CHAPTER 10. WORD PARSING

The source code discussed in this chapter is in the file KERNEL86.BLK, screens 62 to 64.

10.1. TEXT PROCESSING

In communicating with a user through a console, the computer must be able to accept a line of commands and find out what the intention of the user is. The computer then can carry out the commands and do some useful work. In most conventional operating systems, the task which accepts commands from console and interprets the contents of the commands is called a command line interpreter (CLI). The user has to observe a set of rules in entering commands, because the computer uses this set of rules to determine what has to be done, given these commands. These rules are the syntax rules, or more generally, the grammar of the command line interpreter. When the command line interpreter becomes more powerful and has more functionalities built into it, its syntax becomes more complicated and the syntax rules multiply very quickly.

Forth uses a very simple and straightforward syntax rule in interpreting command lines. The command line consists of a sequence of words, separated by blanks (spaces). The words represent either commands pre-compiled in the Forth dictionary or numbers. Thus the Forth command line interpreter (the ext interpreter) can be extremely simple compared to CLI's in other languages or operating systems. The interpreter just has to parse out words using blanks as delimiters, search the dictionary to locate the executable code of the commands, and execute the code. If a word is not a command in the dictionary, the interpreter will try to convert it into a number and push the number on the stack. If the word is neither a command nor a number, it is beyond the capability of the computer to do anything about it, and the interpreter will send an error message to the user protesting his stupidity in a very mild manner.

The tool that provides the interpreter with the ability to parse out words from a command line, or an input stream of characters, is the Forth definition WORD. Before we get into the details of WORD, a few other supporting definitions have to be clarified.

10.2. INPUT STREAM AND INPUT BUFFERS

First, what is an input stream? Where does the interpreter get the command lines? The Forth interpreter can accept commands from two different types of sources: a console terminal or a disk. Two special areas in the computer memory are dedicated to storing commands coming from these sources: a terminal input buffer (TIB) for commands entered through the console, and one or more disk buffers for commands coming from the disk. The terminal input buffer is managed by a number of variables and definitions:

```
VARIABLE 'TIB                    Contains the starting address of the terminal input buffer.

: TIB   ( --- addr )     `       Return the address of the terminal input buffer.
        'TIB @   ;
```

VARIABLE #TIB Maximum number of characters thatcan be held in the terminal input buffer.

```
VARIABLE >IN            Pointer to the character currently being processed.  It is an
                        offset from the starting address of the input buffer, which is
                        either the terminal input buffer or a disk buffer.
```

The disk buffers are managed by the virtual memory management in Forth. The details of this virtual memory system are discussed in a separate chapter. Here we are only concerned with the one disk buffer which is assigned to the interpreter so that the interpreter will get its commands from this buffer. The disk block number is stored in a user variable:

```
VARIABLE BLK            Block number of source on disk to be interpreted.
```

The convention adopted by most Forth systems, including F83, is that if BLK contains a zero, the terminal input buffer is used for interpretation; otherwise, the disk block specified by BLK is used.

10.3. LOW LEVEL PARSING COMMANDS

```
DEFER SOURCE            Vectored to (SOURCE).  Return the starting address and
                        length of the buffer used to hold current input stream.

: (SOURCE)  ( --- addr len )    Return the string to be processed by the text interpreter.
                        Addr is the beginning address of the input buffer and len is
                    the length of the input buffer.
    BLK @               Get the block number from BLK.
    ?DUP IF             If the block number is not zero,
        BLOCK           fetch the block of commands from  disk and return with the
                        address of the disk buffer.
        B/BUF       Length of disk buffer is 1024 bytes.
    ELSE                If the block number is zero,
        TIB             get the address of the terminal input buffer,
        #TIB @          and the length of it.
    THEN ;
```

Here are the hard stuff. Two code definitions that scan the input stream to locate special characters in the stream. SKIP is used to skip over the leading spaces in front of a word, because words can be separated by a number of spaces allowing source commands to be free-formatted. SCAN, on the other hand, will stop at the first match. Separating these two functions into two definitions gives F83 much more versatility in handling strings than older versions of Forth like figForth.

```
LABEL  DONE             A common returning point when the input stream is
                        exhausted.
    CX PUSH             Push the contents of CX on stack and return.  CX register
                        has the remaining length of the stream.
    NEXT

CODE SKIP   ( addr len char --- addr1 len1 )Given the address and length of a string, and
                        a character to look for, scan through the string while we
```

```
                continue to find the character. Leave the address of the
            mismatch and the length of the remaining string.
        AX POP              Move char to AX register.
        CX POP              Move len to CX register.
        DONE JCXZ           If length of string is zero, jump to DONE and return.   DI
POP             Move addr to DI register.
```

```
        DX DX MOV DX ES MOV  Set ES=DS for string manipulations.
        REPZ BYTE SCAS       Repeatedly scan the string until we find a character different
                             from that in AX.
        0<> IF               CX now has the count of characters in the remaining string.
                             If CX is not zero, DI is pointing to the first mismatched
                        character.
                CX INC       Backspace.
                DI DEC       Pointing to the last matching character.
        THEN
        DI PUSH              Addr1.
        CX PUSH              Len1.
        NEXT                 Return.
        END-CODE

CODE SCAN   ( addr len char --- addr1 len1 )    Given the address and length ofa string, run
                             through the string until we find the character. Leave the
                        address of the match and the length of the remaining
                        string.
        AX POP   CX POP
        DONE JCXZ            Same as SKIP.
        DI POP
        DS DX MOV  DX ES MOV
        CX BX MOV            Set up looping parameters.
        REP BYTE SCAS        Repeat if character mismatches. Scan the string.
        0= IF                If the string is exhausted,
                CX INC       Backspace.
                DI DEC
        THEN
        DI PUSH              Restore string registers.
        CX PUSH
        NEXT END-CODE
```

