ist. FIRST wird entsprechend geändert. Vergleiche FREEBUFFER und ALL-BUFFERS .

b/blk (-- &1024) "bytes pro block"
Die Länge eines Blocks (bzw. Screens, immer 1 KByte) wird auf den Stack gelegt. Siehe B/BUF .

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

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

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 ISFILE . BLOCK prüft den Pufferbereich auf die Existenz des Blocks Nummer u. Befindet sich der Block u in keinem der Blockpuffer, so wird er vom Massenspeicher in einen an ihn vergebenen Blockpuffer geladen. Falls der Block in diesem Puffer UPDATEd , wird er auf den Massenspeicher gesichert, bevor der Blockpuffer an den Block u vergeben wird. Nur die Daten im letzten Puffer, der über BLOCK oder BUFFER angesprochen wurde, sind sicher zugreifbar. Alle anderen Blockpuffer dürfen nicht mehr als gültig angenommen werden (möglicherweise existiert nur 1 Blockpuffer). Vorsicht ist bei Benutzung des Multitaskers geboten, da eine andere Task BLOCK oder BUFFER ausführen kann. Der Inhalt eines Blockpuffers wird nur auf den Massenspeicher gesichert, wenn der Block mit UPDATE als verändert gekennzeichnet wurde.

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 undefiniert.

convey (1st.block last.block to.blk --)
Verschiebt die Blöcke von 1st.blk bis last.blk einschließlich nach to.blk folgende. Die Bereiche dürfen sich überlappen. Eine Fehlerbehandlung wird eingeleitet, wenn last.blk kleiner als 1st.blk ist.
Die Blöcke werden aus dem File in FROMFILE ausgelesen und in das File in ISFILE geschrieben. FROMFILE und ISFILE dürfen gleich sein.
Beispiel :

```
4 6 5 convey
```

kopiert die Blöcke 4,5,6 nach 5,6,7. Der alte Inhalt des Blockes 7 ist verloren, der Inhalt der Blöcke 4 und 5 ist gleich.

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

core? (blk file -- addr/false)"core-question"
Prüft, ob sich der Block blk des Files file in einem der Blockpuffer befindet und liefert dann die Adresse addr des ersten Datenbytes. Befindet sich der Block nicht im Speicher, so wird false als Ergebnis geliefert. Vergleiche BLOCK , BUFFER und ISFILE .

drive (#drv --)
Selektiert das Laufwerk mit der Nummer #drv als aktuelles Laufwerk. 0 BLOCK liefert die Adresse des ersten Blocks auf dem aktuellen Laufwerk. Vergleiche >DRIVE und OFFSET.

drv? (block -- #drv) "drive-question"
#drv ist die Nummer des Laufwerks auf dem sich der Block block befindet. Vergleiche OFFSET >DRIVE und DRIVE.

empty-buffers (--)
Löscht den Inhalt aller Blockpuffer. UPDATEd Blockpuffer werden nicht auf den Massenspeicher zurückgeschrieben.

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
Sichert alle UPDATEd Blocks auf den Massenspeicher und löscht dann alle Blockpuffer. Vergleiche SAVE-BUFFERS und EMPTY-BUFFERS .

freebuffer (--)
Entfernt den Blockpuffer mit der niedrigsten Adresse aus der Liste der Blockpuffer und gibt den dadurch belegten Speicherbereich frei. FIRST wird entsprechend verändert (um B/BUF erhöht). Ist der Inhalt des Puffers UPDATEd , so wird er zuvor auf den Massenspeicher gesichert. Gibt es im System nur noch einen Blockpuffer, so geschieht nichts. Vergleiche ALLOTBUFFER .

fromfile (-- adr)
adr ist die Adresse einer Variablen, deren Wert die Nummer eines Files ist, aus dem CONVEY und COPY die Blöcke auslesen, um sie in das File, das in ISFILE steht, zu kopieren. Siehe das Kapitel zum Fileinterface

isfile (-- 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. Siehe das Kapitel zum Fileinterface.

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, deren Inhalt zu der Blocknummer addiert wird, die sich bei Aufruf von BLOCK BUFFER usw. auf dem Stack befindet. OFFSET wird durch DRIVE verändert.

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 systemabhängige Wort ausgeführt wird, das einen Block vom Massenspeicher holt. Dabei ist addr die Anfangsadresse des Speicherbereichs für den Block block, file die Filenummer des Files, in dem sich der Block befindet und n=0, falls der Block vom Speicherbereich auf den Massenspeicher geschrieben 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
Sichert alle als UPDATEd gekennzeichneten Blöcke aus den Blockpuffern auf den Massenspeicher. Das UPDATE - Kennzeichen wird für alle Blöcke zurückgesetzt, die Blöcke werden jedoch nicht verändert und bleiben für weitere Zugriffe im Speicher erhalten.

update (--) 83
Der zuletzt mit BLOCK BUFFER usw. zugegriffene Block wird als verändert gekennzeichnet. Blöcke mit solcher Kennzeichnung werden auf den Massenspeicher zurückgeschrieben, wenn ihr Blockpuffer für einen anderen Block benötigt oder wenn SAVE-BUFFERS ausgeführt wird. Vergleiche PREV.

Atari ST - spezifische Worte

#col (-- addr) "number-col"
addr ist die Adresse einer Variablen, die die Nummer der aktuellen Cursorzeile enthält.

#esc (-- n) "number-escape"
n ist der Ascii-Wert für Escape.

#lf (-- n) "number-linefeed"
n ist der Ascii-Wert für Linefeed.

