title	description	stack_effect	return_stack_effec	expected_input	compiler_stack_	special
~	root context	--	--	word	--	context
~.	display context order	--	--	--	--	no
<<	shift to the left	n1 n2 -- n1'	--	--	--	no
<\	start coroutines	--	--	--	--	no
<#	start pictured output	--	--	--	--	no
>>	shift to the right	n1 n2 -- n1'	--	--	--	no
|	comment	--	--	rest of line	--	no
-	n3 := n1 - n2	n1 n2 -- n3	--	--	--	no
,	here ! 1 cells allot	n --	--	--	--	no
;	end colon definition	--	(r --)	--	--	no
;;	terminate colon definition	--	(r --)	--	--	no
:	start colon definition	--	--	word	--	mode on
::	start noname definition	-- xt	--	--	--	mode on
!	set n the memory cell at p	n p --	--	--	--	no
?	ask dictionary position of word	-- p	--	word	--	no
/	n3 := n1 / n2	n1 n2 -- n3	--	--	--	no
.	output number	n --	--	--	--	no
."	output string	--	--	string"	--	no
	ask xt of a word	-- xt	--	word	--	no
"	string	-- p c	--	string"	--	immediate
[	turn off execution mode	--	--	--	--	immediate, mode off
[?]	ask dictionary position of word	-- p	--	word	--	immediate
[']	ask xt of a word	-- xt	--	word	--	immediate
]	turn on execution mode	--	--	--	--	mode on
{	start temporary compilation	--	--	--	--	no
}	end temporary compilation	--	--	--	--	no
@	get n from memory cell at p	p -- n	--	--	--	no
@@	similar to: DUP @ SWAP CELL+ @	p -- n1 n2	--	--	--	no
@+	similar to: CELL+ DUP CELL- @	--	--	--	--	no
*	n3 := n1 * n2	n1 n2 -- n3	--	--	--	no
*/	n4 := n1 * n2 / n3	n1 n2 n3 -- n4	--	--	--	no
\>	end coroutines	--	--	--	--	no
\\	separate couroutines	--	--	--	--	no
#	convert digit for pictured output	d -- d'	--	--	--	pictured

#>	end pictured output	d -- p c	--	--	--	pictured
+	addition n3 = n1 + n2	n1 n2 -- n3	--	--	--	no
+!	add n to cell at p	n p --	--	--	--	no
0;	terminate if zero	n -- n|	r -- r|	--	--	no
1-	similar to: 1 -	n -- n'	--	--	--	no
1+	similar to: 1 +	n -- n'	--	--	--	no
2-	similar to: 2 -	n -- n'	--	--	--	no
2/	similar to: 2 /	n -- n'	--	--	--	no
2+	similar to: 2 +	n -- n'	--	--	--	no
2drop	remove topmost two stack entries	n1 n2 --	--	--	--	no
2dup	duplicate topmost two stack entries	d1 -- d1 d1	--	--	--	no
2swap	swap topmost 4 stack entries	d1 d2 -- d2 d1	--	--	--	no
5+	similar to: 5 +	n -- n'	--	--	--	no
abs	n2 := ABS(n1)	n1 -- n2	--	--	--	no
again	part of BEGIN ... REPEAT	--	--	--	--	no
>again<	(implementation detail)	p -- q	--	--	--	no
align	align data in dictionary space	--	--	--	--	no
allot	allocate bytes in dictionary space	--	--	--	--	no
and	n3 := n1 and n2	n1 n2 -- n3	--	--	--	no
and:	continue execution if flag is true	flag --	r -- r|	--	--	no
ans~	context for ans compatiblity	--	--	word	--	context
arg	get argument n	n -- p c	--	--	--	no
argc	number of arguments	-- n	--	--	--	no
back	compile a back jump	xt --	--	--	--	no
base	current conversion base	-- v	--	--	--	no
begin	start BEGIN ... REPEAT loop	--	--	--	--	immediate
*begin	start binary search	n -- a	-- i1 i2	--	--	no
bnf~	context for BNF parser functionality	--	--	word	--	context
branch!	compile on position p a branch to t	t p --	--	--	--	no
bye	exit HelFORTH	--	--	--	--	no
c!	set character on postion p	ch p --	--	--	--	no
c@	get character on position p	p -- ch	--	--	--	no
case	start of Eaker-CASE	--	--	--	-- 0	no
cat	copy content of file to output	--	--	filename	--	no
cell-	subtract cell-size from value	n -- n'	--	--	--	no

