APPENDIX: GLOSSARY

back-tab -- 9
Returns control to word which called the present one. Generated by semi-colon. Return address must be on top or R-stack, hence cannot be used inside of #[ ]# or if >R values are on R-stack. It can be used within conditionals to abort. Synonym for RET .

<control-break> 52
The <control-break> is not a normal word. However, it may be issued at any time to cause a Warm re-start. The keyboard interrupts must be enabled.

! val addr -- 15
Stores 2nd at the address in the top.

!* -- 72
Compile address of this word into this definition. This allows recursion. It is the user responsibility to test for the exit condition; otherwise, the R-stack will overflow.

" -- address 64
Begins a quoted string. This is a state-smart word which may be use either inside or outside of colon definitions. The string is terminated by another " mark and a following space, tab, or carraige return. The count of characters within the string is found at the word located at "address-2". In addition, the string is terminated by an ASCII null, not included in the string count. If the string is used outside of a colon definition, it is transitory, and will be over-written by other words which use PAD, just above the stack. A " mark may be placed in the text string by using two of them, "" . A " mark may also be followed within the string by any other printable character other than a space.

lo hi -- char lo' hi' 66
Divide double number on top by value in BASE, leave digit as an ASCII character, with double quotient on top.

#>. ? 0 0 -- 66
Discard final quotient (value 0.0), then output ASCII digits from stack until NULL character is reached.

#R dbl -- ? 66
Convert double top to ASCII digits using # and output enough spaces to right justify when digits are printed.

#S dbl -- ? 67
convert double top to ASCII digits (one digit per word on stack) using # . Leaves quotient value of zero on top. This is useful when it is necessary to force leading zero digits.

#VOCS -- addr 61
Address of maximum number of vocabularies in search order.

#[n -- 66
Marks start of a decrementing indexed loop. Initial index value is moved to the R-stack. The end is marked by]# . Must be used within a colon definition.

$< addr1 addr2 -- flag 33
Two strings are compared for the purpose of ordering them. Note that no count is specified! At the first mis-match, the comparison stops. The flag is true (-1) if the byte at the first string is less than that at the second string. Note that addr1 must not be the same as addr2.

' -- addr 37
Reads string; pushes the address of that word onto the stack. Searches first the SEARCHING , then the ROOT vocabulary. If the string is not found, the bell rings and the cursor is backed up to the beginning of the input string. This continues until a string is found. If necessary to get out of this, use something like: DUP DROP

'LAST -- addr 37
Pushes the address of the last word in the GROWING vocabulary.

'PRE addr1 -- addr1 addr2 36
Pushes address of proceeding dictionary word. Initially top must be a word address. At end of a dictionary thread, addr2 has a value of 0 .

(" -- addr
Primitive in-line string operator. The count of the string is at the word preceeding the address. The string is terminated by a null (not included in the count).

(' -- n

This is a primitive version of ' . When executed, get the next word from the input stream and search the dictionary for a match. If found, return the execution address on the top of the stack. If not found, return a value of 0.

(." --
In-line primitive for ." strings. Prints the succeeding in-line characters until a Null is encountered.

(;C --
A primitive used to change from normal high-level interpretation to code level.

(=?[n1 n2 -- or n1 n2 -- n1
Primitive CASE branch. When executed within a colon definition, the top two stack elements ar tested for equality. If the top two elements have a different value, the top element is dropped and the next in-line value is taken as the branch address. If the two elements are equal, both elements are dropped, the in-line value is skipped, and execution continues.

(?] n --
Primitive conditional branch. When executed within a colon definition, the top of the stack is popped and tested. If the result is zero, the next in-line value is taken as the branch address. If the result is non-zero, the in-line value is skipped, and execution continues.

(] --
Primitive unconditional branch. When executed within a colon definition, the next in-line address is taken as a branch address, and execution is transfered to that point.

(E] --
Primitive branch on error. When executed within a colon definition, an internal error flag EFLAG is tested. If the flag is non-zero, the next in-line value is taken as a branch address. Otherwise the in-line value is skipped and execution continues.

