Liebe Forth-Freunde,

es war für mich immer unhandlich mit dicken Dokumentationen zu arbeiten. Wenn ich eine Sprache inhaltlich erfaßt und verstanden hatte, brauchte ich meist nur die Syntax. IBM und Robotron haben zu den Rechnern Faltkarten mit konzentrierten Befehlsübersichten herausgegeben. Konzentriert auf einige wenige A4-quer-Karten standen die Anweisungen und Definitionen in drei Spalten handlich für die Arbeit aufbereitet. Im Zick-Zack gefaltet war sie sehr handlich.

So eine Faltkarte möchte ich Euch hier vorstellen. Dieser Entwurf vom 1.10.97 ist sicher noch nicht perfekt. Grundlage ist der ANS-Standard ANSI X3.215-1994. Bis auf Floatings sollten alle Standardworte enthalten sein. Die letzte Spalte zeigt die Zuordnung im Standard zu den Wortgruppen.

Die Einteilung der Worte erfolgt nach Funktion. Manchmal ist es sehr subjektiv. Und dann habe ich anglisiert, weil sich alles so schön kurz darstellt. Wenn mein Job und meine Familie mitspielen, versuche ich auch noch eine deutsche Version.

Diskutiert bitte mit mir über Inhalt und Form und natürlich über Fehler, die sicher noch drin sind.

Tschüssikowski - Ekkehard (eks@velten.forth-ev.de)

Begriff	Erklärung
current Voc	means compiling Vocabulary
wordlist	means one vocabulary identified by ist wid
dictionary	means all vocabularies with their associated data
S:	means DataStackEffekts
R:	means ReturnStackEffekts
C:	means ControlflowStackEffekts
F:	means FloatingPointStackEffekts
a-ad	aligned address. Often addresses a word of 16 bit.
c-ad	character aligned address. Often addresses a byte of 8 bit
AdrUnit	is basic adressable unit of system memory. May address a byte of 8 bits.
DataSpace	memoryarea including wordlists etc.
SearchOrder	list of vocabularies identified by some wid's.
Basic SearchOrder	SearchOrder just after Start of the ForthSystem. Example: widF
i/o	Input/Output
fac	FileAccess
trm	terminal (User-i/o-Device)
str	Charakter-String; Counted String.
sys	system
voc	vocabulary
deb	debugging
def	definition
dfw	definitionWord
cfl	controlFlow
ari	aeithmetik
log	logical
sgl	single
dbl	double
dst	dataStack
kbd	keyboard (UserInputDevice)
dev	means UserOutputDevice
dsp	display (UserOutputDevice)
var	variable
con	constant
wid	identifier for a Voc
widF	identifier for FORTH-Voc
widA	identifier for ASSEMBLER-Voc
widE	identifier for EDITOR-Voc

ForthWord	StackEffects	What Word Is Doing	ANS
Arithmetik	**Addition**		
+	(n1 n2 -- n3)	n3 = n1 + n2	cor
1+	(n1 -- n2)	n2 = n1 + 1	cor
+!	(n a-ad --)	add n to 1-cell-contens at a-ad	cor
D+	(d1 d2 -- d3)	d3 = d1 + d2	dbl
M+	(d1 n -- d2)	d2 = d1 + n	dbl
Arithmetik	**Subtraktion**		
-	(n1 n2 -- n3)	n3 = n1 - n2	cor
1-	(n1 -- n2)	n2 = n1 - 1	cor
D-	(d1 d2 -- d3)	d3 = d1 - d2	dbl
Arithmetik	**Multiplikation**		
*	(n1 n2 -- n3)	n3 = n1 * n2	cor
M*	(n1 n2 -- d)	d = n1 * n2	cor
UM*	(u1 u2 -- ud)	ud = u1 * u2	cor
Arithmetik	**Division**		
/	(n1 n2 -- n3)	n3 = n1 / n2 (ignore remainder)	cor
/MOD	(n1 n2 -- n3 n4)	n4 = n1 / n2 (n3 is the remainder)	cor
UM/MOD	(ud u1 -- u2 u3)	u3 = ud / u1 (u2 is the remainder)	cor
MOD	(n1 n2 -- n3)	n3 is the remainder of n1 / n2	cor
FM/MOD	(d1 n1 -- n2 n3)	n3 = d1 / n1 (n2 - remainder of floored division)	cor
SM/REM	(d1 n1 -- n2 n3)	n3 = d1 / n1(n2 - remainder of symmetric division)	cor
Arithmetik	**Logic**		
2*	(x1 -- x2)	shift right one bit (x2 = x1 * 2)	cor
2/	(x1 -- x2)	shift left one bit (x2 = x1 / 2)	cor
D2*	(xd1 -- xd2)	xd2 = xd1 * 2 (shift left)	dbl
D2/	(xd1 -- xd2)	xd2 = xd1 / 2 (shift right)	dbl
Arithmetik	**Scalation**		
*/	(n1 n2 n3 -- n4)	n4 = (n1*n2)/n3 (interm. result is dbl-cell; ignore remainder)	cor
*/MOD	(n1 n2 n3 -- -- n4 n5)	n5=(n1*n2)/n3 (n4 - remainder; intermediate result is dbl-cell)	cor
M*/	(d1 n1 +n2 -- d2)	d2 = (d1 * n1) / +n2 (intermediate result is triple-cell)	dbl
Arithmetik	**Converting**		
NEGATE	(n1 -- n2)	n2 = -n1	cor
DNEGATE	(d1 -- d2)	d2 = -d1	dbl
ABS	(n -- u)	u = absolute of n	cor
DABS	(d -- ud)	ud is absolute value of d	dbl
Arithmetik	**System/Variable**		
BASE	(-- a-ad)	contains the current radix (2..36)	cor
DECIMAL	(--)	set radix in BASE to ten	cor
HEX	(--)	set radix in BASE to sixteen	cor-e

