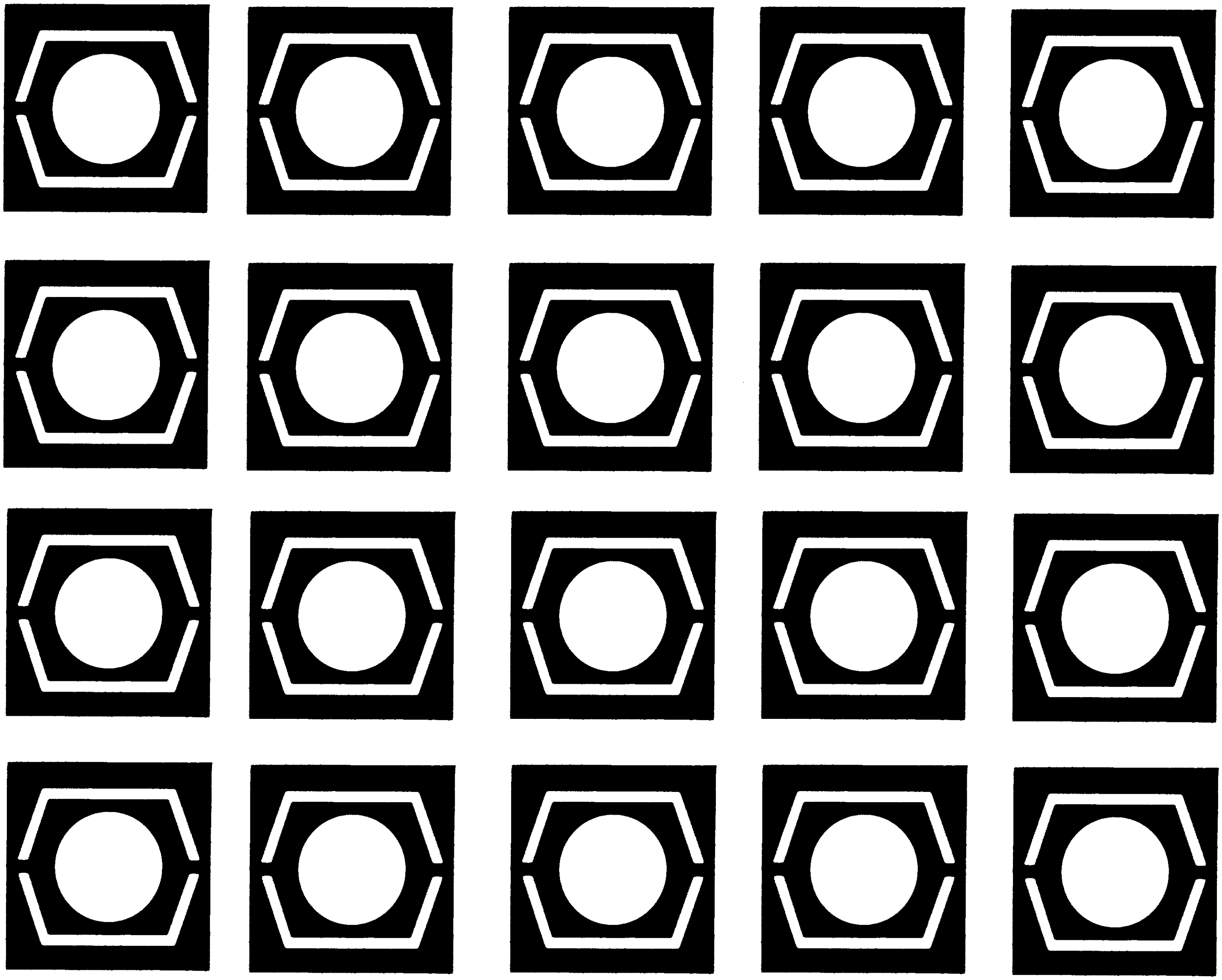


DECUS PROCEEDINGS

SPRING 1968

PAPERS AND PRESENTATIONS

of

Digital Equipment Computer Users' Society
Maynard, Massachusetts

SPRING 1968

PAPERS AND PRESENTATIONS

of

Digital Equipment Computer Users' Society

SPRING SYMPOSIUM

PHYSICS, EDUCATION,
BIOMEDICINE, TYPESETTING

APRIL 26, 27

Bellevue Stratford Hotel
Philadelphia, Pennsylvania

CONTENTS

APPENDIX 2

APPENDIX 3

* Abstract Only

PREFACE

This 1968 Spring DECUS Symposium boosted a record attendance of 270. The two-day session included, along with the presentation of papers, workshops on software systems for the PDP-8, PDP-9, PDP-10, Education, Modules, and a panel discussion on Clinical Laboratory automation. The Education workshop was the first endeavor of the Education sub-group and it proved both informative and interesting.

The meeting facilities provided by the Bellevue-Stratford Hotel enabled us, at times, to hold three simultaneous sessions successfully.

Papers published in this volume have been printed as received from authors with no editorial changes. In some cases, papers were not received in time for publication and abstracts of these papers have been substituted. If the omitted papers are at some time submitted to the users group, they will be published in the newsletter, DECUSCOPE. Reprints of papers presented here are available from the DECUS Office, Maynard, Massachusetts 01754.

This proceedings also contains a list of meeting attendees, the program, and an author/speaker index.

Special thanks to Mr Henry Sparks, University of Pennsylvania, for his able assistance in the preparation of the meeting and to, Professor Philip Bevington, Meetings Chairman, Professor Thomas Day, Dr. Belmont Farley, Dr. Sylvia Charp, Richard McQuillin, and Michael S. Wolfberg, session chairmen.

A STIMULUS-RESPONSE PROGRAMME FOR HOTEL ROOM INVENTORIES

D. W. Roberts, E. D. P. Manager
The Strand Hotel Limited, London, W. 1.
England.

ABSTRACT

The policy aims fulfilled by the programme are listed and an outline of the original booking system is given. The difficulties of this system are highlighted and the new system is described.

The action of a single function is detailed and the development of a single message is traced through seven phases in response to user requirements. The details of a useful method of handling dated information are given.

The system has been extended to two other hotels.

The purpose of this paper is to describe a somewhat unusual application in the field of business management. It is unusual in being the first such application, in having only intangible benefits, and in having been an unqualified success. The justification has always been expressed in terms of reducing the problems inherent in the management of a major hotel; the managers who now operate this system are all convinced that it has simplified the decision-taking in the control of the advanced bookings by presenting adequate and accurate information in an easily assimilated format. The unit, which was installed in the Strand Palace Hotel in April 1967, was the first time an hotel anywhere in the world had had a computer exclusively for its advance booking office.

A brief description of the hotels in the group would be advantageous here. The hotel which had the first of these installations has 780 bedrooms, catering for up to 3,500 guests per week, and the other two central London hotels in the group have between them a further 2,000 rooms, catering for up to 9,000 guests a week. These two hotels have been used as control units for comparison of the agreed policy criteria for the value of the new system. The avowed policy aims in the hotel group can be summed up very simply in that the percentage of beds let (the occupancy) should be maximised, and further, the proportion of the rooms which are let to guests who have prior written reservations (as opposed to walk-in or "chance" guests) should be maximised, within the first constraint. There are, of course, many other criteria employed by the manager in running an hotel but these are the only ones relevant to this computer system.

The system of controlling bookings employed previously in a hotel of this size has been the so-called "target" system, which consists chiefly of estimating the number of vacant rooms on any given day and accepting bookings up to that number (plus a few to allow for non-arrivals) and disregarding any information available about lengths of stay. This system has proved reasonably effective during most of the year because the trade shows a very clear seasonal and day-to-day pattern and managers have learned, by experience, how to allow for this. For instance, "Tuesday is the busiest night of the week", "there is always room on Sunday night", and similar maxims work very well when combined with good detailed knowledge of the reasons for sudden influxes e.g. important exhibitions, sales, etc.

The pattern was frequently disrupted in two ways: a) serious departures from previous years' demand e.g. the World Cup football matches which resulted in forecasts being made with no data available; b) errors of judgement or of forecasting from known events e.g. Easter 1967 fell early enough to coincide with the optimum marriage dates for income tax refund purposes, thus causing an unusually high proportion of honeymoon business which has a significantly greater length of stay than normal business. In both of these types of disruption serious errors (of over - or under-booking) could, and sometimes did, occur. It was quite clear that the only way to overcome this problem was to change from selling "targets" of vacant rooms to selling actual numbers of vacant room-nights (a total booking system).

The "target" system of recording bookings was based on charts on which one mark was made for each booking (on the expected date of arrival). This involved an average of 350 marks per day. It seems likely that about 5% of these were errors of one type or another.

In the total booking system each of these bookings would require, on average, about three marks each on separate booking cards, thus raising the volume of work to perhaps a thousand marks a day, and it is very likely that the error rate would have increased exponentially rather than linearly. It was therefore decided that it would be impracticable to run a manual total booking system for an hotel of this size because staff sufficiently intelligent to perform this task accurately would not be prepared to perform so monotonous a task. The exponentially increasing error rate is equivalent initially to an asymptotic monotonically decreasing efficiency function.

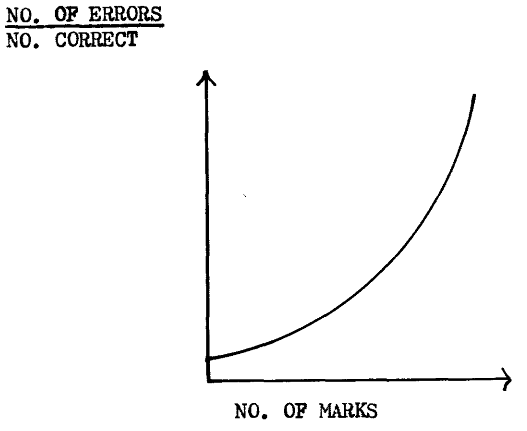


FIGURE 1

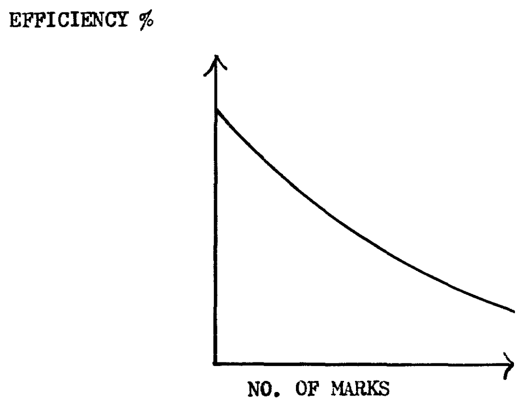


FIGURE 2

In neither of these systems is any attempt made to allocate specific rooms to guests prior to their arrival as a non-arrival proportion of 10% of expected guests is quite normal; figures as high as 40% have been known. This simplifies the problem slightly, but nevertheless it was felt that a form of electronic aid was necessary. The information necessary for a total booking system was available to the staff concerned but there was no method of retreiving and reducing this data into a usable form.

The original plan for this aid involved a purpose-built machine, containing approximately one thousand words (365 days, each containing three classes) of core store to be used as counters of available rooms with hardware logic to add and subtract bookings and cancellations. With the advent of the PDP-8/S, a stored-programme computer capable of providing this service, and yet cheap enough to be used exclusively for this purpose, was available on the market. One of these was ordered in January of this year and we took delivery early in March of a basic 4K machine with a teletype. This machine is a 12-bit word, serial logic machine with a cycle time of 6 microseconds costing, in the table model, £4,357 including duty. The programming was done in the machine's mnemonic assemberlanguage. All input-output is via the ASR-33 teletype in the configuration we use.

The programme performs several separate and distinct functions in a stimulus-response mode on data stored in a diary allocated 200$_8$ to 3703$_8$. Each day in the diary consists of five words, word one containing the date and day of week, word two the available single rooms, word three the available double-bedded rooms, word four the available twin-bedded rooms and word five the available three-bedded rooms for that day.

The most important function is BOOK. The message for this operation is initiated by the operator entering B on the teletype. The computer responds by typing BOOK, provided that this is the first letter of a message entered. The numbers and classes of rooms are then entered and typed back by the machine. The operator enters F and the computer types FROM, the date of arrival is then entered and typed. The operator enters ALT MODE (a non-printing key) and the computer replies TO. The departure date is then entered and typed back. The end-of-message signal is given by a full-stop. The message is stored (using only characters entered by the operator) in locations 3704$_8$ onwards, up to 3777$_8$. The message is then checked for validity and the rooms booked subtracted from the numbers available. In the event of the number of available rooms of any class on any day passing pre-arranged thresholds, warning messages are typed to the operator, instructing her to refer the particular days to the Manager for possible closure of bookings.

Since this paper is an applications paper I will go over the system development in some detail.

The inception of the project grew out of a brief to examine the systems in use in the hotel in the light of modern management methods and equipment. It was evident in the early stages of this study that advance bookings were a source of many management problems, and that although a total booking system would solve many of these problems it was impossible to use such a system satisfactorily using manual methods.

A study group was set up consisting of the Hotel Manager, the Assistant Manager, the Bookings Supervisor, the O.R. Manager, the O & M Manager and myself (then, Systems Analysis Manager) to investigate the problem of introducing a total booking system and from this group a recommendation was made to purchase a PDP-8/S to maintain an inventory of rooms available for a year ahead.

The initial installation went into full use during April, 1967 and the hotel bookings have been controlled entirely on the basis of the computer-provided information since the first week in May.

During the early stages of implementation various amendments were made to the basic programme design to accommodate the desires of the staff of the office using the machine, and the managers controlling them. These I shall demonstrate by tracing one message's development through the various phases of programme amendment.

1. The original warning messages from the programme to refer single bookings on 20/8 urgently to the manager for action as they are 5 rooms overbooked. This is the message as it was designed in the first instance.

2. The first modification was to eliminate each crossed zero and replace it by the letter O as it was claimed that crossed zero would confuse the staff. This amendment was introduced at the planning stage before the installation of the machine.

3. Within a few days of introduction of the system double spacing of these messages was eliminated because the messages were quite legible single spaced.

```
1) URGENT 2Ø/8 SINGLE  -5

2) URGENT 20/8 SINGLE  -5

3) URGENT 20/8 SINGLE  -5
4) U 20/8  S  -5
5) U 20/8$  S  -5
6) 20/8$  S  -5
7)
```

FIGURE 3

4. One second per line was saved some weeks later by reducing the message by 10 characters and printing only the initial letters of the URGENT and the SINGLE. This step had been contemplated initially but was delayed until the users were familiar with the system messages.

5. The day of week was introduced into all computer-generated dates on the specific request of the manager who chiefly took decisions based on the machine. The codes used are $ for Sunday, M for Monday, ? for Tuesday, W for Wednesday, T for Thursday, F for Friday and S for Saturday.

6. Since the reference of dates to the manager for action was never immediate the degree of urgency indication was next deleted, together with a sizeable programme sector which performed the relevant tests.

7. Finally, the messages were all made optional on the action of Switch 11 and are now only printed out when they are called for by the manager for decision-taking sessions; typically these occur two or three times during the course of each day.

Further amendments to this message are, I now feel, unlikely, though we may decide that it is totally redundant. The indications from detailed discussions with the managers are, however, that this is improbable.

The derivation of the date and day of week is identical throughout the programme and is dependent on a subroutine called DTOUT which interprets a condensed bit-pattern date from the diary of the machine. This consists of a single word split into three fields: bits 0 to 4 inclusive contain the day number in the month, bits 5 to 8 inclusive contain the month number and bits 9 to 11 inclusive contain the day of the week (1 for Sunday, 2 for Monday, etc.). If the day-of-week field contains all zero bits the day-letter is printed as N and if the entire word is zero the programme recognises this as a non-existent day. This is useful because it enables us to have a 372-day year (12 months each of 31 days) and simply suppress any operations for non-existent days.

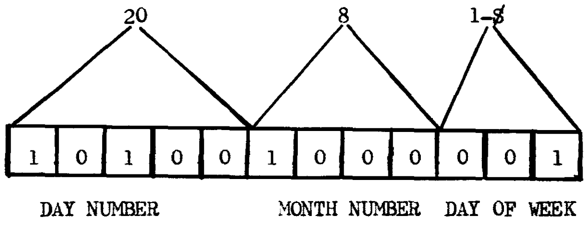


FIGURE 4

The address of a given date in the diary is therefore (37*(M-1)+D)*5+175 where M=Month and D=Day in octal.

The extension of the use of this system to the other two major hotels in the group took place last autumn, taking advantage of the seasonal lull immediately after the Motor Show.

THE FASBAC REMOTE ACCESS SYSTEM

Dan W. Scott
Manager, Remote Processing Services
Technical Services Division
University Computing Company
Dallas, Texas

ABSTRACT

FASBAC is a conversational system which facilitates setting up runs for the University Computing Company Direct Access Computer utility, via remote low speed terminals in the customers' offices. It is also the basis for a general-purpose direct access file system.

SUCCESSFUL TIME SHARING

Time sharing has become within the computing world the subject of lengthy debates on milli-second response time, swapping strategies, exotic memory management techniques, and whether bits should be colored blue or green. Were these truly critical parameters, the world would still be waiting for its first commercial quick response remote access system. Fortunately these are not the real issues, since time sharing has become highly successful with what are regarded as primitive systems. It would be well to examine the reasons behind the success.

First, it should be noted that time sharing and batch processing systems have two things in common: they use essentially the same kind of hardware, and they are both shared by many users.

Unlike batch processing, however, time sharing has brought convenience and simplicity to computing.

The successful time sharing system differs from the typical batch processing system in four significant areas.

1. It offers convenient physical access.

2. It offers quicker response.

3. It offers simple communication in both programming and control languages.

4. It offers program and data storage and editing facilities.

Simple communication with the computational facility is obviously attractive to the commercial user of calculational services; but, historically, progress in making general purpose systems easy to use was relegated to a secondary priority for many years after the appearance of FORTRAN. However, in specialized applications areas such as numerical control, report generation, CPM/PERT, and circuit analysis, new languages reduced the communication problems between the user and the computer. Finally, JOSS, developed by the RAND Corporation, and BASIC, developed by Professor Kemeny of Dartmouth College, appeared. Both the JOSS and the Dartmouth BASIC Systems reduced greatly the impedance mismatch between programming systems and the general engineering user.

These systems demonstrated that an acceptable commercial system must not only be physically convenient, but must be intellectually approachable as well. A sense that the machine is non-critical is important; these direct access systems appealed to many customers because errors are more rapidly corrected in privacy; a mistake is pointed out by the computer with a slap on the wrist, instead of the face.

As important as well thought-out programming languages, are simplified, well thought-out text editors and file systems. This element is not even now widely appreciated: the advantage to a user of ready access to calculational power, and, at the same time, of ready access to the power of computer-aided program filing, data filing, and editing. This file manipulation facility is the very foundation which makes simplicity technically attainable in the successful general purpose, remote access, computation system.

So, people are willing to pay for access to computational facilities. But this access must be both physically and intellectually convenient, and must apply as much as possible to all aspects of the computer utility business--commercial as well as technical, filing as well as computational. In summary, these requirements must be satisfied: reasonable programming languages, reasonable file languages, convenient access to the facilities, and for many applications, quick response.

THE UNIVERSITY COMPUTING COMPANY COMPUTER UTILITY DESIGN

There are several approaches to remote computer sharing design. On the one hand, we have the systems typified by the GE-265, the Dartmouth GE-625, and the SDS-940 small-scale time sharing systems. What these facilities do, they do well; however, they are of limited value to many potential users because of the limited resources allocated to each user. The objective of other designs has been to maximize the usefulness of a large processor. The central processor and its peripheral storage is intended to be fully available to each remote user, at the same time that it is carrying on conversations with all users. But the centralized approach has two drawbacks: (1) Complex and expensive implementation; (2) Poor cost/performance ratio.

Our approach with the UCC FASBAC design is to decentralize the large-scale processing; to distribute functions among subsystems; to be eclectic, selecting small, slow, processors and memories to do those things they do more economically, to allow the user of the large processor its unrestricted power. We combine the two implementation approaches just mentioned.

The UCC direct access design comprises three independent subsystems; the large-scale UNIVAC 1108 subsystem for compilation and numerical computation, the COPE remote terminal subsystem for high-speed card in put and print output, and the FASBAC remote terminal subsystem for conversational input and output editing. The UCC Direct Access System might well be called a large-scale time sharing system.

Two of these subsystems have required major development efforts by the University Computing Company for their creation, and all require substantial continuing development.

In order to supply physical access for large volume work, we have put high-speed card reader and printer terminals as close as possible to the user: this is our COPE communications facility. Second, we are developing a low volume communications facility, FASBAC, for file manipulation and text editing. This conversational text editing facility uses low to medium speed terminals in the users' offices and mass storage at the UCC utility center, and is connectable on demand to the 1108 and to the high-speed terminals.

The COPE subsystem multiplexes remote high-speed card readers and printers into the UNIVAC 1108 (Fig. 1). Figure 1 is simplified, and does not show, for example, the COPE communications controller. Both this central multiplexor and the remote terminal controllers use Digital Equipment Corporation PDP-8 processors. The COPE development is proprietary to UCC, but was a natural outgrowth of the UNIVAC 1107 EXEC-2 remote 1004 facility. However, quantitatively, its performance to cost ratio is much better. COPE provides unrestricted access to the 1108 when masses of data are to be moved. The very speed of the data transfer inhibits, of course, meaningful dialogue with a person.

THE FASBAC SUBSYSTEM DESIGN

As we have just seen, COPE facilitates bulk input and output of files by readers and printers which are close to the customer. However, file editing, output file browsing, input file corrections--the low volume, conversational elements of accomplishing jobs --are done via the FASBAC Teletype terminals.

FASBAC has as its objectives the increase in usefulness of the powerful UNIVAC 1108 systems to the low volume conversational user, and the provision of a basis for immediate remote access to bulk storage.

Technically, FASBAC is simply a communications-oriented text editing facility with bulk storage (Fig. 2); that is, it has a file system for mass storage, it has editing programs, and these facilities are conveniently available to individual customers through remotely located terminals which time-share a small central processor.

Text can be entered, stored, retrieved, and manipulated via low-cost, low-speed terminals suitable for text entry and display. This text may also be input to or output from the UNIVAC 1108 processor. The processing by the 1108 is then in its normal batch mode of scheduling, with multi-priority queueing. To the 1108, the PDP-9 processor looks like a card reader and printer.

In addition, FASBAC terminals can indirectly access the COPE terminals through the 1108 (Fig. 3).

The constraints of the FASBAC subsystem must be observed: low volume input, low volume output, and low volume file processing (even though the files themselves can be large). Because of the relatively small memory of the PDP-9, quicker response to sequential processing of large files is obtained when the processing is done by the large-scale machine.

The first objective of FASBAC is to provide convenience and quick response to the large-scale computer programmer. The files which he manipulates via FASBAC in a conversational mode are UNIVAC 1108 program files and data files to be executed by the large-scale processor. Formerly, users had to edit or control the editing of their files by means of physical manipulation of punched cards. FASBAC automates these procedures of entering and updating, doing away with the constraints of physical unit records, and making the file editing facilities context oriented. The user then has personal access to the UNIVAC 1108 without an operator being required for the mechanics of program and data entry.

The text editing programs include both line-replacement methods, as used in JOSS and BASIC, and context-replacement methods, as used in the MIT, DEC, and other editors. In addition, FASTRAC, a re-entrant resident interpretive program, is used for general purpose text manipulation. The choice of methods is left to the preference of the user.

The file system is straightforward, like that of BASIC, from the viewpoint of file naming, but includes more features to control access. An unusual feature allows efficient use of a file both for sequential and random access purposes.

Now for a description of the implementation details of these FASBAC features.

In contrast to the centralized approach, the FASBAC design has a hierarchy of processors. The control of the conversational complex is the operating system and the programs executed in the PDP-9. All the seemingly intelligent and friendly conversational aspects of character and file manipulation are carried out by the small processors, not the 1108.

The PDP-9 subsystem is a large configuration for its model. It has 32,000 words of 1-microsecond 18-bit memory, all processor options, a half million word drum with 8.65 millisecond average latency, a 1,000 card per minute reader, a 600 line per minute ASCII printer, 3 DEC magnetic tapes, and high speed paper tape I/O. The PDP-9 is the heart of the conversational system. It directs all functions and controls directly all I/O devices except those low speed devices connected to the PDP-8.

The PDP-8 supplies the low speed communications interfaces. It is connected to the PDP-9 through a high-speed data channel. Each PDP-8 can handle the equivalent of 32 Teletype circuits, each 110 baud, or any of three other speeds. It has no line bit buffers, and samples each line under program control at 8 times the line bit rate. This feature allows a variety of data formats and clock rates. The PDP-8 also does character transliteration, monitors the lines, answers the phone, hangs up, and dials out. The drum buffering of the messages is done by the PDP-9. A PDP-8 can also be located remotely to the PDP-9, and used as a line concentrator, using a duplex voice circuit.

ASCII character codes are used in the file system and from the Teletype terminals through the PDP-9 up to the UNIVAC 1108 interface. The UNIVAC 1108 software presently requires six-bit Fieldata code.

The mass storage device is the UNIVAC FASTRAND II drum. A multi-access controller, which also does drum address translation, was developed by the University Computing Company and Weismantel Associates. This controller allows the UNIVAC 1108 and two PDP-9 processors to share access to the FASTRAND II drum. It also provides separate duplex paths for core to core processor communication, between the 1108 and two PDP-9's.

The UNIVAC 1108 subsystem is reserved for the computational muscle of the complex. From the viewpoint of the PDP-9, it is used as a peripheral device, for unrestricted batch queued work on the 1108. Note that the conversational user enters jobs only indirectly into the 1108, via the PDP-9.

HARDWARE SELECTION

Time-sharing hardware is essentially similar to batch processing hardware. There are certain differences of emphasis. In addition to general criteria, such as anticipated reliability, maintainability, and satisfactory delivery schedule, the cost of the system must be evaluated in terms of other criteria regarding performance:

First, the processor must have a good means of handling I/O. For example, the interrupt scheme must be efficient, and allow multiple, dynamically changeable priorities. A variety of various speed devices must be attachable at a cost, in each case, appropriate to the device's requirements. These devices range from bulk store through swapping drums to large numbers of slow-speed terminals of various speeds and line disciplines.

Secondly, a variety of communications-oriented interfaces must exist for the subsystem. The low-speed modem control must handle dial in and dial out, as well as private wire service. There must be high-speed modem interfaces.

The processor and memory word size must handle efficiently characters of at least seven bits each, as this size character is better for conversational terminals than the six bit character. The choice of word size is also dictated by the requirements of interfacing with the 36-bit UNIVAC 1108.

Finally, the designer of the conversational processor must have made the appropriate economic trade-offs, for a given total cost, between processor and memory speed on the one hand, and elaborateness of the instruction repertoire on the other hand. If there are to be many simultaneous users, lots of memory is required; but the elementary operations performed by the processor are rather simple-minded. Therefore, it is essential for the designer of an economic processor not to sacrifice memory size and speed for an elaborate instruction set.

In summary, the criteria for the FASBAC processor hardware involved the exercise of judgment, to form an opinion regarding the final cost/performance ratio of a configuration, as of a certain date, but using different criteria than the usual ones of arithmetic capability. Needless to say, the ideal hardware was not found.

CONCLUSIONS

The major engineering advantage of the UCC

FASBAC design concept is the decoupling of many of the conversational elements of remote access from the highly efficient batch processing 1108 and the COPE high-speed card readers and printers. This master/slave design approach has been used in about 80% of the successful general purpose time-sharing systems of the past. We are just pushing the concept a little. We feel that this decoupling from the large, expensive, processor is the only economically rational approach to give conversational access to a major computer such as the UNIVAC 1108. Quick reaction to a keystroke is not economical with the same expensive processor and memory that can also invert a hundred by a hundred matrix in a few seconds. The hierarchy of processors and memories is a present-day economic necessity.

It has been demonstrated in the software area that several stages of impedance matching are necessary between the user and the computer system. The UCC Direct Access utility utilizes stages of impedance matching in order to handle efficiently the wide range of computer applications which today's users create.

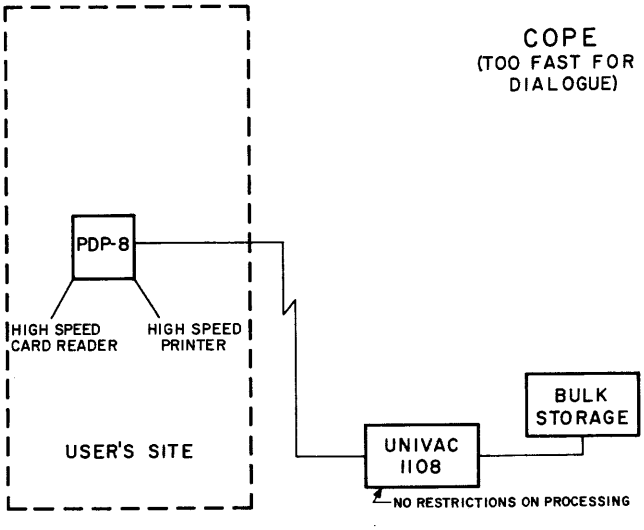


Figure 1 - COPE

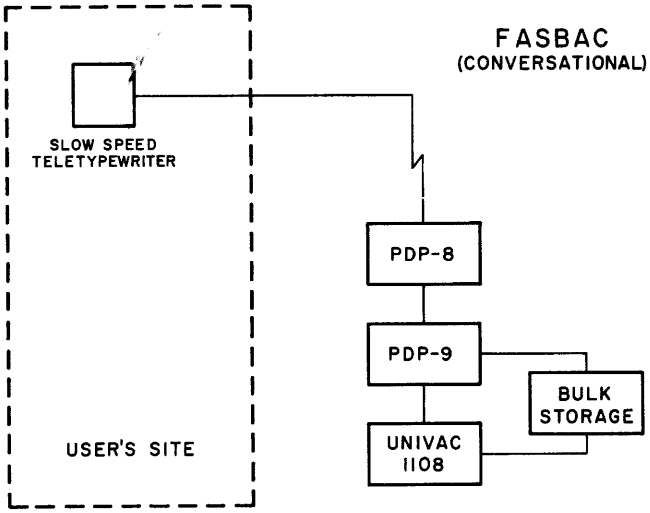


Figure 2 - FASBAC

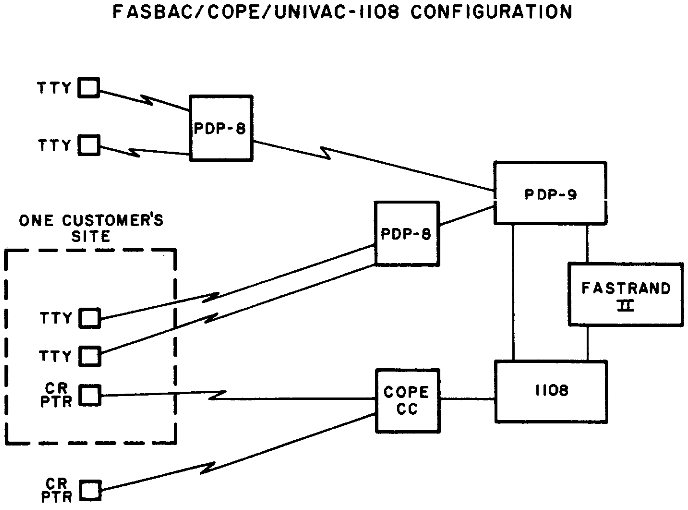


Figure 3 - The UCC Direct-Access Computer Utility

ON-LINE ANALYSIS OF WIRE SPARK CHAMBER DATA

P. F. Niccolai, R. H. Bicker, M. Edwards, and C. Swannack
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

A PDP-7 background/foreground mode of time sharing for on-line analysis of wire spark chamber data will be discussed. The background mode analyzes a representative sample of wire spark chamber data in the time available between interrupts from the wire spark chamber logic, an interface between the wire spark chambers and the PDP-7. The foreground mode logs data from the wire spark chamber logic onto magnetic tape. The experimenter is given control of the cyclic operations via the Teletype keyboard and may interrupt the program at any time to retrieve the output from the background mode. The particular analysis required from the background mode must be decided prior to load time and selected from the programs stored on DECtape. This program will be discussed as applied to pion absorption by light nuclei experiment at the Carnegie-Mellon University, Nuclear Research Center.

INTRODUCTION

The Carnegie-Mellon University Nuclear Research Center has built up a flexible hardware system for utilizing wire spark chambers, the center of which is a Digital Equipment Corporation PDP-7. Figure 1 illustrates the expansion of the basic PDP-7 with 8K of memory to include the following items.

1. Two TU-55 transports with the 550 control.

2. A sequential/random access Ampex core memory with 8192, 18 bit words and a cycle time of 1.5 μsec.

3. An Ampex TM-11 IBM compatible magnetic tape transport with a speed of 120 ips and a recording density of 556/800 bpi. The transport is now hard wired at 556 bpi. The magnetic tape control was built by laboratory personnel and is essentially similar to DEC's 57A magnetic tape control with the exception of unit and density selection options.

4. Anelex line printer - 647A option, 300 lpm.

5. A Houston Omnigraphic Model 6650 incremental plotter with a speed of 3 inches per second.

6. An interface between the ferrite core wire spark chambers and the PDP-7, henceforth referred to as the wire spark chamber logic (WSCL). The WSCL will scan 4000 wires in 1 msec.

7. We also have on order a Tektronix 611 storage display with a screen size of 16 x 21 cm.

This equipment and the ferrite core wire chambers are presently being used for the pion absorption by light nuclei experiment and will later be used for scattering experiments.

The laboratory's future hardware plans include the addition of a magneto strictive spark chamber readout system for experiments conducted at larger accelerator sites.

The repetition rate for a wire spark chamber experiment may be as high as 2000 events per second. At this repetition rate an experiment may typically consist of several million events, each with 100 numbers. The all important advantage of an on-line computer is its capacity for monitoring the performance of the apparatus and immediately analyzing the results of the experiment. Prompt feedback to the experimenter facilitates time-saving alterations of experimental parameters.

A complete and immediate analysis of each event would require large blocks of time on a large scale computer and at a high data transfer rate. Thus, this laboratory's economic alternative was the PDP-7, which we use to record the digitized WSCL readout on magnetic tape while calculating randomly sampled events. We call this a background/foreground mode of time sharing. Processing random events is the background operation. Data logging is the foreground operation. (Figure 2).

The software which we are currently engaged in programming is a set of program modules which may be selected by the experimenter at load time. He will either create a new program by typing in the names of available program modules displayed on the CRT or retrieve a previously created program from DECtape.

OPERATIVE SOFTWARE MODULES

These independent modules are similar in structure to FORTRAN subroutines and most of the background mode program modules are in our FORTRAN library for post-run analysis - the main difference being in the I/O handling.

Each module has 3 paths: 1) initialization, 2) analysis, and 3) a hard copy output cycle. Its path is selected by the main program flow in communication with the interrupt system. The argument list of each module consists of a set of global parameters which are typically the path selector parameter, the location of data arrays input to the output from a given module.

Foreground Modules

The first module selected by the experimenter must always be the foreground module, two of which are currently in our library, CHAMIN for the ferrite core WSCL input data and MAGIN for the magnetic tape input of data. The initialization path of the foreground modules sets up the program interrupt service routine. After the first pass, the program cycles through the background modules, returning to the foreground module only to service interrupts from the wire spark chamber logic (CHAMIN) or to load external memory from a magnetic tape record (MAGIN).

Both CHAMIN and MAGIN use the external Ampex memory as an I/O buffer. CHAMIN loads data from the WSCL into the accumulator, from there into the external 8K memory and finally into the one event buffer. (Figure 2) Although a hardware flag is associated with each WSCL word, an entire event is read into external memory and the one event buffer after each WSCL interrupt. The mean time between the formation of a data word in the WSCL, 35 μsec, is less than the time required to process instructions for storing each data word in its appropriate areas of memory and dismissing the interrupt. The filling of the one event buffer sets a software flag which is monitored by the background program only as it requires information for its process buffer. The filling of the process buffer from the one event buffer is done by a background module and necessitates the turning off of the program interrupt. Before dismissing the CHAMIN foreground module, a check is made on each event to determine the need for dumping the contents of external memory onto a record of magnetic tape. The dumping of external memory onto magnetic tape is done via the gather write mode of the 57A interface and takes approximately .7 seconds per record which includes time for start-up, writing, back spacing, and re-reading data for verification.

MAGIN fills the external memory via the scatter read option of the 57A interface. The one event buffer is then filled from the external memory instead of from the WSCL as in CHAMIN.

Background Modules

Before proceeding to some typical background modules, a description of the WSCL data word and the geometry of the planes is in order. The WSCL data word is 18 bits, 4 of which are used for flags and 14 of which are used to locate the wire. The number of words per event is variable due to multiple sparks or no sparks in a chamber. Two types of chambers are currently in use at the Nuclear Research Center; 1) beam chambers and 2) range chambers. Beam chambers are true wire spark chambers in that the ground plane of each consists of parallel wires spaced at 20/ inch. They are used to detect the precise location of a particle track. The range chambers measure a particle's energy by its range. All that is required of these chambers is an indication that a particle has passed through a given gap. The plane for each gap is divided into three concentric squares whose areas are labeled middle, guard, and outer. (Figure 3).

The flag bit assignment for both types of chambers is as follows for the given bit on a 1:

BIT	USE
0	Flags the presence of a pair - two adjacent wire sparks in a beam chamber or two successive middle, guard or outer areas going off in a range chamber (as contrasted to two adjacent wires sparking in a beam chamber).
1	Flags the presence of a triple - three adjacent wires sparking in the beam chambers or three adjacent areas in the range chambers.
2	Flags the first word of an event.
3	Flags the last word of an event.

Our beam chambers average approximately 1.5 wires per spark and, since approximately 50% of our data contains pairs and triples, the reservation of bits 0 and 1 to flag pairs and triples respectively, saves a corresponding amount of magnetic tape when logging data. It also considerably simplifies the spark configuration background module.

The remaining 14 bits are used to denote the wire number (4 bits - 16 wires per group) and group number (10 bits - 1024 possible groups). A given chamber has a variable number of groups.

A data buffer, referred to as the geometry buffer, adapts all program modules to a given chamber geometry. The data buffer is split into two parts, 1) the beam chamber descriptors, and 2) the range chamber descriptors.

The beam chamber descriptors have the following 18 bit format:

BIT	USE
0	1-flags the end of a segment (clustering of chambers along a straight line)
1	0-flags a horizontal chamber 1-flags a vertical chamber
2-8	reserved to indicate the spacing between chambers if it varies - they are currently spaced equidistantly
9-18	last group number for a given chamber

The range chamber descriptors are essentially the same, except that bits 1 and 2 are used to indicate the presence of a middle, guard or outer area, with bits 3-8 reserved for the gap size. Figure 4 illustrates the chamber configuration for the pion absorption by light nuclei experiment.

The main program flow depends upon the order in which the background modules were selected and the order of selection, as well as communication with the interrupt service routine, determines the path through the background modules from initialization to analysis to output. The cycling of the background mode is terminated by a keyboard interrupt when the experimenter selects either 1) the beginning of the output cycle, # or 2) a return to the loading phase by typing a $ on the keyboard. The background mode is also automatically forced into the hard copy output cycle for overflowing the event counter or when the end of the magnetic tape is reached.

The currently operative background modules are primarily of a maintenance nature. Complete kinematic analyses are performed during post run analysis on both the PDP-7 and the Univac 1108. The background modules operational on line are SEVTB, WIRMAP, CONFIG, EFFB and DEVB.

SEVTB monitors the one event buffer, fills the process buffer from the one event buffer, and computes single number data such as the total number of events analyzed and the number of blank events. This module exits only after processing a non-blank event with a monotonically increasing set of numbers.

WIRMAP computes a histogram of the number of times each wire is sparked, thereby giving an immediate indication of such malfunctions as group drivers and sense amplifiers in the WSCL. Output from WIRMAP may be selected on both the line printer and the plotter. Its input data is the path selector, the process buffer and the geometry buffer. Its output data is the wire map.

CONFIG calculates the configuration of the sparks in a given chamber and forms a histogram which counts the number of blanks, singles, pairs, triples, quadruples, 2-separated, 3-separated and others in each chamber. CONFIG also computes the co-ordinates for a given set of horizontal and vertical wire numbers. Thus its input data is the path selector, the process buffer and the geometry buffer. Its output data is the configuration histogram and the co-ordinate data for each plane.

EFFB computes the efficiencies of the beam chambers. Its input data is the path selector, the set of co-ordinates for each plane and the geometry buffer. Its output data is the efficiency calculation for each plane.

DEVB is used for a cluster of 3 beam chambers to compute the deviation of the middle chamber from a straight line. DEVB then computes a histogram of deviations, ± 4 wires from a straight line, for each chamber. The step size is quarter wire numbers. Its input is the co-ordinate data calculated in CONFIG, the path selector and the geometry buffer. Its output data is the deviation histogram.

The list of both foreground and background modules is open ended. We are currently operating with DECTRIEVE and the basic software for the PDP-7 to prepare, assemble, and load our modules and have been awaiting the PDP-9 advanced software, in particular MACRO-9 before completing the executive which will select and load our program modules from DECtape. We are also awaiting the arrival of the Tektronix scope display. Consequently the user must now prepare a paper tape with a list of the program modules and their arguments, and a paper tape with the geometry buffer. Finally he must select the appropriate paper tape sources for the modules and macro-definitions before obtaining an absolute binary from the PDP-7 symbolic assembler. The only changes to the PDP-7 symbolic assembler which have been made are a tailoring so that it fits on the DECTRIEVE system and a modification of the TEXT pseudo to generate 6 bit ASCII (eliminates need for the wordy teletype package in our system). The basic Editor has also been altered to fit into the DECTRIEVE system, output a page on the line printer, and use ASCII as its normal I/O mode. All absolute binaries are compatible with DECTRIEVE.

FUTURE SOFTWARE DEVELOPMENT

The executive as it is being written will rely upon a bootstrap in the upper portion of memory to retrieve the executive itself. The loaded executive would first inquire about the preservation of external memory and any portion of internal memory which should be stored in DECtape either as a permanent part of the system or temporarily. The executive would then list on the scope display, programs capable of fitting into available memory. Selections would be made from the teletype by typing a single alphanumeric call character displayed with the program name. The display list would be updated after each selection or a subsequent page displayed if a carriage-return were typed in lieu of any displayed selection. Program modules will be checked before displaying their name to eliminate those whose individual memory requirements exceed the area remaining. Arguments required by the program modules will be inquired of the experimenter as each program module is selected. All output from the executive would be on the scope display, all input via the teletype keyboard of DECtape. Teletype input into the executive is for non-routine runs or experimental variations. Routine procedures would be prepared on pre-punched paper tape and read onto a DECtape file.

The selection procedure will be terminated by the experimenter or the executive itself when all remaining program modules individually require more memory than remains. The program modules will be stored in external memory after retrieving them from one pass over DECtape, building up an image of the internal 8K memory. After building up the program in external memory a map will be output and control transferred to a resident program stored just below the bootstrap which will load the operational program from external memory into central core filling in absolute addresses thereby destroying the executive. The executive program may be retrieved at any time by its resident bootstrap by restarting the computer at the bootstrap starting address or typing the $ when running a program loaded via the executive system.

We have delayed finalizing this system in order to avoid a duplication of effort in writing a relocatable macro-assembler and believe once MACRO-9 is working on our PDP-7 that the major portion of the software development will be the writing of a loader compatible with our proposed executive.

* This work has been supported in part by the U. S. Atomic Energy Commission, contract AT(30-1)-882.

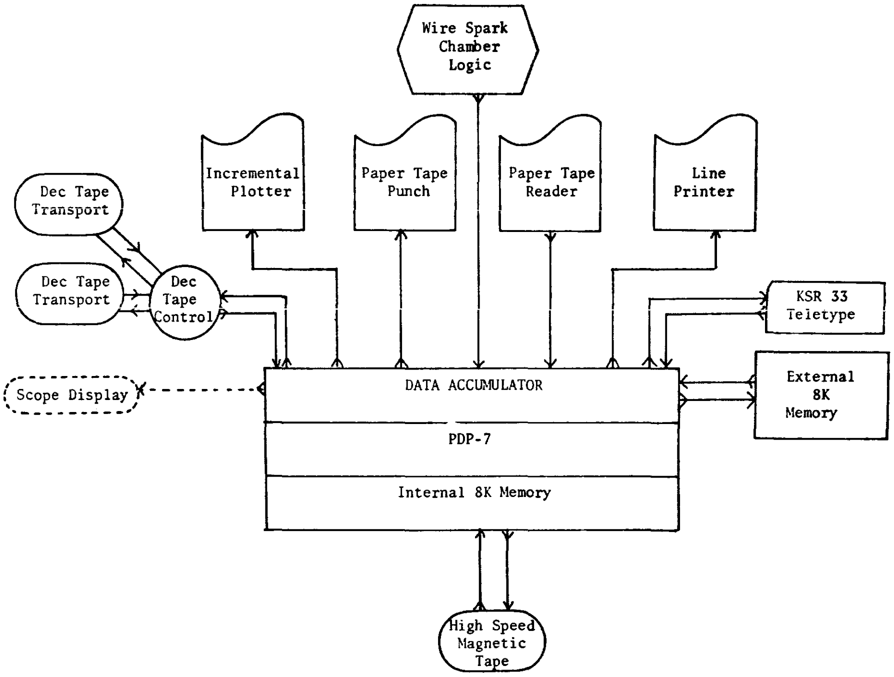


FIG. 1

EXPANDED PDP-7

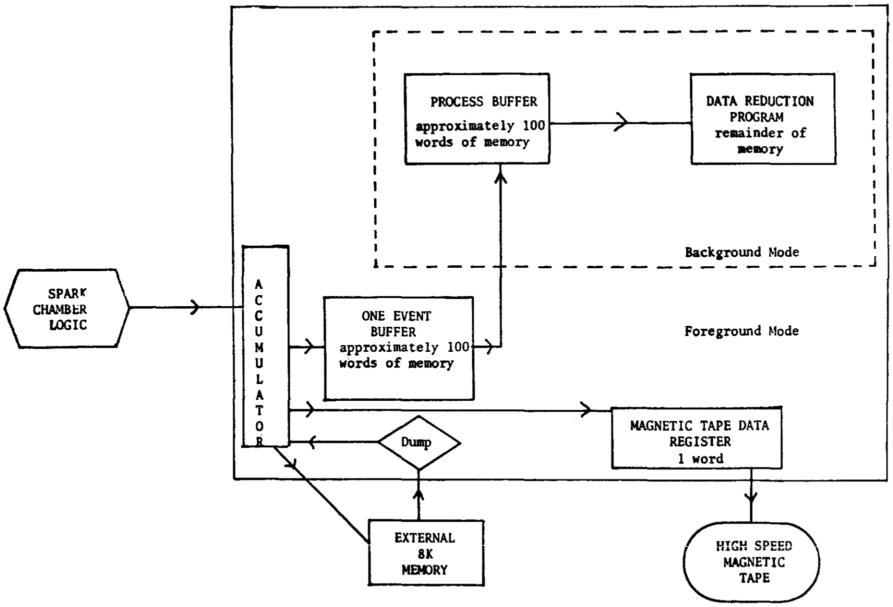

PROCESS BUFFER approximately 100 words of memory
DATA REDUCTION PROGRAM remainder of memory
Background Mode
SPARK CHAMBER LOGIC
ACCUMULATOR
ONE EVENT BUFFER approximately 100 words of memory
Foreground Mode
MAGNETIC TAPE DATA REGISTER 1 word
Dump
EXTERNAL 8K MEMORY
HIGH SPEED MAGNETIC TAPE

FIG. 2

BACKGROUND/FOREGROUND DATA FLOW

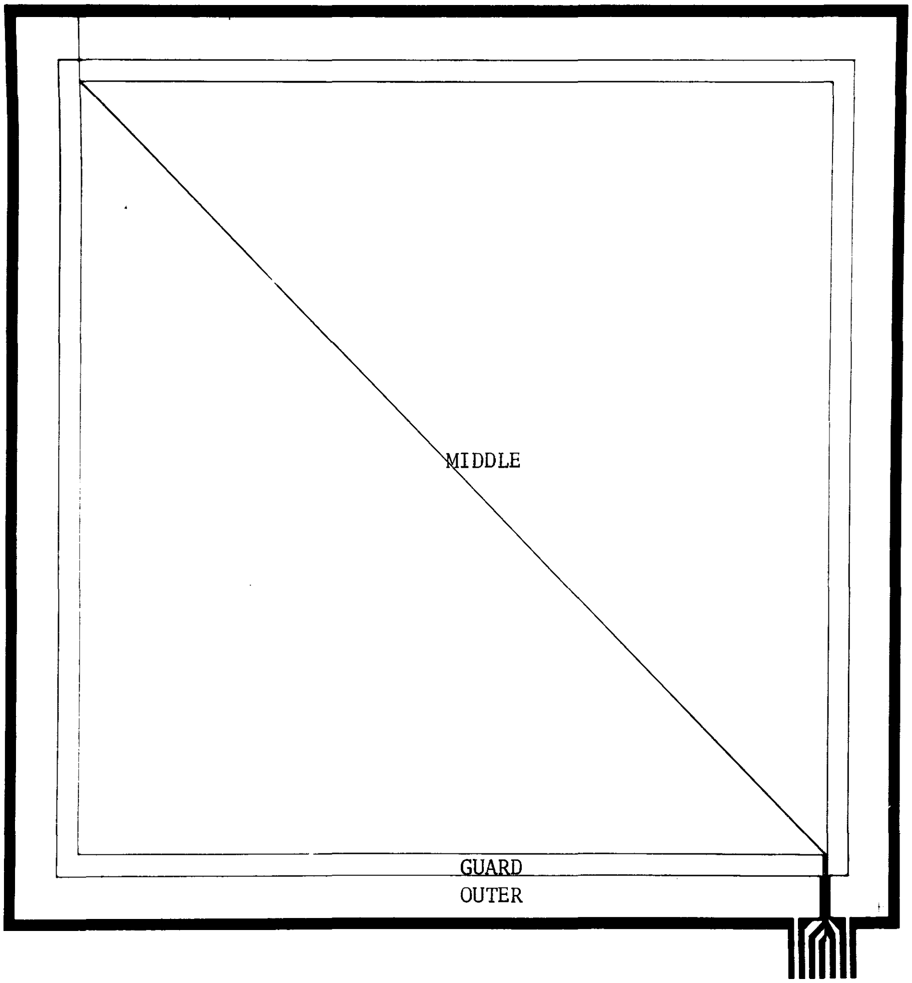


FIG. 3

RANGE CHAMBER CONSTRUCTION

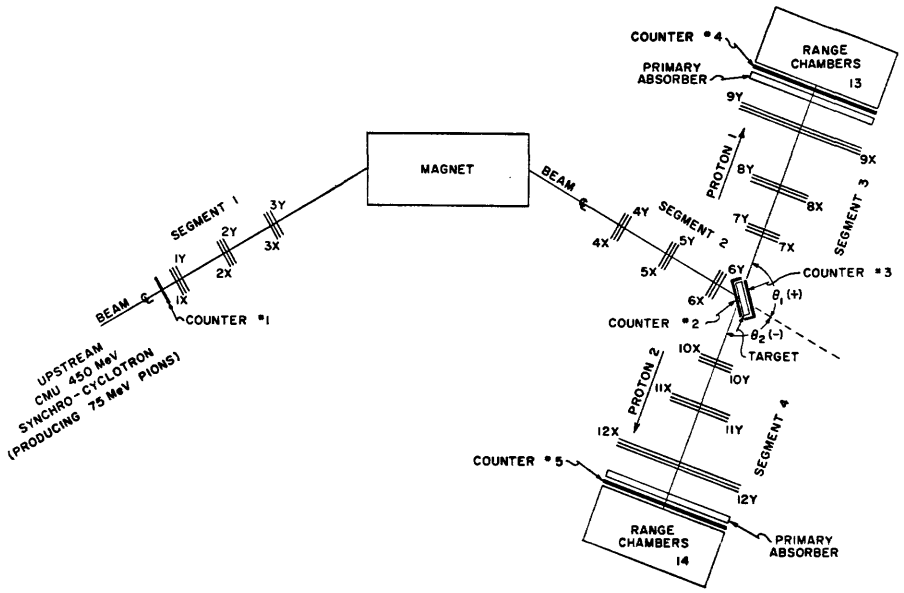


FIGURE 4
SPARK CHAMBER CONFIGURATION (PLAN VIEW)
PION ABSORPTION BY LIGHT NUCLEI

PDP-8 ON-LINE DATA ACQUISITION SYSTEM FOR HIGH ENERGY PHYSICS*

Paul Shrager and Larry Taylor
University of Pennsylvania
Philadelphia, Pennsylvania

Abstract

This presentation will be a description of an on-line data acquisition system for magneto strictive spark chamber readouts in high energy physics. The system outputs to an incremental magtape unit and CRT display tube. The system includes a real-time clock, high speed paper tape reader, 24-channel A-D converter for experimental parameter monitoring.

