FORTH-79 HANDY REFERENCE

Stack inputs and outputs are shown; top of stack on right. See operand key at bottom.

STACK MANIPULATION

DUP	(n → n n)	Duplicate top of stack.
DROP	(n →)	Discard top of stack.
SWAP	(n1 n2 → n2 n1)	Exchange top two stack items.
OVER	(n1 n2 → n1 n2 n1)	Make copy of second item on top.
ROT	(n1 n2 n3 → n2 n3 n1)	Rotate third item to top. "rote"
PICK	(n1 → n2)	Copy n1-th item to top. (Thus 1 PICK = DUP , 2 PICK = OVER)
ROLL	(n →)	Rotate n-th item to top. (Thus 2 ROLL = SWAP , 3 ROLL = ROT)
?DUP	(n → n (n))	Duplicate only if non-zero. "query-dup"
>R	(n →)	Move top item to "return stack" for temporary storage (use caution). "to-r"
R>	(→ n)	Retrieve item from return stack. "r-from"
R@	(→ n)	Copy top of return stack onto stack. "r-fetch"
DEPTH	(→ n)	Count number of items on stack.

COMPARISON

<	(n1 n2 → flag)	True if n1 less than n2. "less-than"
=	(n1 n2 → flag)	True if top two numbers are equal. "equals"
>	(n1 n2 → flag)	True if n1 greater than n2. "greater-than"
0<	(n → flag)	True if top number negative. "zero-less"
0=	(n → flag)	True if top number zero. (Equivalent to NOT) "zero-equals"
0>	(n → flag)	True if top number greater than zero. "zero-greater"
D<	(d1 d2 → flag)	True if d1 less than d2. "d-less-than"
U<	(un1 un2 → flag)	Compare top two items as unsigned integers. "u-less-than"
NOT	(flag → ¬flag)	Reverse truth value. (Equivalent to 0=)

ARITHMETIC AND LOGICAL

+	(n1 n2 → sum)	Add. "plus"
D+	(d1 d2 → sum)	Add double-precision numbers. "d-plus"
-	(n1 n2 → diff)	Subtract (n1−n2). "minus"
1+	(n → n+1)	Add 1 to top number. "one-plus"
1-	(n → n−1)	Subtract 1 from top number. "one-minus"
2+	(n → n+2)	Add 2 to top number. "two-plus"
2-	(n → n−2)	Subtract 2 from top number. "two-minus"
*	(n1 n2 → prod)	Multiply. "times"
/	(n1 n2 → quot)	Divide (n1/n2). (Quotient rounded toward zero) "divide"
MOD	(n1 n2 → rem)	Modulo (*i.e.*, remainder from division n1/n2). Remainder has same sign as n1. "mod"
/MOD	(n1 n2 → rem quot)	Divide, giving remainder and quotient. "divide-mod"
*/MOD	(n1 n2 n3 → rem quot)	Multiply, then divide (n1*n2/n3), with double-precision intermediate. "times-divide-mod"
*/	(n1 n2 n3 → quot)	Like */MOD , but give quotient only, rounded toward zero. "times-divide"
U*	(un1 un2 → ud)	Multiply unsigned numbers, leaving unsigned double-precision result. "u-times"
U/MOD	(ud un → urem uquot)	Divide double number by single, giving remainder and quotient, all unsigned. "u-divide-mod"
MAX	(n1 n2 → max)	Leave greater of two numbers. "max"
MIN	(n1 n2 → min)	Leave lesser of two numbers. "min"
ABS	(n → |n|)	Absolute value. "absolute"
NEGATE	(n → −n)	Leave two's complement.
DNEGATE	(d → −d)	Leave two's complement of double-precision number. "d-negate"
AND	(n1 n2 → and)	Bitwise logical AND.
OR	(n1 n2 → or)	Bitwise logical OR.
XOR	(n1 n2 → xor)	Bitwise logical exclusive-OR. "x-or"

