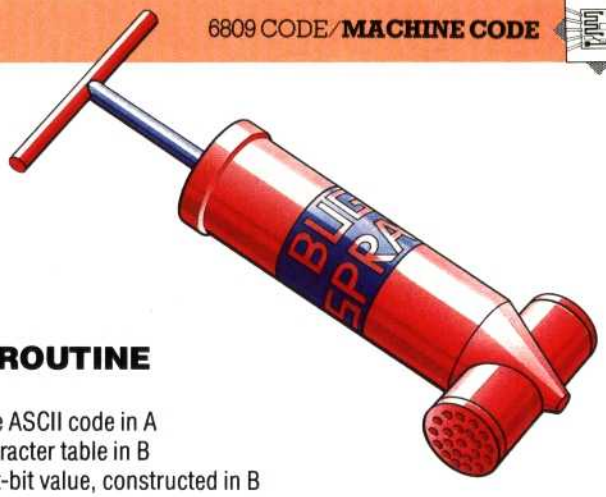
PEST CONTROL



We continue to develop the debugger program, which we began on page 758. First, we must complete the set of routines that we need for the module that handles input and output, and then construct the module itself.

There are four more routines to develop for the I/O module: GETHX2, GETHX4, PUTHEX and PUTCR. The first two processes are used to get hexadecimal digits from the keyboard: GETHX2 gets a two-digit hex number and GETHX4 gets a four-digit number. The first thing we must do, when designing these routines, is decide whether we will insist on two or four digits always being entered (which is easier to program but less user-friendly), or allow fewer characters followed by a Return. A further problem is whether to allow the use of a backspace character to delete characters already entered.

We will use the simplest method for the GETHX4 routine: four digits must be entered, and the backspace will not be allowed. The 16-bit value (signifying an address) can be returned in the D register.

GETHX2 is more of a problem if we consider the circumstances in which it will be used. Eight-bit quantities will have to be entered for the function to inspect and change memory (command M), which involves accessing an address. The contents of this address are displayed, and the user may then enter a Return (to move on to the next location in sequence) or a two-digit hex number (which will be stored at that location) or some other character (a dot, for example, to exit to the command level). We can add the two extra valid characters to the end of the string of valid hex digits. GETHX2 must then accept either two hex digits or a Return or a dot. The eight-bit value can be returned in B and we must use A to indicate which of these situations has arisen. A will have the value 0 if a two-digit number was entered, 1 if a Return was entered, or –1 if a dot was entered. These values enable us to test the value in A without having to compare it with another value.

Let's assume for the moment that the following declarations have been made for this module:

```
HEXCHS    FCC'0123456789ABCDEF'
DOT       FCB '.
RETURN    FCB 13 (ASCII code for Return)
```

We can pass 16 as the length of the string for GETHX4, where we only need the hex digits, and 18 as the length for GETHX2, where we need the other two characters as well.

THE GETHX2 ROUTINE

Data:

Next-Character is the ASCII code in A
Offset into Valid-Character table in B
Hex-Value is an eight-bit value, constructed in B
Flag is either 0, 1 or –1 in A

Process:

```
Get Next-Character
IF Character is a dot (Offset = 16) then
   Set Flag to –1
ELSE if character is a Return (Offset = 17) then
   Set Flag to 1
ELSE
   Save Offset temporarily
   Get Next-Character (hex digits only valid at
   this point)
   Construct Hex-Value
ENDIF
```

The final coded form of GETHX2 is given on page 779. The coding of the GETHX4 routine is now made slightly easier by using parts of this routine. By making HX4 an alternative entry point to the GETHX2 routine, we can call that routine and ensure that only valid hex digits are accepted — provided we load B with 16 before the call. Thus, the process for getting four hex digits is made considerably less complex.

THE GETHX4 ROUTINE

Data:

Hex-Number is a 16-bit value to be returned in D
Most-Significant-Byte and
Least-Significant-Byte are both eight-bit values to be returned in B

Process:

```
Get Most-Significant-Byte
Save Most-Significant-Byte temporarily
Get Least-Significant-Byte
Construct Hex-Number
```

The routine is given, in its final form, after the GETHX2 code.

The routines for displaying characters are less complicated to design. For the PUTHEX routine, we will assume that the eight-bit number we require is to be found in B.

THE PUTHEX ROUTINE

Data:

Number is the eight-bit value found in B
Offset is the four-bit offset put into HEXCHS

Process:

```
Extract most significant four bits of Number as
 Offset
```