In addition to data acquisition and output data verification, simple on-line analysis is performed, including histograms showing distribution of sparks in chambers. The oscilloscope display includes a reconstruction of the elementary particle event that occurred in the spark chamber.

*This paper was not received for publication.

THE ON-LINE USE OF A PDP-9 AND AN IBM 360/65 IN A PROTON-PROTON BREMSSTRAHLUNG EXPERIMENT USING WIRE CHAMBERS

D. Reimer
Institute for Computer Studies, University of Manitoba, Winnipeg, Canada
and
J. V. Jovanovich, J. McKeown, and J. C. Thompson
Physics Department, University of Manitoba, Winnipeg, Canada

ABSTRACT

The system allowing conversation between two computers (PDP-9 and IBM 360/65) and an on-line data analysis from a wire chamber experiment is described. Wire chambers and scintillation counters are interfaced to the PDP-9 which is connected to the IBM 360/65 via a standard DEC high speed data link. The PDP-9 performs preliminary analyses and selection of detected events. A FORTRAN program residing in a small partition of the 360/65 memory completes kinematic analyses of events accepted by the PDP-9, stores the results on magnetic tape, and returns formed histograms to the PDP-9 for visual display or graphical plotting.

INTRODUCTION

The two computer system at the University of Manitoba cyclotron is composed of a PDP-9 linked to an IBM 360/65 via a standard DEC high speed data link. The PDP-9 part of the system and its applications in three different physics experiments was described previously[1]. In this paper the use of both computers in the proton-proton bremsstrahlung experiment is discussed. A simplified schematic presentation of the experimental setup (the wire chamber spectrometer and computer system) is given in Fig.1, and a brief description is presented below (see ref.1 for more details).

A beam of 45 MeV protons from the Manitoba cyclotron strikes a 20 cm long gaseous H_2 target (see Fig.1). Each of the two outgoing protons pass through two of the four wire chambers (WCH1-WCH4) and into plastic scintillation counters (S1 and S2). Information from wire chambers defines proton trajectories, and pulse heights from the counters determine their energies. The PDP-9 assembly program tries to reconstruct a vertex from proton trajectories, i.e. to establish within an accuracy of several millimeters whether both observed protons come from the same point in the target. If they do, the relevant coordinates of the proton trajectories and counter pulse heights are sent to the IBM 360/65 computer, otherwise the event is rejected. The latter computer then performs a full kinematic and statistical analysis of the event using Fortran programs, and returns some histograms of interest to the PDP-9 for visual on-line display on the oscilloscope, or for plotting on an x-y plotter (CALCOMP). Data computed from each good event is also stored on a magnetic tape by the IBM 360/65 for later off-line sorting, selecting and histogramming.

In Section (A) the data-link hardware and software is discussed. Section (B) explains software written for and used in p-p bremsstrahlung experiment while concluding remarks are made in Section (C).

(A) DATA-LINK

Data-Link Hardware

The data-link[2] is the standard DEC long line high speed data-link (DX36B-DX09B) connected to the PDP-9 via a Data Channel and to the 360 via a Selector Subchannel (Channel). The distance between the two computers is about 1,900 feet, allowing a maximum data transfer rate of 50,000 bytes per second.

The 360 controls whether the Input/Output (I/O) operation is to be a read or write, to which the PDP-9 must respond by reading Data-Link status registers to ascertain the operation and set appropriate bits. The data-link hardware packs PDP-9 words to be transferred in four different ways to make them compatible with the byte oriented System 360. These options are:

1) 1 byte (8 bits) to/from rightmost 8 bits of 1 PDP-9 word,
2) 2 bytes (16 bits) to/from rightmost 16 bits of 1 PDP-9 word,
3) 3 bytes (6 binary zeros and 18 bits) to/from 1 PDP-9 word,
4) 4 bytes (14 binary zeros and 18 bits) to/from 1 PDP-9 word.

The last option has been used to transfer data to/from Fortran programs.

To establish the reliability of the data-link a large block of data was written from the 360 into PDP-9 memory, the same data read back, and then compared. This experiment was carried out at various times and for various time intervals, ranging from several minutes to three hours. The number of bytes transferred ranged from 1.6 million to 500 million. A total of 12 bits were dropped on one data transfer (this during the three hour run). The 500 million bytes transferred during the three hour run is an order of magnitude greater than the amount of data expected to be transferred during the lifetime of the wire chamber experiment.

Data-Link Software

It is possible to drive the PDP-9 from the 360 using normal Fortran READ and WRITE statements. This allows a maximum data transfer of only 20 PDP-9 words for each READ or WRITE. To circumvent this limitation and to allow certain types of error recovery, channel programs[3] were written in 360 Assembly language. Thus the Fortran program calls an assembler subroutine with two entry points called PDPR and PDPW, to initiate data-link transfers. Each transfer is accomplished in three stages:

1) Initial Selection,

2) Data Transfer, and

3) Ending Sequence.

They are all handled by the 360 Selector Channel to which the PDP-9 Automatic Priority Interrupt (API) service routine must respond.

The flow of information between the two computers is presented in Fig.2. As indicated in this figure, the communication sequence is always initiated by the 360/65 with a standard CALL PDPR or CALL PDPW statement placed anywhere in the Fortran program which runs on-line with the PDP-9. After the subroutine PDPR (or PDPW) has been called a READ/WRITE is requested from the 360/65 Operating System which subsequently issues a Start Input Output (SIO) instruction to the channel. The Operating System returns to batch processing mode and no further 360 action is required for the data transfer until the PDP-9 signals that the operation is complete. The Channel now sets some control bits and Initial Selection Done (ISD) flag which causes an API interrupt on the PDP-9. As there are altogether six flags which can cause a data-link API interrupt, the API service routine first determines which flag caused the interrupt and then clears it. When an ISD flag is detected, another status register is read to determine whether a READ or WRITE operation was requested by the Channel. The PDP-9 Data Channel (DCH) is initialized by providing the starting address minus one and word count of the data to be transferred to the PDP-9 memory. The Data Transfer (DT) bit, the R/W bit, and pack option are set up for the data-link. Then control is returned to the interrupted PDP-9 program by breaking from the API service routine.

At this point the memory to memory data transfer takes place requiring no PDP-9 or 360/65 CPU attention. The time required for the data transfer depends on the number of PDP-9 words and pack option used. It takes about 80 μsec to transfer one PDP-9 word (18 bits) to/from one 360 word (32 bits). When the 360/65 byte count and/or the PDP-9 word count goes to zero, the Channel sets the Data Transfer Done (DTD) flag causing an API interrupt on the PDP-9. The service routine clears DTD flag and tests if a parity error was made. To signal the PDP-9 users program that the 360 has transferred data, the Data Link State (DLS) word in PDP-9 memory is set to one. Control is then returned to the interrupted PDP-9 program.

The Channel is "hung" until the users program calls a PDP-9 subroutine DLINK which allows the third stage of the three stage data transfer sequence to proceed. This is feasible since the data link is the only device at present attached to the Channel. Thus completion of the three stage sequence occurs only when required by the logic of the users program. The subroutine DLINK tests if the data transfer is completed and if not, waits until it has, then sets Ending Sequence (ES) flag, sets the DLS word to zero, and sets Channel and device end. The Channel then sets the Ending Sequence Done (ESD) flag which causes an API interrupt on the PDP-9. As before, this flag is cleared and control is returned to the interrupted PDP-9 program. As all three stages of the data transfer are now complete the Channel causes an IO interrupt on the 360 and control returns to the originally called assembler subroutine (PDPR or PDPW).

From the PDP-9 point of view the transfer is completed at the end of the second stage. The reason for organizing the software in the above manner is because at this stage of 360 software development the 360 cannot be interrupted by the PDP-9. To allow the 360 to process data while the PDP-9 collects more events the Fortran program on the 360 is usually so organized that each "CALL PDPW" is followed by a "CALL PDPR". This allows the two computers to function asynchronously. If the sequence involves only successive READ's from or WRITE's to the same memory locations in the PDP-9 the two computers must function synchronously.

(B) SOFTWARE FOR THE p-p BREMSSTRAHLUNG EXPERIMENT

Several different programs are used in the p-p bremsstrahlung experiment. Most of them belong to three general classes: (1) programs with only PDP-9 on-line to the experimental equipment, (2) programs with both PDP-9 and 360/65 on-line to the experiment and (3) programs using both computers but not on-line to the experiment. The last category will always be referred to as "off-line" programs.

We have three programs from the first class. They are mainly used for testing of experimental equipment. When taking and processing data a machine language PDP-9 program (called "Vertex") and a Fortran 360/65 program (called "Kinematics") are used together. The needs of the experiment require several different versions of these programs (coded as V1, V2, ... KIN1, KIN2, ... etc.) to be readily available. For off-line data analyses several Fortran programs are used and in general they can be called from the PDP-9 teletype console. To cope with this large and ever expanding variety of programs, we have organized all of our software within the framework of two specially written monitors, one for the PDP-9 and the other for 360/65.

Description of Monitors

The PDP-9 Symbolic Assembler Language Monitor (SALMON) is present all the time in the PDP-9 memory, occupying the lower 2600_8 locations. It handles three basic input-output operations, namely teletype, DEC tape, and data-link, performs some elementary bookkeeping, and requests and accepts control messages from the experimenter. SALMON uses Program Interrupt (PI) and Automatic Priority Interrupt (API) features of the PDP-9 to allow interleaving of the above and any of the other Input-Output devices used. By typing appropriate mnemonics of two characters SALMON loads the selected program from DEC tape into a predesignated section of the PDP-9 memory and starts its execution. This feature enables us to "overlay" programs without disturbing areas of memory which might include the collected data or constant parameters. SALMON also handles the above mentioned Input-Output operations whenever they are requested by the users program. In this case normal entrance points into SALMON are through a set of JMS (jumps to subroutine) instructions to absolute addresses. This feature enables programmers to assemble their programs independently of SALMON.

The monitor for Fortran programs used on the 360/65 (called FORMON) is a very short program also written in Fortran. It is based entirely on a feature of the 360 system of overlaying Fortran subroutines. Therefore, all of the Fortran programs we are using are declared as subroutines and FORMON merely selects the one which was called from PDP-9 by typing its four character name.

Description of On-Line Programs

So far three different PDP-9 programs have been used for testing the equipment and they do not require the 360/65 on-line. These programs are: (1) "Hardware Test" used for the initial testing of ferrite core readouts, interfaces and electronics, (2) "Wire Histogram" used to test the relative efficiency of spark chamber wires and ferrite cores by making histograms of individual wire firings, and (3) "Wire Chamber Efficiency" used for the initial or routine testing of sparking efficiency. Wire Histogram and Wire Chamber Efficiency programs normally display selected histograms on the oscilloscope or plot them on the CALCOMP plotter. All three can also have data printed on the teletype or written on the DEC tape. As these programs are rather simple and are not on-line to 360 they will not be discussed here.

The Vertex and Kinematics programs are the most important ones as they are used for on-line real time data taking and analysis with the two computers. They are also the most interesting from the software point of view as they employ virtually all available facilities, hence, they will be described in some detail.

A simplified flow diagram of Vertex and Kinematics programs is given in Fig.3. The following procedure is used in loading and executing these programs: With SALMON residing in the memory of PDP-9, the experimenter types the mnemonic "IN" which loads a special initialization program from the DEC tape. This program is used first to receive constants from the teletype or paper tape and store them in the memory. These constants are to be used by the Vertex program to be loaded later. Afterwards, the 360 operator is asked, via an interphone, to load FORMON into partition. On execution, FORMON requests the 360 operator to enable the Data Link by manually flipping a switch. When this is done, FORMON requests from the experimenter the name of the program to be executed by the 360/65. The experimenter responds by typing the name of the program (say KIN1) which is automatically loaded into 360 partition together with the program constants. The execution of KIN1 is begun at this point and the experimenter must specify whether magnetic tape or printer are to be used and must type the bookkeeping information essential for a complete record of the run. Afterwards, he is asked by KIN1 if he wants to change program constants and if so, he does it in a way described below and in Fig.4. When KIN1 is ready to accept data, the experimenter loads a Vertex program (say V1) which initializes itself by reading in the program constants previously stored in memory by the IN program. The program V1 then turns on the fast electronics, jumps into an oscilloscope display routine and waits for an interrupt. When a wire chamber interrupt comes, the ferrite cores are read into the computer, decoded, track coordinates computed, double track ambiguities resolved, and an attempt is made to reconstruct a vertex. If successful the track coordinates, scintillation counter pulse heights, and event number are stored in a buffer whereupon the PDP-9 returns to the display routine. If at any time the experimenter wants to terminate or suspend the run he types an appropriate code on the teletype and a key word (termination index) is entered into the output data buffer. When this buffer is full, its contents are transferred to the 360/65. KIN1 begins the analysis of each event by recomputing the vertex and imposing more stringent acceptance criteria than the PDP-9. If the event passes this test the angles and energies of outgoing protons (corrected for energy lost before reaching the scintillators), gamma ray, and incident proton are calculated. In addition the errors of each of these quantities and a χ^2 value are computed. At this point all of the information received from PDP-9 and that computed by KIN1 is stored on magnetic tape. Finally, three single histograms, a two-dimensional histogram and a scatter plot are updated. After all the data from a given transfer have been processed the histograms and scatter plot are sent back to the PDP-9 for display. The termination index is then tested and depending on its value the data taking and processing is continued, suspended or ended. If suspended KIN1 requests a change in histogram constants and the experimenter enters the new values. KIN1 then clears histograms to zero and the data taking continues as if it were not interrupted. If an end of run is recognized some bookkeeping information is requested, the cross sections are calculated and some relevant information is returned to PDP-9. At this point the experimenter may move the obtained histograms into a separate buffer area and initiate an independent CALCOMP routine which can plot histograms simultaneously with the execution of the next run.

The flow diagram of the basic write and read sequence used throughout the initialization program is given in Fig.4. A message composed of a certain number of alphameric characters which is defined and stored in the Fortran program is first translated from the EBCDIC code (used by our 360/65) into ASCII code (used by PDP-9) by calling an assembler subroutine. Another assembler subroutine unpacks words from 4 characters to 2 characters/word and by calling the PDPW subroutine, the message is sent to the PDP-9 which types it. At this point the experimenter must reply and has a choice between alphameric or numeric information. The numeric information can be entered into the PDP-9 from paper tape or the teletype and can be changed after entry but before transmission to 360. The 360 must know whether to expect alphameric or numeric information. If alphameric information is received it is packed from 2 to 4 characters per word and translated from ASCII to EBCDIC code. As numeric information is always transferred in integer form the received numbers are usually floated and normalized as needed by KIN1 or any other FORTRAN program.

Description of Off-Line Programs

The most important off-line program is Correlate (CORR) which reads from magnetic tape data taken in the past in one or more runs, stores it temporarily on disk, combines it as requested, and produces histograms in a manner very similar to that of KIN1. The histogram constants are read in from PDP-9 using the same initialization program (IN) as in the case of the V1-KIN1 programs. The histograms are also returned to the PDP-9 for display on the oscilloscope or plotting on CALCOMP plotter, or they can be printed on the 360 printer.

For practical reasons it is very important to be able to call and run CORR from the PDP-9 console as it provides a facility for "quasi-on-line" data processing. Often a situation would arise when the information obtained from V1-KIN1 programs is insufficient to make some decisions concerning further data taking. In this case the experimenter may wish to summarize results of certain runs taken in the past and with CORR he can obtain this information within a few minutes.

Some other off-line programs of minor significance are used. For instance, when processing data with a program sitting in 360 partition the printing must be done completely off-line. Therefore, in those infrequent cases when we want printer output (the availability of CALCOMP plotter and teletype reduces significantly the need for printed output), the results must first be temporarily stored on disk or tape and then printed out using a special program to be run in batch processing mode, rather than controlled from PDP-9.

(C) CONCLUSION

Not many two computer systems have been used for on-line data processing in accelerator experiments. The system which seems to be in the most advanced stage is the one existing at CERN[4]. Our experience with the system described here is still limited and therefore it would be difficult to present a detailed analysis of its advantages and shortcomings. Nevertheless, some remarks are in order.

1) In our system the PDP-9 and 360/65 are used in a complimentary manner. The PDP-9 controls directly all experimental equipment, all output devices an experimenter needs while taking data, and performs some data processing. These operations abound in logic decisions and special input/output instructions but need very little arithmetic to be performed. Therefore, the programming for PDP-9 is done in assembly language. Long arithmetic operations are relatively easy to program in Fortran and not likely to be changed very often so they are done on 360/65. This division of the software between the two computers confines most of the alterations to the more accessible PDP-9.

2) The 360/65 Operating System used is able to overlay subroutines belonging to the same program. This feature enables us to overlay Fortran programs in a very simple manner and run them in a relatively small partition with 56K bytes (14,336 words) only. If need be, our program could be overlayed in a more complex manner and fitted into a smaller partition. When this parti-

tion is reserved for our programs there are 110K bytes (27,160 words) of the 360/65 memory available for batch processing wnich is sufficient for most of the regular 360/65 users. As a consequence, the scheduling of the 360/65 allows us to use the partition up to 16 hours/day and 5 days/week. (Presently, the 360/65 is being used 15 shifts/ week.)

3) The PDP-9 monitor enables us to overlay our PDP-9 programs resulting in an efficient use of its memory. The monitor also allows us to change programs with ease using DEC tape although this is rather a slow device. The use of disk or drum would be preferable.

4) The selection of 360/65 programs is controlled entirely by the experimenter from the PDP-9 teletype. The 360/65 operator must intervene only to load and unload the partition, to change magnetic tapes, and to initiate off-line printout on the printer.

Not all of the software described in this paper has been tested over an extended period. Software development has been greatly assisted by the use of raw information from the wire chambers (non-processed output from ferrite cores and pulse heights from analog to digital converters) which was stored on DEC tape during a previous run. In Fig.5a we present a photograph of three histograms which have been computed on-line by the 360/65, returned to the PDP-9 and displayed on the oscilloscope. In Fig.5b we reproduce a scatter plot also computed on-line by the 360/65 and returned to the PDP-9 but now plotted on the CALCOMP plotter. These histograms and scatter plot have been produced using only very preliminary calibration constants, hence scales are not presented.

ACKNOWLEDGEMENTS

We are very grateful to Professor B. A. Hodson, Director of the Institute for Computer Studies, who has made every effort to allow our extensive use of the 360/65; and to Mr. J. Epp and Mr. P. Caufield for scheduling the use of the 360/65.

It is a great pleasure to express our thanks to Professor D. O. Wells for some discussions concerning data-link hardware and software and for supervising the installation of the data-link; to Mr. R. Kawchuk for writing CALCOMP subroutines; to Mr. D. Peterson for writing Hardware Test and Wire Histogram programs; and to Mr. L. Greeniaus for the Correlate program.

We acknowledge the financial support of the Atomic Energy Control Board of Canada which made this work possible.

REFERENCES

[1] L. W. Funk, J. V. Jovanovich, R. Kawchuk, R. King, J. McKeown, C. A. Miller, D. Peterson, D. Reimer, K. G. Standing, and J. C. Thompson; Proc. DECUS Fall 1967 Symposium, Nov. 11-12, 1967, p.187-192.

[2] S. Booth, "PDP-9 to IBM System/360 Selector Channel High Speed Data Link at the University of Manitoba"; Digital Equipment Corp., December 20, 1967 (unpublished).

[3] IBM System/360 Operating System Manuals: Form C28-6550, "System Programmers Guide"; Form C28-6647, "Supervisor and Data Management Macro-Instructions"; Form A22-6821, "Principles of Operation".

[4] T. R. Bell, B. C. Levrat, P. J. Marcer, and E. M. Palandri, "A Data Link between Two Computers"; CERN 67-31, December 21, 1967 (unpublished).

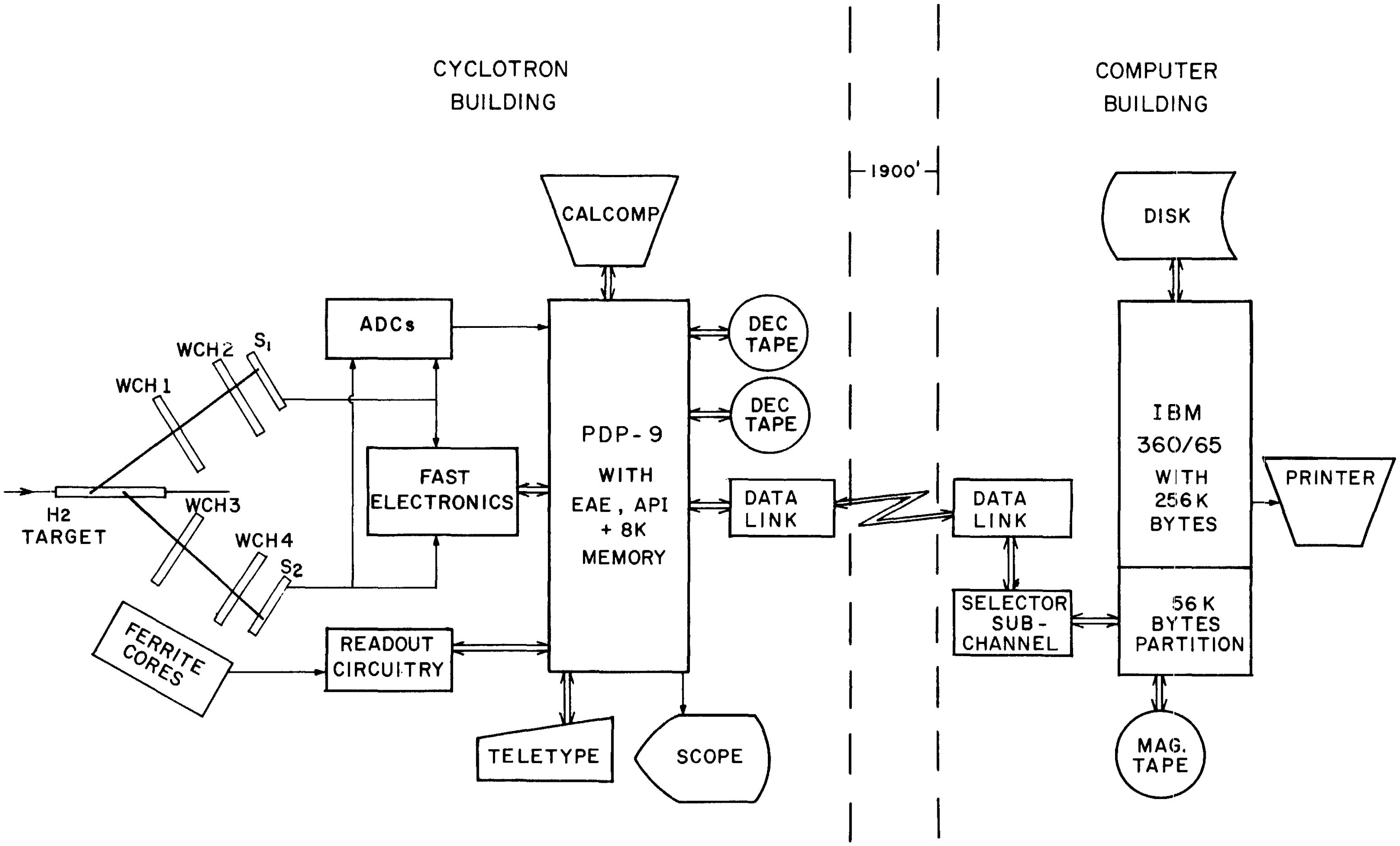


Figure 1 Manitoba cyclotron computer system as used in the p-p bremsstrahlung experiment. The Fortran programs used on-line with the PDP-9 reside in a 56K byte partition of the 360/65.

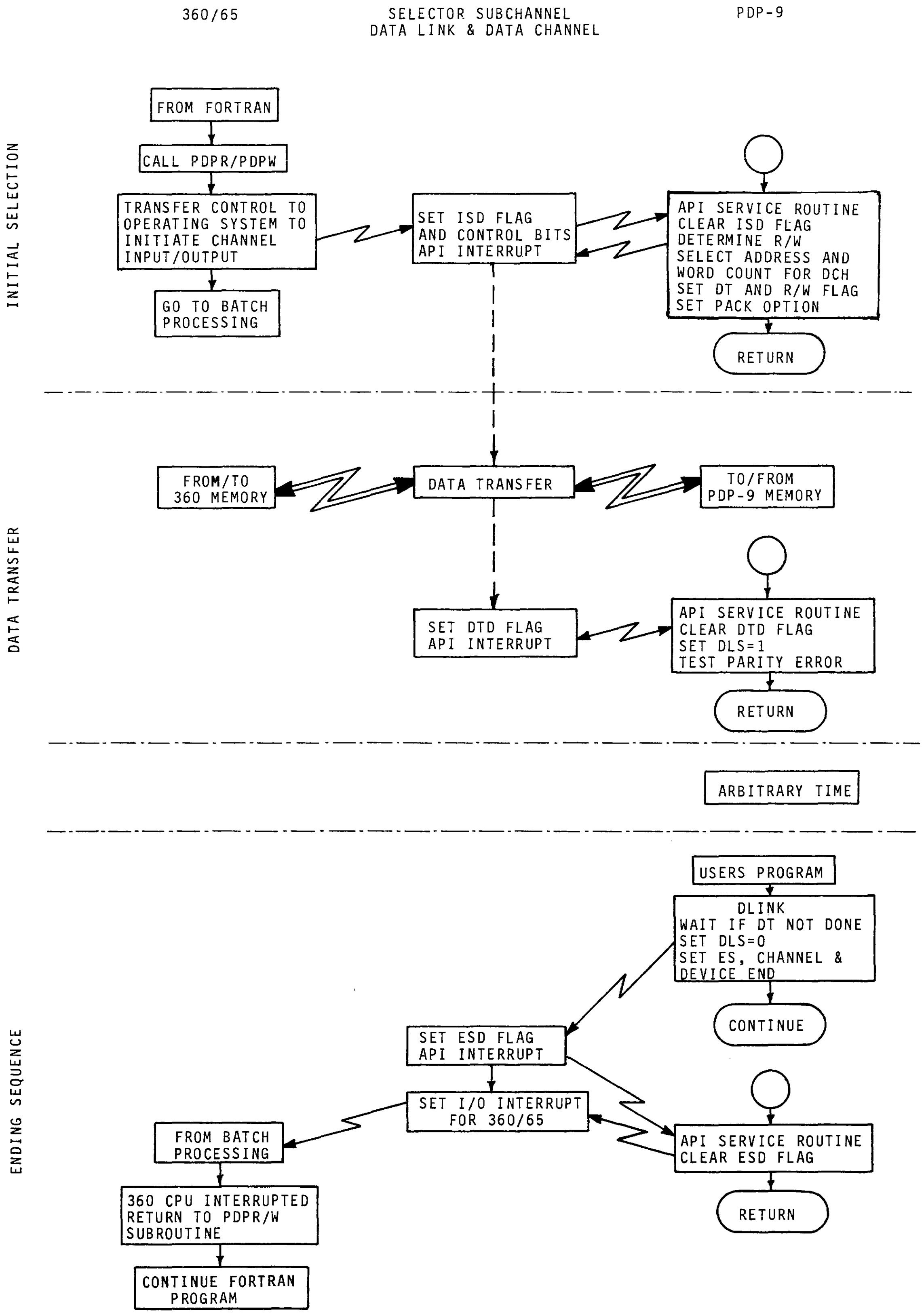


Figure 2 Flow diagram of the basic communication sequence between the PDP-9 and 360/65. Dashed-dotted horizontal lines separate the three stages of the sequence. "Arbitrary Time" is determined by the logic of the users program.

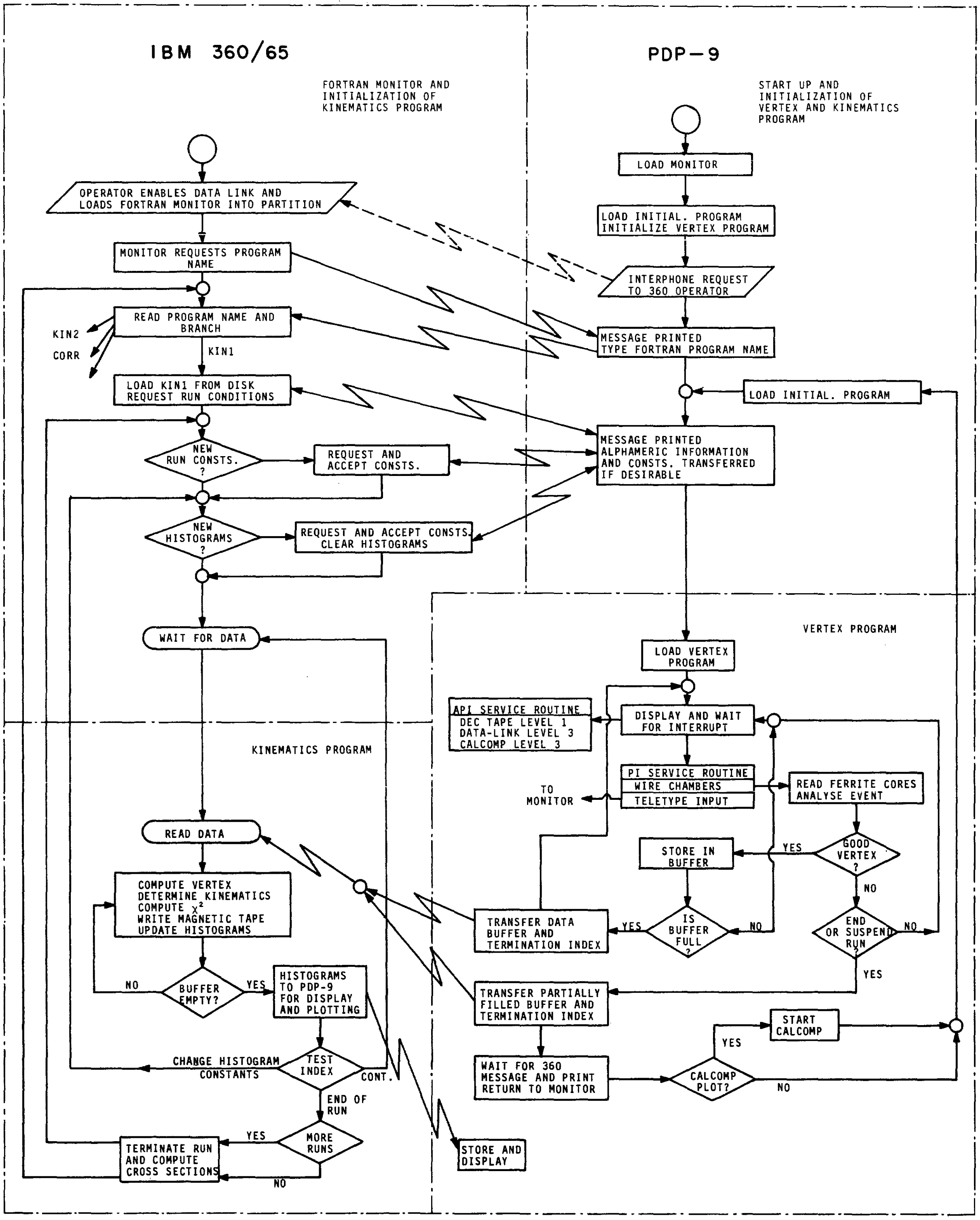

IBM 360/65
PDP-9
FORTRAN MONITOR AND INITIALIZATION OF KINEMATICS PROGRAM
START UP AND INITIALIZATION OF VERTEX AND KINEMATICS PROGRAM
OPERATOR ENABLES DATA LINK AND LOADS FORTRAN MONITOR INTO PARTITION
LOAD MONITOR
LOAD INITIAL. PROGRAM INITIALIZE VERTEX PROGRAM
MONITOR REQUESTS PROGRAM NAME
INTERPHONE REQUEST TO 360 OPERATOR
READ PROGRAM NAME AND BRANCH
KIN2
CORR
KIN1
MESSAGE PRINTED TYPE FORTRAN PROGRAM NAME
LOAD KIN1 FROM DISK REQUEST RUN CONDITIONS
LOAD INITIAL. PROGRAM
MESSAGE PRINTED ALPHAMERIC INFORMATION AND CONSTS. TRANSFERRED IF DESIRABLE
NEW RUN CONSTS. ?
REQUEST AND ACCEPT CONSTS.
NEW HISTOGRAMS ?
REQUEST AND ACCEPT CONSTS. CLEAR HISTOGRAMS
WAIT FOR DATA
VERTEX PROGRAM
LOAD VERTEX PROGRAM
API SERVICE ROUTINE
DEC TAPE LEVEL 1
DATA-LINK LEVEL 3
CALCOMP LEVEL 3
DISPLAY AND WAIT FOR INTERRUPT
KINEMATICS PROGRAM
PI SERVICE ROUTINE
WIRE CHAMBERS
TELETYPE INPUT
TO MONITOR
READ FERRITE CORES ANALYSE EVENT
READ DATA
STORE IN BUFFER
YES
GOOD VERTEX ?
NO
COMPUTE VERTEX
DETERMINE KINEMATICS
COMPUTE χ^2
WRITE MAGNETIC TAPE
UPDATE HISTOGRAMS
TRANSFER DATA BUFFER AND TERMINATION INDEX
YES
IS BUFFER FULL ?
NO
END OR SUSPEND RUN ?
NO
YES
NO
BUFFER EMPTY?
YES
HISTOGRAMS TO PDP-9 FOR DISPLAY AND PLOTTING
TRANSFER PARTIALLY FILLED BUFFER AND TERMINATION INDEX
START CALCOMP
YES
CHANGE HISTOGRAM CONSTANTS
TEST INDEX
CONT.
WAIT FOR 360 MESSAGE AND PRINT RETURN TO MONITOR
CALCOMP PLOT?
NO
END OF RUN
YES
MORE RUNS
TERMINATE RUN AND COMPUTE CROSS SECTIONS
NO
STORE AND DISPLAY

Figure 3 Flow diagram of the main PDP-9 and 360/65 programs as used in the p-p bremsstrahlung experiment.

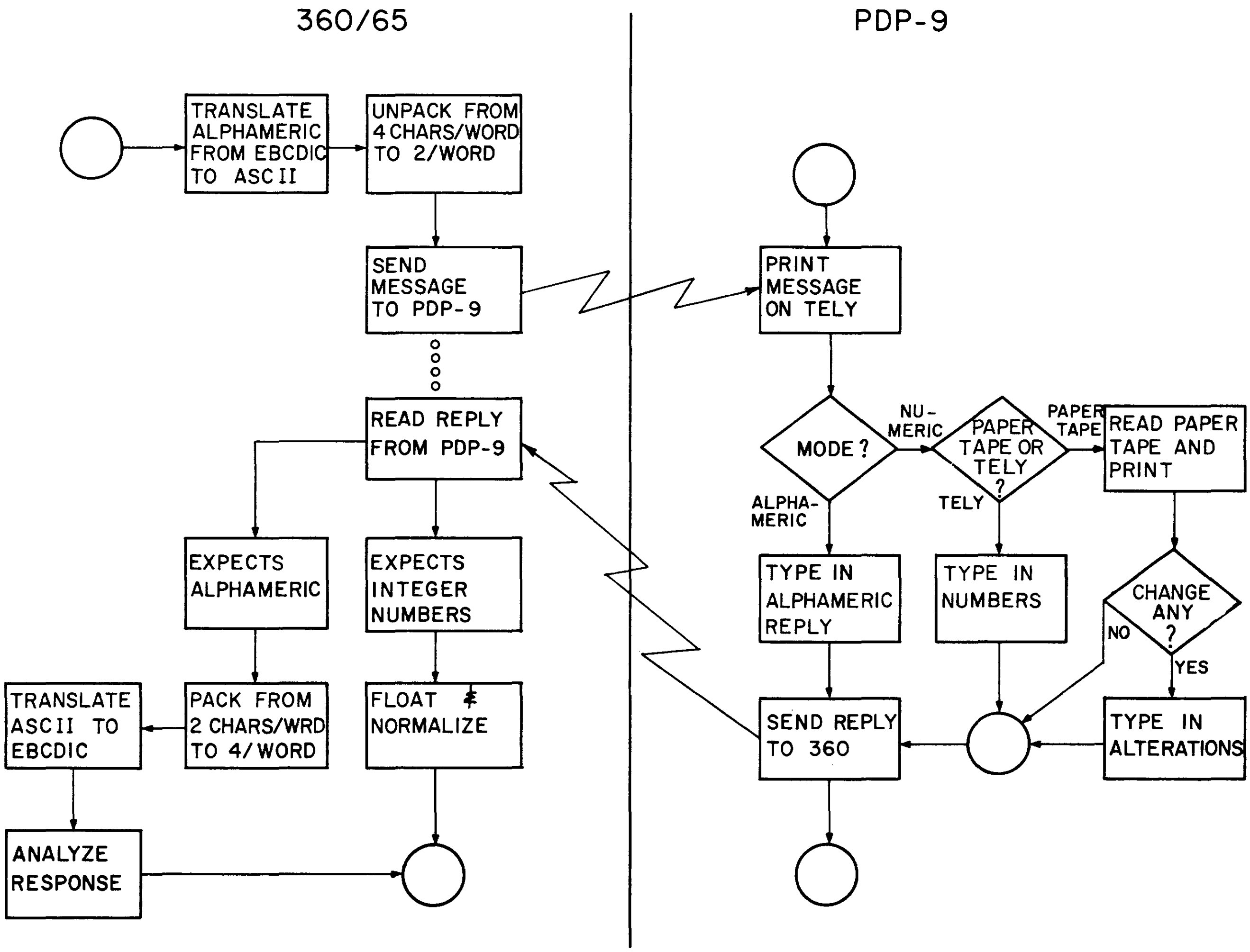


Figure 4 Flow diagram of the basic communication sequence as used in the system described.

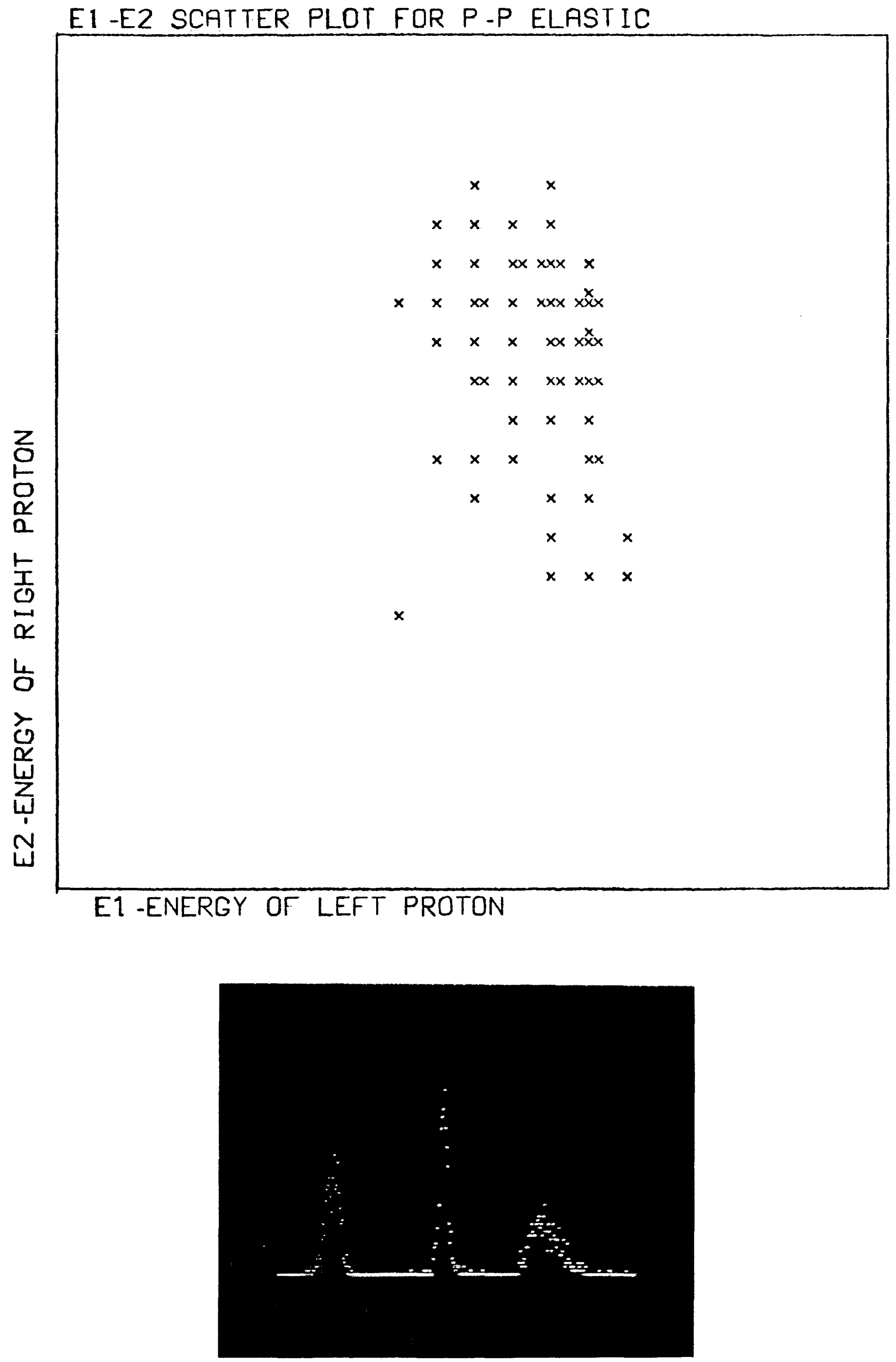


Figure 5 (a) Photograph of three histograms computed on 360/65 by the Kinematics program, returned to the PDP-9 and displayed on the oscilloscope. (b) Two dimensional scatter plot as computed in a similar manner and plotted on CALCOMP plotter. In this case each point represents one event.

AUTOMATIC FILM MEASUREMENT WITH A PDP-9

C. Drum, T. McGrath, R. Van Berg
University of Pennsylvania
Philadelphia, Pennsylvania

ABSTRACT

The University of Pennsylvania's high energy physics group employs a PDP-9 to control and record digitizings from a Hough-Powell flying spot digitizer. It is not possible with the small computer to perform any on-line analysis or tract reconstruction of non-predigitized bubble and spark chamber photographs. Therefore the system relies on CRT displays and simple checking algorithms for monitoring the quality of the digitizing. The system in general and especially the non-standard software and peripherals are described.

Figure 1 shows a more or less typical frame of the bubble chamber film that we are measuring. This format is peculiar to the Penn-Princeton accelerator with a stereo triad as indicated and a BCD data box and strobe mark for automatic frame number recognition. You will note that there are only a small number of tracks in each view; this is basic to the whole philosophy of the series of experiments using the HPD, as the automatic track following programs which do the pattern recognition are still at an early stage of development and are easily confused and greatly slowed up by any large number of tracks. Fortunately the bubble chamber at PPA cycles very rapidly so that it is still possible to obtain a relatively large number of events in a reasonable time.

Our problem, then, is to get precise measurements of the co-ordinates of the tracks and fiducials in each of the three views, and then ship these off to the track following, spatial reconstruction, and fitting programs.

The machine which we use to do the actual measurement, a Hough-Powell, or flying spot, digitizer (HPD), is a large and ungainly mechanical scanner. Figure 2 shows the basic optical set up for one of the two intermeshed orthoganal scan systems -- all focusing and collimatimating lenses, the entire normal scanning system, and most of the mechanical complexity has been omitted, so use your imagination freely. The light from a mercury vapor lamp is collimated onto a pair of small glass fibers that act as crossed cylindrical lenses and bring the light to a point focus. The eight curved fibers (only one is shown) are mounted in a large disk which rotates at about 3500 RPM - thus the moving curved fiber passing over the fixed straight fiber creates a rapidly moving spot or line of light. The light in the moving spot is then split and half of it passes through a ruled grating. As the spot moves across the grating the photomultiplier picks up the variations in intensity and allows one, by counting, to measure precisely (to 1.5 μm) the position of the spot along the scanline. Meanwhile the remainder of light from the spot is bounced around and through the film, which is vacuum clamped to the prism - the variation in output of the video photomultiplier then is a measure of the density of the film and by simply discriminating this video signal and using the fast discriminator signal to interrogate the grating counter one may get the position of a black speck on the film. While the spot is moving left to right (or vertically, in the case of the other channel) the prism to which the film is clamped is being moved hydraulically outward so that by digitizing (to an accuracy of about 2 μm) the position of the hydraulic stage one obtains the other co-ordinate of the speck you have measured.