#row (-- addr) "number-row"
addr ist die Adresse einer Variablen, die die Nummer der aktuellen Cursorspalte enthält.

bconin (dev# -- char) "b-con-in"
char ist ein Zeichen, das vom Gerät mit der Nummer dev# eingelesen wird. BCONIN wartet (und hängt dadurch), bis ein Zeichen bereitsteht. Typische Gerätenummern sind:

```
0 PRT;  Centronics-Schnittstelle (Druckerport)
1 AUX;  RS232-Schnittstelle
2 CON;  Tastatur und Bildschirm
3 MIDI; Midi-Schnittstelle
4 IKBD; Tastatur-Port
```

Gerätenummer 4 ist bei dieser Funktion nicht erlaubt.

bconout (char dev# --) "b-con-out"
char ist ein Zeichen, das an das Gerät mit der Nummer dev# ausgegeben wird. BCONIN wartet (hängt), bis das Zeichen ausgegeben wurde. Die Gerätenummern sind die gleichen wie bei BCONIN . Alle Gerätenummern sind erlaubt.

bconstat (dev# -- flag) "b-con-stat"
flag ist TRUE, wenn ein Zeichen auf dem Gerät mit der Nummer dev# bereitsteht. Sonst ist flag = FALSE. Die Gerätenummern sind die gleichen wie bei BCONIN . Die Gerätenummern 0 und 4 sind bei dieser Funktion nicht erlaubt.

bcostat (dev# -- flag) "b-co-stat"
flag ist TRUE, wenn das Gerät mit der Nummer dev# bereit ist, ein Zeichen zu empfangen. Sonst ist flag = FALSE. Die Gerätenummern sind die gleichen wie bei BCONIN . Alle Gerätenummern sind bei dieser Funktion erlaubt.

con! (8b --) "con-store"
gibt 8b auf die CONsole (Bildschirm) aus. Ascii-Werte < $20 werden als Steuercodes interpretiert.

curleft (--)
setzt den Cursor um eine Spalte nach links.

curoff (--)
schaltet den Cursor aus.

curon (--)
schaltet den Cursor ein.

currite (--)
setzt den Cursor um eine Spalte nach rechts.

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

drv0 (--)
Kurzform für 0 DRIVE .
Das Laufwerk 0 (bzw. A) wird als aktuelles Laufwerk selektiert. Siehe DRIVE.

drv1 (--)
Wie DRV0.

drvinit (--)
Ein Wort, das von RESTART ausgeführt wird. Dieses Wort kann dazu verwendet werden, Floppycontroller, Files o.Ä. zu initialieren.

getkey (-- n)
die unteren 7 Bit von n enthalten den Ascii-Code, die unteren 8 Bit den ANSI-Code (Ach Nein! Sch... IBM) des nächsten Tastendrucks, die oberen 8 Bit den Tastatur-Scancode. War keine Taste gedrückt, ist n = FALSE . (Vergleiche KEY? und KEY bzw. STKEY? und STKEY .)

keyboard (--)
ein mit INPUT: definiertes Wort, das die Tastatur als Eingabegerät setzt. Die Worte KEY , KEY? , DECODE und EXPECT beziehen sich auf die Tastatur. Siehe STKEY , STKEY? , STDECODE und STEXPECT .

rwabs (r/wf addr rec# -- flag) "r-w-abs"
Ist r/wf = TRUE, wird ein Block Daten ($400 bzw. &1024) Bytes vom Disketten-Sektor mit der Nummer rec# nach addr gelesen. Ist r/wf = FALSE, werden Daten von addr auf die Diskette geschrieben. Diese Routine wird von STR/W benutzt.

STat (row col --) "s-t-at"
positioniert den Cursor in die Zeile row und die Spalte col. Ein Fehler liegt vor, wenn row > $18 (&24) oder col > $40 (&64) ??? ist. Vergleiche AT .

STat? (-- row col) "s-t-at-question"
row ist die aktuelle Zeilennummer des Cursors, col die aktuelle Spaltennummer. Vergleiche AT? .

STcr (--) "s-t-c-r"
setzt den Cursor in die erste Spalte der nächsten Zeile. Ein PAUSE wird ausgeführt.

STdecode (addr pos1 key -- addr pos2) "s-t-decode"
wertet key aus. key wird in der Speicherstelle addr+pos1 abgelegt und als Echo auf dem Bildschirm ausgegeben. Die Variable SPAN und pos werden inkrementiert. Folgende Tasten werden besonders behandelt: Cursor rechts und Cursor links beeinflussen nur pos1 und den Cursor. Delete löscht das Zeichen unter dem Cursor und dekrementiert SPAN . Backspace löscht das Zeichen links vom Cursor und dekrementiert pos1 und SPAN . Insert fügt an der Cursorposition ein Leerzeichen ein. SPAN wird inkrementiert. Return positioniert den Cursor auf das letzte Zeichen. Vergleiche INPUT: und STexpect .

STdel (--) "s-t-del"
löscht ein Zeichen links vom Cursor. Vergleiche DEL .

STemit (8b --) "s-t-emit"
gibt 8b auf dem Bildschirm aus. Ein PAUSE wird ausgeführt. Alle Werte werden als Zeichen ausgegeben, Steuercodes sind nicht möglich, d. h. alle Werte < $20 werden als ATARI-Spezifische Zeichen ausgegeben. Vergleiche CON! und EMIT .

STexpect (addr len --) "s-t-expect"
erwartet len Zeichen vom Eingabegerät, die ab addr im Speicher abgelegt werden. Ein Echo der Zeichen wird ausgegeben. Return beendet die Eingabe vorzeitig. Ein abschließendes Leerzeichen wird immer ausgegeben. Die Länge der Zeichenkette wird in der Variablen SPAN übergeben. Vergleiche EXPECT .

STkey (-- 16b) "s-t-key"
wartet auf einen Tastendruck. Während der Wartezeit wird PAUSE ausgeführt. Die unteren 7 Bit von 16b enthalten den Ascii-Code, die unteren 8 Bit den ANSI-Code der Taste, die oberen 8 Bit den Tastatur-Scancode. Steuerzeichen werden nicht ausgewertet, sondern unverändert abgeliefert. Vergleiche KEY .