MEMORY

@	(addr → n)	Replace address by number at address. "fetch"
!	(n addr →)	Store n at addr. "store"
C@	(addr → byte)	Fetch least significant byte only. "c-fetch"
C!	(n addr →)	Store least signficant byte only. "c-store"
?	(addr →)	Display number at address. "question-mark"
+!	(n addr →)	Add n to number at addr. "plus-store"
MOVE	(addr1 addr2 n →)	Move n numbers starting at addr1 to memory starting at addr2, if n>0.
CMOVE	(addr1 addr2 n →)	Move n bytes starting at addr1 to memory starting at addr2, if n>0. "c-move"
FILL	(addr n byte →)	Fill n bytes in memory with byte beginning at addr, if n>0.

CONTROL STRUCTURES

DO ... LOOP	do: (end+1 start →)	Set up loop, given index range.
I	(→ index)	Place current loop index on data stack.
J	(→ index)	Return index of next outer loop in same definition.
LEAVE	(→)	Terminate loop at next LOOP or +LOOP , by setting limit equal to index.
DO ... +LOOP	do: (limit start →) +loop: (n →)	Like DO ... LOOP , but adds stack value (instead of always 1) to index. Loop terminates when index is greater than or equal to limit (n>0), or when index is less than limit (n<0). "plus-loop"
IF ... (true) ... THEN	if: (flag →)	If top of stack true, execute.
IF ... (true) ... ELSE ... (false) ... THEN	if: (flag →)	Same, but if false, execute ELSE clause.
BEGIN ... UNTIL	until: (flag →)	Loop back to BEGIN until true at UNTIL .
BEGIN ... WHILE ... REPEAT	while: (flag →)	Loop while true at WHILE ; REPEAT loops unconditionally to BEGIN . When false, continue after REPEAT .
EXIT	(→)	Terminate execution of colon definition. (May not be used within DO ... LOOP)
EXECUTE	(addr →)	Execute dictionary entry at compilation address on stack (*e.g.*, address returned by FIND).

Operand key:

n, n1, ...	16-bit signed numbers	d, d1, ...	32-bit signed numbers	addr, addr1, ...	addresses	char	7-bit ascii character value
u			unsigned	byte	8-bit byte	flag	boolean flag

TERMINAL INPUT-OUTPUT

CR	(–)	Do a carriage return and line feed "c-r"
EMIT	(char –)	Type ascii value from stack.
SPACE	(–)	Type one space.
SPACES	(n –)	Type n spaces, if n>0.
TYPE	(addr n –)	Type string of n characters beginning at addr, if n>0.
COUNT	(addr – addr+1 n)	Change address of string (prefixed by length byte at addr) to TYPE form.
–TRAILING	(addr n1 – addr n2)	Reduce character count of string at addr to omit trailing blanks. "dash-trailing"
KEY	(– char)	Read key and leave ascii value on stack.
EXPECT	(addr n –)	Read n characters (or until carriage return) from terminal to address, with null(s) at end.
QUERY	(–)	Read line of up to 80 characters from terminal to input buffer.
WORD	(char – addr)	Read next word from input stream using char as delimiter, or until null. Leave addr of length byte.

NUMERIC CONVERSION

BASE	(– addr)	System variable containing radix for numeric conversion.
DECIMAL	(–)	Set decimal number base.
.	(n –)	Print number with one trailing blank and sign if negative. "dot"
U.	(un –)	Print top of stack as unsigned number with one trailing blank. "u-dot"
CONVERT	(d1 addr1 – d2 addr2)	Convert string at addr1+1 to double number. Add to d1 leaving sum d2 and addr2 of first non-digit.
<#	(–)	Start numeric output string conversion. "less-sharp"
#	(ud1 – ud2)	Convert next digit of unsigned double number and add character to output string. "sharp"
#S	(ud – 0 0)	Convert all significant digits of unsigned double number to output string. "sharp-s"
HOLD	(char –)	Add ascii char to output string.
SIGN	(n –)	Add minus sign to output string if n<0.
#>	(d – addr n)	Drop d and terminate numeric output string, leaving addr and count for TYPE . "sharp-greater"