The reason for having the two scanning systems (the normal one with a vertically moving spot and horizontally moving stage, and the orthogonal or abnormal scan with horizontal spot and effectively vertical stage motion) is that the stage motion is fairly coarse (about 50 μm between scan lines), and one is likely to miss or get very few digitizings from a track running roughly parallel to the spot motion.

The sequence of measuring a picture, then, goes something like this: first move the film into position with the desired frame clamped to the prism in the proper spot (the film is prescanned by human scanners, and frames of interest are recorded on punch cards), secondly, position the mechanical stage at the proper place, then measure in the desired mode (there are a number of options on measuring speed and output format which must be specified) from point A to point B, stop and repeat the cycle at a rate of about 2 pictures per minute. The HPD itself has a large amount of electronic hardware hung on it that performs the mundane operations of stage motion and film motion, as well as the somewhat more exotic tasks of grating counting, video discrimination, and co-ordinate transfer. However, there still must be a general purpose computer to organize and control this hardware.

The computer used in this measuring process must do at least two things - 1) send commands to the HPD to set up for and to initialize a given scan, and 2) accept the digitizing from the HPD as the film is read and store these digitizings somewhere.

Most other HPD installations have a very large computer of 7094 to 6600 or B8500 size, and are thus able to do a good deal of track following and noise rejection - sometimes even rough spatial reconstruction for checking purposes - before outputing the digitizings or a reduced set thereof on magnetic tape. However, we grew up with a 7040 as control and output machine so none of this was possible and digitizings are in our case simply dumped with a minimum of processing onto magnetic tape for later analysis by a large computer. Thus the use of PDP-9 to control the HPD is really a step up for us in terms of speed and cost, though loss of certain versatility of the larger 7040 system is somewhat painful.

The HPD/PDP-9 interface (figure 3) essentially imitates the direct data channel of the 7040. The functions to be performed consist of four major activities; (1) sending commands to position the film on the proper frame (movefilm) (2) commands to position the measuring stage in preparation for scanning (move stage) (3) commands for scanning (set mode), (4) accepting the resulting digitizings. Since commands are not given while digitizings are being received, the two types of data are gated onto a bidirectional bus. Due to the high data rate 1 MHz maximum), the long cables (50 ft.), and our previous success with the 7040 data transmission system, it was decided to use IBM N line drivers and terminators.

Each of the three commands requires a 36 bit word (one 7040 word) which together specify the parameters for measuring one view of a frame.

Commands are issued in two words of 18 bits each, and their function and whether 1st or 2nd group of 18 bits is determined by the subdevice bits. One view then requires 6 PDP-9 words. The HPD is assigned two device selectors which in conjunction with the sub-device bits gives eight possible combinations. The extra two words are used to send useful messages, such as END OF FILM, END OF PROGRAM, and certain alarm conditions to the HPD operator. Data is entered into control registers from the HPD bus by means of IOP pulses. IOP 1 sets the function in accordance with the device and sub-device levels, simulating a 7040 sense output reset, IOP 2 enters the data and simulates a channel ready write. IOP 4 is used to initiate the operation specified by the command and has no 7040 analogy. The main difference between IBM data lines and those of the 9 is that the 7040 waits for a signal from the attached device before removing data from its output bus through one of the DMA channels. Digitizings may consist of three types of data. (1) Frame number of picture currently being scanned, (2) stage position coordinates and (3) spot position coordinates. Registers containing this data are alternately gated onto the HPD bus during read time under control of the HPD read logic. At word count overflow the DMA is inhibited and waits for the program to reinitialize the word and address counters and reset the interrupt flag. At the end of a scan the HPD sets an end of file flag which interrupts the machine and resets the READ SELECT flag. The HPD then awaits the six command words for the next view. At this point it is probably useful to go over the software part of the system.

Presently the overall operation is strongly centered around an IBM 7040. The 7040 has a very sophisticated assembly language system called IBMAP. One of the features of the IBMAP system is the ability to define op codes, pseuod-ops etc. Using this feature we were able to define the entire instruction set of the PDP-9 plus our own unique instruction codes for the HPD and for the drum. The reasons that we chose to pursue the use of the 7040 assembly programs rather than the PDP-9 software packages were the following. The first, and main, reason was that the 7040 was available and we were able to write, assemble, and partially debug our major PDP-9 programs before our own PDP-9 was installed. Second, the powerful I/O devices of the 7040, especially the 600 line/minute printer gave us a tremendous advantage over our teletype for assembly listings etc. And, third, our familiarity with the MAP language, plus our inherent reluctance to make a clear break with a working system. As of this report, the system has worked our extremely well although several problems have cropped up; mainly the strong possibility that the 7040 will be phased out of the University sometime this summer. And also the fact that as far as our own group is concerned the available time on the 7040 has been greatly reduced.

The operation of the assembly system is as follows: The users source deck or decks are assembled by the IBMAP system and loaded into 7040 core. (It might be interesting to note that at this point that the programs, especially large programs can be divided up into sub-programs, sub-routines and assembled independently. Thus at load time an overall deck may contain a source deck to be assembled and binary decks of sub-programs, sub-routines that were assembled previously.) The linking, etc. and actual loading is done by the IBLDR section of the system. When the entire program has been loaded into 7040 core, an auxiliary program called MTAPE loader written by our group is called into execution. The function of MTAPE is to translate the program in core from 7040-36 bit instructions to PDP-9-18 bit instructions. The translated instructions are placed on magnetic tape with a format very similar to the rim format Paper tape with two 18 bit words required for the storage of each instruction. The actual time of translation for a program that occupies approximately 8k of PDP-9 memory takes only several seconds of 7040 time. Our PDP-9 is physically located in the same building as the 7040, and therefore it only takes a few minutes to take the tape from the 7040 and prepare it for input on the PDP-9. To load the magnetic tape on the PDP-9 we use a several hundred word hardware read-in paper tape program.

Presently our HPD operating system consists of a resident monitor, and a system tape. The function of the monitor is to call into execution from the system tape any one of n possible programs. The three programs presently on the tape are: utility program for tape copying, tape dumping; the HPD control program; the off-line display program; and fourth, data, giving desired frame numbers and other bookkeeping information which is required by the HPD control program. The present bubble chamber experiment requires that the film be pre-scanned on manual scanning machines and a minimum of information recorded on magnetic tape. For example, certain events with wide angle interactions will have to be scanned twice; one along the incoming beam tracks, second, perpendicular to the incoming beam. A flag indicating whether or not this is required will be fed to the control program as part of the above data.

The control program must issue commands to the HPD to position the film on the desired frame on the basis of frame numbers read by the HPD and transmitted to the PDP-9. While the correct frame is being found commands must be issued so that the stage is positioned to digitize the desired view within the frame. During this experiment, where every frame has three views, at least three scans must be made of each desired frame, and some frames will require an additional orthogonal scan for each view or a total of six scans per frame. When the stage is in proper position and the correct frame found, the control program sends a command to begin digitizing. The control program has three input buffers, each 1040 words long to receive HPD digitizings. The three buffers are filled in a rotating manner and are processed by a data checking program, a CRT scaling program, and the output tape writing program. The buffering scheme operates in an asynchronous manner. This is, the rate at which the buffers are filled by the HPD does not necessarily have to be the same rate at which the data is being processed and written on output tape. However, there lies the danger that on extremely dense portions of the picture the digitizing rate may exceed the ability of the PDP-9 to process and output data and produce a buffer overflow. To avoid this condition, we originally started out with four cycling buffers then reduced the number to three and even in experimenting with two have encountered buffer overflow only occasionally due to the extremely fast processing speed of the PDP-9.

To ensure high quality output from the HPD it is essential that some means be provided for monitoring the digitizings. Lacking a large on-line computer, we have found a CRT display of digitizings (figure 4) almost as good. However, the control program occupies most of 8k of core so that a simple scope display is not feasible. We have therefore added a Vermont Research magnetic drum to the system for temporary storage of coordinates to be displayed, automatic refresh, and for convenience in storing programs for quick recall. The main feature of the drum interface is its ability to output data directly to a 34D scope interface without tying up the PDP-9. In the display mode data is written as 18 bit words with bits 8-17 containing the actual scaled coordinate. Bit 0 is used as a flag for stage co-ordinates and bit 1 on signifies end of data. During display the output of the drum shift register is gated directly into the 34D interface instead of into the DMA. The control program scales incoming data and then writes the scaled data on the drum while simultaneously writing the unscaled digitizings on tape. When the entire picture is on the drum or when the allotted number of tracks set aside for scaled data has been filled, instructions are sent to the drum to display the drum stored information. Displays can be held while the HPD keeps digitizing frame after frame or interesting sections of pictures can be expanded digitally on the CRT. One very useful feature of this display is in the setting up of different HPD parameters; potentiometers can be adjusted and then the result seen immediately by measuring and displaying the same frame over and over while adjustments are being made. The data is also monitored internally by the program where it is checked for monotonicity, scan line separation, etc. The output tape, besides containing digitized information, also contains various records summing all errors detected in checking the digitizings, number of records per view, the scan line separation, and various other useful information.

The next step in the chain of operations is processing the digitized tapes. This brings into play an extremely large basically fortran written program called ATF, automatic track finding program. Currently this is being run on two computers one located on campus, IBM 360/65 and the other at NYU-the CDC 6600. The present processing rate for an event on the MOD 65 is approximately 30 seconds. The output from the ATF program, the coordinates of interesting tracks, are fed into two kinematic programs called thresh and grind which hopefully yield the final physics output on the event.

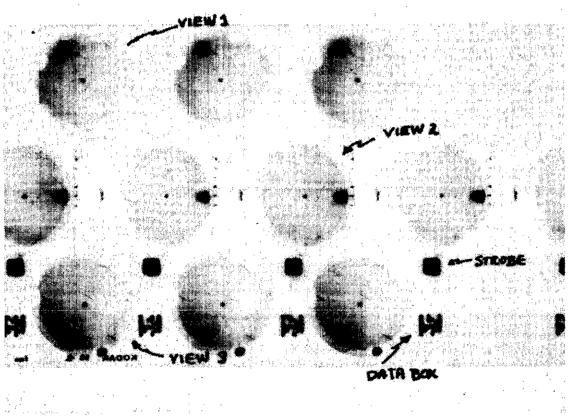


Figure 1 PPA bubble chamber film

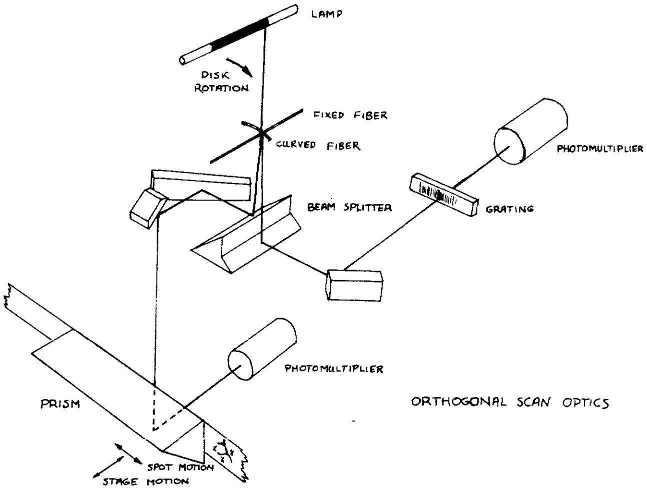


Figure 2 Simplified HPD scan optics

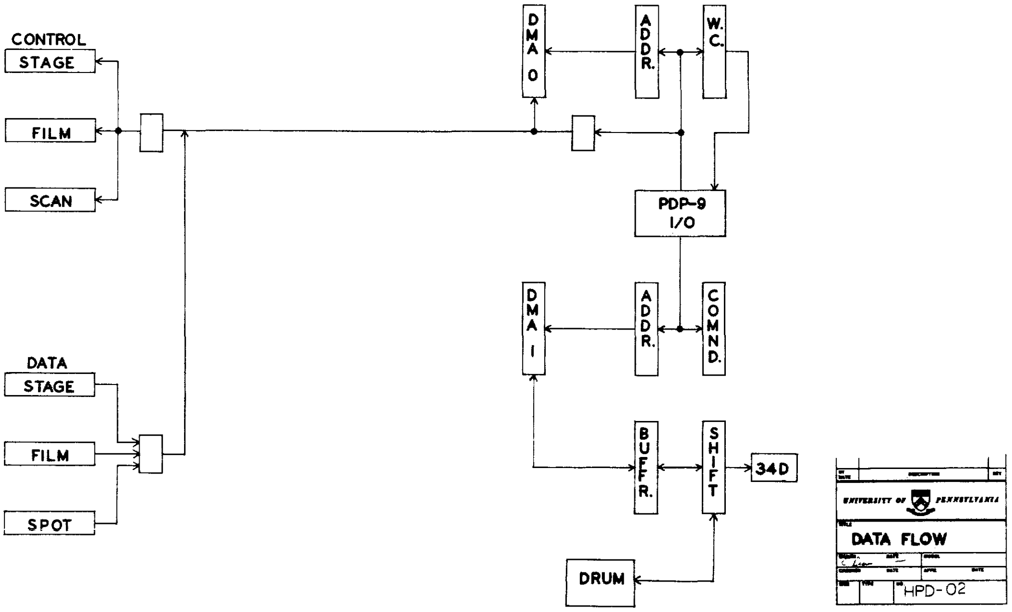


Figure 3 PDP-9 - HPD, drum interfacing block diagram

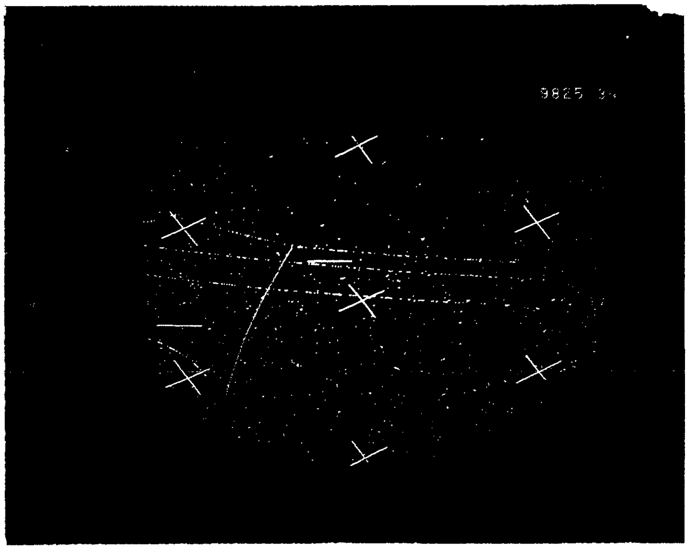


Figure 4 Typical CRT display with frame and view numbers

A COMPUTER-CONTROLLED SYSTEM FOR AUTOMATICALLY SCANNING AND INTERPRETING PHOTOGRAPHIC SPECTRA

C. A. Bailey, R. D. Carver, R. A. Thomas, and R. J. Dupzyk
Lawrence Radiation Laboratory, University of California
Livermore, California

ABSTRACT

In analytical spectrography, the most time-consuming portion of an analysis is the scanning and interpreting of the photographically recorded spectra. A system has been devised to shorten this time considerably by using a small digital computer to control the scanning densitometer and subsequently to calculate abundances from the photographic data.

The following description applies specifically to spark-source mass spectrography; however, adaptation to other systems would be relatively straightforward. A typical photoplate from our spectrograph contains several thousand lines from as many as twenty graded exposures, and represents approximately sixty-five elements. Starting with the most intense exposure, the optical transmission of each line is measured using a Grant microphotometer. These transmissions as well as the position of each line are stored in a PDP-8 computer. The computer initiates and completely controls the scanning, and simultaneously converts each line position to an exact mass number from a calibration performed at the beginning of the scan. The computer is programmed to distinguish between lines and empty areas on the photoplate, and all the graded exposures of each line are recorded before the scanning continues to the next line. Backgrounds are continuously upgraded and recorded along with their adjacent line densities. After the desired area of the photoplate has been scanned, an emulsion calibration is calculated from the data stored in the computer. Then all line densities on the linear portion of the calibration curve are converted to ionic abundances. Total time involved in scanning twenty exposures on a fifteen inch photoplate is now approximately five hours.

The Lawrence Radiation Laboratory has had a CEC 21-110 Spark Source Mass Spectrometer in operation for five years. Two years ago the chemists and electrical engineers started looking for a way to simplify and improve the reduction of the data which is taken on photographic plates.

Several available systems of both plate reader and densitometer were looked into and found to have hard wired programs not applicable to what we wanted to do. We concluded that we had to design the system ourselves.

The Grant Microdensitometer was found to be an excellent plate reader capable of being operated by computer controlled stepping motors. The densitometer was coupled to a Digital Equipment Corporation PDP-8 computer to construct our present system.

The first component in the system is a Spark Source Mass Spectrograph, CEC 21-110. The ions from a sample are collected on a 2" x 15" glass photoplate. The mass lines occurring on the plate come from elemental and compound ions, singly and multiply charged. Twenty graded exposures can be collected on a plate. Each exposure is 2mm in length. For our purposes "exposure" means the number of ions striking the photographic plate; this value is expressed in Coulombs. The second component in the system is a Grant Microdensitometer. It has an accuracy of ± 1 micron in the X and Y directions. The densitometer is stepped in one micron increments in the X direction and five micron increments in the Y direction. There are two viewing screens available. One is a projection through a 22 power zoom lens which allows a closeup of the area being scanned. The second screen is an analog signal from the detector displayed on a cathode ray tube. It is used for adjusting the scanning slits.

The readout is an analog-to-digital converter (ADC) connected to the photomultiplier output. The ADC is capable of twelve-bit resolution with a digitizing time of 35 microseconds.

The actual plate stage drive is accomplished by using two hundred steps per revolution digital stepping motors. The X-drive incorporates a 5:1 gear reduction which results in a scale of one micron per step. No gear reduction is used on the Y-drive, therefore the scale is 5 microns per step. Motor speed was selected to allow instant start-stop-reverse response without resorting to controlled speed-up to compensate for inertial effects.

The interface from the PDP-8 to the microdensitometer uses only one device selector. The device selector, by program control, puts out any one, or a combination of three IOT pulses.

IOT-TWO is used to transfer nine status conditions, such as X ready, Y ready, + X limit, etc., into the accumulator. All but two of the status conditions set the flag. IOT-FOUR actuates any one of nine different control functions depending on what bit or bits are set in the accumulator. Typical control functions are, step + X, step -Y, clear flag, interrupt on, etc.

The interface provides interrupt cap^bility, but it is not being used with the present program.

The third component is the Digital Equipment Corporation PDP-8. It is the 4k version equipped with two Dectape transports and the extended arithmetic element (EAE). The code occupies all of core. The floating point package-D is used for all the input, output, and most of the calculations. Control and operation of the system is done entirely with software.

The following describes the interaction between the operator and the Grant/PDP-8 system. The emulsion calibration for the plate is made by exposing the plate to the rhenium isotope spectrum at varying intensities, changing the magnet setting between exposures in order to offset the lines. These line density-ratios are fitted into the Hull equation relating plate density to exposure (Fig. 1). The exponent R is calculated to be used later to calculate relative abundances of the ions striking the plate.

The input for this process is through the teletype (TTY). The operator types the number of rhenium lines to be read and a value for the transmission at infinite exposure (T_{inf}).

When the specified number of lines has been read the computer waits for the operator to realign the stage to read from low mass to high mass over the range he desires. When alignment is complete the operator types in a starting and a final location in microns, four mass numbers, and their locations on the plate. These masses are carefully chosen prominent lines distributed across the range of interest and are fitted into a mass number calibration equation (Fig. 2). "A" and "B" are determined and used later to calculate the mass number, of each unknown line from its location on the plate. Twenty exposure values are typed in corresponding to the number of Coulombs each line was exposed. At this point the computer is put in complete control, and the operator is no longer needed.

The following is a description of the peaks encountered on a typical photographic plate and the way the computer code handles them.

The first peak is a singlet in a clean portion of the plate. The peak reading process always starts with the most intense exposure. The code calculates a peak detection threshold value. This value is an arbitrary decrease in the transmission calculated from the background values (Fig. 3). This value is updated every one ninth of a mass unit if a peak has not been detected. A peak is determined by testing to see if the incoming transmission value is below the peak detection threshold. If a peak is detected it is checked to see if it is an honest peak. This check is necessary because of noninformity in the emulsion. If it is an honest peak its peak top is determined and two background values one ninth of a mass unit on either side of the peak top are taken. One ninth of a mass unit is an arbitrary distance selected to reach over the very dense lines and yet not encounter other mass lines from multiply charged ions. There are not a constant number of exposures for each mass line, so the following method is used to determine when all the lines for a individual mass have been read. Two questions are asked of the PDP-8. 1) Have twenty exposures been racked? If yes then it does not look for more information; if no the second question is asked. 2) Is the transmission of this peak top below a threshold value calculated from the background values of the previous exposure? If yes, then the stage is racked one exposure in the positive Y direction, the direction of lesser exposure, and the data are stored in a buffer in core. If no, then all the data from that line have been measured and the code racks back to the most intense exposure and continues to look for a new mass.

The second peak is a doublet. The first mass line encountered in a doublet is read in the same manner as a singlet except that the background values are now taken at one ninth mass unit on one side and at the minimum in the valley between the peaks. The computer is aware that there is a mass on its high mass side. When it has finished the first mass line it then scans the high mass line. Because the various exposures are not perfectly perpendicular to each other it was necessary to make sure that the stage returned to the peak top of the most intense exposure of each mass line encountered before starting to look for a new mass line. This was especially necessary in the case of doublets.

The third peak is one that lies in fog. Fog is caused by the ions from a very abundant element scattering due to residual charge build-up. Fog on plates may have a transmission value as small as 5%. The process of updating the peak threshold detection value every one ninth of a mass unit described earlier allows very light lines to be found in the very dark fog. The process of checking to see if the peak top value is darker than the threshold calculated from the previous exposure allows all the exposures for a mass line to be read.

The entire plate is read and the data are stored on the Dectapes in two block segments. The following information is stored: The mass number, from eq. 1 the transmission at the peak top, the transmission at the two background positions, and the exposure value.

The teletype output is shown in Figure 4, 1) The percent transmission of the Re 185 and Re 187, 2) the R values for each Re ratio, calculated from the Hull equation, 3) and the average R value.

Following this are the data for all detected masses on the plate typed in the following fashion (Fig. 5), 1) The mass number, 2) the percent transmission of the peak top and the two background values, 3) the abundance of the peak top and the two backgrounds and a net abundance, 4) the average abundance and the percent error. The average abundance is a weighted average so that all exposure values could be used in the calculation. The weighting function is parabolic.

It was found that the numbers from the Grant/PDP-8 system agreed with hand calculation to within 5%. This is very good agreement as spark source work is usually quoted to a factor of two. There is no reason to believe that the Grant/PDP-8 numbers are not better. Standard samples are difficult to prepare.

To conclude let us compare the time spent in reading and reducing the data from one plate. Starting from the time that the plate is developed, dried, and ready for reading until relative abundances are in hand: 1) The Grant/PDP-8 system takes one half hour of operator time, and for a typical geological sample about eleven hours of system time, approximately 5 hours for reading the plate and 6 hours for typing the data. It can be set up before closing time and answers are ready at opening time the next morning. 2) With the standard system an operator is almost always required to put the data out on chart paper, convert it to IBM cards and run a short code on a CDC 3600 taking a total time of three or four days.

The Grant PDP-8 system is a great improvement over the old system. It should be adapted readily to other plate or film reading systems.

We would like to thank Dr. G. W. Barton, Jr. for his help in deriving the equations to weight the average abundance and to calculate a percent error for the average abundance.

This work was performed under the auspices of the Unites States Atomic Energy Commission.

$$\text{Emi} = \left[\frac{T_o - T}{T - T_{inf}} \right]^{\frac{1}{R}}$$

Figure 1 Hull Equation for Emulsion Calibration

$$\text{MASS} = \left[\frac{\text{Loc} - A}{B} \right]^{2}$$

Figure 2 Mass Calibration Equation

Threshold = Average Bkg. - (% * Average Bkg.)

Figure 3 Threshold Equation

```
%T RE 185 & RE 187
+Ø.4749693E+Ø2       +Ø.3592184E+Ø2
+Ø.49Ø1Ø98E+Ø2       +Ø.363614ØE+Ø2
+Ø.5264959E+Ø2       +Ø.393650ØE+Ø2
+Ø.5235654E+Ø2       +Ø.4119656E+Ø2
+Ø.56Ø1952E+Ø2       +Ø.4483516E+Ø2
+Ø.648596ØE+Ø2       +Ø.5196582E+Ø2
+Ø.6923Ø75E+Ø2       +Ø.56Ø4394E+Ø2
+Ø.691Ø865E+Ø2       +Ø.5633699E+Ø2
+Ø.8329668E+Ø2       +Ø.7479852E+Ø2
+Ø.78Ø2197E+Ø2       +Ø.6656897E+Ø2

R VALUES
+Ø.946730Ø2E+ØØ
+Ø.1Ø2846ØE+Ø1
+Ø.1Ø62Ø21E+Ø1
+Ø.8881528E+ØØ
+Ø.8849648E+ØØ
+Ø.1Ø49275E+Ø1
+Ø.1114296E+Ø1
+Ø.1Ø79682E+Ø1
+Ø.1Ø14555E+Ø1
+Ø.1131831E+Ø1

AVE R VALUE +Ø.9789795E+ØØ
1/R VALUE +Ø.1Ø21472E+Ø1
```

$$R = \text{Log} \left[\frac{\dfrac{T_o - T_{187}}{T_{187} - T_{inf}}}{\dfrac{T_o - T_{185}}{T_{185} - T_{inf}}} \right] \div \text{Log } (1.67)$$

Figure 4 Rhenium Data and R Value

```
MASS  +Ø.3127783E+Ø2
%T                  %B1                 %B2
+Ø.1582418E+Ø2      +Ø.2749695E+Ø2      +Ø.242ØØ24E+Ø2
+Ø.2Ø61Ø5ØE+Ø2      +Ø.3262516E+Ø2      +Ø.3ØØ61Ø4E+Ø2
+Ø.27Ø3295E+Ø2      +Ø.4163136E+Ø2      +Ø.3748472E+Ø2

A(T)                A(B)                A(B)                A(NET)
+Ø.3143612E-Ø4      +Ø.149130ØE-Ø4      +Ø.1787448E-Ø4      +Ø.15Ø4238E-Ø4
+Ø.445ØØ17E-Ø4      +Ø.23Ø9754E-Ø4      +Ø.2616Ø48E-Ø4      +Ø.1987115E-Ø4
+Ø.6113529E-Ø4      +Ø.3Ø88Ø63E-Ø4      +Ø.3698Ø68E-Ø4      +Ø.272Ø463E-Ø4

AVE ABND  +Ø.2337621E-Ø4     PCT ERROR  +Ø.1ØØ89Ø4E+Ø2
```

$$\text{Emi} = \left[\frac{T_o - T}{T - T_{inf}} \right]^{\frac{1}{R}}$$

$$\text{Abund} = \text{Emi} * \sqrt{\text{Mass}} \div \text{Exposure}$$

Figure 5 Data Output Mass, Percent Transmission for Mass, and Relative Abundance for Mass

A PDP-8 SYSTEM FOR BUBBLE CHAMBER MEASUREMENTS*

John Rayner
University of Maryland
College Park, Maryland

ABSTRACT

This paper describes an on-line measuring system in which the PDP-8 is used both as an up-down scaler for an image plane digitizer and to supervise the measure in an attempt to prevent the most common measuring errors. This error prevention is accomplished by having the program institute most of the necessary procedures through messages to the measurer on a Teletype and by elementary checking of the input data. Another aim of the system is to replace cards with IBM compatible magnetic tape as the output medium. To this purpose a Digi-Data Stepping Recorder has been interfaced to the PDP-8. It is planned to expand the system to four measuring stations in the future.

The PDP-8 measuring system is a low cost approach to the film measurement bottleneck in high energy physics. Unlike PEPR which aims at replacing the human operator, the PDP-8 system is an attampt to increasing her efficiency by catching the more common errors on the spot, and by speeding up the measurement process.

The error prevention objective is approached in two ways. First most operator actions are initiated by the computer through messages to the operator on an on-line teletype. Second all input to the system is checked by the computer for format and consistancy.

The measurement speedup is achieved in several ways. The greatest gain in speed is probably due to the use of the computer to buffer the input, thus avoiding the wait for punching the coordinates. In line with this the program is designed to read in each measurement within 70 μs, theoretically allowing measurements to be taken on the fly. It is also hoped that routing all control and paramter information through the teletype will lend itself to simplicity and speed. Finally the choice of an image plane digitizer simplifies the process of moving from point to point on the film.

The present system consists of a PDP-8 computer; a mangiaspago biradial image plane digitizer mounted on an overhead projector scanning table; a teletype for operator-computer communication; a Digi-Data 1420, 556 character inch, 200 step per second, stepping recorder; an on-line clock for program timing; and the necessary interfaces.

To keep the cost of the system as low as possible, the PDP-8 is used as an up-down counter for the measuring machine. This is accomplished through the increment feature of the Data Break. Due to the nature of the optical encoders and the mechanics of up-down decoding it is convenient to keep the two low order bits of each coordinate in the interface. Overflow or underflow of this count of four causes one of four memory locations in the PDP-8 to be incremented. The true displacement can then be calculated by subtracting the down counts from the up counts, and adding in the low order bits, which are read in from the interface when a measurement is made. Since one 12 bit PDP-8 word is insufficient to accumulate the largest possible displacement the scalers are periodically checked for overflow under control of the clock. When overflow is detected an overflow counter is incremented. To avoid the problem of the counter contents changing during read-in for points taken on the fly, two sets of four locations are used alternately.

The digital stepping recorder was chosen for its relatively low cost, and simplicity in interfacing and programming. The tape deck itself has automatic lateral and longitudinal parity, inter-record and end-of-file gap generation, and an optional read head an electronics. The interface converts between the 6-bit tape format and the 12-bit PDP-8 word.

The clock which can be enabled and disabled under computer control has a 880 cycle rate for compatibility with projected serial teletype interfaces, which will be used when additional measuring machines are added to the system.

When the program is started the computer asks for the date and then waits for the operator to type it in. The system is then waiting for a measurer to sign on. This is accomplished by typing S. The computer responds by typing SIGNON and asks for the measurers ID number and then the roll number. The number can be changed between events by typing R, to which the computer responds by typing ROLL and waiting for the new roll number to be typed. To measure an event the measurer types N. The computer then types NEXT EVENT and asks for the various event parameters, such as frame number and event type. Since the event type number is geometry dependent the computer can calculate the number of tracks to be measured. The computer then types VIEW I and requests a home measurement. This is accomplished by placing the measuring head in a slot fixed to the

*Work supported in part by the U.S. Atomic Energy Commission.

table and pressing the record foot switch of H on the teletype. This serves to zero the counters. The counters can then be checked by placing the head in the home slot and typing H. If the counters are within a tolerance limit of zero the computer types HOME OK, otherwise, it types BAD HOME and requests remeasurement of the view. After the home check the computer requests fiducial measurements. When three fiducials have been measured the computer types TRACK I. When track 1 has been measured the measurer types T. The computer then types TRACK 2 and the procedure is repeated until all tracks have been measured. The computer then requests a home measurement to check the counters. If they check, the same procedure is then followed for views two and three. At any time the last measurement, track, view or event can be deleted by typing D followed by M, T, V, or E. When the measurer is finished she signs off by typing E.

The program is completely interrupt driven, that is, all program action is initiated by an interrupt from a peripheral device. Interrupts are handled by a two-phase multipriority interrupt monitor. The first phase of the monitor, which acts with the interrupts disabled, saves the trapped address and then determines which device initiated the interrupt and set a flag for phase two. The second phase then transfers control to the various service routines in the order of their priority. Each device has a unique priority and interrupts by high priority devices are completely serviced before those of lower priority. If a high priority interrupt occurs during the processing of one of lower priority, the high priority interrupt is serviced before processing is continued on the low priority one.

Teletype output is taken care of by two subroutines: a teletype service routine and a teletype monitor. The service routine can output either single ASCII characters or messages stored in a six-bit stripped code. These characters and messages can be queued up by the teletype monitor, thus permitting concatenation of phrases, words, and individual characters with little cost in time to the calling program since it doesn't have to wait for the teletype to be free. When the service routine finishes outputting one message it picks up the next one from the queue until the queue is empty.

The teletype input routine performs two functions: it accepts commands from the measurer in the form of single letters, and it accepts and stores parameters. On receiving a command letter the routine looks it up in the current command table and transfers control to the appropriate routine which types a response and initiates the required action. If the letter is not in the current table, the routine types a question mark and exists. Since there are several command tables it is possible to limit the legal commands to those that are appropriate at the time. In the parameter mode the routine accepts a string of digits, and stores them in a temporary stack. On receiving a terminating character the digits are packed two to a word with leading zeros inserted and transferred to the main input/output buffer. After the last parameter in a group, control is passed to a routine which initiates the next required action. The address of this routine, as well as the number of digits in each parameter and the message requesting it, is stored in a table, thus, allowing the input format to be easily changed. If the input character is not a digit or terminating character a question mark is typed and the temporary stack reset.

Tape output is initiated by the tape monitor and carried out by the tape service routine. To conserve space measurements are output a track at a time. The tape output is swing buffered. That is there are two buffers which interchange roles as input and output buffers. Each output is in the form of a 7094 FORTRAN IV binary record. The record consists of a heading block specifying the type of record, and a possibly void, parameter or measurement data block.

The measurement service routine assembles the coordinates from the up and down counters, the overflow counters and the two least bits read from the interface. Also read from the interface are three bits indicating which views are turned on. These are used to prevent measurement of the wrong view. The home measurements are only used to check the counters and are not stored while the fiducial and track measurements are propagated to the input/output buffer. Internally the coordinates are kept in three word floating point form for the floating point interpreter, but they are stored in the output buffer in fixed point form, with the coordinates packed in three words. We hope in the future to fit a curvature to each three consecutive points, and by requiring that each curvature be within some multiple of the standard deviation from mean, to reject poorly measured tracks. In addition the fiducials can be checked by comparing the interfiducial distances with their nominal values.

STRIP, A DATA DISPLAY AND ANALYSIS PROGRAM FOR THE PDP-8, 8/I

John C. Alderman, Jr., Research Engineer
Georgia Institute of Technology
Nuclear Research Center
900 Atlantic Drive
Atlanta, Georgia 30318

ABSTRACT

This program, using the PDP-8, high speed paper tape reader, and type 34 display, accepts paper tape data listings and displays the result on the display unit. Some elementary computations are made on the data and are also displayed. The program is deliberately designed to be open-ended, and most users will want to add features peculiar to their own problem. Almost all functions are carried out in subroutine form, and these subroutines can be called either from the keyboard or within another subroutine.

INTRODUCTION:

At the Georgia Tech Nuclear Research Center there are in progress a number of small scale experiments, each involving several graduate students. All of these experiments use a data acquisition system which includes an on-line PDP-8. Our need is for a data processing system which will produce clearly interpretable results from the experiment in a relatively short period of time, since otherwise the apparatus may not be available for a repeat of the experiment.

Since most of the experiments take data as a function of some equal increments of an independent variable, a straightforward data display and reduction program has been devised for use with the type 34 display unit.

Two programming assumptions have been made:

(1) While computers are relatively good at doing computations, they are singularly unimaginative in making decisions; while graduate students may be capable of doing the computations, they are singularly unwilling to do so.

Consequently, the present version of STRIP depends on the computer for almost all of the calculations, and the user for all of the decisions.

(2) Any programming system which is to be used by several groups, must be easily expanded in order to change and/or add functions to the original system. In the case of inexperienced programmers in particular, these changes and additions must be facilitated to the extent that the user can make the needed changes without spending a great deal of time learning the nuances of sophisticated assembly (PAL) language programming.

These considerations led to the development of STRIP, a PDP-8 program which produces a two-dimensional display with the independent (equal-increment) variable along the horizontal (X) axis, and the dependent variable along the vertical (Y) axis. Also included in the display is the result of some elementary numeric computations on the displayed data (i.e. the address of the maximum, its value, and the area under the displayed curve). These numbers can be used by the operator to determine parameters for later calculations.

In order to optimize data handling and display, two buffers are used. One contains the original data and the other data to be displayed. The display routine continuously circulates through the later, refreshing the display at a rate of about 20 times a second.

In the current STRIP version the operator/user manipulates the parameters of a calculated Gaussian to fit his data. This is especially useful since many types of experimental data show such a Gaussian distribution, and the parametric form is desired for further data reduction. Since the fitting operation is accomplished by the user implicitly, the background does not have to be specified explicitly, simplifying the operation of obtaining the Gaussian parameters themselves.

Data Storage

The data for the program are stored in two buffers in the computer memory. The floating point data buffer contains each value of the original data stored in a 3 word floating decimal point format, as used by the standard Floating Point Packages. These data are used as the basis for most of the computations, but are not disturbed by these computations (exceptions are the input routine "R", and the permanent Gaussian subtraction routine "#"). The display buffer is stored in a 10 bit one-word integer format, suitable for deposition into the Y axis register of the type 34 display unit. The display routine cycles through this buffer displaying each point in turn while incrementing the horizontal axis by the appropriate horizontal step size.

A feature of the display routines is that as the display buffer is "built" by making computations on the data in the floating point buffer, the result is normalized before conversion to the 10 bit integer which is stored in the display buffer. Thus the display always occupies the maximum vertical displacement on the screen. The routine that calculates the data display also normalizes the horizontal axis step size to make maximum use of the screen.

Keyboard Monitor

The keyboard monitor interprets the characters struck by the operator, and calls the corresponding subroutine from a table of starting addresses stored in page zero. The list of legal characters is expandable, and terminated by a zero. The display routine is incorporated into the keyboard monitor flag test, such that the flag for the keyboard is tested after each look through the display. The display is refreshed about 20 times a second (depending upon the number of points displayed). The most time-consuming operation of the display is the generation of the title, and a NOP can be inserted in the calling location for the titles subroutine, if desired.

The keyboard monitor presently recognizes a number of control characters, which are listed as Table I. The functions are self-explanatory, and the user will become familiar with them very quickly.

Usage

Table I

STRIP CONTROL KEYS

KEY	FUNCTION
L	Lower Boundary Marker
U	Upper Boundary Marker
C	Change to New Boundaries
F	Fetch Between Boundaries
D	Reset Boundaries
R	Read Input Data
S	Strip Trapezoid
J	Display Gaussian
G	Subtract Gaussian (display)
H	Get Gaussian Parameters
CTRL+ BELL	Permanent Upper Boundary
CTRL+ C	Return to Monitor (".")
#	Subtract Gaussian (data)

Let us assume that data has been entered into the data buffer (by using the R command), and that the shape of the observed peaks is a true Gaussian, obscured by noise. (See Figure 1). In order to begin with some reasonable values for the Gaussian parameters, let us narrow the limits by typing an:

```
     L=+  1  102

     U=+ 160  150

     C         (See Figure 2)
```

Now we have narrowed the display to two peaks. Since the taller of the two peaks is the "MAX" on the display, and the endpoints of the display look as if they are on the flat portion of the background we strike the "S" key. This causes the trapezoidal area between the zero reference and the value of the data at the absissa of the end points to be subtracted from the data. (See Figure 3). Notice that the display is renormalized to fill the screen. The new "AREA" and "MAX" are valid for the subtracted display. Notice that nothing has been done to the data in the "data buffer" (as you can discover by striking the "F" key, returning the display to its previous result by again hitting "S"). Now enter the subroutine that gets the Gaussian parameters by striking the "H" key. The program types out (in floating point E format) the current Full Width Half Maximum, and waits for a new value, or some non-numeric character. The standard deviation and the current value of the peak height are typed, and again the program waits for a new number. When the first non-numeric character is typed, the current value of the location of the peak (in units of channels, but not necessarily integer values of the channel number!) is typed and a new value accepted. When the next non-numeric character is typed, the area is computed and typed, and the program returns to the keyboard monitor. Note that there is no change in the display (See Figure 4).

In order to get some idea of the height of the right hand peak, set the L limit to 127 temporarily, and expand the display with the C key (See Figure 4). Since the display is 11077 high, the right hand peak seems to be about 8000. The full width half maximum should be about 8.5, and the peak occurs at 133. Now strike the H key and enter those parameters:

```
H
FWHM=  +0.000000E+00 8.5  Sigma= +0.361162E+01
HEIGHT= +0.000000E+00  8000  At  +0.000000E+00 133
AREA=  +0.724581E+05
```

In order to be able to observe the background, reset the L limit to 102. Now let's look at the Gaussian as it is generated in the program, by striking the J key. (See Figure 5). That seems to be pretty reasonable, so we subtract the curve in Figure 5 from that in Figure 2, and get Figure 6. The parameters entered seem to be good, but it might be possible to improve the "fit" if we moved the channel number .25 to the right.

```
H
FWHM= +0.850000E+01      SIGMA=  +0.361162E+01
HEIGHT=  +0.800000E+04  AT  +0.133000E+03  133.25
AREA=  +0.724581E+05

F
G      (See Figure 7)
```

That doesn't look as good as the previous result. Maybe the width needs to be changed.

```
H
FWHM=  +0.850000E+01 9   SIGMA=  +0.382407E+01
HEIGHT=  +0.800000E+04  AT  +0.133250E+03  133
AREA=  +0.767203E+05

F
G     (See Figure 8)
```

That looks better, let's make it even wider now.

```
H
FWHM=  +0.90000E+ol  9.5  SIGMA=  +0.403652E+01
HEIGHT=  +0.800000E+04   AT  +0.133000E+03
AREA=  +0.809826E+05

F
G      (See Figure 9)
```

Much better. We are pretty close to the trees, so we can examine the forest better from a distance. To get the original full screen display, strike the D key.

```
D
G      (See Figure 10)
```

From this viewpoint, it is obvious that the peak is a little too tall. Let's try 8500 for the HEIGHT parameter.

```
H
FWHM=  +0.950000E+01   SIGMA=  +0.403652E+01
HEIGHT=  +0.800000E+04  8500  AT  +0.133000E+03
AREA=  +0.860.440E+05

F
G     (See Figure 11)
```

That's just a hair too much, try 8400.

```
H
FWHM=  +0.950000E+01   SIGMA=  +0.403652E+01
HEIGHT=  +0.850000E+04  8400  AT  +0.133000E+03
AREA=  +0.850317E+05

F
G     (See Figure 12)
```

That's pretty good. Perhaps you could better the "fit" by spending more time adjusting the parameters, but the improvement in the results would probably not warrant the effort. The differences in the last several moves are on the order of a few percent, and with data of this type, it probably isn't possible to do much better than that without using some sort of least squares technique.

Modification of STRIP

Let us suppose that a user has a requirement for a special routine to subtract a known background run from the current data field. Specifications for the subroutine might be:

Obtain a normalization factor from the operator/ user and then read the data while point-by-point subtracting the product of the normalization factor times the input data from the resident spectrum and leaving the result in the resident spectrum.

The flow chart for this routine is Figure (13), the listing is Figure (14). The normalization factor is obtained by asking the operator for that number. The input routine is setup for reading from the high speed paper tape reader by depositing zero in location 56, then the DO pseudo-operation is used to call the initialization routine for the loop, after which the GET routine is used to get a number from the paper tape reader. The short computation in the floating point package substitutes the result of subtracting NORM times the just obtained number from the contents of the location pointed to by I1 (location 105).

The CONT routine updates the pointers, and tests for the end of the loop. When the loop has been satisfied, the subroutine returns to the keyboard monitor for the next command (and restores location 56 to 7777 to enable keyboard input).

Notice that the program coding is relatively simple and that many functions are really calls to various subroutines, either in the Floating Point Package or the STRIP package.* One tricky point is that the user must be sure that the locations in the keyboard character and directory tables correspond, and do not interfere with other key-called functions active in the package (see page 1 of the HULME routine* for additional keyboard called functions).

Loading and Debugging User-Written Subroutines

The disc resident version of STRIP has some coding at 3600, which tests the switch register at load time, and halts if SR=0. The user may now use the Middle of Core Loader (MOCL) at 3777, and/or the version of ODT (DEC-08-COC1-PA) at 1000. If ODT is to be used, the contents of location 445 (BASE2*) must be changed, since the display buffer will over write ODT (1000-1577) otherwise. Debugging is not usually hampered by moving the display buffer up into the end of the floating point buffer area, since a limited display field is acceptable when debugging. The arrangement is intentionally designed to put the MOCL loader, and ODT in data areas which will be overwritten by data during the normal operation of STRIP, since these programs would presumably need to be used only at load time of STRIP.

The non-disc-resident version of STRIP can use the standard binary (SA 7777) loader and ODT (1000-1577) in a similar manner.

Applications:

STRIP has proved useful in a wide variety of applications, in spite of the fact that it has been available for only $3\frac{1}{2}$ months.

Since the data input routine for STRIP is via the Floating Point Package (FPP), the input has the restrictions mentioned in the FPP writeup. Since the FPP output format is compatable with the input to STRIP, it can be used to plot data generated in FORTRAN, CALCULATOR, or FOCAL, or any other program using the FPP for output. (A minor modification to the input routines will allow the program to be used in installations without the high speed paper tape reader).

Spooner et al[1] use the disc resident version of STRIP for an almost on-line plotter (as well as for initial data reduction) for data from a neutron diffractometer data acquisition system. The facility for rapid turn-around and the availability of Polaroid camera pictures of the display have made a significant improvement in the operation of their diffractometers. For example, the data used as the subject of the example in this paper was taken from such an experiment. The central peak (see Figure 1) of the data is the result of poor collimation of the incident beam, and the availability of the display allowed the experimenters to correct this situation before using up more beam time (each point on the plot represents 10 minutes of neutron beam time!).

In another application, a study of filtration of particles through sand beds by Champlin et al[2] has been made possible by STRIP. The volume of data acquired by the experimenter (about 500, 400 channel spectra) and the difficulties of dealing with the rather complicated background in this experiment were such that some mechanized data reduction scheme is required. Normal fitting techniques proved elusive, because of the aforementioned difficult background situation.

The obvious use of STRIP is for reduction of data from Pulse Height Analysers. The saving in time of this method over hand methods of analysis has significantly improved the work done by a group doing neutron activation analysis. The "accuracy" of the results seems to compare favorably with tedious graphical methods usually involving centroid deter-

*A listing of STRIP and the Gaussian routine HULME is available from the author.

mination, and "block counting" integration methods. By use of the "#" key which permanently subtracts the currently defined Gaussian from the data buffer, it is possible to completely separate the peaks in a complicated spectrum from the background, which may be quite complicated in shape also. In one case, the user was able to separate a small peak of 10% of the area of a large peak, which was well up on the "skirt" of the large peak.

Conclusion:

STRIP is a data display program that is easily used by the experimenter to examine and partially reduce his data. The reliance upon the judgement of the user in fitting operations make it very useful in situations where normal least squares techniques are unsatisfactory, and the facility for expansion and change within the program make it possible for the program to "grow" toward solving the particular needs of a large number of widely different applications.