STkey? (-- flag) "s-t-key-question"
flag ist TRUE, wenn eine Taste gedrückt wurde, sonst FALSE . Vergleiche KEY? .

STpage (--) "s-t-page"
löscht den Bildschirm und positioniert den Cursor in die linke obere Ecke. Vergleiche PAGE .

STr/w (adr blk r/w file -- f) "s-t-r-w"
liest oder schreibt einen Block Daten ($400 oder &1024 Bytes) vom Diskettenblock blk nach adr bzw. von adr auf den Block blk. Ist r/w falsch, so werden die Daten von der Adresse adr geschrieben, ist r/w wahr, so werden sie nach Adresse adr gelesen. STR/W ermittelt das Laufwerk, auf dem sich der Block blk befindet.
file ist die Nummer eines Files, dem der zu bearbeitende Block entstammt. file muß Null sein, da dieses Wort nicht mit Files arbeiten kann !

STtype (addr len --)
gibt den String, der im Speicher bei addr beginnt und die Länge len hat, auf dem Bildschirm aus. Ein PAUSE wird ausgeführt. Vergleiche TYPE , OUTPUT: und STEMIT .

Multitasking

's (Tadr -- usradr) I "tick-s"
wird benutzt in der Form:
... <taskname> 's <username> ...
'S liest den Namen einer USERvariablen <username> und hinterläßt die Adresse usradr dieser USERvariablen in der durch Tadr gekennzeichneten Task. Typisch wird Tadr durch Ausführung von <taskname> erzeugt. Eine Fehlerbedingung liegt vor, wenn <username> nicht der Name einer USERvariablen ist. Vergleiche USER und TASK . 'S ist für den Zugriff auf USERvariablen einer Task durch eine andere Task vorgesehen.

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

lock (semadr --)
Der Semaphor (eine VARIABLE), dessen Adresse auf dem Stack liegt, wird von der Task, die LOCK ausführt, blockiert. Dazu prüft LOCK den Inhalt des Semaphors. Zeigt der Inhalt an, daß eine andere Task den Semaphor blockiert hat, so wird PAUSE ausgeführt, bis der Semaphor freigegeben ist. Ist der Semaphor freigegeben, so schreibt LOCK das Kennzeichen der Task, die LOCK ausführt, in den Semaphor und sperrt ihn damit für alle anderen Tasks. Den FORTH-Code zwischen semadr LOCK ... und ... semadr UNLOCK kann also immer nur eine Task zur Zeit ausführen. Semaphore schützen gemeinsam benutzte Geräte (z.B. den Drucker) vor dem gleichzeitigen Zugriff durch verschiedene Tasks. Vergleiche UNLOCK , RENDEZVOUS und die Beschreibung des Taskers.

multitask (--)
schaltet das Multitasking ein. Das Wort PAUSE ist dann keine NOOP -Funktion mehr, sondern gibt die Kontrolle über die FORTH-Maschine an eine andere Task weiter.

pass (n_0 .. n_{r-1} Tadr r --)
aktiviert die Task, die durch Tadr gekennzeichnet ist, und weckt sie auf. r gibt die Anzahl der Parameter n_0 bis n_{r-1} an, die vom Stack der PASS ausführenden Task auf den Stack der durch Tadr gekennzeichneten Task übergeben werden. Die Parameter n_0 bis n_{r-1} stehen dieser Task dann in der gleichen Reihenfolge auf ihrem Stack zur weiteren Verarbeitung zur Verfügung. Vergleiche ACTIVATE .

pause (--)
ist eine NOOP -Funktion, wenn der Singletask-Betrieb eingeschaltet ist; bewirkt jedoch, nach Ausführung von MULTITASK , daß die Task, die PAUSE ausführt, die Kontrolle über die FORTH-Maschine an eine andere Task abgibt. Existiert nur eine Task, oder schlafen alle anderen Tasks, so wird die Kontrolle unverzüglich an die Task zurückgegeben, die PAUSE ausführte. Ist

mindestens eine andere Task aktiv, so wird die Kontrolle von dieser übernommen und erst bei Ausführung von PAUSE oder STOP in dieser Task an eine andere Task weitergegeben. Da die Tasks ringförmig miteinander verkettet sind, erhält die Task, die zuerst PAUSE ausführte, irgendwann die Kontrolle zurück. Eine Fehlerbedingung liegt vor, wenn eine Task weder PAUSE noch STOP ausführt. Vergleiche STOP , MULTITASK und SINGLETASK .

rendezvous (semadr --)
gibt den Semaphor (die VARIABLE) mit der Adresse semadr frei (siehe UNLOCK) und führt PAUSE aus, um anderen Tasks den Zugriff auf das, durch diesen Semaphor geschützte, Gerät zu ermöglichen. Anschließend wird LOCK ausgeführt, um das Gerät zurück zu erhalten.

singletask (--)
schaltet das Multitasking aus. PAUSE ist nach Ausführung von SINGLETASK eine NOOP -Funktion. Eine Fehlerbedingung besteht, wenn eine Hintergrund-Task SINGLETASK ohne anschließendes MULTITASK ausführt, da die Main- oder Terminal-Task dann nie mehr die Kontrolle bekommt. Vergleiche UP@ und UP! .

sleep (Tadr --)
bringt die Task, die durch Tadr gekennzeichnet ist, zum Schlafen. SLEEP hat den gleichen Effekt, wie die Ausführung von STOP durch die Task selbst. Der Unterschied ist, daß STOP in der Regel am Ende des Jobs der Task ausgeführt wird, SLEEP trifft die Task zu einem nicht vorhersehbaren Zeitpunkt, so daß die laufende Arbeit der Task abgebrochen wird. Vergleiche WAKE .

stop (--)
bewirkt, daß die Task, die STOP ausführt, sich schlafen legt. Wichtige Zeiger der FORTH-Maschine, die den Zustand der Task kennzeichnen, werden gerettet, dann wird die Kontrolle an die nächste Task abgegeben, deren Zeiger wieder der FORTH-Maschine übergeben werden, so daß diese Task ihre Arbeit an der alten Stelle aufnehmen kann. PAUSE führt diese Aktionen ebenfalls aus, der Unterschied zu STOP ist, daß die ausführende Task bei PAUSE aktiv, bei STOP hingegen schlafend hinterlassen wird. Vergleiche PAUSE , WAKE und SLEEP .

