A computer system for automation of a laboratory


IBM Scientific Center
Palo Alto, California

INTRODUCTION

In the past, scientists have applied the digital computer in an off-line fashion to the problems of analyzing the voluminous data produced by their laboratory instruments. The use of computer techniques for digital filtering, peak finding, and spectral decomposition has greatly increased the rate at which experimental data can be analyzed. More recently, many instrument manufacturers and users have also begun to apply computers on-line to their instruments in an effort to increase the rate at which useful experimental data can be obtained. This paper describes a laboratory automation computer system which simultaneously supports multiple closed-loop experiments and data analysis programs.

Let us first distinguish between instrument automation and laboratory automation. In the former case a single computer, usually small, is devoted to a single instrument, or at least to one instrument at a time; whereas in laboratory automation, a group of instruments in a laboratory is automated using one central computer system. A discussion of the relative advantages of these two modes of automation follows.

There are several advantages to a dedicated computer. The most important of these is that of isolation. Each individual user prefers to concern himself with his own problems. He does not want malfunctions of other instruments or their interfaces to jeopardize his experiment. Programming considerations associated with his own instrument are sufficiently complex that he doesn't want to worry about other users' programming problems too. He is rightfully afraid of being forced to factor parts of his programming requirements into general purpose programs which serve the entire laboratory (programming by committee).

Immediate accessibility to the computer system can be guaranteed in a devoted computer configuration. This fact also encourages one to favor multiple computers, one per instrument; however, if a time-shared computer system could almost always be made available in a period of a few seconds, or at most one or two minutes, this would not be intolerable. The prospects, however, of having to schedule one's experiments and schedule the use of the computer facilities long in advance is very unpalatable to most users. In general, users prefer fewer facilities which are routinely available to larger facilities available by appointment only. Hence, the desire for immediate computer access tends to favor multiple instrument automation over laboratory automation.

Relative costs should also be considered. A computer system which is capable of expanding to handle the requirements of an entire laboratory will, of necessity, be more expensive, even in a minimal configuration, than a computer system capable of automating a single laboratory instrument. Therefore, when one is taking the first step toward laboratory automation, that is, the automation of a single instrument, there is a very strong tendency to automate that instrument using one small computer since the initial cost is considerably lower. Quite understandably, an organization is reluctant to commit large capital and manpower resources to a project in which it has little or no experience. However, if the ultimate goal of completely automating the laboratory is considered, many of the considerations discussed below imply that a single, shared laboratory computer would provide more performance per dollar.

To realize all of the potential of automation, one must not only acquire data but also control the instrument during the data acquisition step,
process the data which has been acquired, standardize it, compare it against known parameters (e.g., compare an unknown spectrum against a file of spectra of known compounds for identification), and finally present the results in a form which is usable to the experimenter. Data acquisition and control steps can very often be adequately performed by a small computer. However, the data reduction steps, the comparison with data files, and the presentation of results in usable form often require much larger computer capabilities. One solution to this is to record the raw data, which has been acquired with a small stand-alone computer, on a recording medium such as magnetic tape or punched paper tape. This data may then be processed on a large computer when time is available. However, if the raw data is at all voluminous, and magnetic tape must be used, the cost of the magnetic tape drive relative to the cost of the small computer can become very high. Hence, this approach tends to encourage one to minimize the quantity of data taken, often resulting in less precise results. Furthermore, the turnaround time on very large computer facilities still is much longer than the individual investigator would like to wait between epochs of his experiment. That is, if he could have the data from the last epoch back quickly, these results could be used to determine conditions for his next experiment.

A larger shared computer has several advantages for the automation of a laboratory. It can have sufficient core and processing capabilities to do a large portion of the data reduction required in most laboratories without resorting to a large central computer. By dynamic allocation of system facilities one may take advantage of the fact that most analytical and spectroscopic instruments have low duty cycles (i.e., much of the time is spent in sample preparation or with the instrument completely idle). This dynamic allocation of system facilities will allow one to reduce the total size of the required system below the sum total of each experiment's requirements. In addition, under such a system, background data reduction tasks may be carried out by utilizing excess capacity that exists at any moment.

Another advantage of a shared computer is that more sophisticated input/output devices are available to all users. The advantages of having access to large disk files, line printer, card reader/punch, and magnetic tape units are obvious. Only the largest devoted computer/instrument combination could justify the most modest of such devices.

