SLAC-PUB-2031
October 1977
(I)

A Multi-faceted Data Gathering and Analyzing System*

David B. Gustavson and Keith Rich
Stanford Linear Accelerator Center
P. O. Box 4349
Stanford, California, 94305

We have implemented a low-cost general purpose data gathering and analyzing system based on a micro-processor, an interface to CAMAC, and a phone link to a time-sharing system. The parts cost for the micro-processor system was about $6000. The micro-processor buffers the data such that the variable response of the time-sharing system is acceptable for performing real-time data acquisition. The full power and flexibility of the time-sharing system excels at the task of on-line data analysis once this buffering problem has been solved.

Introduction

This system was developed to fill a need for a computer controlled data acquisition system to aid in the testing of complex particle detectors for use in basic research on elementary particles ("high energy physics"). Goals included low cost (<$10K), flexibility, and general usefulness as a graphics terminal for data analysis (this paper was typed using the microcomputer system as a terminal).

The power and flexibility of the system come largely from utilizing the vast capabilities of a large computer's time-sharing system rather than trying to put too much sophistication into a stand-alone micro-processor. This leaves the micro-processor free to do its primary task well. The main purpose of the micro-processor is to acquire and buffer data, and to control peripheral I/O devices. The fact that this works well enables us to live with the variable response of the time-sharing system rather than needing the fast response of real-time support.

In fact, no special hooks of any kind are needed into the time-sharing system; the micro-processor appears to the time-sharing system to be a standard terminal. As we develop the data analysis software, we can bring to bear the full range of large computer tools available under the time-sharing system. Only for the primitive data gathering and buffering need we program the micro-processor. Having the dominant portion of the programming in a large computer is a significant advantage during the last-minute tuning of the system to an experiment, when constraints lead to annoying delays and errors. Many data analysis features can be implemented and debugged from a normal terminal but yet in the real environment, thus allowing development to occur in parallel with production.

The modesty of the micro-processor system also results in portability of the hardware and in reliability, since there is less to go wrong.

Fig. 1 The microcomputer system, showing one of the control keyboard stations.

The Micro-processor Hardware

The heart of the micro-processor is an Intel 8080 central processing unit in a MITS Altair 8800 cabinet, utilizing the S-100 bus on a Processor Technology Corporation MB-1 motherboard. There is a simple programmed data transfer interface to the SLAC CAMAC[1] branch highway. There are two keyboards and two scrolled TV screens which are driven by video display modules purchased from Processor Technology Corp. Each screen has 16 lines of 64 characters refreshed from 1K of two-port memory dedicated to the display, but also addressable as part of the micro-processor's memory. There is a Qume Sprint-55 daisy-wheel printer rated at 55 characters per second. It can partial space in 1/120" horizontal and 1/48" vertical increments in either direction, thus allowing it to do graphics (lines are formed from dots). There is a 2400 baud hardwired phone line and a 300 baud dial-up phone line. The micro-processor can dial or answer the dial-up line. The memories consist of 32K bytes of Intel 2107A dynamic RAM and 8K bytes of Intel 2708 EPROM on a Cromemco BYTESAVER. Each have access times of slightly less than half a micro-second.

A simple photoelectric paper tape reader and a paper tape punch are included in the system so that it can be operated in a stand-alone fashion, but normally programs are loaded over the 2400 baud phone line. Paper tape will be obsoleted soon by a planned floppy disk system. A simple 200 Hz clock interrupt provides a time base for automatic dialing, process time slicing, and maintaining a time of day clock.

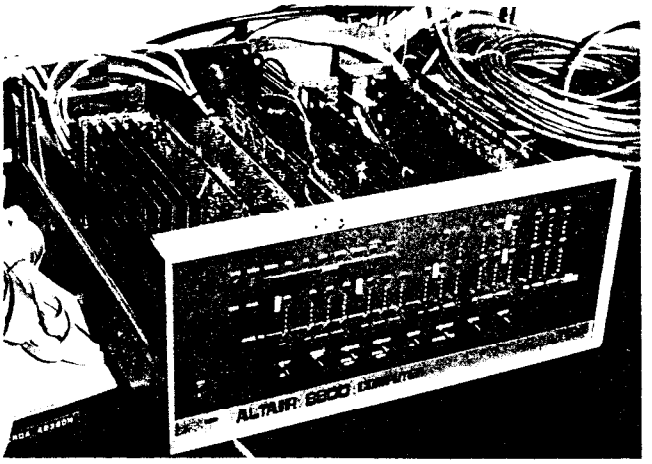


Fig. 2 Inside view of the microcomputer, showing the memory, interface cards, and cpu card.