Task (rlen slen --)
ist ein definierendes Wort, das in der Form:
rlen slen Task <cccc>
benutzt wird. TASK erzeugt einen Arbeitsbereich für einen weiteren Job, der gleichzeitig zu schon laufenden Jobs ausgeführt werden soll. Die Task erhält den Namen cccc, hat einen Stack-Bereich der Länge slen und einen Returnstack-Bereich der Länge rlen. Im Stack-Bereich liegen das Task-eigene Dictionary einschließlich PAD, das in Richtung zu höheren Adressen wächst, und der

Daten-Stack, der zu niedrigen Adressen wächst. Im Returnstack-Bereich befinden sich die Task-eigene USER-Area (wächst zu höheren Adressen) und der Returnstack, der gegen kleinere Adressen wächst. Eine Task ist ein verkleinertes Abbild des FORTH-Systems, allerdings ohne den Blockpuffer-Bereich, der von allen Tasks gemeinsam benutzt wird. Zur Zeit ist es nicht zugelassen, daß Jobs einer Hintergrundtask kompilieren. Die Task ist nur der Arbeitsbereich für einen Hintergrund-Job, nicht jedoch der Job selbst.

Die Ausführung von cccc in einer beliebigen Task hinterläßt die gleiche Adresse, die die Task cccc selbst mit UP@ erzeugt und ist zugleich die typische Adresse, die von LOCK , UNLOCK und RENDEZVOUS im Zusammenhang mit Semaphoren verwendet wird, bzw. von 'S , ACTIVATE , PASS , SLEEP und WAKE erwartet wird.

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

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

up@ (-- Tadr) "u-p-fetch"
liefert die Adresse Tadr des ersten Bytes der USER-Area der Task, die UP@ ausführt. Siehe TASK . In der USER-Area sind Variablen und andere Datenstrukturen hinterlegt, die jede Task für sich haben muß. Vergleiche UP! .

up! (adr --) "u-p-store"
richtet den UP (User Pointer) der FORTH-Maschine auf adr. Vorsicht ist bei der Verwendung von UP! in Hintergrund-Tasks geboten. Vergleiche UP@ .

wake (Tadr --)
weckt die Task, die durch Tadr gekennzeichnet ist, auf. Die Task führt ihren Job 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 erhält, wenn die Backspace-Taste gedrückt wird.

#cr (-- n) "number-c-r"
n ist der Wert, den man durch KEY erhält, wenn die Return- (Enter-) Taste gedrückt wird.

#tib (-- adr) 83 "number-t-i-b"
adr ist die Adresse einer Variablen, die die Länge des aktuellen Textes im Text-Eingabe-Puffer enthält. Vergleiche TIB .

-trailing (adr +n0 -- adr +n1) 83 "minus-trailing"
adr ist die Anfangsadresse und +n0 die Länge eines Strings. -TRAILING verändert +n0 so zu +n1, daß eventuell am Ende vorhandene Leerzeichen nicht mehr in der Stringlänge +n1 enthalten sind. Der String selbst bleibt unangetastet. Ist +n0 = 0 oder besteht der ganze String aus Leerzeichen, so ist +n1 = 0.

. (n --) 83 "dot"
n wird vorzeichenbehaftet ausgegeben.

." (--) 83 I C "dot-quote"
(--) compiling
wird in :-Definitionen in der Form verwendet:
: <name> ... ." cccc" ... ;
Der String cccc wird bis zum abschließenden " so kompiliert, daß bei Ausführung von <name> der String cccc ausgedruckt wird. Das Leerzeichen nach ." und das abschließende " sind Pflicht und gehören 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 werden nicht mit ausgedruckt.

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

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

?cr (--) "question-c-r"
prüft, ob in der aktuellen Zeile mehr als C/L Zeichen ausgegeben wurden und führt in diesem Fall CR aus.

at (row col --)
positioniert die Schreibstelle des Ausgabegerätes in die Zeile row und die Spalte col. AT ist eines der über OUTPUT vektorisierten Worte. Siehe AT?

at? (-- row col) "at-question"
ermittelt die aktuelle Position der Schreibstelle des Ausgabegerätes 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 enthält, die zur Wandlung von Zahlenein- und -ausgaben benutzt wird.

bl (-- n) "b-l"
n ist der ASCII-Wert für ein Leerzeichen.

c/l (-- +n) "characters-per-line"
+n ist die Anzahl der Zeichen pro Screenzeile. Aus historischen Gründen ist dieser Wert &64 bzw. $40 .

col (-- +n)
+n ist die Spalte in der sich die Schreibstelle des Ausgabegerätes gerade befindet. Vergleich ROW und AT? .

cr (--) 83 "c-r"
bewirkt, daß die Schreibstelle des Ausgabegerätes an den Anfang der nächsten 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 rechtsbündig aus. Reicht +n nicht zur Darstellung der Zahl aus, so wird über den rechten Rand des Feldes hinaus ausgegeben. Die Zahl d wird in jedem Fall vollständig dargestellt.

decimal (--)
stellt BASE auf den Wert $0A bzw. &10 ein. Alle Zahlenein- und -ausgaben erfolgen nun im dezimalen System.

decode (adr +n0 key -- adr +n1)
wertet key aus. Typisch werden normale druckbare ASCII-Zeichen in die Speicherstelle adr + +n0 übertragen, als Echo zum Ausgabegerät gesandt und +n0 inkrementiert. Andere Zeichen (#BS #CR Steuercodes) können andere Aktionen zur Folge haben. Eines der über INPUT vektorisierten Worte. Wird von EXPECT benutzt. Siehe STDECODE .

