The General Automation 18/30 as a system for the general analysis and acquisition of data in physiological psychology*

ROBERT M. CHAPMAN and JOHN A. CHAPMAN
Eye Research Foundation of Bethesda, Bethesda, Maryland 20014

We are attempting to handle two classes of problems with our General Automation 18/30 system (Anaheim, Calif.): (1) to use the computer directly at the level of the experiments (a) to control the experiments and (b) to collect the data; and (2) to use the computer with the data from experiments, i.e., to analyze the data and make theoretical calculations. These two classes of problems are similar to those which, in other contexts, have sometimes been characterized as (1) process control and (2) batch processing.

At the level of the experiments, for example, we want the system to time and present various visual stimuli according to a particular experimental paradigm while the S's physiological responses (EEG, averaged evoked potentials, EKG, etc.), behavioral responses (switch closures), and control pulses are being collected. Some of our research led us to elaborate statistical analyses, and these large problems forced us to go beyond the capacity of the small computer system we have been using (a Packard Bell 250). Our difficulties in dealing with large computer centers reinforced the notion of attempting to perform as much computation on site as possible. This was one of the considerations in setting up a medium-size computer system rather than staying with a small system. Another consideration was the lowered costs of computers, especially the central processors. It is possible to obtain a fourth-generation medium-size central processor at about the same or lower cost than previously designed small-scale computers and get more powerful organization and command structure. For example, our GA 18/30 system (8K 18-bit core, 512K-word disk, 8 priority interrupt levels, 5 direct memory-access channels, 3 programmable timers, A/D with 16-channel multiplexer, D/A with 8-channel output holding amplifiers, 16 digital inputs, 16 digital outputs, paper-tape reader, card reader, and teletype) cost less (approximately $45,000) than a small-scale system bought about 10 years ago (4K 22-bit delay-line memory, A/D with 7-channel multiplexer, D/A with 2 channels, 22 digital input-outputs, paper-tape reader, and flexowriter). Furthermore, the cycle time is 10 to 1,000 times faster on the 18/30.

Most of what we have to say applies to a variety of computers. However, one aspect of the GA 18/30 is relatively unique: its command structure has been patterned after two IBM machines from which the 18/30 gets its hybrid name. The IBM machines are the 1800, a process controller discussed in the next talk, and the 1130, which was designed as a batch processor. The GA 18/30 has the same commands plus another class of "register-to-register" commands. This means that the library of routines developed for those IBM machines may be used on the faster GA 18/30.

As one example, we had obtained a large set of data (EEG scores from a special-purpose EEG scorer and answers to a questionnaire) from a large group of Ss. Using FORTRAN, we wrote a program which gave us the correlations among all the variables. In examining the correlation matrix, we saw some interesting patterns which led us to want a multiple-regression analysis that could combine the questionnaire variables in an equation best predicting the EEG scores. We had not written a multiple-regression program and had only a few days left before a meeting. We dipped into the IBM stockpile for the appropriate subroutines, which we linked together with a FORTRAN program. The program read our correlations off our disk (where they had been stored by our correlation program) and typed the results of the multiple regression on the Teletype relatively quickly.

We could have had a computer center perform these analyses; why do it ourselves? (1) Time is one of the considerations. The turn-around time not only includes delivering the data to the computer center, but also communicating what is to be done. (2) We really wanted to operate on these data in an interactive way, i.e., depending on the outcome of an analysis on a subset of variables, we would try various subsets. This is not interactive at the level of the experiment, but rather at the level of data analysis. (3) Computer centers prefer to have data come in on cards, and ours were not on cards. Therefore, considering the long chain of people and expense of the machine involved, it seems less prone to error and cheaper to do whatever analyses we can ourselves.