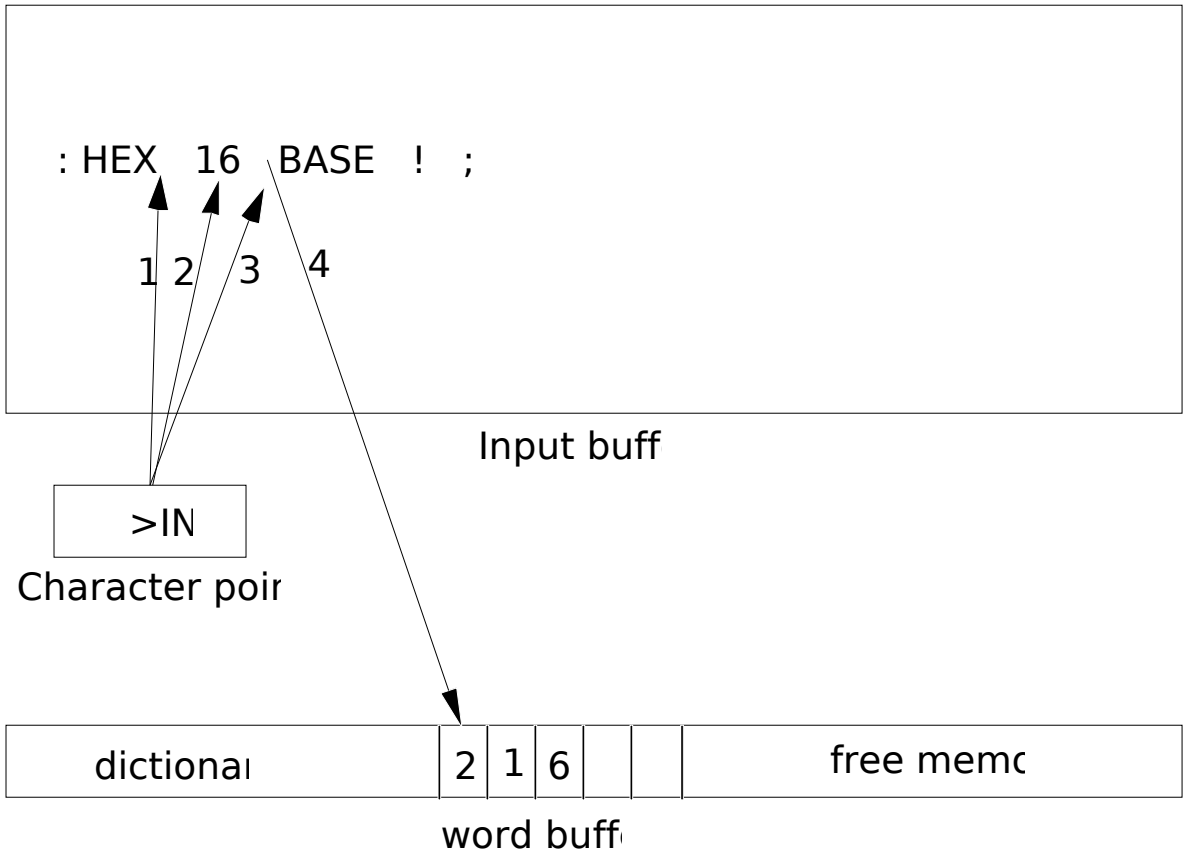
/STRING takes the starting address and length of a buffer, and a pointer to the current character (as returned by >IN), and returns the address of the current character and the remaining length of the input buffer.

```
: /STRING   ( addr len n --- addr1 len1 )    Index into the string by n characters. Return addr+n
                             and len-n.
        OVER MIN             Change n to the smaller of n and len.
        ROT OVER +           Addr+n.
        -ROT -               Len-n.
        ;

: PLACE   ( from-addr len to-addr --- )    Move the characters at from-addr to to-addr.  The
                             final string has a preceding length byte of len.
        2DUP C!              Store the length byte.
        1+                   To-addr+1, address of the first character.
        SWAP MOVE            Copy the string.
```

;

Figure 10.1 Parsing with WORD



1. Start scanning after HEX in the input buffer.
2. Skip leading blank characters.
3. Scan text string to the next blank character.
4. Copy the parsed string into the word buffer.

