CHAPTER 8. DICTIONARY AND VOCABULARY

The source code discussed in this chapter is in the file KERNEL86.BLK, screens 67-68, and 76.

If you had a figForth system, you might have noticed that it took a while to compile a screen of text. If you were to load a sizable system or application program, it might seem to be a long time before the computer came back and put an 'ok' on the screen. The reason is that the dictionary in figForth is basically a single, linearly linked list of words. It takes some time for the text interpreter to travel through this list to find a word. The worst cases are the numbers. If the text interpreter cannot find the word in the context vocabulary, it will search again in the current vocabulary, which in most cases is the same as the context vocabulary with the entire root Forth vocabulary tagged at the end.

F83 improves this situation by breaking the dictionary into four separately linked lists. To locate a word, only a quarter of the dictionary needs to be searched. This strategy visibly enhances the performance of the text interpreter. In this section, I hope that I can explain how this dictionary structure is implemented in F83.

8.1. THREADING OF THE DICTIONARY

First, there are several important system variables which perform the housekeeping chores in managing the vocabulary and the searching of dictionary:

```
VARIABLE DP                 Pointer to the top of the dictionary. Returned by HERE.
VARIABLE CURRENT            Pointer to the current vocabulary to which new definitions
                            are linked.
8 CONSTANT #VOC             The number of vocabularies to be searched, as specified by
                            the array in CONTEXT.
VARIABLE CONTEXT            The context vocabulary pointer.
#VOCS 2* ALLOT              Space to hold 8 transient vocabulary pointers. The array
                            specifies the search order for the text interpreter.
VARIABLE VOC-LINK           Pointer to the most recently defined vocabulary.
                            Vocabularies are thus linked in the order of their creation.
```

The 8 numbers stored in the transient array are the parameter field addresses of up to eight different vocabularies. The text interpreter searches up to eight vocabularies and stops at the first encounter of the name it looks for.

Next, let us see how the vocabularies are defined and how to select the context and current vocabularies.

```
: VOCABULARY  ( --- )       Define a new vocabulary.
        CREATE              Take the following string as the name of the new
                            vocabulary.
        #THREADS 0 DO       Compile four 0's in the parameter field.
              0 ,           They are the four threads
         LOOP               in the dictionary for the new vocabulary.
```

HERE The next cell is for the vocabulary link, VOC-LINK. VOC-LINK @ , Old vocabulary link is placed in this cell.

```
    VOC-LINK !          The new vocabulary is the last in the vocabulary link list. Its
                        link address must be stored in VOC-LINK.
    DOES>               End of the compilation of a new vocabulary entry in
                    dictionary. Next is the vocabulary interpreter:
    CONTEXT !           Store the parameter field address of this vocabulary in the
                    first cell of the CONTEXT array so that this vocabulary will ;
                    be searched first by the text interpreter.

: DEFINITIONS   ( --- )     Link subsequent definitions to the context vocabulary.
    CONTEXT @       Get the address of the context vocabulary.
    CURRENT !           Store it in CURRENT.  New definitions will be linked to the
    ;                   vocabulary pointed to by CURRENT.
```

The interesting things are how new definitions are linked to the current vocabulary and to the threads in the dictionary. Vocabularies are the logical groupings of definitions in the dictionary and the threads are the physical mechanism to group definitions in the dictionary. New definitions are created by CREATE, which invokes "CREATE to build the name fields and link fields. The fields in a definition are shown in Fig. 8.1. Vocabularies and threads are shown in Figures 8.2-3.

```
: "CREATE   ( --- )         Create a header for a new definition. The header consists of
                        a view field, a link field, and a name field.
    COUNT               Character count in the name.
    HERE EVEN 4 +       Address of the name field.
    PLACE               Move the name string into the name field.
    ALIGN               Align the header to a cell boundary because the view field
                    contains a 16 bit integer.
    ,VIEW           Lay down the view field in which the top 4 bits contain a file
                    number and the lower 12 bits contain a block number in the
                    file.
    HERE 0 ,            Save a cell for the link field to be filled later.
    HERE LAST !     Store the name field address in LAST.
    HERE   ( lfa nfa )      Get the name field address.
    WARNING @       If the warning flag is set, search the dictionary to see if the
                name is unique.
    IF FIND             If it is an existing name,
        IF HERE COUNT TYPE      print the name, ."  isn't unique" with an error
                    message.
        THEN
        DROP HERE       Clean the stack after FIND.
    THEN
    CURRENT @ HASH Hash the first character of the name with the current
                    vocabulary to return one of the four threads to be extended. DUP   @
                Get the name field address of the last definition of this
        thread.
    HERE 2-             The link field address of the current definition.
    ROT !               Store this link field address in the head of thread in the
                    current vocabulary.
    SWAP !              Store the link field address of the last definition in the link
```

```
                field of the current definition and extend the linked chain.    HERE
            Name field address saved on stack.
DUP C@                  Character length of name.
```