As a result of the above considerations, we have designed and implemented a monitor system. Our goal was to provide a system which could serve a laboratory containing a group of analytical and/or spectrographic instruments operating in a dynamic mode, i.e., a research or development environment. This system, the Palo Alto Laboratory System (PALS), features complete program independence and complete system independence of each instrument from all others. An application program for one instrument, either at the time it is written or executed, need in no way take into consideration other programs running in the system simultaneously. Furthermore, application programs need not be modified as a result of a change in the total instrument configuration attached to the computer system. Each instrument may have its own individual data path to the core of the computer via data channels, so that there need be no sharing of the interfaces between various instruments. Data acquisition associated with the various instruments is completely asynchronous. Closed-loop control capabilities are provided with a response time of the order of 50 ms. The system dynamically allocates core, disk space, and I/O devices to provide maximum usage of these facilities.

This system uses an IBM 1800 computer. It is possible to operate this system on a computer with 16K words of core; however, it is more useful if the laboratory to be automated is sufficiently large to support a 24K or 32K word machine.

A new device, a digital multiplexer, has been designed and built, specifically to support this laboratory automation system. This digital multiplexer channel provides up to 32 discrete data paths between the laboratory and the core of the computer. Via cycle stealing, it allows data to be acquired from, or presented to, the individual instruments in a demand/response mode with a minimum of computer overhead. The reduction in computer overhead increases the allowable total data acquisition rate from all instruments by more than an order of magnitude. Data acquisition is in a demand/response mode which is of great value in that it allows the instrument to indicate when data is available rather than letting the computer determine when data should be presented. An example of the usefulness of demand/response data acquisition is in acquiring data from an infrared spectrometer which has automatic scan suppression. Since the scan rate is a
function of the first derivative of the absorption, data acquired at equal intervals of time would not be at equal increments in wavelength or wave number. However, with a demand/response interface, the instrument could be run with scan suppression, and demands to take data could be made by the instrument at equal wavelength or wave number increments.

The monitor system

The PALS monitor system is made up of a group of relocatable modules which are initially stored on the disk. Modules serve either input/output devices or are responsible for initiating program loading, linking to the Job Control Language (JCL), controlling multi-task communications, and monitoring time slicing operations.

The only portion of the system which is not relocatable is a 200-word area in low core contains information related to hardware wiring, such as an interrupt transfer vector and the word count and address registers for the various subchannels of the multiplexer.

The various modules making up the system communicate with each other by means of task control blocks and task-complete control blocks. When one module has a task to be performed by a second module, a task control block is generated by the originating module. The address of that block is passed to the second module which performs the task as soon as it is able. When the task has been completed, the second module generates a task-complete control block which is returned to the originating module. The task-complete control block indicates whether the task has been completed successfully or unsuccessfully, and the reason for any unsuccessful completion if appropriate.

Two monitor modules, a real-time module (foreground) and a non-real-time module (midground) are responsible for the execution of all users' programs and for communication between users' programs and the rest of the system. A second non-real-time module (background) is not part of the system's modules, but is read from disk upon request.

Input/output device modules service disk files, all the subchannels of the digital multiplexer, printer, card reader, the typewriter terminals, etc. There may be several modules stored on the disk servicing the same input/output device at various levels of complexity. That is, one may be a very sophisticated package, having many operations included in it, while another services the device at a minimal level having only one or two very simple operations implemented in it.

The system is configured by reading in a pack of control cards which indicate which modules should be loaded and link-edited together. The reconfiguration of the system, depending on the various users' needs, is very easy and requires only a few seconds. At this time the operator may choose, for example, whether he wishes to have full printer support requiring over a thousand words of core, or minimal printer support requiring only a couple of hundred words of core. It is also possible to eliminate devices which are not to be used. After this deck of cards is read in, the system is cold-started with a one-card cold-start routine which loads and link-edits the appropriate system modules in low core.

Dynamic core allocation

All of core not taken up by the system is divided into pages of 512 words and all of these pages are put into a pool of free pages (Figure 1). The loader module is responsible for allocation of the pages of variable core. When a task is given to the loader to load and set a user's program into execution, the loader places that user's program into any pages of core which are presently available. These pages need not be contig-

![FIGURE 1—Core allocation map](From the collection of the Computer History Museum (www.computerhistory.org))
uous, since the loader takes care of altering the relocatable addresses within the user's program at load time, so that it may execute out of non-contiguous pages of core. In addition, the first and last words on each page of a user's program and all words which are not modified during execution of that program are storage-protected at load time. This provides a high degree of protection of one user's program from another.

