ESP32forth "Cheat sheet"

Stack Notation	*Cells*	*Description*
c	1	Character (high byte ignored)
flag	1	Boolean (0 = False, 1 = True)
n	1	Signed number
u	1	Unsigned number
x	1	Non-specific single cell
adr	1	Memory address
d	2	Signed double number
ud	2	Unsigned double number
xd	2	Non-specific double cell
f	2	Floating point number

Immediate words in Green
Immediate and compile-only words in Blue

Stack Manipulation

DROP (x →) Discard TOS (top of stack)
DUP (x → x x) Duplicate TOS
?DUP (x → x (x)) DUP, if non-zero
OVER (x_1 x_2 → x_1 x_2 x_1) Copy second on stack to top
PICK (x_n...x_1 n → x_n...x_1 x_n) Copy n^{th} cell to top
ROLL (x_n...x_1 n → x_{n-1}...x_1 x_n) Rotate n^{th} cell to top
ROT (x_1 x_2 x_3 → x_2 x_3 x_1) Rotate 3rd cell to top
>R (x →) (R: → x) Move TOS to Return Stack
R> (→ x) (R: x →) Retrieve from Return Stack
SWAP (x_1 x_2 → x_2 x_1) Exchange the two top cells

Control Structures

IF (*flag* →) Conditional structure IF..(ELSE)..THEN

ELSE (→) False condition of an IF structure

THEN (→) End of an IF conditional structure

DO (n_1 n_2 →) Counted loop structure DO...LOOP (n_2 = count start, n_1 = count end)
LOOP (→) Increment loop count, terminate if end
+LOOP (*n* →) Add *n* to loop count, terminate if end
I (→ *n*) Get current loop count
I' (→ *n*) Get current loop count limit
J (→ *n*) Get outer loop count
LEAVE (→) Force a DO...LOOP count to end
BEGIN (→) Begin a WHILE or UNTIL loop
UNTIL (*flag* →) Loop until *flag* = true (BEGIN..UNTIL)
WHILE (*flag* →) Exit loop when *flag* = false (BEGIN..WHILE..REPEAT)
REPEAT (→) Jump back to BEGIN in a WHILE loop
EXIT (→) Exit current word execution
EXECUTE (*adr* →) Execute word with compilation *adr*
ABORT (... →) Quit program, clearing data stack
QUIT (→) Quit program, not clearing data stack

Comparison

< (n_1 n_2 → *flag*) True if $n_1 < n_2$
= (n_1 n_2 → *flag*) True if $n_1 = n_2$
> (n_1 n_2 → *flag*) True if $n_1 > n_2$
0< (n → *flag*) True if $n < 0$
0= (n → *flag*) True if $n = 0$
0> (n → *flag*) True if $n > 0$
U< (u_1 u_2 → *flag*) True if $u_1 < u_2$
D< (d_1 d_2 → *flag*) True if $d_1 < d_2$
MAX (n_1 n_2 → n_3) Leave greater of two numbers
MIN (n_1 n_2 → n_3) Leave lesser of two numbers

Logical

AND (x_1 x_2 → x_3) Bitwise boolean AND

OR (x_1 x_2 → x_3) Bitwise boolean OR

XOR (x_1 x_2 → x_3) Bitwise boolean XOR

Integer Arithmetic

+ (n_1 n_2 → n_3) $n_3 = n_1 + n_2$
- (n_1 n_2 → n_3) $n_3 = n_1 - n_2$
***** (n_1 n_2 → n_3) $n_3 = n_1 * n_2$
/ (n_1 n_2 → n_3) $n_4 = n_1 / n_2$
MOD (n_1 n_2 → n_3) Remainder of n_1 / n_2 (sign of n_1) **/MOD** (n_1 n_2 → n_3 n_4) n_3 = remainder of n_1 / n_2 $n_4 = n_1 / n_2$
***/** (n_1 n_2 n_3 → n_4) $n_4 = n_1 * n_2 / n_3$
***/MOD** (n_1 n_2 n_3 → n_4 n_5) n_4 = remainder of $n_1 * n_2 / n_3$ $n_5 = n_1 * n_2 / n_3$
1+ (n_1 → n_2) $n_2 = n_1 + 1$
1- (n_1 → n_2) $n_2 = n_1 - 1$
2+ (n_1 → n_2) $n_2 = n_1 + 2$
2- (n_1 → n_2) $n_2 = n_1 - 2$
ABS (n → u) $u = |n|$ (absolute value)
NEGATE (n_1 → n_2) $n_2 = -n_1$ (two's complement)
U* (u_1 u_2 → ud) $ud = u_1 * u_2$
U/MOD (ud u_1 → u_2 u_3) u_2 = remainder of ud / u_1 $u_3 = ud / u_1$
D+ (d_1 d_2 → d_3) $d_3 = d_1 + d_2$
DNEGATE (d_1 → d_2) $d_2 = -d_1$ (two's complement)

Memory

@ (*adr* → *x*) Read *x* (2 bytes) from *adr*

! (*x adr* →) Store *x* 2 bytes) to *adr*

C@ (*adr* → *c*) Read *c* (1 byte) from *adr*
C! (*c adr* →) Store *c* (1 byte) to *adr*

Character Input/Output

CR (→) Print carriage return and line feed
ASCII *text* (→ *c*) ASCII code of first character in *text*
EMIT (*c* →) Print ASCII *c* character

SPACE (→) Print one space
SPACES (*n* →) Print *n* spaces, if $n > 0$
." (→) Print a string terminated by "
TYPE (*adr n* →) Print *n* characters from *adr*
QUERY (→) Accept entry at the input buffer