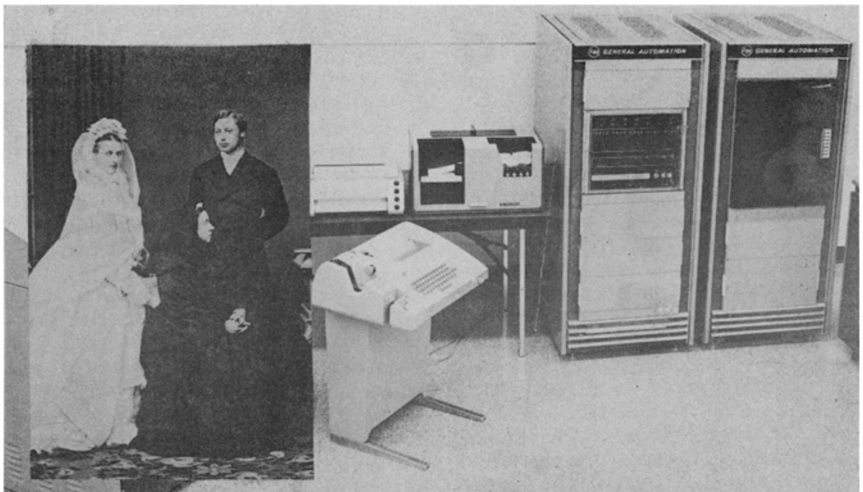

Fig. 1. The Queen is in her counting house.

*The research was supported by Grants PHS 5-S01 RR05698-02 and NIH-NEI-5-R01-EY00490-04. This report is not to be construed as an endorsement of the products named.

Table 1
DBOS: Disk Based Operating System

Logical Units		Physical Devices or Files		
CC	Control command input	TY	Teletype keyboard	
SI	Symbolic input	CR	Card reader	
SO	Symbolic output	WS	Working symbolic file	DISK
BI	Binary input	WB	Working binary file	DISK
BO	Binary output	WB	Working binary file	DISK
LO	Listing output	LP	Line printer	
IS	Intermediate symbolic	WS	Working symbolic file	DISK
OM	Operator messages	TY	Teletype printer	
CI	Core image data	WC	Working core image file	DISK
LB	Binary library	LB	Directoried library file	DISK
SL	System log	LP	Line printer	
SB	Secondary binary library	CR	Card reader	
UL	User library	UL	Directoried user library file	DISK
	User disk temporary	DK	Unformatted disk I/O file	DISK
	User pack disk temporary	DP	Disk	
	NO	NO	Delete I/O	
		DS	Directoried source language	DISK
		DC	Directoried core image	DISK
		DJ	Directoried job string	DISK
		PP	Paper tape punch	

MACHINE CHARACTERISTICS

What are the characteristics of the GA 18/30? It has a 16-bit word, plus a parity bit and a storage-protect bit. The storage-protect bit may be set on any word in core memory, and it prevents changing the contents of those locations without going through special procedures. The storage-protect bit, then, can protect programs that are running when you do something else. Unfortunately, it doesn't get around the problem of protecting the data which have to be changing. We have not used it yet, but possibly the storage-protect feature could be useful in a research environment.

The memory in the GA 18/30 cycles at 960 nsec. Presently, we have 8K of core memory, although the enclosure is wired to hold up to 32K of core. The processor is fast and powerful. It does not have a hardware square-root command (like our PB 250), but it does have all the double-precision arithmetic and shift commands and hardware multiply and divide. The hardware multiply, for example, takes 12 microsec.

Getting a computer is somewhat like buying a car. One of the similar merchandizing practices that often is used is selling the basic machine at a reasonably low price. However, by the time you drive it off the lot, it is almost certain that a few extras have been added on to make the system useful. Most of the features of the 18/30 central processor are available on most other computers, but they are often merchandized as extras. The 18/30 philosophy seemed to have been to include these "extras" as standard. It has 5 direct memory-access channels (cycle-steal data channels), 8 priority interrupt levels, of which 6 are for external use, 3 hardware index registers, 3 internal, programmable, crystal-controlled timers (0.1, 1, 10 msec), and multiple addressing modes. Single-word instructions can specify addressing relative (±128) to a base register (instruction counter or 1 of 3 index registers); double-word instructions can specify direct (up to 32K core memory), indexed, or indirect addressing without need for memory paging. The internal timers give a priority interrupt when they time out and can be preset, read any time, and started and stopped under program control. These timers are very useful in controlling experiments and logging data.

The piece of hardware that has made the biggest difference from our previous computer experience is the disk. It is a mass-storage device that considerably eases our interaction with the system. Disk drives have gotten cheaper and now cost about the same as magnetic tape. Ours uses a single-platter (two-surface) disk cartridge which holds a half-million 16-bit words. Our disk drive has an average access time of 160 msec (adjacent track access time of 1.74 msec). The function of the disk is to hold programs and data in a way that is faster but similar to the use of DEC tape on other systems. Various locations on the disk may be addressed, and thus the disk may be used as an extension of core memory.