Memory			
!	1(x a-ad --)	store x at a-ad	cor
@	1(a-ad -- x)	fetch cell from a-ad	cor
2!	2(x1 x2 a-ad --)	store double-cell at a-ad	cor
2@	2(a-ad -- x1 x2)	fetch double-cell from a-ad	cor
C!	3(ch c-ad --)	store ch at c-ad	cor
C@	3(c-ad -- ch)	fetch one chr from c-ad	cor

ForthWord	StackEffects	What Word Is Doing	ANS
Logic	**Single**		
FALSE	(-- false)	get falseFlag (not all bits set)	cor-e
TRUE	(-- true)	get trueFlag (all bits in cell set)	cor-e
<	(n1 n2 -- fl)	true if n1 < n2, false otherwise	cor
=	(x1 x2 -- fl)	true if n1 = n2, false otherwise	cor
>	(n1 n2 -- fl)	true if n1 > n2, false otherwise	cor
<>	(x1 x2 -- fl)	true if x1<>x2, false otherwise	cor-e
0<	(n -- fl)	true if n<0, false otherwise	cor
0=	(x -- fl)	true if n=0, false otherwise	cor
U<	(u1 u2 -- fl)	true if u1 < u2, false otherwise	cor
0>	(n -- fl)	true if n>0, false otherwise	cor-e
U>	(u1 u2 -- fl)	true if u1 > u2, false otherwise	cor-e
0<>	(x -- fl)	true if x<>0, false otherwise	cor-e
AND	(x1 x2 -- x3)	x3 is LogicalAnd of x1 and x2	cor
INVERT	(x1 -- x2)	invert all bits of x1	cor
OR	(x1 x2 -- x3)	x3 is LogicalOr of x1 and x2	cor
XOR	(x1 x2 -- x3)	x3 is ExclusiveOr of x1and x2	cor
MAX	(n1 n2 -- n3)	n3 = greater of x1, x2	cor
MIN	(n1 n2 -- n3)	n3 = lesser of x1, x2	cor
WITHIN	(n1 n2 n3 -- fl)	true if n2>=n1>n3, false else	cor-e
DMAX	(d1 d2 -- d3)	d3 = max(d1, d2)	dbl
DMIN	(d1 d2 -- d3)	d3 = min(d1, d2)	dbl
LSHIFT	(x1 u -- x2)	shift left (*2) u times bits of x1	cor
RSHIFT	(x1 u -- x2)	shift right (/2) u times bits of x1	cor
Logic	**Double**		
D<	(d1 d2 -- fl)	true if d1 < d2, else otherwise	dbl
D=	(xd1 xd2 -- fl)	true if d1 = d2, else otherwise	dbl
D0<	(d -- fl)	true - d < 0, false otherwise	dbl
D0=	(xd -- fl)	true if xd = 0, false otherwise	dbl
DU<	(ud1 ud2 -- fl)	true if ud1 < ud2, false otherwise	dbl-e