Figure 8.1 Structure of a Forth definition

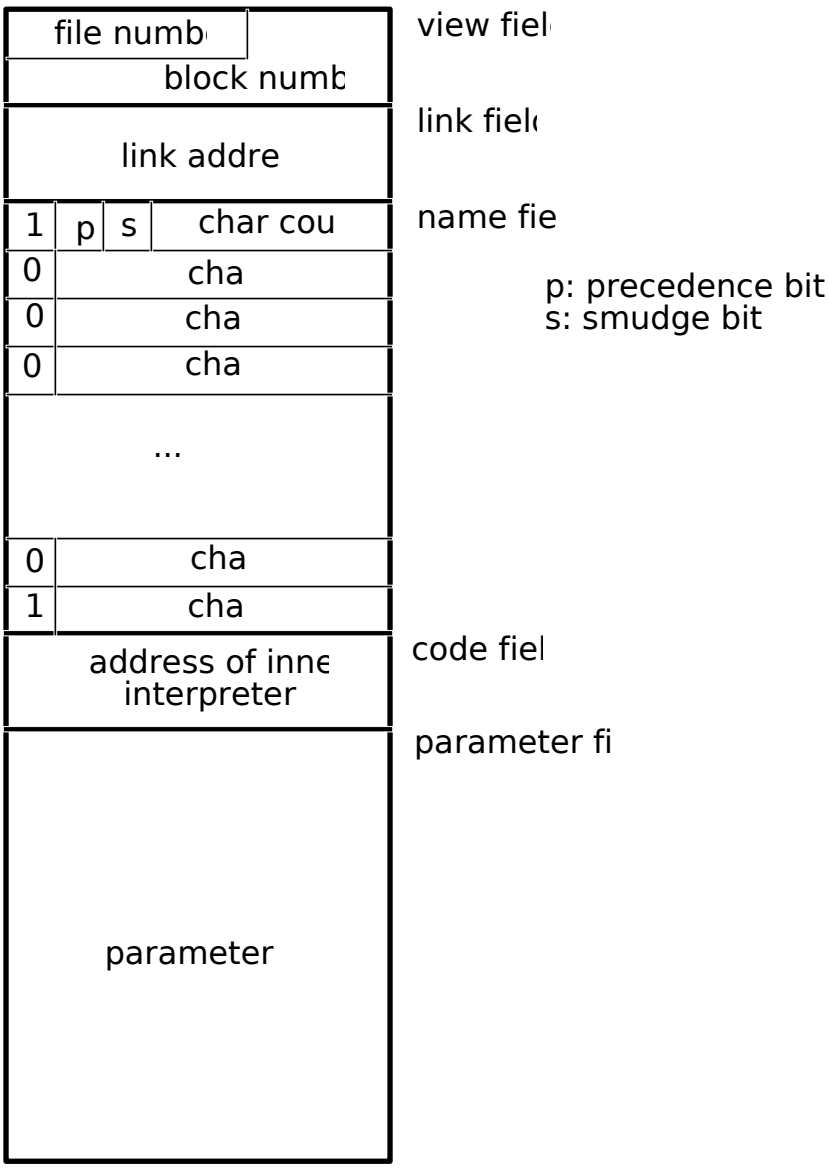


Figure 8.2 Vocabularies and the dictionary structure.