MONITOR PROGRAM PACKAGE

A variety of operating-system packages are available from GA. We have been operating our system under GA's DBOS (Disk Based Operating System), which gives us the power and ease of device-independent programming, job sequencing, programming in symbolic assembly language and/or FORTRAN, building programs from subroutines held on parts of the disk, etc. DBOS is a program which may be rapidly loaded from the disk by an initial program-load feature. Part of this program ordinarily resides in core, occupying approximately 4K.

Table 1 shows part of the DBOS organization which employs a separation between logical input/output units (which have various functions) and physical devices or files (which perform the input or output). There are a standard set of assignments connecting the logical units to the physical devices; for example, control-command inputs (logical I/O unit) is usually assigned to the Teletype (physical device). However, the control-command inputs can be temporarily changed to the card reader or a disk file, known as directoried job string, by typing: $CC=CR or $CC=DJ (Name 1). Thus, we can assign various of these physical devices (files) to various of these logical units at will. Various parts of the disk are thought of as independent devices or files; the large contribution of the disk to the power and ease of the system is attested to in Table 1 by seeing the large number of devices or files which are various parts of the same disk. Several of the logical I/O units (listing output; system log) are usually assigned to a physical device (line printer), which we do not have. At system generation (or later), it is possible to reassign these logical units to a Teletype, which we do have (e.g., $LO=TY,P). It is possible to have several of the logical units connected to the same physical device, so our Teletype does yeoman service. At run time of a program, we can decide which devices are used by equating the logical units used in programming with the desired device files. Some of these functions are illustrated by the example in Table 2.

The DBOS package includes several programs which operate quickly using the disk files. These include FORTRAN IV, the 18/30 Symbolic Assembler, a co-resident DEBUGging program, the Core Image Converter program (CIC), and a library of subroutines (many for FORTRAN use).

The outputs of the FORTRAN IV and Assembler programs are in relocatable binary code, as are all the subroutines in the library file on the disk. The CIC links the main program and the subroutines together by operating on the relocatable binary code. The result is a nonrelocatable core-image binary code where absolute locations have been assigned, and the finished program, residing in the working core-image file on disk (WC), is ready to be run. The program may be run immediately ($LOAD) or stored on the disk in the directoried core image file ($COPY, WC, DC

Table 2
User Controlled Analysis Example

Line	Teletype Typing		Explanation
1	DBOS CC	M:	DBOS monitor running
2	? $JOB	U:	Initialize job and logical files
3	? $SI = DS(AFORM)	U:	Reassign symbolic input to DS(AFORM)
4	? $EDIT	U:	Call EDIT program (to change the No. 2 FORMAT statement)
5	? @B = N	U:	Insert no leading blank spaces
6	? @ +4, 1	U:	At fourth line delete one line (the old No. 2 FORMAT statement)
7	? 2 FORMAT (5x,F15.5//)	U:	(Enter) New No. 2 FORMAT statement
8	? @ C = Y	U:	Yes, complete and close files
9	DBOS CC	M:	DBOS monitor running
10	? $SI = WS	U:	Reassign symbolic input to working symbolic (EDIT output)
11	? $F	U:	Call FORTRAN IV compiler
12	END OF COMPILATION	F:	Compilation complete
13	DBOS CC	M:	DBOS monitor running
14	? $EOD	U:	Write end of data (file) mark
15	? $CIC	U:	Call core image converter program
16	? *BUILD	U:	Execute CIC, construct AFORM program
17	DBOS CC	M:	DBOS monitor running
18	? $SI = DS(SETO3)	U:	Reassign symbolic input to the data on disk file DS(SETO3)
19	? $LO = WS	U:	Reassign listing output to working symbolic file on disk
20	? $LOAD	U:	Load and execute modified AFORM program
21	STOP	P:	End of AFORM program run
22	DBOS CC	M:	DBOS monitor running
23	? $SI = WS	U:	Reassign symbolic input to AFORM output
24	? $LO = TY	U:	Reassign listing output back to Teletype
25	? $AVACE	U:	Call analysis of variance program stored at DC(AVACE) on disk
26	3128.123	P:	A O
27	54436.320		N U
28	Mean		O T
29	17.442		V P
30	13.470		A U
31	16.361		C T
.			E
.			
.			
45 etc.	Source df Sum of Squares Mean Square F Ratio		

?—Typed by computer (keyboard release) *P:—User-program-typed output from computer*
U:—User-typed input to computer *$—Command symbol in DBOS*
M:—Monitor-typed output from computer *@—Command symbol in EDIT*
F:—FORTRAN-typed output from computer **—Command symbol in CIC*

(NAME3)), if one wants to call the program "Name3," and run at later times by a simple command ($NAME3).

