CHAPTER 9. NUMBER INPUT AND OUTPUT

The source code discussed in this chapter is in KERNEL86.BLK file, screens 58 to 61.

The Forth interpreter can only recognize two types of words: commands (Forth definitions compiled into the dictionary) and numbers. A large portion of the Forth system is devoted to processing numbers, including inputting numbers from console or disk, doing arithmetic and logic operations on them, and outputting them to console or other devices in a required format. In the nucleus layer, we've seen lots of arithmetic and logic operators. In this chapter, we will discuss how numbers are transformed from the external representation in ASCII strings to the internal representation in the binary form, and vice versa.

9.1. REPRESENTATION OF NUMERIC DATA

A very interesting aspect of Forth in its external representation of numbers is that numbers can be presented in many different bases. Not only decimal, octal, hexadecimal, and binary, but also in any reasonable base from 2 to 70, limited by the number of ASCII characters available to represent digits. The reason is that in Forth the primitive number input and output words are directly accessible to the user, giving him tools that he can use at will to define and modify rules in doing number input and output.

Internally, all numbers are represented in 16 bit binary form and processed in 16 bit units. In the case that more bits are required to represent large integer numbers, two 16 bit numbers are used together as a 32 bit double precision integer. For data requiring less than 16 bits, they are generally right justified in the 16 bit field and the high order unused bits are cleared to zeros.

F83 uses many different data types. Their ranges are shown in the following table:

TABLE 9.1. DATA REPRESENTATION

Data Type	Range
True flag	-1 (32767) and any non-zero number
False flag	0
Ascii codes	0..127
Byte	0..255
Integer	-32768..32767
Unsigned integer	0..65535
Address	0..65535
Double integer	-2,147,483,648..2,147,483,647
Unsigned double integer	0..4,294,967,295

Forth is not a typed language. We can talk about data types and their external representations, but once they are inside the Forth computer, they are all represented in the uniform 16 bit format. Forth doesn't care what type a number was when it was input into Forth. Thus you can do arithmetic on the flags and ASCII codes like any other numbers. You have to know what you are doing. You must use the right operator to process the data you entered. This is the price you have to pay for the convenience in using the data stack.

F83 maintains three user variables specifically for the purposes of number input/output:

VARIABLE BASE	The current base for number input and number output conversions. A decimal 10 stored in BASE causes input number strings to be treated as decimal numbers. A decimal 16 in BASE makes the conversions done in hexadecimal.
VARIABLE DPL	The decimal point location. It stores the location of the decimal point in an ASCII number string, from the right end of the string. In other words, the number of digits after the decimal point. If no decimal point, DPL=-1.
VARIABLE HLD	The number of digits stored in the number output buffer for output.