Systems subroutines, such as floating addition, multiplication, division, sine, etc., are shared among individual user's programs. These subroutines are stored on the disk, in groups which are assembled into a block that will occupy one page of variable core. At load time, when the loader encounters a call to a systems subroutine from a user's program, the loader checks whether the page on which that system subroutine exists has previously been loaded into core. If it has, the loader links the program being loaded to that systems subroutine. If the systems subroutine has not previously been loaded, the loader loads the page which contains the called systems subroutine from the disk and links it with the program being loaded.

When a program has completed its execution and has called EXIT, a task is given to the loader to return all the pages associated with that user's program to the pool of free variable core. The loader clears all the storage protection bits from these pages and interrogates to see if the terminating program was the only program using any of the systems subroutines. All systems subroutine pages which are not being used by other programs are also returned to variable core. However, those subroutine pages which contain subroutines being used by other application programs still in execution are not affected.

Disk allocation

Many of the problems encountered in a laboratory require random access to individual data or groups of data within large files, hence the file structure on a peripheral disk is of considerable importance. It is usually impossible to define the ultimate size of a data file before it is generated so that a dynamic file allocation is highly desirable. Disk files in the PAL system are organized as logical tapes which are automatically expanded or contracted according to the present length of the data table written thereon. These logical tapes are defined by name and allocated by the system one cylinder of the disk file (2560 words) at a time. Such files may be used in much the same way as a physical magnetic tape, that is, one can read, write, backspace any number of words, rewind, etc. In addition, the inherent advantage of the disk file is preserved, i.e., reading, writing or altering single words anywhere on the logical tape in the direct access mode is still possible. The system allows this by keeping an internal word counter which points to the location where the last access was made. Alteration of this pointer to any value is easily done by giving an instruction to the system. If the data table exceeds the length of the first cylinder, the system automatically adds cylinders to the logical tape, restricted only by the maximum value the word count pointer may attain, 32767 words. The tape may be closed at any value to retain the file for future use. If the length of the tape upon closure is less than a previous length, the excess cylinders are returned to the system, providing automatic contraction of the file. For example, if a previous data file required six cylinders and the present one only requires three and a half cylinders, the fifth and sixth cylinders will be returned to the pool of empty cylinders for reallocation by the system.

Multitasking

Allocation of the CPU is implemented using the multilevel interrupt structure of the 1800. Therefore, if a task is being processed and a task of higher priority is initiated, the low priority task is suspended and the higher priority task is processed immediately.

The entire PALS system is oriented toward performing a variety of tasks, many being associated with input/output. In general, it takes 12 words of a user's program to specify a task for the system. These tasks are accepted by the system and performed as soon as possible. Multiple tasks may be queued for a single input/output device; e.g., the line printer may have a current task in execution while five other tasks are waiting for the line printer to be free. A given application program may have several tasks outstanding simultaneously. For instance, a given application program may instruct the system to (1) acquire a block of data from a given instrument, (2) read a card in the card reader, (3) print a line on the line printer, (4) light a light at the user interface, and (5) write a block of data on a logical tape. These tasks would be given to the system sequentially, but all five tasks would be set into execution before the first one was completed.
CYCLE STEALING
MACHINE CHECK
ANSWER INTERRUPTS
START INPUT/OUTPUT
REAL TIME USERS MONITOR
LOADER
NON-REAL TIME USERS MONITOR
HOUSEKEEPING
JOB CONTROL LANGUAGE
ASSEMBLER
WAIT LOOP

FIGURE 2—Priority list for CPU cycle allocation

If the user's program wishes to initiate an I/O operation, it does so by giving specifications of the task to the user's monitor, which in turn generates a task control block and gives it to the system module responsible for executing that task. As soon as the task has been started or queued, and before the task is completed, control is returned to the user's program. From this point the user's program may send out additional tasks to the system, each time control being returned to the user's program. Two of the arguments in a specified task are entry points to the user's program. One is associated with a normal return address, and one is associated with an abnormal return address. If the task is completed successfully, the user will regain control at his normal return address, whereas if the task is completed unsuccessfully, control will be returned to the abnormal return address.

A priority list for allocation of CPU cycles is given in Figure 2.