The usefulness of the disk system can be easily seen when using the edit system. One disk file is assigned to symbolic input ($SI=DS(INPUT)), and another file is assigned to the symbolic output (normally SO=WS). Then, using the EDIT commands, the program or data is transferred from one file to the other with the insertions and deletions desired. The DBOS control commands can then allow the corrected file to get the same name as the old file had, deleting the old, incorrect file ($REPLACE, DS(INPUT),WS). Combining the input/output assignment features with the EDIT program, step-by-step control of the data analysis can be maintained with ease, flexibility, and speed.

EXAMPLE OF USER CONTROL AND DATA ANALYSIS

To illustrate some operating procedures in controlled data analysis, an example is outlined below. The data for this example were the output from a large program run at a computation center and were received by the 18/30 as a punched card deck. We wished to run 15 separate analyses of variance on subsets of that data array. The analysis of variance program (AVACE), which we had written, required the input data to be arranged in a particular order (case-wise) which was different from what appeared on the punched cards (factor-wise). Thus, the problem was to run efficiently an analysis of variance program, which used almost the entire core memory, on each rearranged column of the data array.

The solution used was to write a short program (AFORM) which selected and rearranged the data into the form required by the analysis of variance program, to use the EDIT program to control the AFORM program, and to use disk files to transfer information between programs. Thus, the problem of insufficient core-memory space for a subroutine was surmounted. The exact method is shown in Table 2, which shows the Teletype printing and commentary.

Beforehand

(1) The data array (160 x 15 numbers) was transferred from the card deck onto the disk in the directoried symbolic file named "SET03" via the card reader (CR) by monitor command ($COPY, CR, DS(SET03)). (2) The analysis of variance program (AVACE) was compiled, converted to core-image binary, and stored in the directoried core image file named "AVACE" on the disk. (3) The AFORM program, in FORTRAN IV, was placed in the directoried symbolic file named "AFORM" on the disk via the Teletype (TY) by a monitor command ($COPY, TY, DS(AFORM)).

Initially

(1) The EDIT program was used to modify the AFORM program to select the desired column of the data array. (See Table 2, Lines 3-8. Note that Line 7 is unique to a particular subset of the data.) (2) The AFORM program was then compiled (FORTRAN IV compiler) and converted to core-image binary (core-image converter). (See Table 2, Lines 10-16.) (3) Then AFORM was run to select and arrange the data column in proper order, and place the data in a temporary disk file (working symbolic) where the AVACE program could retrieve it. (See Table 2, Lines 18-21.)

Finally

With the reassignment of the disk files, the AVACE program was run, yielding accurate results. (See Table 2, Lines 23-45ff.) To do the analysis on the next data column, the procedure using the EDIT program was repeated, changing the FORMAT statement (Table 2, Line 7), i.e., recycling to *Initially* above. The total time per analysis, including typing in and printing out, approached about 5 min as the operator became familiar with the procedure and increased the typing speed.

COMMUNICATION WITH GROSS COMPUTER CENTERS

I wish that our computer centers with the large machines would be set up to read these disk cartridges so that, when we are forced to run large-scale analyses on large machines, we could simply deliver them a disk cartridge with our data on it. The usual medium for carrying data to computer centers is cards or magnetic tape, which is bulky, and the machinery is expensive. We have been using paper tape to get data to the computer center, but it is relatively slow and cumbersome. Our paper tape goes to a special system that converts the paper tape to mag tape, and then the mag tape is read into a computer which punches cards. One of the first uses that we found for the card reader on our GA 18/30 was to check the cards that a computer center punched, because we found a large number of errors. This was very important because the programs that the computer center was going to run on these cards would not have detected the errors and would have given answers tha could have had considerable distortion.

Getting data back from a large computer center can also be a problem. The usual medium is a stack of line printer output. Often we want to feed some of that output into our computer to (1) do further analysis and/or (2) plot graphs. Both of these tasks we often want to do in an interactive way that we can do better with our hands-on system. Our present solution to this problem is to have the large computer center punch cards with the same information they output to their line printer. Then, using either DBOS control commands (e.g., $COPY, CR, DS(Name 1)) or a program we have written, we can copy the information from their cards into a disk file. From the disk file, we can edit the data and transfer it into another file or use the disk file as a source for some desired program.

