the 16-bit value at the address currently in S is loaded into X, and the contents of S are then incremented by two. The second diagram shows these changes.

More than one register can be pushed or pulled at a time. Consider the instruction:

```
PSHS X,Y,U,A
```

When more than one register is pushed like this, the order in which the registers are listed is ignored, and instead the registers are always pushed in this order: PC (the program counter register), U or S, Y, X, DP (the direct page register), B, A and CC (the condition code register). They will, of course, be pulled off in the reverse order. The only real constraint on stack operations is that neither S nor U can be pushed onto its own stack.

The stacks are used in general programming as convenient places for fast, temporary storage, but their major uses come when dealing with interrupts (more about these later in the course) and subroutines. We have already seen how the contents of the program counter register are automatically pushed onto the stack when a subroutine is called, and pulled on return from the subroutine (RTS is equivalent to PULS PC). Either stack, but particularly S, can also be used to pass parameters to a subroutine.

The method we have used so far for passing parameters via the registers (as in the Jump Table program on page 639) has two major weaknesses. First of all, there may be more parameters to pass than there are registers, and, secondly, it can prove awkward when the routine called uses a register holding a parameter that you need to retain. There are, however, two other common techniques for passing parameters:

1) The data can be stored in the middle of the program by using FCB, FDB or FCC directives immediately after the subroutine call. The value of the program counter register pushed onto the stack by the JSR instruction gives the address of the first of these values (since PC always points to the next byte after the current instruction), and can be used to obtain all of them, with suitable offsets. The first example program illustrates this technique. Care must be taken to arrange the RTS instruction so that it passes control to a real instruction, and not to an item of data.

2) The data can be loaded into registers and pushed onto the stack before the subroutine call, from which it can be pulled into the subroutine and used. Care must be taken here that, at the RTS instruction, the stack pointer will access the previously stacked PC return address. The second piece of code illustrates this technique. This is generally a more useful method than the first.

In both methods, the dual role of S and U as index registers as well as stack pointers means that items on the stack can be referenced by indexed addressing in addition to being easily accessed for removal from the stack. This makes it easier to ensure that the correct items are left on the stack for the return.

Multiple-Precision Addition

Here are two pieces of code showing alternative methods of performing multiple-precision addition using the stacks. In the first piece of code, the parameters are placed after the subroutine call. A typical call to add two four-byte numbers at $100 and $104, leaving the result at $108 would be:

```
        BSR    MPADD
        FCB    4
        FDB    $100
        FDB    $104
        FDB    $108
        ORG    $1000
MPADD   PSHS   X,Y,U,A,B,CC
        LDU    9,S
        PULU   X,Y,B
        LDU    ,U
        ANDCC  #%11111110
LOOP    LDA    ,X+
        ADCA   ,Y+
        STA    ,U+
        DECB
        BNE    LOOP
        LDU    9,S
        LEAU   7,U
        STU    9,S
        PULS   PC,U,X,Y,A,B,CC
```

Length of each number (in bytes)
Address of first number
Address of second number
Address of result
Save all registers. This pushes nine bytes onto the stack
This instruction will load into U the PC return address pushed onto the stack by the BSR instruction
Treating the data that comes after the subroutine call as though it were a stack, we pull the addresses of the two numbers into X and Y, and the length into B
The address of the result goes into U
Clear the carry flag
Perform the addition of one byte to the other, adding with carry
Store the result
Check if the addition is finished
If not, LOOP
Else get return address and add seven to skip over the seven bytes of parameters stored after the call
Restore all the registers to the state that they were in before, restoring the PC instead of using an extra RTS

The second example performs the same operatior but pushes parameters onto the stack. The callin sequence would be:

```
        LDU    #$108
        LDX    #$100
        LDY    #$104
        LDA    #4
        PSHS   U,X,Y,A
        BSR    MPADD
        ORG    $1000
MPADD   PSHS   X,Y,U,A,B,CC
        LEAU   11,U
        PULU   X,Y,B
        ANDCC  #%11111110
LOOP    LDA    ,X+
        ADCA   ,Y+
        STA    ,U+
        DECB
        BNE    LOOP
        PULS   PC,U,X,Y,A,B,CC
```

Result address
Operand address
Length
Put parameters onto stack
Save all registers
Saving the registers takes nine bytes, and the return address takes two bytes — a total of eleven in all. U now points to the parameters
Proceed as before