An advanced computer system for medical research

by WILLIAM J. SANDERS, G. BREITBARD, D. CUMMINS, R. FLEXER, K. HOLTZ, J. MILLER and G. WIEDERHOLD
ACME Project, Stanford Medical Center
Stanford, California

INTRODUCTION

The ACME project

The Stanford University School of Medicine is located on the main campus of Stanford University, in Palo Alto, California. It was moved from San Francisco to the Palo Alto campus in 1959, with the purpose of more closely integrating medical research and education with the other activities of the University. It shares, with other departments of the University, the computing facilities of the Stanford Computation Center. These facilities include an IBM 7090, a Burroughs B-5500, and a recently delivered IBM/360-67. In addition, there are currently four PDP-8, four LINC, and one LINC-8 computers in use within the medical school. Although this collection of computers represents a great deal of computing power, its distribution was such that the research needs of the medical school were not being fully met.

The Stanford Computation Center is dedicated to serving the broad needs of the University community. With the current batch-processing systems on the 7090 and B-5500 and the planned time-sharing system on the 360-67, the Computation Center is obliged to provide general-purpose computing to a large number of users with quite diverse interests. Inevitably, the needs of any special group cannot be entirely satisfied. In particular, many of the needs of the medical research program are such that a general purpose computing system is not satisfactory. Among these needs are:

1. Analog-to-digital and digital-to-analog conversion
2. Real-time data collection and analysis
3. On-line control of experiments
4. High-speed data acquisition and distribution
5. Support of satellite computers.

The small computers within the medical school provide some relief, in that they are equipped for analog-digital conversion and are being used for data collection and analysis and for control of experiments. However, being small and having few peripheral units, they are limited in the amount of data analysis they can do economically. They are also more difficult to program, since their software is generally quite primitive. In addition, not all research projects can afford the luxury of having their own computer nor the specialized manpower to program it.

In order to determine how the needs of the medical school research program could best be met, a medical school computer policy committee was formed. The committee worked actively with the medical school computer user's group, the Stanford Computer Sciences Department, and the Stanford Computation Center. The result of the committee's work was a proposal for the ACME (Advanced Computer for Medical Research) Project. Initial funding for the Project was a planning grant from the Josiah Macy Foundation, while ongoing support has been provided by a grant from the National Institute of Health, the Macy Foundation, and sharing of costs with other projects at the Stanford Medical School.

The purpose of the Project is to provide a computer system specifically designed for medical research. The ACME system is designed to act as a complement to currently existing facilities. It is assumed that a great deal of data-processing still will be done using the facilities of the Computation Center. By providing data storage and analysis facilities for the small computers, their capabilities will be greatly enhanced. In addition, by providing central signal-processing equipment, the system will support researchers that do not have their own computers.

A unique aspect of the Project is that it is a research project in support of research projects. In order to provide the medical school with the computational facilities necessary to do research, much development must be done by the project. It is planned that the ACME system will always be in a state of flux. As the more general-purpose computer systems are able to provide a service that is also being provided by ACME, that service will be dropped by ACME in favor of the other systems. At the time a need is found for a service not then available, that need will be met be extensions to the system. Thus, it is planned that the ACME system will be constantly devoted to the new and untried areas of computation, and in this way also will be a complement to existing facilities.

Within the ACME project, there is equal emphasis on hardware and software development. Hardware development is mainly concerned with interfacing the ACME system with the users and their experiments. Software development is concerned with providing non-computer oriented researchers with the tools necessary to do their work in a rapid and convenient manner. There has been a great deal of effort expended in developing a hardware/software complex where the hardware and software closely match both each other and the user and his experiment. Much effort has also been devoted to developing more suitable, and often less expensive, alternatives to manufacturer supplied hardware and software.

Both in terms of size and budget, the ACME Project is modest compared to many. But because of the fact that its goal is also relatively modest, that of providing a specialized set of services to a small and quite homogeneous set of users, it has already made a great deal of progress toward achieving that goal.

ACME system

The main purpose of the ACME system is the acquisition, analysis, storage, and retrieval of medical research data. In addition, there will be writing and debugging of programs necessary to support these activities. In the light of these requirements, it was decided that the most reasonable mode of operation for the ACME system would be time sharing; with emphasis on real-time data acquisition, and data storage and retrieval. This is an area where equipment that is currently available is very weak. Only very large specialized systems, mainly in military and space exploration environments, have achieved operational status. In order to make program writing and debugging as easy as possible it was decided that a compiler for a simple, yet relatively powerful programming language, would be written specifically for the ACME system.

It should be noted that the design criteria for the ACME system are relatively different from those of most other time sharing systems. Because of the demands of real time data acquisition, emphasis is not on giving fair and equitable service to a large group of users. Rather, it is to give high performance service to a relatively small group of users, while also answering the lighter demands of on-line program creation and debugging. Because of the demands of real-time operation, it is often better to refuse service to a user (telling him to try again later) than to offer a service that is degraded beyond usefulness to him or others using the system. Special provisions in the hardware and software have been made to reflect this philosophy.

Hardware for the ACME system

A general sketch of the ACME system is shown in Figure 1. The central processor for the ACME system is an IBM/360-50. This was chosen for several reasons. First, it is supported by a large amount of IBM-supplied software. Inasmuch as is possible, this software is used in preference to expending the effort to create similar software. Second, it is supported by a large variety of peripheral units, both mass-storage and input/output. Third, it provides a large measure of computing power and I/O versatility for a relatively low cost. Fourth, it provides upward-and-downward-compatibility with a broad line of computers, so that the system can be easily up-graded or down-graded to meet future conditions. Finally, it is highly compatible with the 360-67 at the Computation Center.