DOCUMENTATION

Our text books and courses about computer programming emphasize the importance of flow diagrams. Why is it so rare to find flow diagrams as part of the documentation supplied by computer companies? One answer sometimes heard from the pros: "This program was written by a genius who just sat down and wrote the program without the need of flow diagrams." Perhaps this explains some of the troubles we mortals have when we try to use those programs. Another answer seems to be: "You users don't need to know what is in the machine or in the program; please just board the bus and leave the driving up to us." The problem is that we're not always interested in riding nonstop from one end of the program to the other; often, we want to use the "general-purpose" nature of the computer and alter the programs to suit our needs. We would like to see all programs accompanied by flow diagrams with locations marked. Several levels of diagrams would be useful, so that the user can first look at a general diagram which leads him to look at the right detailed flow diagram and listing. And why not have a glossary-index with every program? Imagine the joy of being able to look up a term by alphabetically consulting the index and finding a definition of the term (how it is used in the program or machine) and references to where it is used in the flow diagrams, the program listing, and program description. It is regrettable that the orderly thinking required in programming and engineering gets burned out in the process, leaving ashes in place of flow diagrams, glossaries, indexes, and documentation, in general.

EXPERIMENTAL INTERFACING

Some of the research we're doing is a combination of what Dr. Armington and Dr. Murdock talked about earlier today. We are studying the relation of physiological to sensory and cognitive processes. The first stage of some of our analyses involves averaging analog signals. Our 12-bit analog-to-digital converter takes 40 microsec per sample. If that much accuracy is not required, time can be saved by stopping the conversion process sooner to obtain the accuracy desired. Some small computers have only a 12-bit word, so, in averaging programs which sum samples to increase signal-to-noise ratio, a 12-bit machine may overflow by the time two 12-bit samples are taken. With an 8-bit converter, only 31 samples may be taken before overflow might occur. Consequently, most of the averaging programs written for these 12-bit machines require two words of memory for each piece of data which is being sampled. With a 16-bit machine (like our 18/30), you can make an 8-bit conversion which leaves 2^8 number of times that the converted samples may be added before a one-word overflow. This is sufficient for most purposes, and a large number of core-memory locations may be saved by using one word per sample point. This is important in our work because we are sorting averaged evoked potentials into a variety of categories, which requires much space. We also can extend our capacity by using the disk storage to hold the several accumulated sums.

One of the points in Dr. Armington's presentation concerned the use of an interface between the experiment and the computer input. One of the uses of such an interface is to prevent damage to the system, to match voltages, etc. We would like to underline the usefulness of relatively simple devices that he referred to: operational amplifiers. Op amps, together with a few resistors and capacitors, can do a variety of useful operations, such as scaling signals, shifting dc levels, filtering out unwanted frequencies, inverting signals, integrating signals, converting signals into triggering pulses, etc. Learning to do these operations with op amps is about as easy as learning to program (see Swinnen, 1968, for a good introduction). One of the things we have found useful was to bring out the legs of the op amps to terminal posts so that the input and output are accessible, as Walter Kropfl did some years ago at Walter Reed. This permits ease in changing the resistors and capacitors used to control the gain, filtering, etc., of the op amps. We put a variable resistor with a dc voltage on the positive input to act as a balancing pot which adjusts the dc bias. Integrated circuit op amps are very inexpensive.

Many uses of the computer require multiple passes of the data, not only because of the limited storage capacity, but also because of the desire to run quite different programs on the same data. For this, the data must be on mag tape and the computer is not used on-line in a strict sense. Advantages of recording the data on tape are that (1) the analysis can be done at a more convenient time and (2) often the playback tape speed can be faster than real time. For example, we routinely run the analog mag tape four times faster than the speed at which it was recorded, resulting in a large savings in computer time when multiple passes of the data are required.

DATA ACQUISITION EXAMPLE

Dr. J. R. Jennings has used the system to look at the way heart rate relates to cognitive processes. His behavioral paradigm involves presenting various levels of information-processing problems to the S. He has programmed the GA 18/30 to look at the heart rate on a beat-by-beat basis. There were options as to how to program this. One option was to have a priority interrupt whenever a beat occurred. This forces the difficulties of beat detection on external equipment, and the external interface (perhaps using op amps) would have to be designed to perform that detection task. If the central processor were busy doing other things at the same time, then the priority-interrupt option would be a reasonable solution. Since the computer was not "busy," another option was used. The EKG signal was put into an analog input, and the internal programming capability was used to detect the occurrence of a heart beat. This required sampling sufficiently often to detect the beat and measure the time interval to the desired accuracy. The programmable internal timers were used to determine the interbeat interval. After a simple conversion to instantaneous rate, the measure was stored in a disk file. The information about the experimental periods came in as pulses on another analog channel, which also was sampled by multiplexing under program control the analog-to-digital converter in the same fashion. This information allowed the program to sort out the chains of heart beats into different experimental classes. All of the data was stored on the disk so that later analyses could be made by referencing the appropriate disk file.