cell+	add cell-size to value	n -- n'	--	--	--	no
cells	calculate size of n cells	n -- n'	--	--	--	no
char	get character from example	-- ch	--	word	--	no
[char]	get character from example	-- ch	--	word	--	immediate
class	variable that contains the default class	-- p	--	--	--	no
class>	define class of last defined word	--	--	--	--	no
class'	set the class of the last defined word	xt --	--	--	--	no
cmove	move bytes in memory	p1 p2 c --	--	--	--	no
cmove>	move bytes in memory	p1 p2 c --	--	--	--	no
compile	compile a call to a xt	xt --	--	--	--	no
constant	create a new constant	n --	--	word	--	no
context	create a new context	--	--	word	--	no
count	convert byte counted string	p -- p' c	--	--	--	no
cp	variable with pointer to last compilation	-- v	--	--	--	no
cr	similar to: 10 EMIT	--	--	--	--	no
create	create a new word	--	--	word	--	no
creates	create a word from string	p c --	--	--	--	no
depth	return depth of stack	-- n	--	--	--	no
depth0	(depreciated)	-- p	--	--	--	no
>digit	convert number to digit	n -- ch	--	--	--	no
disasm	disassemble about n lines from p	p n -- p'	--	--	--	no
disasm~	disassembler context	--	--	word	--	context
do	begin of DO ... LOOP	n1 n2 --	--	--	--	no
?do	begin of ?DO ... LOOP	n1 n2 --	--	--	--	no
doer	create a new doer	--	--	word	--	no
does>	define behaviour of last word	--	r --	--	--	no
drop	remove topmost stack entry	n --	--	--	--	no
drop?	return true if last compiled thing is DROP	-- flag	--	--	--	no
dup	duplicate topmost stack entry	n -- n n	--	--	--	no
?dup	duplicate if TOS is nonzero	n -- (0 | n n)	--	--	--	no
dup,	compile DUP	--	--	--	--	no
dup?	return true if last compiled thing is DUP	-- flag	--	--	--	no
e:	compile or interpret semantic	--	--	word	--	immediate
e{	evaluate string up to }	--	--	string}	--	no
else	part of IF ... ELSE ... THEN	--	--	--	p1 -- p2	no

emit	output character	ch --	--	--	--	no
endcase	part of Eaker-CASE	--	--	--	0|p --	no
endof	part of Eaker-CASE	--	--	--	0|p1 p2 -- p3	no
enter	enter a context	--	--	context	--	no
>enter	(depreciated)	p c --	--	--	--	no
;enter	end effects of topmost context	--	--	--	--	no
env	query environment variable	p c -- (0 | p' c')	--	--	--	no
.env	display environment entries	p c --	--	--	--	no
env+	query next environment variable	-- p c	--	--	--	no
ep	variable with environment query state	-- v	--	--	--	no
equal	compare strings	p1 p2 c -- flag	--	--	--	no
eval	evaluate string	p c -- ...	--	--	--	no
exec	execute arg and xt at p	p -- ...	--	--	--	no
execute	execute xt	xt -- ...	--	--	--	no
?execute	execute or compile xt depending on mode	xt -- ...	--	--	--	no
fileio~	file I/O context	--	--	word	--	context
fill	fill memory area with byte	p c ch --	--	--	--	no
find	find word in dictionary	p c -- 0|p	--	--	--	no
fm/mod	n3 := d / n1; n2 := d MOD n1	d n1 -- n2 n3	--	--	--	no
for	begin of FOR ... LOOP	n --	-- i	--	-- x	no
forth	set CLASS to normal word creation	--	--	--	--	no
forward	compile place holder for forward ref	-- p	--	--	--	no
here	get top of heap	-- p	--	--	--	no
hold	add character to pictured output	ch --	--	--	--	pictured
i	get loop index	-- n	ls -- ls	--	--	no
if	conditional execution	n --	--	--	-- p	immediate
<>if	conditional execution	n1 n2 --	--	--	-- p	immediate
=if	conditional execution	n1 n2 --	--	--	-- p	immeditate
>if	conditional execution	n1 n2 --	--	--	-- p	immediate
immediate	give last defined word the immediate class	--	--	--	--	no
inline	inline code at p	p --	--	--	--	no
?inline	inline or execute code	... p -- ...	--	--	--	no
interpret	enter the interpret loop	--	--	--	--	no
is	set execution vector	xt --	--	word	--	immediate
j	get second loop index	-- n	ls1 ls2 -- ls1 ls2	--	--	no