In order to provide real-time capability in a straightforward manner, it is necessary that the user program and data areas can be accessed very rapidly in a random manner. Core memory is the only device satisfying this requirement.

Estimates of the amount of memory required led to the following balance of the core memory versus number of users.

Assumptions:

(1) A control system of 7090 size.
(2) A resident compiler, library and input/output system, requiring 3 times the available memory of a 7090.
(3) User problem size distribution as on current large machines (i.e., IBM 7090's etc.) leading to an average of 14,000 words. See Figure 2.
(4) Program writing and translation will occupy the system 25% of the time.

With these assumptions:

$$\text{Memory required} = 6000 + 3 \times 26000 + n \times .75 \times 14000 + n \times .25 \times 6000$$
$$\text{or } 84000 + n \times 12000$$

This meant that a balance could be achieved at 15 users and 264000 words of memory.

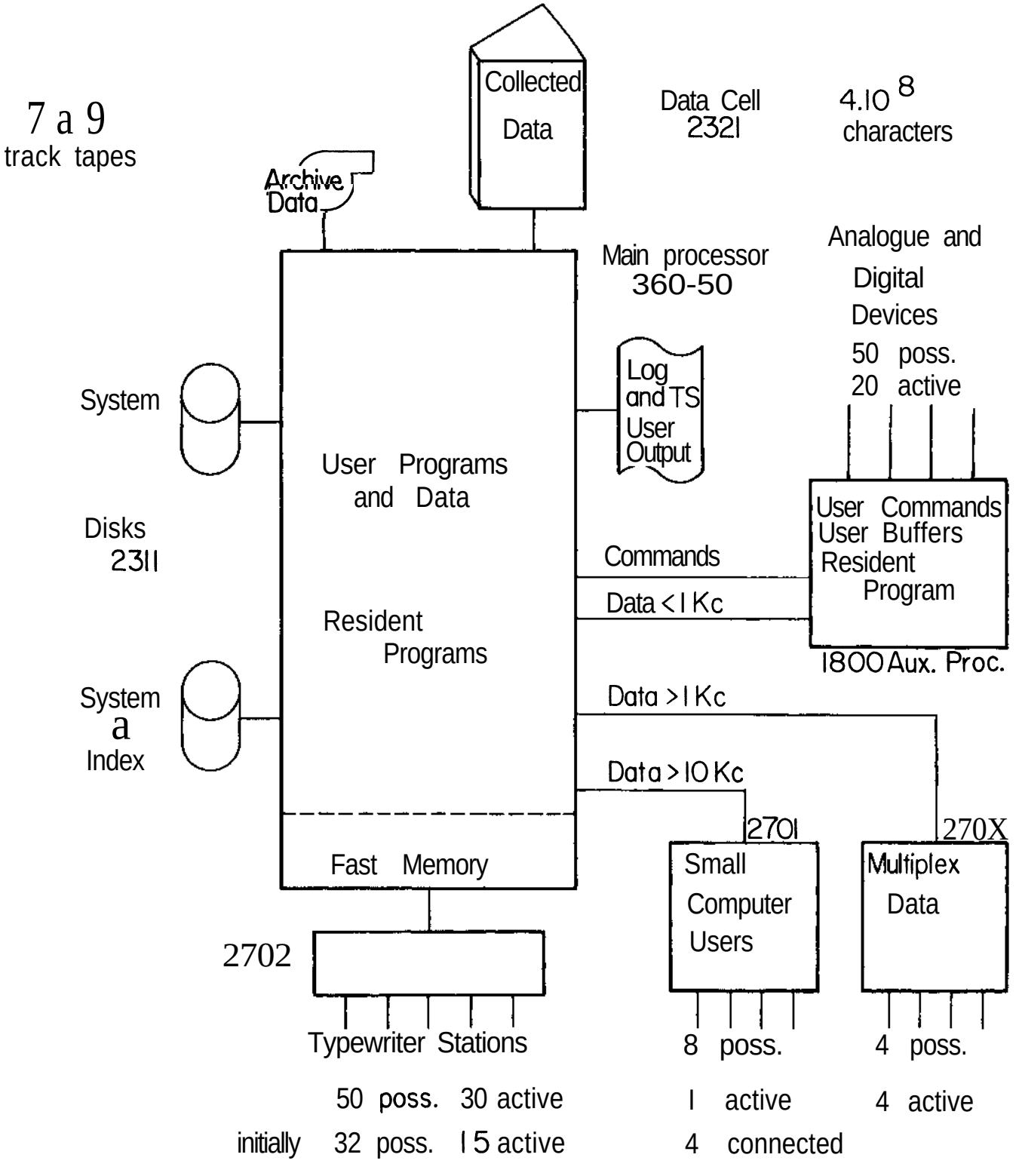
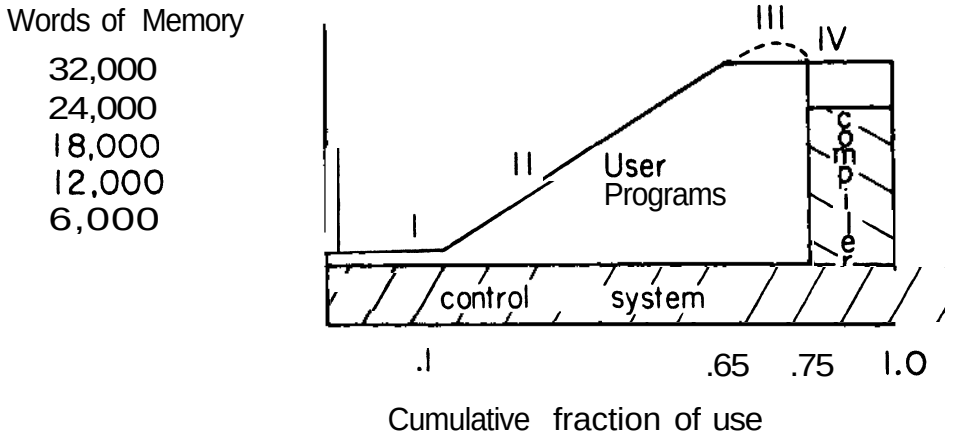