9.2. INPUT NUMBER CONVERSION

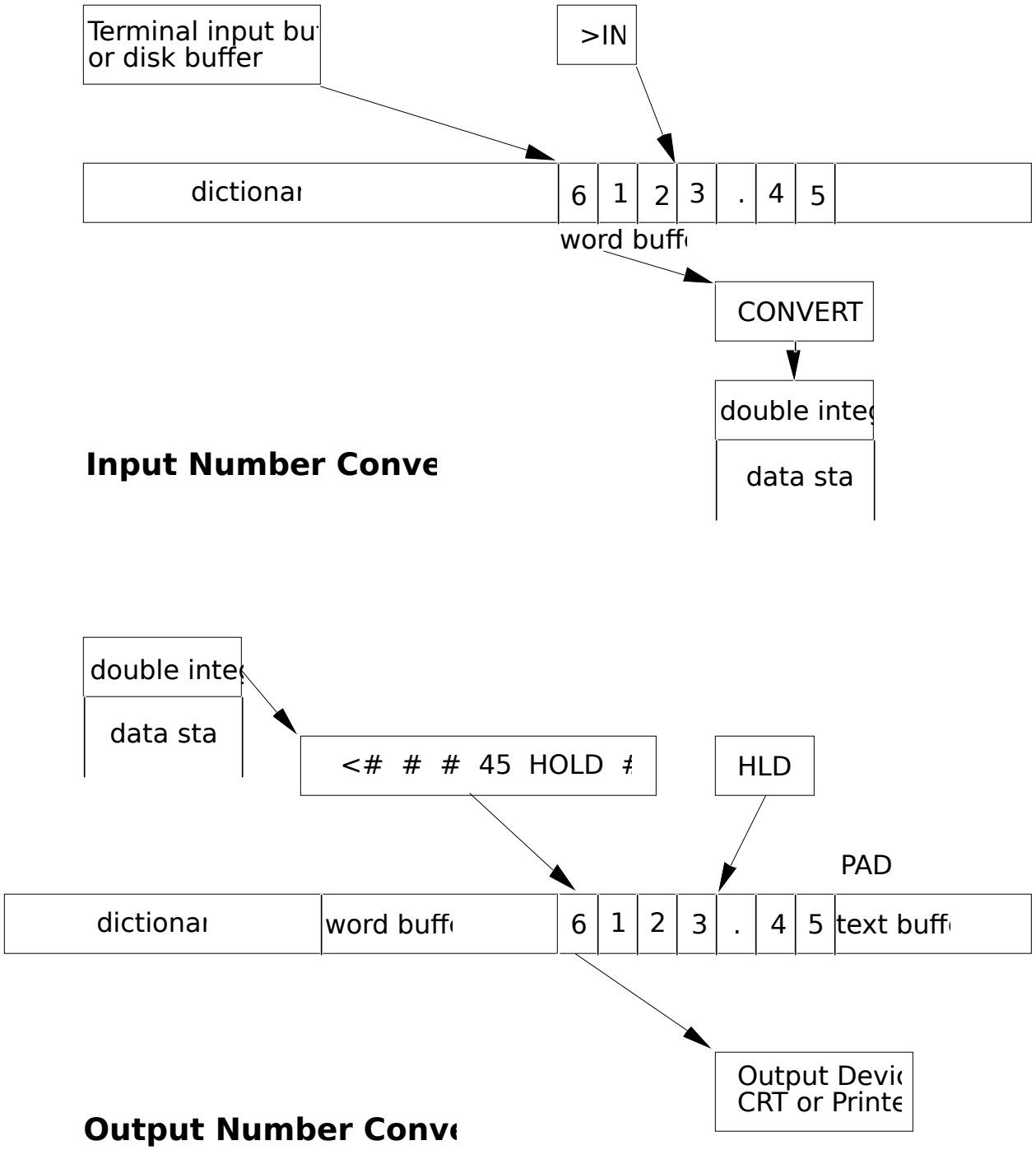
The text interpreter parses a word out of the input stream and places the parsed word in the word buffer, just above the last entry in the dictionary. It first searches the dictionary to see if the word is a pre-defined Forth command or definition. If it fails to match the parsed word to a definition, the parsed word is left in the word buffer for the number conversion routine to convert it to a number. The following set of F83 words supports the number conversion process.

```
LABEL  FAIL                 A common return routine used when failed to convert the
                            string to a  number due to a number of reasons.
       AX AX SUB            Push a false flag on the stack
       1PUSH                and return to the NEXT routine.

CODE  DIGIT   ( char base --- n f )  Return a flag indicating whether or not the character is a
                                    valid digit in the current base.  If so, return the converted
                                    value with a true flag.  Otherwise, return the character with a
                                    false flag.
       DX POP               Pop base into DX.
       AX POP               Pop character into AX.
       AX PUSH              Push character back to stack just in case of a conversion
                            failure.
       ASCII 0 # AL SUB     Subtract ASCII 0 (48) from the code of the given character.
       FAIL JB              If char is below 0, it is not a valid digit.  Jump to FAIL.
       9 # AL CMP           Is char > 9?
       > IF                 No, a regular digit.  Skip to DIGI1.
            17 # AL CMP     Is char between 9 and A?
            FAIL JB         Yes.  Invalid digit.  Jump to FAIL.
```

```
        7 # AL SUB      Eliminate the gap between 9 and A.  A must be the next digit
                        following 9.
THEN
DL AL CMP               AL has the converted value.  Is it in the range of BASE?
FAIL JAE                If the value is equal or above the base value, it is not a valid
                        digit.  Jump to FAIL.
AL DL MOV               Copy value to DL for DPUSH.
```

Figure 9.1 Input and output number conversions



```
        AX POP                  Discard char from the stack.
        TRUE # AX MOV           Put true flag in AX.
        2PUSH                   Push value and flag on stack and return.
        END-CODE
```