The parts of the program that deal with the experimental connections (analog sampling, timing, etc.) were written as Assembly (machine) language subroutines. However, most of the program, including the executive routine, was written in FORTRAN, with the resultant savings on programmer time. This also makes it relatively quick and easy to change parts of the program and recompile, letting the core-image converter program do the work of mixing FORTRAN and Assembly language and building the final running program.

The present use of this system does not exhaust the speed and power of the GA 18/30 system. This leaves considerable power for uses in the future. For example, the direct memory-access channels could be used simultaneously for controlling a variety of experiments and logging data. One of the possibilities is a scope display which takes advantage of the block-chaining capability of the direct memory-access channels, which could link various subsets of the display and refresh the display while allowing the main program to continue.

Another expandable reserve is the six external priority-interrupt levels, each of which may be associated with 16 different events. These may be connected to several experiments so that the central processor need not waste time testing for relatively rare events. The high speed of the central processor, especially the hardware multiply and divide, makes possible on-line digital filtering and correlational adjustments. Some additional interesting uses of the GA 18/30 system have been made by the Acoustical and Behavioral Research Lab at Bell Labs at Murray Hill, New Jersey.

REFERENCES

SWINNEN, M. The design of biomedical instrumentation made easy through the use of operational amplifiers. Psychophysiology, 1968, 5, No. 2.

Time-shared control of a variety of psychological laboratories using the IBM 1800 data acquisition and control computer

PAUL ELLEN, C. HERBERT DeLOACHE, and JOSEPH BONDS*
Georgia State University, Atlanta, Georgia 30303

The use of digital computers is a fairly common practice within psychology departments. These devices are being used for the scoring of tests, complex statistical analyses, and similar applications. Less common, however, is the utilization of the computer for the actual conduct of the experiment. Developments in the technology of operant conditioning with its requirement of behavioral feedback (Weiss & Laties, 1965) and the real-time analysis of bioelectric data such as the averaging of evoked potentials (Uttal, 1967) have gradually focused attention on the applicability of the computer as a laboratory instrument which can be used to conduct experiments. The present paper is a report of our department's experience in implementing the concept of a centralized departmental data-acquisition facility.

*The authors wish to acknowledge the invaluable assistance of Mr. Herman Long, IBM Systems Engineer, whose skill, patience, and devoted effort brought the idea described in this paper to fruition.

The centralized data-acquisition concept is an exercise in the practice and art of sharing—sharing of common hardware, machine time, technical talent, etc. Many techniques have been worked out to make this sharing as painless as possible. In fact, it is possible in some cases to make users entirely unaware that they are sharing the resources of the system.

By indicating some of our thinking in undertaking this venture, as well as the kinds of problems encountered, we hope to provide some insights which will be of value to other groups who might contemplate a similar enterprise.

GENERAL CONSIDERATIONS

The issue of major concern was whether a centralized data-acquisition facility should be obtained for the entire department or a number of dedicated systems assigned to each departmental laboratory. It is not uncommon today for individual laboratories to have their own data-acquisition computer. These are not overly expensive and materially upgrade the quantity and complexity of the research carried on. However, these are usually dedicated to a specific task or group of tasks.

The major factor which led us to choose a centralized facility in preference to a number of dedicated systems was that of growth, not cost. The initial cost differential between a number of dedicated systems and one central system is negligible. Growth possibilities, however, with dedicated systems are generally limited, since the system itself is the basic module in the experimental environment. In contrast, the resources of a centralized data-acquisition computer, namely the operating system and the working core, are allocated among many users, and the basic modules will not be the computer (CPU) but rather the interfaces between the laboratory equipment and the computer. This allows for a relatively inexpensive incremental growth which is an important consideration in a developing program. That is, new laboratories can be added by simply increasing the number of interfaces—a relatively inexpensive action as compared to adding another CPU. Moreover, as application programs tend to be small and operating systems large, this is a very significant savings in total core requirements for a department.

Another factor often overlooked is the fact that with a centralized system, the input/output devices such as disk drives, card read/punch modules, etc., are all shared by the various users. Such devices are generally prohibitively expensive to the individual user of a dedicated system.