(B --
Back up the cursor by one character.

(CALL --
A primitive operator which compiles a CALL instruction followed by the calculated offset address based on the contents of the word following the (CALL word.

(DEFER --
Primitive routine for defered words.

(DOS ds dx cx bx ax -- ax' ds' dx' cx' 0
or ds dx dx bx ax -- ax ax' -1
Primitive operator for DOS calls. If the top value returned is 0, there were no errors. It the top element is -1, then the requested operation results in an error specified by ax' .

(JMP --
A primitive operator which compiles a machine language JMP instruction followed by the calculated offset address based on the contents of the word following the (JMP instruction.

(NUM -- n 0 or -- dbl cnt or -- -1
Convert, normally using value in BASE, the ASCII string just input with -WORD . If the string begins with a $ character, use 10 as a temporary base. If the string begins with a # character, use 16 (Hexadecimal) as a temporary base. A minus sign may be used to input a negative number. If the string contains a decimal point, the string is converted to a double number. If the string cannot be converted, a flag of -1 is returned. If a single precision number is indicated, a flag of 0 is returned. If a double number is returned, a positive number is returned containing the number of digits to the right of the decimal point, plus 1 .

(TEXT n --
Obtain characters from the input stream and store them in the buffer area.

(]# --
Test the value at the top of the return stack. If it is non-zero, decrement it by 1 and then jump to the location specified by the next in-line value. If the top of the Return stack is zero, pop it from the Return stack, jump over the in-line value, and continue execution.

* n1 n2 -- n3
Multiply top by 2nd, leaving single precision signed product top.

*/ n1 n2 n3 -- n4
Unsigned multiply n1 by n2, keeping 32 bit accuracy, divide this by n3.

+ n1 n2 -- n3
Add top to 2nd, leaving their sum.

+! n addr --
Add 2nd to word whose address is on top.

+R n --
Add top to the value on top of the R-stack.

, n --
Move top word to top of dictionary, HERE. Then bump DP by 2.

- n1 n2 -- n3
Subtract top from 2nd, leaving the difference.

-$< addr1 addr2 -- flag
Two strings are compared for the purpose of ordering them. Note that no count is specified! At the first mis-match, the comparison stops. The flag is true (-1) if the byte at the first string is less than that at the second string. Note that addr1 must not be the same as addr2. Further note that the strings run backwards in memory.

-1 -- -1
Push a value of -1 on the stack.

-2 -- -2
Push a value of -2 on the stack.

-ROT n1 n2 n3 -- n3 n1 n2
Rotate the top three items on the stack so that the former top of stack becomes the 3rd item on the stack.

-WORD n -- addr
Top=delimiter. Put a Null in the dictionary. Fetch characters from the input stream, skipping initial occurences of the delimiter character. The non-delimiter characters are stored in the dictionary in reverse order until a delimiter or a carraige return character is encountered. Add a Null after the string in the dictionary, and return the address of that null on the stack. The dictionary pointer is not updated.

.. ? --
Prints out the entire stack as though dots had been entered in sufficient quantity to just empty the stack.

. n --
Print the signed value of top followed by one space. Numberic conversion determined by value in BASE.

." --
Print the string which follows. A " mark followed by a space, tab, or carraige return terminates the string. The string can be any length and contain any 8-bit ASCII character. If an imbedded " mark is desired, it can be followed by any printable non-blank character, or alternatively each pair of "" characters may be used to represent a single " mark. Imbedded Line-Feeds and carraige returns are allowed. Note, however, that a Line-Feed does not automatically follow a Return--even though on input Return does echo the Line-Feed.

."? --
The same as ." except in swallows the first character of the text string and uses it for the terminating delimiter. This delimiter can be any character except NULL. This allows " to be in the text strint.

.$ n1 cnt --
Print the value n1 in a field cnt characters wide. The count includes a possible leading minus sign, a "$" sign, and an imbedded decimal point, and two digits to the right of the decimal point.