key	get key from input	-- ch	--	--	--	no
last	variable with last dictionary entry	-- p	--	--	--	no
last'	return pointer to last tick information	-- p	--	--	--	no
later>	execute following code later	--	r1 -- r2 r1	--	--	ends word
lcp	(depreciated)	-- v	--	--	--	no
leave	leave a loop	--	--	--	--	no
like	get where a vector points to	-- xt	--	word	--	immediate
lit	fetch compiled literal and remove it	-- n	--	--	--	no
lit?	was last compiled thing a literal?	-- flag	--	--	--	no
literal	compile literal	-- n	--	--	n --	immediate
?literal	compile literal in compilation mode	n -- n|	--	--	--	no
loop	end FOR ... LOOP	--	ls|i --	--	x --	no
+loop	end FOR ... +LOOP	n --	ls --	--	x --	no
m	use generic macro	--	--	--	--	no
m{	define generic macro and use it	--	--	string}	--	immediate
m*	d := n1 * n2	n1 n2 -- d	--	--	--	no
macro	set CLASS to macro word creation	--	--	--	--	no
make	set execution vector	--	--	word	--	immediate
mod	n3 := n1 MOD n2	n1 n2 -- n3	--	--	--	no
*/mod	n5 := n1 * n2 / n3; n4 := n1 * n2 MOD n3	n1 n2 n3 -- n4 n5	--	--	--	no
/MOD	n4 := n1 / n2; n3 := n1 MOD n2	n1 n2 -- n3 n4	--	--	--	no
move	move memory from p1 to p2	p1 p2 c --	--	--	--	no
ndrop	remove n topmost stack entries	n --	--	--	--	no
negate	negate	n -- -n	--	--	--	no
nip	remove second stack entry	n1 n2 -- n2	--	--	--	no
nop	do nothing	--	--	--	--	no
not	n2 := NOT n1	n1 -- n2	--	--	--	no
>num	convert a number	p c -- (0 | n -1)	--	--	--	no
>number	convert a number	ud p c -- ud' p c	--	--	--	no
of	part of Eaker-CASE	n --	--	--	-- p	no
off	set cell at p to zero	p --	--	--	--	no
on	set cell at p to -1	p --	--	--	--	no
;opt	end literal optimization	--	(r --)	word	--	immediate, mode off
:opt	start literal optimization	--	--	--	--	mode on
opt?	is code optimizeable for literal?	z -- (z 0 | n z)	--	--	--	no

optimize	depreciated	-- v	--	--	--	no
or	n3 := n1 OR n2	n1 n2 -- n3	--	--	--	no
or:	continue execution if flag is false	flag --	r -- r|	--	--	no
over	copy snd over tos	n1 n2 -- n1 n2 n1	--	--	--	no
pad	get temporary buffer	-- p	--	--	--	no
>pad	move string to temporary buffer	p c -- p' c	--	--	--	no
/pad	bytes in a pad	-- n	--	--	--	no
page	clear screen	--	--	--	--	no
parse	parse string from input	ch -- p c	--	--	--	no
pic	variable with pointer to pictured output	-- v	--	--	--	no
pick	get specified stack entry	... n1 -- n2	--	--	--	no
place	copy to counted string	p1 c p2 --	--	--	--	no
pop	get value from named stack	-- n	--	stack	--	immediate
push	put value to named stack	n --	--	stack	--	immediate
quote	(depreciated)	-- p c	--	word ...	--	no
[quote]	(depreciated)	-- p c	--	word ...	--	immediate
r	get top of return stack	-- n	n -- n	--	--	inline
>r	put value to return stack	n --	-- n	--	--	inline
,r	store relative value	n --	--	--	--	no
!r	set relative value	n p --	--	--	--	no
@r	fetch relative value	p -- n	--	--	--	no
r>	get value from return stack	-- n	n --	--	--	inline
rdrop	remove value from return stack	--	n --	--	--	inline
repeat	part of BEGIN ... REPEAT	--	--	--	x --	immediate
*repeat	end binary search	f(a) b -- a	i1 i2 --	--	x --	immediate
;;ret	end current word without jump	--	r --	--	--	inline
rp	pointer to first return stack entry	-- p	--	--	--	inline
rr	get second return stack entry	-- n1	n1 n2 -- n1 n2	--	--	no
>rr	move to second return stack entry	n1 --	n2 -- n1 n2	--	--	no
rr>	get from second return stack entry	-- n1	n1 n2 -- n2	--	--	no
#s	finish pictured output of a number	d -- d	--	--	--	pictured
s,	similar to: TUCK HERE PLACE 1+ ALLOT	p c --	--	--	--	no
s++	step over first character	p c -- p' c'	--	--	--	no
screen~	context for screen output	--	--	word	--	context
s>d	convert single to double	n -- d	--	--	--	no