del (--)
löscht das zuletzt ausgegebene Zeichen. Eines der OUTPUT-Worte. Bei Druckern ist die korrekte Funktion nicht immer garantiert.

emit (16b --) 83
die unteren 8 Bit werden an das Ausgabegerät ausgegeben. Ist das Zeichen nicht druckbar (insbesondere Steuercodes), so wird ein Punkt ausgegeben. Eines der OUTPUT-Worte.

expect (adr +n --) 83
empfängt Zeichen und speichert sie ab Adresse adr im Speicher. Die Übertragung wird beendet, bis ein #CR erkannt oder +n Zeichen übertragen wurden. Ein #CR wird nicht mit abgespeichert. Ist +n = 0 , so werden keine Zeichen übertragen. Alle Zeichen werden als Echo, statt #CR wird ein Leerzeichen ausgegeben. Eines der INPUT-Worte. Vergleiche SPAN

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

input (-- adr) U
adr ist die Adresse einer Uservariablen, die einen Zeiger auf ein Feld von vier Kompilationsadressen enthält, die für ein Eingabegerät die Funktionen KEY KEY? DECODE und EXPECT realisieren. Vergleiche die gesonderte Beschreibung der INPUT- und OUTPUT-Struktur.

key (-- 16b) 83
empfängt ein Zeichen vom Eingabegerät. Die niederwertigen 8 Bit enthalten den ASCII-Code des zuletzt empfangenen Zeichens. Alle gültigen ASCII-Codes können empfangen werden. Die oberen 8 Bit enthalten systemspezifische Informationen. Es wird kein Echo ausgesandt. KEY wartet, bis tatsächlich ein Zeichen empfangen wurde. Eines der INPUT-Worte.

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

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

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

ouput (-- adr) U
adr ist die Adresse einer Uservariablen, die einen Zeiger auf sieben Kompilationsadressen enthält, die für ein Ausgabegerät die Funktionen EMIT CR TYPE DEL PAGE AT und AT? realisieren. Vergleiche die gesonderte Beschreibung der Ouput-Struktur.

page (--)
bewirkt, daß die Schreibstelle des Ausgabegerätes auf eine leere neue Seite bewegt wird. Eines der OUTPUT-Worte.

query (--) 83
Zeichen werden von einem Eingabegerät geholt und in den Text-Eingabe-Puffer übertragen, der bei TIB beginnt. Die Übertragung endet beim Empfang von #CR oder wenn die Länge des Text-Eingabe-Puffers erreicht wird. Der Inhalt von >IN und BLK wird zu 0 gesetzt und #TIB enthält die Zahl der empfangenen Zeichen. Um Text aus dem Puffer zu lesen, kann WORD benutzt werden. Siehe EXPECT.

row (-- +n)
+n ist die Zeile, in der sich die Schreibstelle des Ausgabegerätes befindet. Vergleiche COL und AT? .

space (--) 83
sendet ein Leerzeichen an das Ausgabegerät.

spaces (+n --) 83
sendet +n Leerzeichen an das Ausgabegerät.

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

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

stop? (-- flag) "stop-question"
Ein komfortables Wort, das es dem Benutzer gestattet, einen Programmablauf anzuhalten oder zu beenden. Steht vom Eingabegerät ein Zeichen zur Verfügung, so wird es eingelesen. Ist es #ESC oder CTRL-C , so ist flag TRUE, sonst wird auf das nächste Zeichen gewartet. Ist dieses jetzt #ESC oder CTRL-C , so wird STOP? mit TRUE verlassen, sonst mit FALSE . Steht kein Zeichen zur Verfügung, so ist flag FALSE .

tib (-- adr) 83
liefert die Adresse des Text-Eingabe-Puffers. Er wird benutzt, um die Zeichen vom Quelltext des aktiven Gerätes zu halten. Siehe >TIB .

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

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

u.r (u +n --) "u-dot-r"

die Zahl u wird vorzeichenlos rechtsbündig in einem Feld der Breite +n ausgegeben. Reicht +n zur Darstellung von u nicht aus, so wird über den rechten Rand hinaus ausgegeben.

Erweiterte Adressierung

Auf den modernen Rechnern stehen zumeist mehr als 64 Kbyte Speicher zur Verfügung. Forth ist gewöhnlich auf einen Adreßbereich von 64 Kbyte beschränkt, da die Stacks nur 16 Bit breit sind. Daher kann Forth zunächst nicht sehr gut mit den 32Bit Adressen, die für solche Betriebssysteme erforderlich sind, umgehen.
Das Forth wird bei diesen Betriebssystemen normalerweise nicht an den Anfang des Speichers geladen. Da die meisten Operationen nur 16Bit Adressen verarbeiten (wie ! @ TYPE COUNT usw.), zählen sie diese Adressen ab dem Anfang des Forthsystems.
Die folgenden Operationen verarbeiten dagegen absolute 32Bit Adressen.

forthstart (-- laddr)
laddr ist die (erweiterte) Anfangsadresse des Forthsystems im Speicher des Rechners. d.h.
forthstart l@ und
0 @
lesen dieselbe Speicherzelle aus.

l! (16b laddr --) " long-store "
16b werden in den Speicher ab Adresse laddr geschrieben. Im Gegensatz zu ! wird bei dieser Operation die Adresse ab dem Anfang des Speichers und nicht ab dem Anfang des Forthsystems gezählt. Siehe ! und FORTHSTART

l2! (32b laddr --) " long-two-store "
Analog zu L! , jedoch werden 32b geschrieben.

l2@ (laddr -- 32b) " long-two-fetch "
Analog zu L@ , jedoch werden 32b gelesen.

l@ (laddr -- 16b) " long-fetch "
16b wurden aus dem Speicher ab Adresse laddr gelesen. Im Gegensatz zu ! wird bei dieser Operation die Adresse ab dem Anfang des Speichers und nicht ab dem Anfang des Forthsystems gezählt. Siehe @ und FORTHSTART.

lc! (8b laddr --) " long-c-store "
Analog zu L! , jedoch werden 8b geschrieben.

lc@ (8b laddr --) " long-c-fetch "
Analog zu L@ , jedoch werden 8b gelesen.

lcmove (laddr1 laddr2 u --) " long-cmove "
Beginnend bei Adresse laddr1 werden u Bytes zur Adresse laddr2 kopiert. Wenn u Null ist, wird nichts kopiert. Siehe L! und CMOVE .