FIGURE 14

```
              /SUBROUTINE TO SUBTRACT BACKGROUND
              /SPECTRUM TIMES OPERATOR-SUPPLIED
              /NORMALIZATION FACTOR, FROM THE
              /RESIDENT SPECTRUM.
              /CALL WITH "W" KEY, AND SUPPLY
              /NORMALIZATION  FACTOR AS ASKED FOR.
              /FAST TAPE READER WILL THEN READ
              /BACKGROUND SPECTRUM WHILE SUBTRACTING
              /NORMALIZED SPECTRUM FROM EACH POINT.

               DO=JMS I 111
               CONT=JMS I  112
               FNTR=JMS I  7
               FNEG=1Ø
               I1=1Ø5
              /SETUP TNIZE W KEY
              *267
0267  7451              -327
0270  0000              0
              *154
0154  4100              W /ENTRY IN DIRECTORY TABLE
              *4100
4100  0000    W,        0 /BACKGROUND SUBTRACTION ROU-
4101  4726              JMS I CRLFP             /TINE
4102  4727              JMS I MESSAG /MESSAGE PRINTOUT
4103  1617              1617 /NO              /ROUTINE
4104  2215              2215 /RM
4105  7540              7540 /= SP
4106  0000              0
4107  4531              JMS I 131 / INPUT NORM
4110  4407              FNTR  /ENTER FLOATING POINT
4111  6330              FPUT NORM / STASH FACTOR
4112  0000              FEXT
4113  3056              DCA 56  /56=0 IS FAST READER
4114  4511              DO                    /CONDITION
4115  4531              JMS I 131 /INPUT A NUMBER
4116  4407              FNTR
4117  3330              FMPY NORM
4120  0010              FNEG
4121  1505              FADD I I1
4122  6505              FPUT I I1 /(I1)=(I1)-NORM*NUM-
4123  0000              FEXT                  /BER
4124  4512              CONT
4125  5700              JMP I W
4126  4170    CRLFP,    4170  /SEE LISTING
```

```
4127  4274  MESSAG,    4274
4130  0000  NORM,      0
4131  0000             0
4132  0000             0

CONT        4512
CRLFP       4126
DO          4511
FNEG        0010
FNTR        4407
I1          0105
MESSAG      4127
NORM        4130
W           4100
```

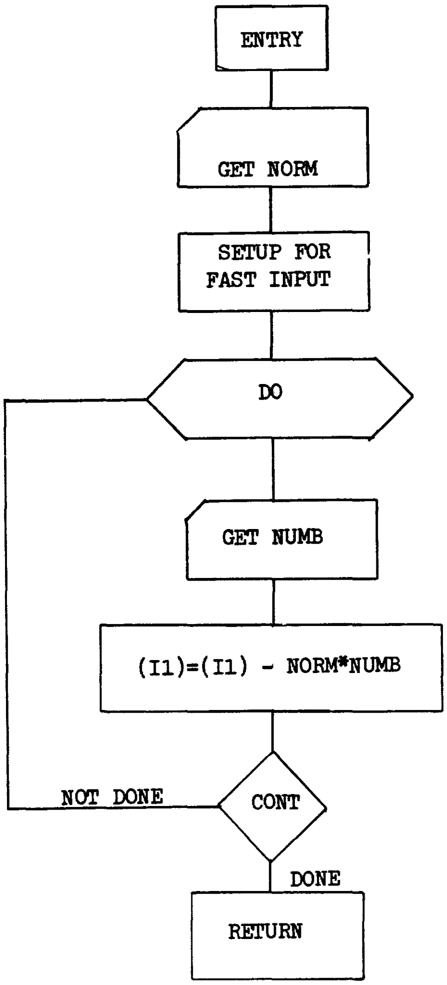
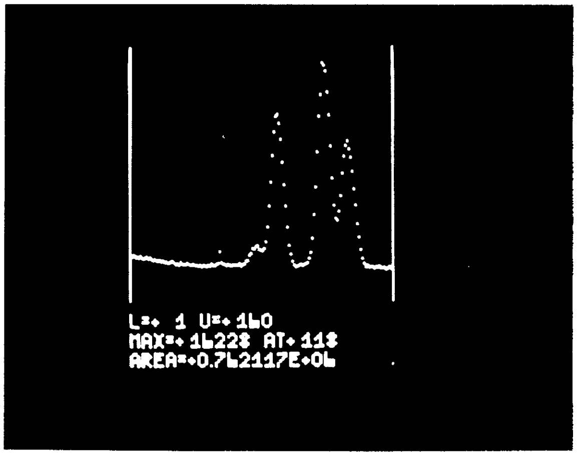
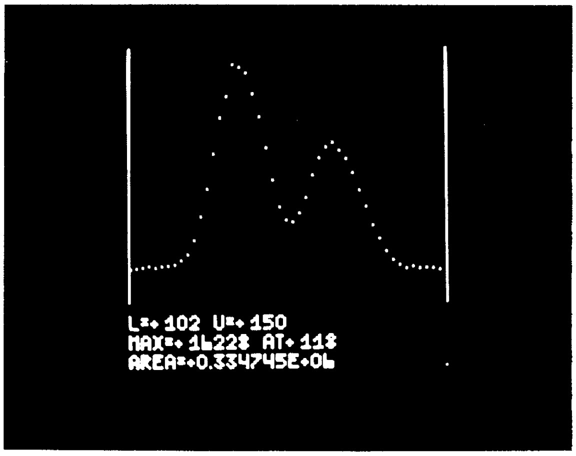


FIGURE 13



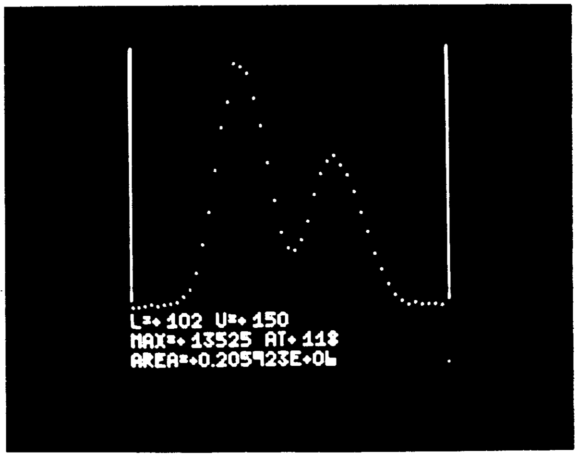
```
L=+ 1 U=+ 160
MAX=+ 16228 AT+ 118
AREA=+0.762117E+06
```

Figure 1



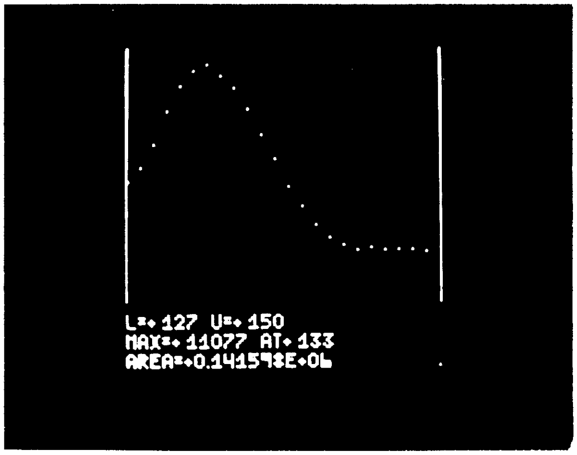
```
L=+ 102 U=+ 150
MAX=+ 16228 AT+ 118
AREA=+0.334745E+06
```

Figure 2



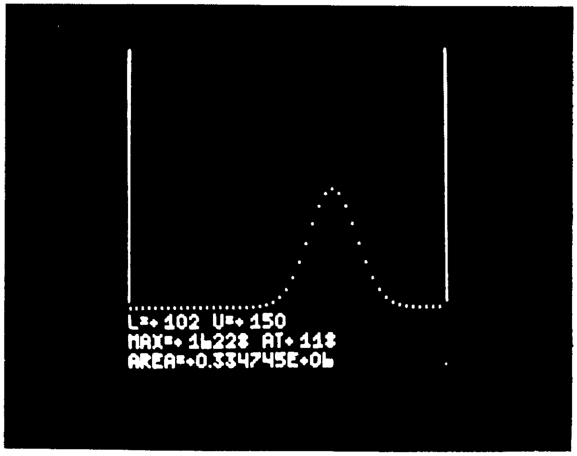
```
L=+ 102 U=+ 150
MAX=+ 13525 AT+ 118
AREA=+0.205923E+06
```

Figure 3



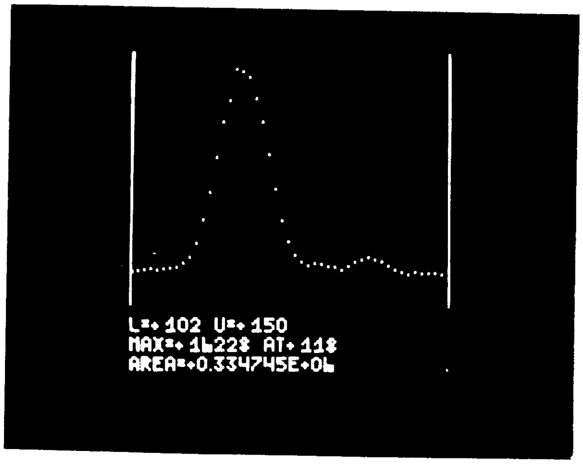
```
L=+ 127 U=+ 150
MAX=+ 11077 AT+ 133
AREA=+0.141598E+06
```

Figure 4



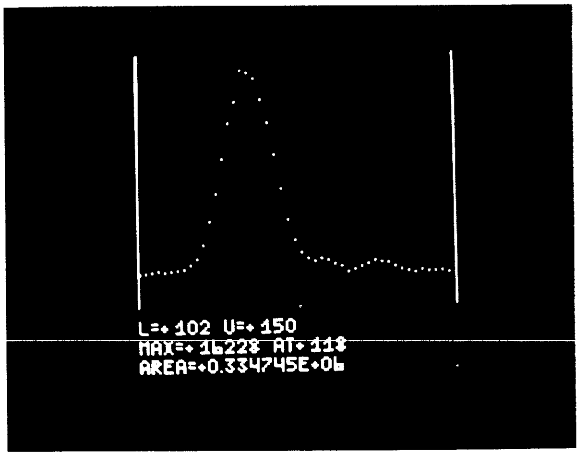
```
L=+ 102 U=+ 150
MAX=+ 16228 AT+ 118
AREA=+0.334745E+06
```

Figure 5



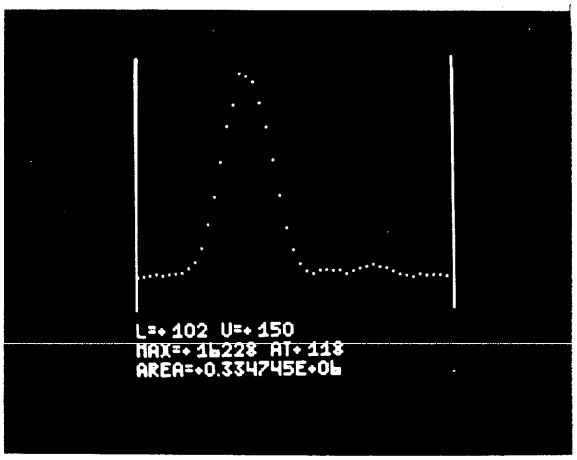
```
L=+ 102 U=+ 150
MAX=+ 16228 AT+ 118
AREA=+0.334745E+06
```

Figure 6



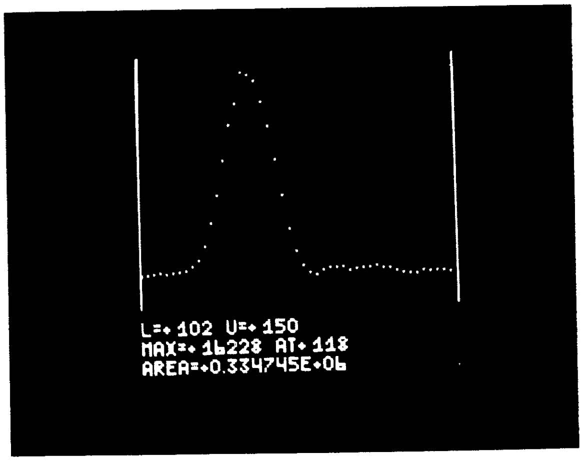
```
L=+102 U=+150
MAX=+16228 AT+118
AREA=+0.334745E+06
```

Figure 7



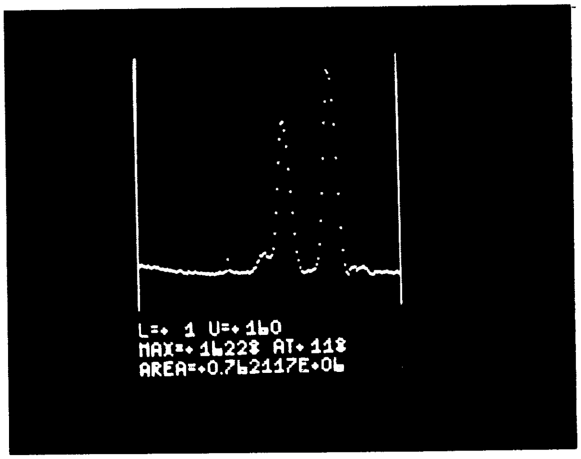
```
L=+102 U=+150
MAX=+16228 AT+118
AREA=+0.334745E+06
```

Figure 8



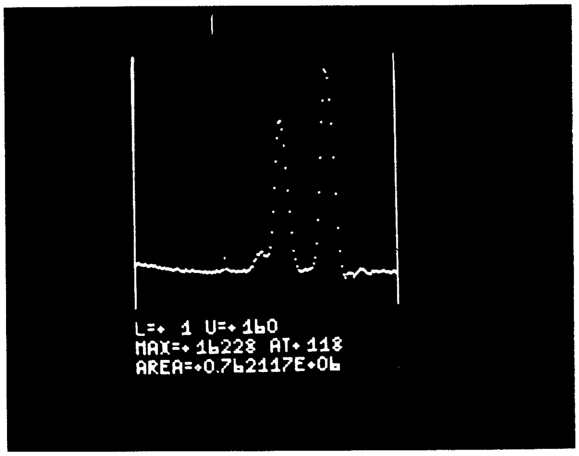
```
L=+102 U=+150
MAX=+16228 AT+118
AREA=+0.334745E+06
```

Figure 9



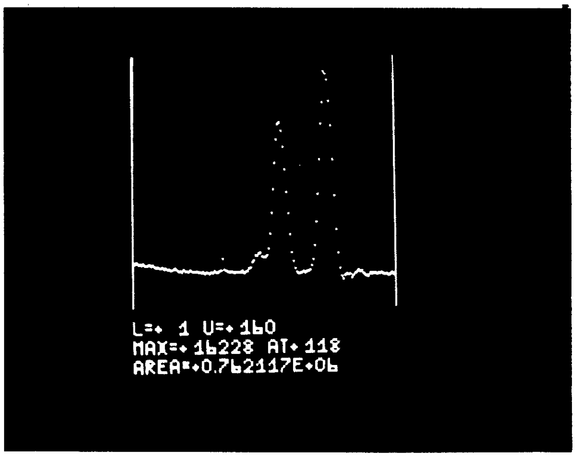
```
L=+ 1 U=+160
MAX=+16228 AT+118
AREA=+0.762117E+06
```

Figure 10



```
L=+ 1 U=+160
MAX=+16228 AT+118
AREA=+0.762117E+06
```

Figure 11



```
L=+ 1 U=+160
MAX=+16228 AT+118
AREA=+0.762117E+06
```

Figure 12

BRINGING THE COMPUTER INTO THE HIGH SCHOOL CLASSROOM

Michael L. Doren
Deerfield High School
Deerfield, Illinois

ABSTRACT

This paper is geared primarily for high school mathematics teachers, department chairmen, supervisors and others interested in computer education at the secondary level. Some ideas on how the PDP-8/S, in combination with an inexpensive closed-circuit TV setup, can be used to enrich concepts taught in all levels of high school classes are presented. Emphasis is on the goals and ways and means of more effective use of the computer in the high school classroom.

This paper is presented to the Digital Equipment Computer Users Society 1968 Spring Symposium in the spirit of sharing some ideas on how to use a computer installation effectively in a high school classroom. All over this country, schools are becoming increasingly aware of the important role of the computer in our society. Indeed, many schools could not schedule their classes without the use of a computer; yet too few schools include computer education as part of their schedule of classes. Schools are becoming increasingly anxious to get the computer "into" their curricula. Decisions are being made; ways and means are being settled upon. Should the school purchase a small computer that will allow its students to get their hands on it and really see how it operates, even though it has limited capabilities in problem solving? Should the school enroll in a time-sharing plan giving students little opportunity for hands on operation but providing tremendous problem-solving potential? Until recently, any high school (of average financial means) desiring computer education for its students has had only the above options. Financially well-off districts could have both the small accessible computer and the teletype line to a large installation.

But now a new type of computer has been offered on the market that provides a third, and a preferable, alternative. Digital's PDP-8/S and PDP-8/I sell and lease for prices that are within the financial capabilities of many schools. Yet these machines offer the two advantages of computer education: (1) They allow capable students to get their hands on the computer; program in machine language, watch the lights that step through their programs, see and investigate the working parts, and learn how the computer works; and (2) they have sufficient core memory to accept more sophisticated programming languages and to allow for the successful solution of almost any problem that would arise in a high school classroom. At Deerfield High School, we have had computer education for three years. For the first two years we had the small, accessible, limited type of computer which I mentioned earlier. This year we purchased a PDP-8/S, and we are quite pleased with its ability to fulfill our needs. Along with the new computer, we have initiated some new ideas that we feel add greatly to our ability to make the computer a vital part of our mathematics curriculum. Under department chairman, Karl Wildermuth, I believe that Deerfield has played a pioneering role in computer education. This paper concerns ways and means; it is not a technical dissertation on a complex subject but rather a way of putting pen to the ideas that Mr. Wildermuth and our staff have developed so that other schools may be encouraged to enter into the computer education field and may share some of the benefits of our experience.

COMPUTER EDUCATION AT DEERFIELD

At Deerfield High School, we have two primary purposes for computer education within the mathematics department: (1) To teach computer operation and programming to those students who are capable of learning it; and (2) to use the computer to provide enrichment for our courses at all levels within the department. To implement the first of these two objectives, we offer two courses in computer education. Fundamentals of Digital Computation, M-22, is a one-semester course for above-average students who have had at least three years of mathematics. The course of study includes Boolean algebra, the electronic hardware of the computer, and intensive work in computer programming and operation. (see Computers and Automation, March, 1968, page 28 for an outline of this course) In addition to M-22, we are preparing a course for interested sophomores and juniors called Introduction to Computers, M-16. This six-week course will enable students to use the Calculator Mode, Fortran, FOCAL, and other languages to solve problems that they meet in their mathematics, science, and business courses.

Approximately fifty students enroll in our M-22 course each year, and we hope to have about fifty more in the introductory course. These students are given complete access to the computer itself and the off-line teletype that we have located in our computer laboratory. Many of these students become very good programmers, and others show remarkable talent in working with the electronic and technical aspects of computer operation. Indeed, I am confident that several of these students will eventually select vocations in the computer field, and I feel that their high school experience has been an important factor in this selection. However, this alone does not justify the existence of the computer at Deerfield High School. We feel that the second objective named above is equally important. We believe that it is extremely worthwhile to use the computer at all levels of coursework within the department, from modified algebra to calculus. Any teacher who has tried to teach the quadratic formula to freshman algebra students or Pascal's Theorem to a statistics class would agree that the computer enables him to do a better job of teaching many topics. Secondly, such use of the computer exposes nearly the entire student body to the capabilities and limitations of the computer. In this age, when those who do not understand the complexities of mathematics and automation fear or belittle the contribution of the computer to our society, this friendly exposure can be very helpful. Furthermore, it is far more reasonable to justify the expense of computer education on the basis of 2000 students rather than on one or two hundred. Therefore, I feel that it is just as important that our curriculum seeks to give exposure to the average and below-average student as it is to give depth and competence to the above-average student.

Reaching the above-average student is relatively more easily done. We merely (if I may be allowed literary understatement) place some good students in a room with a competent teacher and a computer. Throw in a few textbooks and some manuals and we have computer education that is as good as our teacher, computer, and students can provide. But when we are talking about 2000 students, 15 teachers, and one computer, the problem takes on some new aspects, and it is just these aspects to which I devote this paper.

CLASSROOM USES FOR A COMPUTER

The problem boils down to trying to develop some system which will enable all teachers to use the computer in all of their classes in such a way as to improve genuinely the level of instruction and create a real interest among the students. First, we must decide what goals, or educational objectives, are served by the use of a computer demonstration. I have found that the computer can be a definite aid in both the deductive and inductive learing processes. After I have proved a theorem to my class (Hero's formula for the area of a triangle, for example), the computer can be used to give repeated examples to demonstrate that the theory is valid. The theory is introduced, black-board examples are given, then several examples can be worked on the computer to be verified by the students. The homework assigned, which is designed to practice and apply the theory, can be checked the next day on the computer. This is good deductive learning, adding the incentive for the student to be able to "match the computer." In addition, the program can be written to detect unrealistic data entries (such as negative sides if Hero's formula were being programmed); and a diagnostic statement can be typed, making the student more aware of the dangers and exceptions found in a theorem.

With good students and advanced concepts, the computer can be a strong encouragement to inductive learing. Several well-planned computer examples enable many students to "discover" theorems for themselves. Using a Fortran program that computes binomial coefficients, I was able to show my statistics class (or they were able to show me) that

$$\binom{n}{r} + \binom{n}{r+1} = \binom{n+1}{r+1}$$

Thus, they were more willing to attempt a formal proof of Pascal's Rule after they had discovered it for themselves. After a few more examples, they were ready to prove that

$$\sum_{r=o}^{n} \binom{n}{r} = 2^n.$$

In any course that involves a great amount of theory, a computer demonstration can create the need for that theory -- a nice alternative to teacher-imposed needs.

In the realm of high school mathematics there are many topics which lend themselves effectively to computer demonstrations. A teacher or group of teachers can go through a textbook and find many ideas that can be programmed for classroom use. At Deerfield we have written some seventy programs for use in our curriculum. They range from simple concepts such as "prime factorization of an integer" to more advanced ideas like the "upper triangulation of a 3x4 matrix." Every course should offer at least five or six opportunities each semester to bring the computer into play. In an advanced algebra course, for example, the computer can be used to demonstrate such topics as:

1. Quadratic Formula
2. Distance Formula, Midpoint Formula
3. Linear Equations
4. Law of Sines
5. Law of Cosines
6. Arithmetic Progressions
7. Geometric Progressions
8. Evaluation of Determinants
9. Solution of System of Equations
10. Parabola Discussion
11. Circle, Ellipse, Hyperbola
12. Conic Section Identifier
13. Operations on Complex Numbers
14. Factorials
15. Permutations, Combinations
16. Binomial Coefficients
17. De Moivre's Theorem
18. Logarithms

In addition, the calculator mode of PDP-8/S will demonstrate the following ideas met in advanced algebra:

1. Order of Operations
2. Group, Field Properties
3. Roots and Powers
4. Evaluation of Polynomials
5. Trigonometric Identities
6. Fraction-to-Decimal Conversion

PROGRAMMING FOR CLASSROOM USE

After having decided what topics are appropriate for computer demonstrations in a high school classroom, the actual programming of the concepts must be done. PDP-8/S offers two programming languages that are relatively simple and very well-suited to this purpose: a version of Fortran that is similar to those used with the large computers and a new language, FOCAL. These languages are easy to use, but they allow the solution of quite sophisticated problems. This past summer, several of our faculty members spent three weeks writing programs in Fortran. A source language print-out of each program, along with a few example problems, were compiled into a loose-leaf booklet. Copies were made and distributed to each member of our department faculty. As new programs are added and old programs are revised, each teacher is given a copy to place in his booklet. We are now in the process of rewriting these programs in the new FOCAL language. Each of these programs has five features that I believe are essential in a program that is to be used for classroom demonstrations.

Clear request for data input. Each program begins with a very specific request for data. This allows the operator or teacher (who may not have written the program) to enter the correct data in the correct order.

Well-formatted output. The results and answers are very clearly stated so that the teacher need do very little interpretation of the results for his class.

Diagnostics and "IF" tests. All entries of data are carefully checked for appropriateness. If a number must be positive, an "IF" test should be applied to be sure that it is. If the data must meet other requirements, these too should be tested; and the program should be written to inform the operator of the error that he has made. These error diagnostics come into play when a sincere error is made (which can happen when the operator is unfamiliar with the limitations of a concept) or may be called up intentionally by the teacher in his selection of data (to encourage the students to find out why the computer has rejected the data). The error diagnostics can be programmed in a light-hearted way to give the students a chance to see the computer with a more "human" personality. It is, however, important for the students to know that these "human" characteristics are the result of humans. The computer is quick, accurate, obedient, and dumb. It can often find answers when the data are completely unrealistic; it takes a human being to reason that the data are not right.

Programmed recycle. The first statement of a program should be numbered and the last statement should recycle the program to its start. In this way, many examples can be done without having to re-address to console. Using the computed GO TO function of Digital's Fortran, the same program can be used to perform several related tasks, such as computation of mean, third, or fourth proportionals or finding either the missing leg or hypotenuse of a right triangel. In this way, the time needed to read in a new program can be saved.

Algorithmic solution and mnemonic variable names. Whenever possible the program should be written so that solutions are performed in the same manner in which they would be performed by students. With due respect for the restrictions imposed by integer and floating-point considerations, the variables used in the program should be mnemonically close to the ideas they represent (e.g., DIST for "distance", SUMX for "sum of the X's", etc.) By doing this, a program becomes a useful resource for a student who is trying to learn how to program a computer.

The following excerpts from teletype input-output (from a program entitled "Parabola Discussion") will serve to illustrate the first two features mentioned above:

Data Request:

```
PLEASE TYPE IN A, B, AND C
OF Y = AXX + BX + C
```

Input:

```
1, -6, 9
```

Output:

```
P = + Ø.25ØØØØE+Ø
H= +Ø.3ØØØØØE+1
K = +Ø.ØØØØØØE+Ø

FOCUS IS (+Ø.3ØØØØØE+1   ,+Ø.25ØØØØE+Ø       )
DIRECTRIX IS Y = -Ø.25ØØØØE+Ø

Y-INTERCEPT IS (Ø, +Ø.9ØØØØØE+1      )

PARABOLA IS TANGENT TO THE X-AXIS
AT (+Ø.3ØØØØØE+1      ,Ø)

TABLE OF COORDINATES

-Ø.2ØØØØØE+1          + Ø.25ØØØØE+2
-Ø.175ØØØE+1          + Ø.225625E+2
-Ø.15ØØØØE+1          + Ø.2Ø25ØØE+2
-Ø.125ØØØE+1          + Ø.18Ø625E+2
```

(Computer will give 40 ordered pairs.)

The following example illustrates an error diagnostic being called up:

Data Request:

```
PLEASE TYPE IN A, B, AND C
OF Y = AXX + BX + C
```

Input:

```
Ø, 5, 8
```

Output:

```
SWEETHEART! IF A = Ø,
THIS AIN'T NO PARABOLA !
```

This excerpt from the actual Fortran program shows the use of mnemonic variable names and the algorithmic approach:

```
7Ø;     P = 1./(4.*A)
        VH = -2.*P*B
        VK = C - VH*VH/(4.*P)
        FY = VK + P
        DIR = VK - P
        DISC = B*B - 4.*A*C
        IF(DISC) 71, 72, 73
```

BRINGING THE COMPUTER INTO THE CLASSROOM

The five features I have mentioned above will make a program for classroom use far more effective than a hastily prepared program. It is important to acquaint each teacher in the department with the programs available for his use in the classroom so that he may be encouraged to make full use of the computer facility. Implicit in all of this discussion, however, is the need to bring the computer physically "into" the classroom. Several different techniques are available for any school, and I believe that we have tried them all. We, of course, had to rule out having a PDP-8/S in each classroom, which would be the ideal situation but a little too expensive. When we had a very small computer, we placed it on a cart enabling us to roll it into any classroom. This worked fairly well but caused considerable wear and tear on the computer as well as a few anxious moments when we "lost" it for a period of time. Next, we tried keeping the computer in one room and moving classes about in a "musical classrooms" situation. While all of these ideas have some merit, they involved difficulties which discourage computer use by teachers. For this reason, I see only two worthwhile alternatives. First, an input-output device may be placed in each classroom. However, not all teachers are able to effectively control the physical entry of programs and data; and if the output device is a teletype, only few members of the class can actually see the output. And having a teacher read numbers from a sheet of yellow paper does not provide a very dynamic classroom use of the computer.

For the above reasons, we at Deerfield have settled upon a closed-circuit television system as the most effective means of bringing our computer into each classroom. Our system consists of five elements:

1. Computer Laboratory
2. Closed-Circuit Television
3. Two-Way Communication
4. Computer Laboratory Assistants
5. Computer Laboratory Supervisor

I must preface this discussion by noting that our school built a new wing last summer, enabling us to plan an effective computer installation with the architect and electrician. However, these ideas can be used in an existing structure without having to tear down walls or build new ones.

Our computer laboratory is centrally located among our ten mathematics classrooms. It is about half the size of a classroom and has one door opening directly into the corridor and one to the adjacent mathematics department office. The lab contains our PDP-8/S, two teletypes (one on-line and one off-line), two television cameras (one for focusing directly on the teletype output and the other for use in blackboard demonstrations), a video tape recorder, an intercom panel (which offers two-way voice communication to any classroom), microphones, headphones, and various amplifiers, transformers, and signal splitters. This area also houses our mathematics library and six study carrels. Students can use the lab area to work at the computer, to make up a test, or to prepare reports or projects using the available resources. Each classroom is equipped with a wall-mounted 23-inch television, a ceiling-mounted speaker-microphone, and a call-button which activates a light and buzzer in the computer lab.

Each period of the school day, a team of two or three student lab assistants is assigned to the lab in lieu of a regular study hall assignment. These students were selected by the department after an initial petitioning meeting last fall. I had five times as may applicants as there were openings for Computer Lab Assistants (CLA's). With the help of my fellow faculty members, I picked those students who seemed most dependable and able to learn the necessary tasks. Training took place in six after-school sessions held over a period of three weeks. During this time, they were able to learn how to operate the television equipment, the intercom, the basic fundamentals of the teletype, and how to read a Fortran or FOCAL program into storage. It is not essential that these students know how to program, but I offered "voluntary" lessons in Fortran programming at which the attendance was quite good. When there is no other "business" in the lab, these CLA's use their assigned periods to increase their knowledge of computer operation and programming. Many of them have become quite skilled and I feel have obtained a truly salable skill in the computer field. Our system depends upon their ability and willingness to help. They have petitioned the Student Council for recognition as a school organization, and with the possibility of carry-over members for next year, they should require less of my supervisory time. In return for my time as computer laboratory supervisor, I am released from an assignment in a study hall or lunchroom.

A classroom computer demonstration is initiated by the teacher. He needs merely to press the call-button in his classroom, and a CLA will answer from the lab and find out what he would like to have done. The necessary program can be read into storage and the television cameras can be focused while the teacher turns on his TV, pulls the curtains, and dims the classroom lights. When the program is in storage, the CLA presses "start" and the data request is typed; the whole class can see this request. The teacher specifies the data to be entered, the assistant enters the data, and the output is typed. Again the whole class sees all of this and hears the sounds of the teletype as it goes through its paces. The students themselves can have a chance to specify the data and to have the computer at their command. Perhaps their choice of data will call up one of the programmed error diagnostics, and they will have the added advantage of trying to learn why the computer would not accept their data. The whole demonstration can be a very exciting learning process.

With the extremely simple new FOCAL language, a skilled teacher can actually do the programming for and with the class. In this way, the students can get a real feeling for the programming process and for the algorithm being performed. The simple "ASK", "SET", and "TYPE" commands give the students a sense of participation and involvement, which in my opinion cannot be paralleled in any other type of learning experience -- not to mention its positive effects on the enrollment for M-16 and M-22.

As we become more experienced, we are finding more and more uses for the computer and the television equipment. Since this equipment involves a considerable expense, full usage is important to the department and essential to the school board. We have found the TV to be a very successful means of team teaching. On an experimental basis this year, two other teachers and I taught advanced algebra to juniors using a TV-Team approach. Two or three times per week a twenty-minute lecture was presented in the lab and shown to the three separate classes over the TV. The CLA's act as camera men and technical assistants. New material can be presented in this way, and questioning can be carried out through the use of the intercom. Although we are still in the process of analyzing the data collected after the first semester, we expect to find that despite a few inherent "hostilities" on the part of the students, they were able to achieve at a level comparable to the non-team classes.

Every mathematics faculty has some members who are more skilled in certain areas than their colleagues. We have found that the TV and video tape recorder provide an excellent means of sharing this talent with the whole department. One member of our staff who has had more background in non-Euclidean geometry prepared a thirty-minute tape on this subject that was shown to all of our geometry classes over a two-day period. Similar tapes have been prepared on a variety of subjects that have enriched our curriculum at all levels. If a teacher needs to be absent from a class, he can prepare a taped lecture to be shown to his class. Certainly these ideas can be extended to bring about more efficient use of the available staff. Teachers have also found the closed-circuit TV useful in showing graphs or charts to their classes right out of the answer book, thus saving the time and expense of recopying these visual aids on a transparency or ditto master. Incidental uses include instructional TV network programs and the showing of a movie to many classes when we have had access to the film only one day. All of these ideas support my belief that closed-circuit TV is by far the best method to bring the computer into the classroom as well as offering many ways to improve the efficiency and effectiveness of instruction.

SOME PROBLEMS

Our first year with this system has not been without its problems. Generally these difficulties fall into one or two categories; problems of a technical nature with the equipment, and problems that involve personnel. At times, some of the technical bugs seemed overwhelming, but we have made progress in solving them. The "Jungle" is what I refer to as the area of the lab containing the various amplifiers, signal splitters, and the intercom panel. Some sort of outside, technical help is needed so that the necessary changes (ex: switching from live to tape TV, from the computer output camera to the long-range camera) can be reduced to a few toggle switches rather than the time consuming and confusing reshuffling of input and output wires, plugs, jacks, and terminals. A semester of experimentation is advisable in order to discover exactly what the needs will be, but after that time a professional electrician should be called upon to help simplify the lab operation.

Another rather serious problem arises when the computer itself runs down (as all of them do from time to time, especially if students are allowed to operate them). Our experience with the Digital Equipment Corporation has been quite favorable thus far. To any school that is planning to buy a computer, I would recommend finding a company that has a branch office in your area and one that is anxious to increase its participation in the computer education field. Companies that sell or lease computers to educational accounts must not fail to realize that vast differences exist between educational and business concerns.

While schools have very definite costs, both fixed and variable, they have nothing tangible to claim as output or production and therefore no "profits" from which to finance costly repairs. High schools cannot employ computer experts or technicians, and existing staffs need all the help they can get in keeping their equipment running and in receiving technical help and ideas.

We are far more able to solve problems that are concerned with personnel and students. Our greatest need is for in-service training for our teachers in the use of the computer. A summer workshop, conducted by a qualified faculty member, or after-school sessions can help out in this area, but school administrations must be willing to bear the necessary costs. A teacher needs to feel competent in the use of new equipment, or he will not use it. The demands on his time require that the equipment be as easy to use as possible. The use of a computer involves a fundamental change in teaching technique, which is difficult even for teachers of short tenure. But if we agree that computer education is a valuable and worthwhile undertaking for today's high school, then it is up to us to begin discussing our goals and the means for implementing them. I hope this paper has helped.

(I wish to gratefully acknowledge the help of Karl Wildermuth, Louis Crouch, and Michael Knight in preparing this paper.)

PDP-8/S IN THE HIGH SCHOOL CLASSROOM

Bud R. Pembroke and Dave Gillette
Computer Instruction NETWORK
Salem, Oregon

ABSTRACT

The presentation will cover the present use of the PDP-8/S as a portable computer in several curricular areas in schools within the Computer Instruction NETWORK. The use of machine language will be discussed along with the use of CINIC as a "Load and Go" conversational compiler. CINIC "Computer Instruction NETWORK Instructional Compiler" was patterned after a subset of BASIC for the 4K core memory of the PDP-8/S. The authors will include a description of the instructions, examples of programs, and a candid explanation of advantages and limitations of this language.

The Computer Instruction NETWORK is an ESEA Title III Federal Project, covering a four-county area in Oregon. Our purpose is to assist high school students in the learning of computer concepts. Computers as an area of study, is our main goal, rather than using computers to assist the student in problem-solving in other phases of the curriculum. We feel that in order to fulfill our objectives, a student must have hands-on experience. Each pupil should be able to press the buttons in running his own program. Computers are supplied for each of the schools cooperating in the C. I. NETWORK. We have been using the least expensive general-purpose PDP-8/S and other similar machines. These are portable enough to allow a sharing of machinery among several schools.

We strive to make effective use of classroom time. In addition to the computer and the on-line teletype, each classroom also has another teletype leased from the telephone company. Thus, one student can be pre-punching programs on tape, while another is running or debugging his program.

Also, to conserve machine time during the class period, we use what we call load-and-go preparation programs. These allow the student to pre-punch an appropriate tape on the off-line teletype, and read it into the computer. As the tape is being read, the preparation program is translating the teletype codes into machine language instructions which are immediately stored in the computer's memory. Now as soon as the tape has been read, the program is ready to be run. There is no waiting for intermediate tapes to be punched or processed. Operational at the present time are the Machine Language Loader (MALL) and C. I. NETWORK's Instructional Compiler (CINIC). Assembly Loader of C. I. NETWORK (ALCIN) is still in the developmental stage.

The Machine Language Loader program allows the programmer to type the first address of a block of computer storage, and then type the instructions or data to be deposited in that block. The MALL program translates the octal teletype codes into machine binary configuration and deposits each word in successive storage locations. A new block may be started at any time, by typing the first address of that block. When using this procedure, the only storage locations not available to the programmer are the page and a half containing the RIM, BIN, and MALL loading programs.

As the student learns the language that the computer uses, he can grasp a much clearer idea of the concepts involved in machine operation. One way of teaching about an instruction or programming concept is to allow the machine to be in Single Instruction mode. The students can observe the various registers and notice how the computer deals with the data and addresses during each instruction. Debugging can be done in a similar way. A student can try a program that does not run correctly, operating one instruction after the other on his own. By observing the console lights, he can find the incorrect steps in his procedure. This, of course, is meaningful only to the programmer who is familiar with the octal representation of the instructions, data, and addresses.

Since the C. I. NETWORK is, in most cases, teaching the most basic concepts of computer use to completely inexperienced people, we must start slowly and simply. The first programs are restricted to page zero. This eliminates much of the complicated explanation that would have to be covered for addressing on the other pages.

The first program introduced is extremely elementary. (See diagram one.) But many concepts are illustrated by this program. First, the students must realize that the computer stores each instruction and piece of data in a memory location.

```
0020   1023   START,   TAD X
0021   1024            TAD Y
0022   7402            HLT
0023   0005     X,
0024   0006     Y,
```

Then starting the program at the first instruction allows the machine to process the instructions in sequential order until a Jump or Halt occurs. The use, operation, and procedure for writing memory reference instructions, must be introduced. The form and use of data in the machine is explained in terms of their previous knowledge of Binary and Octal arithmetic. All these fundamental concepts can be learned by working with straight-line programs like this one.

The idea of causing the computer to make a choice based upon the value of a datum can be presented along with an appropriate skip instruction or two. As more kinds of instructions are introduced, the concept of looping is needed. One of the simplest loops to explain is a multiplication by repeated addition. The use of a counter as an end of loop decision is given in this type of loop. Print-out loops give a variety to the loop concept. Teaching the needed IOT instructions, and showing the process of the modification of instructions, allows students to have the computer type messages. Another basic concept that is introduced at this time is the use of the comparison of teletype codes as an end-check on the loop. Additional concepts that we feel are basic are the initialization of variables and the use of subroutines.

If time allows, other more sophisticated programming ideas can be presented to the class. But all additional programming concepts would simply be combinations of the above-mentioned basic concepts. Of course the hardware features of the particular machine in use, such as the increment and skip instruction, indirect addressing, auto-indexing, interrupt, etc., can also be learned if the computer is to be available for an extended period of time.

The third and final phase of our computer language requirements was that of a compiler language. We felt that this language should satisfy the following requirements:

The language should give an understanding and reflect recent developments in computers and compilers.

It should be a language that would simplify the teaching of fundamental programming techniques with a minimum of extraneous and superfluous compiler requirements.

Third, it should be a language that would allow high school students to use the compiler as a tool in math, science, and social studies classes.

There should be a minimum turnaround time and not use up valuable class time compiling the programs.

And finally, we felt that hands-on experience is valuable and would prefer on-line debugging.

The model we picked to pattern our language after was the compiler language "BASIC". This is a time-sharing language with a wide-spread and growing usage, and, as a result, would be a "living language" satisfying our requirements.

Our language, "CINIC", Computer Instruction NETWORK's Instructional Compiler, is a subset of BASIC. It is a "load and go" conversational compiler on a standard PDP-8/S with a 4K memory and a model 33 A.S.R. teletype input and output.

The following is a description of CINIC's instruction and command repertory:

Writing the Program

Below is a sample program written in CINIC. The program computes the total amount paid back in a year for a given principal and rate of interest.

```
10 PRINT "PLEASE TYPE THE PRINCIPAL AND INTEREST RATE"
20 INPUT P, I
30 LET A = P*I+P
40 PRINT "AMOUNT IS" A
50 GO TO 10
60 END
```

Explanation of program:

Statement 10:
The computer will type on the Teletype:

PLEASE TYPE THE PRINCIPAL AND INTEREST RATE

Statement 20:
The computer will type a "?" and wait while you type the principal and then the interest rate.

Statement 30:
(P times I) plus P will be computed.

Statement 40:
The computer will type out

AMOUNT IS (and the value for the amount)

Statement 50:
The computer will then "loop" back to statement 10 and repeat the above operations.

Statement 60 indicates the last instruction.

Statement Numbers

Each statement must be preceded by a two-digit number (10 through 99). You may type statement numbers in any order. The program will run in numerical order.

Instructions

Following each statement number is an instruction. The instructions possible are:

a. PRINT
b. INPUT
c. LET
d. GO TO
e. IF, THEN
f. END

a. PRINT

The PRINT instruction commands the computer to type. If PRINT is followed by a letter or series of letters, the values assigned to these variables will be typed. If a comma is used to separate the variables, five spaces will be output between the numerical values. If PRINT is followed by any series of characters enclosed within quotation marks, the enclosed characters will be reproduced exactly. Examples:

27 PRINT A
The Tty would print the value for A in exponential form. If A were equal to 25, then +0.2500000E+02 would be typed. (See explanation of exponential form.)

35 PRINT "AMOUNT IS"
The Tty would type
AMOUNT IS

41 PRINT
The above statement is interpreted by the computer as a wish for a blank line.

b. INPUT

The input instruction should be followed by a letter or series of letters separated by commas. When the instruction is executed, the computer will type a "?", stop, and wait for you to type the values

you wish assigned to each variable.
Example:

```
21 INPUT A, B
```

On executing this instruction, the computer would give the following results if you typed a "5," and a "7,"

```
? 5,
? 7,
```

c. LET

The LET instruction is used to define a value for a variable. This assigned value may be a single number or an algebraic expression involving some arithmetic operations. The arithmetic operations possible are:

sign	operation	example
+	add	A + Z
-	subtract	3 - 5.03
*	multiplication	B * C
/	divide	12/3
↑	exponentiation (exponents may be integer constants only)	A↑ 2 (means A^2)
( )	parentheses may be used in pairs to clarify any algebraic expression.	

Order of priority of operations:

1. Values inside parentheses
2. Powers or exponents
3. Multiplication and division
4. Addition and subtraction

Operations are performed from left to right for all operations with equal priorities. In the absence of parentheses in a formula involving only multiplication and division, the operations are performed from left to right. This means that A/B*C gives a value for (A/B)*C.

```
32 LET S = 5
```

Defines the variable S as equal to 5.

```
40 LET Y = 4*A*X↑2+X
```

Defines Y to equal $4AX^2+X$

```
55 LET Y = 2*(2*A-B)/3
```

Defines Y to equal $\frac{2(2A-B)}{3}$

d. GO TO

A GO TO instruction is always followed by a statement number, directing the computer to go to another statement. The computer will execute instructions in numerical order unless re-directed by a GO TO statement or an IF, THEN statement.

```
23 GO TO 14
```

This statement redirects the computer to take statement 14 as its next instruction.

e. IF, THEN

This statement allows the computer to make a decision.

```
20 IF X = 0 THEN 85
```

The computer will go to statement 85 if X = 0; otherwise, it will execute the next statement after 20.

```
34 IF X < N THEN 97
```

If X is less than N, statement 97 will be executed. If not, the next statement after 34 will be executed.

f. END

The END statement must be the last statement in the program, and must contain the largest statement number used in the program. END signals the computer that the entire program has been loaded.

Commands

When the computer types "READY", it is indicating that it is waiting for a command or instruction. It will type "READY" after:

You type an END statement indicating the program is complete.
It has executed the program.
It receives a stop command, "S", from the keyboard.

If at this time you type:

NEW the computer clears the memory of an old program and waits for your new program.

RUN the computer will execute your program.

A statement number and statement the computer will add this new instruction to the existing program.

S If, while the computer is running a program, you wish it to stop, type "S". It will stop and type READY. (This will not operate during an input statement.)

STORED PROGRAM

Computers are unable to solve a problem unless a "program", or list of instructions, has been stored in the computer memory. To solve a problem, then, the programmer must follow a sequence of three steps:

1. Write the program. Using the CINIC language, write a list of instructions which will solve the given problem.

2. Load the program. Load the list of instructions into the computer memory. This is usually done by simply typing the program on the Teletype keyboard.

If someone else is using the computer, you may punch a program tape by typing your program on a separate, off-line Teletype with the tape punch ON. Then carry your program tape to the computer's Teletype and insert the tape in the tape reader. After the computer types READY, type "NEW", a Carriage Return, and place the tape reader switch in START position. The program will be quickly loaded into memory. After the tape has loaded, place the tape reader switch in STOP position.

3. Run the program. Loading a program is like loading a gun: You are only prepared for action; nothing happens until you pull the trigger. After a program has been loaded, you execute or run the program by typing "RUN" and a Carriage Return on the Teletype.

Numbers

All numbers are limited to seven digits. Decimal points may be used if preceded by a number. If a number larger than seven digits is desired, exponential form may be used.

Exponential Form

Numbers written in exponential form (or "scientific nation") consist of two parts separated by the letter "E": the significant digits of the number, followed by a power of ten. To find the true number, multiply the indicated number by the indicated power of ten. For example:

```
    0.5120000E+03
```

would be interpreted as 0.512 X 10^3, or 512. (The "E" stands for "Exponent".) Similarly, 0.1000000E-09 would be the same as 0.1 X 10^{-9} or 0.000000001. All numbers output from the computer will be normalized and printed in exponential form. Thus:

```
    0.8357210E+03
```

would be interpreted as 835.721

Variables

A variable may be defined as any single alphabetic character, A through Z.

Legal Characters

Any numeric or alphabetic character may be used in a program. All other characters and punctuation marks are illegal, with the following exceptions:

1. Those operators defined in a LET statement.
2. Anything inside quotation marks in a PRINT statement.
3. Commas used outside quotation marks in a PRINT statement to generate five spaces or to separate variables in an INPUT statement.
4. =, >, < may be used in an IF, THEN statement.
5. Decimal points may be used if preceded by a number. (Example: 0.52)
6. Spaces may be used anywhere, as needed.
7. Carriage returns to indicate the completion of a statement or command.
8. Line feeds when necessary after a carriage return.

Error Messages

If a statement is incorrectly typed, an error message may be typed by the computer. All error messages start with an "E" and are followed by an error code letter and a statement number.

EA indicates a general format error.
Correction: Check your statement with description of correct format under Instructions.
Check also forms for Numbers, Variables, and Legal Characters

EB indicates a statement number error.
Correction: See Statement Numbers

EC indicates the length of statement exceeds acceptable limits for CINIC. (about 65 characters)
Correction: Divide into two statements

ED indicates the total length of the program exceeds acceptable limits for CINIC.
Correction recompile and:

1. Omit errors
2. Omit any unnecessary groups of parentheses in LET statements
3. Reduce size of and/or omit PRINT statements

EE indicates the total number of constants exceeds the limits of CINIC (about 50)
Correction: Recompile and limit the number of constants

```
Example:
  you type        70 LET X-5
  Error message   EA 70 (Format error in statement 70)
```

Making Corrections in a Program (On-Line)

If an error is made while typing on-line (connected to the computer), simply type a semi-colon, carriage return, and re-type the statement.

After a program has been executed, if you wish to insert or change a statement, just type the desired statement or statements.

To delete an entire statement, type the same statement number followed by a carriage return.

To replace a statement, type the same statement number and the new statement.

To insert a new statement between two other statements, choose any previously unused statement number that falls numerically between the two other statements.

After all corrections are made, type "RUN" and a Carriage Return to execute the corrected program.

CINIC has some definite advantages in classroom instruction. As can be seen from the preceding paragraphs, corrections may be made easily on-line. There is no classroom time lost in compiling the programs; spacing is completely optional, and turnaround time is an immediate response. The following example requires no format statement.

```
  60 PRINT "THE ANSWER IS" A
```

There are no fixed or floating point requirements as this let statement demonstrates.

```
  35 LET A = 21*(X*I)↑3-(5.21*6.23E-3)
```

The concepts of looping can be taught readily as in the example below. Also, this gives a student the chance to study relationships between dependent variables. In this case, the radius is allowed to range and the area can be observed.

```
  20 PRINT "RADIUS AREA"
  30 LET R = 1
  40 LET A - 3.1416*R↑2
  50 PRINT R, A
  60 LET R = R+1
  70 GO TO 40
```

Given the mathematical background with only a few minutes' instruction, students can write programs such as the one below. Then, by experimenting, will discover more about programming and the mathematics involved as in this case we must consider those cases where "D" is zero.

$$\begin{aligned} AX + BY &= C \\ GX + HY &= I \end{aligned}$$

```
10 INPUT A B C G H I
20 LET D = A*H-G*B
30 LET X = (C*H-I*B)/D
40 LET Y = (A*I-G*C)/D
50 PRINT "X="X, "Y="Y
```

CINIC does have limitations. The first of these is the necessity to keep programs short. Programs should be kept to less than 20 to 30 statements depending on the type of statement. The output mode is limited to exponential form. There are no functions available and lastly the student cannot list his program. In order to overcome these limitations we have attempted to supplement study by using time-sharing with a larger computer for those problems that require a stronger language.