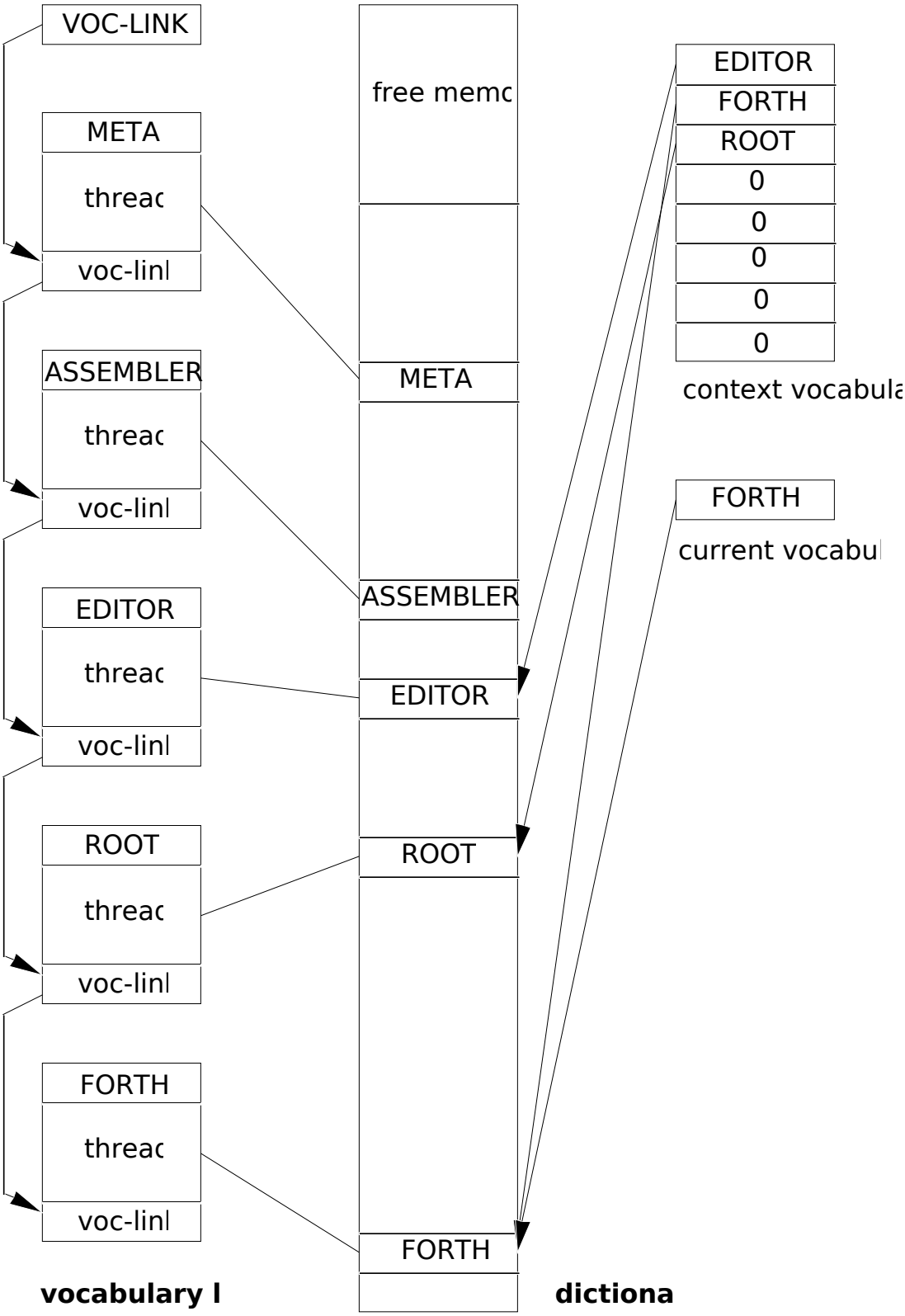
