subroutine held in ROM, or for a library routine that always occupies the same position in memory — parts of the disk operating system, for example.

When the processor encounters a BSR or JSR instruction, the current value of the program counter is 'pushed' onto the system stack using the S (stack pointer) register.If your subroutine uses the S register for anything other than a further subroutine call, you must ensure that it gets restored to the correct value. The address of the subroutine is calculated (in the case of BSR) and loaded into the program counter. Thus, the next instruction to be accessed will be the first one of the subroutine. You must be sure, therefore, that the subroutine begins with an instruction and not a byte of data.

A subroutine must end with an RTS (ReTurn from Subroutine) instruction, the effect of which is to 'pull' the old value of the program counter back off the stack. Execution of the program will then continue from where it left off before the subroutine call.

The example program we give here is rather more complex than those we have given previously, but it can be made more manageable by the use of a subroutine. The program searches a table containing strings of unequal length, and extracts a value associated with one particular string. The strings are held in the normal way: beginning with a byte indicating the string's length, followed by the characters that make up the string, and ending with a 16-bit address associated with the string.

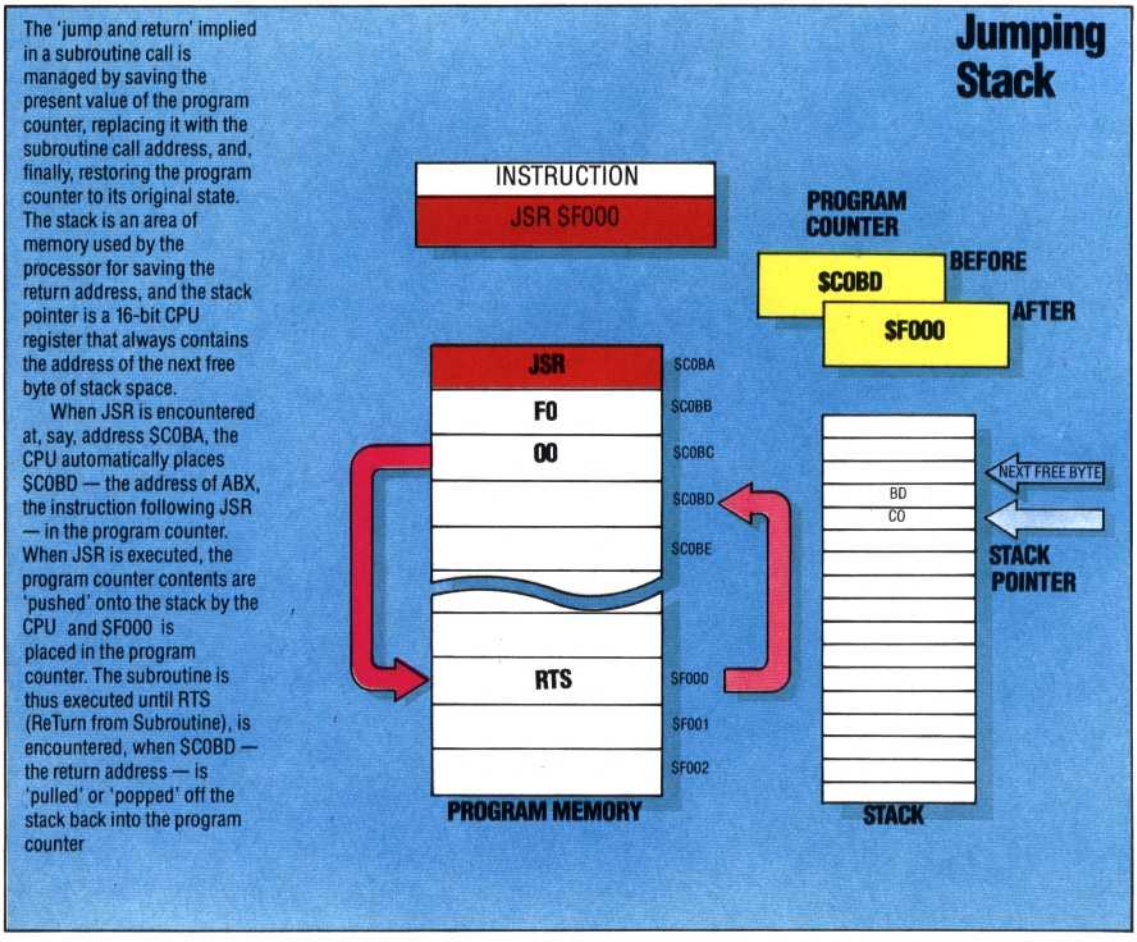
The end of the table is marked by a zero length string — in other words, there is a value of zero where the length byte should be. We shall assume that the address of the start of the table is held in $10, and the address of the string whose match we have to search for is held in $12. If the duplicate is found in the table, then the corresponding address is to be held in $14. If the string is not found, then $12 and $14 should be both set to zero.

STRING-MATCHING

String-matching is a task that occurs in many situations — most notably in managing a BASIC interpreter's string variable accesses: each identifier (or variable name) must be replaced by the address in which the value of that variable is stored.

The problem divides easily into two parts: we must step through the table until either the string we are looking for is found or the end of the table is reached. At each stage in the search we must compare two strings (the one we are looking for and the one at the current position in the table) to see if they match.

This string comparison is an obvious candidate for a subroutine, because not only is it going to be used more than once in the program, but it also enables us to split up the problem into useful



The 'jump and return' implied in a subroutine call is managed by saving the present value of the program counter, replacing it with the subroutine call address, and, finally, restoring the program counter to its original state. The stack is an area of memory used by the processor for saving the return address, and the stack pointer is a 16-bit CPU register that always contains the address of the next free byte of stack space.

When JSR is encountered at, say, address $C0BA, the CPU automatically places $C0BD — the address of ABX, the instruction following JSR — in the program counter. When JSR is executed, the program counter contents are 'pushed' onto the stack by the CPU and $F000 is placed in the program counter. The subroutine is thus executed until RTS (ReTurn from Subroutine), is encountered, when $C0BD — the return address — is 'pulled' or 'popped' off the stack back into the program counter