see	disassemble a word	--	--	word	--	no
short	variable: on means compile short jumps	-- v	--	--	--	no
sm/rem	n3 := d / n1; n2 := d MOD n1	d n1 -- n2 n3	--	--	--	no
snd	get second entry from named stack	-- n	--	stack	--	immediate
sp	pointer to first stack entry	-- p	--	--	--	no
space	output space character	--	--	--	--	no
spaces	output n space characters	n --	--	--	--	no
sqrt	sqare root	n1 -- n2	--	--	--	no
stack	create new named stack	n --	--	word	--	no
state	pointer to systems state description	--	--	--	--	no
stub	call a stub	--	--	--	--	no
stub:	define a stub	--	--	--	--	no
subr	compiles or executes, turns off jump at end	xt --	--	--	--	no
swap	exchange first two stack entries	n1 n2 -- n2 n1	--	--	--	no
sz	count asciiz-string	p -- p c	--	--	--	no
then	end of IF ... THEN	--	--	--	p --	immediate
;then	end of IF ... THEN	--	(r --)	--	p --	immediate
then>	compile or execute following code	--	--	--	--	no
times	repeat the word n times	n --	--	word	--	immediate
tos	get top of named stack	-- n	--	stack	--	immediate
tuck	copy tos under second	n1 n2 -- n2 n1 n2	--	--	--	no
type	output a string	p c --	--	--	--	no
u.	output unsigned number	u --	--	--	--	no
uchars	count characters in UTF-8 string	p c -- n	--	--	--	no
ud/mod	ud2 := ud1 / u1; u2 = ud1 MOD u1	ud1 u1 -- ud2 u2	--	--	--	no
uemit	output UTF-8 character	ch --	--	--	--	no
uhold	like HOLD but für UTF-8 character	ch --	--	--	--	no
u<if	conditional execution	u1 u2 --	--	--	--	immediate
um*	ud := u1 * u2	u1 u2 -- ud	--	--	--	no
um/mod	u3 := ud / u1; u2 := ud MOD u1	ud u1 -- u2 u3	--	--	--	no
undo	set vectors default behaviour	--	--	word	--	immediate
>upper	convert character to uppercase	ch -- ch'	--	--	--	no
>upstr	convert string to uppercase	p c -- p c	--	--	--	no
used	number of used bytes in compilation area	-- n	--	--	--	no
utf8@	get utf8 character from string	p c -- p' c' ch	--	--	--	no

u<while	part of BEGIN ... REPEAT	u1 u2 --	--	--	--	no
v:	compile a call to vector content	--	--	word	--	immediate
variable	create a new variable	--	--	--	--	no
vector	compile vector	--	--	--	--	no
verbose	variable: on means more verbose output	-- v	--	--	--	no
while	part of BEGIN ... REPEAT	--	--	--	--	immediate
<>while	part of BEGIN ... REPEAT	n1 n2 --	--	--	--	immediate
=while	part of BEGIN ... REPEAT	n1 n2 --	--	--	--	immediate
>while	part of BEGIN ... REPEAT	n1 n2 --	--	--	--	immediate
with	search also this context	--	--	context	--	no
;with	end searching additional context	--	--	--	--	no
word?	similar to: TYPE '? EMIT CR	p c --	--	--	--	no
words	display word list	--	--	--	--	no
wsparse	similar to: 32 PARSE	--	--	--	--	no
x:	compile execution semantic	--	--	word	--	no
xor	n3 := n1 XOR n2	n1 n2 -- n3	--	--	--	no
XXX	mark last defined word as bad	--	--	--	--	immediate