prev text next

12. REQUIRED WORD SET

12.1 The Required Word Set Layers

The words of the Required Word Set are grouped to show like characteristics. No implementation requirements should be inferred from this grouping.

Nucleus layer

```
    !  *  */  */MOD  +  +!  -  /  /MOD  0<  0=  0>  1+  1-  2+
    2-  2/  <  =  >  >R  ?DUP  @  ABS  AND  C!  C@  CMOVE
    CMOVE>  COUNT  D+  D<  DEPTH  DNEGATE  DROP  DUP  EXECUTE
    EXIT  FILL  I  J  MAX  MIN  MOD  NEGATE  NOT  OR  OVER  PICK
    R>  R@  ROLL  ROT  SWAP  U<  UM*  UM/MOD  XOR
```

Device layer

```
    BLOCK  BUFFER  CR  EMIT  EXPECT  FLUSH  KEY SAVE-BUFFERS
    SPACE  SPACES  TYPE  UPDATE
```

Interpreter layer

```
    #  #>  #S  #TIB  '  (  -TRAILING  .  .(  <#  >BODY  >IN
    ABORT  BASE  BLK  CONVERT  DECIMAL  DEFINITIONS  FIND
    FORGET  FORTH  FORTH-83  HERE  HOLD  LOAD  PAD  QUIT  SIGN
    SPAN  TIB  U.  WORD
```

Compiler layer

```
    +LOOP  ,  ."  :  ;  ABORT"  ALLOT  BEGIN  COMPILE  CONSTANT
    CREATE  DO  DOES>  ELSE  IF  IMMEDIATE  LEAVE  LITERAL  LOOP
    REPEAT  STATE  THEN  UNTIL  VARIABLE  VOCABULARY  WHILE  
    [']  [COMPILE]  ]
```

12.2 The Required Word Set Glossary

```
!            16b addr --                  79            "store"
     16b is stored at addr.

#            +d1 -- +d2                   79            "sharp"
     The remainder of +d1 divided by the value of BASE is
     converted to an ASCII character and appended to the output
     string toward lower memory addresses.  +d2 is the quotient
     and is maintained for further processing.  Typically used
     between <# and #> .

#>           32b -- addr +n               79    "sharp-greater"
     Pictured numeric output conversion is ended dropping 32b.
     addr is the address of the resulting output string.  +n is
     the number of characters in the output string.  addr and +n
     together are suitable for TYPE .

#S           +d -- 0 0                    29          "sharp-s"
     +d is converted appending each resultant character into the
     pictured numeric output string until the quotient (see: # )
     is zero.  A single zero is added to the output string if the
     number was initially zero.  Typically used between <# and
     #> .

#TIB         -- addr                      U,83   "number-t-i-b"
     The address of a variable containing the number of bytes in
     the text input buffer.  #TIB is accessed by WORD when BLK is
     zero.  {{0..capacity of TIB}}  See:  "input stream"

'            -- addr                      M,83           "tick"
     Used in the form:
             ' <name>
     addr is the compilation address of <name>.  An error
     condition exists if <name> is not found in the currently
     active search order.

(            --                           I,M,83        "paren"
             --   (compiling)
     Used in the form:
             ( ccc)
     The characters ccc, delimited by ) (closing parenthesis),
     are considered comments.  Comments are not otherwise
     processed.  The blank following ( is not part of ccc.  ( may
     be freely used while interpreting or compiling.  The number
     of characters in ccc may be zero to the number of characters
     remaining in the input stream up to the closing parenthesis.

*            w1 w2 -- w3                  79            "times"
     w3 is the least-significant 16 bits of the arithmetic
     product of w1 times w2.
```

12. REQUIRED WORD SET

*/ n1 n2 n3 -- n4 83 "times-divide"
n1 is first multiplied by n2 producing an intermediate 32-bit result. n4 is the floor of the quotient of the intermediate 32-bit result divided by the divisor n3. The product of n1 times n2 is maintained as an intermediate 32-bit result for greater precision than the otherwise equivalent sequence: n1 n2 * n3 / . An error condition results if the divisor is zero or if the quotient falls outside of the range {-32,768..32,767}.
See: "division, floored"