With our multi-level language approach to computer science in the high school, we, of the C. I. NETWORK, feel that the students get a good, well-rounded exposure to computers and programming. Although we do not claim to train technologically-capable programmers, a student, upon finishing our introductory course, can more easily decide whether a profession in computers would appeal to him or not. If the desire is present, he can go on for more training that will finish preparing him for a job in the vast computer field. Our plan is to continue this procedure in the future, adding refinements as they are completed.

PROJECT ASC

Research and Development in the
Application of Small Computers

Report of Progress and Findings
April 26, 1968

Robert M. Metcalfe, Director
Project ASC, Room 13-3013
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

This, the first progress report of Project ASC, will briefly describe the origin of Project ASC, its goals, and opening hypotheses. Findings on the injection of a particular small computer (Digital Equipment Corporation - PDP-8/S) into an atmosphere of education and research are discussed and some conclusions are made. (The bulk of interesting data presented in the ASC progress report as found in its descriptive program listings, have been omitted from this report.)

PRELIMINARY HYPOTHESIS AND GOAL

The computer world is in the midst of a raging debate concerning the relative advantages of big-time time-sharing and the dedicated small computer. This debate includes the time-sharing vs batch-processing disagreement. Project ASC was borne with the idea that the range of applications for which computers are suited is such as to require more than just one computer or one computer mode. It has been felt that the large computer system is being over sold and the small computer given less attention than it deserves in this debate. Project ASC began with the hypothesis that there are many applications suitable for the small computer and its goal is to discover these applications.

THE COMPUTER FACILITY

The point to be made is that the words "computer facility" as used here do not denote the traditional massive conglomeration of men, equipment, and paper but a small, compact, relatively inexpensive laboratory with a markedly unimpressive and yet work-oriented atmosphere.

The Project ASC computer is a basic PDP-8/S digital computer (costing under $10,000) with its standard paper tape-teletype input-output terminal. The computer shares a small room with a desk, standard electrical outlets, a bookcase, a mimeograph machine, an IBM time-sharing console, a handy shoebox for spurious paper tapes, a chalkboard, and a number of chairs varying from one to four.

The most important contents of the room turn out to be, in a addition to the computer itself, the chalkboard and the shoebox, and one chair (not to mention chalk).

OPERATIONS

It is the general philosophy of the project that with a little help the computer is its own best salesman. The general method for publicizing the availability of the computer to would-be users is word-of-mouth.

Three main audiences are approached. The first of these is composed of members of the M.I.T. undergraduate student body who might find use for the computer in their schoolwork and in pursuing their own interests. The second audience consists of high school students enrolled in the computer programming courses offered by the M.I.T. High School Studies Program as sponsored by the Technology Community Association. The third audience is the M.I.T. faculty who might find use for the small computer in their research.

The approach used in all three cases is to introduce the goals of the project, to enthusiastically offer technical assistance in writing programs, and to arrange for computer availability with the six keys to room 13-3013 allocated to the project. The burden of generating ideas for using the small computer is left largely to the individual from whom initiative must come.

A norm is established among those who express interest in using the computer which holds that time spent by people familiarizing others with the computer operation is to be matched, in turn, by those taught in teaching others. The establishment and maintenance of this norm is found to be surprisingly easy and amazingly successful in encouraging productive activity on the computer.

RESEARCH PROJECTS

There are generally three types of programming done on the Project ASC computer. There is programming to learn how to program the computer, termed "Experience" programming; there is programming directed toward the development of useful production-type programs, termed "Do" programming; and there is programming to aid the programmer in understanding ideas, termed "Concept" programming. In all of these instances it has become clear that the development of the computer program is often equal to, if not greater than, in importance its actual production running. It should be understood that "small computer programming" is different in several qualitative ways from programming the big machines.

COMMENTS ON THE PROGRAMMING LANGUAGE: FOCAL

If the success of the project can be attributed to any one thing, it would be FOCAL. The language fulfills the basic requirements of a good programming language. It is easy to learn, possesses powers which can be tapped conveniently at successively higher levels of sophistication and has been designed well for its intended purpose.

The FOCAL programming language on the PDP-8 gains much in the way of convenience and efficiency by its implementation in an interpretive system. This allows for unreluctant test for speedy debugging. This debugging capability might be harnessed for broader applications if a FOCAL compiler were written to allow the efficient running of debugged algorithms on the PDP-8 or other machines. It is suggested strongly that there is sufficient evidence to prompt a large scale interpreter-compiler system experiment in various environments (PDP-10 say) to test its effectiveness in increasing programming efficiency.

COMMENTS ON THE COMPUTER - THE PDP-8/S

The PDP-8/S has proven itself to be an extremely useful and reliable small computer. Perhaps the only comment to be made which does not reflect the highest esteem for the 8/S in its class is the often-heard complaint about the slow paper tape reader on the ASR-33. Enough said.

CONCLUSION

Above all else Project ASC has discovered to date that the small small computer has its greatest potential in the field of education. It has been observed that an unobtrusive device tucked crisply away in a small room has been able to attract the attentions (in varying degree) of well over thirty students and several faculty members without any offering of specific rewards such as salary or course credit. It has been observed that by some mechanism, which at this time escapes identification, the availability of a small computer as briefly described here has led to unexpected excitement in the work of these students in their varying disciplines.

A scheme of computer usage which emerges from the experience of this project so far would have small computers scattered about educational institutions as ready tools for people engaged in the process of learning - students and faculty alike. This scattering should very definitely involve the placement of small computers in living groups. The small computer offers low cost computation with low units of introduction, simplicity of operation, and reliability.

CLOSING THE EDUCATIONAL LOOP IN APPLIED MATHEMATICS (THE ON-LINE CLASSROOM)

J. W. Elder
Department of Applied Mathematics and Theoretical Physics
Cambridge, England

ABSTRACT

Lectures supported by demonstration have an immediate impact on students impossible with chalk and blackboard alone. In essentially conceptual areas of knowledge, such as applied mathematics, demonstrations are often impossible and the cumbersome input/output procedure of note taking and understanding after much midnight oil and personal supervision is inevitable. The educational loop can be closed right in the classroom in the following way. The lecturer is provided with a control box on which are some knobs and switches connected to a computer (housed elsewhere) and a closed circuit TV monitor(s), the camera of which is watching the computer display screen. Parameters are entered from the knobs and tasks initiated from the switches and the results are displayed in graphical form. The lecturer has continuous control over his problem parameters, and may choose settings arising from discussion in the class. Typical problems involve systems of ordinary or partial differential equations. Separate "workshops" which simulate the equivalent of a physicists laboratory session reinforce the lecture material and provide the student with an opportunity to use his initiative.

An "experimental" hybrid computer system incorporating a PDP8, currently in use in the D.A.M.T.P., Cambridge, will be described and illustrated in a movie. A system using a PDP9 is now being designed.

POINT OF VIEW

The idea of feedback control is an old one. Advance towards some desired state is achieved by repeatedly altering the course of action in a manner determined by the proximity of that state. In some situations, for example in process industries, such techniques are highly developed. Until recently however in the case of digital computers only the most primitive feedback was possible. Consider the user of a batch processing machine with the continual need to inspect acres of line printer output and repeatedly resubmit his job. Of late this situation has been partially alleviated by the use of on-line consoles in Project Mac, the University of Cambridge Titan on-line system and several others. A similar situation has existed in the laboratory, but this is rapidly changing, notably due to firms like DEC, towards the on-line experiment. Even so very few on-line experiments involve much feedback.

Let me briefly refer to a scheme which does. In this Department we have a small portable computer called HADES = Hybrid Analogue Digital Experimental System. It is designed to control, measure, analyse and display data from laboratory models of fluid dynamical processes in the oceans (Elder 1967). The idea is to obtain a closed experimental environment, including the investigator, with a high degree of interaction and feedback. The hardware is in essence a PDP8, TAG30 analogue computer, an analogue/digital interface and a variety of experimental apparatuses and their associated servo-systems. The entire experiment is run from the teletype keyboard just as if one were at a very sophisticated on-line console to a "computer". This machine gives one a completely different attitude to experimental work. Our experience, as yet in early days, has nevertheless been so favourable that we simply asked the question: "If we can have an on-line experiment why not an on-line classroom?".

The control situation in the classroom is normally even more primitive than for many of the machines referred to above. We are all familiar with the large lecture room, the recitation of the lecturer, and no interaction at all. Admittedly this is an extreme case. Small classes, tutorial sessions and seminars relieve this situation but are only part of the solution. The student must still spend an unnecessary amount of time finding out what its all about. In subjects, e.g. Physics, where lecture room demonstrations and laboratory classes are possible the opportunity for interaction between the subject material, the teacher and the student is high. This is not always the case in subjects, like applied mathematics, where a lot of the material is conceptual rather than concrete. It is here that the idea of the on-line classroom is so important. This is not to suggest that similar techniques are not of value in other subjects, they are (Thwaites 1967), but since the material of applied mathematics is readily represented by means of a computer it is a good subject with which to explore the idea of the on-line classroom.

The feature of a computer which opens up the new possibilities for using it as a teaching aid in applied mathematics is that a complex calculation can be carried out and the result displayed apparently instantaneously. Thus, once the computer programme for a certain type of problem has been prepared, both teacher and students can concentrate their minds on the mathematical structure and the physical significance of the results obtained in a continuing dialogue with the machine. Out of this opportunity of examining a problem directly the student should develop a feeling for what is likely to happen and for the intrinsic mathematical and physical properties of the system represented by the problem. A computer provides the same kind of possibilities, in teaching a mathematically formulated problem, provided by demonstrations and laboratory work in science; teacher and students can participate directly in the investigation, and can learn by inference.

There are two main ways in which a computer could help the teaching of applied mathematics. One is as an aid in the delivery of lectures, corresponding to a demonstration in in the course or a science lecture. The other is as a means of allowing 'practical' work by a student, corresponding to the planned laboratory work of a science student.

IN THE LECTURE THEATRE

A lecture theatre is equipped with a display screen and a control box linked to a computer elsewhere in the building. The display screen is part of a closed-circuit television system, with the camera recording the output screen of the computer. The lecturer's control box contains a number of push-buttons, each of which can initiate a separate task, and knobs with variable settings, each one of which allows the lecturer to change the value of a parameter involved in the computational problem. For each task button on the lecturer's box there is a list of commands in an auxiliary digital controller. Real-time working by the lecturer is essential, and a hardware system which does this is described below.

Such a computer-display system can be called on by the lecturer at the appropriate points in his exposition. The image on the display screen is normally a curve traced out by a moving spot which corresponds to the evolving output from the computer. The lecturer could of course come to the lecture with the same set of curves already drawn on a slide for projection. The chief advantages of the computer-display are first that it has more impact on the audience, through being a 'new' investigation, the results of which unfold before the eyes of the students, second that a continuous range of variation of the relevant parameters is available to the lecturer, and third that the timing of the various stages of the computation is at the lecturer's choice, so that he can make a smooth integration of the demonstration and his own exposition.

Some idea of what can be done was shown in a film at the meeting for the following three mathematical problems drawn from different parts of applied mathematics:

(1) The harmonic oscillator with a forcing term

The behaviour of a simple damped oscillator is shown first. The dependence of the form of the oscillations on frequency and rate of damping is seen both in a graph of displacement against time and displacement against velocity. (As an aside, the combination of two oscillators produces Lissajous figures and beats.) The output of one oscillator is then allowed to feed the input of the other to represent

$$\ddot{y} + k\dot{y} + a^2y = A \sin bt$$

The output is seen to be in the form of beats, whose amplitude and length increase as resonance is approached. Change of the damping constant effects the peak amplitude of the oscillations, the 'sharpness' of the resonance, and the form of the beats.

(2) Van der Pol's equation and limit cycles

The computer integrates the equation

$$\ddot{y} - e(1 - y^2)\,\dot{y} + y = 0$$

for various initial conditions, and the result is displayed both as a graph of y against t and as a curve in the $(y,\dot{y})$- plane. This is the simplest model of self-excited non-linear oscillations, e.g. relaxation oscillations in a triode circuit. The parameter e , governing non-linear behaviour, is varied by turning a knob, and the effect on the development of steady non-linear oscillations is shown. The approach to a steady limit cycle is a striking feature of the display.

(3) Linear oscillator in quantum mechanics

This example is an attempt to convey the idea of quantisation, or the existence of eigenvalues and eigenfunctions. The Schrodinger equation in the form $d^2y/dx^2 = (x^2 - a)$ is integrated, and the solutions for y are displayed. The energy parameter a is varied over a continuous range, to show changes in the general properties of y (e.g. y is oscillatory for $-a^{\frac{1}{2}} < x < a^{\frac{1}{2}}$, has points of inflexion for $x = \pm a^{\frac{1}{2}}$. and is of exponential type for $x^2 > a$). The boundary conditions are that y be normalizable, which implies that, for $x^2 > a$, y must be of decreasing exponential behaviour as x increases. That this is possible only for certain discrete eigenvalues of a is demonstrated; and several of the eigenfunctions may be "tuned in" to show the relationship between the eigenvalue and the number of nodes in the eigenfunction.

IN THE WORKSHOP

The idea of students undertaking work on a computer either individually or in small groups is more familiar, and the desirability of such work as a complement to lectures on numerical analysis is widely recognised. The only novel feature of our scheme is that practical work with a computer, when co-ordinated with lecture courses, makes a contribution to the students' understanding of subjects other than numerical analysis itself. For applied mathematics the contribution is particularly important since that subject involves both advanced mathematical methods and end-products in numerical form. Work of this type has been familiar for many years to the engineer who uses analogue computers.

Practical computer work in applied mathematics serves two purposes, the students learn the techniques of numerical work by practice, and they improve their understanding of the mathematical structure and physical significance of the problem. These twin purposes are analogous to those served by practical work in a laboratory by a science student; experimental techniques are acquired, and the topics on which they are employed are understood better. And just as laboratory practical work provides scope for initiative and manual skill, so too does machine-aided practical work provide the student of applied mathematics with scope for the development of qualities not normally needed in study from books and analytical exercises.

One of the most instructive ways of using the machine is as a simulation of an experimental investigation. For example, the quantum mechanical interaction between two particles might be represented on the analogue computer by an interaction potential whose form is unknown to the student. The student performs "scattering experiments" on this simulation and collects data, on the basis of which he can try to develop simple models of the unknown potential function and test them by making predictions for comparison with "experiment". Exercises of this kind place the student in a conceptual environment difficult to achieve with conventional teaching aids.

Three examples of typical practical exercises are given below:

(1) Evaluation of a Simple Function

Find the second zero of the Bessel function Jo(x) to four-figure accuracy, using the routine for calculating Bessel functions provided.

Instructions: (i) Write a program that evaluates, tabulates and plots the function $J_0(x)$ over the range $x_1 < x < x_2$ $(x_1 > 0)$, dividing this range into N intervals. These processes can be done using routines described below, which are available in the system library; (ii) Use this program to obtain $J_0(x)$ for $0 < x < 10$ with an appropriate choice for N. This gives a preliminary estimate of the zero; (iii) Now repeat this process in the neighbourhood of the zero to get an accurate result.

Details of routines: (i) Bessel functions: routine calculating Bessel functions $J_0(X)$ for specified values of X and N to accuracy given by D ; (ii) Tabulation: routine for tabulating values of the function Y(X) for N values of the argument X.

(2) Numerical Solution of Parabolic Partial Differential Equations

Write a program for solving the heat flow equation

$$\partial T/\partial t = \kappa\, \partial^2 T/\partial x^2$$

with boundary conditions $T = T_1$ at $x = 0$, $T = T_2$ at $x = 1$, and initial values $T(x) = f(x)$ at $t = 0$. Use a two-level explicit finite difference scheme to solve the problem with your own choice of f(x) and investigate the stability of this scheme.

Instructions: Choose units of time so that the equation can be rewritten $\dot{T} = T_{xx}$ and take fixed intervals Δt and Δx in t and x . Let

$$T_j^n = T(j\Delta x, n\Delta t)$$

Then integrate the equation using the finite difference scheme

$$T_j^{n+1} = T_j^n + (T_{j+1}^n + T_{j-1}^n - 2T_j^n) \cdot \Delta t/(\Delta x)^2$$

Choose $\Delta x = 0.05$ (which is sufficiently accurate) and take $\Delta t = 0.001$. Solve the problem until an effectively uniform temperature is attained $(t = 1$, say), and tabulate and plot T(x) at intervals of 0.1 in t . Now investigate the effect of varying the value of $s = \Delta t/(\Delta x)^2$ by altering the timestep Δt . The difference scheme is stable for $s < \frac{1}{2}$ (see Richtmyer and Morton: Difference methods for initial value problems). Show that the accuracy of the solution is not significantly affected if $s < 0.48$ and discuss the development of instability for $s = 0.52, 0.6, 1.0$. Are the growth-rates in accordance with theoretical predictions?

(3) A Quantum Scattering Problem

We wish to investigate the possibility that two particles (reduced mass m) interact through some unknown potential V(r) , where r is their separation. We limit ourselves to the case of states of zero relative orbital angular momentum, when the radial Schrodinger equation becomes

$$\left[\frac{d^2}{dr^2} - u(r) + k^2\right] y = 0$$

with boundary condition $y(o) = 0$, where $u(r) = \frac{2m}{\hbar^2} V(r)$ and $k^2 = \frac{2m}{\hbar^2} E$, E being the kinetic energy of relative motion when the particles are too far apart to interact (we assume that $u(r) = 0$ for $r >$ some r_0).

The analogue computer has been wired up to represent this system and you may 'experiment' with it in the following way:

Knob 1 controls the value of k and you may assume that at full setting of this knob, $k^2 \gg u(r)$, any r .

The Display shows (1) if switch 1 is up, a graph of y(r) against r for $r > r_1$, where $r_1 > r_0$ but if otherwise unknown; (2) if switch 1 is down a graph of A sin kr against r , for the same values of r as in (1), A being an unspecified constant.

Knob 2 controls the scale of the r axis.

Knob 3 controls the scale of the y and a sin kr axes.

Instructions: (a) Using knobs 1, 2, 3 and switch 1 carry out 'experiments' to determine the scattering phase shift for a range of different energies E and plot a graph of scattering cross section against energy; (b) Using the sub-routine for the integration of ordinary differential equations which you developed in an earlier class, programme the digital computer to investigate solutions for various functions u(r) of your own choice; the idea being to try to find a u(r) which will give a theory in good accord with your 'experimental' phase shifts. (e.g. you might first try a square well of adjustable depth d and range a , and find the 'best' d and a); (c) When you have found a suitable u(r) , determine the number of bound states which it can support.

HARDWARE

The essence of the hardware problem is the need for real-time and reasonably fast working. Many parts of the student curriculum involve systems of ordinary differential equations, which presents a task ideally suited to an electronic analogue computer. Other parts involve algebraic problems, especially those arising from finite-difference representations of partial differential equations. This demands a digital computer of sufficient power to handle a partial differential equation in time and one space dimension with ease and some simple equations in two space dimensions perhaps with difficulty. The hybrid computer system meets this need, and also satisfies the real-time demand for most of the tasks at a cost very much smaller than would be possible with only a digital computer. A convenient aspect of the proposed system is that each of the three main components, digital computer, analogue computer, and controller, can be used independently. A system adequate for this task is sketched in the diagram below.

HADES already makes available the core of the system but it lacks the extra consoles and an adequate digital computer. Hence at the moment only one task or group can be handled at a time. In the machine envisaged, the controller, a PDP8/I with 8k of core, will spend its time handling the devices, editing text and data and doing only a small amount of "arithmetic" - that required by the analogue simulation and short tests of Fortran routines. Problem solutions will be evaluated in the two main computers a TAG60 and a PDP9.

It will make it easier for the reader unfamiliar with analogue computers to indicate how our machine works. The analogue computer used by Hades is a small desk top machine with 20+ amplifiers used as: 10 integrator-summers, 6 multipliers, 5 inverter summers, and a number of special servo systems. The following controls are available all of which can be under PDP8 control in "slave" mode: pot set, reset, hold, compute, sample, and repetitive operation. The control state can be determined from the PDP8 by reading the digital input buffer or controlled by setting the digital output buffer. In "master" mode the analogue computer runs independently of the PDP8.

Analogue computer programmes are set up by point to point wiring on a removable patchboard. These matters are dealt with in detail in numerous books.

SOFTWARE

Because of the small store of our PDP8 and the need for real time operation a command language based entirely on compiler techniques is not suitable. Hence for the moment we use a simple interpretive system similar to those in DEC programmes like DDT, Editor, Multianalyser, etc. The programme works in two modes. In "outer mode" commands are obeyed as soon as they are read from the teletype, either manually inserted or from a prepared tape. On returning to outer mode the bell is rung to signify that the programme is waiting for a command. In "inner mode" commands can only be obeyed as part of a sequence (possibly containing only one member). Commands, messages and data are entered in outer mode in numbered text strings. These strings may be edited if required. Some of these strings may be arithmetic commands which can be compiled in outer mode or compiled and run in inner mode. The arithmetic system we have used so far is basically the DEC floating point package. An outline of the commands available is shown in the table below.

Let me illustrate the use of the commands. The demonstration shown in the film and described above used the command string:

```
[HIC < 300, S1, AXKUAYK, T4, > D300, HQ2, (0)]
```

Which in words is: inspect the remote switch register, initiate the data table, set the analogue computer to compute, display and store on the fly 300 pairs of points pausing for 4 clock pulses between each pair of readings, then reset the analogue computer and display all the data points as a graph; after that inspect the remote switches which control pause, continue or restart, if continued display text, and then go and restart endlessly. In this case we had:

(1) the analogue computer in slave mode

(2) the ADC channels 1 and 2 connected via the ganglion to the analogue computer

(3) the interval timer set to 0.01 sec

(4) the three problems wired solely on the analogue patchboard

(5) problems selected by direct switching and parameters entered directly from the patchboard.

In calculations involving digital arithmetic the command T4, above might be replaced by (2) where string 2 is for example:

```
#2, [S3, <4, AL4, Z1, T1, >]
```

which in words is: select channel 3, convert and use this as an input variable to arithmetic routine 4, with output sent to DAC1, pause for 1 clock pulse and do 4 times. Here (2) represents a function generator and L4, might be

```
#4, [F AB S E X]
```

which in words is: float the current word (call it u) and evaluate the function $y = \exp(\sin(u + b))$, fix it and place in the current word. In this case ADC input 3 and DAC output 1 would be wired on the patchboard as if they were the input and output of a variable function generator.

The basic arrangement of labelled strings allows easy modification of individual strings and is particularly convenient when setting up a new task.

THE FUTURE

One does not need to be a Jules Verne, an H.G. Wells or a Malthus to make grandiose predictions about the future developments of on-line/computer/ display systems in education. But the essence of the situation is this. The computer can act as a store and processor of information - a knowledge machine. Access to such a machine is one way to give the student an opportunity to take part in the business of gathering, manipulating and processing knowledge at an early age and in an enjoyable way. He can become a user and a creator rather than an uninterested spectator.

This use of a computer-display facility is too novel for reliable predictions of its scope and value. Some fields of mathematical physics; for example ones which involve ordinary differential equations, obviously allow wider use of a computer-display in lectures than others. But it is clear that many topics can be made more interesting and more readily understandable with the help of computation carried out and displayed in the lecture theatre and that there is need for extensive use of this relatively new aid.

ACKNOWLEDGMENTS

The HADES project is supported by the U.K. Natural Environmental Research Council and indirectly, in providing facilities in the department available to the project, by the British Admiralty and I.B.M. (U.K.) Ltd. The co-operation of Digital Equipment Corporation (U.K.) Ltd., Systems Computers (Newcastle-upon-Tyne, U.K.) and Pye T.V.T. (Cambridge, U.K.) is gratefully acknowledged as is the continuing day-to-day involvement in developing the machine of: Mr. Peter Blench, Mr. David Cheesley, Mr. John Davidson, Mr. David Lipman, Mr. Mike Wakefield. I especially wish to thank the graduate students of my "Hybrid simulation" class for their enthusiasm and help, notably that of Mr. Philip Hazel. Dr Alan Burgess and Dr Nigel Weiss have been heavily involved with the teaching project. But more than any other factor we are indebted to the help and encouragement of Professor George Batchelor, Head of the Department.

REFERENCES

Elder J.W. 1967 "The laboratory is my computer" Decus Proceedings, Third European Seminar pp. 19 - 22

Thwaites, Bryan 1967 "1984: Mathematics/Computers?" Bull Inst. of Maths. and Applications. Dec.

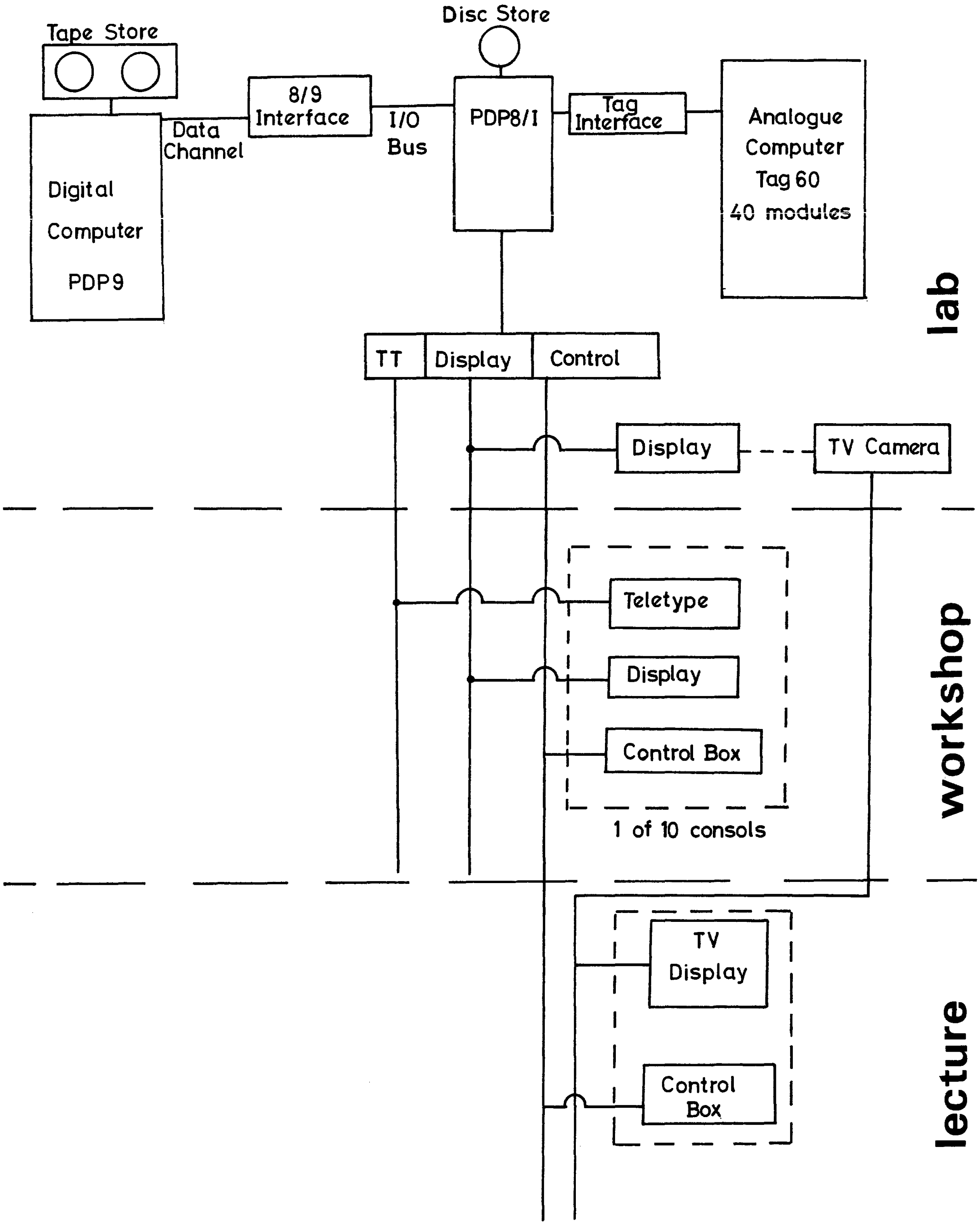
TABLE OF COMMANDS

Outer Mode

#n,	Enter text string number n as [text]
$n,	List text string n
%n,	Edit text string n
!n,	Compile text string n to Floating Point package interpretive code
[	Enter text into master control string, equivalent to #0,
]	Pass control to start of master control string

Inner Mode

A	Read ADC into current word (a register on page zero)
B	Bring contents of current data table location into current word
C	Set analogue computer to compute
Dn,	Display n pairs of points (the graph is normally drawn 1000 times)
En,	Examine location n and put in current word
Fn,	Enter n into the current word
G	Ring gong
I	Initialize data table pointer
H	Examine digital input buffer (reads e.g. remote switches)
K	Keep the contents of the current word in the data table
Ln,	Load and run compiled arithmetic code string
N	Print CRLF
On,	Print text string n
P	Print the contents of the current word
Qn,	Display text string n on the oscilloscope
R	Reset analogue computer
Sn,	Select multiplexor channel
Tn,	Wait n clock pulses
U	Increase the multiplexor channel by 1
V	Lower pen on X-Y recorder
Wn,	Write contents of current word in location
X	Load 34D display X buffer from current word
Y	Load 34D display Y buffer from current word
Zn,	Send the contents of the current word to DAC number
(n)	Do command string n
<n,text>	Evaluate the text n times and continue.


Disc Store
Tape Store
8/9
Interface
Data
Channel
I/O
Bus
PDP8/I
Tag
Interface
Analogue
Computer
Tag 60
40 modules
Digital
Computer
PDP9
lab
TT
Display
Control
Display
TV Camera
Teletype
Display
Control Box
1 of 10 consols
workshop
TV
Display
Control
Box
lecture

A SYSTEM FOR PRESENTING PROGRAMMED INSTRUCTION TO THE DEAF AND HEARING IMPAIRED

K. E. Rigg and James A. Boehm, III
New Mexico State University, Department of Speech
Las Cruces, New Mexico

ABSTRACT

A digital system for presenting programmed instruction of language concepts to hearing impaired and deaf children is discussed. The system presents controlled visual and auditory stimuli to the learner, requiring either a matching-to-sample response with four solutions or the solution of a straight four choice task. The system reinforces correct responses with a variety of visual, auditory, and primary reinforcers including pulsed pure tones, colored lights, tokens and candies. This system is complete in that it includes the basic teaching unit, its own instrumentation, data reduction, and provisions for making programs.

PROBLEM

The major problems in education of the deaf and the hearing impaired are summarized by the single word "language." Deaf children entering a school for the deaf at the age of five have had so few language experiences that their language behavior is idiosyncratic. Instructors of the deaf agree that it is generally a simple matter to teach the deaf child to name objects; but a very difficult matter to teach the spatial relationships of these objects. The concepts "over," "under," "by," and "on" often require four years of patient instruction. Since these concepts are vital to effective communication, we decided that they could be presented more efficiently through programmed instruction.

A systems analysis demonstrated the cost effectiveness of straight-line intrinsic programming. The basic premise of this type of programmed instruction is that the student is led to an operationally defined goal behavior through a series of small step approximations (frames) which are designed so that the student will make a minimum number of errors; further, the accuracy of the learner's performance is confirmed at every step and he can pace his learning rate. Programs of this type are designed for a specific population and are thoroughly tested to determine their efficiency. An instrumentation system was needed to facilitate the testing of a program to teach the concepts "over," "under," "by," and "on" to deaf children.

The expected parameters of this instructional system, as generalized from other types of programmed learning for normal children, were:

1. That each set (daily unit) of the program would take no more than 15 minutes to complete.

2. The error rate would be less than 10%.

3. Latency would be a function of the amount of information contained in each frame.

4. No response strategy would be noted.

SYSTEM DESCRIPTION

General

The design criteria for the instrumentation system were determined by the predicted measurement parameters. The instrumentation system is in two separate units: a teaching machine unit consisting of a control logic system, a stimulus presentation system, and a data instrumentation system; and a program preparation and analysis unit consisting of a code generator system and a data reduction system. This separation was necessitated by the fact that the deaf test population is located at Santa Fe, New Mexico, which is three hundred miles from New Mexico State University.

The teaching machine is housed in a rectangular console measuring 4'x2'x3' and is designed to present both auditory and visual stimuli. A tape recorder and master control panel are located on the back of the unit. An optics system and slide projector are located inside the console above the control logic and data instrumentation modules, all of which are accessible through a locking door on the rear of the unit. The student sits at a fold-down desk at the front of the unit facing an eye level rear projection screen with the appropriate response hardware. Two screen/response units are easily fitted to the machine. The first consists of a 6 3/4"x10" rear projection screen with four response push buttons directly under it, and an audio output jack in the lower right hand corner. This is designed for tasks requiring full screen presentation. The second screen/response unit is designed for a match-to-sample task and is built on the same frame design except that the response push buttons are incorporated into the screen. This screen is divided horizontally into two equal areas. The top half is divided into three sections, with the center section having enough freedom of movement so that when any part of it is pushed, a micro-switch located behind it will close. This center section is the "sample" area. The bottom half of the screen is divided into four equal areas, designed to move independently of each other. These areas and their associated micro-switches constitute the response decision indicators.

The teaching machine presents programmed instruction in a corrective mode. This requires that the machine present information continuously until a response is made. If the response is correct, the machine immediately advances to the next frame; if incorrect, it re-presents the frame until a correct response is made. The accuracy of response to a visual stimulus is confirmed by flooding the screen with green light when the response is correct or by turning off the slide projector lamp when the response is incorrect. Auditory information is presented repetitively at a rate variable between two and thirty seconds. The accuracy of response to an auditory stimulus is confirmed by a 0.3 second 125 Hz tone and green light if the response is correct or by the absence of these stimuli if the response is incorrect.

The teaching machine optics system is designed to operate in the high ambient light encountered in elementary classrooms, and the auditory system is designed to provide a calibrated binaural output with sufficient power for the hearing impaired child. The auditory system optimum output is 120 db re .0002 dynes/cm^2 complex noise with peak limiting at 130 db. Each channel of the auditory system is adjustable from the master control panel in calibrated 10 db steps from 80 db.

The teaching machine data instrumentation system provides information about the student's performance. The information encoded in this data output consists of the following:

1. The number of times the learner listened to an auditory stimulus before making a response.

2. Response accuracy.

3. Response definition, i.e., which button was pushed.

4. Response latency.

Response latency is defined in three ways, the selection of definition being dictated by the type of program administered. In a match-to-sample program, the student must confirm the sample by pushing the "sample" area at the top of the screen in order to see the possible choices; thus, in this mode, latency is defined as the time elapsed between sample confirmation and decision response. In a program using the full screen presentation, latency is defined as the elapsed time between presentation of a new slide and decision response. In an auditory program, latency is defined as the time between the onset of the verbal stimulus and the decision response. The appropriate latency definition is selected by the operator from a three positon rotary switch located on the master control panel.

The teaching machine tape recorder serves a dual purpose. It provides the recorded auditory stimuli for verbal programs, and also provides the coded information serving as the input to the control logic. The control logic system performed its operations on an electronically defined "frame" as shown in Figure 1. The code pulses, when read into the control logic, define the correct response for that frame, advance the slide projector, and control the direction of tape travel.

Code Generator

The first requirement in designing the overall system was a code generator to be used in recording code groups on magnetic tape. The design criteria for the code generator were:

1. Frequency modulated output.

2. Fifteen unique code words.

3. Variable frame duration.

4. Manual, manual-auto and automatic control of code generation.

Previous work had shown that recording code groups in amplitude form, using inexpensive tape recorders, was unreliable due to the A-C coupled amplifiers removing the D-C component and causing distortion of the code pulses. Tape drop-outs of 20 db lasting 200 ms are common with inexpensive tape recorders, further militating against AM recording. Because of its inherent insensitivity to these problems, frequency modulation was selected.

The code generator output is a quiescent frequency of 3 KHz which is deviated to 6 KHz. These frequencies were selected because they lie within the response of the inexpensive tape system. The 3 KHz deviation means that a 2.5 ms pulse provides a minimum of 12 zero crossings for the frequency discriminator in the teaching machine to detect a pulse. The FM code word is derived by digital methods block diagrammed in Figure 2. The quiescent and deviation frequencies are derived from a crystal clock, insuring stability and uniform pulse width and improving reliability.

The code generator word control is supplied by five gates, each of which can be set to generate words of one to fifteen pulses. Thus, distinct code groups can be generated for each response button and a special purpose code can be generated. The teaching machine is wired for the following correspondence:

Button	Number of Pulses
1	1
2	2
3	4
4	8
Auto Clear	10

The auto clear code allows the insertion of information frames in the body of a program which do not require responses from the student.

The operator, having set up the word control gates for the number of pulses desired, can now control the generation of these pre-set codes in three modes:

1. Manual--The operator opens the appropriate word control gate at whatever interval he desires allowing completely variable frame duration.

2. Manual-Auto--The operator pre-selects frame length with the timing unit controls (Fig. 2), and initiates the timing cycle by opening the first word control gate. In this mode the frame time is preset and can be varied from two to thirty seconds.

3. Automatic--The operator punches the appropriate code sequence on paper tape, pre-selects the frame length, and starts the timing cycle. The paper tape reader is advanced and the word control gate is read by the code generator control unit.

The code generator fires a "talk light" 0.6 seconds after each code group so that the speaker can record the appropriate auditory stimulus for each frame.

Teaching Machine Control Logic

The design criteria for the teaching machine control logic were as follows:

1. Present auditory, visual, and audio/visual frames in a corrective manner.

2. Present visual information continuously and auditory information repetitively until a response is made.

3. Present visual information in either full screen-four choice response mode or match-to-sample mode.

4. Allow only one response per presentation.

5. Confirm responses with appropriate reinforcer.

6. Reject incompatible codes generated by momentary power failures or other line voltage transients.

The control logic consists of 34 DEC logic modules whose inputs are derived from the code channel of the tape recorder and the student response buttons; their outputs control the slide projector, tape recorder, confirmation circuits, audio system, and data instrumentation. All power system controls are solid state, eliminating unreliability due to relay contact arcing and bounce and minimizing power transients.

The control logic system, as block diagrammed in Figure 3, functions as follows. At T_1 (Fig. 1) the frequency modulated code pulses of frame one are introduced into the frequency modulation discriminator. Here they are detected and made to conform to DEC logic level and rise time requirements. The code is then stored until a correct response is made by the student. At T_2 the student hears the auditory stimulus if appropriate. Time T_2-T_1 allows the tape recorder to come up to full speed after a reverse. If the student does not respond by T_3, the first pulse read into the control logic from frame two code word will cause the tape recorder to reverse direction. At this point the audio input is gated off since the tape recorder must playback in both directions. The code group at T_1 will reverse tape direction and turn on the audio. This operation continues until the student makes a response. The slide projector advances when new code words are read into storage; therefore, the visual presentation remains the same during this operation. The student can respond at any time during this sequence. If his response is correct, appropriate reinforcers are fired and the stored code is cleared. The audio and the projector lamp are turned off until the code for frame two is read into storage, which turns them back on and advances the slide projector. If the student makes an incorrect response at any time during the control sequence, the audio and slide projector lamp are turned off and the response buttons are locked out. When the frame one code at T_1 next reverses the tape recorder the audio and slide projector are turned on and the student must respond again.

In the match-to-sample mode the operation of the control logic is identical to the above with the added constraint that the student is required to confirm the sample before pressing any of the response decision buttons. This procedure must be followed for every new frame and after every error.

The code ten detection circuit functions as a clear command, turns on the audio and slide projector lamp, and cycles the slide projector.

Santa Fe, New Mexico (machine location) experiences frequent momentary power failures during late April and May which are caused by daily thunder showers lasting about an hour each afternoon. For this reason it was necessary to incorporate invalid code detection in the control logic. For example, a power line transient could cause a count of nine to be stored and without detection the student would never get out of that frame. Invalid code detection operates as though a correct response had been made; the system advances to the next frame, causing only a minimal loss of data.

Teaching Machine Data Instrumentation

The design requirements for the system were:

1. The system should be reliable and noise insensitive.

2. The system should measure latency of response with a repeatability of $\pm$ 2 ms, error $\leq$ 10 ms.

3. The system should record which response button the student pushed.

4. The system should record the accuracy of response.

The operation of the data instrumentation system is diagrammed in Figure 4. The data output consists of two channels of FM information. The design considerations for these frequency modulated channels are the same as those for the code generator. These two channels of information are recorded on an inexpensive tape recorder for reduction at a later time. Channel A of the recorder records the start latency pulse which has a 20 ms width. The pulse generator which drives the channel B frequency modulator is wired for the following correspondence:

Button Pushed	Number of 2.5 ms Pulses
1	1
2	3
3	5
4	7

The right/wrong detector adds one pulse when the response is correct.

Data Reduction System

The design criteria for the data reduction system were to display the following information in printout:

1. Latency (1500 counts/second).

2. Response accuracy.

3. Number of times the student listened to an auditory stimulus before making a response.

4. Latency error.

The data reduction system incorporates 28 DEC logic modules, a Beckman 1453 printer and a Beckman 7360A counter, as diagrammed in Figure 5. Any start latency pulse on channel A of the data tape resets the Beckman counter and opens its input gate; it begins totalizing one half the counts of the channel A quiescent frequency. When a response code word is detected on channel B of the data tape, the Beckman counter input is gated off. The response code word is counted by DEC logic and the response decision button number is gated out to the printer. An odd (wrong)/even (right) detector controls the printer ribbon so that right responses are printed in red and wrong responses are printed in black. DEC logic modules also count the number of start latency pulses without intervening responses and gate this information out to the printer, which displays the number of times the student listened to an auditory stimulus before making a response. Detection of a response code word on channel B of the data tape causes a print command to be issued to the printer. Because the DEC logic modules are so much faster than the printer, it is not necessary to delay the print command to insure that the data are stored in the printer.

The Beckman counter-printer combination locks out all incoming data during a print sequence, necessitating provisions for detecting coincidence between print cycle and latency start. When coincidence is detected, a nine (9) is generated on the latency error line and a false start is generated by the DEC logic so the response will not be missed. The latency error print-out shows that a maximum error of 400 ms has been introduced into that datum. In practice, the only time a latency error can be generated is when the student is using the machine improperly. That is, he can only generate a latency error by responding before he is presented with the information in the frame. If he does this the data print-out indicates it clearly.

CONCLUSION

The system functioned as expected in all respects. Its ruggedness was tested and found adequate in 3,600 miles of demonstration traveling in a compact van which was several orders of magnitude less severe than the environmental test it received during its tenure at the School for the Deaf. It presented and instrumented over 15,000 frames without system failure, and mechanical wear is negligible. The teaching machine was operated by people unskilled in electronics who performed well under no supervision. Five teachers of the deaf and one undergraduate college student majoring in teacher education were trained to operate the machine in two hours. In that time they were trained to operate the teaching machine and instrumentation recorder and to perform minor maintainance--replace projector lamp, clean heads, etc.

Initially a small amount of data was lost due to operator error and defective instrumentation tape. This was quickly remedied without effecting the experiment.

The data format proved to be invaluable in analysis of the programmed instruction materials under test. The concept program met all of its expected objectives, in fact the error rate was so low that many of the fast latency frames will be deleted in the interest of economy. The measurement parameters proved to be more than adequate, and latency was demonstrated to be a good indicator of the difficulty of a frame. The greatest improvement indicated by our use of the system would be the addition of a small computer to take over the computational load.

ACKNOWLEDGEMENTS

Support for this research was from the Bureau of Education for the Handicapped, contract number CEC-4-7-000183-0183.

The authors wish to thank Jean C. Rigg, who wrote the concept program, and Jerry Wiant, who did the majority of the digital design.