DataStacks	SingleNumbers		
?DUP	(x -- 0 | x x)	duplicate only if not zero	cor
DROP	(x --)	delete x from DataStack	cor
DUP	(x -- x x)	double x on DataStack	cor
OVER	(x1 x2--x1 x2 x1)	copy second StackItem on Stack	cor
ROT	(x1 x2 x3 -- -- x2 x3 x1)	rotate top three items last to 1st	cor
SWAP	(x1 x2 -- x2 x1)	exchange the top two StackItems	cor
NIP	(x1 x2 -- x2)	drop second item of DataStack	cor-e
PICK	(xu ... x1 x0 u -- - - ... x1 x0 xu)	copy the u-th item of DataStack on top	cor-e
ROLL	(xu xu-1 ... x0 u-- -- xu-1 ... x0 xu)	move u-th StackItem on top	cor-e
TUCK	(x1 x2 -- x2 x1 x2)	copy the top item unter 2nd item	cor-e
DataStacks	**DoubleNumbers**		
2DROP	(d --)	drop cell-pair from stack	cor
2DUP	(d -- d d)	duplicate cell-pair on stack	cor
2OVER	(d1 d2--d1 d2 d1)	copy second cell-pair on the stack	cor
2SWAP	(d1 d2 --d2 d1)	exchange DblCell-pair on stack	cor
2ROT	(d1 d2 d3 -- -- d2 d3 d1)	rotate d1 to Top of DataStack	dbl-e

ForthWord	StackEffects	What Word Is Doing	ANS
CtrlFlow	**Alternation**		
IF	Co:(C: -- orig) Ru:(x --)	save place for RunTimeBranch if false, continue at noted Loc.	cor
ELSE	Co:(C: orig1 -- orig2) Ru:(--)	note Loc. after ELSE in orig1, save place for RunTimeBranch continue execution at noted Loc.	cor
THEN	Co:(C: orig --) Ru:(--)	note Loc. of THEN in orig continue execution at next Loc.	cor
CtrlFlow	***Looping***		
DO	Co:(C: -- do-sys) Ru:(n1 n2 --) (R: -- loop-sys)	note next Loc. for RunTimeBr. note index and limit as loop-sys	cor
?DO	Co:(C: -- do-sys) Ru:(n1 n2--)(R: -- | loop-sys)	note next Loc. for RunTimeBr. x1 = x2: continue execution at Loc. given by do-sys; otherwise note index and limit as loop-sys	cor-e
LOOP	Co:(C: do-sys --) Ru:(--) (R:loop-sys1-- | loop-sys2)	save do-sys-Loc. for RunTimeBr. add 1 to loop-counter. if limit n1 is reached, continue execution at Loc. after +LOOP, otherwise at Loc. given by do-sys	cor
+LOOP	Co:(C: do-sys --) Ru:(n --) (R:loop-sys1-- | loop-sys2)	save do-sys-Loc. for RunTimeBr. add n to loop-counter; if limit n1 is reached, continue execution at Loc. after +LOOP, otherwise at Loc. given by do-sys	cor
I	Ex:(-- n)	n is inner loop-indcx	cor
J	Ex:(-- n)	n is next outer loop-index	cor
LEAVE	Ex:(--) (R: loop-sys --)	continue execution at Loc. noted in loop-sys	cor
UNLOOP	Ex:(--) (R: loop-sys --)	drop loop-sys of current Do- Loop (nessesary for each Do- Loop before EXITing)	cor
CtrlFlow	**Case**		
CASE	Co:(C: -- case-sys) Ru:(--)	save place for RunTimeBranch continue execution at next Loc.	cor-e
OF	Co:(C: -- of-sys) Ru:(x1 x2 -- | x1)	save place for RunTimeBranch x1 = x2: 2drop and continue execution at next Loc. else drop and continue execution at Loc. noted in of-sys	cor-e
ENDOF	Co:(C: case-sys of-sys -- case-sys) Ru:(--)	note Loc. after ENDOF in of-sys continue execution at noted Loc.	cor-e
ENDCASE	Co:(C: case-sys --) Ru:(x --)	note Loc. of ENDCASE drop x and continue execution at Loc. after ENDCASE	cor-e

DataStacks	Returnstack		
>R	Ex:(x --) (R: -- x)	store x on ReturnStack	cor
R>	Ex:(-- x) (R: x --)	get x from ReturnStack	cor
R@	Ex:(-- x)(R: x -- x)	copy x from ReturnStack	cor
2>R	Ex:(d --) (R: -- d)	store DblCell on ReturnStack	cor-e
2R>	Ex:(-- d) (R: d --)	get DblCell from ReturnStack	cor-e
2R@	Ex:(-- d) (R: d -- d)	copy DblCell from ReturnStack	cor-e