*/MOD n1 n2 n3 -- n4 n5 83 "times-divide-mod"
n1 is first multiplied by n2 producing an intermediate 32-bit result. n4 is the remainder and n5 is the floor of the quotient of the intermediate 32-bit result divided by the divisor n3. A 32-bit intermediate product is used as for */ . n4 has the same sign as n3 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the range {-32,768..32,767}.
See: "division, floored"

+ w1 w2 -- w3 79 "plus"
w3 is the arithmetic sum of w1 plus w2.

+! w1 addr -- 79 "plus-store"
w1 is added to the w value at addr using the convention for + . This sum replaces the original value at addr.

+LOOP n -- C,I,83 "plus-loop"
sys -- (compiling)
n is added to the loop index. If the new index was incremented across the boundary between limit-1 and limit then the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues to just after the corresponding DO . sys is balanced with its corresponding DO . See: DO

, 16b -- 79 "comma"
ALLOT space for 16b then store 16b at HERE 2- .

- w1 w2 -- w3 79 "minus"
w3 is the result of subtracting w2 from w1.

-TRAILING addr +n1 -- addr +n2 79 "dash-trailing"
The character count +n1 of a text string beginning at addr is adjusted to exclude trailing spaces. If +n1 is zero, then +n2 is also zero. If the entire string consists of spaces, then +n2 is zero.

. n -- M,79 "dot"
The absolute value of n is displayed in a free field format with a leading minus sign if n is negative.

12. REQUIRED WORD SET

```
."           --                           C,I,83    "dot-quote"
             --   (compiling)
     Used in the form:
             ." ccc"
     Later execution will display the characters ccc up to but
     not including the delimiting " (close-quote).  The blank
     following ." is not part of ccc.

.(           --                           I,M,83    "dos-paren"
             --                           (compiling)
     Used in the form:
             .( ccc)
     The characters ccc up to but not including the delimiting )
     (closing parenthesis) are displayed.  The blank following .(
     is not part of ccc.

/            n1 n2 -- n3                  83           "divide"
     n3 is the floor of the quotient of n1 divided by the divisor
     n2.  An error condition results if the divisor is zero or if
     the quotient falls outside of the range {-32,768..32,767}.
     See:  "division, floored"

/MOD         n1 n2 -- n3 n4               83       "divide-mod"
     n3 is the remainder and n4 the floor of the quotient of n1
     divided by the divisor n2.  n3 has the same sign as n2 or is
     zero.  An error condition results if the divisor is zero or
     if the quotient falls outside of the range
     {-32,768..32,767}.  See:  "division, floored"

0<           n -- flag                    83        "zero-less"
     flag is true if n is less than zero (negative).

0=           w -- flag                    83      "zero-equals"
     flag is true if w is zero.

0>           n -- flag                    83     "zero-greater"
     flag is true if n is greater than zero.

1+           w1 -- w2                     79         "one-plus"
     w2 is the result of adding one to w1 according to the
     operations of + .

1-           w1 -- w2                     79        "one-minus"
     w2 is the result of subtracting one from w1 according to the
     operation of - .

2+           w1 -- w2                     79         "two-plus"
     w2 is the result of adding two to w1 according to the
     operation of + .

2-           w1 -- w2                     79        "two-minus"
     w2 is the result of subtracting two from w1 according to the
```

operation of - .

12. REQUIRED WORD SET

```
2/          n1 -- n2                      83       "two-divide"
```
n2 is the result of arithmetically shifting n1 right one bit. The sign is included in the shift and remains unchanged.

```
:           -- sys                        M,79          "colon"
```
A defining word executed in the form:

```
        : <name> ... ;
```
Create a word definition for <name> in the compilation vocabulary and set compilation state. The search order is changed so that the first vocabulary in the search order is changed so that the first vocabulary in the search order is replaced by the compilation vocabulary. The compilation vocabulary is unchanged. The text from the input stream is subsequently compiled. <name> is called a "colon definition". The newly created word definition for <name> cannot be found in the dictionary until the corresponding ; or ;CODE is successfully processed.