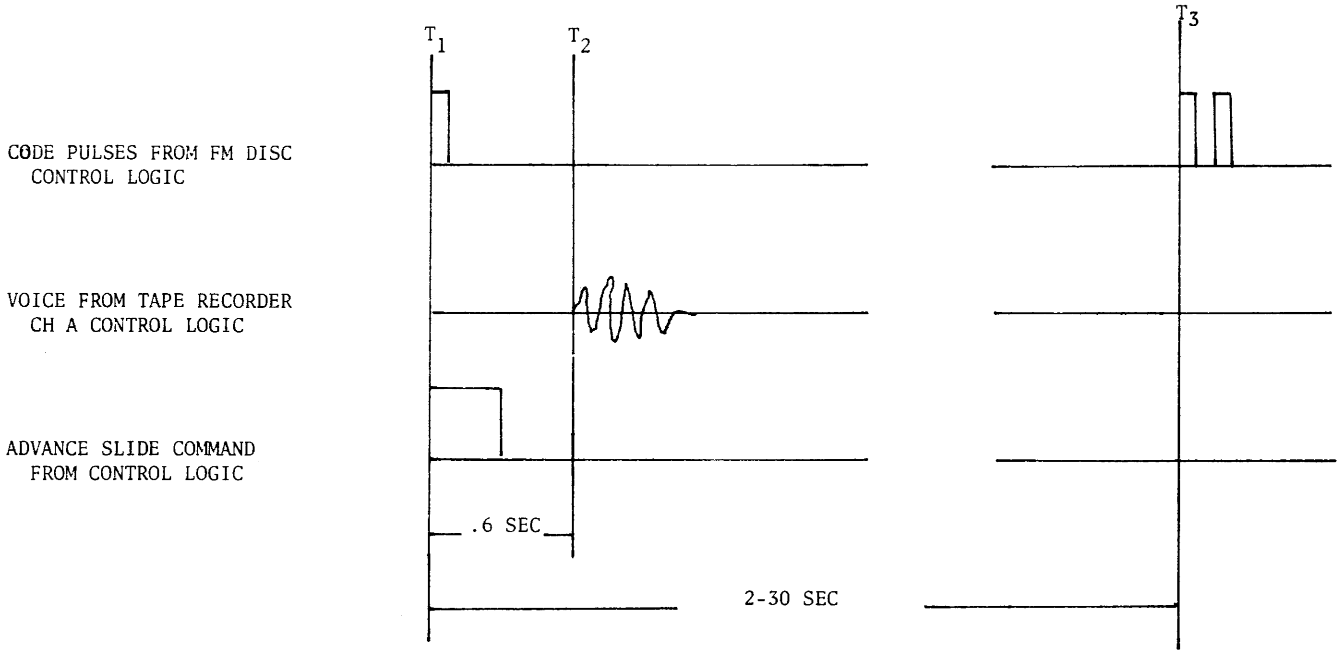

T1
T2
T3
CODE PULSES FROM FM DISC CONTROL LOGIC
VOICE FROM TAPE RECORDER CH A CONTROL LOGIC
ADVANCE SLIDE COMMAND FROM CONTROL LOGIC
.6 SEC
2-30 SEC

Figure 1 Electronically Defined "Frame"

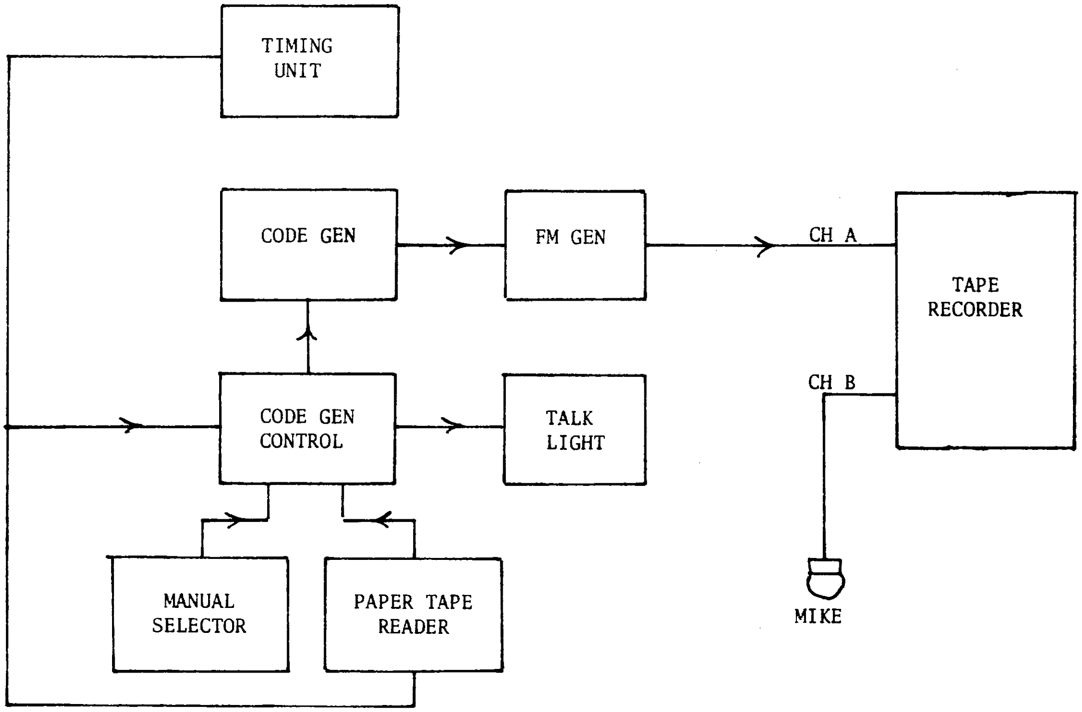


Figure 2 FM Code Generator System

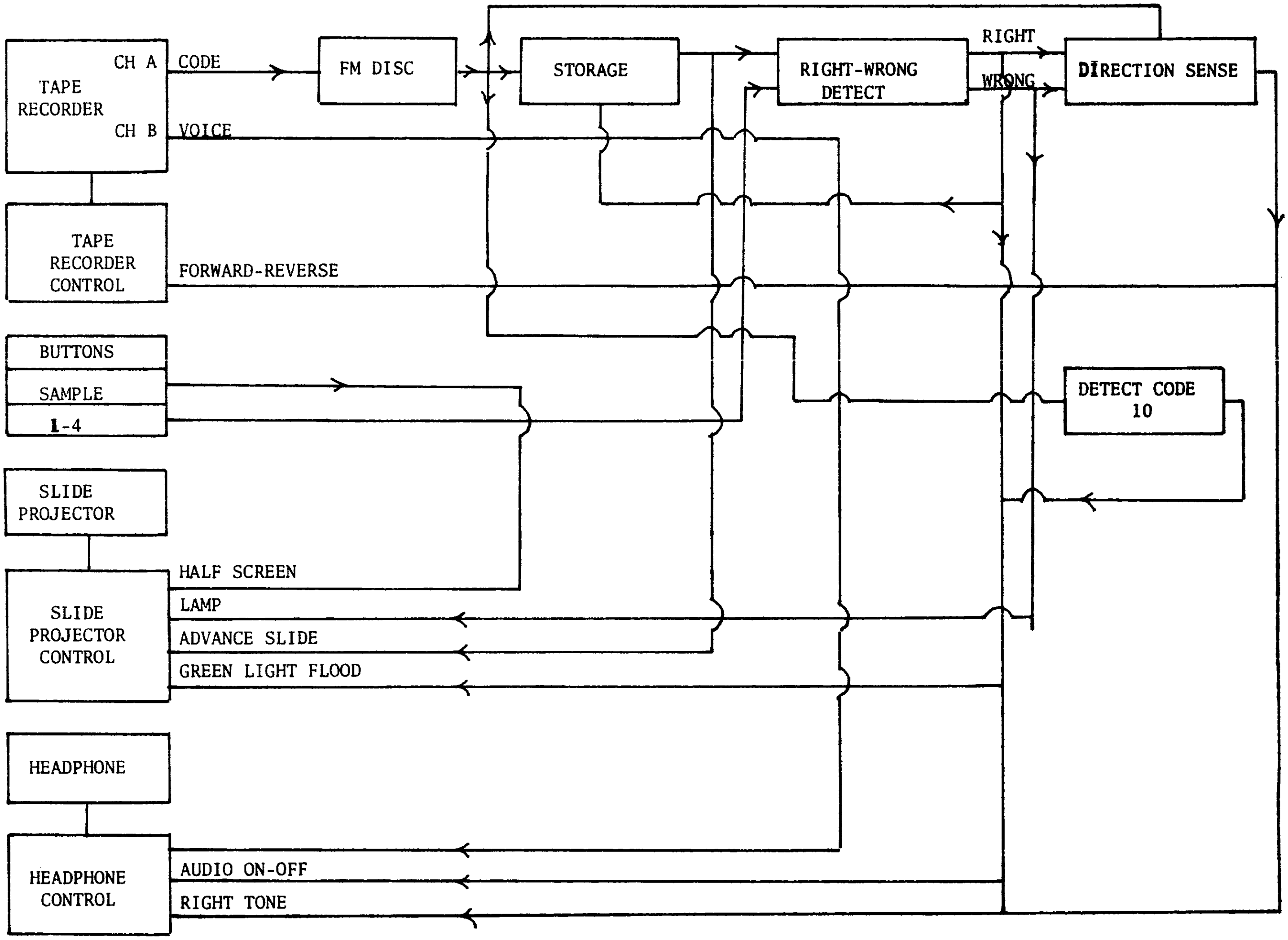


Figure 3 Control Logic System

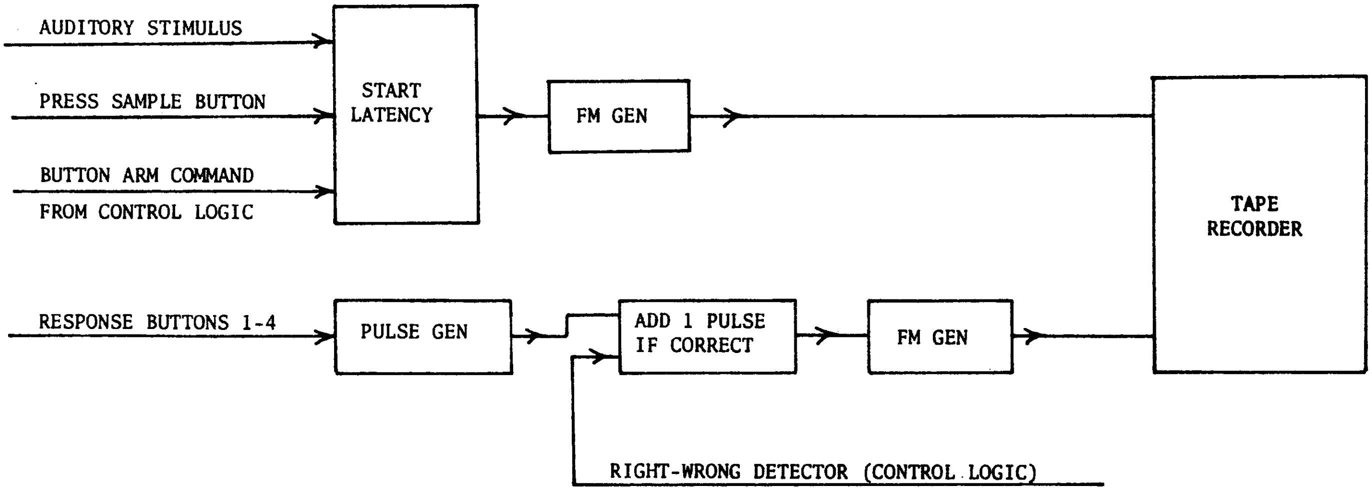


Figure 4 Data Instrumentation System

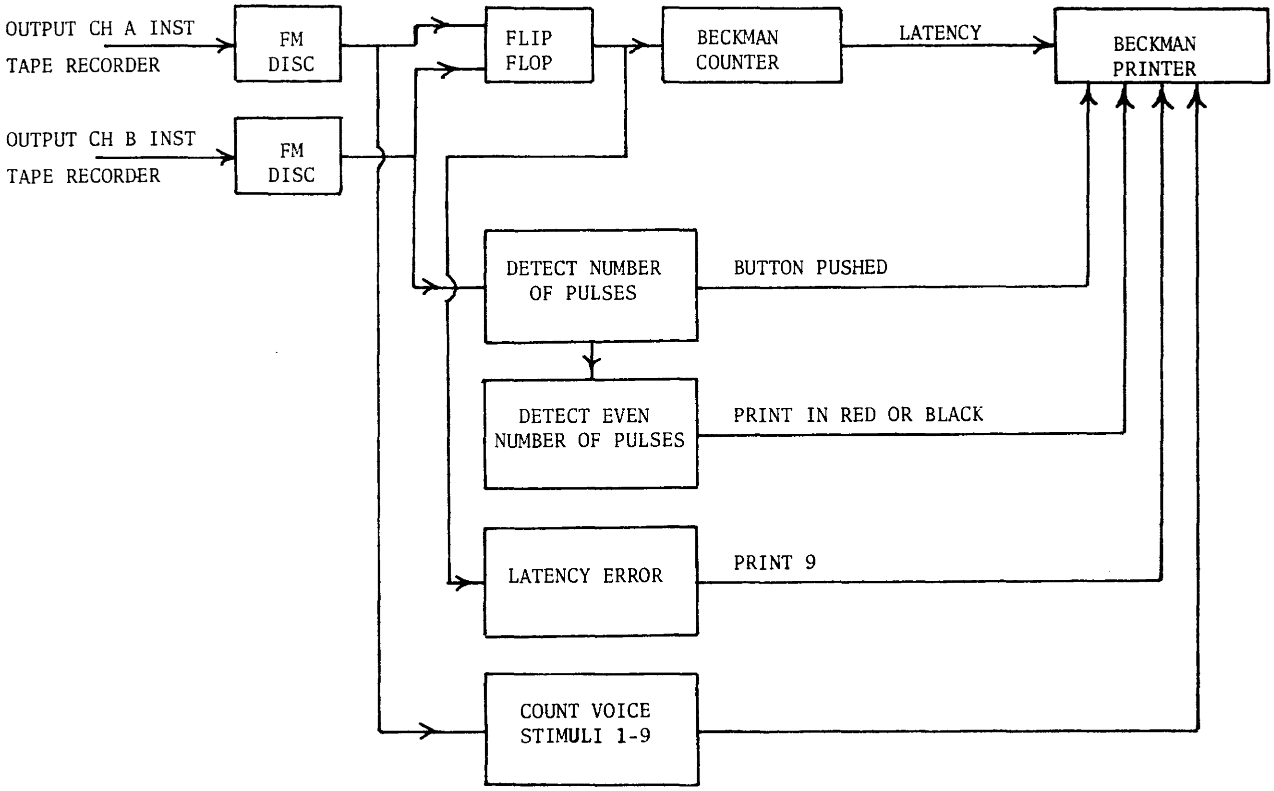


Figure 5 Data Reduction System

A COMPUTER SYSTEM FOR ELECTRICAL ENGINEERS

David M. Robinson
University of Delaware
Department of Electrical Engineering
Newark, Delaware 19711

ABSTRACT

Educational computer applications usually center on the problem solving capabilities of general-purpose machines. The electrical engineer is peculiar in that he must become more deeply involved in the computational system than is suggested by this casual use. His concern arises by virtue of his responsibility for the conception and design of the computer itself and for its hardware adaptation to a variety of applications.

A system has been evolved which is functionally directed at the problems generated by the realization of computers or computer-like systems. This system is described and a number of typical student problems discussed. The problem examples chosen illustrate the range of levels which may be encompassed using the system, the versatility of the system and problems which may be of some general interest.

INTRODUCTION

The advent of modern electronic computers has expanded the scope of nearly all areas of scientific endeavor. The electrical engineer is perhaps most acutely affected by this expansion by virtue of his two-fold interest in computer processes. He is, as are his colleagues of other scientific disciplines, excited by the use aspects of the capabilities now at his disposal. He is, perhaps, even more deeply involved by virtue of his responsibilities for the conception and design of the computer itself, and for its hardware adaptation to a variety of applications.

It is to the second phase of the electrical engineers involvement with computers that our educational activities are directed, that is, to his involvement in the realization of computers or computer-like systems. Several courses have been instituted in this area and others have been modified or updated to bring about what we feel is a reasonable balance between the usual treatment of continuous or analog systems and the treatment of discontinuous or digital systems. Related laboratory studies have been enhanced by the purchase of a small digital computer (PDP-8) and the introduction of this machine into a system which permits physical access to all of the essential computer functions. This system has been called a "generalized digital system" since it also incorporates facilities for patching connections to external digital building blocks so that an extension of the computer or an interfacing system may be rapidly established.

Several laboratory experiments and exercises have been developed about this system. Some of these are extremely simple exercises which serve to establish familiarity with the machine, its coding, logic levels, etc. Some experiments are rather sophisticated real-time data processing adventures. These experiments were designed to support a number of course activities at quite different levels.

In this paper, this generalized digital system will be described and several example problems outlined. These examples are by no means exhaustive; they have been chosen to illustrate the range of levels which may be encompassed using the experimental system, the versatility of the system, and to have an example from several of the particular related course areas.

THE GENERALIZED SYSTEM

The generalized system is conceptually and practically simple. Central to the system is a Digital Equipment Corporation PDP-8 Computer. A number of supporting digital building blocks are mounted in adjacent frames with a patch panel which permits the rapid establishment of interconnections between these peripheral elements and the computer. All of the usual computer interfacing lines appear as terminated points on this patch panel.

The majority of the logical building blocks are completely unspecified, that is, any logic module may be substituted in the patching arrangement. It has been found that a few specific functions are repeated in a great many interfacing problems, and these functions have therefore been prewired on the patch panel (two binary up-counters and one binary up-down counter). Seven device selectors with precut codes are included in the interface. In addition, two 12 bit switch registers, two 12 bit light registers, some momentary contact switches and free indicator lights are available as a portion of the generalized interface. Trunk lines are available for two 12 bit registers (24 lines) which may be connected to divorced equipment such as analog tape transport, digital tapes, etc.

An analog to digital converter is included in the system. Present planning includes the addition of a multiplexer and facilities for digital to analog conversion. The system is by no means static: we are presently adding additional equipment racks for the inclusion of micrologic circuits. In this subsection of the system all computer interfacing lines are level converted to the appropriate voltages for connection to this logic line.

EXAMPLE EXPERIMENTS

As previously indicated, several example experiments are outlined in this section. When the system was first conceived, the faculty felt responsible for specification of a number of problems to be implemented on this system. We felt that we would be hard pressed to find a sufficient number of examples to insure full utilization of the system. However, the students have suggested problems which cover the gamut from time-sharing and multi-programming to automatic control of the coffee pot. Most of the following examples were chosen from student suggested projects.

Experiment A

The Electrical Engineering Department is responsible for the instruction of computer science majors of the College of Arts and Science in a course that is oriented towards the hardware and architecture of computing systems. For the most part these students will have had no experience with a digital computer, at a more intimate language level than Fortran. We find that a simple machine language program tracing experiment is extremely effective in establishing the system remoteness of the Fortran language. A simple type-out routine is coded in Fortran; this program is compiled and loaded along with the Fortran operating system. The routine is then executed in a single step machine language mode so that all of the required steps of masking, code conversion, communication with a peripheral device, etc. may be examined. This experiment is, of course, extremely simple; however, it does illustrate the fact that the generalized digital system finds considerable use even at early instructional levels.

Experiment B

The computer science students soon become moderately proficient assembly level language programmers on this machine. This is not a part of the course per se; but the relation between "hardware" and "software" which is discussed, quite often naturally brings up coding problems. Near the end of the course they are capable of more ambitious experiments in which additional commands are added to the repertoire of the computer. An example of this is the addition of a "hardware" inclusive OR command. In this experiment a program controlled input-output transfer is initiated to transfer the contents of two core memory locations to external registers. The peripheral portion of the system exclusive OR's these registers and transfers the data back into the accumulator of the computer. Now, of course, the computer can accomplish a similar end result with a subroutine of some 15 or so statements. The student is thus faced with a real example of what is often called the "hardware-software" trade off.

Experiment C

Certain electrical engineering courses place emphasis on the electronic circuitry involved in computers. A design problem in this area is assigned in which the students must do a rather complete "worst-case" design of a discrete element NOR gate. This design requires that a certain fan-in, fan-out requirement be met with any transistor from a given distribution. The computer system is used in the evaluation of this design, that is, in testing of the circuits. The students use the University Central Computing Center in the design computations. The generalized system is used in testing the physical circuit which they have designed and built. These circuits are plugged into the system interfaced and output loads are connected by the computer while the circuit voltages are tested with the analog to digital converter. The computer gives the student a grade on the lab experiment which indicates how well he met the design objectives.

Experiment D

A course discipline area is developed in the theory of simple sequential systems. As an example problem, an asynchronous, sequential, single error correction, coder and decoder are realized using NAND gates. This subsystem is patched into the generalized system and the computer is used to generate code groups which are transmitted to and received from the transmission system. A random error generator (computer subroutine) creates errors in the transmission path. The computer further analyzes the transmission and reports the performance statistics of the transmission system.

Experiment E

The single cycle and three cycle data-break transfers are difficult concepts for the students to assimilate. This is not because they are conceptually difficult but because of the large number of signals which must be recognized and carefully timed. A simple experiment serves to illustrate both of these data-break facilities. We call this experiment a hardware clear core. In this interface the single cycle data-break is first called to set 0 into core location 0 and 1 into location 1. The three cycle data-break is then initiated with the word count register as location zero accompanied by presentation of all zeroes on the data lines. This has the net effect of clearing the remaining core locations. The single cycle data-break may again be called to clear location 0 and 1 if total core clearing is required. However, the application here is to illustrate the data-break facilities.

Experiment F

A number of student projects are being executed using this generalized system. In project courses of this nature, rather comprehensive problem areas are suggested to the students. They may then pursue a solution of the problem for either one or two terms of their senior year. One example of such a problem is a pulse height analyzer. This problem will be described in a bit more detail than have the previous problems, since it serves to illustrate the students approach and the analyzer may be of some general interest.

This pulse height analyzer is a bit unique in that the pulses are of only about 30 nano-seconds duration and the counting interval must be short (about 50 micro-seconds) with no gaps between successive count intervals. Two senior students have solved this problem by building an asynchronous sequential circuit which transmits an output pulse whenever its input pulse meets the proper amplitude criterion. A description of this system is shown in Figure 1. Two comparators (DEC-W520) are used as decision elements to determine if the input signal has passed either the low threshold voltage V_L, the high threshold voltage V_H, or both. The results of these decisions, that is, the output of the comparators, are described by Boolean variables H and L. A flow table which describes the required circuit action for a variety of input sequences is shown in Figure 1. (Note that flow tables of this type are described in Reference 1).

This flow table may be successfully assigned internal state variables f_1 and f_2 as shown. From the flow table, the excitation table of Figure 1 may be derived. From these tables, the excitation functions (F1 and F2) and the output function (Z) may be derived.

A logic diagram which will realize these excitation and output functions using NOR elements is shown in Figure 2. Figure 3 indicates the performance of this pulse height detector in response to rather narrow in time pulses. Notice that pulses less than the low threshold produce no output as is also true for pulses greater than the high threshold. Pulses with amplitudes between these thresholds produce standard 100 nano-second output pulses.

These output pulses are directed to one of a pair of up-counting registers in a sequential sub-system. These registers alternately store the count for the appropriate counting interval and then dump the stored count directly into a core memory location using the computer three cycle data-break facility. The entire analyzer is patched on the generalized system. The computer controls the count interval and keeps track of the appropriate core locations.

At the moment the two threshold voltages (V_H and V_L) are manually set. We have proposed to add digital to analog conversion facilities to the system so that the computer may be used to control the amplitude thresholds which establish the pulse height criterion.

The support software for this problem has also been developed by the students. In this instance a rather short symbolic program suffices to accumulate the data and simply punch it out for later entry into a larger computer for analysis. In this sense, this system is functioning as an online data retrieval system for later offline processing.

Experiment G

A final example problem is a shock measurement system. In this system two pressure transducers are mounted on a moving vehicle. A shock wave is transmitted past these two transducers. The relative time of arrival of the shock wave at each transducer and the length of shock duration at each transducer is measured by a system which is attached to the moving vehicle. This portion of the system further converts this information for transmission over a telemetry link to a receiver. The typical input sequences shown in Figure 4 represents a possible received signal in this system. The time T_0 to T_1 represents the shock duration time on one transducer while the time T_2 to T_3 represents the shock duration time on the other transducer. The physical reasoning is not important to our discussion, but the times of interest are the time difference T_0 to T_1 and T_0 to T_2. In some instances, for example, the second typical input sequence, T_2 may precede T_0. Notice that the two transducers modulate the signal differently so that it is always possible to identify T_0 as an amplitude increase of two units while T_2 results in an amplitude increase of one unit. Typical order of magnitude times for these events are T_0 to T_1 about 200 to 400 μs and T_0 to T_2 from about 800 μs to -300 μs. It is deduced from other engineering calculations that ±1 micro-second would yield sufficient accuracy in the measurement of these time durations.

The received signal is fed to three comparators with three threshold voltages established for these comparators. The comparators then yield decisions about crossing threshold level V_A as a Boolean variable A, V_b as variable B and V_c as variable C. These inputs are further decoded to produce the Boolean variables X_1,X_2,X_3 and X_4 which indicate respectively the number of thresholds which have been crossed. These signals and their logical decoding are all shown in Figure 4. The information of interest could be recovered if these X variables were fed to a subsystem which produced 1 megocycle output pulses on 3 lines called Z_1, Z_2 and Z_3. The Z_1 output should then drive an up-counter which records T_0 to T_1 time differences. The Z_2 output should drive the up count line while Z_3 drives the down count line of an up-down-counter which records T_0 to T_2 time differences.

A natural solution to this problem then is suggested as a synchronous or clocked sequential system. A flow table for such a system is shown in Figure 5. (Note that this flow table must be interpreted differently from the previous flow table and is described in Reference 2). The clock is not shown in the flow table; it's operation being understood. A state assignment is executed and the excitation tables, also shown in Figure 4, are derived from this flow table. These excitation tables are of the type described by Marcus.[2] From these tables the excitation and output functions, shown in Figure 5 may be derived. If these functions are realized using the standard Digital Equipment Company's flip-flops and gates, a possible logic diagram is as shown in Figure 6.

This subsystem does not complete the shock measuring system. These outputs Z_1, Z_2 and Z_3 are fed to two counters or registers which, upon completion of an experiment, store the register contents in specified core locations by calling the three cycle data-break facility. The total experiment consists of observing several hundred of such shock waves which are generated in bursts at a possible rate of some 6000 shocks per minute.

The support programming for this system was also executed by the students. In this instance considerable calculation must be made with the data. It was felt that the Fortran language was a more efficient vehicle for such calculations. Hence, the programming problem became one of establishing proper linkage to control the interface and data-break entry within the framework of Fortran.

CONCLUSIONS

The present status of this system then is one in which a number of experiments have been developed in support of a rather large variety of course work. A tremendous possibility exists for future developments of this sort. That is, the system configuration is sufficiently versatile so that only lack of the students imagination precludes his open-minded approach to a problem. It thus seems that this modest investment has sparked considerable interest and serious thoughts.

There is a major drawback to our present system configuration. When the students actually become involved, the interest is dampened by a tedious wait for the machine. The student must suppress his desire to see his system work while he waits a considerable time for the compiler or assembler or system to be loaded into the computer. We are anticipating remedying this situation by the addition of a high speed reader and punch and perhaps a disc pack in the very near future. Even with this present drawback, we find this system is effective. It is used in regularly scheduled laboratories which support this effort; these are only really effective for some of the early very short experiments or for demonstrations. This system is at all times available to the students. It operates in open laboratory environment. We have scheduled courses which would usually be involved in the use of this equipment in alternate semesters to help alleviate the timing problems.

REFERENCES

1. Maley, G.A. and Earle, John, The Logic Design of Transistor Digital Computers, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1963.

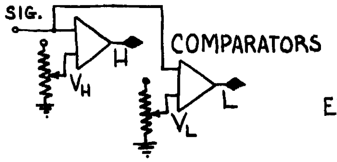
2. Marcus, M.P., Switching Circuits for Engineers, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1962.

f_1f_2 \ HL	00	01	10	11	Z
00	①	2	4	5	1
01	3	②	4	5	1
11	③	2	4	5	0
10	1	⑥	④	⑤	1

FLOW TABLE

f_1f_2 \ HL	00	01	10	11
00	00	01	10	10
01	11	01	10	10
11	11	01	10	10
10	00	10	10	10

EXCITATION TABLE



EXCITATION FUNCTIONS

$$F_1 = H + L' f_2 + L f_1 f_2'$$

$$F_2 = H' f_2 + H' L f_1'$$

$$Z = (f_1 f_2)'$$

FIGURE 1 PULSE HEIGHT DETECTOR - DESCRIPTION

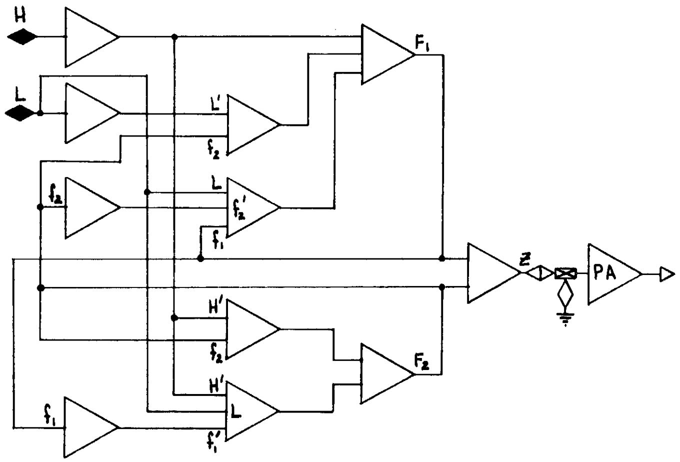
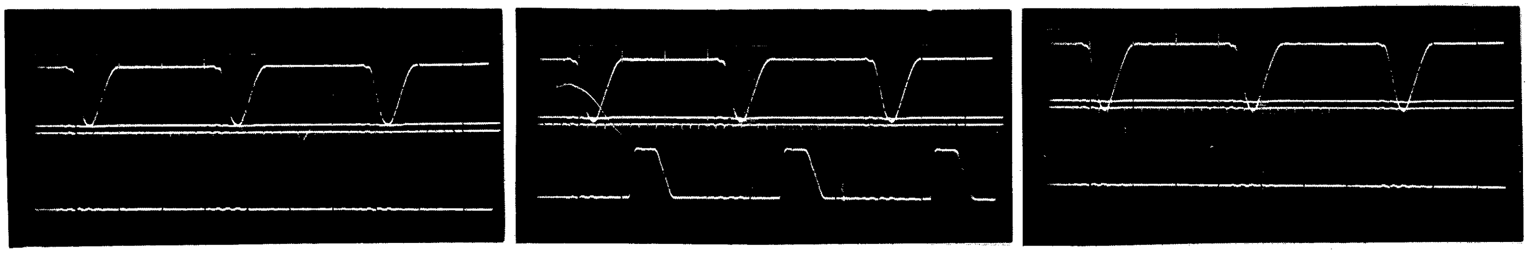


FIGURE 2 PULSE HEIGHT DETECTOR - LOGIC DIAGRAM



PULSE LESS THAN LOW THRESHOLD

PULSE SATISFIES COUNT CRITERION

PULSE GREATER THAN HIGH THRESHOLD

TIME SCALE 100 ns/CM
AMPLITUDE 500 MV/CM
(OUTPUT 2V/CM)

FIGURE 3 PULSE HEIGHT DETECTOR - PERFORMANCE

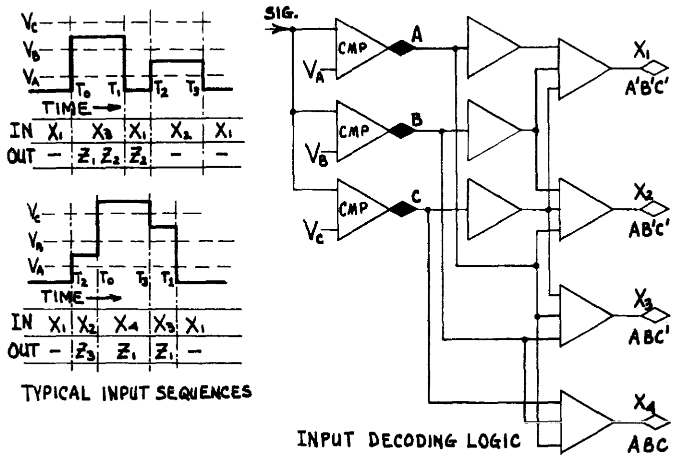


FIGURE 4 SHOCK MEASUREMENT SYSTEM - FRONT END

$y_1 y_2$	X_1	X_2	X_3	X_4
00	00^-	10^3	$01^{1,2}$	11^1
11	00^-	11^-	11^1	11^1
01	01^2	11^-	$01^{1,2}$	11^1
10	10^3	10^3	11^1	11^1

ASSIGNED FLOW TABLE

		X_1		X_2		X_3		X_4	
	$y_2 \backslash y_1$	0	1	0	1	0	1	0	1
FF_1	0	0	1	1	1	0	1	1	1
	1	0	0	1	1	0	1	1	1
FF_2	0	0	0	0	0	1	1	1	1
	1	1	0	1	1	1	1	1	1

EXCITATION (MARCUS) TABLE

$S_1 = X_2 + X_4$ $\qquad R_1 = X_1 y_2$

$S_2 = X_3 + X_4$ $\qquad R_2 = X_1 y_1$

R-S FLIP-FLOP

EXCITATION FUNCTIONS

$Z_1 = X_3 + X_4$

$Z_2 = X_3 y_1' + X_1 y_1' y_2$

$Z_3 = X_2 y_2' + X_1 y_1 y_2'$

OUTPUT FUNCTIONS

FIGURE 5 SHOCK MEASUREMENT SYSTEM - DESCRIPTION

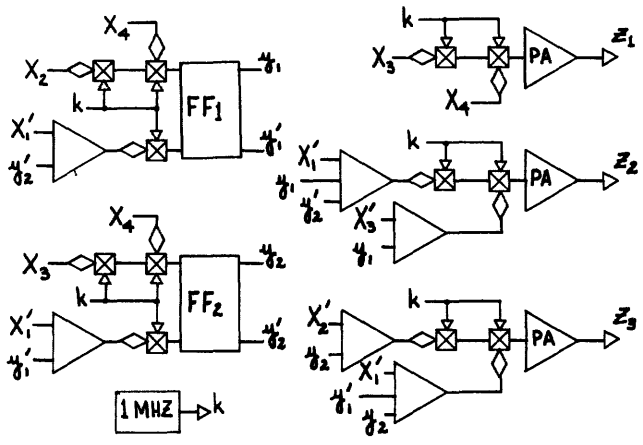


FIGURE 6 SHOCK MEASUREMENT SYSTEM - LOGIC DIAGRAM

COMPUTER TYPESETTING OF MATHEMATICAL TEXT: THE INPUT LANGUAGE PROBLEM

Richard J. McQuillin
Inforonics, Inc.
Cambridge, Massachusetts 02139

ABSTRACT

This paper presents some results of research in computer typesetting of mathematical text. In particular, attention is given to the representation of complex symbolism using a conventional keyboard. Emphasis is on how keying conventions can be established to provide an input system that is useable by the editorial staff of a publisher of mathematics articles.

Experimental results are given based on a test sample using these keying conventions. The results show how the system can be utilized to computer typeset Mathematical Reviews.

An extension of the symbol representation scheme is presented, whereby complex two-dimensional mathematical expressions may be conveniently expressed and proofread at the input keyboard.

INTRODUCTION

There are a number of levels of complexity in information processing in the environment of a scientific professional society. At one level is the roster of members; at another the processing of bibliographic headings and indexes in a review journal; and yet another in the processing of scientific journal information in general. It has been known for a long time how to maintain a computer file with names, addresses, affiliations, etc. Mailing lists are usually printed using an upper case line printer. Many society membership lists are maintained and published in this manner. Comparatively recently there have been two (at least) new hardware developments that have influenced the storage of information within computer files: the upper and lower case line printer and the high speed photocomposition devices. Typical of this latter development are the inexpensive and rapid Photon Model 713 and the very high speed Photon 900, Merganthaler Linotron, RCA Videocomp, and Alphanumeric Photocomposition System. In order to produce output worthy of these devices it is necessary to store two more items of information: case shifts and diacritical marks. At this point it is possible to represent all output codes on the input keyboard. Typical input devices are the Friden Flexowriter and the Dura Mach 10 tape punching typewriters. With this degree of sophistication it is possible to produce good quality membership rosters.

The next thing that may be automated is the production of bibliographic headings, such as would appear in Mathematical Reviews, or indexes. From now on we will be talking photocomposition exclusively, and immediately there is a new dimension, font shifts. The author's name and title of his work are typeset in bold face, while the source document is typeset in italic font. This font information is acquired by a typesetting program in two ways: explicitly and implicitly. Most of the font information is acquired implicitly in the Inforonics' Text Processing System. The program "knows" when it is processing the title, for example. However, computer typesetting is a world of non-conformity, and there are times when the implicit method is not enough. The keyboard operator must have the capability of explicitly specifying a font. This is done in a manner similar to the ASCII specifications. (See reference[1].) A special code is assigned the keyboard as an escape code. Thus the symbol following an escape code will specify a new font. If the escape code is the graphic symbol □ , we may specify the italic font by □ i. The escape code concept may be generalized to include non-Roman alphabets such as Greek and mathematics. In this way the input keyboard may be used to represent many, many symbols. An example of a mathematics keyboard is given in Figure 1. This, then, gives the capability of composition of bibliographic headings and indexes.

At this stage, we have been discussing aspects of traditional text processing. Clearly if we are to compose general mathematical expressions, something more must be considered: two-dimensionality. There must be some way of expressing the relative position of one symbol to another. We have developed a language to deal with this problem called STIL, the Standard Typesetting Input Language. Since the problem of computer composition is open-ended, the language is also open-ended. It also follows that any translator to process the language must be extendable (self-extending).

An important constraint on any language of this type is that it must be easy to use for average editorial personnel; that is, non-programmers. This research was initiated with the hope of evolving the composition programs into a production system. Time-motion studies were carried on at the American Mathematical Society to determine keyboard conventions. Several different keyboard languages were studied with the hope of optimizing, keying, proofreading, and error correction.

THE MATHEMATICAL REVIEWS EXPERIMENTS

In order to test our concepts, a sample of bibliographic headings were keyed at the Mathematical Reviews Office in Ann Arbor, Michigan. Some 2000 headings were keyed, using a Dura Mach 10 tape producing typewriter. The information was keyed in a format similar to the existing MR card files. Therefore, the information had to be analyzed implicitly. Thus, if the title had been identified, then the following possibilities could occur: (1) two spaces would introduce the language and/or summary statement; (2) carriage return-tab would introduce the source document; and (3) carriage return n: - tab would introduce notes. The entire process can be described in a recognition diagram, as given in Figure 2.

It turns out that as the recognition diagram gets more complicated, the chance of error increases. As long as it is simple it is an effective means of item recognition. However, in the test sample, it was found that there were some 300 headings that had at least one format error. Presumably the percentage would decrease with increasing experience of the keyboard operator, however.

A second method of item recognition is the explicit tag. In order to test this method, some 96 Mathematical Reviews headings were prepared in the American Mathematical Society Office in Providence, Rhode Island. Here we have a two column format where the left column contains the tag identifiers, and the right column the information. Thus, the left column might contain the tag doc, which the right column J. London Math. Soc. A sample of bibliographic headings prepared in this way is given in Figure 3.

Another purpose of the test information keyed at Providence was to investigate problems of keying mathematics text, as would appear in the text part of a mathematical review. In the test, the text information was keyed separately from the headings. The two were then merged together after they were identified. A sample of text, math review number, and subject classification number is given in Figure 4. The samples were selected from an actual issue of Mathematical Reviews (Vol. 33, No. 6, June 1967) under the following constraints: (1) no multiple line mathematical expressions; (2) no symbols that were not on the existing AMS Photon 200 disc.

In the sample in Figure 4, one sees the use of the font shift. For example□i F (x,l) □n shifts into italic font and then back to normal font (for that item). The information is then typeset to give an output as shown in Figure 5.

In general, it was concluded that a great deal of mathematics text could be composed using a bibliographic typesetting programs but in general some means had to be developed to handle multiple line mathematic expressions before a product system to do anything beyond an index would be practical.

GENERAL MATHEMATICAL TYPESETTING SYSTEM

It was clear from experiments with Mathematical Reviews text that a new composition system had to be designed to handle mathematical text if any type of production system was to be realized. Indexes can be composed for a good percentage of the time, but occasionally titles contain complex mathematical expressions.

The problem of creating a production system for mathematics text may be thought of in two parts: (1) design of a keyboard language; (2) design of computer programs to accommodate an essentially open-ended computer language. A computer system to process mathematics text is given in Figure 6. It consists of three main parts:

A. The Translator translates codes and symbols of the keyboard language into a Standard Typesetting Input Language, called STIL.

B. The Preprocessor, or Scanner, translates mathematical text as expressed in STIL into printing codes, suitable to drive a photocomposition machine. It also transmits spatial information from the keyboard language into the typesetter, as well as handling macros.

C. The Typesetter, or Composer, composes the mathematical text into a format suitable to drive a photocomposition machine.

THE KEYBOARDING LANGUAGE

The keyboard language has been designed in a two-column format. The right column always contains straight text matter, while the left column contains symbols of various types. When the input information is being processed, the translator processes the information on a symbol-by-symbol basis. The following symbol types can exist:

A. Mnemonics - Mnemonics are used to represent printing symbols. When a mnemonic is seen it is translated into one of the standard fonts of STIL.

B. Primatives - A primative is a parameter of the typesetter. For example, line measure, point size, page size, etc.

C. Composition Operators - The composition operator is used to call procedures within the typesetter. For example, sub and sup invoke routines within the typesetter which compute the proper spatical orientation for subscripts and superscripts.

D. Action Operators - The action operator, when seen invokes some procedure within the preprocessor. For example, new symbols may be defined with the define action operator, and jobs are terminated with the finis action operator.

E. Hybrid Operators - The hybrid operator is a combination of the composition operator and the action operator. When it is encountered, procedures are invoked in both the typesetter and the preprocessor. For example, the hybrid operators author, title, footnotes require processing in both programs.

F. Macros - Macros may be used in the keyboard language as a shorthand notation. Obviously, when there are repetitive groupings of symbols they need not be keyed over and over again. A production system must contain macro capability. As the keyboard operator became more experienced, one would expect more use of macros.

A representative list of these symbol types is given in Figure 7. A sample of input keying is given in Figure 8.

THE STANDARD TYPESETTING INPUT LANGUAGE STIL

In order to do research in keyboard languages, it is necessary to have some invariant language which most of the system can process. The concept is to have any keyboard language immediately translated into the standard language. The standard language STIL has been specified. It is a functional language, and operations are represented in functional notation for ease of processing. STIL is not intended to be especially easy to proofread. The language may be considered as two things: operators, or functions, and operands. The method of the ASCII escape codes was used to identify operators. If we denote the ESC code by the graphic □, then all operators are preceeded by a □. In addition to operators following the □, there may be parentheses □(, □) and separators □, . Thus the mathematical expression:

$$\frac{a + b}{a - b}$$

would be represented in STIL as:

□div □(a + b □, a - b □).

(Here the □ ( ) and , are overprinted for readability.)

Characters are represented by font overlays in STIL. For example, the ₵ might be represented as □4c, where it would be in the c position of the 4th font. There are 200_8 symbols per font.

SYSTEM EXTENDABILITY

The composition system has been designed to be easily extended. The mechanism for this has been the define action operator. A number of things can be defined: macros, composition operators, action operators, hybrid operators, character tables, and symbolic locations within the Preprocessor and Typesetter. With this capability new and more sophisticated definitions can be built up from previous definitions.

ACKNOWLEDGEMENTS

The author wishes to express graditude to the staffs of the American Mathematical Society and Mathematical Reviews for their assistance in language specifications and test information input preparation. The work was supported by Grant GN-583 to the American Mathematical Society by the National Science Foundation.

REFERENCES

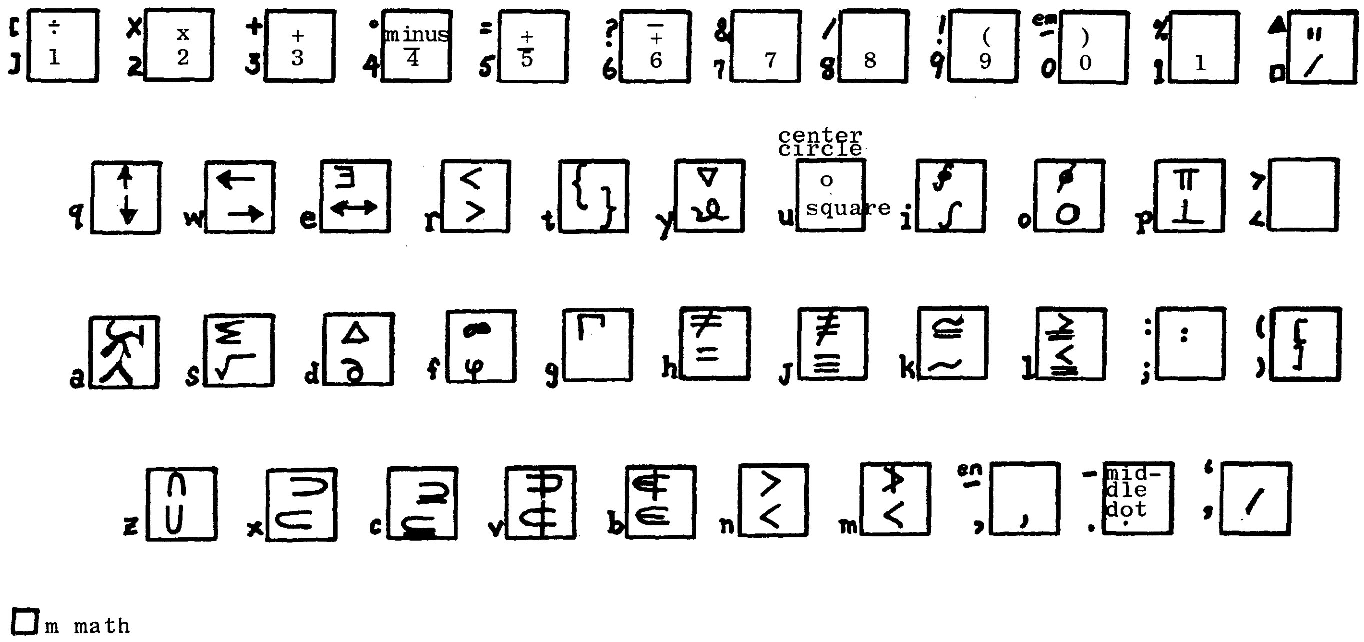
1. "USA Standard Code for Information Interchange" Document USAS X3.4 - 1967. United States of American Standards Institute, 10 East 40th Street, New York, 10018.

2. "Development of Computer Aids for Tape-Control of Photocomposition Machines: Report No. 1, Mechanization of Mathematical Reviews Office Procedures," American Mathematical Society, Providence, Rhode Island, March 1967.

3. "Development of Computer Aids for Tape-Control of Photocomposition Machines: Report No. 2: Extension of the System of Preparing Computer-Processed Tape to Include the Setting of Multiple Line Equations." American Mathematical Society, Providence, Rhode Island, July 1967.

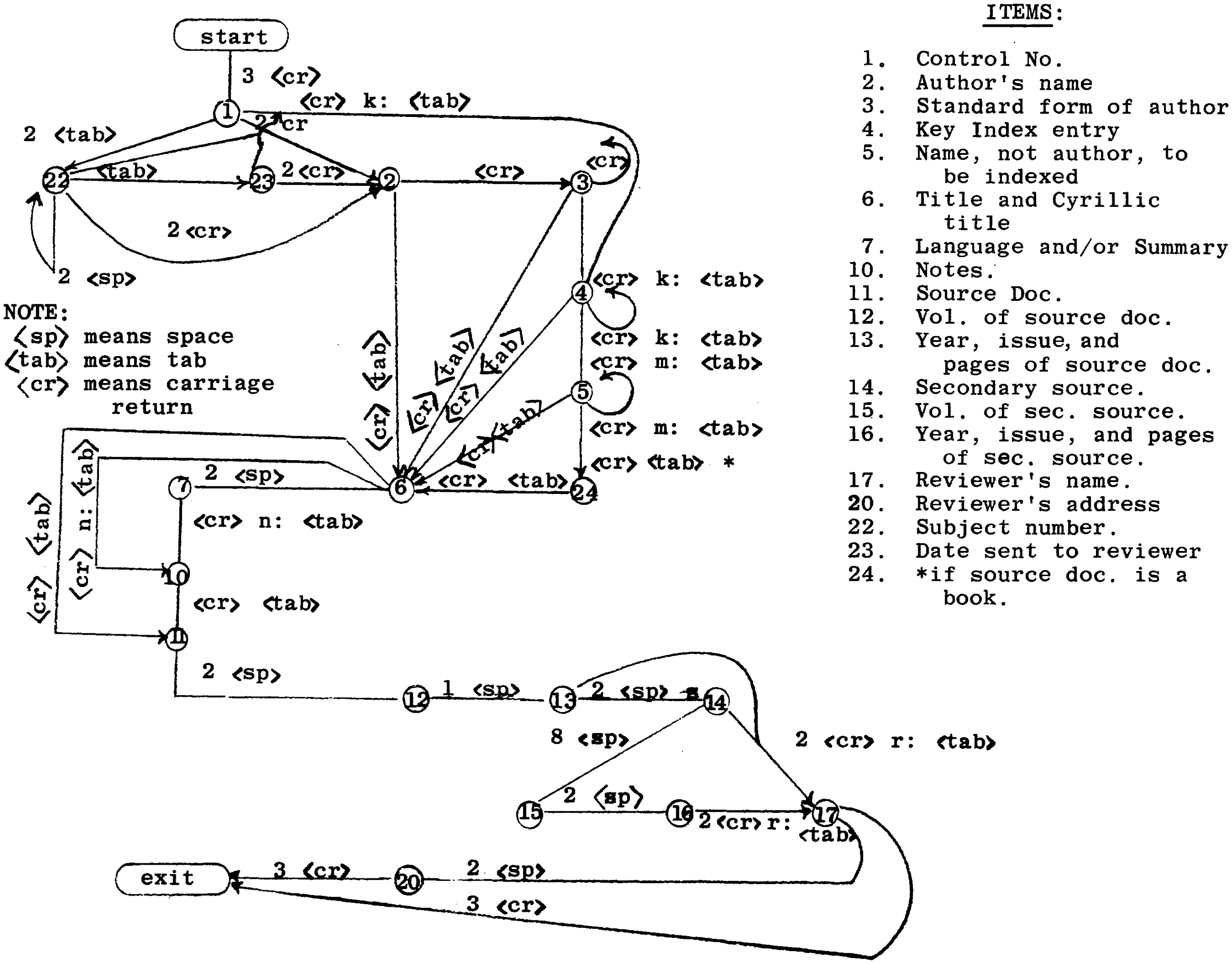
4. R. J. McQuillin, "Development of Computer Aids for Tape-Control of Photocomposition Machines: Part C, Computer Typesetting of Multiple Line Expressions" Inforonics, Report to the American Mathematical Society, March 1968.

AMS KEYBOARD OVERLAYS



MATHEMATICS FONT

Figure 1. AMS Keyboard Overlays



NOTE:
<sp> means space
<tab> means tab
<cr> means carriage return

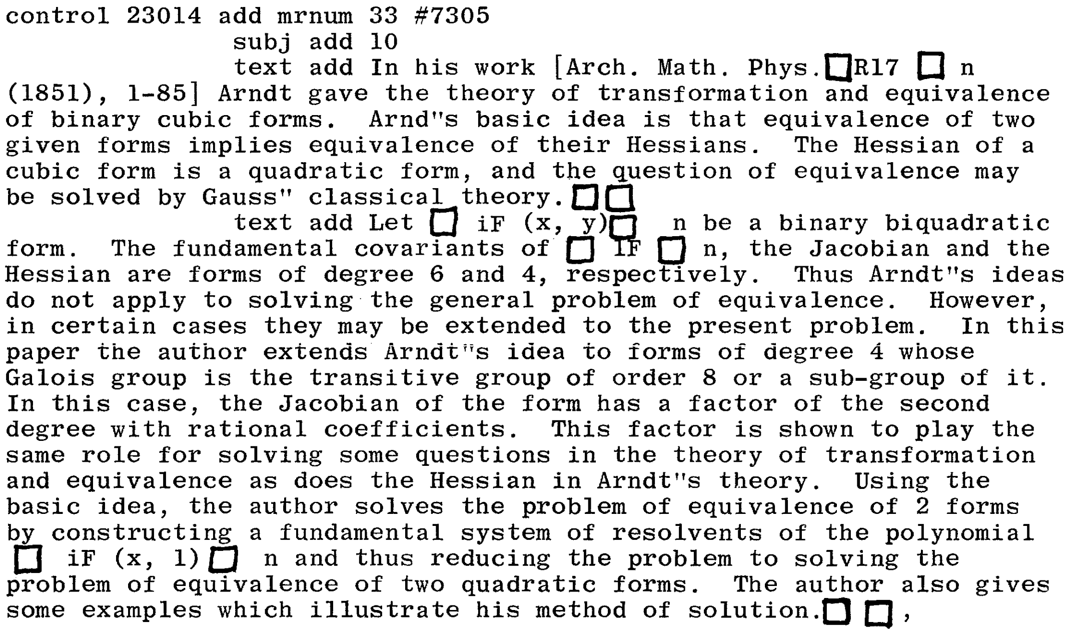
ITEMS:

1. Control No.
2. Author's name
3. Standard form of author
4. Key Index entry
5. Name, not author, to be indexed
6. Title and Cyrillic title
7. Language and/or Summary
10. Notes.
11. Source Doc.
12. Vol. of source doc.
13. Year, issue, and pages of source doc.
14. Secondary source.
15. Vol. of sec. source.
16. Year, issue, and pages of sec. source.
17. Reviewer's name.
20. Reviewer's address
22. Subject number.
23. Date sent to reviewer
24. *if source doc. is a book.