ForthWord	StackEffects	What Word Is Doing	ANS
CtrlFlow	***Cyklus***		
BEGIN	Co:(C: -- dest) Ru:(--)	note next Loc. for a RunTimeBr. continue execution at next Loc.	cor
WHILE	Co:(C: dest -- -- orig dest) Ru:(x --)	save place for RunTimeBranch if false, continue execution at Loc. noted in orig	cor
UNTIL	Co:(C: dest --) Ru:(x --)	save Loc. in dest for RunTimeBr. if false, continue execution at Loc. given by dest	cor
REPEAT	Co:(orig dest --) Ru:(--)	save dest-Loc. for RunTimeBr. and note next Loc. in orig continue execution at dest-Loc.	cor
AGAIN	Co:(C: dest --) Ru:(--)	save dest-Loc. for RunTimeBr. continue execution at dest-Loc.	cor-e
EXIT	Ex:(--) (R: nest-sys --)	return to calling word via nest- sys	cor
RECURSE	Co:(--)	append the ExecutionSemantics of the current definition	cor
CtrlFlow	**System**		
BYE	(--)	leave 4th to Host-System	too-e
ABORT	(i*x --)(R: j*x --)	empty stacks and perform QUIT without any massage (: ABORT -1 THROW ;)	cor exc- e
ABORT"	Co:("ccc<quote>"--) Ru:(i*x x1 -- | i*x) (R: j*x -- | j*x)	note ccc for RunTime-Use if any bit of x1 is one, display ccc, generate a abort-sequence and perform the functions of ABORT, otherwise drop x1	cor exc- e
[IF]	Ex:(fl | fl "<sps>na ... " --)	ICompiling IF - change ControlFlow during compilation	too-e
[ELSE]	Ex:("<sps>na ... "--)	ICompiling ELSE - change ControlFlow during compilation	too-e
[THEN]	Ex:(--)	ICompiling THEN - change ControlFlow during compilation	too-e
AHEAD	Co:(C: -- orig) Ru:(--)	save place orig for RunTimeBr. continue execution at orig-Loc.	too-e
CATCH	(i*x xt -- -- j*x 0 | i*x n)	is like saving all nessisary information and make a interupt-call of xt EXECUTE	exc
THROW	(k*x n -- -- k*x | i*x n)	return from interupt-call if called via CATCH, else it works DataStack-depended	exc
CS-PICK	Ex:(C: elu ... el0-- --elu ... el0 elu) (S: u --)	pick element u of ControlFlowStack	too-e
CS-ROLL	Ex:(C: elu elu-1 ... ... el0 -- elu-1 ... ... el0 elu)(S: u --)	roll u elements of ControlFlowStack	too-e

Diese Faltkarte ist als **FALTKAR4.DOC** ist auf der **KBBS** 0431-5339898 (8N1) im file **forth/vd/vdl973_4.zip** zu finden.

ForthWord	StackEffects	What Word Is Doing	ANS
I/O	***UserOutput***		
.	(n --)	display n	cor
U.	(u --)	display u	cor
D.	(d --)	display d	dbl
.R	(n1 n2 --)	display n1 right in an n2-chr- field	cor-e
U.R	(u n --)	display u right in an n-chr-field	cor-e
D.R	(d n --)	display d right in a n-chr-Field	dbl
EMIT	(x --)	display x	cor
."	Co:("ccc<quote>"--) Ru:(--)	note ccc for RunTimeUse display ccc	cor cor
TYPE	(c-ad u --)	display CharString (c-ad u)	cor
SPACE	(--)	display one space	cor
SPACES	(n --)	display n spaces	cor
CR	(--)	move cursor to start of next line	cor
PAGE	(--)	clear for new page	fac
AT-XY	*(u1 u2 --)*	*posit Cursor on to Position u1, u2*	*fac*
I/O	**FileAccess**		
BIN	(fam1 -- fam2)	change fam1 to fam2	fil
CLOSE-FILE	(fid -- ior)	close file fid	fil
CREATE-FILE	(c-ad u fam -- -- fid ior)	create file with filename (c-ad u) for FileAccessMethod fam	fil
DELETE-FILE	(c-ad u -- ior)	delete file identified by filename *in CharString (c-ad u)*	fil
FILE-POSITION	(fid -- ud ior)	get Input/Output-Position of File identified by fid	fil
FILE-SIZE	(fid -- ud ior)	get Size of File fid	fil
OPEN-FILE	(c-ad u fam -- -- fid ior)	open file (c-ad u) with FileAccesMethod fam	fil
R/O	(-- fam)	get FileAccessMethod read only	fil
R/W	(-- fam)	get FileAccessMethod read/write	fil
READ-FILE	*(c-ad u1 fid -- -- u2 ior)*	*read at most u1 AdressUnits to* CharString at c-ad (ignore eol)	*fil*
READ-LINE	(c-ad u1 fid-- -- u2 fl ior)	read at most u1 Bytes until eol (including) or u1 Byte (without eol) to CharString at c-ad	fil
REPOSITION-FILE	(ud fid -- ior)	set Input/Output-Position of file fid to ud	fil
RESIZE-FILE	(ud fid -- ior)	change Size of file fid to ud	fil
W/O	*(-- fam)*	*get FileAccessMethod write only*	*fil*
WRITE-FILE	(c-ad u fid -- ior)	write String (c-ad u) to file fid (starting at aktual I/O-Position)	fil
WRITE-LINE	(c-ad u fid -- ior)	write String (c-ad u) to file fid followed by eol (start at aktual I/O-Position)	fil
FILE-STATUS	(c-ad u -- x ior)	get status of file fid	fil-e
FLUSH-FILE	(fid -- ior)	save all changes into file fid	fil-e
RENAME-FILE	*(c-ad1 u1 c-ad2 u2 -- ior)*	*rename File (c-ad1 u1) to File* (c-ad2 u2)	*fil-e*