An error condition exists if a word is not found and cannot be converted to a number or if, during compilation from mass storage, the input stream is exhausted before encountering ; or ;CODE . sys is balanced with its corresponding ; . See: "compilation" "9.4 Compilation"

```
;           --                            C,I,79   "semi-colon"
            sys --   (compiling)
```
Stops compilation of a colon definition, allows the <name> of this colon definition to be found in the dictionary, sets interpret state and compiles EXIT (or a system dependent word which performs an equivalent function). sys is balanced with its corresponding : . See: EXIT : "stack, return" "9.4 Compilation"

```
<           n1 n2 -- flag                 83        "less-than"
```
flag is true if n1 is less than n2.

```
        -32678 32767 < must return true.
        -32768 0 < must return true.
```

```
<#          --                            79       "less-sharp"
```
Initialize pictured numeric output conversion. The words:

```
        #  #>  #S  <#  HOLD  SIGN
```
can be used to specify the conversion of a double number into an ASCII text string stored in right-to-left order.

```
=           w1 w2 -- flag                 83           "equals"
```
flag is true if w1 is equal to w2.

```
>           n1 n2 -- flag                 83     "greater-than"
```

```
     flag is true if n1 is greater than n2.
             -32768 32767 > must return false.
             -32768 0 > must return false.
```

12. REQUIRED WORD SET

```
>BODY        addr1 -- addr2                 83          "to-body"
     addr2 is the parameter field address corresponding to the
     compilation address addr1.  See:  "9.2 Addressable Memory"

>IN          -- addr                        U,79          "to-in"
     The address of a variable which contains the present
     character offset within the input stream {{0..the number of
     characters in the input stream}}.  See:  WORD

>R           16b --                         C,79           "to-r"
     Transfers 16b to the return stack.  See  "9.3 Return Stack"

?DUP         16b -- 16b 16b                 79    "question-dupe"
     or      0 -- 0
     Duplicate 16b if it is non-zero.

@            addr -- 16b                    79            "fetch"
     16b is the value at addr.

ABORT                                       79
     Clears the data stack and performs the function of QUIT .
     No message is displayed.

ABORT"       flag --                        C,I,83  "abort-quote"
             --   (compiling)
     Used in the form:
             flag ABORT" ccc"
     When later executed, if flag is true the characters ccc,
     delimited by " (close-quote), are displayed and then a
     system dependent error abort sequence, including the
     function of ABORT , is performed.  If flag is false, the
     flag is dropped and execution continues.  The blank
     following ABORT" is not part of ccc.

ABS          n -- u                         79         "absolute"
     u is the absolute value of n.  If n is -32,768 then u is the
     same value.  See:  "arithmetic, two's complement"

ALLOT        w --                           79
     Allocates w bytes in the dictionary.  The address of the
     next available dictionary entry is updated accordingly.

AND          16b1 16b2 -- 16b3              79
     16b3 is the bit-by-bit logical 'and' of 16b1 with 16b2.

BASE         -- addr                        U,83
```

The address of a variable containing the current numeric conversion radix. {{2..72}}