Figure 2. Recognition Diagram for Mathematical Reviews Information Input

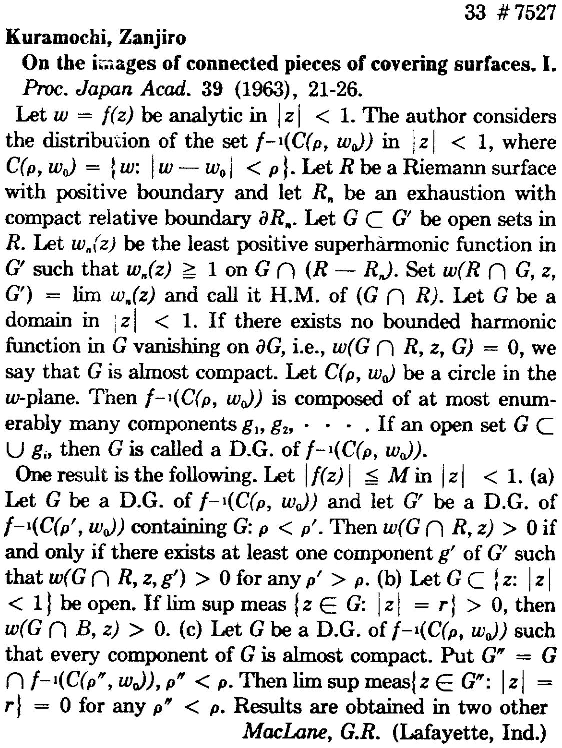
cont	32021
aut	Cossels, J. W. S.
title	Diophantine equations with special reference to elleptic curves.
doc	J. London Mathematical Society
val	41
yr	1966
iss	193-291
rev	Lewis, D. J.
reva	(Ann Arbor, Michigan)

Figure 3. Input by Explicit Tagging



```
control 23014 add mrnum 33 #7305
               subj add 10
               text add In his work [Arch. Math. Phys.□R17 □ n
(1851), 1-85] Arndt gave the theory of transformation and equivalence
of binary cubic forms.  Arnd"s basic idea is that equivalence of two
given forms implies equivalence of their Hessians.  The Hessian of a
cubic form is a quadratic form, and the question of equivalence may
be solved by Gauss" classical theory.□□
               text add Let □ iF (x, y)□  n be a binary biquadratic
form.  The fundamental covariants of □ IF □ n, the Jacobian and the
Hessian are forms of degree 6 and 4, respectively.  Thus Arndt"s ideas
do not apply to solving the general problem of equivalence.  However,
in certain cases they may be extended to the present problem.  In this
paper the author extends Arndt"s idea to forms of degree 4 whose
Galois group is the transitive group of order 8 or a sub-group of it.
In this case, the Jacobian of the form has a factor of the second
degree with rational coefficients.  This factor is shown to play the
same role for solving some questions in the theory of transformation
and equivalence as does the Hessian in Arndt"s theory.  Using the
basic idea, the author solves the problem of equivalence of 2 forms
by constructing a fundamental system of resolvents of the polynomial
□ iF (x, 1)□  n and thus reducing the problem to solving the
problem of equivalence of two quadratic forms.  The author also gives
some examples which illustrate his method of solution.□ □,
```

Figure 4.   Text Input for Mathematical Reviews



33 # 7527

Kuramochi, Zanjiro

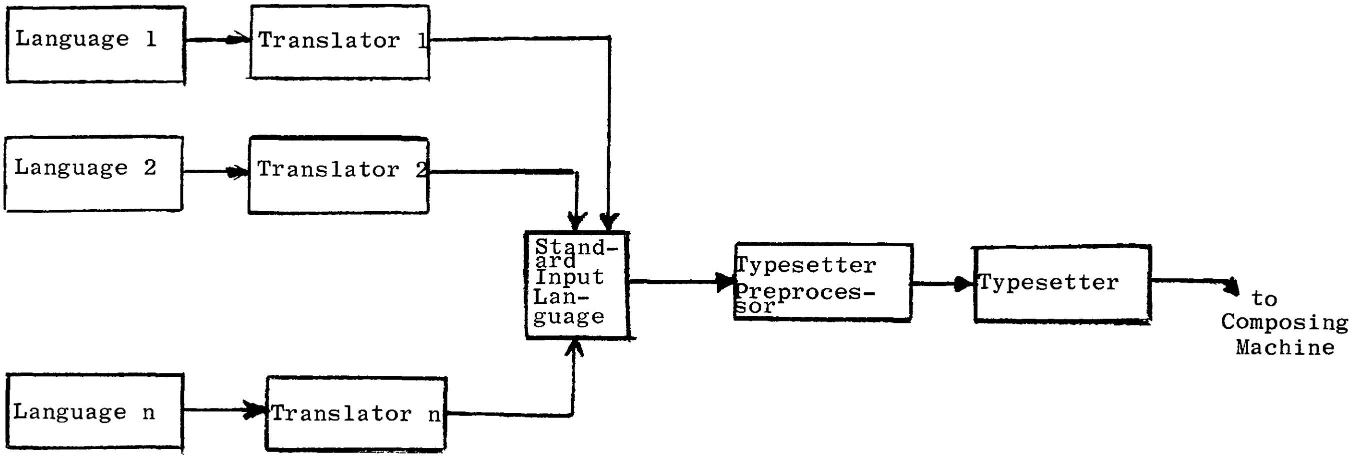
On the images of connected pieces of covering surfaces. I.
Proc. Japan Acad. 39 (1963), 21-26.

Let $w = f(z)$ be analytic in $|z| < 1$. The author considers the distribution of the set $f^{-1}(C(\rho, w_0))$ in $|z| < 1$, where $C(\rho, w_0) = \{w: |w - w_0| < \rho\}$. Let R be a Riemann surface with positive boundary and let R_n be an exhaustion with compact relative boundary ∂R_n. Let $G \subset G'$ be open sets in R. Let $w_n(z)$ be the least positive superharmonic function in G' such that $w_n(z) \geqq 1$ on $G \cap (R - R_n)$. Set $w(R \cap G, z, G') = \lim w_n(z)$ and call it H.M. of $(G \cap R)$. Let G be a domain in $|z| < 1$. If there exists no bounded harmonic function in G vanishing on ∂G, i.e., $w(G \cap R, z, G) = 0$, we say that G is almost compact. Let $C(\rho, w_0)$ be a circle in the w-plane. Then $f^{-1}(C(\rho, w_0))$ is composed of at most enumerably many components $g_1, g_2, \cdot \cdot \cdot \cdot$. If an open set $G \subset \cup g_i$, then G is called a D.G. of $f^{-1}(C(\rho, w_0))$.

One result is the following. Let $|f(z)| \leqq M$ in $|z| < 1$. (a) Let G be a D.G. of $f^{-1}(C(\rho, w_0))$ and let G' be a D.G. of $f^{-1}(C(\rho', w_0))$ containing G: $\rho < \rho'$. Then $w(G \cap R, z) > 0$ if and only if there exists at least one component g' of G' such that $w(G \cap R, z, g') > 0$ for any $\rho' > \rho$. (b) Let $G \subset \{z: |z| < 1\}$ be open. If lim sup meas $\{z \in G: |z| = r\} > 0$, then $w(G \cap B, z) > 0$. (c) Let G be a D.G. of $f^{-1}(C(\rho, w_0))$ such that every component of G is almost compact. Put $G'' = G \cap f^{-1}(C(\rho'', w_0))$, $\rho'' < \rho$. Then lim sup meas$\{z \in G'': |z| = r\} = 0$ for any $\rho'' < \rho$. Results are obtained in two other

MacLane, G.R. (Lafayette, Ind.)

Figure 5.   Typeset Review



THE MATHEMATICAL TYPESETTING SYSTEM

Figure 6. The COMPOSE System

Mnemonics

alpha

gera

mdash

less

thin

Composition Operations

sub

sup

int

div

Macros

m(x,y,z)

Action Operators

define

fix

finis

Hybrid Operators

dpy

dit

author

title

Primatives

ls 10

meas 260

it

Figure 7. COMPOSE System Symbol Types

VERBALIZATION

Title L sup p	behavior of power series with positive coefficients
Author	Richard Askey
footnote number (1)	
footnote	Supported in part by N. S. F. grant GP-3483.
return para	Heywood
obrak bo 4 return cbrak	and others have considered integrability theorems for power series and Laplace transforms that are analogous to known results for Fourier series and transforms. These results are all weighted
L sup 1 baseline	results. Here we obtain an
L sup p baseline	theorem which is analogous to the well-known
L sup p baseline	result of Hardy and Littlewood
obrak bo 3 return cbrak.	
lead six para sm	Theorem.
ital	Let
f (x) = sum udr k = 0	
baseline ovr inf baseline a	
sub k baseline x sup k	
baseline, a sub k baseline	
gto 0 , 0 lto x lt 1.	Then for
1 lto p lto inf	
flush left (1) center	

Figure 8. The Two Column Format

USE OF PDP-8 FOR DRIVING PHOTOCOMPOSITION MACHINES*

Richard Falt
Digital Equipment Corporation
Maynard, Massachusetts

Abstract

A brief look at the use of our DECtape and DECtape Disk Systems to produce punched paper tape input to various photographic units will be given.

*This paper was not received for publication.

PDP-9T TIMESHARING: PHASE I, MULTIPROGRAMMING

D. M. Forsyth*
Department of Psychology, Harvard University

M. M. Taylor
Defence Research Establishment Toronto

S. Forshaw
Defence Research Establishment Toronto

ABSTRACT

The PDP-9T is a PDP-9 with the addition of paging hardware, special traps, and modifications which translate IOT instructions into specific calls to the system monitor. Phase 1 of software development for the system permits several processes to occupy core simultaneously. All input/output is handled by the system monitor. Real-time tasks have high priority and are generally interrupt driven, i.e., are activated as necessary to process data. Background tasks such as editing, assembling and debugging are allocated time by an algorithm which seeks to give all background tasks equal amounts of running time. A total of three to six users may be simultaneously active.

I. HARDWARE SPECIFICATIONS

The PDP-9T is a standard PDP-9 to which certain modifications have been made.[1] These modifications affect the way in which memory is addressed, the trapping of certain instructions, and the handling of the IOT instruction.

Memory addressing has been modified by introduction of a paging system. Memory is organized in pages, each page 2048 words in length. The high-order four bits found in the Memory Address Register (MAR) on each memory fetch refer to a virtual page, and are used to specify one of 16 mapping registers. The contents of this register point to the real page of core in which the system has placed the virtual page specified by the users program. The low order 11 bits in the MAR specify a word within the page. The user's pages may be non-contiguous; the address transformation, or mapping, is automatic and invisible to the programmer. All users have a potential virtual memory of 16 pages (32K). Each mapping register consists of 9 bits. Bit Ø is used to specify Memory Out of Core (MO), indicating that the requested page exists but is not currently in core. Bit 1 is used to indicate Read Only (RO) pages, which may not be modified. These bits will be of particular use to the Phase II System which will swap pages between core and a one-million word disk.

The PDP-9T will operate in one of two modes: User Mode or Monitor Mode. Two separate sets of mapping registers are provided, a User Map and a Monitor Map. The map used on any memory fetch is a function of the mode currently in effect. A few instructions may not be executed in User Mode, and attempts to do so will cause an Error Trap. Two such instructions are HLT and OAS, which have no meaning in a time-sharing environment. The unique nature of the XCT command requires that XCT chains be aborted when an interrupt is pending; the sequence will be trapped before granting the interrupt. Faults related to the memory mapping (MO, RO) will also result in Error Traps.

No IOT commands are permitted in User Mode. The instruction class is preserved, however, and utilized to effect communication with the Monitor: when an IOT abbcc is encountered in user mode, the 8-bit quantity abb is used as a pointer to one of 256 entries in a monitor table. The table entry is executed, providing rapid access to read-only system parameters (e.g., time of day) as well as low-overhead entry to system routines. This class of instructions is relabeled as XMR - execute monitor register.

The software which is sketched in the following paragraphs is being implemented at Harvard and Toronto on 32K machines with EAE, API, 8 or more DECtapes (there are two Tape Control Units on the Harvard system), a Type 340 Display system with character and vector modes (on the Toronto system), 4 teletypes, lineprinter, a 200 usec core clock (the Real-Time Clock), a 1.2 usec peripheral clock (the Quantum Timing Clock), and a variety of other peripherals.

II. PRIORITY ALLOCATION

The goal of all time-sharing systems is the proper allocation of resources (e.g. core, CPU time, I/O channels) among various users so as to optimize,

*Work done at Harvard was supported by Grant 1-R01-GM-15258 from the National Institute of General Medical Sciences.

according to some external criterion, the utilization of these resources. This requires assignment of priority; if two or more processes are ready to run in the same CPU at the same time, one must be given a higher priority than the other or neither will run. The PDP-9T is basically an interrupt-driven machine designed to act in response to signals generated in the external world. An attempt is made to establish priorities of service based on the response latency requirements of each process. Some processes must respond to an external signal rapidly, but execute a small amount of code. Others may be able to defer their response, but will require longer running times.

The standard Automatic Priority Interrupt option on the PDP-9 mediates this type of situation. Four hardware and four software levels of priority are provided, in addition to background (in the PDP-9T, the standard Priority Interrupt is disabled). Eight or more devices may be multiplexed to each hardware level, and any number of queues attached to each software level. Hardware interrupt levels intercept signals from the external world, strobing in data and releasing the signaling devices. Processing of incoming data will ordinarily be deferred to a lower priority routine.

Software priorities in the PDP-9T do not closely parallel the hardware priority structure. Level 4, the highest software level, is reserved for the execution of certain system routines, e.g. those invoked by user's XMR instructions. These will always be short routines, running less than 2 msec. They are placed at this level to avoid the problem of re-entrance. Hardware level coding is forbidden use of these routines. Levels 5, 6 and 7 are used for various monitor functions such as TCB queues, etc. The lowest level in the machine, background, is the level at which all user code is executed. Users are explicitly denied the use of API channels, but may define sections of code which are to be run in response to hardware interrupts. The decision as to when to run a user must be made by the monitor in terms of algorithms which are not hardware implemented.

The distribution of functions by priority in the Phase I System will be as follows:

API Level	Function
Ø	Error Trap handling routines, Disk, Power Failure
1	DECtape, A/D conversion, Fast experimental devices
2	Data-phone, Light Pen, 340 Display, Line Printer, Paper Tape Reader/Punch
3	Teletype LT09A System (to 16 teletypes), 200 msec. Clock, 1.2 msec. Clock, slow experimental devices.
4	Monitor code whose execution may be charged to the user, e.g. I/O setup, arithmetic routines.
5,6	Monitor functions.
7	Hierarchy of queues of user TCBs (see Section III).
Background	All user code.

N.B. All user code is executed at background level, all Monitor code at level 7 or higher.

III. PHASE I MONITOR MODULES

Multiprogramming software is being produced for the PDP-9T as a temporary system. A more sophisticated time-sharing system using page swapping techniques and a back-up disk will be completed within 12-18 months. In the interim, however, a simple multiprogramming capacity will serve two major functions. It will permit three or four users to edit and debug programs while one or two real-time experiments are being run. It will also provide a useful test bed in which to experiment with modules designed for the final system. However, it should be emphasized that it is not the first step in the evolution of a final system. While the Phase I (multiprogramming only) and Phase II (swapping) Systems may have some common components, Phase II is regarded as a totally separate entity.

Each active user in the multiprogramming environment will be represented in the monitor by a User Control Block (UCB). The UCB contains bookkeeping information such as the user's name, time of day when logged in, CPU time used, etc. The UCB also contains information concerning peripheral equipment which has been assigned to the user, such as DECtape units, teletype, etc., and a number of pointers. Two of these are a pointer to a string of Virtual Memory Control Blocks (VMCBs), and a pointer to a string of Task Control Blocks (TCBs).

The VMCB contains the memory map image of the user which must be loaded into the mapping registers at run time. Information necessary to maintenance of this image is also preserved in this area. In the Phase I System, a user will ordinarily have but one VMCB. In the Phase II System, any number are possible.

The TCB contains all of the data relevant to a task (process). The task is the basic element of both the Phase I and Phase II Systems. A task is a logically autonomous sequence of code. Most programs which have been written for the PDP-4/7/9 would be considered as single tasks in this sense; those which utilize interrupts, however, might be regarded as multiple-task systems, consisting of a background task and separate tasks which handle each interrupt. Thus, there may be more than one task in a single virtual memory. In the Phase I System, most users will operate a single task in a single virtual memory, although this will be largely a decision made by the programmer and not dictated by constraints in the system.

The entries in the TCB are those needed to activate the task, such as the address of the VMCB used by the task, and the values of active registers such as the PC, AC, MQ, etc., which must be saved and restored when tasks are de-activated or activated. Each TCB contains the value of the time quantum allocated to the task, a pointer to the UCB to which the task belongs, pointers to routines, which can activate or de-activate the task, and a number of other desiderata.

The disposition of each page in core is recorded in 16 Core Control Blocks (CCBs), and a number of other control blocks are maintained by the Monitor for various purposes. The action of the system can be adequately described, however, in terms of UCBs, VMCBs, TCBs, and the scheduling algorithms which utilize them.

IV. PHASE I TASK SCHEDULING

Scheduling in a time-sharing system is the process by means of which the CPU is made available to tasks in an orderly manner. Each task is allowed to run for a duration determined by its time quantum, at the expiration of which the next task is run. Ordinarily, tasks are organized into running queues, sets of tasks strung together by pointers in the TCBs. A task which has just finished running might be placed at the end of the queue, while the new task at the head of the queue is fired up. Often, particularly in the PDP-9T systems, tasks will run to completion or reach a point at which I/O is required before the time quantum expires. Such tasks are removed from the running queues, placed in a wait condition, and are said to be in a sleep state. When a signal arrives at the CPU for a task which is asleep (e.g. from an I/O completion, or an active experimental device), the task must be awakened and placed in the appropriate running queue. Any number of running queues may be hierarchically arranged to impose a priority structure, establishing a two-fold order in which tasks are to be run (e.g. all tasks in queue A before any in queue B).

Scheduling may conveniently be regarded as a two-part process. First, decisions must be made as to the time quantum to be allocated to each task, when to place a task in a queue, when to remove it, etc. These decisions must then be implemented by a process which activates and de-activates tasks at the appropriate times.

At any given point in time there are a large number of tasks in core, only one of which will actually be running. If it is a user task, running at background level (with respect to the API), it may be temporarily suspended by a hardware interrupt. When the hardware interrupt is serviced, control may return to the interrupted task. Or, the hardware interrupt service may have invoked a software interrupt which will be granted before control returns to the interrupted user task. During execution, the user may issue an XMR to perform some computation, which will be performed by the monitor at API level 4.

Eventually, the user's time quantum will run out, or the user will request a lengthy I/O transfer. In the first case, the user will be placed at the bottom of the running queue. In the second, the user will be 'put to sleep', to be 'awakened' when the transfer is complete. In either event, the CPU will then be given to the task which is next in the priority structure. User's time quanta are metered by the 1.2 usec external clock, which is automatically gated by API activity: the clock runs free at background and at level 4, but does not tick when other API levels are active. Thus the user is charged only for CPU time expended directly in his behalf.

The allocation problem, of course, is the heart of time sharing, and in the PDP-9T is accomplished by a system program called the Operator. In the Phase I Monitor, the Operator performs rather simple chores. Tasks will be given a time quantum (e.g. 20 msec.), and placed on one of several queues. An attempt will be made to distribute CPU time to foreground users (real-time tasks) as needed, and to provide background users with equal amounts of time. It is inevitable, of course, that even with four or five users the system will spend most of its time doing nothing, i.e. waiting for I/O transfers. The Phase I System will generally be core-bound, and active users will receive excellent service.

The Operator will also be responsible for allocating peripheral devices to users, as requested by the user at run time via the teletype. The Operator allocates pages of core to users, runs the user's DECtapes, retrieving and writing user files, and loads system programs such as DDT into the user's virtual memory.

V. THE USER MACHINE

The preceding background is sufficient to permit a brief consideration of the kind of machine an ordinary user will be working with. The User Machine (as it might be specified in a PDP-9T Handbook) has an order code which resembles that of the PDP-9, but is vastly superior. While HLT and OAS have disappeared, along with all IOTs, the XMR instruction class has added such instructions as 'LOG1Ø', 'DIVIDE', 'ARCTAN', etc. All I/O is handled by XMR instructions, such as TDN (type contents of AC as a decimal number), DTREAD (read a DECtape record), PUNCH (punch low order 8 bits of AC, if punch is available to this task). All I/O coding will reside in the monitor, and will be shared by all users.

The fact that the machine is paged is of no interest to the programmer in the Phase I System, although it will become tactically useful in the Phase II System (e.g. fetching pages into core from the disk is time-consuming, and should be avoided even at the expense of duplicating constants in each page).

The user will start a session by logging in, requesting DECtape assignment and such other peripheral devices as may be desired, load whatever programs may be required from his DECtape, perhaps ask for DDT, and proceed as though on an ordinary PDP-9. When finished, he will log out to free the console and core for the next user.

VI. AUXILLIARY SOFTWARE

A number of software packages are being developed for the Phase II System which will be available on the Phase I System (and on most PDP-4/7/9s as well). The necessity of incorporating many features relevant to the time-sharing monitor in the assembler package, as well as the very special requirements on I/O handling, forced us to conclude that it would be more efficient to write our own software than to tailor the Advanced Software Package. Thus we are writing an assembler, linker, DDT, DECtape package, teletype package and disk package, together with a number of other systems of some general utility. These packages will be described in future papers.

[1] PDP-9T: Compatible Time-sharing for the Real-Time Laboratory, M. M. Taylor, D. M. Forsyth & L. Seligman, DECUS Proceedings, Fall, 1967.

EXTENDED MEMORY FORTRAN WITH AN 8K PDP-7[*]

Philip R. Bevington
Physics Department, Stanford University
Stanford, California

Abstract

A hardware modification to the PDP-7 and a FORTRAN subroutine are described which permit the use of Extended Memory coding in FORTRAN II with an 8K memory PDP-7. Normally, this coding permits the storage of large data arrays outside the basic 8K of memory which contains the program and the Operating Time System. In the present system the extra memory is supplied by DECtape. A scratch pad consisting of several pages of 128 words each is retained within the basic 8K memory so that access to the DECtapes is relegated to transfers of blocks. Interpretation of extended memory addresses is accomplished by trapping indirect addresses outside of basic memory and using software to modify these addresses. Such a system permits the use of larger arrays for data manipulation at the expense of time required for DECtape handling. In most cases, however, improved techniques of manipulation through the use of larger and more arrays than offsets this expenditure of time. The philosophy of design and the relative advantages and disadvantages of such a system are discussed.

Introduction

One of the most frustrating disadvantages of using FORTRAN on small scientific computers such as the PDP-7/9 is the inefficient use of computer memory. In FORTRAN II of the basic software for these computers, for example, the operating time system occupies $3k_{10}$ locations in memory, and I/O and arithmetic subroutines can fill another $2k_{10}$ locations. For a computer with an 8k memory this leaves only $3k_{10}$ locations for programming and data storage.

Similarly the version of FORTRAN IV supplied with the Advanced Software Package is so inefficient that even with 16k reasonably sophisticated programs are hard pressed for room for data storage. The obvious solution of increasing the core memory storage is not always economically feasible.

In order to expand the capability of FORTRAN (and of Assembly language programs) we have developed for use with the SCANS (Stanford Computers for the Analysis of Nuclear Structure) system a method of storing and accessing large arrays of data outside of the core memory of a PDP-7. Two types of storage may be utilized by slightly different techniques. The more efficient, but more expensive, is another similar computer, complete with processor and memory. The more easily available type is a conventional bulk storage device, such as DECtape, disk, or drum.

The SCANS system has provision for both of these types of storage, using either an 8k PDP-9 or DECtapes, according to the demands of the particular situation. This report will describe the philosophy and techniques developed to implement Extended Memory FORTRAN II on an 8k PDP-7 with these memory storage devices.

Hardware

In order to provide compatibility with existing software, the entry to bulk storage must simulate the normal use of Extended Memory. But since only data arrays are to be treated in this manner, most of the complexity of the paging hardware of the Extended Memory can be omitted. It is sufficient for most purposes that the interpreting monitor respond to any indirect addressing outside of the conventional 8k of core memory.

In FORTRAN II of the Basic Software System there is provision for storage of arrays of data in Extended Memory. The entire program, including the Operating Time System, must reside in the basic 8k core memory. The specification EXTEND MODE acts in the same way tnat COMMON does to identify arrays which are to be stored in

[*] Supported in part by the National Science Foundation

the Extended Memory. All references to elements of these arrays are by indirect address instructions. The address can be any location up to 32k, using bits 3 and 4 to address locations outside of the basic 8k.

Addresses cannot exceed 15 bits or 32k because the interrupt procedure and JMS instruction utilize bits 0, 1, and 2 to store the contents of the LINK, EM and TRAP. Indirect address instructions such as JMP I ∅ to return from an interrupt must not be interpreted as referring to Extended Memory.

Figure 1 shows how the I/O Trap of the PDP-7 was modified to provide a hardware interrupt under the proper conditions. The Trap flip-flop on the right is controlled by the Trap switch on the console. Actually this flip-flop could have been replaced completely by a single inverter, but since the Trap was already wired it was easier to leave it like this.

If the Trap is enabled by the Trap switch, the Trap flag on the left is set (at time T4 of the cycle) whenever the processor is in the Defer mode (indicating indirect addressing) and either or both bits 3 or 4 of the Memory Buffer (which holds the indirect address) are non-zero. The Trap flag initiates a Program Interrupt with the PC forced to location 2. At time T4 of the Break cycle the Trap flag is reset. All other conventional use of the Trap has been removed.

Figure 2 shows additional modifications to the Central Processor. All normal device flags are gated by the Program Interrupt Enable flag before requesting a Program Interrupt. The Trap flag is inserted in parallel so that it can interrupt even without the Program Interrupt being enabled. The software Trap-on and Trap-off instructions were bypassed completely.

This particular modification was installed because the original version of FORTRAN II was not compatible with use of the Program Interrupt. We have modified the I/O library subroutines to protect them from interrupts even while using wait loops for all I/O. This hardware modification will therefore be restored for compatibility with routines using the Program Interrupt. Since both the PI and the Trap make use of location ∅, they must be able to inhibit each other until the contents of location ∅ can be saved.

The most important modification to the Central Processor is that the Execute cycle is inhibited if a Trap-Interrupt occurs. Normally an Execute cycle will follow any Defer cycle. If, however, the Trap flag is on, the Execute cycle is inhibited and a Break cycle occurs instead. This allows the interpreting monitor to interrupt an Extended Memory instruction before execution.

Software for Computer Storage

The interpreting monitor for data storage in another computer is illustrated in Figure 3. XTEND7 is a FORTRAN II subroutine written almost exclusively in Basic Assembler language which uses an Interprocessor Buffer to a PDP-9 for storage and access of data. The three lines of instruction preceding the location BEGIN perform initialization. Somewhere in the main program there must be a call to the subroutine. When the subroutine is called it deposits a JMP instruction in location 2 and issues a PBLP command (raise Parameter flag) to the Interprocessor Buffer to synchronize the corresponding interpreting monitor in the PDP-9.

The program XTEND9 in the meantime is waiting idly in the associated PDP-9. When the PBLP command is issued by the PDP-7, an interrupt occurs in the PDP-9 which starts the program at the beginning, restarts the interrupt, and waits idly for the first data word.

When a trap interrupt occurs in the PDP-7, the subroutine XTEND7 is entered at BEGIN from location 2. The contents of the AC are transferred to the PDP-9 (sent by PBLT and received by RDLO) and stored in the variable AC. The next few lines retrace the path of the interrupt to retrieve the offending instruction and combine the instruction code with the indirect address to form a new instruction to be executed in the PDP-9. For example, the instruction LAC I (35000 would be interpreted as LAC 15000 (truncating the address modulo 8k).

The new instruction is transferred to the PDP-9 and executed. If the PC is incremented or the LINK is set during execution, this is noted, and the LINK and PC increment are returned to the PDP-7. The contents of the AC after execution are also returned to the PDP-7 and the interpreting monitor returns to the main program. The PDP-9 remains in a state of waiting idly for the next such instruction.

Philosophy of Bulk Storage

The philosophy of approach for bulk storage is that since access to individual elements of bulk devices is intolerably slow, data must be transferred to and from the device in blocks and stored in the memory in pages. In principle Extended Memory need only be used for large arrays of data, and the data are generally referred to successively in blocks, as, for example, in DO loops of FORTRAN.

The interpreting monitor thus has two parts. The main part interprets an indirect address instruction and modifies the address to refer to the appropriate location within a temporary scratch page in memory. This requires a comparison between the actual address and the range of addresses corresponding to each of the scratch pages. This part of the interpreter is thus similar to that for computer storage and introduces a similar time delay.

The second task of the interpreting monitor is to retrieve blocks of data from the bulk storage device whenever the appropriate data block is not already in core. This requires replacing one of the pages already in core back into the bulk storage before loading the desired block into the same page.

For DECtape a convenient size of page is 256 words, corresponding to the maximum data storage per DECtape block. Where there is room in the program for more than $1k_{10}$ words of scratch pages this may be a reasonable size. But in general, for reasonably complex programs, $1k_{10}$ of memory is an upper limit to the space available for scratch pages. Experience has shown that for most programs it is desirable to have provision for 6-8 pages simultaneously, so that references to elements of several arrays can be made in the same set of calculations.

It is important, of course, that the arrays stored for use in this manner should be referred to by the main program only in blocks, i.e., with iterative references to successive elements within a block. The size of the blocks referred to need not be the same sizes as the pages, but one must avoid referring to such data in a true random manner.

Software for Bulk Storage

For storage and access of data in bulk storage, therefore, the interpreting monitor must be slightly more sophisticated than that for computer sotrage, as shown in Figure 4. XTENDT is a FORTRAN II subroutine written almost wholly in Basic Assembly Language which utilizes a DECtape for Extended Memory. As before the subroutine inserts a JMP instruction into location 2 when first called by the main program. It also rewinds DECtape unit ∅ (or 8).

When a Trap Interrupt occurs, the contents of the AC are stored and the original instruction is retrieved to find the corresponding instruction code and desired address. The address is first compared with a table to see if the contents of that location are already stored on a scratch page in memory. If the most significant bits of the desired address agree with one of the table addresses, the routine jumps to the appropriate page handler. Otherwise a new page containing the new address must be retrieved from the DECtape.

This subroutine uses up to 8 such scratch pages with each page containing 128 locations. The actual number of pages desired is specified by the main program as NPMAX. The pages are stored at the top of COMMON from location 12000_8 down.

The transfer subroutine writes the oldest page in memory back onto DECtape and reads the appropriate block from DECtape back into that same page making it the youngest page. The variable NPAGE indicates the number of the youngest page, i.e., the one most recently retrieved from DECtape. The block number of the DECtape is taken as the appropriate high order bits 3-10 of the address, right justified, so that each block contains 128 locations. The first 100_8 blocks of the tape are not used because they correspond to the basic 8k core memory. The next 300_8 blocks provide the equivalent of an additional 24k of core.

Figure 5 shows how the instruction is executed once the appropriate page is in core. Only the seven least significant bits 11-17 of the address are used and these are added to the starting address of the page to find the correct address. This is combined with the instruction code and deposited in the location XECUTE. The contents of the AC and LINK are restored and the instruction is executed with appropriate incrementing of the PC before returning to the main program.

Conclusions

In the case of the computer to computer transfer, the routine occupies almost no memory and requires 52 cycles or 91 μsec to execute Extended Memory instructions. For the bulk storage routine the memory requires about $1k_{10}$ (depending on the number of pages) and the time is 55-76 cycles or about 100 μsec per instruction if the address is already in core. For most FORTRAN programming, the LINK and PC servicing portions of the interpreting monitor can be omitted. For such a version, the corresponding times are 75 μsec for computer-computer transfer and 90 μsec for bulk storage.

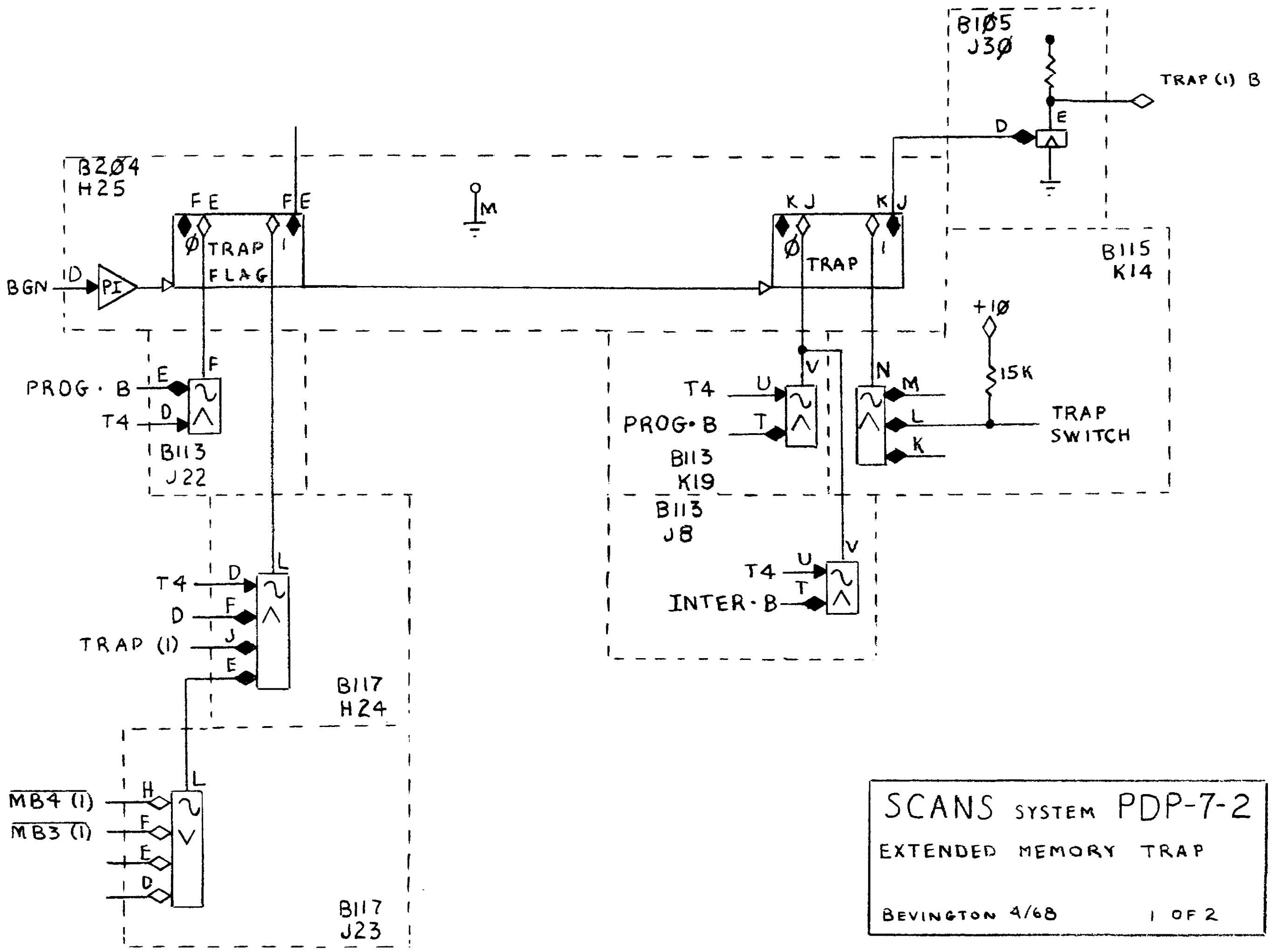
For data arrays in FORTRAN this extra time is not generally much of a disadvantage. In floating point manipulation, for example, the time required for an arithmetic or library subroutine operation is considerably larger. Even for displays the increment time is usually tolerable. The time added to a general computation is a small percentage of the total time.

For DECtape storage, however, the time added per transfer is about 1 sec. Care must be taken, therefore, to optimize the program flow so that references to arrays occur mainly in blocks. For example, iterations in a DO loop which treat many elements of a few arrays should complete all possible manipulations of each element or group of 128 elements before proceeding to the rest of those arrays.

Experience with a typical data reduction program (GRASP) has shown that if the rate of transfer is kept down to an average of less than 1 per 5 sec, the increase of time in calculations required for the transfer is compensated for by the fact that the availability of more arrays simplifies the calculation. In normal GRASP, for example, a polynomial background curve must be computed from the polynomial coefficients at each step of the least-squares fitting procedure. In GRASP with Extended Memory, the background curve is calculated once and stored in an array for subsequent use. Similarly, the display routine normalizes each data point on the fly before display to avoid storing a normalized spectrum. With Extended Memory the normalized spectrum can be stored so that the added time for retrieval is balanced by the loss of time needed for normalization.

This system can, of course, be used equally well for programs written in Assembly language, such as data acquisition programs. In this case the Trap Interrupt and interpreting monitor are not necessary, but they provide an easy way of updating 8k programs to 16k memory with almost no change in the basic program. Data acquisition programs cannot utilize the bulk storage technique without considerable modification because of their random access nature.

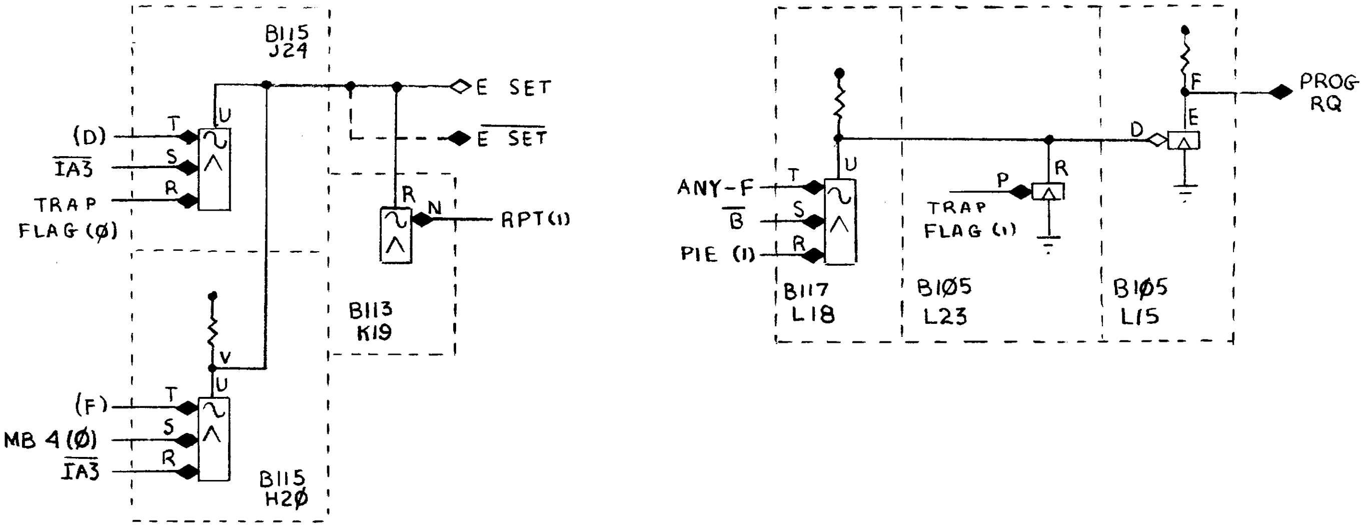
In conclusion, the computer-computer transfer is a very viable technique, where facilities exist. The DECtape storage simulates Extended Memory at some noticeable disadvantage. A compromise utilizing a disk or drum should provide the optimum response.



SCANS SYSTEM PDP-7-2

EXTENDED MEMORY TRAP

BEVINGTON 4/68 1 OF 2



SCANS SYSTEM PDP-7-2

EXTENDED MEMORY TRAP

BEVINGTON 4/68 2 OF 2

```
XTEND TAPE SUBROUTINE  3/68
      SUBROUTINE XTEND TAPE (NPAGE, NPMAX)
C     EXTENDED MEMORY FORTRAN SUBROUTINE USING DECTAPE
C     PHIL BEVINGTON, STANFORD UNIVERSITY
C     NPAGE = PRESENT PAGE NUMBER
C     NPMAX = NUMBER OF PAGES (MAXIMUM OF 6)
SOCTAL      LMQ=652ØØØ        LACQ=641ØØ2
S           LRS=64Ø5ØØ        ALS=64Ø7ØØ       DECIMAL
      DIMENSION MATRIX(8,128)
      COMMON MATRIX
      WRITE 3ØØØ
      NEXTB = 1
SOCTAL
S     LAC (JMP .1            DAC 2                /ENTRANCE FROM TRAP
      RETURN
S
S.1,  DAC #AC                                     /SAVE AC
S.2,  LAM-1       ADD Ø      DAC #TEMP            /FETCH INSTRUCTION
S     LAC I TEMP  AND (NOP   DAC #INSTR
S     LAC I TEMP  AND (17777 DAC TEMP
S     LAC I TEMP  ADD (1     DAC #ADDR
S     AND (776ØØ                                  /CHECK IF PAGE IN MEMORY
S     SAD ADDR1   JMP PAGE1
S     SAD ADDR2   JMP PAGE2
S     SAD ADDR3   JMP PAGE3
S     SAD ADDR4   JMP PAGE4
S     SAD ADDR5   JMP PAGE5
S     SAD ADDR6   JMP PAGE6
S     SAD ADDR7   JMP PAGE7
S     SAD ADDR8   JMP PAGE8
S     DAC #NADDR                                  /PAGE NOT IN MEMORY
S     JMS TRANSFER           JMP .2               /LOAD PAGE INTO MEMORY


SPAGE1,           LAC ADDR   AND (177             /ADDRESS IN PAGE 1
S     ADD (116ØØ             JMP EXECUTE
S
SPAGE2,           LAC ADDR   AND (177             /ADDRESS IN PAGE 2
S     ADD (114ØØ             JMP EXECUTE
S
SPAGE3,           LAC ADDR   AND (177             /ADDRESS IN PAGE 3
S     ADD (112ØØ             JMP EXECUTE
S
SPAGE4,           LAC ADDR   AND (177             /ADDRESS IN PAGE 4
S     ADD (11ØØØ             JMP EXECUTE
S
SPAGE5,           LAC ADDR   AND (177             /ADDRESS IN PAGE 5
S     ADD (1Ø6ØØ             JMP EXECUTE
S
SPAGE6,           LAC ADDR   AND (177             /ADDRESS IN PAGE 6
S     ADD (1Ø4ØØ             JMP EXECUTE
S
SPAGE7,           LAC ADDR   AND (177             /ADDRESS IN PAGE 7
S     ADD (1Ø2ØØ             JMP EXECUTE
S
SPAGE8,           LAC ADDR   AND (177             /ADDRESS IN PAGE 8
S     ADD (1ØØØØ             JMP EXECUTE
S
SEXECUTE,         ADD INSTR                       /EXECUTE INSTRUCTION
S     DAC XECUTE  LAC AC
SXECUTE,   XX     SKP        ISZ Ø
S     JMP I Ø
S
SADDR1,    76ØØ
SADDR2,    76ØØ
SADDR3,    76ØØ
SADDR4,    76ØØ
SADDR5,    76ØØ
SADDR6,    76ØØ
SADDR7,    76ØØ
SADDR8,    76ØØ
SDECIMAL
      END
```

```
XTEND 7  3/1/68
       SUBROUTINE XTEND7
       CONTINUE
SOCTAL
S      PBLP=7Ø2222             PBLT=7Ø2224
S      TDLO=7Ø2225             RDLO=7Ø2253
S
S      LAC (JMP BEGIN           DAC 2                /TRAP ENTRY
S      PBLP                                          /INITIALIZE PDP-9
       RETURN
S
SBEGIN,             PBLT                             /SEND AC
S      LAM-1        ADD Ø       DAC #ADDR            /LOCATE INSTRUCTION
S      LAC I ADDR   AND (NOP    DAC #INSTR           /ORIGINAL INSTRUCTION
S      LAC I ADDR   AND (17777  DAC ADDR             /ORIGINAL ADDRESS
S      LAC I ADDR   AND (17777  ADD INSTR            /NEW INSTRUCTION
S      TDLO         JMP .-1                          /SEND INSTRUCTION
S      RDLO         JMP .-1                          /READ LINK AND PC
S      RCR          SZA         ISZ Ø                /SET LINK AND PC
S      LAC Ø        SPA         STL
S      RDLO         JMP .-1                          /READ AC
S      JMP I Ø                                       /RETURN
       END
```

```
XTEND 9  3/1/68

       PBPF=7Ø22Ø1             PBEP=7Ø2264
       TDLO=7Ø2225             RDLO=7Ø2253

1/     PBPF         SKP         JMP BEGIN            /INITIALIZE LOOP FOR PDP
       CAF          ION         JMP I Ø              /IGNORE PDP-9 INTERRUPTS
22/
BEGIN,      CAF     PBEP        ION                  /ENABLE INTERRUPT

CONTINUE,           CLL         DZM #PC              /CLEAR LINK AND PC+1
       RDLO         JMP .-1     DAC #AC              /READ AC
       RDLO         JMP .-1     DAC .+2              /READ INSTRUCTION
       LAC AC       XX                               /EXECUTE INSTRUCTION
       SKP          ISZ PC      DAC AC               /INCREMENT PC+1
       LAC PC       RAL                              /SEND LINK AND PC+1
       TDLO         JMP .-1
       LAC AC       TDLO        JMP .-1              /SEND AC
       JMP CONTINUE

START BEGIN
```

IMPLEMENTATION OF AN ON-LINE REACTIVE (TYPEWRITER) LANGUAGE*

David Z. Polack
University Computing Company
Dallas, Texas 75207

Abstract

The language processor to be discussed is designed for use via reactive typewriter. It accepts, names, stores and manipulates character strings which may be used as names, data and/or procedure. List processing techniques are utilized in the processor implementation.

The presentation is in the form of a tutorial session, which first places the language processor within the framework of the University Computing Company's FASBAC System. Subsequent discussion will include:

1. a brief description of the language for those unacquainted with it,

2. discussion of memory allocation in terms of the necessary coding, strings, stacks, vectors, communication zones, etc.,

3. the methodology of handling various strings,

4. dynamic "Garbage Collect",

5. special handling of defined primitives,

6. additional primitives not included in previous literature,

7. discussion period.

Reference may be made to: TRAC, A Procedure-Describing Language for the Reactive Typewriter; Calvin N. Mooers; Communications of the ACM, Volume 9/Number 3/March, 1966.

*This paper was not received for publication.

DISC VERSION OF STRIP
A DATA DISPLAY AND ANALYSIS PROGRAM
FOR THE PDP-8, 8/I

John Alderman
Georgia Institute of Technology
Nuclear Research Center
900 Atlantic Drive
Atlanta, Georgia 30318

ABSTRACT

A version of STRIP has been developed to take advantage of the storage capabilities of the DF32 Disc Storage Unit. Techniques of overlay generation and calling, data storage and retrieval, and programming philosophy for open-ended programs to be modified by unskilled users are described.

INTRODUCTION

This paper is primarily concerned with the difficulties of the programmer in using the new Disc System Monitor software, the subject of this morning's report by Mr. Conley. Examples of coding are taken from the presently on-going revision of STRIP*, which will make extensive use of the disc, for both program overlay and data storage.

The new systems programming has the very limiting assumption that all disc software must be compatable with DECTAPE hardware. Thus, even though the disc hardware is randomly addressable, with word transfers of up to 4096 words, the monitor software assumes all files are 128 words long, and that each block is to be addressed and transfered individually.