.A n --
Displays the low order 8 bits from top as an ASCII character. Control characters are shown as ^ followed by the character, other characters are shown as a space then the character. This is useful to make the control characters displayable.

.B --
Display top, right 8-bits, with 2 hex digits. No space follows and value in BASE has no effect.

.D$ d cnt --
Display the signed double precision number in a field cnt characters wide. If the field is wider than necessary, leading spaces will be displayed. If the field is too narrow, characters may be truncated. If the double number is negative, the count must include a place for a leading minus sign. The count must also include room for a "$" sign, an imbedded ".", and two numbers to the right of the decimal point.

.NAME addr --
Print out the name of the word whose address is on top.

.R n1 n2 --
Top specifies the width of the field, 2nd is unsigned and printed with a space following, BASE determines the conversion.

.TEXT segment addr --
Print text, stopping on a NULL. Top=start address, 2nd=segment.

.TYPE addr --
Type a null-delimited string.

.VOCAB addr --
Print the vocabulary whose pointer address is on top. Typical use is: SEARCHING .VOCAB

.W --
Display top word with 4 hex digits. No space follows and the value in BASE has no effect.

/ n1 n2 -- quot
Unsigned divide 2nd by top, leaving single precision quotient top. Arguements and result are unsigned.

/MOD num den -- rem quot

Treat the top two operands as unsigned numbers. Divide the second by the top. The quotient replaces the top value, and the remainder replaces the second value.

0 -- 0
Push a value of 0 on the top of the stack.

0< n -- flag
If the sign bit of the number at the top of the stack is set (signifying a negative number), replace the number with -1 . Otherwise, replace the number with 0 .

0= n -- flag
If the value on the top of the stack is 0, replace it with -1 . Otherwise, replace the top of the stack with 0 .

1 -- 1
Push a value of 1 on the stack.

1+ n1 -- n2
Add a value of 1 to the top item on the stack.

1- n1 -- n2
Subtract 1 from the top of the stack. Faster and shorter then 1 - .

2 -- 2
Push a value of 2 onto the stack.

2* n1 -- n2
Shift the value in top left one bit position.

2- n1 -- n2
Decrease the value of top by 2. Faster and shorter than 2 - .

2/ n1 -- n2
Shift the value in top right one bit position. A 0 bit is shifted in to the high bit position.

2! dbl addr --
Store the double number 2nd and 3rd on the stack at the address on the top of the stack.

2+ n1 -- n2
Add a value of 2 to the top item on the stack. Faster and shorter than 2 +

2- n1 -- n2
Subtract 2 from the top item on the stack. Faster and shorter then 2 -

2@ addr -- dbl
Replace the address on the top of the stack with the double number found at the address.

2DROP n1 n2 --
Drop the top two numbers from the stack (or the top double number).

2DUP dbl -- dbl dbl
Duplicates the double number on top of the stack.

2I -- n
Pushes a value onto the Parameter stack which is twice the value of the contents of the top of the R-Stack.

2OVER d1 d2 -- d1 d2 d1
Pushes a copy of the second double word on the the stack. In terms of 16-bit words, the stack behavior is:
n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2

2SWAP d1 d2 -- d2 d1
Exchange the top two double numbers on the stack.

3 -- 3
Push a value of 3 on the stack.

4 -- 4
Push a value of 4 on the stack.

: --
Create a new dictionary word. : definitions must be terminated with a semi-colon, or aborted with a \.

:ARRAY n1 n2 --

Father word which creates a two dimension word array. Dimension parameter which is 2nd is used at reference time as multiplier times top index which is then added to index which is second on stack.

:BUILD --
Used in a "Father word" to define a new "Child word". Usage is:
: father :BUILD creation logic ;: inherrited logic ;
Each time the "father" is executed :BUILD take following input string for the "child" name.

:CON n --
Fetch a word name from the input stream. Add this name to the dictionary as a constant with the value specified by the top of the stack. When the name is later executed, the value will be pushed on the stack.