ForthWord	StackEffects	What Word Is Doing	ANS
I/O	**Keyboard**		
KEY	(-- ch)	get chr by keyboard	cor
KEY?	(-- fl)	true - chr available, false else	fac
EKEY	(-- u)	*get one KeyBoardEvent*	fac-e
EKEY>CHAR	(u -- u false | ch true)	ch true - u is a chr, u false - else	fac-e
EKEY?	(-- fl)	true - KeyBoardEvent, false else	fac-e
EMIT?	(-- fl)	*true if UserOutputDevice is* ready, false else	fac-e
I/O	**System**		
BLOCK	(u -- a-ad)	transfer Block u to Starage from *MassStorage (a-ad is adr of the* first chr of block)	blk
SOURCE	(-- c-ad u)	get Loc. of next possible UserInputBuffer	cor
.(	Ex:(“ccc<paren>” --)	display ccc until ”)” (f.c. Compiling Commment)	cor-e
EXPECT	(c-ad +n --)	store a UserInputString of at most +n chrs until eol at c-ad and its length in SPAN (word is concession to older 4th)	cor-e
BUFFER	(u -- a-ad)	get adr of next usable Buffer for block u (without transfer)	blk
EVALUATE	(i*x c-ad u -- j*x)	make (c-ad u) the InpuBuffer; set BLK and interprete (c-ad u)	cor blk
FLUSH	(--)	transfer updated blocks to MassStorage; mark all Buffers as unmodified and unassigned	blk
SAVE-BUFFERS	(--)	transfer updated blocks to MassStorage and mark all Buffers as unmodified	blk
UPDATE	(--)	mark current Buffer as modified	blk
EMPTY-BUFFERS	(--)	mark all Buffers as unassigned without transfer to MassStorage	blk-e
ACCEPT	(c-ad +n1 -- +n2)	store a UserInputString of at most +n1 chrs until eol (+n2 is length of geted string)	cor
BLK	(-- a-ad)	contains blocknumber of MassStorage just interpreted (if 0 look for SOURCE-ID)	blk
SCR	(-- a-ad)	contains number of last listed blk	blk-e
\	Ex:(”ccc<eol>”--)	ignore ccc to eol (interpreting)	cor-e blk-e
(	Ex:(”ccc<paren>”--)	comment eot or ”)” (ignore eol)	cor fil

Der Standard **ANSI X3.12-1994** findet sich auf
ftp://ftp.uu.net/vendor/minerva/uathena.htm