12. REQUIRED WORD SET

```
BEGIN        --                          C,I,79
             -- sys   (compiling)
```
Used in the form:
```
        BEGIN ... flag UNTIL
or
        BEGIN ... flag WHILE ... REPEAT
```
BEGIN marks the start of a word sequence for repetitive execution. A BEGIN-UNTIL loop will be repeated until flag is true. A BEGIN-WHILE-REPEAT will be repeated until flag is false. The words after UNTIL or REPEAT will be executed when either loop is finished. sys is balanced with its corresponding UNTIL or WHILE .
See: "9.9 Control Structures"

```
BLK          -- addr                     U,79          "b-l-k"
```
The address of a variable containing the number of the mass storage block being interpreted as the input stream. If the value of BLK is zero the input stream is taken from the text input buffer. {{0..the number of blocks available -1}}
See: TIB "input stream"

```
BLOCK        u -- addr                   M,83
```
addr is the address of the assigned buffer of the first byte of block u. If the block occupying that buffer is not block u and has been UPDATEed it is transferred to mass storage before assigning the buffer. If block u is not already in memory, it is transferred from mass storage into an assigned block buffer. A block may not be assigned to more than one buffer. If u is not an available block number, an error condition exists. Only data within the last buffer referenced by BLOCK or BUFFER is valid. The contents of a block buffer must not be changed unless the change may be transferred to mass storage.

```
BUFFER       u -- addr                   M,83
```
Assigns a block buffer to block u. addr is the address of the first byte of the block within its buffer. This function is fully specified by the definition for BLOCK except that if the block is not already in memory it might not be transferred from mass storage. The contents of the block buffer assigned to block u by BUFFER are unspecified.

```
C!           16b addr --                 79          "c-store"
```

```
     The least-significant 8 bits of 16b are stored into the byte
     at addr.

C@           addr -- 8b                    79          "c-fetch"
     8b is the contents of the byte at addr.

CMOVE        addr1 addr2 u --              83           "c-move"
     Move u bytes beginning at address addr1 to addr2.  The byte
     at addr1 is moved first, proceeding toward high memory.  If
     u is zero nothing is moved.
```

```
CMOVE>       addr1 addr2 u --              83        "c-move-up"
     Move the u bytes at address addr1 to addr2.  The move begins
     by moving the byte at (addr1 plus u minus 1) to (addr2 plus
     u minus 1) and proceeds to successively lower addresses for
     u bytes.  If u is zero nothing is moved.  (Useful for
     sliding a string towards higher addresses).

COMPILE      --                            C,83
     Typically used in the form:
             : <name> ... COMPILE <namex> ... ;
     When <name> is executed, the compilation address compiled
     for <namex> is compiled and not executed.  <name> is
     typically immediate and <namex> is typically not immediate.
     See:  "compilation"

CONSTANT     16b --                        M,83
     A defining word executed in the form:
             16b CONSTANT <name>
     Creates a dictionary entry for <name> so that when <name> is
     later executed, 16b will be left on the stack.

CONVERT      +d1 addr1 -- +d2 addr2        79
     +d2 is the result of converting the characters within the
     text beginning at addr1+2 into digits, using the value of
     BASE , and accumulating each into +d1 after multiplying +d1
     by the value of BASE .  Conversion continues until an
     unconvertible character is encounter.  addr2 is the location
     of the first unconvertible character.

COUNT        addr1 -- addr2 +n             79
     addr2 is addr1+1 and +n is the length of the counted string
     at addr1.  The byte at addr1 contains the byte count +n.
     Range of +n is {0.255}  See:  "string, counted"

CR           --                            M,79            "c-r"
     Displays a carriage-return and line-feed or equivalent
     operation.

CREATE       --                            M,79
     A defining word executed in the form:
```

CREATE <name>

Creates a dictionary entry for <name>. After <name> is created, the next available dictionary location is the first byte of <name>'s parameter field. When <name> is subsequently executed, the address of the first byte of <name>'s parameter field is left on the stack. CREATE does not allocate space in <name>'s parameter field.

```
D+           wd1 wd2 -- wd3                79           "d-plus"
```
wd3 is the arithmetic sum of wd1 plus wd2.

```
D<           d1 d2 -- flag                 83      "d-less-than"
```
flag is true if d1 is less than d2 according to the operation of < except extended to 32 bits.

```
DECIMAL      --                            79
```
Set the input-output numeric conversion base to ten.

```
DEFINITIONS  --                            79
```
The compilation vocabulary is changed to be the same as the first vocabulary in the search order.
See: "vocabulary, compilation"

```
DEPTH        -- +n                         79
```
+n is the number of 16-bit values contained in the data stack before +n was placed on the stack.

```
DNEGATE      d1 -- d2                      79         "d-negate"
```
d2 is the two's complement of d1.

```
DO           w1 w2 --                      C,I,83
             -- sys   (compiling)
```
Used in the form:
```
        DO ... LOOP
```
or
```
        DO ... +LOOP
```
Begins a loop which terminates based on control parameters. The loop index begins at w2, and terminates based on the limit w1. See LOOP and +LOOP for details on how the loop is terminated. The loop is always executed at least once. For example: w DUP DO ... LOOP executes 65,536 times. sys is balanced with its corresponding LOOP or +LOOP .
See: "9.9 Control Structures"

