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.

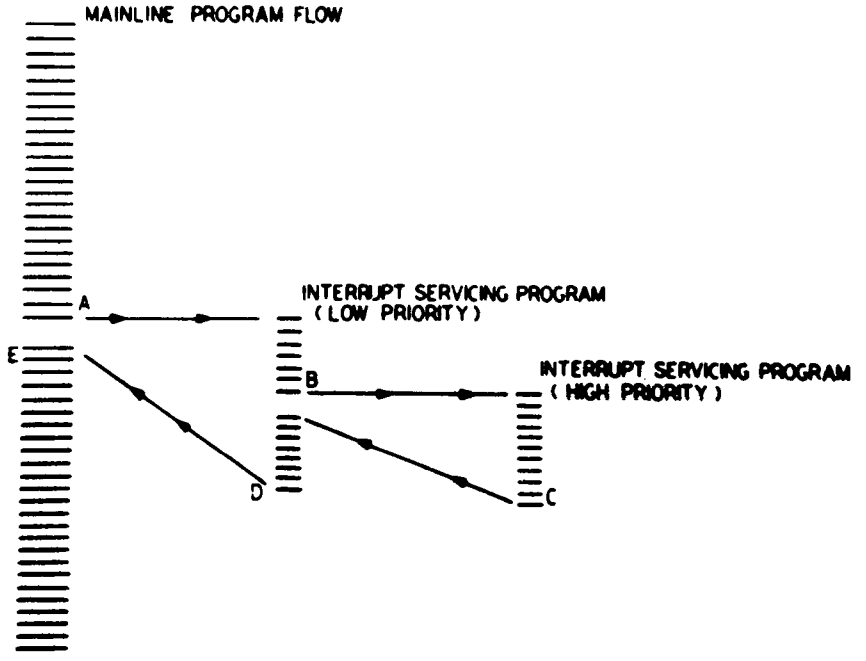

Fig. 1. Order of interrupt servicing. A high-priority event interrupts the servicing of a low-priority interrupt, and, at the completion of the servicing of the high-priority interrupt, control is returned to the interrupted routine. The completion of the latter returns control to whatever program was first interrupted.

REQUIREMENTS OF A CENTRALIZED DATA ACQUISITION SYSTEM: TIME SHARING

Effective utilization of a centralized data-acquisition and control facility requires that the various elements of the system should be simultaneously available for use in real time by a number of different users. In our case, the various departmental laboratories needed to have access to the computer capability on demand without loss of data or lack of control of the operations within each laboratory. This meant that not only must the computer operate in real time, but, since there were many users requiring this kind of service, there was also the requirement of time-sharing capability.

A data acquisition and control computer (DAC) is primarily designed to deal with discrete events taking place asynchronously and totally outside of itself. This is not to imply that a DAC computer cannot also function as a data processor, but only to point out that its main function is the acquisition of data and the control of external events in real time. Before the time-sharing concept was developed, much valuable computer time was simply devoted to a single application. The specific application may have been some real-time control function such as monitoring an EEG or a batch data-processing job such as as a statistical analysis, but not both at the same time.

This type of serial usage failed in two ways—it did not use all the computer's time and allowed for no temporal flexibility. In the case of batch jobs, the data were processed in the order in which they were received. In the case of real-time process control functions, the computer was unable to do anything besides wait for the next event, thus wasting the majority of its time.

Time sharing, then, was developed to enable the computer to service various jobs on an asynchronous basis. In other words, instead of waiting for something to happen in the real world, if the computer were, for example, monitoring a process, it could now be doing some unrelated task such as a data analysis or even monitoring some other process, thereby utilizing its time more efficiently.

TIME SHARING IN A DATA ACQUISITION AND CONTROL SYSTEM

The term "data acquisition and control" implies that the computer has some means whereby it can dynamically interact with its environment. One way to accomplish this is to have the computer continually scanning the external world under program control to determine if an event has occurred that requires servicing. This is an inefficient utilization of computer time and is awkward to program, since it requires program-monitoring loops which must be executed frequently. In addition, if other tasks were to be executed in a time-sharing mode, it would be necessary to continually return control from these programs to the program which is monitoring the environment.

In view of these problems, most computers have a hardware feature (IBM, 1966) called an interrupt system. This is nothing more or less than a set of registers in which the bits can be turned on by events occurring in the real world. The hardware is constructed in such a manner that if any bit is on, a hardware branch is made to a location in core which has a subroutine available for servicing that particular bit. It will be noted that the bit immediately identifies the locus of action in the real world.