ForthWord	StackEffects	What Word Is Doing	ANS
Vocabulary			
WORDS	(--)	display words in 1st voc of SearchOrder	too
ASSEMBLER	(--)	make 1st SearchOrderVoc the ASSEMBLER-Voc	too-e
EDITOR	(--)	make 1st SearchOrderVoc the EDITOR-Voc	too-e
FORGET	(”<sps>na” --)	delete na and following words in CompVoc	too-e
DEFINITIONS	(--)	make CompVoc same as 1st Voc of the SearchOrder	sea
FIND	(c-ad -- c-ad 0 | xt 1 | xt -1)	Find Word in SearchOrder (0 - not found, 1 - immediate, -1 - otherwise)	sea
FORTH-WORDLIST	(-- widF)	get Ident of ForthVoc	sea
GET-CURRENT	(-- wid)	get Ident of CompilingVoc	sea
GET-ORDER	(--widn ...wid1 n)	get SearchOrder (n is number of Elements)	sea
SEARCH-WORDLIST	(c-ad u wid -- -- 0 | xt 1 | xt -1)	find word (c-ad u) in Voc wid (0 - not found, 1 - immediate, -1 - otherwise)	sea
SET-CURRENT	(wid --)	make Voc wid the CompilingVoc	sea
SET-ORDER	(widn ...wid1 n --)	set SearchOrder to wid1 .... widn	sea
WORDLIST	(-- wid)	wid is Ident of a new empty Voc	sea
ALSO	(--)	double first Item of SearchOrder	sea-e
FORTH	(--)	change 1st SearchOrderVoc to FORTH-Voc	sea-e
ONLY	(--)	set SearchOrder to BasicOrder	sea-e
ORDER	(--)	Display SearchOrder + CompVoc	sea-e
PREVIOUS	(--)	Delete first Item of SearchOrder	sea-e
Vocabulary	**System**		
ALLOT	(n --)	add n (cells) to DataSpacePointer	cor
'	(”<sps>na” -- xt)	get xt of na	cor
,	(x --)	*append x at HERE; reserve it*	cor
C,	(ch --)	append one ch at HERE; res. it	cor
FIND	(c-ad -- c-ad 0 | xt 1 | xt 1)	find word (c-ad) in SearchOrder (c-ad 0 - definition not found, *xt 1 - immediate, xt -1 otherwise*)	cor
HERE	(-- ad)	get DataSpacePointer on stack	cor
>BODY	(xt -- a-ad)	get DataFieldAdr of xt	cor
COMPILE,	Ex:(xt --)	append xt to current definition	cor-e

Debugging			
DEPTH	(-- +n)	+n is the depth of Stack (in cells)	cor
.S	(--)	display contens of DataStack	too
?	(a-ad --)	display value at a-ad	too
DUMP	(ad u --)	display contens of u AdrUnits beginnig with ad	too
SEE	(”<sps>na” --)	display SourceCode of na	too

ForthWord	StackEffects	What Word Is Doing	ANS
Strings			
-TRAILING	(c-ad u1--c-ad u2)	cut last sps of CharString1	str
/STRING	(c-ad1 u1 n -- -- c-ad2 u2)	cut leading n chrs of CharString1	str
FILL	(c-ad u ch --)	fill (c-ad u) with u-times ch	cor
BLANK	(c-ad u --)	fill CharString with sps	str
ERASE	(ad u --)	clear all bits in (ad u) (AdrUnits)	cor-e
MOVE	(ad1 ad2 u --)	move blockwise (ad1 u) to ad2	cor
CMOVE	(c-ad1 c-ad2 u --)	Move (c-ad1 u) to c-ad2 (u AdrUnits; first to last)	str
CMOVE>	(c-ad1 c-ad2 u --)	Move (c-ad1 u) to c-ad2 (u AdrUnits; last to first)	str
COMPARE	(c-ad1 u1 c-ad2 u2 -- -- 0 | 1 | -1)	compare Strings (first to last) (0 - identical and u1=u2, -1 - identical in Min(u1; u2) chrs, 1 - otherwise)	str
SEARCH	(c-ad1 u1 c-ad2 u2 -- -- c-ad3 u3 fl)	search String2 in String1 (false - not found (String3 is String1), true - found at c-ad3 (u3 remaining chrs)	str
COUNT	(c-ad1 -- c-ad2 u)	make CountedString (c-ad1) available as String (c-ad2 u)	cor

Converting	Numbers/Strings		
S>D	(n -- d)	convert n to d	cor
D>S	(d -- n)	change d to n (looks like DROP)	dbl
<#	(--)	start converting a dbl-cell number	cor
#	(ud1 -- ud2)	convert one digit of ud1 (ud2 is the quotient of ud1/BASE, digit is the remainder)	cor
#S	(ud1 -- ud2)	convert digits until ud2 is zero	cor
SIGN	(n --)	if n is negativ, add chr for minus to <#string	cor
HOLD	(ch --)	append ch at converted string	cor
#>	(xd -- c-ad u)	*end converting and get (c-ad u)* CharString of convertet digits	cor
CONVERT	(ud1 c-ad1 -- -- ud2 c-ad2)	add digits of CountedString c-ad1 to ud2 until no-number-char (ud2 = ud1 * BASE + digit); c-ad2 is rest of String c-ad1; (word is concession to older 4th)	cor-e
>NUMBER	(ud1 c-ad1 u1 -- -- ud2 c-ad2 u2)	convert digits of CharStringstring (c-ad1 u1) to ud2 starting with ud1 until not convertible charakter (CharStringstring (c-ad2 u2) is the rest of CharStringstring1 (c-ad1 u1))	cor