The highest priority is cycle stealing for the transfer of data between the various I/O devices, including the instrument interfaces and memory.

The next highest priority is for servicing machine check conditions.

... Below that, hardware interrupts are answered and I/O devices are started. Each of the system modules which is responsible for an I/O device may queue, to an indefinite length, tasks to be performed in conjunction with its particular I/O device. When a hardware interrupt occurs, indicating the completion of a task or a subtask, the next task or subtask is initiated as a portion of the interrupt handling routine. In addition, when a task has been completed as a portion of the interrupt handling routine, a task-complete control block is returned to the system module which originally issued the task.

... Next is the real-time user's monitor under which all real-time programs are executed. A user's program in execution under the auspices of the real-time user's monitor may be in one of three states: passive, queued, or active (Figure 3). When a program originally goes into execution it is placed at the bottom of the queue. As programs are taken from the top of the queue and placed in execution, the program at the bottom of the queue works its way up through the queue. Finally, the program is taken from the top of the queue into active status and the user's monitor transfers control to the user's program. The user's program maintains control for doing processing.

After giving out a number of tasks and having done a certain amount of processing, one of two things can happen which will take a user out of the active status. The first possibility is that he has completed all processing and has given out all the tasks he desires at that time and can do nothing more until one of his I/O tasks has been completed. At this point the user executes a relinquish operation which then causes the user's monitor to remove him from the active status and place him in the passive status. Control then passes to the next
programming in the active queue. The user in the passive status does not relinquish core, only his position in the active queue.

The other possibility is that of a "time-out." Each time a user's program is put into execution, the user's monitor sets an interval timer for a nominal five milliseconds, and at the expiration of this time the user's monitor takes control away from the user's program, saves all status and registers, and puts the program from the active status to the bottom of the queue (time slicing). The user then must work his way up through the queue to the top again to resume processing. Using five millisecond time slices, it has been our observation that a real-time user usually relinquishes before he is timed out.

When a user is put into the passive status, the only way he can return to active queue is as a result of a task-complete control block being returned to the user's monitor, indicating that one the user's I/O operations has been completed, either successfully or unsuccessfully. At this point the user's monitor takes the user's program out of the passive status and places it at the bottom of the active queue. A guaranteed response time, i.e., the time from completion of an I/O operation until the time that a user's program may act upon that completion, may be implemented by limiting the number of users allowable in the real-time execution and by implementing five millisecond time slices. If, for instance, the real-time user's monitor limits the number of users to five, a worst case condition would be four users in the queue. When an I/O operation has been completed or a hardware interrupt occurs, the maximum length of time necessary to answer would be five milliseconds for each of the people in the active queue, plus not more than 20 milliseconds overhead associated with the higher priority operations (if overhead were 100 percent). This would mean a total of 40 milliseconds maximum from the time that the interrupt condition occurred until the time that the user's program gained control of the CPU for a five millisecond time slice. Note that interrupt servicing and queue manipulating do not interfere with on-line data acquisition which proceeds via CPU cycle-stealing.

Immediately below the real-time user's monitor in priority is the module controlling loading of programs and dynamic allocation of core. It was given a lower priority than the real-time monitor in order to avoid interference with operational real-time programs. Because the loader module can accept tasks from other modules, operations such as having a real-time program load a non-real-time program and vice versa are possible. A typical example of load time is 1.7 seconds for a seven page program, during which time all of the operations previously discussed under core allocation are performed. This overhead occurs just once at the time each program is initially loaded, and it is negligible compared to the manual set-up times needed to ready an instrument or experiment.

Next is the non-real-time user's monitor. Programs executing in a non-real-time status are executed under the control of this monitor which has the same algorithm for scheduling time to the various user's programs as does the real-time monitor, except the time out period in the non-real-time monitor is a nominal one hundred milliseconds.

Below the non-real-time monitor in priority are various housekeeping modules. These modules are in general responsible for code conversion and spooling operations between various I/O devices. For instance, a user's program may give a task to the system to print an entire logical tape. This task would be executed by a housekeeping module, which in turn would give out subtasks to read sectors of the logical tape and to print the individual lines. This module would also be responsible for the code conversion from EBCDIC to printer code.

Below the housekeeping modules in priority is the module which deals with job control language. This module offers a fairly high level of conversational interaction between the operator of the system and the system. From the console typewriter the operator may load a program or set a program into execution, may cancel a program, may get a dynamic dump of a program while it is in execution, or a dynamic dump of any area core, may get a dump of a logical tape, may define or scratch a logical tape, or may get a status of the entire system.
The lowest priorities are devoted to the language assembler, and a wait loop.