:INDEX1 --> 0
Creates an array of word values which are ' addresses of other words. The size in words must be in Top. On execution of the child word the zero origin element number must be on top, the returned value is the address from the array. If this is followed by EXECUTE this makes a CASE statement.

:VAR n --
Fetch a word name from the input stream. Add this name to the dictionary as a variable, and store the value at the top of the stack as the initial value of the variable. When the name is later executed, the address of the variable will be pushed on the stack. The word @ is required to replace the address with the contents.

:VECT n --
This is a word which creates a vector of the number of words specified by top. The child word uses a parameter and returns the address of that element of the vector.

:VOC --
A defining word which creates a new vocabulary. Word 0 of the new vocabulary body contains a vocabulary number (in the range 0 to 31), which is incremented for each new vocabulary. Word 1 contains a link to a prior vocabulary to be searched, if a sought after word is not found in the current vocabulary. This is nominally set to ROOT . Word 2 is a vocabulary linkage used by a "smart" FORGET . Word 3 contains the address of a word to be executed when the search fails. This is normally set to (NUM to specify a search for a literal number. Word 4 contains the address of a word to be executed when the attempt to make a literal fails. This is normally set to (B , which will back up the cursor, and beep.

; --
Terminate Colon definition. Check and warn if parenthesis and brackets don't balance.

;[--
Suspend : definition temporarily. When]: is subsequently executed the stack must be as it was when ;[was used.

;: --
Marks where the inherited properties in a "Father-word" begin. That which follows ;: will get exectued when the "Child-word" is invoked. Used with :BUILD to create "Father-words".

< n1 n2 -- flag
If 2nd is less than top replace them with -1 . Otherwise, replace them with zero.

<# n1 n2 -- 0 n1 n2
Insert zero under top word on stack. Used to begin numeric output of a double precision number.

= n1 n2 -- flag
If 2nd is equal to to replace them with -1. Otherwise, replace them with zero.

=?[n1 n2 -- or n1 n2 -- n1
Used inside a colon definition as a case construct. If the top two elements are equal, they are both dropped from the stack, and the code following is executed. If the two elements are not equal, the statements following are skipped up to the occurrence of the next][or the trailing]]? which terminates a series of case statements. A rarely used alternative is to terminate with a]? .

> n1 n2 -- flag
If 2nd is greater than top replace them with -1. Otherwise, replace them with zero.

>R n --
Remove top and push it onto the R-Stack. Within a colon definition must be balanced with R> so that ;S generated by ; can properly execute. Can be used outside a : definition but the user must guard against removing more values from the R-stack than are there.

?[n --

Test. Do following if value on top is true, otherwise skip to matching]? or][or]]

?] n --
Jump back to [[if value on top is zero.

?COMP --
If STATE is true, i.e. compiling, do nothing. If STATE is zero issue message "CANT EXECUTE" and call QUIT.

?CSP --
Check stack pointer to verify it is at the same place following a : definition as at the beginning. This will detect unmatched looping constructs.

?DEF -- n
Searches the GROWING vocabulary for the word just obtained from the input stream. If the word is found, return the address of the word. If the word is not found, return a 0.

?DUP n -- n n or 0 -- 0
Dup top unless it is 0.

?STACK --
Test for stack underflow, and issue "EMPTY STACK" and call QUIT. Also tests for stack full and reports if less than 256 bytes remain. You can make more stack space by forgetting from the dictionary or dropping words from the stack. You have 256 bytes to use before the stack overruns the dictionary.

[[--
Mark position where ?] or]] will go back to. Alternate use is to mark the beginning of a series of conditionals like ?[or =?[and terminated by]]? .

\ --
Begin a comment. The comment is terminated by another \ of the end of a line.

]] --
Unconditional jump back to [[

]# --
End of indexed loop. If the index value at the top of the R-stack is non-zero, decrement the index value by one, and loop back to the start of the loop (just after the #[ ). If the index is zero, pop the value from the R-stack, and exit from the loop. Must be used within a colon definition.