An error condition exists if insufficient space is available for at least three nesting levels.

```
DOES>        -- addr                       C,I,83         "does"
             --   (compiling)
```
Defines the execution-time action of a word created by a high-level defining word. Used in the form:
```
        : <namex> ... <create> ... DOES> ... ;
```

```
     and then
             <namex> <name>
     where <create> is CREATE or any user defined word which
     executes CREATE .

     Marks the termination of the defining part of the defining
     word <namex> and then begins the definition of the
     execution-time action for words that will later be defined
     by <namex>.  When <name> is later executed, the address of
     <name>'s parameter field is placed on the stack and then the
     sequence of words between DOES> and ; are executed.

DROP         16b --                         79
     16b is removed from the stack.

DUP          16b -- 16b 16b                 79               "dupe"
     Duplicate 16b.
```

12. REQUIRED WORD SET

```
ELSE         --                             C,I,79
             sys1 -- sys2   (compiling)
     Used in the form:
             flag IF ... ELSE ... THEN
     ELSE executes after the true part following IF .  ELSE
     forces execution to continue at just after THEN .  sys1 is
     balanced with its corresponding IF .  sys2 is balanced with
     its corresponding THEN .  See:  IF  THEN

EMIT         16b --                         M,83
     The least-significant 7-bit ASCII character is displayed.
     SEE:  "9.5.3 EMIT"

EXECUTE      addr --                        79
     The word definition indicated by addr is executed.  An error
     condition exists if addr is not a compilation address

EXIT         --                             C,79
     Compiled within a colon definition such that when executed,
     that colon definition returns control to the definition that
     passed control to it by returning control to the return
     point on the top of the return stack.  An error condition
     exists if the top of the return stack does not contain a
     valid return point.  May not be used within a do-loop.
     See: ;  "stack, return"  "9.3 Return Stack"

EXPECT       addr +n --                     M,83
     Receive characters and store each into memory.  The transfer
     begins at addr proceeding towards higher addresses one byte
     per character until either a "return" is received or until
     +n characters have been transferred.  No more than +n
     characters will be stored.  The "return" is not stored into
     memory.  No characters are received or transferred if +n is
     zero.  All characters actually received and stored into
```

