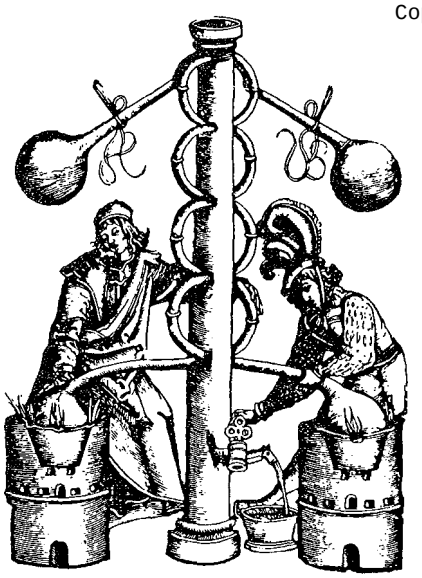

Microcomputer interfacing

Subroutines and stacks

By David G. Larsen, Peter R. Rony, Jonathan A. Titus, and Christopher Titus

IN SOME of our previous columns, we have provided examples of short *subprograms* or *subroutines* that could be used by a main program. Subroutines are powerful software building blocks. They facilitate program development since they may be written and tested apart from the main body of software. In addition, they can be adapted for use with almost any type of program. In this month's column, we will focus upon their operation as well as on the use of stack instructions.

We have previously discussed the use of both *unconditional* and *conditional jump* instructions, which transfer computer control to another software task starting at the sixteen-bit address specified within the jump instruction itself. The jump instruction is a one-way branch since it points to a single address, as illustrated in *Figure* 1. However, short subprograms that are used repeatedly exist in many software tasks. Examples of such tasks include mathematical computation, control, and teletypewriter input/output routines. It seems wasteful to duplicate these subprograms throughout the main program, so an attempt is made to separate them at the end of the main program and, in some manner, branch to them when they are needed.

The use of jump instructions to access these subprograms will not be successful since there will be no *link* back to the main program once the subprogram's task is completed. The use of an additional jump instruction at the end of the subprogram that points back to the main task is unsatisfactory since jump instructions can point to a single address. This is .also illustrated in Figure I. The jump instructions at 2 and 3 point to the same subprogram, but upon completion of the subprogram's task, the jump instruction at 4 can only provide a link to one place. A new operation, the *call* instruction, is required that has the effect of inserting the subprogram's software steps in the main program flow at points 2 and 3 but without the problems associated with the use ofajump.

The call instruction, like the jump instruction, transfers control to another portion of the software. When that portion has completed its task, however, control is returned to the main program. *Figure 2* illustrates two subroutines used by the main program, each being accessed by a call instruction, which specifies the starting address of the subroutine as a sixteen-bit, or two-byte, word. *At thecompletion ofthe subroutine, control* is *returned to the next instruction that follows the three-byte call instruction.* Through the use of call instructions, the program shown in Figure 2 has inserted the subroutine program steps in the flow of the main software task. Subroutine 2 is used only once, but subroutine 1 has been used twice, although it is present only once in the microcomputer's memory.

Each subroutine is accessed via a call instruction and ends with a

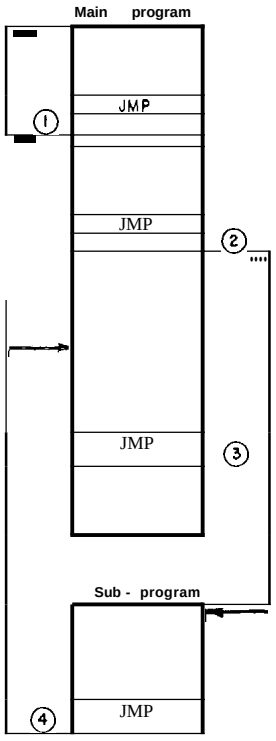

Figure 1 *Diagram illustrating the characteristics of the jump instruction.*

Correction: In the column entitled, "Microcomputer interfacing: Using Digital-to-Analog Converters," May, page 127, an instruction byte was inadvertently omitted from Table I. Immediately after the MOVAM instruction at HI = 000 and LO = 011, the INXH instruction (043) must be inserted. All subsequent instruction bytes must be moved to the next higher address.

Mr. Larsen, Department of Chemistry, and Dr. Rony, Department of Chemical Engineering, are with the Virginia Polytechnic Institute and State University. Mr. J, Titus and Dr, C. Titus are with tv-chon, Inc.

Table 1

Software example showing a typical assembler output

```
003 000 061   START,  .003 000
003 001 377           LXI SP
003 002 000           377
003 003 333           000
003 004 005   LOOP,   IN
003 005 376           005
003 006 026           CPI
003 007 312           026
003 010 015           JZ
003 011 003           DETECT
003 012 303           0
003 013 003           JMP
003 014 003           LOOP
003 015 171           0
003 016 323   DETECT, MOVAC
003 017 007           OUT
003 020 166           007
                      HLT

                      / SYMBOLIC ADDRESS OF START

                      /INPUT DATA FROM PORT 5

                      /COMPARE IT TO 026

                      /IF IT MATCHES GO TO MDETECT-

                      /IF IT DOESN'T MATCH, GO TO
                      /LOOP AND CHECK AGAIN
```

return instruction, RET. The return is a one-byte instruction that does not contain any address information, yet it acts to return Control to the main program. The return of control takes place since the call instruction saves a *linking*, or return, *address* that acts to branch the computer back to the address of the instruction immediately following the three-byte call. The return instruction causes the microcomputer to retrieve the address from storage and to use it as the link back to the main task.