]: --
Resume suspended : definition. The stack must be as it was when ;[was executed.

]? --
Demarks end of ?[logic.

][--
Otherwise. Seperates the 'true' and 'false' logic in conditional.

]] --
Repeat. Branches back to [and also marks where ?[will go when the proceeding ?[finds top 'false'

]]? --
Terminate a series of conditionals. Used in the sequence like:
[[... ?[... ?[...]]? or:
[[... =?[...][... =?[...][...]]?

@ addr -- n
Use top as an address to get word which replaces it.

~ --
Terminate a colon definition, if necessary, then delete the last dictionary entry from the GROWING vocabulary. Finally, print the name of the dictionary entry which is now the most recent.

ABS n1 -- n2
Replace value of top with its absolute value. Make positive.

ALLOT n --
Reserve number of bytes specified by top at the end of the dictionary. Frequently used to allow space in a table following a :BUILD definition.

AND n1 n2 -- n3
Logical, bit-for-bit, "and" of 2nd with top.

BASE -- n
Variable containing the radix for number conversions on input or output. Value must be greater than 1 and less than 127.

BINARY --
Sets the value of BASE to two.

BT -- seg addr
Gets the segment and address of the beginning of the text buffers.

BUF --

Defining word which creates a CONSTANT whose value is the beginning of a new buffer. Also terminates the last buffer.

BYE --
Leave Forth and return to the operating system.

C@ addr -- n
Replace the address at the top of the stack with the contents of the byte at the specified address. 8 high zero bits are appended to the byte.

C! n addr --
Store the least significant 8 bits of the value 2nd on the stack at the address specified on the top of the stack.

C, n --
Move right 8-bits from top word to top byte of dictionary, HERE. Then bump DP by one.

CDN dd mm yy -- cdn Top=YY, 2nd=MM, 3rd=DD.
Converts these three values to a value which is the number of days into the Century. This does not have only 28 days in February 1900, hence it is not reliable prior to 1 Jan 1901. 7 MOD of this result is the day of the week with 0 being Sunday.

CHAR n --
Father word which creates an N element vector. At reference time a single index is used and the address left on the stack is that of the N-th character in the vector.

CHOP addr --
Top must contain the address of a dictionary word. CHOP deletes all words back through that word from the dictionary.

CLIT -- n
A primitive word which is followed by a byte literal in a colon definition. The value of the literal is concatenated with 8 high order 0 bits and pushed on the stack when CLIT is executed.

CMOVE source dest count --
Character move. Top=byte count, 2nd=to address, 3rd=from address. Moves one byte at a time from low addresses to higher addresses. This is a "smart" move, which does not cause a fill.

COLD --
Deletes all words back through the word FORTH from the dictionary, recreates FORTH then executes WARM.

COMP n1 -- n2
Compliment the value at the top of the stack.

COMPILE --
Copy next word in this definition into the word being compiled. This word is not IMMEDIATE; it is frequently used in words which are IMMEDIATE.

CONSTANT n --
Defining word. The word defined will push the value which is now in top, when it is executed.

CR --
Sends a RETURN, LINE-FEED sequence to the terminal.

D+ d1 d2 -- d3
Replace the top two double numbers with their sum.

D- d1 d2 -- d3
Subtract the top double number on the stack from the second double number. Replace the two double numbers with that difference.

D0= dbl -- flag
If the double number at the top of the stack is 0.0, drop the double number and push a -1. Otherwise drop the double number and push a 0 .

D2* ud1 -- ud2
Shift left by 1 the double number at the top of the stack.

D2/ ud1 -- ud2
Shift right by 1 the double number at the top of the stack. The most significant bit of the result is 0.

D< d1 d2 -- flag
If the second double number on the stack is less than the top double number, replace them with a -1 . Otherwise replace them with a 0 .

DECIMAL --
Sets the value of BASE to ten.

DEFER -- (DEFER name)
Creates a DEFERed word. The operation of the word may be changed later by an operation such as ' name2 NEW name .