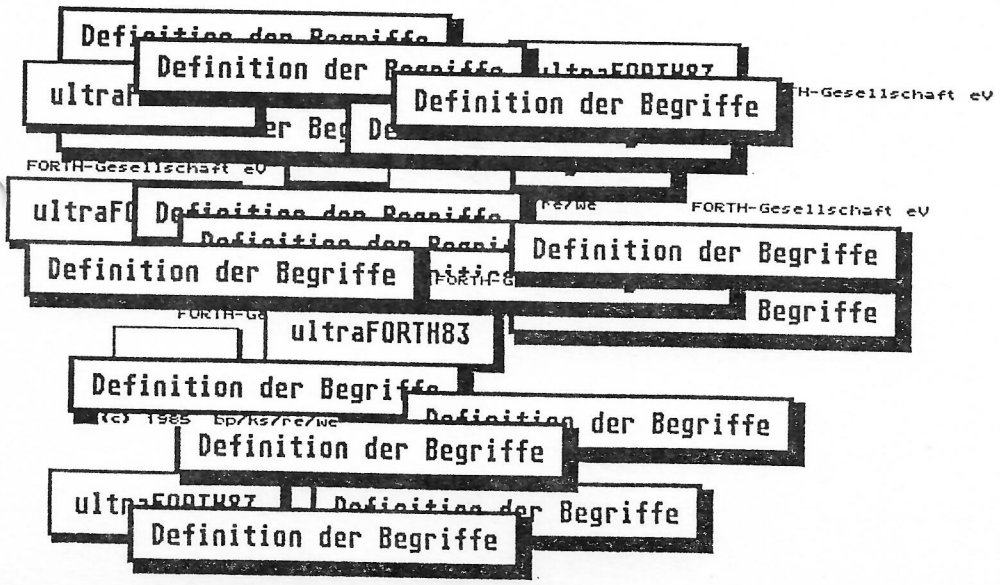
ln+! (w laddr --) " l-n-plus-store "
w wird zu dem Wert in der Adresse laddr addiert. Benutzt die + Operation. Die Summe wird in die Adresse ab laddr geschrieben. Siehe L! und +! .

Wortname	S.	Gruppe	S.
bl	III-52	In/Output	III-51
blank		Diverses	A-41
blk	III-33	Interpreter	III-33
blk/drv	III-40	Massenspeich	III-40
block	III-40	Massenspeich	III-40
bounds	III-28	Kontroll	III-28
buffer	III-40	Massenspeich	III-40
buffers		Relocate	A-45
bye	III-38	Sonstiges	III-38
c		Tools	II-41
c!	III-8	Speicher	III-8
c,	III-22	Dictionary	III-22
c/l	III-52	In/Output	III-51
C:		Fileinterface	A-33
c>0"		Strings	A-47
c@	III-8	Speicher	III-8
c_flag	A-5	GEM	A-4
c_height	A-5	GEM	A-4
c_width	A-5	GEM	A-4
capacity		Fileinterface	A-33
capital	III-16	Strings	III-15
capitalize	III-16	Strings	III-15
caps		Strings	A-47
case?	III-6	Logik/Vergl	III-6
circle	A-11	GEM	A-4
clear	III-22	Dictionary	III-22
clear_disp_list	A-15	GEM	A-4
clearstack	III-12	Stack	III-12
close		Fileinterface	A-33
clrwk	A-5	GEM	A-4
clsvwk	A-5	GEM	A-4
cmove	III-8	Speicher	III-8
cmove>	III-8	Speicher	III-8
Code		Assembler	A-17
col	III-52	In/Output	III-51
cold	III-38	Sonstiges	III-38
compare		Strings	A-47
compile	III-31	Compiler	III-31
con!	III-44	ST-spezifisch	III-44
Constant	III-19	Datentypen	III-18
context	III-25	Vokabular	III-25
contourfill	A-11	GEM	A-4
contrl	A-4	GEM	A-4
convert	III-16	Strings	III-15
convey	III-41	Massenspeich	III-40
copy	III-41	Massenspeich	III-40
copyopaque	A-13	GEM	A-4
core?	III-41	Massenspeich	III-40
count	III-8	Speicher	III-8
cpush		Diverses	A-41
cr	III-52	In/Output	III-51
Create	III-19	Datentypen	III-18
cross	A-12	GEM	A-4
ctoggle	III-8	Speicher	III-8
curdown	A-14	GEM	A-4