The Micro-processor Software

The distinctive features of the micro-processor software are the very flexible multi-programming and message switching capabilities which have only modest hardware requirements. For example, the multi-programming supervisor occupies only 427 bytes of code space.

*Work supported by the Department of Energy
(Contributed to the IEEE 1977 Nuclear Science Symposium, San Francisco, Ca., October 19-21, 1977)

The Multi-programming Supervisor

The multi-programming supervisor consists of routines which systematically manage processes, resources and memory. Processes are the invocations of algorithms and process management allows multiplexing of these algorithms. Each process is executed serially while the scheduler controls the concurrency of execution of the active processes. Resources are physical I/O devices or any abstract entities which are to be used serially rather than concurrently. Dynamic memory allocation facilitates reentrant code (code which may be executed concurrently by multiple processes).

Process management includes dynamic initiation, termination and scheduling of concurrent processes. This mechanism is the essence of multi-programming and these process management routines provide a framework within which the working programs may be coded without knowledge of the eventual process configuration. There are four process related routines:

```
CALL FORK(ADDR,VAL1,VAL2,VAL3)
```

is similar to the usual subroutine call:

```
CALL ADDR(VAL1,VAL2,VAL3)
```

except that FORK merely enters the called routine into the concurrent process table and then returns. Execution of the called routine proceeds concurrently with any other existing processes as well as the calling routine.

```
CALL STOP
```

terminates the calling process.

```
CALL CNCL(RESOURCE)
```

terminates the process which controls the specified resource (unless that is the calling process).

```
CALL WAIT
```

causes the calling process to release control of the CPU. Control is then given to the scheduler and will return to the calling process when the scheduler so decides. Any scheduling algorithm may be employed, but a simple round-robin queue will suffice unless otherwise dictated by special circumstances. Time slicing is provided by the interval timer interrupt. If no WAIT occurs between two consecutive timer interrupts, the interrupt routine merely simulates a "CALL WAIT" prior to the instruction to which the interrupt would normally return.

It should be noted that this process structure differs significantly from most. Many systems have a fixed number of processes which are connected to interrupt levels (usually called foreground tasks) plus one process which is not connected to any interrupt level (usually called the background task). In contrast, this system treats all processes as abstract entities which are not connected to interrupt levels, but are rather controlled by a centralized scheduler. Interrupt routines (if used at all) do not execute as part of any process. Instead, they act as part of an external I/O processor, although they may invoke the scheduler when they exit rather than merely returning to the point of interruption.

The centralized scheduler is therefore not constrained by the structure of the hardware, and may dynamically vary the number of processes over a wide range as well as adjusting priorities if appropriate. The details of the scheduler are transparent to the processes, which simplifies the writing of most of the code.

Resource management requires that processes request and be granted exclusive control of resources dynamically before they use them. The processes should relinquish control when they no longer require it and process termination must ensure that all resources controlled by the terminating process are freed. The request and grant mechanism fits logically into the concept of "open" while the relinquish mechanism is associated with "close". The exclusiveness of control is necessary to resolve conflicts as concurrent processes (which need not otherwise be aware of each other) compete for resources. Processes which request busy resources must wait until those resources are available and the resource management routines have granted exclusive control of the resources. There are two resource management routines:

```
CALL GETR(RESOURCE)
```

requests exclusive control of the specified resource by the calling process. Control is returned only when this has been granted; until then the calling process waits.

```
CALL RETR(RESOURCE)
```

returns exclusive control of the specified resource to the system when it is no longer needed. In order to avoid system interlocks in more complex systems, it is sometimes necessary to have a third resource management routine:

```
CALL TSTR(RESOURCE)
```

which acts exactly like GETR if the specified resource is immediately available. However, if the specified resource is unavailable, it simply returns with an indication that it failed rather than waiting for the resource.