ForthWord	StackEffects	What Word Is Doing	ANS
System			
CHAR	("<sps>na" -- ch)	get value of first ch of na	cor
ENVIRONMENT?	(c-ad u -- false | i*x true)	ask for environment (c-ad u), if defined: parameters and true, otherwise false	cor
EXECUTE	(i*x xt -- j*x)	execute word given by its xt	cor
IMMEDIATE	(--)	make last definition immediate	cor
LIST	(u --)	display block u	blk-e
WORD	(ch "<chs>ccc<ch>" -- c-ad)	store "ccc<ch>" in TransientArea as CountedString (c-ad)	cor
PARSE	(ch "ccc<ch>" -- c-ad u)	locate ccc in UserInpBuffer delimited by ch	cor-e
POSTPONE	Co:("<sps>na" --)	append the CompSemantics of na to current definition	cor
QUIT	(--) (R: i*x --)	reset InpSource, interprete UserInput linewise	cor
[	Ex:(--)	make STATE to interprete	cor
]	(--)	make STATE to compile	cor
[']	Co:("<sps>na" --) Ru:(-- xt)	note na for RuntimeUse get xt of noted na on DataStack	cor
[CHAR]	Co:("<sps>na" --) Ru:(-- ch)	note na for RuntimeUse get the first chr of noted na	cor
[COMPILE]	Co:("<sps>na" --)	append ExSemantics of na to current definition	cor-e
PAD	(-- c-ad)	get adress of transient datastore	cor-e
QUERY	(--)	reset UserInputSource (word is concession to older 4th)	cor-e
REFILL	(-- fl)	load next line of UserInputSource (true - input o.k., false - else)	cor-e blk-e fil-e
RESTORE-INPUT	(xn ... x1 n -- fl)	restore InputSourceSpecifikation saved by SAVE-INPUT (true - not possible; false - otherwise)	cor-e
SAVE-INPUT	(-- xn ... x1 n)	save InputSourceSpecifikation for later use (RESTORE-INPUT)	cor-e
UNUSED	(-- u)	u AdrUnits free from HERE	cor-e
LOAD	(i*x u -- j*x)	start interpreting at block u of UserInputSource (end depends on BlockContens)	blk
THRU	(i*x u1 u2 -- j*x)	LOAD block u1 to u2 (end depends on BlockContens)	blk-e
MS	(u --)	wait for u milliseconds	fac-e
TIME&DATE	(-- +n1 ... +n6)	get aktual date and time (jj:mm:dd is +n6:+n5:+n4; hh:mm:ss is +n3:+n2:+n1)	fac-e
INCLUDE-FILE	(i*x fid -- j*x)	save InputParameters, interpret contents of file fid linewise until eof and restore InputParameters	fil
INCLUDED	(i*x c-ad u -- j*x)	save InpParam., interpret contents of file identified by filename in CharString (c-ad u) linewise until eof and restore InpParam.	fil

ForthWord	StackEffects	What Word Is Doing	ANS
System	**Adressing**		
ALIGN	(--)	align DataSpacePointer	cor
ALIGNED	(ad -- a-ad)	ad is aligned to a-ad	cor
CELL+	(a-ad1 -- a-ad2)	add AdrUnits of a Cell to a-ad1	cor
CELLS	(n1 -- n2)	n2 - AdrUnits of n1 Cells	cor
CHAR+	(c-ad1 -- c-ad2)	add AdrUnits of a chr to a-ad1	cor
CHARS	(n1 -- n2)	n2 - AdrUnits of n1 Characters	cor
ALLOCATE	(u -- a-ad ior)	Allocate u AdrUnits of Memory to ForthSystem	mem
FREE	(a-ad -- ior)	Free allocated Space	mem
RESIZE	(a-ad1 u -- a-ad2 ior)	Change Space a-ad1 to u Units from a-ad2	mem
System	**Literals**		
C"	Co:("ccc<quote>" --) Ru:(-- c-ad)	note ccc for RuntimeUse get Counted String containing noted ccc	cor-e
SLITERAL	Co:(c-ad1 u --) Ru:(-- c-ad2 u)	note CharString for RuntimeUse return the noted CharString	str
S"	Co:("ccc<quote>" --) Ru:(-- c-ad u) In:("ccc<quote>" -- c-ad u)	note ccc for RuntimeUse get noted ccc (c-ad u) note ccc in CharString (c-ad u)	cor fil fil
System	**Const./Variables**		
BL	(-- ch)	get value of chr space	cor
SOURCE-ID	(-- 0 | -1 | fid)	get InpSource (0 - UserInput-Device, -1 via EVALUATE, fid - FileIdent of Textfile)	cor-e fil
>IN	(-- a-ad)	contains InputBufferOffset	cor
STATE	(-- a-ad)	contains the compilation-state	cor
#TIB	(-- a-ad)	contains the length TrmInpBuffer	cor-e
SPAN	(-- a-ad)	contains length of EXPECTed str (word is concession to older 4th)	cor-e
TIB	(-- c-ad)	get Adr of TrmInputBuffer	cor-e
STATE	(-- a-ad)	contains the compilation-state	too-c