There is the possibility that an interrupt can occur while the computer is servicing a previous interrupt or even two or more interrupts may occur at the same time. To resolve these conflicting demands on the computer capability, a priority structure is imposed on the interrupt system by assigning the various interrupt terminal points a priority level which determines the order of servicing. An interrupt occurring at a high level of priority will cause servicing of an interrupt at a lower level to be deferred until servicing of the higher level is completed. This is illustrated in Fig. 1.

Events or processes or experiments in the real world are assigned different priority levels, depending on the minimum computer response time that is required. For example, the reading of analogue EEG data would be assigned a higher priority than the sensing of a contact closure in an operant conditioning chamber. A ±10-msec delay in the operation of a feeder device or the sensing of a barpress is not significant, whereas the same error in reading brain electrical waves could introduce major difficulties.

In short, individual process in the real world are hard-wired to discrete hardware terminal points which, when turned on, can call specific servicing programs. In this way, many processes can be handled either in conjunction with or independent of each other, with the nature of the hardware interfacing to the processes in the real world providing the mechanism of sharing the resources of the computer. To add another process or another experiment, all one needs to do is wire it to an interrupt terminal point and

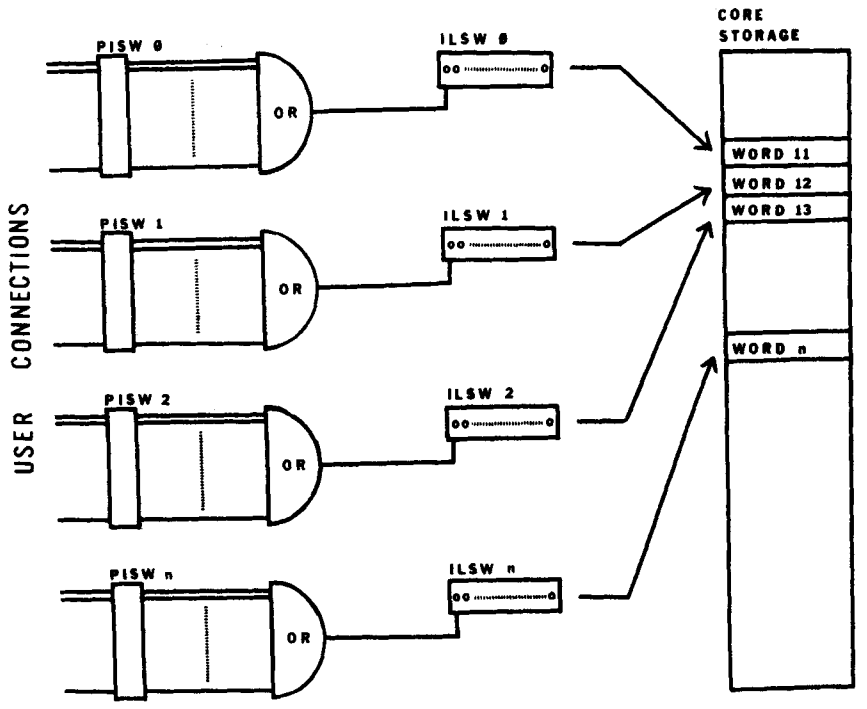

Fig. 2. Hardware interfacing of laboratory process to the 1800 computer with priority level established by ILSW.

add the necessary servicing subroutines to the contents of core.

THE GEORGIA STATE UNIVERSITY DATA ACQUISITION AND CONTROL SYSTEM

The foregoing has described the hardware mechanism by which the IBM 1800 data-acquisition system provides time sharing among the various departmental laboratories at Georgia State University. However, to complete the description of our application and to understand how the various laboratories are serviced, we need to address ourselves to the operating system.

Software

The system which was provided with the 1800 at GSU is known as TSX or Time-Sharing Executive (IBM, 1970). This system is provided by the manufacturer to facilitate the generation and implementation of user-written programs. Within TSX is a system director program in which two functions are crucial to our application. The first of these, and perhaps most important is the Master Interrupt Control, or MIC, program. The primary function of this program is to provide the linkage between process interrupt points and the user-designated interrupt servicing programs. Every interrupt causes MIC to become active. MIC "intercepts" each interrupt and determines if a user subroutine has been designated by the system programmer to deal with it. If so, the designated routine is called. The routine passes control back to MIC upon its completion. MIC then resets the interrupt bit and passes control back to the interrupted program. The manner in which this is accomplished is illustrated in the next two figures.