Memory management requires that processes dynamically request and be granted memory space for data. They should free the memory space when it is no longer needed and process termination must also free any data memory space associated with the terminating process. Whereas processes request specific resources, they merely request the amount of memory which they need. The memory management routines decide which memory to grant and cause the requesting process to wait if the requested amount of memory is unavailable. There are two memory management routines:

```
CALL GETM(SIZE)
```

requests SIZE bytes of dynamic memory. The starting address of the acquired memory is returned when its use has been granted. In the application described here, SIZE is always assumed to be 256 bytes.

```
CALL RETM(ADDRESS)
```

returns the memory when it is no longer needed. Under highly dynamic conditions, it may be valuable to have a third memory management routine:

```
CALL TSTM(SIZE)
```

which acts exactly like GETM if the requested memory is immediately available. If insufficient memory is available, it simply returns with an indication that it failed rather than waiting for the memory.

The I/O System

There are a number of queues which provide for the buffered transfer of data between the physical devices and the processes which control them. The output queues are normally emptied by the interrupt service routine of the associated device. The input queues may be filled by the interrupt service routine of the associated device, but may also be filled by output from a process. These devices and queues are associated as follows:

device	queue	usage: input (/output)
KB1	any	physical keyboard one
KB2	any	physical keyboard two
	c	input for KB1 control process
	d	input for KB2 control process
	k	keyboard input for slow phone process
	l	keyboard input for fast phone process
PI1	i	slow phone input
PI2	j	fast phone input
	x	input to BASIC, Assembler, etc.
	y	output from BASIC, Assembler, etc.
	z	input to CAMAC process
	w	output from CAMAC process
RD1	r	paper tape reader

device	queue	usage: output (only)
	G	garbage pail (discard output)
PR1	Q	Qume printer
PU1	P	paper tape punch
PO1	O	slow phone output
PO2	N	fast phone output
TV2	T	TV2 screen
TV1	V	TV1 screen

The existence of the queues allows the I/O drivers to run on interrupt level (when appropriate) while the processes merely fill and empty these queues asynchronously. With the exception of the keyboard driver, the I/O drivers associate a physical device directly with a queue.

The keyboard driver is specially extended such that either keyboard may be associated with any input queue. This is accomplished by typing the appropriate key sequence from the keyboard and thus it is under control of the operator rather than any process. This allows the operator to multiplex the use of his keyboard for input to any input queue and hence to any process which reads from a queue.

The phone drivers have several special features which facilitate the half duplex communication mode supported by our time sharing system. When transmitting, the phone output driver feeds each character both to the phone line and to the phone input driver to provide local echoing. It keeps track of the characters sent, until encountering a carriage return, so that any transmission interrupted by a reverse break (caused by an incoming message) can be retransmitted automatically. This eliminates the well known phenomenon in which someone always sends you a message when you have nearly finished typing a very complex formula, forcing you to type it again. Our timesharing system optionally sends a DC1 character when it is ready for input. We use that character to switch from receive to transmit mode, and we switch from transmit to receive upon transmission of carriage return or break. Because of the queues, the keyboard is always enabled so that commands or responses can be typed ahead even while text is being received and printed. The typed-ahead text is not echoed until it is actually transmitted, so the printed record is chronologically correctly ordered. The typeahead capability is much more convenient than one would at first imagine.

When entering text on an ordinary terminal, one has to be alert not to begin typing until the next line number prompt is complete, or characters will be lost from the beginning of the line. With this system one is free to type full speed even if time sharing response time is slow. Another use is to start a listing and type ahead the logoff command before leaving for lunch. After the listing is complete, the logoff command will be transmitted automatically, thus eliminating unnecessary connect-time charges.

The keyboard driver takes special action when it sees the break key while communicating with a terminal process input queue (k or l). The break key causes all typed-ahead text to be discarded, to facilitate recovery from an unanticipated situation such as a mistyped command. If it is desired to type-ahead a break, the control-C character will produce a break when it is encountered by the phone output driver.

The Qume output driver includes graphics routines for vector generation and allows production of plots coded in Tektronix storage display terminal format (see Fig. 4) or coded in Gencom or DTC format.