Definitions	VocWords		
MARKER	("<sps>na" --) naEx(--) :	note Voc- and SearchOrderInf restore noted information	cor-e
CONSTANT	(x "<sps>na" --) naEx(-- x) :	create a constant na; init it with x get x on stack	cor
VARIABLE	("<sps>na" --) naEx(-- a-ad) :	create a variable na get adress of value on stack	cor
LITERAL	Co:(x --) Ru:(-- x)	note the Constant in current Def get noted Constant on DataStack	cor
2CONSTANT	(d "<sps>na" --) naEx(-- d) :	create a DoublConstant get DblConstant na on DataStack	dbl
2VARIABLE	("<sps>na" --) naEx(-- a-ad) :	create a DoubleVariable get adr of DblVariable na	dbl
2LITERAL	Co:(d --) Ru:(-- d)	note the DblConst. in current Def get noted DblConst. on Stack	dbl

ForthWord	StackEffects	What Word Is Doing	ANS
Definitions	**DefinitionWords**		
CREATE	("<sps>na" --) naEx(-- a-ad) :	create a word na get DataFieldAdr of word na	cor
DOES>	Co:(C: colon-sys1 -- colon-sys2) Ru:(--)(R:nest-sys --) Init:(i*x -- i*x a-ad) (R: -- nest-sys) naEx(i*x -- j*x) :	use for Definition of Defining-Words. Used like: : DefTypeName <DefineTimeAction> NameCreatingWord <CompileTimeAction> DOES> <RunTimeAction> ; execute the appended RunTimeSemantics	cor
:	(C: "<sps>na" -- colon-sys) Init:(i*x -- i*x) (R: -- nest-sys) naEx(i*x -- j*x) :	start ColonDef.; save colon-sys for inner use save information of calling word stack depends on used words	cor
;	Co:(C: colon-sys --) Ru:(--) (R: nest-sys --)	end colon-definition return to calling word using saved nesting information	cor
:NONAME	(C: -- colon-sys) (S: -- xt) Init:(i*x--i*x) (R: --nest-sys) xt(i*x -- j*x) Ex:	like colon-definition without name save information of calling word stack depends on used words (use only with xt EXECUTE)	cor-e
CODE	("<sps>na" --) naEx(i*x -- j*x) :	start CODE-Definition (Ass.) execute defined word	too-c
;CODE	Co:(C: colon-sys --) Ru:(--)(R:nest-sys --)	end CODE-Definition (Ass.) return to nest-operation	too-c

Locals			
(LOCAL)	Ex:(c-ad u --) loEx (-- x) :	if u is zero: use last Local, otherwise define a new Local named by CharString (c-ad u) get value of lo on stack	loc
LOCALS |	Co:("<sps>na1"... ..."<sps>nan" "|"--) Ru:(xn ... x1 --)	note up to eight locals for RunTimeUse values to noted locals (xi to nai)	loc-e
TO	Int:(x "<sps>na" --) Co:("<sps>na" --) Ru:(x --)	store x in local na note na of local for RunTimeUse stor x in noted local-name	loc
VALUE	(x "<sps>na" --) naEx(-- x) :	create na as value; init it with x get x on stack	cor-e
TO	In:(x "<sps>na" --) Co:("<sps>na" --) Ru:(x --)	store x in na (def. by VALUE) note na for RuntimeUse store x in noted na	cor-e

Forth-Gesellschaft e.V.; PF 1110; D-85701
Unterschleißheim; 089-3173784;
secretary@admin.FORTH-eV.de