Figure 1—Sketch of ACME system



NOTES:

I. This region represents users using facilities on data stored in files, or doing desk calculator level work.

II. This region represents the normal computing workload. Much of this is used for data array storage. Declared arrays are generally only partially filled, leading to the apparent steepness of the curve.

III. Some percentage of users find that they tend to exceed the capabilities of the system and operate just below the maximum.

IV. During translation, program text and symbol tables very rarely exceed 6000 words.

Figure 2—User program size distribution

In order to get this amount of memory economically, it was decided that a minimum amount of relatively expensive fast memory would be obtained. The fast memory is used for residence of the 360 operating system and the most frequently used portions of the ACME software.

The major portion of the memory for the ACME system consists of a one million byte Large-Capacity Storage unit. This is attached to the central processor, and is addressable contiguously to the fast memory. Although its cycle time is much slower (8 μ sec vs. 2 μ sec), its cost per bit is about one-fourth of the central processor memory. Furthermore, the processor of the Model 50 can do some parts of a process without core references, and tests have indicated a factor at 2.3 performance degradation. In this way, enough memory could be obtained to keep the entire ACME system and all active user programs core-resident at all times. This is extremely important to the philosophy of providing high-performance service and is possible because of the small number of users. Because an active program is always core-resident, it is always executable, and hence can respond rapidly to real-time demands. Because all active programs are core-resident, memory allocation problems are minimized and such techniques as program relocation become unnecessary. The lack of memory swapping, paging, and concomitant problems such as special I/O buffering, greatly reduces system overhead.

Mass storage for the ACME system consists of three hierarchical levels. The lowest level is magnetic tape, which will be used for archival purposes. One 7-track unit is provided for compatibility with the 7090 and B-5500. One 9-track unit is provided, because of its higher performance and because it is necessary for generating the 360 operating system.

The next level of mass storage is the 2321 data cell. It has a storage capacity of 400 million bytes, with an average access time of 600 ms. This is the main storage device for the ACME system. On it, all user source programs and data are stored. Its capacity is large enough that it is expected that all programs and data will be permanently resident in the data cell, with no need to dump or reload from tape, except for back-up purposes. Special programming techniques have been developed to minimize the effect of its relatively slow access rate and to take advantage of its large capacity.

Another level of mass storage is the 2311 disc storage drive, of which there are only two in the ACME system. No user storage is provided here. One disc drive is used to store the non-core-resident portions of the 360 operating system. The other disc drive is used for two purposes. First, all of the ACME software is stored there, for initial loading when the system is started. The second, and most important, function of the drive is to store indices to information on the data cell. Whenever a user file on the data cell is opened, its location is determined from a catalog. An index to all records in the file is then moved from the data cell to the disc. Subsequent references to records in the file are then made using the disc-resident index.

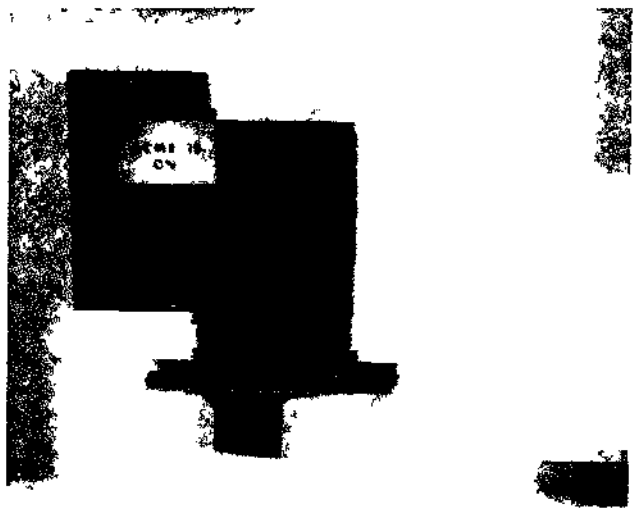


Figure 3—Terminal indicator panel

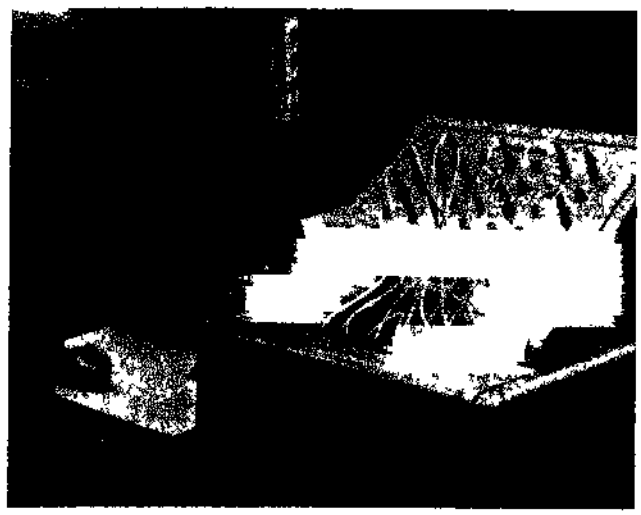


Figure 4—Switchboard

When the file is closed, the up-dated index is copied back from the disc to the data cell.