DEFS --
Makes GROWING have the value of SEARCH.

DICT --
Prints out the location, in hex, and the names in the SEARCH vocabulary, from last defined down. DICT prints slow for ease of reading and to be stoppable, it uses SCR.

DLIT -- dbl
A primitive to be used during compilation, and typically followed by a double-word in-line literal. When DLIT is executed, the two following words are pushed onto the stack.

DMY cdn -- dd mm yy
Converts the Century Day number to Year Month and Day. Top=Year, 3rd=day.

DOS ds dx cx bx ax -- ax' ds dx cx
Make a DOS 21H call to request some DOS service.

DP -- addr
Variable which points to the next free byte at the top of the dictionary.

DROP n --
Drops top from the stack.

DSADDR -- ds
Push the Data Segment DS on the stack.

DUP n -- n n
Duplicates the top word on the stack.

ECHO -- addr
Variable yeilding the address of an echo flag. If the flag is zero, the normal characters will not be echoed on input. This is useful when input has been re-directed from a file, and you do not wish to see the text displayed on the screen.

EMIT char --
Puts the low order 8 bits from top out to the terminal as a character.

ESC --
Emits an ASCII Escape Character. It avoids having the escape character in the definition

EXC addr1 addr2 --
Exchanges the values of the two words whose addresses are in top two words on stack.

EXECUTE addr --
Execute the word whose address is on top.

FILL addr cnt value --
Fills a block of memory with any specific character. Top=the fill character, 2nd is the count of characters, 3rd is the low order address of the block.

FORGET --
Forget from the dictionary the word specified by the next string, and all the words which were defined after it.

FORTH --
This is an infinite loop of INTERPRET. It is used to invoke the entire FORTH system from within a word. Executing the word 'back-space' will return control to the word which used the word FORTH.

GCD n1 n2 -- n3
Finds the Greatest Common Divisor of two top words.

GET n -- val
Obtain the n-th item from the stack and push it on top. Note that 0 GET is equivalent to DUP .

GETCHAR -- char
Obtain the next character from either the keyboard, or the current input buffer, whichever is active. For input from the keyboard, bit 8 indicates an ALT or other special function key. Keyboard characters are normally echoed to the screen.

GO addr --
A Jump-to-Subroutine is executed going to the address in top. The JSR instruction leaves its return address on top, unless preserved in the subroutine the RTS will remove it.

GROWING -- addr
Variable whose value is a pointer to the vocabulary in which new definitions ar placed. HEAD, makes all new words go into the vocabulary specified by this variable. The value at addr is a vocabulary number in the range 0 to 31.

HEAD, --
Obtain the next word from the input stream and create a dictionary entry containing the name field and linkage, but no action part of the new word.

HERE -- addr

This simply places the address of the first free byte above the dictionary on top.

HEX --
Sets the value of BASE to sixteen.

I -- n
Pushes a copy of the top of the R-stack. Used to get the index value in an indexed FOR-NEXT loop.

IMM --
Makes the last defined word have the property that it executes when used inside a : definition rather than be compiled. Used to create the conditional and looping words.

INCNT -- n
Return the number of characters in the circular input buffer.

INSTALL addr --
Install the vocabulary specified by the address in the SEARCHING vocabulary.

INTERPRET --
Input a string and interpret and execute it.

J -- n
Pushes a copy of the 2nd word from the R-stack. This is the index of the outer of two nested indexed loops. To get deeper nesting use: RP@ value + @

KEY -- n
Read a character from the keyboard and push it with 8 leading zero bits as a word. There is no echo of the character. If the character would have been a special character, it is shifted right by 8 bits and bit 8 is set to a one.

KIN --
Executes FORTH if there are any characters in the input buffer. Used to provide the ability to stop when a character is received from the keyboard. The time it delays is determined by the count value in RATE.

LIT n --
Useful, only within a : definition to get the value of the word which follows pushed onto the stack. Note the following word can not be an Immediate word, for this to work.