memory will be displayed, with the "return" displaying as a space. See: SPAN "9.5.2 EXPECT"

```
FILL         addr u 8b --                  83
```
u bytes of memory beginning at addr are set to 8b. No action is taken if u is zero.

```
FIND         addr1 -- addr2 n              83
```
addr1 is the address of a counted string. The string contains a word name to be located in the currently active search order. If the word is not found, addr2 is the string address addr1, and n is zero. If the word is found, addr2 is the compilation address and n is set to one of two non-zero values. If the word found has the immediate attribute, n is set to one. If the word is non-immediate, n is set to minus one (true).

```
FLUSH        --                            M,83
```
Performs the function of SAVE-BUFFERS then unassigns all block buffers. (This may be useful for mounting or changing mass storage media).

```
FORGET       --                            M,83
```
Used in the form:

```
        FORGET <name>
```
If <name> is found in the compilation vocabulary, delete <name> from the dictionary and all words added to the dictionary after <name> regardless of their vocabulary. Failure to find <name> is an error condition. An error condition also exists if the compilation vocabulary is deleted. See: "10.2 General Error Conditions"

```
FORTH        --                            83
```
The name of the primary vocabulary. Execution replaces the first vocabulary in the search order with FORTH . FORTH is initially the compilation vocabulary and the first vocabulary in the search order. New definitions become part of the FORTH vocabulary until a different compilation vocabulary is established. See: VOCABULARY

```
FORTH-83     --                            83
```
Assures that a FORTH-83 Standard System is available, otherwise an error condition exists.

```
HERE         -- addr                       79
```
The address of the next available dictionary location.

```
HOLD         char --                       79
```

```
     char is inserted into a pictured numeric output string.
     Typically used between <# and #>.

I            -- w                           C,79
     w is a copy of the loop index.  May only be used in the
     form:
             DO ... I ... LOOP
     or
             DO ... I ... +LOOP
```

```
IF           flag --                        C,I,79
             -- sys   (compiling)
     Used in the form:
             flag IF ... ELSE ... THEN
     or
             flag IF ... THEN
     If flag is true, the words following IF are executed and the
     words following ELSE until just after THEN are skipped.  The
     ELSE part is optional.

     If flag is false, the words from IF through ELSE , or from
     IF through THEN (when no ELSE is used), are skipped.  sys is
     balanced with its corresponding ELSE or THEN .
     See:  "9.9 Control Structures"

IMMEDIATE    --                             79
     Marks the most recently created dictionary entry as a word
     which will be executed when encountered during compilation
     rather than compiled.

J            -- w                           C,79
     w is a copy of the index of the next outer loop.  May only
     be used within a nested DO-LOOP or DO-+LOOP in the form, for
     example:
             DO ... DO ... J ... LOOP ... +LOOP

KEY          -- 16b                         M,83
     The least-significant 7 bits of 16b is the next ASCII
     character received.  All valid ASCII characters can be
     received.  Control characters are not processed by the
     system for any editing purpose.  Characters received by KEY
     will not be displayed.  See:  "9.5.1 KEY"

LEAVE        --                           C,I,83
             --   (compiling)
     Transfers execution to just beyond the next LOOP or +LOOP .
     The loop is terminated and loop control parameters are
     discarded.  May only be used in the form:
             DO ... LEAVE ... LOOP
     or
             DO ... LEAVE ... +LOOP
     LEAVE may appear within other control structures which are
     nested within the do-loop structure.  More than one LEAVE
     may appear within a do-loop.  See:  "9.3 Return Stack"

LITERAL      -- 16b                       C,I,79
             16b --   (compiling)
     Typically used in the form:
             [ 16b ] LITERAL
     Compiles a system dependent operation so that when later
     executed, 16b will be left on the stack.
```

```
LOAD         u --                         M,79
     The contents of >IN and BLK , which locate the current input
     stream, are saved.  The input stream is then redirected to
     the beginning of screen u by setting >IN to zero and BLK to
     u.  The screen is then interpreted.  If interpretation from
     screen u is not terminated explicitly it will be terminated
     when the input stream is exhausted and then the contents of
     >IN and BLK will be restored.  An error condition exists if
     u is zero.  See:  >IN  BLK  BLOCK

LOOP         --                           C,I,83
             sys --   (compiling)
     Increments the DO-LOOP index by one.  If the new index was
     incremented across the boundary between limit-1 and limit
     the loop is terminated and loop control parameters are
     discarded.  When the loop is not terminated, execution
     continues to just after the corresponding DO .  sys is
     balanced with its corresponding DO .  See:  DO

MAX          n1 n2 -- n3                  79               "max"
     n3 is the greater of n1 and n2 according to the operation of
     > .

MIN          n1 n2 -- n3                  79               "min"
     n3 is the lesser of n1 and n2 according to the operation of
     < .

MOD          n1 n2 -- n3                     83
     n3 is the remainder after dividing n1 by the divisor n2.  n3
     has the same sign as n2 or is zero.  An error condition
     results if the divisor is zero or if the quotient falls
     outside of the range {-32,768..32,767}.
     See:  "division, floored"

NEGATE       n1 -- n2                        79
     n2 is the two's complement of n1, i.e, the difference of
     zero less n1.

NOT          16b1 -- 16b2                    83
     16b2 is the one's complement of 16b1.

OR           16b1 16b2 -- 16b3               79
     16b3 is the bit-by-bit inclusive-or of 16b1 with 16b2.

OVER         16b1 16b2 -- 16b1 16b2 16b3     79
     16b3 is a copy of 16b1.

PAD          -- addr                         83
     The lower address of a scratch area used to hold data for
     intermediate processing.  The address or contents of PAD may
     change and the data lost if the address of the next
     available dictionary location is changed.  The minimum
     capacity of PAD is 84 characters.
```

```
PICK         +n -- 16b                       83
     16b is a copy of the +nth stack value, not counting +n
     itself.  {0..the number of elements on stack-1}
             0 PICK is equivalent to DUP
             1 PICK is equivalent to OVER

QUIT         --                              79
     Clears the return stack, sets interpret state, accepts new
     input from the current input device, and begins text
     interpretation.  No message is displayed.

R>           -- 16b                          C,79         "r-from"
     16b is removed from the return stack and transferred to the
     data stack.  See:  "9.3 Return Stack"

R@           -- 16b                          C,79        "r-fetch"
     16b is a copy of the top of the return stack.

REPEAT       --                              C,I,79
             sys --   (compiling)
     Used in the form:
             BEGIN ... flag WHILE ... REPEAT
```