Each user of the ACME system has an IBM 2741 terminal. This is the device used to input programs, debug them, and control their execution. Since it is basically a modified Selectric typewriter, it has the advantage of being familiar and easy to use for non-computer oriented users. ACME has modified its 2741 by the addition of a 4 light indicator panel (Figure 3). Lights on the indicator panel are controlled by non-printing characters transmitted to the 2741. There is one light that reads, "ACME IS ON." It is driven by the transmission control signals to indicate that the system is operational. Another light reads, "YOU ARE ON." It is pulsed at a rate proportional to the amount of computing time the user is getting. The flicker rate thus indicates the performance of the system with respect to that user. Another light reads, "WAITING FOR YOU." It is on whenever the system is expecting input from the user. The final light reads, "SPECIAL RUN ON." It is on whenever a high demand, real-time data transmission process is active. It indicates that severely degraded terminal performance can be expected.

Most of the 2741's are connected directly by cables to the ACME system. Because of the fact that most of the 2741's are within 2000 feet of the system, no intervening cable drivers are necessary. The cables terminate in a switchboard-like patch panel (Figure 4). When a user wants to use his terminal, he telephones a computer operator, who connects his terminal to the system via the patch panel. Initially, there are 32 terminals with the possibility of 15 active at one time. Because of this low ratio, and the fact that terminal sessions are expected to be lengthy, manual switching is feasible at a great reduction in cost over other modes of operation. Communication with the 2741's is processed by a 2702 communication multiplexor.

The need for other than typewriter output devices has been obvious for a long time. But the cost of commercial devices in a medical environment is such that they are scarcely defensible in comparison to other medical aids.

However, two Sanders Associates character-oriented CRT displays have been ordered as experimental adjuncts to the ACME system. These will be used for development in areas of text editing and information retrieval. Because of the high data rate necessary to support these devices, they will be connected to the ACME system via a 2701 with a parallel data adapter. The interface to the 2701 is being supplied by Sanders Associates.

A special CRT display has been designed and built by the ACME Project for the input/output of graphical data (Figure 5). It is driven by vector-drawing logic and a core refresh memory. This unit is capable of displaying 2046 vectors simultaneously. It was built from integrated circuits, and the component cost, including the memory, was about $8,500. It will be connected to the ACME system via another parallel data adapter on the 2701.

A need still exists for a silent hard copy device for developing data distribution to hospital wards.

For the processing of user analog and slow (less than 1 Kc; i.e., 1000 samples per second) digital data an IBM 1800 computer is used. The 1800 is a small process control computer with a large complement of signal processing attachments. It will be used to do analog-to-digital and digital-to-analog conversions, as well as some primitive signal processing such as smooth-

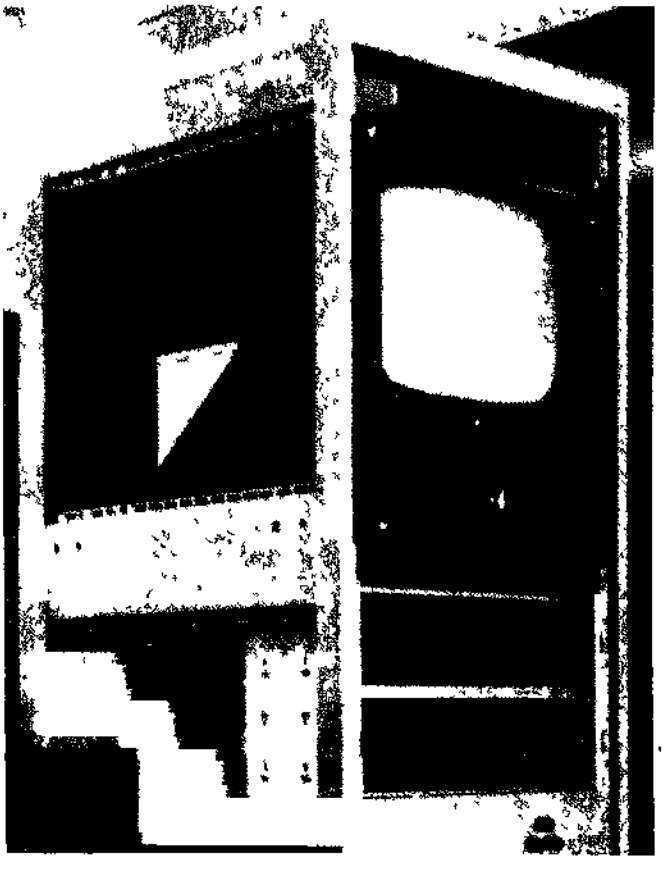


Figure 5—Prototype of graphical display

ing and validity checking. It is connected to the central processor via a high speed data channel so that it behaves essentially like an input/output device as far as the system is concerned. When relatively low bandwidth analog signals are transmitted, and no more than 8 bits of accuracy is needed, analog-digital conversion is done on the 1800 For more demanding signal processing, the analog-digital conversion has to be done in the laboratory and digital data is transmitted to the 1800. There are currently 32 analog-to-digital inputs, 8 digital-to-analog converters, 20 digital inputs, 12 digital outputs, and 80 process interrupt lines on the 1800. The timing of the data acquisition and distribution is user controlled through the process interrupt lines so that no synchronization of multiple experiment data rates is required. All analog and digital transmission equipment was designed and built by the ACME Project. One of the most interesting pieces of analog transmission equipment is an FM analog transmitter and receiver that is magnetically and acoustically coupled to an ordinary telephone. This will be used to process analog signals over distance or where no cables have been pulled.