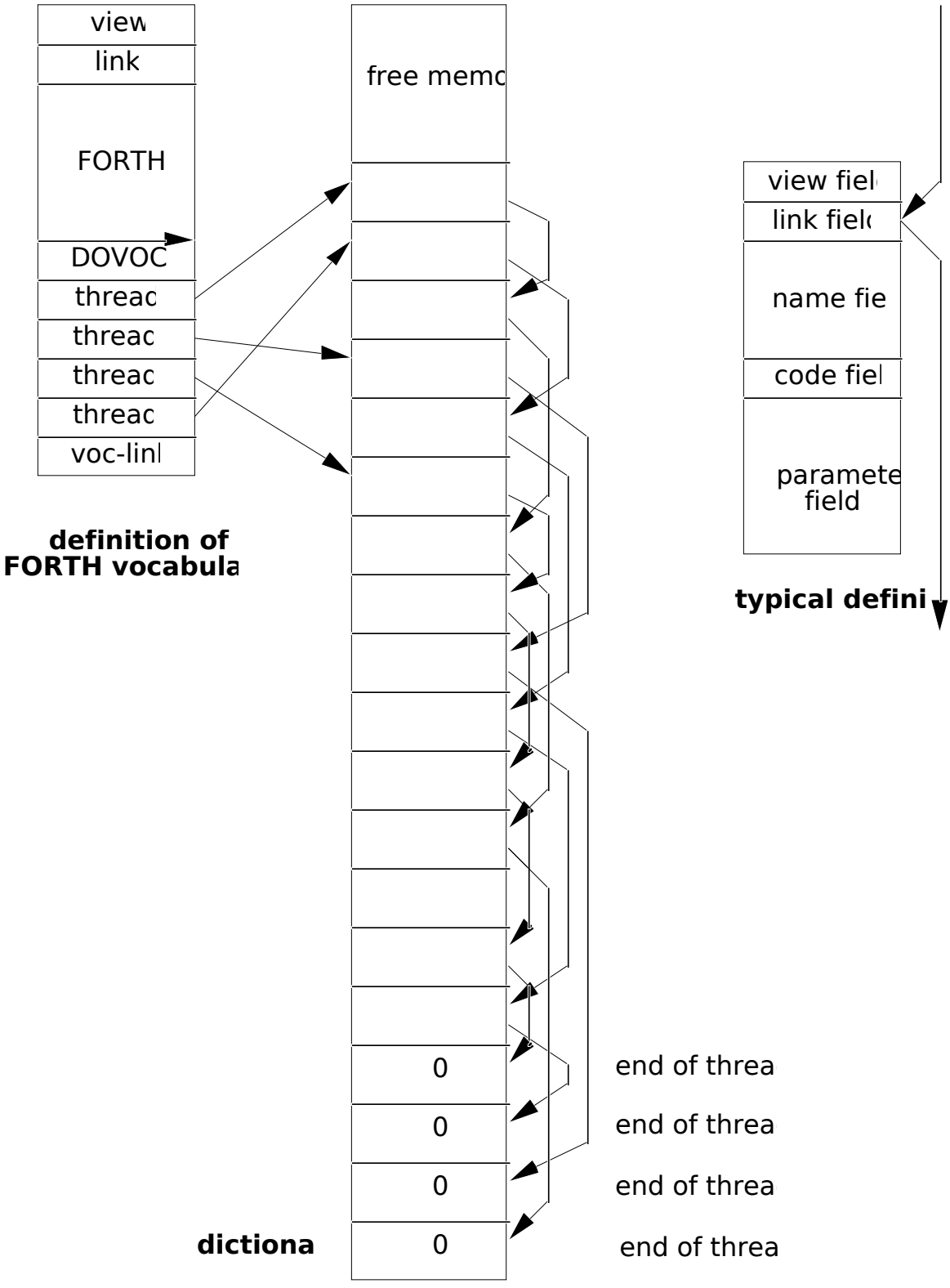