At execution time, REPEAT continues execution to just after the corresponding BEGIN . sys is balanced with its corresponding WHILE . See: BEGIN

ROLL   +n --   83

The +nth stack value, not counting +n itself is first removed and then transferred to the top of the stack, moving the remaining values into the vacated position. {0..the number of elements on the stack-1}

```
        2 ROLL is equivalent to ROT
        0 ROLL is a null operation
```

ROT   16b1 16b2 16b3 -- 16b2 16b3 16b1   79   "rote"

The top three stack entries are rotated, bringing the deepest to the top.

SAVE-BUFFERS --   M,79   "save-buffers"

The contents of all block buffers marked as UPDATEed are written to their corresponding mass storage blocks. All buffers are marked as no longer being modified, but may remain assigned.

SIGN   n --   83

If n is negative, an ASCII "-" (minus sign) is appended to the pictured numeric output string. Typically used between <# and #> .

SPACE   --   M,79

Displays an ASCII space.

SPACES   +n --   M,79

Displays +n ASCII spaces. Nothing is displayed if +n is zero.

SPAN   -- addr   U,83

The address of a variable containing the count of characters actually received and stored by the last execution of EXPECT . See: EXPECT

STATE   -- addr   U,79

The address of a variable containing the compilation state. A non-zero content indicates compilation is occurring, but the value itself is system dependent. A Standard Program may not modify this variable.

SWAP   16b1 16b2 -- 16b2 16b1   79

The top two stack entries are exchanged.