The sequence of digits is from 0 to 9, and from A up if the base value is greater than 10. Theoretically the sequence can go up to tilde (~ ASCII 126). Then anything you type could be converted to a number.

```
: DOUBLE?   ( --- f )           Return a true flag if a period is encountered in the number string.
        DPL @                   Get the contents of DPL. If no period is in the number string,
                                DPL is -1 as initialized.
        1+                      0 if no period.
        0<>                     Logic NOT.
        ;

: CONVERT   ( ud1 addr1 --- ud2 addr2 )     Starting with the unsigned double integer ud1 on
                                stack and the number string at addr1, convert the string to a
                                number and add to ud1 according to the current base. Leave
                                the resulting double integer and the address of the
                                unconvertible digit addr2 on stack.
        BEGIN                   This is an indefinite loop.
            1+                  Get the next character in the string.
            DUP >R C@           Get a digit. Save a copy of its address on the return stack.
            BASE @ DIGIT        Convert one digit.
        WHILE                   Exit the loop if the digit is invalid.
            SWAP BASE @ UM* Left shift the upper half of the double integer by one digit.
            DROP                Keep only the lower half of the product.
            ROT BASE @ UM*      Left shift the lower half of the double integer by UM*.
                                Result is a double integer sitting on top of the value of
                                the converted digit and the left-shifted upper half of ud1.
            D+                  This is tricky, but the result s ud1*base+value.
            DOUBLE?             Have we seen a period?
            IF 1 DPL +! THEN    Yes, we get one more digit after the period. Increment DPL.
        R>                      Recall the character address.
        REPEAT
        DROP                    Discard the invalid digit left by DIGIT.
        R>                      Address of the invalid digit.
        ;

: (NUMBER?)   ( addr --- d flag )   Given a string at addr with at least one digit, convert it to
                                a double integer.
        0 0                     The initial value of the double integer serving as accumulator.
        ROT                     Get addr to top of stack.
        DUP 1+ C@               Get the first digit.
        ASCII - =               Compare it to Ascii - sign.
        DUP >R                  Save the negative flag on return stack.
        -                       If the first digit is a - sign, proceed to the next character.
                                Otherwise, start conversion at the current address.
```

```
    -1 DPL !                    Initialize DPL.
    BEGIN
        CONVERT                 Convert the number string.
        DUP C@                  Get the invalid digit.
        ASCII , ASCII /
        BETWEEN                 Is the invalid digit a punctuation between , and /?  Any of
                                them is a valid punctuation mark, equivalent to a period.
    WHILE 0 DPL !               A punctuation mark is encountered. Reset DPL.
    REPEAT                      Ignore the punctuation mark and continue converting the rest
                                of the number string.
    -ROT                        Rotate the invalid character address below the double integer.
    R>                          Get the negative flag.
    IF DNEGATE THEN             If the number is preceded by a - sign, negate the double integer.
    ROT C@ BL =                 Compare the last invalid digit with blank and leave the result
                                on stack as a flag.
    ;
```

F83 accepts numbers with an optional preceding - sign for negative numbers. Within the number string, four punctuation marks, ',', '-', '.', and '/' are allowed. When any of these punctuation marks appears in the string, DPL is reset to zero so that CONVERT can keep track of the number of digits following the punctuation mark, and the converting process continues on until an invalid digit other than these punctuation marks is encountered.

```
: NUMBER?   ( addr --- d f )    Convert the number string at addr to a double integer.  The
                                number string may be preceded by a - sign, but must be
                                terminated by a blank.  The location of the last punctuation
                                mark is saved in DPL.  A true flag is left on the stack if successful.
    FALSE                       Put up a default flag on stack.
    OVER COUNT BOUNDS           Set up loop limits to scan the supposed number string.
    ?DO                         Scan the string for valid digit.
        I C@ BASE @ DIGIT           Is this a valid digit?
        NIP                     I don't care its value now.
        IF DROP TRUE LEAVE      Leave the loop with a true flag if a valid digit is found in
                                the string.
        THEN
    LOOP                        The purpose of this test is to filter out a mis-typed word in
                                which   case it is just a waste of time to do the number conversion.
    IF (NUMBER?)                Do the conversion if the string is potentially a number.
    ELSE                        No valid digit in the string.
        DROP                    Discard its address.
        0 0                     Put a null double integer on stack.
        FALSE                   Top it with a false flag.
    THEN ;

: (NUMBER)          ( addr --- d )  Convert a counted number string to a double integer.  The
                                string may have optional leading - sign and embedded
                                punctuation. It must be terminated by a blank.
    NUMBER?                     Conversion.
```

```
        NOT                     If not a number or not terminated by a blank,?MISSING
                                print an error message and abort.
        ;

DEFER NUMBER                    Vectored to (NUMBER).
```

With this set of input conversion tool, we can type in numbers like:

```
        415-424-3001  12/25/1983  123.45  -0.4567  987,654,321   534,234.00
```

If we are in hexadecimal base the following numbers are also valid:

```
        A1  F9  BAD-FAD  FEED/BEAD  -1B2A5D.0
```

However, after conversion, they are all internally represented by double integers. The embedded punctuation marks have no effect on the conversion except the contents of DPL.