Figure 8.3 Four-way threading in the vocabulary.



```
WIDTH @ MIN 1+          Width of the name field.
ALLOT ALIGN             Name field allocated.
128 SWAP CSET           Set the MSB of the length byte, which is the first byte in
                        name field, as a name field delimiter.
128 HERE 1- CSET        Set the MSB of the last byte in name field as another
                        delimiter.
COMPILE [               Turn on the interpreter.
DOCREATE ,              Compile the variable interpreter in the code field.
;                       Thus complete the header.
```

The header in this F83 Forth is not the same as the other more popular Forth systems. A view field is added to help the user locating a definition in one of the CP/M files containing Forth source screens. The link field is placed before the name field so that the string comparisons can be performed more quickly without traversing through the name field. The linking of dictionary entries involves only the link fields. Name fields are no longer involved in the linkage.

8.2. HASHING AND SEARCHING THE DICTIONARY

Two important words in "CREATE above were not fully explained: HASH and FIND. These are the key definitions used by the text interpreter to search specific words in the dictionary. HASH is a code definition. FIND, however, is a high level colon definition which eventually calls a code definition (FIND) to do the actual searching. Let us first look at HASH and (FIND):

```
CODE  HASH   ( string-addr vocabulary-pfa --- thread-addr )        Given a string address and a
                                pointer to a vocabulary, return the address of the thread in
                        the parameter field of the vocabulary.
        CX POP                  Pfa of the vocabulary.
        BX POP                  Address of the string.
        BX INC                  Address of the first character.
        0 [BX] AL MOV           Get the first character which is the key of hashing.
        3 # AX AND              Use only the two LSB bits.
        AX SHL                  Multiply it by 2 to get the cell offset to the proper thread.     CX
AX ADD                  The actual address of the thread.
        1PUSH                   Push the thread on stack and return.
        END-CODE

CODE  (FIND)   ( here lfa --- cfa true, if found; here false, if not found.)   Given   the   address   of   a
                                string and the link field address of a word in dictionary,
                                search the dictionary and return an address and a flag on the
                                stack. Flag=1 for an immediate word; flag=-1 for a regular
                        word; and flag=0 if the word is not found. If not found, the
                string address remains on the stack.
        DX POP                  The link field address.
        DX DX OR                Test it.
        0= IF
                AX AX SUB       Lfa is 0.
                1PUSH           Push a false flag and return.
```

```
THEN                Lfa not 0.  Start comparing strings.
BEGIN
        DX BX MOV
        BX INC   BX INC     BX now points to the name field of the dictionary entry.
DI POP              Here.
        DI PUSH             Get the string address to DI.
```

```
        0 [BX] AL MOV       Get the length byte of the dictionary entry.
        0 [DI] AL XOR       Compare it with the string length.
        63 # AL AND         Mask off two most significant bits, delimiter and precedence
                            bits.
        0= IF               Length bytes not equal, go for the next entry in the thread.
BEGIN
        BX INC              Length bytes equal, now scan the strings.
        DI INC              Next character.
        0 [BX] AL MOV       From the dictionary entry.
        0 [DX] AL XOR       Compare with the one at HERE.
        0<> UNTIL           If equal, continue the comparison.
        127 # AL AND        Not equal.  See if it is the last character in the name field.
0= IF                   Not the last character.  Strings are not the same.  Go for the
                        next entry in the thread.
        DI POP              Rid of the HERE.
        BX INC              Get the code field address.
        BX PUSH             Push it on the data stack.
        DX BX MOV           Get the link field address back to BX again, checking
                            precedence bit.
        BX INC BX INC       Increment to the name field address.
        0 [BX] AL MOV       Get the length byte again.
        64 # AL MOV Examine the precedence bit.
        0<> IF          Not an immediate word.
                1 # AX MOV  Set indicator to 1 for immediate word.
        ELSE
                -1 # AX MOV Not immediate, set AX to -1.
        THEN
        1PUSH               Push the indicator on stack and return.
        THEN
        THEN
        DX BX MOV           String comparison failed.  Prepare to test the next entry in
                            the thread.
        0 [BX] DX MOV       Get the link field address of the next entry in the thread from
                            the link field of this entry.
        DX DX OR            Is the next link field address zero, end of the thread?
0= UNTIL                    Not the end of thread.  Loop back for the next entry. AX  AX  SUB
                End of the thread,
1PUSH                       push a false flag on stack and return.
END-CODE
```