LT -- segment addr
Pushes the segment and address of the Last Text entered into the text buffer. It can be used with MT to discard the last text.

M* n1 n2 -- udbl
Unsigned multiply 2nd by top. Leave 32 bit result with high order part on top and low order part 2nd.

MAX n1 n2 -- n3
Replace 2nd and top with the larger of the two.

MEM -- n
Gets the number of bytes of free memory available for stack and dictionary entries.

MIN n1 n2 -- n3
Replace 2nd and top with the smaller of the two.

MINUS n1 -- n2
Negate the top value.

MOD n1 n2 -- n3
Divide 2nd by top and leave the remainder.

MT segment addr --
Purges text buffer down to the segment and address which is contained in top.

NEG n1 -- n2
Negate the top item on the stack.

NEW n --
Puts the value in top into the constant whose name follows. This is the safe way to change the value of a :CON

NIP n1 n2 -- n2
Deletes the second item on the stack.

NUM -- n
This requests a number from the keyboard and puts it on top.

OCTAL --
Sets the value of BASE to eight.

OPEN seg addr -- handle
Open a file for reading and writing. The segment and address point to an ASCIIZ string specifying the file. The "handle" is returned.

OR n1 n2 -- n3
Logical, bit-for-bit, "OR" of 2nd with top.

OVER n1 n2 -- n1 n2 n1
Push a copy of the second item on the stack.

P@ addr -- n
Fetch the 16 bit value from the port address specified.

P! n addr --
Store the 16 bit value at the port address specified.

PAD -- addr
The address of a temporary storage area, generally placed at SO + 2.

PC@ addr n --
Fetch the 8 bit value from the port address specified.

PC! n addr --
Store the 8 bit value at the port address specified.

PH addr1 -- addr2
Prints line of hex from address on top through the end of that hex decade. It leaves the address of the next hex decade on the stack so that successive lines are dumped if PH is used several times. This is the basis for DUMP.

PRE addr1 -- addr2 addr1
Top must contain a word address. PRE puts the address of the preceeding dictionary word under it. PRE calls QUIT when at the end of the dictionary. Used in the dictionary printing words.

PREVIOUS --
Removes the most accessible vocabulary from the search order and restores the previous search order.

PROG --
Enters text into the text buffer as does TEXT, but this then automatically RUNs the text which was just entered. This is useful in the development of programs, because it captures in the text buffer the source of each definition. See REDO.

PUT val n --
Store "val" at the n-th item on the stack, losing the previous contents. Note that -1 0 PUT will replace the top item with a value of -1 .

QUIT --
Clears the computational and R-stacks, then pushes the address of the input area, zero for keyboard, and executes CR.

R> -- n
Removes top of R-stack and pushes it onto the stack.

RAND -- n
Pushes a word of random bits onto the stack. The seed is two words at INPTR 2- and INPTR 4 - . The sum of the two words in the seed must be odd.

RDROP --
The value at the top of the R-stack is dropped.

READ seg addr count handle -- count'
Read from the file whose handle is specified into the buffer at the segment and address is given.

REDO --
This empties the last entered text in the TEXT buffer, and then does a PROG. Useful with PROG in capturing the source of new word definitions.

RET --
Returns control to word which called the present one. Generated by semi-colon. Return address must be on top or R-stack, hence cannot be used inside of #[ ]# or if >R values are on R-stack. It can be used within conditionals to abort.

ROOT --
The basic default vocabulary. WARM establishes this as both SEARCH and GROWING.

ROT n1 n2 n3 -- n2 n3 n1
Move 3rd word in the stack to top.

RP! --
Restores the Return Stack to its initial state. Clears the Return Stack.

RUN seg addr --
Transfers top to IN. Used to execute from a text buffer. Contents of the text buffer are read from low address to high and control returns to the keyboard when a NULL character is encountered.

S= --
Nondestructively prints out the entire contents of the stack.

SCAN addr char -- addr count
Scan the string beginning at addr until char or a delimiter occurs. The delimiter is at addr+count.