Figure 2 shows the previously referred-to hardware linkages between the real world and core. The users' laboratory equipment is interfaced by connection to a bit in a PISW (Process Interrupt Status Word) register. Any bit that is on in a PISW register turns on bit 0 in an ILSW (Interrupt Level Status Word) register. The particular ILSW activated determines the priority level of that interrupt. Associated with each ILSW, i.e., each priority level, is a fixed location in core. At this location is a word that contains the address of a routine that services all interrupts on that level.

Figure 3 shows how MIC transfers control to the appropriate user routine which services the interrupt. When an interrupt occurs on ILSW, the hardware picks up the address contained in, for example, Word 12 of core and branches to that address which is in the MIC program. On this ILSW, there may also be, in addition to the PISW, I/O devices such as the disk, printer, card read/punch, and so forth. MIC reads the ILSW to determine whether the interrupt came from one of these I/O devices or a laboratory process. If the former, MIC branches to an I/O device routine which is part of the system. If the latter, MIC reads the PISW and branches to the user program associated with the PISW bit that had been turned on. As indicated above, the user routine passes control back to MIC upon its completion. The interrupt is reset and control is restored to whatever program was interrupted, i.e., nonprocess or process program.

The second function of the system director is the Interval Timer Control (ITC). The 1800 has three hardware timers which may be set under program control to cause an interrupt at a later time. When the timer interrupt occurs, ITC calls a subroutine designated by the user. Two of the hardware timers (A and B) are millisecond clocks. The third timer

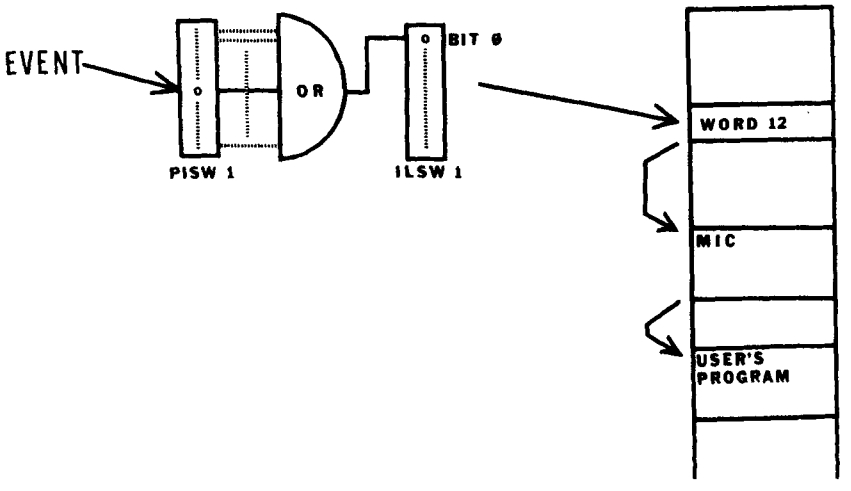

Fig. 3. Master interrupt control program transferring control to user-written subroutine which services a process interrupt.

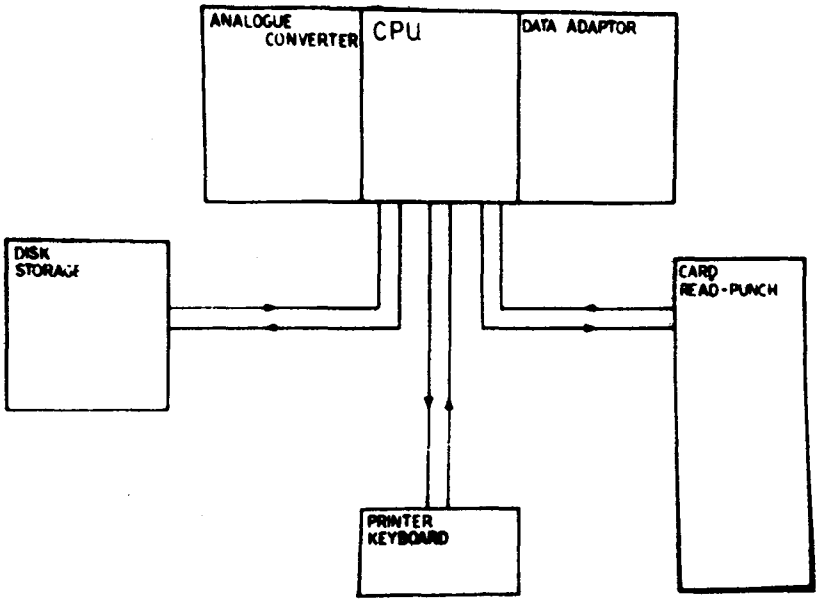

Fig. 4. Floor plan of the data-acquisition system showing the various I/O modules associated with the CPU.