For transmitting digital data in the speed range between 1 and 10 kc, a special multiplexor has been designed and constructed by IBM for the ACME Project. For lack of a better name, it is called the 270X. It is capable of multiplexing 8-bit digital transmission over cables between the ACME system and up to 16 remote units. Each remote unit is called a 270Y. There are currently four 270Y's. The 270Y's are designed for laboratory use, and are rack mountable. Each has binding posts for 8-bits of input and output, and for timing and control signals. Each has a push-button to terminate a transmission, and a signal line to cause a central processor attention interrupt. In addition, each 270Y has a built-in variable frequency oscillator to control sampling rate, or the sampling rate may be controlled via an external line. In addition to the local sampling-rate oscillator in each 270Y, there is a program-selectable oscillator in the 270X whose rate is adjusted to drive a digital incremental plotter. Thus, for example, a typical use of the 270X-270Y might be as follows. The inputs would be connected, via a suitable analog-to-digital converter, to an instrument such as a gas chromatograph. The sampling rate would be determined by the setting of the local oscillator on the 270Y. The outputs would be directly connected to an incremental plotter. A trial would be run, the data analyzed by a program that the user has written, and the results plotted immediately at the experimenters station. It takes little imagination to see the potential for this mode of operation.

It is expected that data rates in excess of 10 kc will be generated only by the CRT displays and the small computers. These will be connected to a 2701 with four parallel data adapters. Each parallel data adapter has a data path 16 bits wide, and is capable of sustaining speeds in excess of any demand now foreseen. Because of this high data rate, the 2701 has been connected to a central processor selector channel, which can sustain a high data rate with lower interference to computation. But since the selector channel can control only one transmission at a time, there must be a limit to the amount of time each transmission takes. Thus, it was decided that the normal mode for high data rate transmission (i e., over 10 kc) will be short, very high speed bursts of data (i.e., 20 kc or higher). To connect the small computers to the 2701, a special interface was designed. The interface transmits data in parallel over cables on a demand/response basis. Provision has been made for remote computers to gain the attention of the central processor via an attention interrupt. In addition, a 25 ms "dead-man" timer in the interface limits the time any single burst of data transmission can take.

The remaining hardware consists of unit record equipment on the 360 and 1800. This will be used

mainly for batch-mode operation during software development and checkout. In addition, during the early stages of time-shared operation the 360 printer is being used to keep a log of all transmissions to and from terminals. This should greatly facilitate the detection of problems that develop in using the system, both from the point of view of user difficulties and the detection of bugs in the software.

Software for the ACME system

The software for the ACME system will be divided into the following categories for purposes of discussion:

1. The 360 Operating System
2. The ACME compiler
3. Resource allocation (including time-slicing)
4. Data-file management
5. Terminal input/output
6. Real-time input/output
7. Library subroutines

One of the early, and significant, decisions in the design of the ACME system was the decision to use manufacturer supplied software whenever this was feasible. Thus, the entire ACME software system was designed to run as a single job under Operating System/360. During the operation of the ACME system, OS/360 provides such services as low-level input/output management and memory allocation, dynamic subroutine loading, and interval timing. This mode of operation has two significant advantages. First, a great deal of highly specialized programming can be avoided. Second, most of the remaining programs are machine independent, and hence can be written in a machine-independent manner. In fact, most of the ACME software has been written in FORTRAN. It was found that the IBM H-level FORTRAN compiler is capable of producing very efficient code and thus there was virtually no advantage in not using this machine-independent language. The few routines that were written at an assembly language level were mostly for such machine-dependent operations as character and bit-manipulation and for communication with the Operating System and for machine code skeletons.

Almost no modifications were made to OS/360. The few exceptions were in areas where the operating system provides for user-supplied modifications and these modifications were relatively straight forward Hence, almost anyone with a similarly configured 360 should be able to use the ACME software with a minimum of effort. In fact, with the multi-programming versions of OS/360, the ACME software is useable concurrently with other modes of operation.

The ACME compiler will be discussed fully in a future paper, and hence will be only briefly discussed here. The compiler is for a subset of PL/1 that includes many of the most useful features of the language. It is incremental; that is, it compiles one statement at a time and a program is always capable of execution. It compiles all the way to machine language, and thus produces relatively efficiently executing code. All system-user communication is processed by the compiler, hence the system command language is a subset of the compiler language, with identical syntax. Also included in the compiler language are text-editing functions for modification of program texts. For such processes as input/output, subscript range checking, and the computation of mathematical functions such as SIN, COS, and standard statistical procedures, the compiler generates calls to a resident library of subroutines. Some of these subroutines are also written in FORTRAN, and in fact use the IBM-supplied FORTRAN subroutine library for such things as input/output formatting.

Resource allocation consists mainly of memory allocation and time-sharing. Memory is allocated to users in 4048 byte (1024 word) quantities called pages. When a user has logged in he is assigned one such page in which his file names, etc., are kept. For programming two more pages will be assigned for program and data storage. More pages are assigned as they are needed during the compilation of his program. Memory for object-program arrays is not allocated until each array is referenced during execution. Hence there is some saving in memory during the time a user is creating his program. It is not necessary that the pages for a user program be contiguous in memory. Hence the problems of memory allocation are greatly simplified.