10.4. HIGH LEVEL PARSING COMMANDS

The real word parsing actions are embodied in the following two words, which scan the input stream and parse out words with specified delimiting character.

```
: PARSE-WORD   ( char --- addr len )      Scan the input stream until char is encountered.  Skip
                                          over leading chars.  Update >IN pointer.  Leave the address
                                          and length of the parsed word.
        >R                                Save char on return stack.
        SOURCE TUCK                       Get the address and length of the input buffer.
        >IN @ /STRING                     Get the current character pointer in >IN and modify addr and
                                          length accordingly.
        R@ SKIP                           Skip over leading chars in the input stream starting at >IN.  OVER
SWAP R> SCAN      Scan for the next occurrence of char.
        >R                                Save length of the remaining string.
        OVER - ROT                        Addr and length of the parsed string.
        R>                                Retrieve the length of string.
        DUP 0<> + - >IN +!                Update >IN to one character after the parsed word.
                                  However, if the parsed string is a null string,
                do not move >IN.
        ;

: PARSE        ( char --- addr len )      Do the same as PARSE-WORD without skipping the leading
                                          char.
        >R
        SOURCE >IN @ /STRING
        OVER SWAP R> SCAN                 SCAN instead of SKIP.
        >R                                Len.
        OVER - DUP                        Addr and length of parsed string.
        R> 0<> - >IN +!                   Update >IN to end of string.
        ;

: 'WORD        ( --- addr )               Leave on stack the address of the word buffer, which is on
                                  top of the dictionary.
        HERE ;                            In F83 'WORD is the same as HERE. They might differ as
                                  indicated in 83-Standard.
```

Finally, we get to the most important word WORD, which parses the next word in the input buffer and copies the word to the word buffer for the text interpreter to do searching or number conversion. WORD will skip over leading delimiters so that words in the input stream can be spaced out to conform to various formatting conventions.

```
: WORD         ( char --- addr )          Parse the input stream for char and return a count delimited
                                          string in the word buffer at HERE.  Note that there is always
                                          a blank following the word in the word buffer.       PARSE-
WORD              Get the address and the length of the next word in the input
                stream.
        'WORD PLACE                       Move the word into the word buffer, with a length byte as
```

```
                                the first character.
'WORD DUP COUNT +       The address following the string.
BL SWAP C!              Append a blank at the end of string.
; .
```

10.5. STRING COMMANDS DEFINED USING 'PARSE'

A couple of examples are handy here to illustrate the usefulness of these parsing commands:

```
: (             ( --- )         The Forth comment command.  The input stream is skipped
                                until a ) is encountered.  The enclosed comments are thus
                                ignored by the text interpreter.
        ASCII )                 Use ) as the delimiter.
        PARSE                   Move >IN to the character after ).
        2DROP                   Nothing will be done with the comments.  Discard its
                                address and length.
        ; IMMEDIATE             Declare ( to be immediate so that it will be executed inside a
                                olon definition.

: .(            ( --- )         Type the following string on the console during
                                interpretation or compilation.
        ASCII )                 Use ) as delimiter.
        PARSE                   Parse out the next string upto but not including the )
                                character.
        >TYPE                   With addr and len on stack, type out the string.
        ;

: >TYPE         ( addr len --- )        Same as TYPE.  The string is copied to the PAD buffer
                                before outputting for multi-tasking environment.
        TUCK PAD SWAP CMOVE     Copy the string to PAD buffer.
        PAD SWAP TYPE           Type from the PAD buffer which is private to a task.;
```

10.6. END OF BUFFER CONDITION

The blank character appended to the end of the parsed word in the word buffer is very important to the F83 system. It serves many important functions. One of them is for the number conversion routine to recognize the correct end of a number string. Another function is to help the text interpreter to detect the end of an input stream so that the text interpreter can prepare itself to process the next input stream or command line. For those familiar with the figForth system, there the end of an input stream is artificially terminated by one or more Ascii NUL characters. During console inputting, when a carriage return is received from the keyboard, the input routine appends a NUL at the end of the input stream. When using source texts in disk blocks, each disk buffer has two trailing NULs as the tail of the buffer. These artificial NULs force the interpreter loop to be terminated in a non-obvious and hard to document fashion. F83 tries to treat the end of line condition explicitly.

When WORD reaches the end of the input stream, the length of the parsed word will be zero. A null string without characters is then moved into the word buffer. The count byte is zero with a blank character appended to it. This null string, two bytes (one cell) long, has a hex value of 2000. In the dictionary, there is a word of this name, whose hex value in the name field is A080. Masking off the MSB in these two bytes (the delimiters of the name field), the real name has a hex value of 2000, exactly the same as the parsed null word. The function of this null word is to turn on the end-of-buffer flag. Seeing that this flag is set, the text interpreter knows it has reached the end of the input stream. It

will terminate the loop, and ready itself for the next line of input.