(Timer C) is also a millisecond clock. However, working in conjunction with this timer are nine auxilliary software timers which have ½-sec time bases. These software timers are nothing more than words in core that are incremented by ITC after x number of milliseconds have elapsed on Timer C. The value and the actual time base of each timer is determined by the user when the system is generated. ITC checks the various programmed timers and transfers control to appropriate servicing routines when the programmed timers have timed out.

When a user wishes to activate a subprogram at some time in the future, he passes the name of the subprogram, the number of the timer (hardware or software) to be used, and the length of time to ITC. ITC returns control to the calling program almost immediately. When the timer expires, the designated subprogram is executed at the highest priority level.

Finally, one last software concept needs to be developed before we can begin to describe in detail the application at GSU. In addition to the system director, there is a group of programs in TSX which falls under the heading of the nonprocess monitor. Under this heading are programs such as the assembler, computer startup routines, FORTRAN compiler, and, in particular, a program called the core load builder. This latter program is fundamental to the system because it is the core load builder that takes an assembled user program and all associated programs, groups them together, gives them a name, and stores this group in a format on the disk such that when the machine is started up by the user, these programs can be brought into the core as a group by their name and immediately used.

Laboratories

Figure 4 shows the CPU and various I/O devices that are utilized. The CPU contains 16,000 words of core, of which 5,500 words are taken by the TSX system.

The laboratories are interfaced to the computer through assignment to a given hardware priority level. Those laboratories having the highest priority will have interrupts originating from them taking precedence over interrupts originating from lower-priority laboratories.

Figure 5 shows the kind of laboratories which have been interfaced with the system. In each laboratory, an in-house-built control box was developed. This box was interfaced with the CPU and provides the E in each laboratory with a means of initiating and controlling his experiments. He signals his intent to start an experiment by operating a switch which interrupts the computer. The computer, in turn, tells him, by means of turning on lights on his console, to enter whatever identifying information is called for in his programs. When this is completed, control is switched to the CPU, which begins to monitor and control the various devices in his laboratory. These devices may consist of five operant conditioning chambers, GSR conditioning and recording apparatus, an EEG, or an assortment of pinball machines and toy games in the playroom. The interrupt servicing programs for the various laboratories were previously written in the assembly language and stored on disk.

Prior to our recent expansion, the system operated in the following manner: Each day, the first laboratory to begin operation would start up the computer and load the appropriate group of programs from the disk into core. Investigators in the other laboratories merely had to go to their console to begin work. When each laboratory had completed its daily work, an interrupt from the console in that laboratory would signal the computer that there was no more

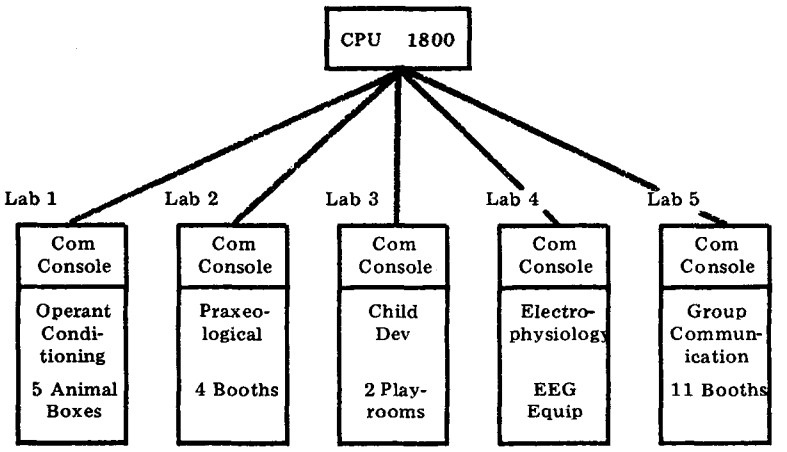

Fig. 5. The various laboratories serviced by the IBM 1800.

experimentation to be done and all accumulated data had been transferred to disk storage. When the last laboratory signed off, all of the data which had been collected from each laboratory and stored on disk was converted to a decimal format and delivered in the form of punched cards. This latter operation is a nonprocess application as contrasted to the data acquisition and control function. Our original system did not allow for time sharing between process control functions and nonprocess functions because of the lack of core. Thus the first laboratory to sign off could not get its data until the machine had been dedicated to nonprocess activities. Moreover, the lack of core severely restricted the number of laboratories that could be serviced by the system. It will be recalled that the operating system required 5,500 words of core. With an 8K system, this left only 2,500 words to be distributed among user programs. Thus, when a core load was developed, it could not exceed 2,500 words. Any number of laboratories could be serviced, providing the servicing programs for these laboratories could all be contained in a 2,500-word core load. By servicing, we are referring only to the data acquisition and control function. No core was available for data retrieval, new program assembly, or other nonprocess applications. By advance scheduling and building of core loads with different laboratory routines, we were able to function, although somewhat inefficiently.