(FIND) searches through one thread, with a given link field address of a dictionary entry. To pick up one thread among four for searching and to do the searching, a high level definition FIND has to be used. The four-way threading of the dictionary is shown in Fig. 8.3.

```
4  CONSTANT  #THREAD        Number of threads implemented in this Forth system.

:  FIND   ( string-addr --- cfa true, if found; string-addr false, if not found)
        DUP C@ IF           If the string is not a null string, do the dictionary searching.
```

```
                Otherwise, do the end of line processing.
PRIOR  OFF      PRIOR is a user variable storing the last vocabulary
                searched. Clear PRIOR to begin searching.
FALSE           This is a dummy flag for the next do-loop.
#VOCS 0 DO      #VOCS=8, the number of vocabularies to be searched.
DROP            Drop the flag on the stack.
```

```
            CONTEXT I 2* + @  Get the vocabulary address in the CONTEXT array.
        DUP IF                If the vocabulary address is zero, skip it because no
                              vocabulary was specified for this CONTEXT entry.
            DUP PRIOR @       Get the contents of PRIOR, the last vocabulary
                              searched.
            OVER PRIOR !      Update PRIOR with the vocabulary to be searched
                              now.
            = IF              If the PRIOR vocabulary is the same as the present
                              vocabulary, here is no need of repeating the
                          searching.
                          DROP FALSE Drop the vocabulary and replacing with a false flag.
                                     Loop  back.
            ELSE              Now search the new vocabulary.
                          OVER SWAP Save a copy of the string address.
                          HASH Hash the string and return the address of the head of
                              a thread  in the present vocabulary.
                          @         Pick up the thread, the link field address of the last
                              entry in  this thread in the dictionary.
                          (FIND)    Search the dictionary.
                          DUP ?LEAVE If tos is a true flag, a word is found in the dictionary.
                                     Leave the loop immediately.  If tos is false, repeat the
                                     loop and  search the next vocabulary.
            THEN
            THEN
        LOOP
    ELSE                  Null string processing.
        DROP              Discard the string address.
        END? ON           Turn on the end-of-line flag.
        ['] NOOP 1        Push the NOOP address on the stack with a true flag so that
                          the nd-of-line process will happen immediately.
    THEN ;
```

FIND thus scans the CONTEXT array, where up to 8 vocabularies can be specified and are to be searched in the order of the array. When a vocabulary is to be searched, HASH selects one of the 4 threads, which are the link field addresses of the last entries in each of the threads stored in the parameter field of the vocabulary, and hands the proper link field address to (FIND) to scan the thread for a name matching the given string. When a vocabulary was searched, its address was preserved in PRIOR to avoid searching the same vocabulary repeatedly. This allows the same vocabulary to be specified in the CONTEXT array more than once without being searched more than once. FIND can also skip nulls in the CONTEXT array. Nulls and multiple vocabulary entries in CONTEXT are conveniences in manipulating vocabulary searching order, which will be discussed in a moment.

```
: DEFINED   ( --- addr flag )    Parse out the next word in the nput stream and search the
                              dictionary. If a matching entry is found, return its cfa and an
                              1 or -1. If not found, return HERE and a false flag.
    BL WORD               Parse the next word, delimited by blank characters, and copy
                          he word to HERE, the word buffer.
    CAPS @ IF             If the contents of CAPS is true, he word will be converted to
```

```
                                upper case characters.
        DUP COUNT UPPER         Upper the cases.
THEN
FIND                            Now do the searching.
;
```

If an immediate word is found by FIND, the return flag is 1. If the found word is a regular, non-immediate, word, -1 is returned. It is important for the colon compiler to know whether a word is immediate or not. The colon compiler normally compiles the code field addresses of regular words, but executes the immediate word to take care of special compiling conditions or to build structures in a colon definition.

In F83, because of the more complicated CONTEXT structure, it requires a few more words to handle the vocabularies and to use then effectively. When a vocabulary is invoked, its parameter field address is stored into the first cell in the CONTEXT array. (See Fig. 8.2.) The next time a search is done, this vocabulary will be the first vocabulary to be searched. The word ONLY is used to initialized the CONTEXT array. It places the address of a very small searching control vocabulary in the first and the last cell of the CONTEXT array. The words in this control vocabulary allow us to select appropriate working vocabularies like FORTH, etc. The word ALSO copies the first CONTEXT entry to the second entry and moves the second and subsequent entry up by one cell, adding one entry to the searching order. This set of words can be used to specify any searching strategy within the size of the CONTEXT array.