Because most of the ACME software was compiled under H-level FORTRAN, which is not capable of producing re-entrant code, allocation of time to users is not done in the usual manner. Instead of switching from one user to another at the end of some arbitrary time interval, switching is done only at so-called "re-entrant points." A re-entrant point is defined as a point at which:

1. All required information concerning the current user is located in storage peculiar to that user.
2. The next operation to be executed on behalf of the user is a call to a subroutine which will not return to the calling routine.

These two conditions are sufficient to insure a somewhat limited, but nevertheless adequate form of re-entrant programming. The scheme relies on the fact that the H-level FORTRAN compiler generates a prologue which is always executed upon entry to a subroutine. This prologue initializes information internal

to the subroutine, but peculiar to the current invocation of the subroutine.

The necessity of limiting switching to only certain points leads to the interesting situation in which the current user graciously yields his control of the machine, rather than having it wrested from him. However, proper etiquette in this regard is assured in several ways.

1. When a user is compiling a program, the structure of the compiler assures a reasonable discipline for user switching.
2. All requests for input/output activity require a yield.
3. A check is built into the code generated by the compiler at each GO TO and END statement for the end of a time interval. A yield will result if the time interval has elapsed.

In the process of yielding, a "resume routine" is indicated. This routine is entered when the user is next given control. Yields that accompany input/output requests generally specify a wait until the input activity is completed, due to the interactive nature of most system use. Users currently are served in strict order, on a round-robin basis. There is some probability that a priority scheme may be introduced later, if experience indicates that high data-rate experiments cannot be served with the round-robin scheme.

Because one of the major purposes of the ACME system is information storage and retrieval, special data management procedures have been designed to facilitate these operations. User data, and programs, are stored in data sets on the data cell. Data sets are ordinarily catalogued by user name, then by a project name, and finally by a user assigned data set name. When a data set name is mentioned in a program, it is automatically qualified by the user name and project name supplied at log-on time. This automatic qualification is overridden by explicit qualification of the form USERNAME.PROJECTNAME.DATASETNAME. This method of data set naming is essentially identical to that used by OS/360, although OS/360 cataloguing procedures are not used. In addition, certain public data sets will be available. Among these will be a set of standard user-oriented programs.

Within the data sets, data is stored in the form of records. Records may be stored sequentially or randomly and may be retrieved sequentially or randomly. Records may be of arbitrary length. Data items are stored and retrieved from records by name. Hence, data items may be retrieved in an order different from the one in which they were stored. Moreover, fewer data items may be retrieved than are stored in a given record. This mode of operation is unlike that used in most current programming systems, where item order, not name, is significant. It is felt that the mode chosen, although somewhat less flexible, is much more in line with the thinking of the non-computer oriented personnel who will be the prime users of the ACME system.

The terminal input/output procedures used by the ACME system were also designed by the ACME Project. Output from the system is generally in the form of a message and a prompt. The message portion is the result of the last operation performed and the prompt portion is used to indicate what should be done next. A question mark (?) is used to indicate the end of the output. For example, during the log-on procedure the output from the system is the prompt NAME? The user then supplies his name. If the name is not acceptable, the system types an error message and re-prompts NAME? During compilation, the system prompts the line number of the next line when compilation of the preceding line is successful. If the compilation of the preceding line is not successful, the compiler supplies an error message and re-prompts the previous line number. Messages are generally quite long, so that a maximum amount of information can be conveyed. If a user recognizes a message and does not want to see it again in its entirety, he pushes the ATTENTION button on his terminal, which causes an ellipsis (. . .) to be typed and the rest of the message to be skipped.

The prompt portion of a message can usually not be ignored because it remains in a buffer as part of the user's next input. All prompts are recognizable to the compiler as commands, with the question mark treated as a blank. Hence, if PROJECT? is prompted and the user types DOGWEIGHTS, the input to the system is *PROJECT* DOGWEIGHTS. In fact, the system forgets what it has typed out; it only looks at the next input line to decide what to do next. If the user wishes to ignore a prompt, he may back-space over it, which causes it to be deleted, or he may push the ATTENTION key which causes the current input line to be ignored and a prompt of ? to be typed. Similarly, in the case of some syntax errors, the compiler merely prompts a corrected version of the statement. If the user wants to use the corrected version, he merely types carriage return, which causes the prompt to be used as input.

Real time input/output was also designed to maximize user convenience. With the exception of the 270X, which is still in the developmental stage, all real-time input/output is generated by satellite computers, either the ACME 1800 or the remote small computers. A communication protocol has been established for all transmission between computers. The protocol is designed

mainly for batch-mode operation during software development and checkout. In addition, during the early stages of time-shared operation the 360 printer is being used to keep a log of all transmissions to and from terminals. This should greatly facilitate the detection of problems that develop in using the system, both from the point of view of user difficulties and the detection of bugs in the software.