Wortname	S.	Gruppe	S.
eeol	A-14	GEM	A-4
eeos	A-14	GEM	A-4
ellarc	A-11	GEM	A-4
ellpie	A-11	GEM	A-4
ELSE	III-28	Kontroll	III-28
emit	III-53	In/Output	III-51
empty	III-23	Dictionary	III-22
empty-buffers	III-41	Massenspeich	III-40
end-trace	III-39	Sonstiges	III-38
enter_cur	A-14	GEM	A-4
eof		Fileinterface	A-33
erase	III-9	Speicher	III-8
error"	III-36	Fehler	III-36
errorhandler	III-36	Fehler	III-36
even	III-4	Arithmetik	III-3
events	A-6	GEM	A-4
evnt_button	A-6	GEM	A-4
evnt_dclick	A-7	GEM	A-4
evnt_keybd	A-6	GEM	A-4
evnt_mesag	A-6	GEM	A-4
evnt_mouse	A-6	GEM	A-4
evnt_multi	A-6	GEM	A-4
evnt_timer	A-6	GEM	A-4
ex_butv	A-13	GEM	A-4
ex_curv	A-13	GEM	A-4
ex_motv	A-13	GEM	A-4
ex_time	A-13	GEM	A-4
execute	III-29	Kontroll	III-28
exit_cur	A-14	GEM	A-4
exor	A-11	GEM	A-4
expect	III-53	In/Output	III-51
extend	III-10	32-Bit-Worte	III-10
false	III-6	Logik/Vergl	III-6
File		Fileinterface	A-33
file?		Fileinterface	A-33
files		Fileinterface	A-33
files"		Fileinterface	A-33
fill	III-9	Speicher	III-8
fillarea	A-11	GEM	A-4
find	III-33	Interpreter	III-33
first	III-41	Massenspeich	III-40
flush	III-41	Massenspeich	III-40
forget	III-23	Dictionary	III-22
form_adv	A-15	GEM	A-4
form_alert	A-8	GEM	A-4
form_center	A-8	GEM	A-4
form_dial	A-8	GEM	A-4
form_do	A-8	GEM	A-4
form_error	A-8	GEM	A-4
Forth	III-25	Vokabular	III-25
forth-83	III-25	Vokabular	III-25
forthfiles		Fileinterface	A-33
freebuffer	III-42	Massenspeich	III-40
from		Fileinterface	A-33
fromfile		Fileinterface	A-33

Wortname	S.	Gruppe	S.
ldump		Tools	II-41
LEAVE	III-29	Kontroll	III-28
limit	III-42	Massenspeich	III-40
list	III-53	In/Output	III-51
Literal	III-31	Compiler	III-31
load	III-34	Interpreter	III-33
loadfile	III-34	Interpreter	III-33
loadfrom		Fileinterface	A-33
lock	III-48	Tasking	III-48
longdash	A-11	GEM	A-4
LOOP	III-29	Kontroll	III-28
m*	III-10	32-Bit-Worte	III-10
m/mod	III-10	32-Bit-Worte	III-10
m_filename	A-15	GEM	A-4
make		Fileinterface	A-33
makedir		Fileinterface	A-33
makefile		Fileinterface	A-33
makeview	III-39	Sonstiges	III-38
malloc		Allocate	A-43
max	III-4	Arithmetik	III-3
mem>scr1	A-13	GEM	A-4
memMFDB1	A-13	GEM	A-4
menu_bar	A-7	GEM	A-4
menu_icheck	A-7	GEM	A-4
menu_ienable	A-7	GEM	A-4
menu_register	A-7	GEM	A-4
menu_text	A-7	GEM	A-4
menu_tnormal	A-7	GEM	A-4
message	A-6	GEM	A-4
meta_extents	A-15	GEM	A-4
mfree		Allocate	A-43
min	III-5	Arithmetik	III-3
mod	III-5	Arithmetik	III-3
mofaddr	A-8	GEM	A-4
more		Fileinterface	A-33
move	III-9	Speicher	III-8
multitask	III-48	Tasking	III-48
n		Tools	II-41
name	III-34	Interpreter	III-33
name>	III-23	Dictionary	III-22
negate	III-5	Arithmetik	III-3
nest		Tools	II-41
next-link	III-39	Sonstiges	III-38
nip	III-12	Stack	III-12
noop	III-39	Sonstiges	III-38
not	III-6	Logik/Vergl	III-6
notfound	III-34	Interpreter	III-33
nullstring?	III-16	Strings	III-15
number	III-16	Strings	III-15
number?	III-17	Strings	III-15
objc_add	A-7	GEM	A-4
objc_change	A-7	GEM	A-4
objc_delete	A-7	GEM	A-4
objc_draw	A-7	GEM	A-4
objc_edit	A-7	GEM	A-4

Wortname	S.	Gruppe	S.
wind_create	A-9	GEM	A-4
wind_delete	A-9	GEM	A-4
wind_find	A-9	GEM	A-4
wind_get	A-9	GEM	A-4
wind_open	A-9	GEM	A-4
wind_set	A-9	GEM	A-4
wind_update	A-9	GEM	A-4
word	III-35	Interpreter	III-33
words	III-26	Vokabular	III-25
write_meta	A-15	GEM	A-4
xor	III-7	Logik/Vergl	III-6
|	III-27	Heap	III-27


Definition der Begriffe
ultraFORTH83
FORTH-Gesellschaft eV
(c) 1985 bp/ks/re/we

Entscheidungskriterien

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

1. Korrekte Funktion - bekannte Einschränkungen, Eindeutigkeit
2. Transportabilität - 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. Ausführungsgeschwindigkeit
7. Kompaktheit
8. Kompilationsgeschwindigkeit
9. Historische Kontinuität
10. Aussprechbarkeit
11. Verständlichkeit - es muß einfach gelehrt werden können

Definition der Begriffe

Es werden im allgemeinen die amerikanischen Begriffe beibehalten, es sei denn, der Begriff ist bereits im Deutschen geläufig. Wird ein deutscher Begriff verwendet, so wird in Klammern der engl. Originalbegriff beigefügt; wird der Originalbegriff beibehalten, so wird in Klammern eine möglichst treffende Übersetzung 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 gehörigen Maschinencode aufzufinden.