The PALS language

Language requirements for a laboratory automation system include the ability to easily program multiple on-line data acquisition and control tasks as well as off-line data reduction or analysis tasks. Data acquisition and control tasks demand programming of a number of input/output interactions with sensor-based devices. Logical operations are required for the various control functions. Past experience indicates that data reduction and analysis tasks are best served by the FORTRAN or PL/I type of language.

The approach chosen for the PALS system was a macro language with statements natural to the laboratory environment. There are statements for easy handling of sensor-based input/output (e.g., multiplexer channel commands, analog inputs, and a series of special logical statements used to set up bit patterns for control of instrument interfaces, etc.). FORTRAN-like statements are available for data analysis, and they require very little re-learning for users familiar with FORTRAN.

Some examples of the various types of PALS statements may serve to indicate the salient features of the language.

I/O OPERATIONS

SUBCHANNEL OPERATE
SCOP SC1, 3, DTNAM1, NRET, ARET

Write the contents of table DTNAM1 out over subchannel 1 in demand/response mode (operation code 3). After successful completion of the operation, return control to normal entry point NRET, if not successful to abnormal entry point ARET.

READ LOGICAL TAPE
DRTP WC, NAME, BUF

Read the number of words contained in location WC from logical tape NAME and place them in core starting at location BUF.

READ CARD INTO CARD BUFFER
CRDR

CONVERSION

CARD BUFFER TO INTEGER VECTOR
CBIV N, M, VEC, AI

The contents of address N is the number of card columns to be used for each element; location M contains the number of elements per card; VEC is the starting address of the vector; and location AI contains the subscript of the first vector element to be filled by the present card buffer.

INTEGER TO EXTENDED PRECISION FLOATING POINT
FLOT I, E

Convert the integer at location I to extended precision floating point number at location E.

INTEGER TO EBCDIC
IEBC I, CH, EBCDIC

Convert the integer at location I to CH number of characters starting at location EBCDIC.
MATHEMATICS

STANDARD FLOATING ADD
FADD A, B, C, ERR

MULTIPLY VECTOR ELEMENTS
IMPX A, I, B, J, C, K, ERR

SQUARE ROOT FUNCTION
FSQT A, B, ERR

Add the standard precision floating point numbers located at A and B and put the result in location C. ERR is the entry point of a user error correction routine.

Perform a floating point multiply of the Ith element of vector located at A by the Jth element of vector located at B and place result in the location of the Kth element of vector located at C. Branch to ERR if any error.

Take the square root in floating point of the number at A and place the result in location B. Branch to ERR if any error.

The macro language permits programmers to mix assembler language with macro statements. It is at the user's discretion to define new statements to meet the needs and level of programming experience of the scientists writing applications.

The PALS macro processor is treated somewhat as if it were an application program, except that it is executed in background mode. It is not paged but is loaded into a partition at the high end of variable core. When the loader is asked to load the assembler, the request is queued until the 12 pages at the high end of core are all free. At that point, the limits of variable core are lowered about 6000 words from the top end of core and the assembler is loaded into this partition. When the assembler has completed its operation, its core block is returned to variable core. This means that several minutes may be required from the time it is requested to load the assembler until the assembler is actually loaded. However, since this system is built on the basis that assembly should be a background operation, this allocation scheme appears to be satisfactory.

Application programs

A joint study was carried out with Varian Associates in order to investigate and demonstrate the usefulness of a time-shared, laboratory automation computer system. A simulated research laboratory, containing an M-66 medium resolution mass spectrometer, an A-60 NMR spectrometer, two Aerograph gas chromatography columns, and a Statos recorder, was linked to an on-line IBM 1800 computer (Figure 4). Each of the instruments was interfaced with the computer via prototype Varian interfaces, an example of which is diagrammed in Figure 5.

Each instrument interface provided facilities which allowed the spectroscopist to have complete control over his use of the computer from his remote spectrometer. These facilities consisted of backlighted pushbuttons, lights, and thumb switches. Prompting lights, operated under program control, served to indicate the present status of the operating program. Functions executable from the remote console included loading and aborting application programs, controlling branch points within the application programs, and entering parameters during the execution of programs. In general, the recorder associated with a particular instrument was used as the graphical output device; the Varian Statos recorder being used for the two chromatographs. Tabular reports were printed on the shared-line printer.