Software for the ACME system

The software for the ACME system will be divided into the following categories for purposes of discussion:

1. The 360 Operating System
2. The ACME compiler
3. Resource allocation (including time-slicing)
4. Data-file management
5. Terminal input/output
6. Real-time input/output
7. Library subroutines

One of the early, and significant, decisions in the design of the ACME system was the decision to use manufacturer supplied software whenever this was feasible. Thus, the entire ACME software system was designed to run as a single job under Operating System/360. During the operation of the ACME system, OS/360 provides such services as low-level input/output management and memory allocation, dynamic subroutine loading, and interval timing. This mode of operation has two significant advantages. First, a great deal of highly specialized programming can be avoided. Second, most of the remaining programs are machine independent, and hence can be written in a machine-independent manner. In fact, most of the ACME software has been written in FORTRAN. It was found that the IBM H-level FORTRAN compiler is capable of producing very efficient code and thus there was virtually no advantage in not using this machine-independent language. The few routines that were written at an assembly language level were mostly for such machine-dependent operations as character and bit-manipulation and for communication with the Operating System and for machine code skeletons.

Almost no modifications were made to OS/360. The few exceptions were in areas where the operating system provides for user-supplied modifications and these modifications were relatively straight forward Hence, almost anyone with a similarly configured 360 should be able to use the ACME software with a minimum of effort. In fact, with the multi-programming versions of OS/360, the ACME software is useable concurrently with other modes of operation.

The ACME compiler will be discussed fully in a future paper, and hence will be only briefly discussed here. The compiler is for a subset of PL/1 that includes many of the most useful features of the language. It is incremental; that is, it compiles one statement at a time and a program is always capable of execution. It compiles all the way to machine language, and thus produces relatively efficiently executing code. All system-user communication is processed by the compiler, hence the system command language is a subset of the compiler language, with identical syntax. Also included in the compiler language are text-editing functions for modification of program texts. For such processes as input/output, subscript range checking, and the computation of mathematical functions such as SIN, COS, and standard statistical procedures, the compiler generates calls to a resident library of subroutines. Some of these subroutines are also written in FORTRAN, and in fact use the IBM-supplied FORTRAN subroutine library for such things as input/output formatting.

Resource allocation consists mainly of memory allocation and time-sharing. Memory is allocated to users in 4048 byte (1024 word) quantities called pages. When a user has logged in he is assigned one such page in which his file names, etc., are kept. For programming two more pages will be assigned for program and data storage. More pages are assigned as they are needed during the compilation of his program. Memory for object-program arrays is not allocated until each array is referenced during execution. Hence there is some saving in memory during the time a user is creating his program. It is not necessary that the pages for a user program be contiguous in memory. Hence the problems of memory allocation are greatly simplified.

Because most of the ACME software was compiled under H-level FORTRAN, which is not capable of producing re-entrant code, allocation of time to users is not done in the usual manner. Instead of switching from one user to another at the end of some arbitrary time interval, switching is done only at so-called "re-entrant points." A re-entrant point is defined as a point at which:

1. All required information concerning the current user is located in storage peculiar to that user.
2. The next operation to be executed on behalf of the user is a call to a subroutine which will not return to the calling routine.

These two conditions are sufficient to insure a somewhat limited, but nevertheless adequate form of re-entrant programming. The scheme relies on the fact that the H-level FORTRAN compiler generates a prologue which is always executed upon entry to a subroutine. This prologue initializes information internal

to the subroutine, but peculiar to the current invocation of the subroutine.

The necessity of limiting switching to only certain points leads to the interesting situation in which the current user graciously yields his control of the machine, rather than having it wrested from him. However, proper etiquette in this regard is assured in several ways.

1. When a user is compiling a program, the structure of the compiler assures a reasonable discipline for user switching.
2. All requests for input/output activity require a yield.
3. A check is built into the code generated by the compiler at each GO TO and END statement for the end of a time interval. A yield will result if the time interval has elapsed.

In the process of yielding, a "resume routine" is indicated. This routine is entered when the user is next given control. Yields that accompany input/output requests generally specify a wait until the input activity is completed, due to the interactive nature of most system use. Users currently are served in strict order, on a round-robin basis. There is some probability that a priority scheme may be introduced later, if experience indicates that high data-rate experiments cannot be served with the round-robin scheme.

Because one of the major purposes of the ACME system is information storage and retrieval, special data management procedures have teen designed to facilitate these operations. User data, and programs, are stored in data sets on the data cell. Data sets are ordinarily catalogued by user name, then by a project name, and finally by a user assigned data set name. When a data set name is mentioned in a program, it is automatically qualified by the user name and project name supplied at log-on time. This automatic qualification is overridden by explicit qualification of the form USERNAME.PROJECTNAME.DATASETNAME. This method of data set naming is essentially identical to that used by OS/360, although OS/360 cataloguing procedures are not used. In addition, certain public data sets will be available. Among these will be a set of standard user-oriented programs.

Within the data sets, data is stored in the form of records. Records may be stored sequentially or randomly and may be retrieved sequentially or randomly. Records may be of arbitrary length. Data items are stored and retrieved from records by name. Hence, data items may be retrieved in an order different from the one in which they were stored. Moreover, fewer data items may be retrieved than are stored in a given record. This mode of operation is unlike that used in most current programming systems, where item order, not name, is significant. It is felt that the mode chosen, although somewhat less flexible, is much more in line with the thinking of the non-computer oriented personnel who will be the prime users of the ACME system.

The terminal input/output procedures used by the ACME system were also designed by the ACME Project. Output from the system is generally in the form of a message and a prompt. The message portion is the result of the last operation performed and the prompt portion is used to indicate what should be done next. A question mark (?) is used to indicate the end of the output. For example, during the log-on procedure the output from the system is the prompt NAME? The user then supplies his name. If the name is not acceptable, the system types an error message and re-prompts NAME? During compilation, the system prompts the line number of the next line when compilation of the preceding line is successful. If the compilation of the preceding line is not successful, the compiler supplies an error message and re-prompts the previous line number. Messages are generally quite long, so that a maximum amount of information can be conveyed. If a user recognizes a message and does not want to see it again in its entirety, he pushes the ATTENTION button on his terminal, which causes an ellipsis (. . .) to be typed and the rest of the message to be skipped.

