BREAK EVEN POINT

We continue to develop our debugging program. First, we will complete the module of routines to handle breakpoints, which we started coding in the last instalment (see page 777). Then we look at the procedures necessary to handle each of the commands.

We have yet to define two subroutines for the Breakpoint module — one to remove inserted breakpoints and the other to restore the original op-code where we have placed a temporary SWI-opcode. The first routine we need to consider is called Uninsert-Breakpoint (from the Breakpoint-Table).

Up to 16 breakpoints have been allowed for in the Breakpoint-Table (BPTAB). To remove one we must be supplied with its number as an offset (in the range 0 to 15) into this table. The table entry is removed by shifting all subsequent entries in the table back one place (two bytes) and decrementing the Number-Of-Breakpoints.

UNINSERT-BREAKPOINT

```
Data:
    Number-Of-Breakpoints is an eight-bit value
    Breakpoint-Number is an eight-bit counter
    Breakpoint-Table is a table of 16-bit addresses
    Entry-to-be-Removed is an eight-bit offset (with a
    value in the range 1 to 16)
Process: Uninsert-Breakpoint
    Decrement Number-Of-Breakpoints
    If Entry-to-be-Removed <= Number-Of-
    Breakpoints (one before last) THEN
        For Breakpoint-Number = Entry-to-be-
        Removed to Number-Of-Breakpoints (one
        before last)
        Move Breakpoint-Table(Breakpoint-Number +
        1) to Breakpoint-Table(Breakpoint-Number)
        Move Removed-Values(Breakpoint- Number + 1)
        to Removed- Values(Breakpoint-Number)
        EndFor
    Endif
End of Process
```

The parameter Entry-to-be-Removed can be passed in B. The counter Breakpoint-Number can then also be placed in B, and will get automatically set to its correct initial value. After comparing it with Number-Of-Breakpoints, it must be decremented to form the offset into the eight-bit Removed-Values table and then shifted (multiplied by two) to form an offset into the 16-bit Breakpoint-Table. We can keep the eight-bit offset in B and the 16-bit offset in A. The addresses of the entries in the two tables can be in X and Y, so we can use auto-increment to step through the table. The 16-bit entry can be shifted through U, but the eight-bit entry will have to use A again.

The last process used in this module physically removes a breakpoint by replacing the SWI-opcode with the original op-code from the table of Removed-Values.

UNSET-BREAKPOINT

```
Data
    Breakpoint-Number is the eight-bit offset
    into Breakpoint-Table
Process:
    Get value in Removed-Values(Breakpoint-
    Number)
    Store it in address in Breakpoint-Table
    (Breakpoint-Number)
```

We will assume that the parameter Breakpoint-Number is passed in B in the usual form as a number in the range from one to 16, which must be converted to function as an offset into the tables.

We are now at the stage where we can start constructing a module to execute the eight single-letter commands that operate the system (see page 758). A number of these commands can be directly executed by the routines that we have already written. However, for the sake of completeness and a proper modular structure we shall incorporate calls to them from this module.

The command B, to insert a breakpoint, is covered completely by the routine Insert-Breakpoint (BP01). In this module, therefore, we simply need:

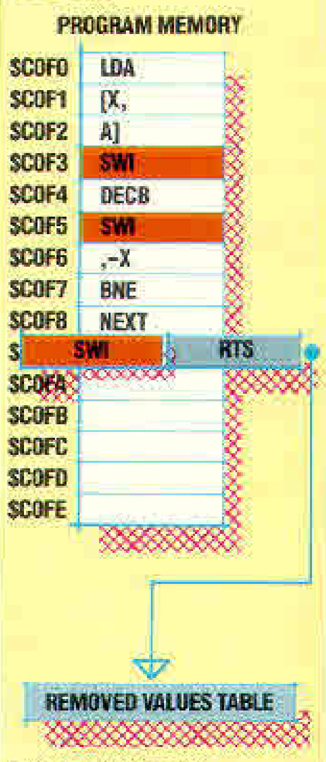
```
CMDB  BRA BP01
```

Command U, to Un-insert a breakpoint, is almost covered by the routine that we have just written (BP04). However, we must first get the address of the breakpoint to be removed and search the Breakpoint-Table to find that address. If it is not there, then we ignore the command; if it is there, then we can pass the offset to the subroutine at BP02.

COMMAND U

```
Data:
    Prompt is to be displayed
    Breakpoint-Address is the input
    Breakpoint-Table
    Breakpoint-Number
Process:
    Display prompt
    Get Breakpoint-Address
    Set Breakpoint-Number to 16
    While Breakpoint-Table (Breakpoint-Number)
    <> Breakpoint-Address
```

Breaking The Code



The debugger inserts breakpoints in the object code under test by first saving the code from that address into the Removed Values Table and incrementing the Number Of Breakpoints Counter, and then by overwriting the contents of the breakpoint byte with the SWI op-code. The Removed Values Table looks like a stack but is, in fact a heap, so values can be taken from any byte within it, not just from the last byte entered. When a breakpoint is removed, the appropriate Removed Value is copied from the table back into Program Memory, and the redundant byte is eliminated by moving down all the table bytes above it in memory, and, finally, decrementing the Number Of Breakpoints counter.