A special driver handles putting output characters into input queues. This has the appearance of a loop-back mode in which characters appear to exit from an output queue only to appear again in an input queue.

The TV screen driver writes directly to the screen, which operates at memory speed and therefore needs no buffering.

The driver for G merely discards the characters which it receives. It is a sharable resource.

The Scanner

The scanner is at the heart of the micro-processor application software. It contains the command interpreter, provides for data flow switching, and implements a useful form of output device sharing. It is reentrant and many processes use this routine concurrently. It is started with three parameters: an input queue, an output queue and an interlock character. For our typical application the scanner is normally running with the following queue pairs and interlock characters.

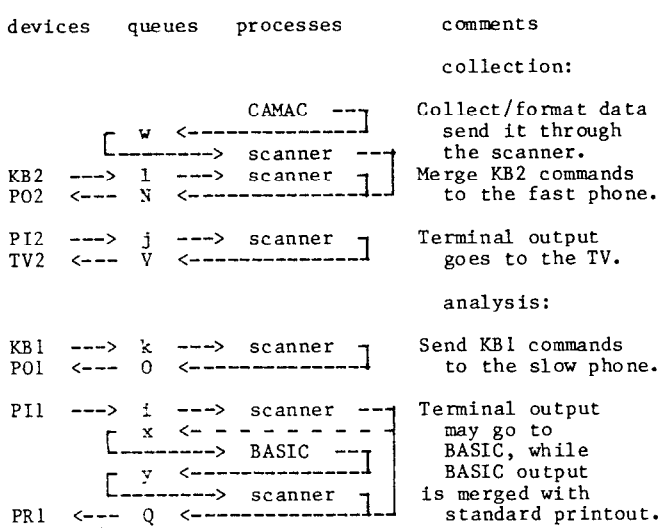
	in	out	char
process 1	c	G	CR
process 2	d	G	CR
process 3	k	O	CR
process 4	l	N	CR
process 5	i	T	DC1
process 6	j	Q	DC1
process 7	y	V	DC1
process 8	w	N	CR

When a process initially enters the scanner, it requests exclusive control of the input queue. It retains this control for the duration of the process. The main body of the scanner is a repeat forever loop with no exit conditions.

During steady state operation, the scanner reads characters from the input queue (which was filled by an interrupt level service routine or another process) and writes characters to the output queue (which will be emptied by an interrupt level service routine or another process). Exclusive control of the output queue is requested (if not previously obtained) only when a character has been acquired which requires outputting. After the character has been accepted by the output queue, it is checked to see if it was the interlock character. If it was, then exclusive control of the output queue is relinquished (otherwise it is retained). In any case, the input queue is then interrogated for the next character. This interlock mechanism allows multiple processes to share output devices without intermingling lines. The use of interlock characters other than carriage return (e.g. form feed) allows other than line by line interlocking (e.g. page by page).

A coded sequence of characters is recognized by the scanner as a command to switch to a different queue for input or output. The output can be switched from the default established at process initiation time to any other output queue. Another coded sequence returns the output assignment to the default queue. When input is switched it remains so until it is switched again or an "end of data" is encountered. This switching of input was intended for implementing command files in subroutine-like fashion and is little used at present, though it should be valuable when the system is expanded to include a floppy disk. For loading BASIC programs from paper tape, the input (of any scanner process) may be switched to r and the output to x. The generality of this mechanism allows the scanner to act as a dynamic "patch-panel jumper". Although this may seem totally confusing to the uninitiated, the power to patch software together in this fashion is tremendously useful. One fact that may not be obvious is that the coded sequences of characters which do this switching may be generated dynamically under control of time-sharing or BASIC programs as well as by being typed by the operator. Also, the special ability of the keyboard to drive any input queue allows for operator intervention by inserting command character sequences where desired in order to make temporary reassignments of output.

A real example which we have found useful while running an actual physics experiment[2], collecting 40 channels of analog data, worked as follows:



Thus with one microcomputer we were able to collect data, simulate two computer terminals, and run an analysis program in BASIC at the same time.