The prompt portion of a message can usually not be ignored because it remains in a buffer as part of the user's next input. All prompts are recognizable to the compiler as commands, with the question mark treated as a blank. Hence, if PROJECT? is prompted and the user types DOGWEIGHTS, the input to the system is *PROJECT* DOGWEIGHTS. In fact, the system forgets what it has typed out; it only looks at the next input line to decide what to do next. If the user wishes to ignore a prompt, he may back-space over it, which causes it to be deleted, or he may push the ATTENTION key which causes the current input line to be ignored and a prompt of ? to be typed. Similarly, in the case of some syntax errors, the compiler merely prompts a corrected version of the statement. If the user wants to use the corrected version, he merely types carriage return, which causes the prompt to be used as input.

Real time input/output was also designed to maximize user convenience. With the exception of the 270X, which is still in the developmental stage, all real-time input/output is generated by satellite computers, either the ACME 1800 or the remote small computers. A communication protocol has been established for all transmission between computers. The protocol is designed

around the concept of a conversation between the main computer and a satellite computer that can be initiated by either party. Once initiated, the conversation continues until there is no more data to be transferred in either direction. Because some satellite computers, such as the 1800, may be processing several different sets of input/output concurrently, provisions have been made for several concurrent conversations between the same pair of computers.

As far as the user is concerned, real-time input/output is programmed in exactly the same way as data-file input/output. Each real-time input/output path is treated as a data set by the ACME system. The data sets are catalogued in the same way as data-file data sets and accessed by the same set of commands. Data set attributes, catalogued with each data set, allow the software to distinguish between real-time and data-file sets. Conversion is automatically done by the ACME software to provide compatability between satellite computer data formats and ACME system data format.

The subroutine library consists of commonly used subroutines for such operations as statistical analysis, graph plotting, etc. Although some subroutines may be written in the ACME compiler language, the majority will probabily remain in FORTRAN or assembly language, due to the higher object program efficiency that results. The most commonly used routines will always be core-resident, with direct linkage provided by the ACME compiler. These are not re-enterable, so their execution time must be small. Less frequently used subroutines, or subroutines with long execution time, will be dynamically loaded and assigned to individual users. When these routines are no longer needed, the memory they occupy will be released and made available to other users.

CONCLUSION

As of July, 1967, all of the hardware described in this paper is operational, with the exception of the Sanders Associates displays which have not been delivered yet. Connection has been established between the 1800 and two research laboratories. One small computer has been connected to the system. The ACME compiler is almost complete. Timing tests have indicated that its object code efficiency compares favorably with that of similar compilers. The resource allocation software is complete and allows time-shared use of the system. The terminal input/output routines also have been completed. The data-file management routines allow only program storage, due to a delay in IBM software support for the data cell. The real-time input/output routines are in an advanced stage of development, and currently allow primitive real-time input/output. The subroutine library is partially complete, with work continuing to expand its scope.

It is hoped that this paper will provide encouragement to those who believe that a successful time-sharing system is possible and is realizable by an organization with limited resources.

At a time when many highly-touted time-sharing systems are proving to be less than successful, the ACME Project is quite proud of its accomplishments. If there is any lesson to be learned, it is that a small group, with specific and well defined goals and a highly cooperative user community, can quickly and effectively provide a needed service to that community.

ACKNOWLEDGMENTS

The planning work was sponsored by a Josiah Macy Foundation planning grant and further funding has been granted from NIH (Grant No. FR-00311) and Macy Foundation. Much credit for these ideas and procedures goes to other computer installations and other people, notably project MAC at M.I.T., MEDLAB at the Latter-Day Saints Hospital in Salt Lake City, the Computer Center and ARPA project at the University of California, Berkeley; University of California San Francisco Medical School; U.C.L.A. Health Sciences, etc., and of course the Computation Center and the Computer Science Department of Stanford itself.

REFERENCES

1 H D HUSKEY W WATTENBURG
A basic compiler for arithmetic expressions
Communication of the ACM January 1961

2 W KEESE H D HUSKEY
An algorithm for the translation of Algol statements
Proceedings of the IFIP Congress 1962

3 F J CORBATO et al
The compatible time-sharing system
MIT Press 1963

4 G WIEDERHOLD
A proposal for a simple system allowing direct access to the computer
Internal Paper Berkeley Computation Center 15 May 1963

5 J H SALTZER
TYPSET and RUNOFF
Memorandum editor, MAC Memo 193-2 11 January 1965

6 G WIEDERHOLD
Student, a fast FORTRAN IV compiler
Internal documentation Berkeley Computation Center 1965

7 J SHARP C GRAM B PANZL
Student language manual, The language and the processor
Department of Electrical Eng and Computer Science Berkeley 1 March 1966

8 H BERG et al
Report of the SHARE Advanced Language Development Committee 1 March 1964

9 A J SCHERR
Time sharing measurement
Datamation February 1966

10 G WIEDERHOLD
A summary of the ACME system
Proceedings of the ONR Computer and Psychobiology Conference Monterey 17 May 1966

11 G Y BREITBARD et al
ACME notes
Internal documentation Stanford Computation Center ACME Facility November 1965 to present

12 V WIEDERHOLD
How to use PL/ACME
Document No 80-50-00 Stanford Computation Center 15 July 1967