MASS STORAGE INPUT/OUTPUT

LIST	(n –)	List screen n and set SCR to contain n.
LOAD	(n –)	Interpret screen n, then resume interpretation of the current input stream.
SCR	(– addr)	System variable containing screen number most recently listed.
BLOCK	(n – addr)	Leave memory address of block, reading from mass storage if necessary.
UPDATE	(–)	Mark last block referenced as modified.
BUFFER	(n – addr)	Leave addr of a free buffer, assigned to block n; write previous contents to mass storage if UPDATEd.
SAVE-BUFFERS	(–)	Write all UPDATEd blocks to mass storage.
EMPTY-BUFFERS	(–)	Mark all block buffers as empty, without writing UPDATEd blocks to mass storage.

DEFINING WORDS

: xxx	(–)	Begin colon definition of xxx . "colon"
;	(–)	End colon definition. "semi-colon"
VARIABLE xxx	(–)	Create a two-byte variable named xxx ; returns address when executed.
	xxx: (– addr)	
CONSTANT xxx	(n –)	Create a constant named xxx with value n; returns value when executed.
	xxx: (– n)	
VOCABULARY xxx	(–)	Create a vocabulary named xxx ; becomes CONTEXT vocabulary when executed.
CREATE ... DOES>	does: (– addr)	Used to create a new defining word, with execution-time routine in high-level FORTH. "does"

VOCABULARIES

CONTEXT	(– addr)	System variable pointing to vocabulary where word names are searched for.
CURRENT	(– addr)	System variable pointing to vocabulary where new definitions are put.
FORTH	(–)	Main vocabulary, contained in all other vocabularies. Execution of FORTH sets context vocabulary.
DEFINITIONS	(–)	Sets CURRENT vocabulary to CONTEXT.
' xxx	(– addr)	Find address of xxx in dictionary; if used in definition, compile address. "tick"
FIND	(– addr)	Leave compilation address of next word in input stream. If not found in CONTEXT or FORTH, leave 0.
FORGET xxx	(–)	Forget all definitions back to and including xxx , which must be in CURRENT or FORTH.

COMPILER

,	(n –)	Compile a number into the dictionary. "comma"
ALLOT	(n –)	Add two bytes to the parameter field of the most recently-defined word.
."	(–)	Print message (terminated by "). If used in definition, print when executed. "dot-quote"
IMMEDIATE	(–)	Mark last-defined word to be executed when encountered in a definition, rather than compiled.
LITERAL	(n –)	If compiling, save n in dictionary, to be returned to stack when definition is executed.
STATE	(– addr)	System variable whose value is non-zero when compilation is occurring.
[	(–)	Stop compiling input text and begin executing. "left-bracket"
]	(–)	Stop executing input text and begin compiling. "right-bracket"
COMPILE	(–)	Compile the address of the next non-IMMEDIATE word into the dictionary.
[COMPILE]	(–)	Compile the following word, even if IMMEDIATE. "bracket-compile"

MISCELLANEOUS

(	(–)	Begin comment, terminated by) on same line or screen; space after (. "paren", "close-paren"
HERE	(– addr)	Leave address of next available dictionary location.
PAD	(– addr)	Leave address of a scratch area of at least 64 bytes.
>IN	(– addr)	System variable containing character offset into input buffer; used, e.g., by WORD. "to-in"
BLK	(– addr)	System variable containing block number currently being interpreted, or 0 if from terminal. "b-l-k"
ABORT	(–)	Clear data and return stacks, set execution mode, return control to terminal.
QUIT	(–)	Like ABORT , except does not clear data stack or print any message.
79-STANDARD	(–)	Verify that system conforms to FORTH-79 Standard.