However, the recent expansion of our system from an 8K to a 16K core will enable us to use another method of operating. Core is divided into basically two regions: one area is devoted to the system skeleton which includes the operating system; the second is called variable core. The skeleton area will contain the basic system routines—such as skeleton I/O, MIC, ITC, etc. User subroutines will also be added to the system skeleton by a program, similar to the Core Load Builder, called the Skeleton Builder. By including all laboratory control programs in the skeleton rather than in the variable core, as previously done, the variable core may be used exclusively for nonprocess core loads—the assembler, core load builder, data retrieval routines, statistical analysis programs, etc. The nonprocess activities will then be carried on without disturbing the on-line data acquisition programs that service the various laboratories. Since the data acquisition programs to service the various laboratories vary between 400 and 1,000 words, and 5,000 words of core are available in the skeleton section of core for user programs, it can be seen that a fairly large number of laboratories, each containing a number of devices can be included in the skeleton, leaving a considerable amount of core in the variable core section for nonprocessing applications. Still additional laboratories can be added to the variable core section, provided their servicing requirements can be kept within the time it takes to switch the nonprocess programs out of core and replace them with process programs.

In short, with a 16K-word centralized data-acquisition system, most, if not all, of the departmental laboratories can be serviced with the various users functionally unaware of the fact that they are, in fact, sharing the resources of the system. Finally, since 16K is only one-half of the 1800 machine's modular growth capacity (32K is maximum capacity), there is still room for additional expansion. This can be achieved at nominal cost, since additional core can be installed in 8K blocks.

REFERENCES

IBM 1800 Data Acquisition and Control System, functional characteristics. Order No. GA26-5918-6. IBM Systems Development Division, San Jose, California, 1966.

IBM 1800 Time Sharing Executive System, concepts and techniques. Order No. GC26-3703-1. IBM Systems Development Division, San Jose, California, 1970.

UTTAL, W. R. *Real time computers: Technique and applications in the psychological sciences.* New York: Harper & Row, 1967.

WEISS, B., & LATIES, V. G. Reinforcement schedule generated by an on-line digital computer. Science, 1965, 148, 658-661.

Computer-managed instruction: IBM System 360

JOHN A. CONNOLLY and THOMAS F. LAMBERT
American Institutes for Research
8555 Sixteenth Street, Silver Spring, Maryland 20910

Managing the educational process by means of an on-line computer system is a relatively recent application of computers to education. It is only within the last few years that large-scale instructional systems have been developed and managed from a computer base (Baker, 1971). The potential for on-line computer management of these instructional systems has not yet been fully explored.

The Instructional Management Program (IMP) is an on-line computer-management system. Its educational approach is based on principles derived from the psychology of individual differences, organized by system design techniques and implemented by means of computer technology. The educational and psychological aspects of the system have been described elsewhere (Connolly, 1970). The present report focuses on the computer-management features of the model.

IMP was developed by the American Institutes for Research for the School District of Philadelphia under a Title III grant. It is not related to any other learning program, such as Project PLAN or IPI. The system has been implemented on an IBM 360 computer, which is presently available at the School District headquarters. IBM was not responsible for the development of the program, and other general-purpose computer systems may be equally appropriate.

AN OVERVIEW OF THE SYSTEM

IMP is a computer-based system of individualized instruction designed to assure that all children master all of the basics (e.g., reading, writing, arithmetic). The prototype was developed and tested in a poverty-area school in Philadelphia, where it is now operating. The system is currently being disseminated to five additional schools in Philadelphia.

The model consists of three basic components. First is the evaluation system, which involves four different kinds of tests. Terminal measures indicate mastery or deficiency with respect to 70 different educational objectives. Diagnostic tests assess prior knowledge of the instructional subtopics available in the learning sequence for each objective. Aptitude tests measure learning strengths and weaknesses in terms of reading, verbal aptitude, learning style (i.e., visual, auditory, or kinesthetic), and cognitive style (i.e., abstract or concrete). Progress tests indicate knowledge of the material just presented in each