SCNT -- n
Pushes count of words on the stack, not counting itself.

SCOMP addr --
Complements the sign bit of the byte addressed by top.

SEARCH -- addr
Variable whose value is changed by executing a vocabulary word. SEARCH @ points to a pointer within the vocabulary word. ' and its assembly search routine (' depend on the value in SEARCH and GROWING to find words in the dictionary.

SIGN n -- n
Outputs a dash if the value on top is negative.

SKIP addr1 char -- addr2
Skip over leading occurrences of char at the string beginning at addr1. Leave the address addr2 which points to the first character not equal to char .

SMUDGE --
Makes the current dictionary header invisible. Used in : to allow renaming instead of recursion.

SO -- addr
Pushes address of stack orign.

SP! --
Empties the computational stack.

SP@ -- addr
Gets the address of the top of the computational stack then pushes that value.

SPACE --
Outputs one space character to the terminal.

SPACES n --
Outputs the number of space characters specified by top.

STATE -- n
Variable whose value is 0 when not compiling in a : definition. Used to allow IMMEDIATE words which behave differently when compiling than when simply executed from the input stream.

SWAB n1 -- n2
Exchanges the right and left 8-bit bytes in top.

SWAP n1 n2 -- n2 n1
Interchange the values in top and 2nd.

SYS --
This is the vocabulary where words which are primitives used in the implementation of FORTH are hidden.

T: --
Find the dictionary entry of the next word in the input stream. Put the address of the word in a Break detector.

TEXT! addr1 n -- addr2
Store characters in ascending addresses, top=delimiter, 2nd=address. Terminates when delimiter is reached. A null is stored in place of the delimiter, and its address is put on the stack.

TP -- address
Returns the address of the Text Pointer of the current buffer in the buffer segment.

U< n1 n2 -- flag
Treat the top two items on the stack as unsigned integers. If the second item is less than the top item, replace them with a value of -1 . Otherwise, replace them with a 0 .

UD< d1 d2 -- flag
Treat the top two double numbers on the stack as unsigned double integers. If the second double number is less than the first, replace them with a -1 . Otherwise replace them with a value of 0.

UD* ud1 ud2 -- uquad
Unsigned multiply of two double precision numbers, yielding a quadruple precision result.

UDMOD/ uquad udbl -- udquot udrem
Divide a quad precision number by a double precision number. Note that the unsigned double quotient is 2nd on the stack and the unsigned double remainder is on the top.

USER -- addr
Variable which points to top of a user constant area. Typical use is: n :CON xxx 2 USER +!

TIME -- lo hi
Returns the double precision "tick" from the IBM clock.

VOCNUM -- addr
The address returned is the location of the most recently assigned vocabulary number. The value at the address should be in the range from 0 to 31.

VOCTABLE-- addr
The address returned is the location of the Vocabulary table. There are 32 word entries in the table. Each word points to the first character of the name of a word in a linked list.

WARM --
Executes DECIMAL ROOT DEFS CR , issues the entry message and calls QUIT .

WDAYS dd1 mm1 yy1 dd2 mm2 yy2 -- n
This takes two calander dates in the CDN form and calculates the number of working days (Monday - Friday) there has been between the two dates.

WENDS cdn -- n
This takes a Century Day Number and calculates the number of Saturdays and Sundays which occured in the century. This is used in WDAYS, the difference between the two values is accurate; the number of week-end days may be off by a fixed amount.

X@ segment addr -- n
Replace the address and segment with the contents of the word addressed.

X! n segment addr --
Store the 16 bit value which is 3rd on the stack at the address specified by the segment and offset. Drop all three items from the stack.

XC@ segment addr -- n
Replace the address and segment with the contents of the byte addressed.

XC! n segment address --
Store the 8 bit value which is 3rd on the stack at the address specified by the segment and offset. Drop all three items from the stack.

XOR n1 n2 -- n3
Logical, bit-by-bit, exclusive OR of 2nd with top.