save the value of the stack pointer as the first operation after the interrupt occurs, so that it can be used as a reference.

In coding the R command, we will assume that this has been done, so that we can retrieve these register contents. The structure of the routine is perfectly straightforward — we simply take each value in turn without actually pulling them off the stack and display them with appropriate labels. The only exception will be the value of S — this should be the value prior to the interrupt and can be obtained by adding the appropriate amount to the saved value of S that we use to reference the stacked register values.

COMMAND R

Data:

Stack-Pointer is the value of the top of stack after interrupt in X
Single-Byte-Value holds the values of single-byte registers in B
Two-Byte-Value holds the values of 16-bit registers in D
Labels holds the labels for the nine registers

```
Get Stack-Pointer
Load CC into Single-Byte-Value
Display label(1), Single-Byte-Value
Repeat the above for A, B, and DP
Load X into Two-Byte-Value
Display label(5), Two-Byte-Value
Repeat the above for Y, U and PC
Add 12 to original value of Stack-Pointer
Display label(9), Stack-Pointer
```

There are two remaining commands: Q, to quit the program, does not need a special routine of its own; and G, to resume program execution after a breakpoint. At this point we have to replace the SWI instruction that caused the break with the original instruction that it replaced and then pass control back to that instruction. We can restore the registers to their original contents easily enough, simply by using an RTI, which unstacks them all. We must, however, be careful that the value of the PC that is unstacked is going to be the value for the next instruction; since this is one greater than the value we require, we must adjust the value on the stack before we return.

COMMAND G

Data:

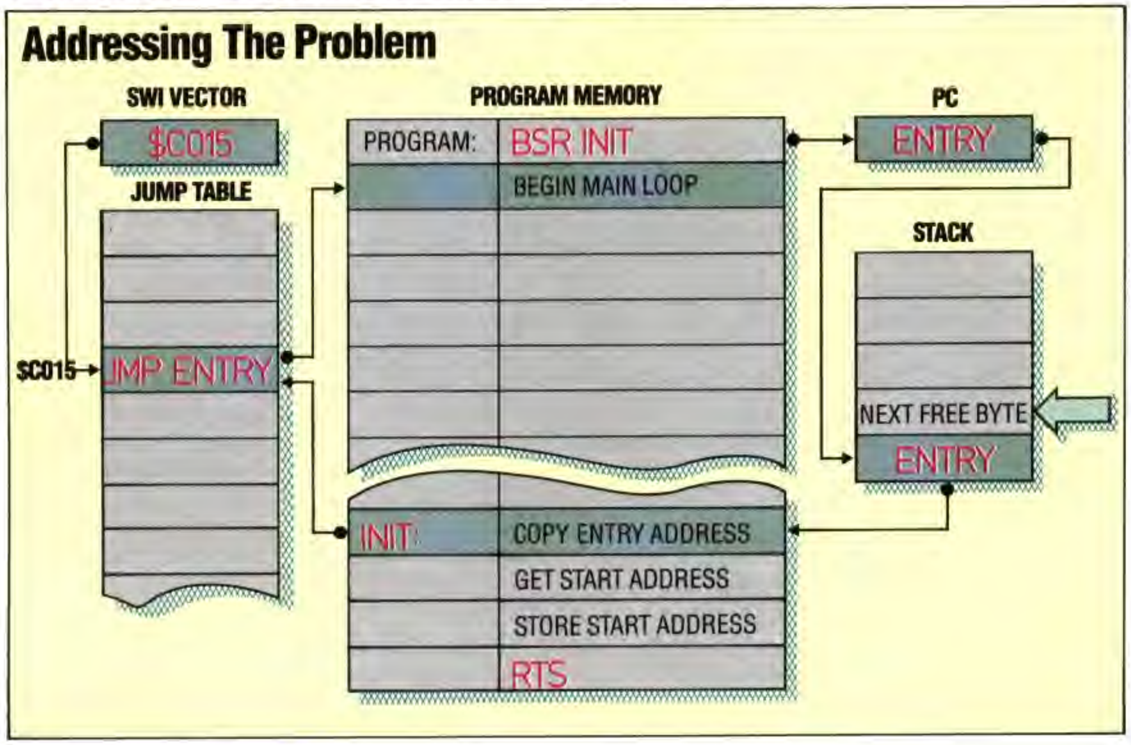
Breakpoint-Table is a table of 16-bit addresses of breakpoints
Removed-Values is a table of op-codes replaced with SWIs
Next-Breakpoint is a number in the range 1 to 16
Stack-Pointer is the saved value of the stack pointer after the SWI

Process:

```
If Next-Breakpoint >0 and <=16 then
    Get op-code from Removed-Values (Next-Breakpoint)
    Store it at address in Breakpoint-Table (Next-Breakpoint)
    Set S to Stack-Pointer
    Decrement value of PC on stack
    Increment Next-Breakpoint
    Return from interrupt
else
    Return from subroutine
```

Our 6809 machine code series concludes in the next instalment, when we code the main module of our debugger, and look at the operation of the program as a whole.

A Stack In Time
The debugger program begins with a BSR call to the initialisation routine, followed by the start of the main program loop. One of the initialisation tasks is to ascertain the absolute address of this loop start, and to copy it into the interrupt jump table so that when an SWI is executed control will pass through the jump table and back to the loop start. This address cannot be known in advance because the program must be fully relocatable; fortunately, the return address stacked by the BSR is precisely the address in question, so the initialisation routine needs merely to copy it from the stack to the jump table


Addressing The Problem
SWI VECTOR
$C015
JUMP TABLE
$C015
JMP ENTRY
PROGRAM MEMORY
PROGRAM:
BSR INIT
BEGIN MAIN LOOP
INIT:
COPY ENTRY ADDRESS
GET START ADDRESS
STORE START ADDRESS
RTS
PC
ENTRY
STACK
NEXT FREE BYTE
ENTRY