9.3. OUTPUT NUMBER CONVERSION

The primitive Forth output conversion routine converts a double integer to an Ascii string suitable for outputting to a console or to a printer. The user can explicitly format the string and insert special characters into the string to design formats he desires. Let's look at these small tool words first and then see how they are strung together to build number output words often used in routine Forth programming.

```
: HOLD      ( char --- )        Insert the character char into the output string.
            -1 HLD +!           HLD contains a character pointer to the output text buffer
                                where he number output string is being constructed. The
                                number character string is built backwards from the least
                                significant digit to the most significant digit. To insert a
                                character into this string HLD has to be decremented.
        HLD @                   Get the character pointer.
        C!                      Insert char to where HLD points.
        ;

: <#        ( --- )             Initialize the number conversion process.
        PAD                     PAD returns the location of the text buffer used for output.
        HLD !                   Point HLD to PAD so that the number string can be built in
                                the PAD buffer.
        ;

: #>        ( d --- addr len )  Terminate the output number conversion and leave the
                                address and length of the number string on stack suitable for
                                TYPE to print out.
        2DROP                   The double integer on stack is no longer needed.
        HLD @                   The address of the number string.
        PAD                     The end of the string.
        OVER -                  The length of string.
```

```
        ;

: SIGN          ( n --- )       If n is negative insert a minus sign into the number string.
        0< IF                           If n is negative,
                ASCII - HOLD            Insert the minus sign.
        THEN ;

: #             ( d1 --- d2 )           Convert one digit and add the digit to the number string.
                                        The conversion is done by dividing d1 by base.  The
                                        quotient d2 is left on the stack and the remainder is converted
                                        to ASCII code and added to the output buffer.
        BASE @ MU/MOD                   Divide d1 by the base.  The remainder and the double integer
                                        quotient are left on stack.
        ROT                             Get the remainder to the top.
        9 OVER <                        If the remainder is greater than 9,
        IF 7 + THEN                     add 7 to make A.
        ASCII 0 + HOLD                  Convert to ASCII code and HOLD it in the output buffer
.       ;

: #S            ( d --- 0 0 )           Convert a double integer until finished.
        BEGIN
                #                       Convert one digit.
                2DUP OR                 Is the quotient 0?
        0= UNTIL                        If it is zero, exit the loop. Otherwise, continue converting.
        ;
```

With these tools, we can format numbers for output in any format we want. However, it is always nice to look at how the F83 designers build some of the standard number output commands.

```
: (U.)  ( u --- addr len )      Convert an unsigned single integer to a number string.
        0                       Make the unsigned integer into a double integer.
        <#                      Initialize the conversion.
        #S                      Convert all digits.
        #>                      Prepare for output.
        ;

: U.            ( u --- )       Output an unsigned single integer with one trailing space.
        (U.)                    Convert.
        TYPE SPACE              Print.
        ;

: U.R   ( u len --- )           Output an unsigned integer in a field of len columns.
        >R                      Save the column width.
        (U.)                    Convert.
        R>                      Recall column width.
        OVER - SPACES           Output appropriate number of spaces so that the number
                                string will come out right justified.
```

```
        TYPE                    Output the string.
        ;

: (.)           ( n --- addr len )      Convert a signed single integer to a number string.
        DUP ABS                 Get the absolute value of n.
        0                       Make it a double integer.
        <# #S                   Convert all digits.
        ROT SIGN                Add a minus sign if n is negative.
        #>                      Finish the output string.
        ;

: .             ( n --- )               Output a signed integer with a trailing space.
        (.)                     Convert.
        TYPE SPACE              Type.
        ;

: .R            ( n col-len --- )       Output a signed integer right justified in len columns.
        >R (.)                  Convert n first.
        R> OVER - SPACES        Pad with leading blanks.
        TYPE                    Now print the number right justified.
        ;
```

9.4. DOUBLE INTEGER OUTPUT

```
 : (UD.)        ( ud --- addr len )     Convert an unsigned double integer to a number string.
        <# #S #>   ;

: UD.   ( ud --- )                      Output an unsigned double integer with a trailing space.
        (UD.)   TYPE SPACE   ;

: UD.R          ( ud len --- )          Output an unsigned double integer right justified in len columns.
        >R (UD.) R>   OVER - SPACES   TYPE   ;

: (D.)  ( d --- addr len )              Convert a signed double integer to a number string.
        TUCK                    Save a copy of the upper half  of the double integer under
                                he double integer.  We will need its sign.
        DABS                    Convert the double integer to its absolute value.
        <# #S                   Convert all digits.
        ROT                     Get the saved upper half of the original double integer.
        SIGN                    Put up its sign.
        #>                      All done.
        ;

: D.            ( d --- )       Output a signed double integer with a trailing space.
        (D.)   TYPE SPACE   ;
```

```
: D.R   ( d len --- )              Output a signed double number right justified in len columns.
        >R (D.) R>   OVER - SPACES   TYPE   ;
```