range (one to Number-Of-Breakpoints — 1) which we use as an offset into the two tables. Note, however, that one table is of 16-bit values whereas the other is of eight-bit values. We will assume that N is passed in A. The address of the breakpoint obtained from that table will be put into X. The removed op-code will be put into B for transfer to the Removed-Values table. B can then be used to put the SWI op-code into the appropriate address.

We give the final coded form of Process2 (Set-Up-Breakpoint Module) here; our next task is to develop a module to handle input and output. As you will have seen from the design of the debugger so far, there are a number of I/O tasks to be performed by the program. For the moment, we will assume the existence of two subroutines: INCH, which will input a single character into the A register from the keyboard; and OUTCH, which will send a character from A to the screen at the current cursor position. The routines required by this module are:

1. GetCommand: Input the next command from the keyboard.
2. GetAddress: Get a hex address (one to four characters long) from the keyboard.
3. GetValue: Get a hex value (one or two characters long) to modify the value of a memory location.
4. DisplayValue: Display a two-character hex value on the screen.
5. DisplayAddress: Display a four-character hex address on the screen.

Our approach illustrates the difference between the top-down and the bottom-up methods of programming. The top-down approach might lead us to define and code these operations independently, thus ending up with a number of separate routines that do essentially the same thing. The bottom-up approach can produce a saving in time, effort and space by simply writing a few useful routines that are used in a number of different circumstances. These routines are:

GETCH: To input a single character into A, checking against a list of valid characters (command letters or hex digits), echoing valid characters and ignoring others.
GETHX2: To use GETCH to get two hex digits and convert them into an eight-bit number.
GETHX4: To get four hex digits to form a 16-bit number.
PUTHEX: To display an eight-bit number as two hex digits. (This can be called twice to display a 16-bit number.)
PUTCR: To output a carriage return (or carriage return and line feed if necessary).

These five routines need to be developed in turn. First, we will consider the design of GETCH.

GET CHARACTER ROUTINE

Data:
Inchar is an ASCII character input from the keyboard (held in A)
Valid-Chars holds the 16-bit address of the table of valid characters
Number-Of-Valid-Chars is an eight-bit value
Chars-Searched is an eight-bit counter

Process:
REPEAT
  Get next Inchar
  Set Chars-Searched to (Number-Of-Valid-Chars – 1)
  While Valid-Chars(Chars-Searched) <> Inchar
    AND Chars-Searched >=0
    Decrement Chars-Searched
Until Chars-Searched >=0
DISPLAY Inchar

In order to code this, we must use A to store Inchar, and the 16-bit Valid-Chars value can be passed and kept in X. The Number-Of-Valid-Chars can be passed in B, but will need to be kept more permanently, by pushing it onto the stack. B can then be used for Chars-Searched. Note that B will return the offset into the table, which will be useful in command interpretation and hex conversion.

We give the final coded form of this routine here. In the next instalment of the course, we will develop the other routines required by the input/output module.

GETCH Routine

```
GETCH   PSHS    B           Save B
REPTOO  BSR     INCH        Get Next Inchar
        LDB     1,S         Set Chars-Searched
        DECB                Subtract one from max offset
WHILOO  BLT     ENDWOO      While Chars-Searched >=0
        CMPA    B,X         AND
        BEQ     ENDWOO      Inchar <> Valid-Chars(Chars-
                            Searched)
        DECB                Decrement Chars-Searched
        BRA     WHILOO
ENDWOO  TSTB
UNTLOO  BLT     REPTOO      Until Chars-Searched >=0
        BSR     OUTCH       Display Inchar
        LEAS    1,S         Increment S to 'forget' the
        RTS                 original value of B
```

Set-Up-Breakpoint Module

```
                            Data declarations
BPTAB   RMB     32          Breakpoint-Table
REMTAB  RMB     16          Removed-Values
NUMBP   FCB     0           Number-Of-Breakpoints
NEXTBP  FCB     0           Next-Breakpoint
SWIOP   FCB     $3F         SWI-Opcode
MAXBP   FCB     16          Maximum number of Breakpoints
                            Process2 — Set-Up-Breakpoint
BP02    PSHS    B,X         Save the registers we will alter
        LSLA                Multiply the offset by two
        LEAX    BPTAB,PCR   Base address of table
        LDX     A,X         Get Address in Breakpoint-Table(N)
        LDB     ,X          Get Op-code at that address
        LSRA                Restore A to original value
        STB     A,X         Store Op-code in Removed-Values(N)
        LDB     SWIOP,PCR   Get SWI-Opcode
        STB     ,X          Store it at the address
        PULS    B,X,PC      Restore and return
                            End of Process2
```