THEN   --   C,I,79

```
        sys --   (compiling)
```
Used in the form:
```
        flag IF ... ELSE ... THEN
```
or
```
        flag IF ... THEN
```
THEN is the point where execution continues after ELSE , or IF when no ELSE is present. sys is balanced with its corresponding IF or ELSE . See: IF ELSE

```
TIB          -- addr                    83            "t-i-b"
```
The address of the text input buffer. This buffer is used to hold characters when the input stream is coming from the current input device. The minimum capacity of TIB is 80 characters.

```
TYPE         addr +n --                 M,79
```
+n characters are displayed from memory beginning with the character at addr and continuing through consecutive addresses. Nothing is displayed if +n is zero. See: "9.5.4 TYPE"

```
U.           u --                       M,79          "u-dot"
```
u is displayed as an unsigned number in a free-field format.

```
U<           u1 u2 -- flag              83      "u-less-than"
```
flag is true if u1 is less than u2.

```
UM*          u1 u2 -- ud                83        "u-m-times"
```
ud is the unsigned product of u1 times u2. All values and arithmetic are unsigned.

```
UM/MOD       ud u1 -- u2 u3             83   "u-m-divide-mod"
```
u2 is the remainder and u3 is the floor of the quotient after dividing ud by the divisor u1. All values and arithmetic are unsigned. An error condition results if the divisor is zero or if the quotient lies outside the range corresponding BEGIN . See: BEGIN

```
UPDATE       --                         79
```
The currently valid block buffer is marked as modified. Blocks marked as modified will subsequently be automatically transferred to mass storage should its memory buffer be needed for storage of a different block or upon execution of FLUSH or SAVE-BUFFERS .

```
VARIABLE     --                         M,79
```
A defining word executed in the form:

```
        VARIABLE <name>
A dictionary entry for <name> is created and two bytes are
ALLOTted in its parameter field.  This parameter field is to
be used for contents of the variable.  The application is
responsible for initializing the contents of the variable
which it creates.  When <name> is later executed, the
address of its parameter field is placed on the stack.
```

```
VOCABULARY    --                            M,83
     A defining word executed in the form:
             VOCABULARY <name>
     A dictionary entry for <name> is created which specifies a
     new ordered list of word definitions.  Subsequent execution
     of <name> replaces the first vocabulary in the search order
     with <name>.  When <name> becomes the compilation vocabulary
     new definitions will be appended to <name>'s list.
     See:  DEFINITIONS  "search order"
```

```
WHILE        flag --                       C,I,79
             sys1 -- sys2   (compiling)
     Used in the form:
             BEGIN ... flag WHILE ... REPEAT
     Selects conditional execution based on flag.  When flag is
     true, execution continues to just after the WHILE through to
     the REPEAT which then continues execution back to just after
     the BEGIN .  When flag is false, execution continues to just
     after the REPEAT , exiting the control structure.  sys1 is
     balanced with its corresponding BEGIN .  sys2 is balanced
     with its corresponding REPEAT .  See:  BEGIN
```

```
WORD         char -- addr                  M,83
     Generates a counted string by non-destructively accepting
     characters from the input stream until the delimiting
     character char is encountered or the input stream is
     exhausted.  Leading delimiters are ignored.  The entire
     character string is stored in memory beginning at addr as a
     sequence of bytes.  The string is followed by a blank which
     is not included in the count.  The first byte of the string
     is the number of characters {0..255}.  If the string is
     longer than 255 characters, the count is unspecified.  If
     the input stream is already exhausted as WORD is called,
     then a zero length character string will result.

     If the delimiter is not found the value of >IN is the size
     of the input stream.  If the delimiter is found >IN is
     adjusted to indicate the offset to the character following
     the delimiter.  #TIB is unmodified.

     The counted string returned by WORD may reside in the "free"
     dictionary area at HERE or above.  Note that the text
     interpreter may also use this area.  See:  "input stream"

XOR          16b1 16b2 -- 16b3            79             "x-or"
     16b3 is the bit-by-bit exclusive-or of 16b1 with 16b2.

[            --                           I,79   "left-bracket"
             --   (compiling)
     Sets interpret state.  The text from the input stream is
     subsequently interpreted.  For typical usage see LITERAL .
     See:  ]

[']          -- addr                      C,I,M,83    "bracket-
             --   (compiling)                              tick"
     Used in the form:
             ['] <name>
     Compiles the compilation address addr of <name> as a
     literal.  When the colon definition is later executed addr
     is left on the stack.  An error condition exists if <name>
     is not found in the currently active search order.
     See:  LITERAL
```

```
[COMPILE]    --                           C,I,M,79    "bracket-
             --   (compiling)                           compile"
     Used in the form:
             [COMPILE] <name>
     Forces compilation of the following word <name>.  This
     allows compilation of an immediate word when it would
     otherwise have been executed.

]            --                           79    "right-bracket"
     Sets compilation state.  The text from the input stream is
     subsequently compiled.  For typical usage see LITERAL .
     See:  [
```