The sixteen-bit return addresses associated with call instructions are stored in an area of read/write memory called the *stack*. The transfer of address information is performed automatically by the 8080 microprocessor chip to and from the stack during call and return operations. Thus, the 8080 chip *pushes* the return address onto the stack during execution of a call and *pops* it off the stack during a return. The actual memory area set aside for the stack is determined by the programmer through the use of an LXI SP instruction, which loads the sixteen-bit starting address of the stack into the *stack pointer register* located within the 8080 chip. It is the programmer's responsibility to set up a stack pointer before calls and returns are used; the programmer must also make certain that the stack area will not be used for other purposes during program execution:

In the program example shown m *Table* 1, we decided that the stack should have a starting address of 003 377. The first step in the main program, therefore, is to set the stack pointer to this address using the LXISP instruction. Later, when a call instruction is executed, the 8080 chip transfers the return address to the stack area of R/W memory. If the stack pointer is initially set at

---

SUbprogram — A section of a program that may perform a particular operation to be used with a larger program. Subprograms are not general purpose and are generally used by one program.

Subroutine — A general-purpose program that may be called or used by a main program or another SUbroutine.

Main program — A short notation to indicate the software tasks that will occupy most of the computer's time.

Link — A pointer address that will point the computer to another section of a program or back to a program that it may not be currently Using.

Nesting — The Operation of one SUbroutine within another, I.e., a.1-min delay Subroutine may call a 1-sec delay subroutine 60 times.

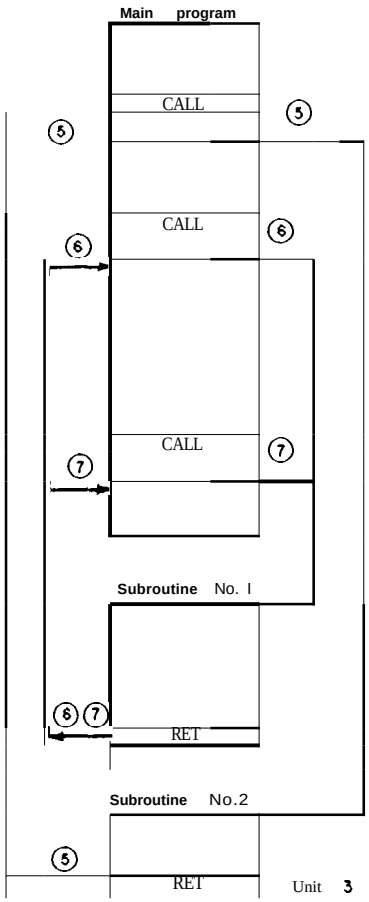

Figure 2 *Diagram illustrating the characteristics of the call and return instructions.*

address X, the return address is stored with the low address byte in location X-2 and the high address byte in location X-I. Thus, *the stack adds address data at addresses below the address value of the stack pointer.* When the return address is popped back into the 8080 chip, the stack pointer is automatically incremented back to address X as the return address is retrieved byte by byte. When the next subroutine is called, the same stack locations are used for storage of the new return address since the old return address has already been popped back into the 8080.

Subroutines may be placed one within another, or *nested.* This means that one subroutine may call another. In this way, a control subroutine may, in turn, call a timer subroutine. When the timer subroutine has completed its task, it causes a return to the control subroutine. This situation requires two levels on the stack, or four R/W memory locations, since two full 16-bit return addresses must be maintained on the stack while the timer subroutine is in operation: 1) the return address for the timer-to-control link and 2) the return address for the control-to-main-task link. *The stack operations take place automatically whenever a call or a return* is *executed.* The call and return instructions may be either conditional or unconditional, but each subroutine must contain at least one return instruction.

Recall that the 8080 chip contains seven 8-bit general-purpose registers, the accumulator (A), B, C, D, E, H, and L. In programs where subroutines are used, there may be register conflicts since the subroutine and the main task may both require the use of a specific register. Sometimes this problem may be solved by choosing another register, but this is not always possible, particularly when the A register and the flags are involved. To avoid register conflicts, it is possible to use the stack for temporary data storage. All of the internal 8080 registers may be pushed onto the stack and popped back into the 8080 as needed. Data are stored and retrieved as register pairs, with register A and the flags forming a two-byte word that is treated as a register pair.

The subroutine in Table 1 is a time delay routine in which registers D, E, A, and the flags are stored on the stack. At the completion of the subroutine, the data stored on the stack are retrieved and placed back in the internal registers. The complementary operations of stack storage and retrieval are called *push* and *pop,* respectively. Notice that the stack pointer is initialized at the start of the program, before any other instructions are executed.

The use of subroutines in a program allows many complex tasks to be subdivided into small segments that are easy to link together and relieve the problem of continuously rewriting frequently used program steps and routines. You will find that a personal library of frequently used subroutines is indispensible when doing programming..