The CAMAC Handler

The CAMAC handler polls the CAMAC interface until a data ready condition occurs. It then reads the event, re-enables the apparatus, formats the event and transfers it to output queue w which normally is spooled by the scanner to the output queue N. The existence of the scanner in this flow is to allow patching to some device other than N (e.g. T while testing). If our CAMAC transfer rates were high, we would go to the trouble of implementing the interrupt capability in the CAMAC interface hardware and inserting a queue for use with CAMAC. This queue would be filled by an interrupt level I/O driver and emptied asynchronously by the CAMAC handler at a lower peak rate.

The BASIC Interpreter

The BASIC interpreter reads from input queue x and writes to output queue y which is normally spooled by the scanner to output queue V. The existence of the scanner in this flow is to allow for patching to some device other than V (e.g. N to transfer BASIC program statements or data to the time-sharing system). Although BASIC on a micro-processor is slow, this tool is quite useful for a wide variety of problems which occur from time to time during the operation of the system (e.g. CAMAC diagnostic programs).

The Time-sharing System

The time-sharing system runs on an IBM 370/168 and talks to the outside world through an IBM 3705 communications controller and a terminal processing system called MILTEN which was developed at Stanford University.

The Editor

The WYLBUR[3] editor of Stanford University is used. It has an active file in which editing is done. The active file consists of a single file of numbered lines. About 5000 to 20000 lines may be held in the active file depending on their length. An extensive associative and explicit editing vocabulary is available. The active file may be filled from or saved into the batch file system, the time-sharing file system, the editor exec file, or a time-sharing program's data buffers. The active file is paged from a drum so that it need not be entirely in memory and it can be recovered if the time-sharing system crashes.

A program of commands may reside in the exec file. Exec files are useful for simplifying routine procedures, reducing routine typing, and facilitating transfers of data and control to and from the microcomputer. The exec file can hold about 250 lines of commands. Variables, expression evaluation, transfer of control, and conditional tests are provided. The exec file may be loaded from the active file or from a batch file. Exec files may contain or generate editor commands, time-sharing supervisor commands, time-sharing program commands or data, or active file data. An exec file may be suspended while the time-sharing supervisor or a time-sharing program has control, but will resume execution when the editor regains control unless an invalid command is encountered or a terminal break occurs.

The Time-sharing Supervisor

The time-sharing supervisor is called ORVYL[4] was developed at Stanford University. It supports a file system which is built on top of the batch file system and has some features not available directly in the batch file system. Although the time-sharing system's file system may occupy several volumes, and files are physically scattered over the file system (with index records to keep track of things), the user merely refers to the files by name and lets the file system find them. Space is allocated by the system as it is requested rather than when files are created. Files consist of numbered blocks of up to 2K bytes. Any file may be referred to in direct access mode by referring to the block numbers. Sequential access is provided by accessing the blocks of a file in ascending order of block numbers. File characteristics may be assumed by a program or specified in a mode word which is associated with the file. The default format has blanks compressed and line numbers associated with the lines. An extensive security system allows controlled access to individual files or entire user file structures by read, write and append permits. A quota system controls the allocation of blocks to users.

Traps may be set, and branches, stores and fetches may be monitored. Memory and registers may be examined, changed and dumped to the active file. The latter allows offline dumps for examination later or semi-automatic debugging under control of editor exec files.

The Data Analysis Program

The operator types on a keyboard and views output on a screen and a printer. He need not be aware of the various components within the system. He simply types commands (e.g. exec from #reload) which trigger sequences of actions by the system, including execution of exec files, execution of time-sharing programs, prompting for further information, loading of programs into the micro-computer, initiating processes, and printing and plotting of information.

The main loop requests the editor to proceed until it finds an input command or an exec file command which it does not understand. These unknown commands are then passed to the main loop of the data analysis program which parses the command and looks up the verb (first word) in its command table. If it finds the command in the command table, it calls the associated command processing routine. This routine then processes the command which may include reading more information from the terminal. If the command is not found in the command table, it is passed to the time-sharing supervisor command processor, which may in turn pass it back to the WYLBUR text editor or to the MILTEN terminal interface control program. After successful execution of the command (at any level) or after an error message (if no match is found), control returns to the top of the main loop.