A disk resident master program was associated with each instrument. When an experiment was to be performed, the master program was loaded into the core memory by depressing a
pushbutton on the interface. The function of the master program was to initialize the spectrometer and to provide a choice of the available application programs to the user. After the master program was loaded, it armed appropriate pushbuttons for program selection. This was indicated by lights behind the several pushbuttons. When

![Figure 4](image1.png)

**FIGURE 4**—An automated analytical laboratory

![Figure 5](image2.png)

**FIGURE 5**—Typical control panel of instrument interface

**DATA MANIPULATION**

**TRANSFER VECTOR ELEMENT**

TVEI  
A, J, B, K  
Transfer integer element Aᵢ into integer element Bᵦ.

**INCREMENT VARIABLE**

INC  
I, J  
Increment integer at I by the integer J, where −128 ≤ J ≤ 127.

**FIND MAXIMUM ELEMENT OF INTEGER VECTOR**

IMAX N, YDATA, I, YMAX, IMAX  
Search N elements of the vector starting with the ith element of vector YDATA and place the maximum values of YDATA and its index I into locations YMAX and IMAX respectively.

**PROGRAM SWITCHING**

IF  
I, J, A, B, C  
Branch to A if integer quantity (I–J) < 0, to B if = 0, and to C if > 0.

**COMPUTED GO TO**

GOTO  
A, B, C, D, POINT  
Branch to A, B, C, D if location POINT contains 0, 1, 2, 3 respectively.

**REPEAT LOOP**

REPT  
LOC, N, I, K  
Execute all statements starting at location LOC through the REPT statement the number of times contained in N. Each time through the loop, increment integer at I by the number K.
one of these programs was selected, it was loaded and the master program would exit (release its core). After the subprogram completed its function, and before it exited, it requested that the master program be reloaded. This mode of operation minimized the core requirements of a single user.

Several application subprograms, which performed some of the more elementary instrument functions, were jointly specified and written by Varian and IBM. With the A-60 NMR it was possible to: acquire data in a demand/response mode by sweeping the magnetic field; time average the data by repeating the scans any desired number of times; digitally smooth the acquired data; and control the magnetic field homogeneity. The mass spectrometer programs acquired data in a demand/response mode; replotted any desired portion of the data on the instrument recorder; found all peaks in the spectrum and normalized their intensities; and found the five highest peaks and identified the compound by comparison with a table of spectra of known compounds on the disk file. The gas chromatography programs acquired data, detected and resolved peaks, calculated their areas and wrote a report giving retention times with peak areas.

Actual experience with the above system was quite good. Programs were easily and quickly written using the macro language. No noticeable interference between users occurred, even when all instruments and data processing I/O devices were running simultaneously.

In conclusion, we believe that we have been able to produce a system for use in laboratory automation which provides each individual user the isolation, the availability, the real-time control responsiveness, and the price per instrument associated with multiple computers, one per instrument. In addition, this system provides powerful input/output devices, disk files, and a large amount of support for the I/O devices and the files, which is not normally found on a small dedicated computer. Each user, then, has the impression that he has a large computer attached to his instrument and completely at his disposal. The 1800 PALS program, excluding the instrument application programs, is available from the IBM Type III library (PID #5778).

**INSTRUMENTATION CONTROL**

**READ DATA AND WRITE LOGICAL TAPE**

```
RDLT
SCPNT, WCPNT, LTPNT, NPRA, APRA
```

From subchannel number contained in SCPNT, read the number of words contained in WCPNT (1 ≤ WC ≤ 32767), and store on a logical tape whose name resides in LTPNT. Return control to normal entry point NPRA for successful completion of the read, and to APRA for abnormal conditions.

**ALTER SUBCHANNEL BIT**

```
ASCB
SC, BIT, N
```

Alter one bit (number of bit contained in location BIT) of a subchannel register (number of subchannel contained in location SC) to a new value N = 0 or N = 1.

**CONVERT THUMB SWITCH TO FLOATING POINT NUMBER**

```
CTTF F
```

Read a set of manual switches on instrument interface and convert the setting to a floating point number at location F.

**ASSIGN PROCESS INTERRUPTS**

```
ASPI PINO, ENT
```

Assign a program entry point ENT to the process interrupt whose number is contained in location PINO.