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

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

anzeigen (display)
: Der Prozess, ein oder mehrere Zeichen zum aktuellen Ausgabegerät 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 Überlaufsituationen. Dadurch ist es möglich, daß die gleichen Operatoren benutzt werden können, gleichgültig, 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 über Blocknummern im Bereich <0...Anzahl_existenter_Blöcke-1> zugegriffen wird. Die exakte Anzahl der Bytes, die je Zugriff auf den Massenspeicher übertragen werden, und die Übersetzung von Blocknummern in die zugehörige Adresse des Laufwerks und des physikalischen Satzes, sind rechnerabhängig.
Siehe: "Blockpuffer" und "Massenspeicher"

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

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

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

Definition (Definition)
Siehe: "Wortdefinition"

Dictionary (Wörterbuch)
Eine Struktur von Wortdefinitionen, die im Hauptspeicher des Rechners angelegt ist. Sie ist erweiterbar und wächst in Richtung höherer Speicheradressen. Einträge sind in Vokabularen organisiert, so daß die Benutzung von Synonymen möglich ist, d.h. gleiche Namen können, in verschiedenen Vokabularen enthalten, vollkommen verschiedene Funktionen auslösen.
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 nächstkleinere ganze Zahl gerundet.
Bemerkung: Ausgenommen von Fehlern durch Überlauf 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 rechnerabhängig.

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

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

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

Floor, arithmetic
Z sei eine reelle Zahl. Dann ist der Floor von Z die größte 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 gehörende Aktion des Computers beschreibt - die Beschreibung der Semantik des Wortes.

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

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

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

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

Massenspeicher (mass storage)
Speicher, der ausserhalb des durch FORTH adressierbaren Bereiches liegen kann. Auf den Massenspeicher wird in Form von 1024byte grossen Blöcken zugegriffen. Auf einen Block

kann innerhalb des Forth-Adressbereichs in einem Blockpuffer zugegriffen werden. Wenn ein Block als verändert (UPDATE) gekennzeichnet ist, wird er letztendlich wieder auf den Massenspeicher zurückgeschrieben.

Programm (program)
Eine vollständige 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 zugeführt wird. Der Quelltext kommt üblicherweise von der aktuellen Eingabeeinheit (über den Texteingabepuffer) oder dem Massenspeicher (über einen Blockpuffer). BLK, >IN, TIB und #TIB charakterisieren den Quelltext. Worte, die BLK, >IN, TIB oder #TIB benutzen und/oder verändern, sind dafür 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, können 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 für Quelltextinterpretation während des Ladens eines Blocks.

Suchreihenfolge (search order)
Eine Spezifikation der Reihenfolge, in der ausgewählte 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 Ausführung eines Vokabularnamens macht es zum ersten Vokabular in der Suchreihenfolge, wobei das Vokabular, das vorher als erstes durchsucht worden war, verdrängt wird. Auf dieses erste Vokabular folgt, soweit spezifiziert, der feste Teil der Suchreihenfolge, der danach durchsucht wird. Die Ausführung von ALSO übernimmt 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 hauptsächlich zum Ablegen von Zwischenergebnissen während des Ausführens von Wortdefinitionen benutzt. Daten auf dem Stack können 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 (Rücksprungstapel)
Eine "Zuletzt-rein, Zuerst-raus" Struktur, die hauptsächlich Adressen von Wortdefinitionen enthält, deren Ausführung durch den Adressinterpreter noch nicht beendet ist. Wenn eine Wortdefinition eine andere Wortdefinition aufruft, so wird die Rücksprungadresse auf dem Returnstack abgelegt.
Der Returnstack kann zeitweise auch für die Ablage anderer Daten benutzt werden.

String, counted (abgezählte Zeichenkette)
Eine Hintereinanderfolge von 8bit Daten, die im Speicher durch ihre niedrigste Adresse charakterisiert wird. Das Byte an dieser Adresse enthält einen Zahlenwert im Bereich <0...255>, der die Anzahl der zu diesem String gehörigen 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 Länge 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 für 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, sodaß die Worte, die diese benutzen, für 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 können mehrere Definitionen mit dem gleichen Namen existieren; diesen Vorgang nennt man redefinieren. Wird das Vokabular nach einem Namen durchsucht, so wird die jüngste 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. Wahrheitswerte, 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 können.

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

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

Wortdefinition (word definition)
Eine mit einem Namen versehene, ausführbare 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 größeren Feldes existieren, so sind die höherwertigen Bits auf Null gesetzt. 16bit Zahlen sind im Speicher so abgelegt, dass sie in zwei benachbarten Byteadressen enthalten sind. Die Bytereihenfolge ist rechnerabhängig. Doppeltgenaue Zahlen (32bit) werden auf dem Stack so abgelegt, daß die höherwertigen 16bit (mit dem Vorzeichenbit) oben liegen. Die Adresse der niederwertigen 16bit ist um zwei größer als die Adresse der höherwertigen 16bit, wenn die Zahl im Speicher abgelegt ist.
Siehe: "Arithmetik, 2er-komplement" und "Zahlentypen"

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

Zahlenausgabe, freiformatiert (free field format)
Zahlen werden in Abhängigkeit von BASE umgewandelt und ohne führende 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 für jedes Bit um eine Zweierpotenz erhöht. Für eine Zahl ohne Vorzeichen ist das am weitesten links stehende Bit in diese Bewertung eingeschlossen, sodaß für eine solche 16bit Zahl das ganz linke Bit den Wert 32.768 hat. Für Zahlen mit Vorzeichen wird die Bewertung des ganz linken Bits negiert, sodaß es bei einer 16bit Zahl den Wert -32.768 hat. Diese Art der Wertung für 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 Binärzahlen geführt und extern durch graphische Zeichen des ASCII Zeichensatzes dargestellt. Die Umwandlung zwischen der internen und externen Form wird unter Beachtung des Wertes von BASE durchgeführt, 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 aufwärts 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 für die gerade umzuwandelnde Zahl dadurch umgangen werden, daß den Ziffern ein "Zahlenbasisprefix" vorangestellt wird. Dabei wird durch das Zeichen "%" die Basis vorübergehend 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. Enthält die Zahl ein Komma oder einen Punkt, so wird sie als 32bit Zahl umgewandelt.

Zeichen (character)

Eine 8bit Zahl, deren Bedeutung durch den ASCII-Standard festgelegt ist. Wenn es in einem größeren Feld gespeichert ist, so sind die höherwertigen Bits auf Null gesetzt.

Zustand (mode)

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