A generalized histogramming package known as HPAK[5] is employed. It is capable of both one dimensional and two dimensional histograms and a wide variety of display capabilities. Non-graphical information is accumulated in the editor's active file for later processing and printing.

A graphics system called the SLAC Unified Graphics System[6] is utilized for the display of graphical information. It allows the choice of two dimensional and three dimensional projected plots and of high resolution Tektronix scope and Versatec printer plots.

Acknowledgements

We wish to thank Leonard J. Shustek for help with the design and debugging of the early microcomputer hardware and software. John E. Zolnowsky provided us with an excellent BASIC interpreter, and helped tailor it to our needs. Shustek and Zolnowsky also provided and supported a cross-assembler and EPROM programming facility. Glenn Herrmannsfeldt assisted in the writing of the application software for the timesharing system. We are especially grateful to Justino Escalera for his rapid and accurate work in laying out and constructing the wire-wrap and printed electronic circuitry. We are also grateful to Dr. Robert L. Anderson for arranging suitable working quarters, and to our group leader, Prof. David M. Ritson, without whose support this project would not have been possible.

References

1) Simple Versatile CAMAC Crate Controller and Interrupt Priority Encoder Module - IEEE Transactions on Nuclear Science Vol NS-22 no. 1.
2) Tests of Proportional Wire Shower Counter and Hadron Calorimeter Modules - R. L. Anderson et. al., Session 2B1, High Energy Physics Instrumentation - Calorimeters, this symposium.
3) WYLBUR/370 - SCIP, Stanford University.
4) ORVYL/370 - SCIP, Stanford University.
5) DPAK and HPAK - SLAC report no. 196, C. A. Logg, A. M. Boyarski, A. J. Cook, R. L. A. Cottrell, Stanford Linear Accelerator Center.
6) The SLAC Unified Graphics System - Robert C. Beach, Computation Research Group, Stanford Linear Accelerator Center.

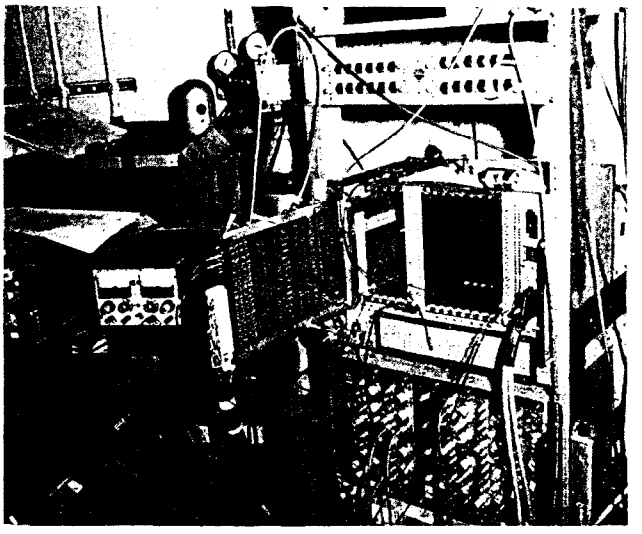


Fig. 3 A typical laboratory setup using the microcomputer for testing CAMAC modules. The local control station is not visible.

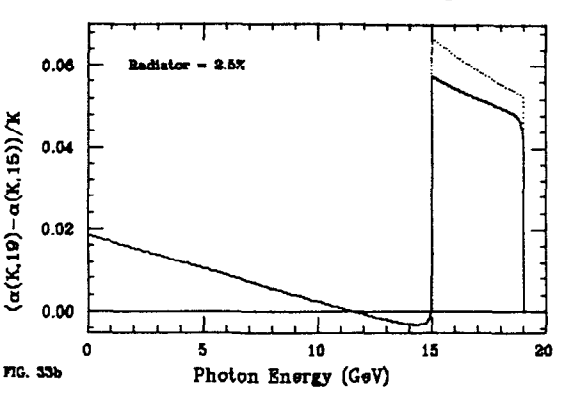


Fig. 4 Typical graphic output from an analysis program running on a large computer. This plot took 5 minutes to produce on the microcomputer. Much of this time is avoided if typewritten rather than fancy plotted lettering is chosen.