I have some objections to this assumption:

(1) Most peripherals on the PDP-8 are serial-by-character in nature, and are capable of handling variable length blocks of data. Examples are: Paper tape, IBM compatable magnetic tape, cards, display units, and communications lines. An appropriate name for these might be "record oriented devices". The restriction to fixed block length requires a rather complicated file access system, in order to fill/empty only fixed length blocks from these record oriented devices.

(2) If a record oriented disc storage system were employed it would be possible to make all I/O resemble (IBM compatable) tape records, including the disc (and DECTAPE too, since it would be a special case). One advantage to such a system is that a modular I/O system can be constructed for such a system, which will result in much simpler, highly recursive, programming for data handling between peripherals on the computer. This record oriented I/O system is used on many larger systems (CDC6600, Univac 1108, are examples) and the efficiency of operation would also benefit the PDP-8, since some of the difficulties of beginner programmer are involved with routine I/O programming.

This is, of course, not to say that variable length storage systems do not have their problems. The primary difficulty for a small system like the PDP-8, is that the Directory Name table would have to be larger, and that some sort of "automatic" disc reshuffling system would have to be employed to restructure the disc when a file is deleted. My feeling is that the efficiency of storage utilization afforded by the variable length system, along with the considerable increase in access speed (the 33 ms access time for the disc must be allotted for each fixed length block, but only one such wait is necessary for a variable length record) available by making use of the disc hardware, would more than offset the difficulties of a longer DN table, and the disc reshuffling at delete time.

Regardless of the above considerations, the presently implemented system is available, and it is possible to use it to do most of the program overlay storage and data manipulation that a user desires.

DATA STORAGE AND RETRIEVAL

Since all storage on the disc is in SAM blocks of 128_{10} words, I have written a subroutine to access the disc (see Figure 1). This subroutine has arguments specifying function, starting core location, and starting SAM block number, and is called IMPORT/EXPORT (IE). The routine is self contained, calls the system I/O routine at 7642, and requires only 26_{10} locations within the user's program. With it, any file on the disc can be accessed, if the starting SAM block number is known. The file will be placed in core with each page stored contiguously (note that it is not necessary to make the core starting address a page boundary if data is being transferred!). With a change of the FUNCTION argument, it is possible to write on a previously saved file. The same routine can be used to handle the actual transfer of program overlays.

As an example of the use of IMPORT/EXPORT, I have written a disc accessing version of STRIP*. This first version is very straight-forward, in that it uses the last few pages of the disc (ours is a two-disc system) as the data storage area for the Floating Point data buffer mentioned in the writeup. Since only one page at a time is accessed from the disc, the version of IE used does not have the test of IELNK. Notice that the scratch blocks are addressed by an absolute SAM block number. Since the scratch file is not defined (by the "SAVE" operation), the DN table has no entry for it, which means that that area of the disc is "blank" as far as subsequent "SAVE" operations is concerned. Of course it is incumbent upon the user to insure that nothing of interest is destroyed when using that scratch area. The feature of using a non-defined file as scratch is very useful, since otherwise the disc tends to get cluttered up with unused files.

In the case of the user's program needing to access a defined file, however, it must obtain the starting SAM block number. Figure 2 is the listing of a program to obtain the starting SAM block number. The main routine interrogates the operator for the NAME of the file, and then call the subroutine DNSRCH (based upon coding supplied by Roger Pyle), which searches the DN table for the name, and then the SAM directory table for the starting block number, which is returned to the calling routine in the accumulator. The main program then types this out in octal. This program is both a useful operational program (especially when debugging disc routines) and an example of the use of the subroutine DNSRCH from within the user's program. With this subroutine (which can be on overlay), the user can find any file on the disc, and by using IE for actual transfers, can use that file from within his program.

DISC STRIP - Features

DISC STRIP will differ from the previous version, primarily in the internal working of the program. The calculations and usage of the program will be very similar.

With the advent of the disc storage unit (DF32), it has become possible to take advantage of the larger storage area available for program overlays and data, as well as complete programs. DISC STRIP will make extensive use of the overlay feature. The present rule is that all overlay programs must not occupy more than two contiguous pages of core (if they are to access data also) and they will be loaded into the overlay area when the keyboard is struck. Since many programs will not occupy a whole page, and it is desireable to make maximum use of the available disc, there will be a call to a keyboard interpreter at the beginning of each overlay section. This interpreter will look up the character typed, in its field of keyboard functions available IN THE CURRENT OVERLAY ONLY. If the function is not in the current overlay function table, the next overlay is called, and the process repeats until a legal function is found, or all the overlays have been called. In the later case, an error exit to the monitor is taken.

This scheme of chaining overlays with a function table stored in each, lends itself to expansion by adding new overlays at the end of the chain, and adjusting the table of available overlays (each of which is referenced by its starting SAM block number). The core resident package will also have provisions for accessing data files from the disc, and (as before) the user need not concern himself about the location of the data, but simply addresses it indirectly via a moving pointer in page zero. The limitation of the number of points displayed is arbitrarily set at 1024 by the 10 bit hardware of the type 34 display unit.

A new feature of DISC STRIP will be the full utilization of the TEKTRONIX type 601 (or 611) storage oscilloscope. Since each point on the display can be displayed permanently as it is computed, there is no requirement for a display buffer, which materially improves the data storage problem. Also there is no requirement for a titles buffer, and the titles will actually be another keycalled function. The storage scopes may be erased under program control, and there will be another keycalled function for doing so.

As a result of the number of keycalled functions available, it becomes of interest to be able to quickly program a series of calls to these functions. A new feature of DISC STRIP will be an elementary interpretive assembler, which will assemble calls to keyboard functions (including numerical arguments), and then allow the user to specify a key to call the new function ensemble. This will be a recursive process, and the main limitations will be the basic functions available, and the amount of disc storage room available for these routines, and the data.

Availability:

I have a working version of STRIP, using the disc for data storage only. This version does work, but only as a patch for the older version. The only advantage to using the patch, is that data fields of up to about 650 data points may be displayed. All of the commands for the patch version are identical with the previous writeup, and the user can only tell the difference, by the inordinate length of time required to do his computations.

DISC STRIP has been partially coded. Most of the computational subroutines are copied from the previous version, so the only additional coding required is the overlay and data access control structures.

The release date on DISC STRIP will probably be in early June. It will be submitted to the DECUS library at that time, but there is naturally some delay in the distribution of a new program through DECUS, since it must be reviewed. I will be glad to provide "under-the-table" copies to prospective users, until it becomes a burden on my time.

Conclusions

The disc CAN be used for both data storage, and program overlay storage. The documentation available from DEC doesn't show the programmer how to do it, but the examples of this paper should be sufficient to get most programmers started.

A new, somewhat fancy, DISC STRIP will be out shortly, for users who might want it.

```
/Page ØØØ1

/IMPORT/EXPORT , A WHOLESALE DISC/CORE SWAPPING ROU-
                /TINE
/
/
/CALL BY        JMS IE  /ACC MUST BE CLEAR
/               3 OR 5  /3=READ, 5=WRITE ON DISC
/               CORE    /CORE STARTING ADDRESS
/               BLOCK   /GIVEN BY DNSRCH&SMSRCH
/               ERROR  RETURN    /
/               NORMAL RETURN    /ACC CLEAR
/AVERAGE ACCESS TIME =25 MS

IE,        Ø     /ENTRY
           TAD I IE      /GET FUNCTION
           DCA IEFUNC
           ISZ IE
           TAD I IE      /GET CORE SA
           DCA IECORE
           ISZ IE
           TAD I IE      /GET BLOCK NUMBER (OCTAL)
           ISZ IE
IELOOP,    DCA IEBLK
           JMS I SYSIO
IEFUNC,    3
IEBLK,     Ø
IECORE,    Ø
IELNK,     Ø
           JMP I IE      /ERROR EXIT
           TAD IECORE
           TAD IE2ØØ
           DCA IECORE    /NEXT PAGE
           TAD IELNK     /FURNISHED BY SYSIO
           SZA           /END OF FILE?
           JMP IELOOP    /NO
           ISZ IE        /YES
           JMP I IE      /EXIT NORMAL
IE2ØØ,     2ØØ
SYSIO,     7642


           /TEST ROUTINE FOR DNSRCH
           /TYPE IN YOU FILE NAME, AND IT
           /REPLIES WITH OCTAL
           /STARTING SAM BLOCK NUMBER

0200  6032  START,      6032
           /INITIALIZE THE FLAGS
0201  6046              6046
0202  4230  RO,         JMS CRLF
           /NORMALIZE TELEPRINTER
0203  4244              JMS GET
0204  7106              RTL CLL
0205  7006              RTL
0206  7006              RTL
0207  3306              DCA WORD1
           /SAVE 1ST LEFT HALF
0210  4244              JMS GET
0211  1306              TAD WORD1
0212  3306              DCA WORD1
           /SAVE 1ST PACKED WORD
0213  4244              JMS GET
0214  7106              RTL CLL
0215  7106              RTL CLL
0216  7006              RTL
0217  3307              DCA WORD2
           /SAVE 2ND LEFT HALF
0220  4244              JMS GET
0221  1307              TAD WORD2
0222  3307              DCA WORD2
           /SAVE 2ND PACKED WORD
0223  4244              JMS GET
           /LOOKING FOR RETURN
0224  1227  WHAT,       TAD QUEST
```

```
                /TOO MANY CHARACTERS
0225  4236  JMS PRINT
0226  5200            JMP START
0227  0277  QUEST,277

0230  0000  CRLF,     0
0231  1276            TAD CR1
0232  4236            JMS PRINT
0233  1277            TAD LF
0234  4236            JMS PRINT
0235  5630            JMP I CRLF
0236  0000  PRINT,    0
0237  6041            6041
0240  5237            JMP.-1
0241  6046            6046
0242  7200            CIA
0243  5636            JMP I PRINT
0244  0000  GET,      0
                /GETS CHARACTERS
0245  6031            6031
0246  5245            JMP .-1
                /NOT READY YET
0247  6036            6036
                /OK, GET IT
0250  6046            6046
                /ECHO IT
0251  3270            DCA CHAR
                /SAVE
0252  1270            TAD CHAR
                /ENTER TESTING ROUTINE
0253  1271            TAD MCR
                /IS IT RETURN
0254  7450            SNA
0255  5303            JMP CR
                /YES
0256  1272            TAD MRO
                /IS IT RUBOUT
0257  7450            SNA
0260  5202            JMP RO
                /YES
0261  1273            TAD MCTRLC
                /IT IT  CTRL C
0262  7650            SNA CIA
0263  5702            JMP I MONRET
                /YES, EXIT TO MONITOR
0264  1270            TAD CHAR
                /GET THE CHARACTER AGAIN
0265  1274            TAD P40
0266  0275            AND C77
                /STRIPPED ASCII+40 OUTPUTED
0267  5644            JMP I GET
0270  0000  CHAR,     0
0271  7563  MCR,      -215
0272  7616  MRO,      215-377
0273  0174  MCTRLC,   377-203
0274  0040  P40,      40
0275  0077  C77,      77
0276  0215  CR1,      215
0277  0212  LF,       212
0300  0400  SRCH,     DNSRCH
0301  0240  SP,240
0302  7600  MONRET,   7600
0303  1277  CR,TAD LF
0304  4236  JMS PRINT
0305  4700            JMS I SRCH
0306  0000  WORD1,    0
0307  0000  WORD2,    0
0310  5224            JMP WHAT
                /ERROR EXIT!

                /NOW WE HAVE THE SAM BLOCK NUMBER
                /IN THE AC
0311  3333            DCA PTEM
                /SAVE IT
```

```
0312  1334                TAD M4
0313  3332                DCA DCN
          /INITIALIZE DIGIT COUNTER TO 4
0314  1333                TAD PTEM
0315  7004                RAL
0316  7004  PNU2,         RAL
0317  7006                RTL
0320  3333                DCA PTEM
0321  1333                TAD PTEM
0322  0335                AND PCON     /7
0323  1336                TAD PCON+1  /260
0324  4236                JMS PRINT
0325  1333                TAD PTEM
0326  2332                ISZ DCN
0327  5316                JMP PNU2
0330  7200                CIA
0331  5202                JMP R0
0332  0000  DCN,          0
0333  0000  PTEM,         0
0334  7774  M4,           -4
0335  0007  PCON,         7
0336  0260                260
           *400
          /DIRECTORY SEARCH SUBROUTINE
          /CALLING SEQUENCE
          /
          /JMS I (DNSRCH
          /CHARACTERS 1,2 /SEE NOTE BELOW
          /CHARACTERS 3,4
          /ERROR RETURN FOR NON-FOUND NAME
          /OR I/O ERROR
          /NORMAL RETURN (AC=START BLOCK)
          /
          /
          /NOTE: EACH CHARACTER IS STRIPPED
          /ASCII+40(8)
           READ=3
0400  0000  DNSRCH        0
0401  1600                TAD I DNSRCH
0402  3276                DCA DNT1
0403  3274                DCA DNSAMC
0404  2200                ISZ DNSRCH
0405  1600                TAD I DNSRCH
0406  3277                DCA DNT2
0407  2200                ISZ DNSRCH
0410  1273                TAD DNSTRT
          /GET ADDR OF 1ST DN BLOCK
0411  3221                DCA DNBLOK
0412  5217                JMP DNIO
0413  1223  DNREAD,       TAD DNLINK
          /GET NEXT BLOCK NR FROM LINK
0414  7450  CDD,          SNA
0415  5262                JMP BADRET
0416  3221                DCA DNBLOK
0417  4774  DNIO,         JMS I SYSIO
0420  0003  DNFUNC,       READ
0421  0000  DNBLOK,       0
0422  0600  DNCOR,        DNBUF
0423  0000  DNLINK,       0
0424  5262                JMP BADRET  /ERROR
0425  1275                TAD DNKT
0426  3301                DCA DNKTR
0427  1220                TAD DNFUNC
0430  1222                TAD DNCOR
          /SET POINTER TO 1ST DN ENTRY
0431  3300                DCA DNT3
0432  1700  DNTEST,       TAD I DNT3
          /COMPARE ENTRY WITH PROGRAM
0433  2274                ISZ DNSAMC
0434  2300                ISZ DNT3
0435  7041                CIA
0436  1276                TAD DNT1
0437  7640                SZA CLA
          /IS THIS THE DESIRED NAME?
0440        5263          JMP DNNOTF  /NO
```

```
0441  1700                TAD I DNT3
          /YES, COMPARE THE NEXT 2 CHARS
0442  7041                CIA
0443  1277                TAD DNT2
0444  7640                SZA CLA
          /ARE THESE CHARS SAME?
0445  5263                JMP DNNOTF  /NO
0446  1220                TAD DNFUNC
          /YES, INCREMENT POINTER TO
0447  1300                TAD DNT3
          /LOOK AT FILE EXTENSION
0450  3300                DCA DNT3
0451  7332                CLA STL RTR
          /IS THIS A BINARY PROGRAM?
0452  0700                AND I DNT3
0453  1302                TAD DN6000
0454  7640                SZA CLA
0455  5264                JMP DNNOTF+1
          /NO CONTINUE SEARCH
0456  1672                TAD I DNFSBN
          /YES, GET 1ST SAM BLOCK NR
0457  3313                DCA CDSAM
          /SETUP CALL TO SAM DIRECTORY
0460  1274                TAD DNSAMC
0461  5304                JMP SMSRCH
0462  5600  BADRET,       JMP I DNSRCH
0463  1220  DNNOTF,       TAD DNFUNC
          /INCREMENT POINTER BY FOUR
0464  7001                IAC
0465  1300                TAD DNT3
0466  3300                DCA DNT3
0467  2301                ISZ DNKTR
0470  5232                JMP DNTEST
0471  5213                JMP DNREAD
0472  0602  DNFSBN,       DNBUF+2
0473  0177  DNSTRT,       177
0474  0000  DNSAMC,       0
0475  7747  DNKT,         -31
0476  0000  DNT1,         0
0477  0000  DNT2,         0
0500  0000  DNT3,         0
0501  7777  DNKTR,        -1
0502  6000  DN6000,       6000
            DN77=C77
            DNBUF=DNSRCH+200
0503  0077  CD77,         77
0504  7041  SMSRCH,       CIA
0505  3371                DCA CDSMVA
          /MINUS SAM NR TO CDSMVA
0506  3364                DCA CDSMPT
          /SET BLOCK COUNTER TO ZERO
0507  1371  CDSMRD,       TAD CDSMVA
0510  3366                DCA CDSMT1
0511  4774                JMS I SYSIO
          /READ THE SAM BLOCK INTO BUFFER
0512  0003  CDRD4,        READ
0513  0000  CDSAM,        0
0514  0600  CDCORE,       DNBUF
0515  0000  CDSMLK,       0
0516  5262                JMP BADRET  /ERROR
0517  1363                TAD CDM2
0520  3372                DCA CDSMSW
0521  1303                TAD CD77
          /SET MASK FOR BLOCKS 0-177
0522  3367                DCA CDSMMT
0523  1365  CDSML2,       TAD CDM200
          /SET COUNTER TO -200
0524  3370                DCA CDSMCT
0525  1314                TAD CDCORE
          /SET BLOCK POINTER
0526  3373                DCA CDCORX
0527  1773  CDSMLP,       TAD I CDCORX
0530  0367                AND CDSMMT
0531  1366                TAD CDSMT1
0532  7650                SNA CLA
          /IS THIS THE DESIRED BLOCK?
```

```
0533  5357                  JMP CDSMFD
           /YES, EXIT SUBROUTINE
0534  2373                  ISZ CDCORX
           /NO, CONTINUE SEARCH
0535  2364                  ISZ CDSMPT
0536  2370                  ISZ CDSMCT
0537  5327                  JMP CDSMLP
0540  1362                  TAD CD7700
           /END OF 1ST PASS, CHANGE MASK
0541  3367                  DCA CDSMMT
0542  1366                  TAD CDSMTk
0543  7106                  CLL RTL
           /SET TEST WORD FOR BLOCKS 200
0544  7006                  RTL
0545  7006                  RTL
0546  0367                  AND CDSMMT
0547  3366                  DCA CDSMT1
0550  2372                  ISZ CDSMSW
0551  5323                  JMP CDSML2
0552  1315                  TAD CDSMLK
0553  7450                  SNA
           /IS THIS THE LAST SAM BLOCK?
0554  5262                  JMP BADRET
           /YES, SAM NR NOT FOUND
0555  3313                  DCA CDSAM
           /NO, CONTINUE SEARCH
0556  5307                  JMP CDSMRD
0557  2200  CDSMFD,         ISZ DNSRCH
           /SAM NR WAS FOUND, LOAD THE
0560  1364                  TAD CDSMPT
0561  5600                  JMP I DNSRCH
           /RETURN TO CALLER
0562  7700  CD7700,         7700
0563  7776  CDM2,           -2
0564  0000  CDSMPT,         0
0565  7600  CDM200,         -200
0566  0000  CDSMT1,         0
0567  0000  CDSMMT,         0
0570  0000  CDSMCT,         0
0571  0000  CDSMVA,         0
0572  0000  CDSMSW,         0
0573  0000  CDCORX,         0

0574  7642  SYSIO,          7642

BADRET  0462
CDCORE  0514
CDCORX  0573
CDD     0414
CDM2    0563
CDM200  0565
CDRD4   0512
CDSAM   0513
CDSMCT  0570
CDSMFD  0557
CDSMLK  0515
CDSMLP  0527
CDSML2  0523
CDSMMT  0567
CDSMPT  0564
CDSMRD  0507
CDSMSW  0572
CDSMT1  0566
CDSMVA  0571
CD77    0503
CD7700  0562
CHAR    0270
CR      0303
CRLF    0230
CR1     0276
C77     0275
DCN     0332
DNBLOK  0421
DNBUF   0600
```

```
DNCOR    0422
DNFSBN   0472
DNFUNC   0420
DNIO     0417
DNKT     0475
DNKTR    0501
DNLINK   0423
DNNOTF   0463
DNREAD   0413
DNSAMC   0474
DNSRCH   0400
DNSTRT   0473
DNTEST   0432
DNT1     0476
DNT2     0477
DNT3     0500
DN6000   0502
DN77     0275
GET      0244
LF       0277
MCR      0271
MCTRLC   0273
MONRET   0302
MRO      0272
M4       0334
PCON     0335
PNU2     0316
PRINT    0236
PTEM     0333
P40      0274
QUEST    0227
READ     0003
RO       0202
SMSRCH   0504
SP       0301
SRCH     0300
START    0200
SYSIO    0574
WHAT     0224
WORD1    0306
WORD2    0307


                /PAGE ØØØØ
                /DISC STRIP
                /PATCHES TO SETUP FOR DISC DATA BUFFER
                SETUP=JMS I DIPNTR
                INDEX=JMS I DAPNTR
                *17Ø
Ø17Ø    4Ø42 DAPNTR,DA
Ø171    4ØØØ DIPNTR,DI
                *642
Ø642    4571 SETUP
                *646
Ø646    457Ø INDEX
                *653
Ø653    4571 SETUP
                *655
Ø655    457Ø INDEX
                *667
Ø667    4571 SETUP
                *677
Ø677    457Ø INDEX
                *735
Ø735    4571 SETUP
                *753
Ø753    457Ø INDEX
                *762
Ø762    4571 SETUP
                *767
Ø767    457Ø INDEX
                *1624
1624    4571 SETUP
                *1631
```

```
1631  457Ø  INDEX
            *164Ø
164Ø  4571  SETUP
            *1676
1676  457Ø  INDEX
            *4343
4343  4571  SETUP
            *4352
4352  457Ø  INDEX
            *4413
4413  4571  SETUP
            *4425
4425  457Ø  INDEX
            *42Ø
Ø42Ø  7ØØØ  NOP
Ø421  7ØØØ  NOP
Ø422  7ØØØ  NOP
Ø423  7ØØØ  NOP
Ø424  7ØØØ  NOP
            *434
Ø434  7ØØØ  NOP
Ø435  7ØØØ  NOP
Ø436  7ØØØ  NOP
            *445




            /PAGE ØØØ1

            /DATA ACCESS INITIALIZATION

            I1=1Ø5
            QUEST=173
            L=114
            READ=3
            WRITE=5
            *4ØØØ
            DAM43=DIM42
            DABUFR=DI-2ØØ
            FILE=JMS IE
4ØØØ  ØØØØ  DI,Ø
4ØØ1  73ØØ  CLA CLL
4ØØ2  3242  DCA DA/CLEAR BLOCKCOUNTER (DA)
4ØØ3  1114  TAD L/GET LOW LIMIT
4ØØ4  124Ø  TAD DIM42/SUBTRACT BLOCKLENGTH
4ØØ5  2242  ISZ DA/STEP BLOCK COUNTER
4ØØ6  75ØØ  SMA/NEGATIVE?
4ØØ7  52Ø4  JMP .-3/NO
4Ø1Ø  1241  TAD DIP42
4Ø11  31Ø5  DCA I1/I/ IS TEMP STORAGE NOW
4Ø12  11Ø5  TAD I1
4Ø13  1275  TAD DAM43
4Ø14  3274  DCA DANUM/INITIALIZE COUNTER
4Ø15  11Ø5  TAD I1/MULTIPLY BY 3
4Ø16  71Ø4  CLL RAL
4Ø17  11Ø5  TAD I1
4Ø2Ø  1321  TAD M3
4Ø21  1317  TAD IE34ØØ
4Ø22  31Ø5  DCA I1/I/NOW POINTS TO FIRST DATA POINT
4Ø23  734Ø  CLA CLL CMA/AC=-1
4Ø24  1242  TAD DA/GET BLOCK COUNTER
4Ø25  1237  TAD DIBLK/GET 1ST FILE NUMBER
4Ø26  3312  DCA IEBLK
4Ø27  724Ø  CLA CMA/AC=-1
4Ø3Ø  1277  TAD DIMFIL/GET -NUMBER OF FILES
4Ø31  1242  TAD DA /ADD BLOCK COUNTER
4Ø32  3276  DCA DAEND
4Ø33  43ØØ  FILE
4Ø34  ØØØ3  READ
4Ø35  4511  JMS I 111
4Ø36  56ØØ  JMP I DI
4Ø37  Ø6ØØ  DIBLK,6ØØ
4Ø4Ø  7725  DIM42,-52
```

```
4Ø41  ØØ53  DIP42,52
            /SUBROUTINE TO INCREMENT & TEST POINTERS
            /IT WILL CALL NEXT PAGE OF DATA AUTOMATI
4Ø42  ØØØØ  DA,Ø/ACCESS DISC SUBROUTINE   /CALLY
4Ø43  21Ø5  ISZ I1
4Ø44  21Ø5  ISZ I1
4Ø45  21Ø5  ISZ I1/INDEX I1 BY 3
4Ø46  2274  ISZ DANUM/TEST COUNTER
4Ø47  5264  JMP DATEST
4Ø5Ø  43ØØ  FILE
4Ø51  ØØØ5  WRITE

            /PAGE ØØØ2

4Ø52  2312  ISZ IEBLK
4Ø53  2276  ISZ DAEND/TOO MANY FILES?
4Ø54  741Ø  SKP
4Ø55  5172  JMP QUEST-1/YES, ERROR EXIT
4Ø56  43ØØ  FILE
4Ø57  ØØØ3  READ
4Ø6Ø  1317  TAD IE34ØØ
4Ø61  31Ø5  DCA I1
4Ø62  1275  TAD DAM43
4Ø63  3274  DCA DANUM
4Ø64  21Ø6  DATEST,ISZ 1Ø6
4Ø65  21Ø7  ISZ 1Ø7
4Ø66  2673  ISZ I CNTRPT
4Ø67  56ØØ  JMP I DI
4Ø7Ø  43ØØ  FILE
4Ø71  ØØØ5  WRITE
4Ø72  5642  JMP I DA
4Ø73  Ø446  CNTRPT,446
4Ø74  ØØØØ  DANUM,Ø
4Ø75  7ØØØ  NOP
4Ø76  ØØØØ  DAEND,Ø/-FILE # FOR TESTING
4Ø77  776Ø  DIMFIL,776Ø/-FILE # STORED ***TEST***
41ØØ  ØØØØ  IE,    Ø        /ENTRY
41Ø1  73ØØ  CLA CLL
41Ø2  17ØØ              TAD I IE      /GET FUNCTION
41Ø3  3311              DCA IEFUNC
41Ø4  23ØØ              ISZ IE
41Ø5  1317  TAD IE34ØØ
41Ø6  3313              DCA IECORE
41Ø7  3314  DCA IELNK
411Ø  472Ø              JMS I SYSIO
4111  ØØØ3  IEFUNC, 3
4112  ØØØØ  IEBLK,  Ø
4113  ØØØØ  IECORE, Ø
4114  ØØØØ  IELNK,  Ø
4115  5172  JMP QUEST-1/ERROR EXIT
4116  57ØØ              JMP I IE       /EXIT NORMAL
4117  36ØØ  IE34ØØ,DABUFR
412Ø  7642  SYSIO,7642
4121  7775  M3,-3
```

```
CNTRPT   4Ø73
DA       4Ø42
DABUFR   36ØØ
DAEND    4Ø76
DAM43    4Ø75
DANUM    4Ø74
DAPNTR   Ø17Ø                     IEFUNC   4111
DATEST   4Ø64                     IELNK    4114
DI       4ØØØ                     IE34ØØ   4117
DIBLK    4Ø37                     INDEX    457Ø
DIMFIL   4Ø77                     I1       Ø1Ø5
DIM42    4Ø4Ø                     L        Ø114
DIPNTR   Ø171                     M3       4121
DIP42    4Ø41                     QUEST    Ø173
FILE     43ØØ                     READ     ØØØ3
IE       41ØØ                     SETUP    4571
IEBLK    4112                     SYSIO    412Ø
IECORE   4113                     WRITE    ØØØ5
```

PDP-8 OSCILLOSCOPE DISPLAY OF MATHEMATICAL FUNCTIONS USING FORTRAN

August E. Sapega, Professor of Engineering
and
Stephen G. Wellcome, Student
Department of Engineering
Trinity College

ABSTRACT

A general-purpose program for oscilloscope display of mathematical functions will be described. Since the main program is written in FORTRAN the user need only insert the FORTRAN statement of his function in a standard location. At object time he specifies the range of the independent variable. Following a scaling computation, the scaled function is computed and a table of values generated. These are displayed on an oscilloscope by means of a binary program which is loaded at FORTRAN object time. Interactive features allow the user to re-specify the range of the independent variable to more closely examine the various ranges of the function under study.

The system described uses a PDP-8 with 4K core, and a type 34D oscilloscope display unit.

INTRODUCTION

Oscilloscope displays of single-valued mathematical functions of the form y = f(x) are readily accomplished using a PDP-8 with 4k memory, and a type 34D display. The system described in this paper uses both FORTRAN and PAL programming languages. A standard Tektronix type 561 or 564 oscilloscope, having an 8 x 10 cm. display area is used. The displayed function consists of over 200 individual x,y coordinate values, and some 50 scale marks for a calibrated display.

Interactive programming features allow the user to specify a re-calculation of function values to display different ranges of interest in either the independent or dependent variable by typing appropriate commands during the display. Scaling of the display is automatically carried out during computation so the user need not anticipate or estimate maximum or minimum values of the function. By use of the interactive feature the user can re-specify function ranges until he obtains an appropriate display for his needs. Functions with variable parameters may be programmed in such a way that these parameters may also be re-specified to investigate the effects of such variations.

GENERAL PROGRAM ORGANIZATION

This programming system combines those features of FORTRAN and PAL best suited to the specific requirements of each part of the program. The main program is written in FORTRAN. This allows the user to specify the function of interest in the form of a simple mathematical FORTRAN statement one or two lines long. To use the programming system for his own specific problem the user need change only this statement so that it represents the function he wishes displayed. All calculations are carried out in the FORTRAN portion of the system. Use is made of all FORTRAN features, such as computation in floating-point mode, accessibility to the library of functions, and use of the input/output routines of FORTRAN.

For the display of the function each function value to be shown must be specified in terms of its x,y coordinates. These values are computed, scaled, converted to integer form, translated so that all values are appropriate positive values, and stored in arrays. Upon completion of these calculations, program control transfers to the binary portion of this system, whose principal function is to access the data stored in arrays. Individual x,y coordinate points are transferred from the array to the x and y registers of the display control. The rate at which the display is processed by the binary program is rapid enough so that a stable, flicker-free, display is present on a conventional oscilloscope.

FORTRAN PROGRAM DETAILS

When the user initially enters the program hė is asked to type in the range he desires for the independent variable. Trial values of the function are computed to estimate maximum and minimum values of the dependent variable. The range of both variables, and the scales of both the x and y axes, in units per scale division, are then typed out so the user has a permanent record of this. Detailed calculation of array values then proceeds, and when completed, the function display is presented on the oscilloscope.

Having inspected the display the user can re-specify a new range for either variable by typing CTRL P on the teletype. He is then required to specify whether he wishes to change the range on x or y. If he specifies a new range on x the scaling of y proceeds as before. However he may re-specify the range on y, as for example to get a convenient scale factor, and thus he re-specifies the range of y for the previously established range of x. No attempt is made when scaling y for a specified x to have the y scale factor adjusted to standard values. Thus the user will usually need to re-specify y after initial examihation of the function to get a good scale factor for the y axis.

In order that meaningful results be obtained from the display, scale markings are computed in the FORTRAN program using scaling data as derived from the program, and these marks are displayed along with the function itself.

BINARY PROGRAM DETAILS

The binary program, assembled using PAL, is loaded at object time by the FORTRAN object system loader. The FORTRAN object system will properly load this binary program, but will give a check sum error, since the check sum is calculated differently in FORTRAN than in PAL. The essential point is that the FORTRAN operating system loader will recognize the location counter setting feature of PAL (e.g. *7400), and load the binary program into the specified locations.

When loading this system for use the binary program is loaded before the FORTRAN object program. Execution of the FORTRAN program can then begin immediatetly upon loading of the FORTRAN object program.

When first loaded the binary program occupies part of the FORTRAN working area, specifically the last page, locations 7400-7546. At the beginning of the FORTRAN program a short binary routine loaded into location 7367 is transferred to. This routine relocates the main binary program to occupy locations 7600-7764. Thus all of FORTRAN working area is available and used for data storage, and all FORTRAN features remain intact.

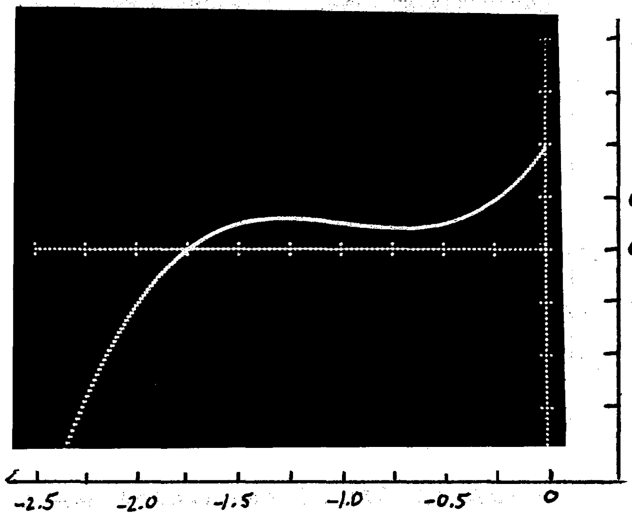
Operation of this dual system of FORTRAN and binary programs is possible because of the "simple-minded" way in which the FORTRAN compiler assigns variable name locations. It starts from location 7577 and works down in storage, allocating locations in the order in which variable names appear in the FORTRAN PROGRAM. By using a standard DIMENSION statement at the beginning of the FORTRAN program the programmer can be certain of the exact location of the data associated with each variable in the program, including the initial address of data arrays. This information is obtained from the Symbolprint program following FORTRAN compilation.

The binary program uses the addresses of the FORTRAN arrays through indirect address link-ups. Once FORTRAN has computed and stored its data in arrays, the binary program is entered. The x and y coordinates of each point are transferred to the x and y registers of the 34D display control and displayed on the oscilloscope. The address of the next pair of points is obtained by use of the ISZ instruction to increment the current addresses. A separate counter is used to note the end of the data array. Upon reaching the end of the array the status of the teletype is checked. If no message is waiting, the data array addresses are re-initialized, and the program loops on the data again. If an appropriate message is in the teletype buffer control is transferred back to the FORTRAN program where the user may re-specify variable ranges as described earlier.

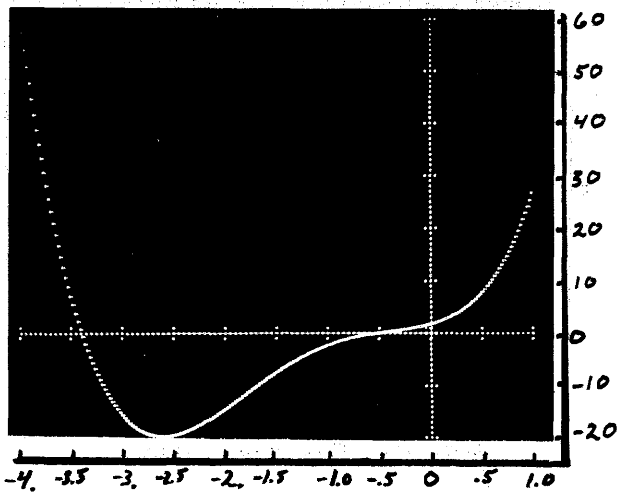
CONCLUSION

By judicious programming using both FORTRAN and PAL displays of mathematical functions are readily accomplished. There is adequate data storage to obtain displays with good detail. User-oriented features allow for readily programming functions of interest, and for specifying ranges of interest for the display. The display is scaled, and accurate values easily read off the display.

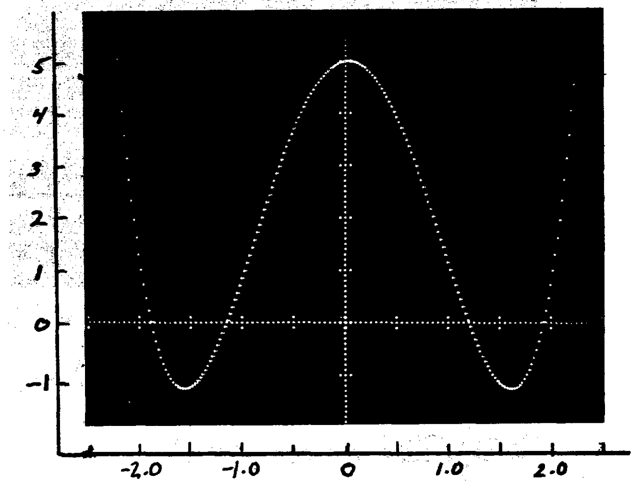
$y = x^3 + 3x^2 + 2.75x + 1.$



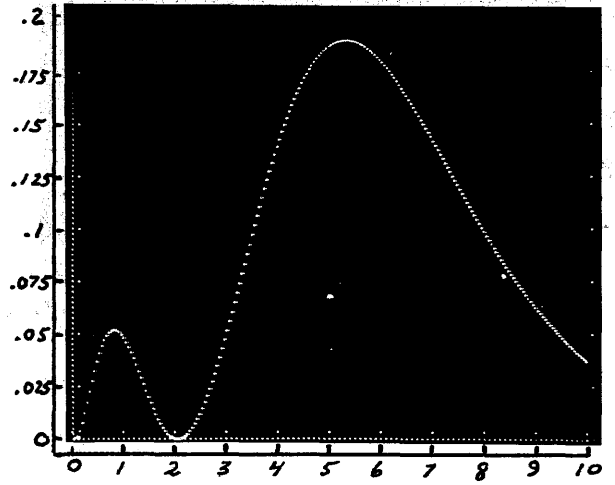
$y = 2x^4 + 9x^3 + 9x^2 + 6x + 2$



$y = x^4 - 5x^2 + 5$

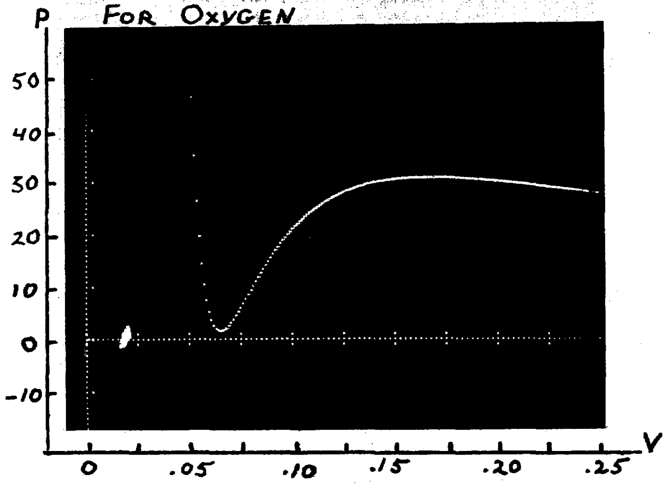


Hydrogen 2s Wave Function

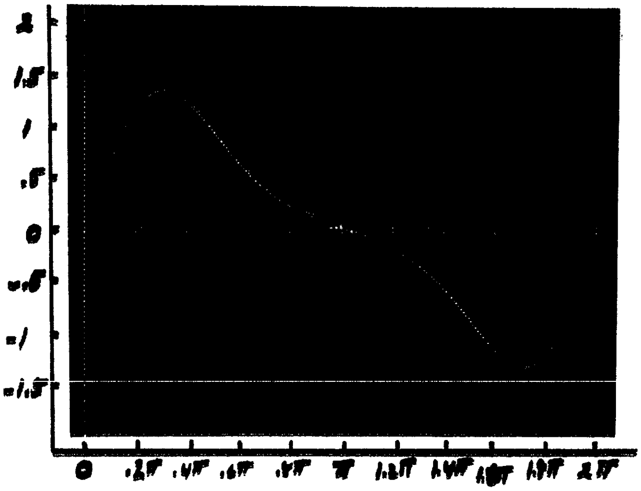


Van der Waals' Equation

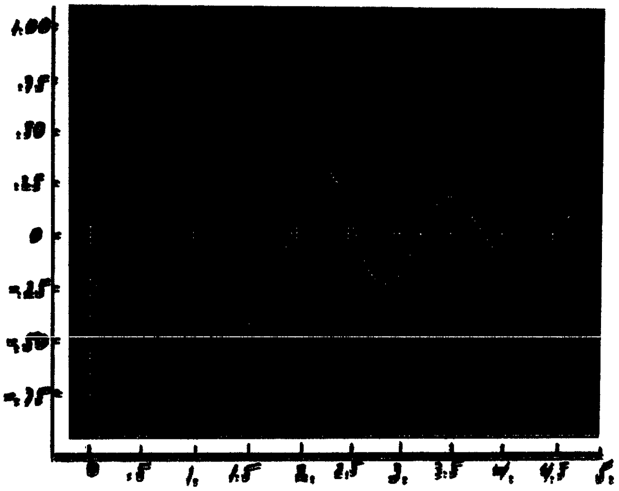
For Oxygen



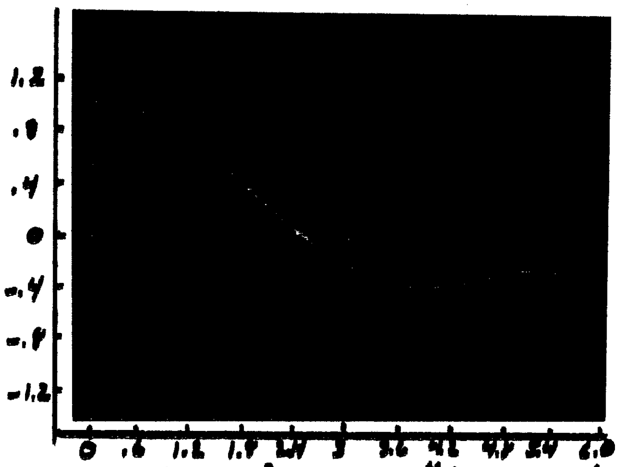
SUM OF TWO SINE WAVES



$y = e^{-.5x} \sin(5x+3)$



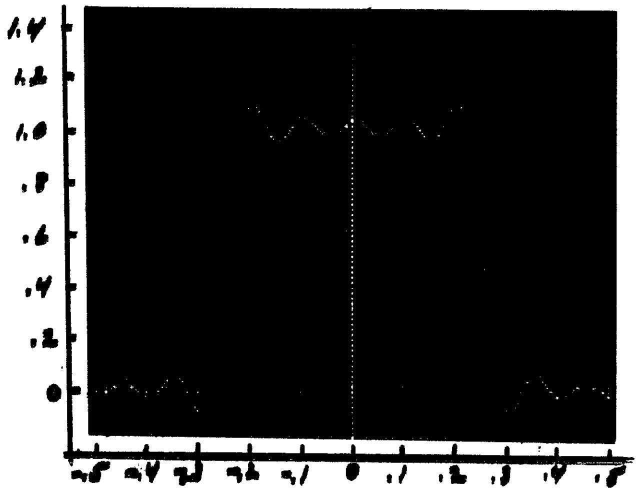
BESSEL FUNCTION FOR J_0



$J_0(x) = 1 - x^2/2^2 1!^2 + x^4/2^4 2!^2 - x^6/2^6 3!^2$

$+ x^8/2^8 4!^2 - x^{10}/2^{10} 5!^2$

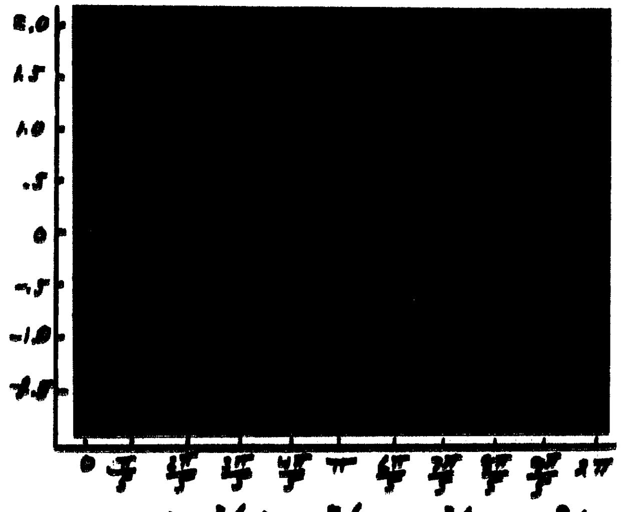
FOURIER SERIES FOR SQUARE WAVE



$y = .5 + 2/\pi \cos(2\pi x) - \frac{\cos(6\pi x)}{3.} + \frac{\cos(10\pi x)}{5}$

$- \frac{\cos(14\pi x)}{7} + \frac{\cos(18\pi x)}{9}$

SERIES EXPANSION FOR SIN(X)



$y = x - x^3/3! + x^5/5! - x^7/7! + x^9/9!$

DPL - A DISPLAY PROGRAMMING LANGUAGE

Jeffrey H. Kulick
Moore School of Electrical Engineering
University of Pennsylvania
Philadelphia, Pennsylvania 19104

ABSTRACT

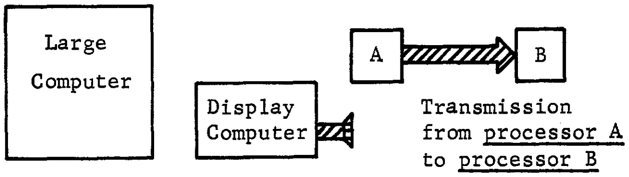
DPL allows the definition of simple data structures such as points and lines and the definition of arbitrarily complex structures called Displaygroups. A class of set operators (<u>FOR</u> A ∈ <u>DO</u>...) allows the user to selectively traverse a data structure. As part of the definitional language, a computational facility is available which allows the definition of structures algorithmically.

DPL operates in two modes. The first, an interactive mode, allows the user to define, display, and modify structures from a Teletype console. The second mode, known as stored program, allows the user to define a sequence of DPL commands and then execute them as a program. Decision and recursive call statements are available when operating in the stored program mode.

BACKGROUND

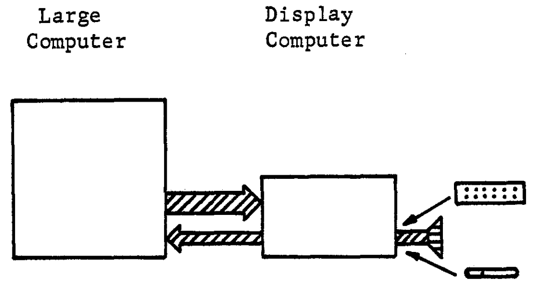
This research on a Display Programming Language (DPL), is the outgrowth of two areas of interest of the author. The first stems from his work in developing a generalized data structure for graphics*. The basic structure developed assigns many attributes to each structure, only one of which is a display attribute. DPL is an extension of the set of primitives developed for the specification of the display attribute.

The second area of interest was studying the effective organization of 2 processors in a graphics problem-solving environment. One of the first problems encountered was the "division of labor" between the two types of processors available, a large scale computer, and a small display computer. We represent pictorially these processors as follows:



MACHINE ORGANIZATION

One possible organization of these processors would be to have a high speed link between the two machines. The large computer would process all requests for each service from the user (interrupts, etc.). Each time a change was to be made, a completely new display file would be sent over to the display computer. The display computer would send the large computer requests for processing, in the form of cues and nothing else.



There are two defects in this type of organization. First, to ensure reasonable response time to the user of the graphics system, the channel connecting the two computers must be very fast. The usually available transmission lines of 2000-2400 bits per second would be much too slow for this type of operation. If one used a high capacity line, then the costs would be much greater. In addition, the number of possible users would also be much lower since each would need his own private line, as opposed to the 2000 bit/second lines which are dial-up, and generally available.

Another defect in this design class, is that it requires a "relatively" dedicated computer. All service requests from the user require servicing by the large computer, and an environment would have to be established that allows fast access. For instance, no swapping of a graphics task would be possible in a multiple task system.

We therefore come to the conclusion that the processing task must be shared between the small and large computers, and that the small computer must be able

*<u>A Generative Grammar and Data Structure for Computer Display of Chemical Graphs, August</u> 1968.

to perform some of the processing functions that the user might want. To do this, the display computer must have some form of data base in it, representing the displayed structure.

DATA BASES

There are two possible organizations for data bases in this multi-processor environment. The first is to send to the small computer a chunk of the data base that exists in the large computer. In particular the chunk that represents the structure to be displayed would be sent to the display computer. When the display computer makes changes in this data base, they must be sent to the large computer, to enable it to make the appropriate changes in its data base. What we have done is to reduce somewhat the need for channel communication and reduce greatly the demands placed on the large computer for processing service requests.

Another possible organization is to have completely different data bases in the display computer and in the large computer. To describe its data base to the display computer, the large computer encodes the data base up in an intermediate language, which is sent to the display computer. Likewise, the display computer sends records of changes to its data base by the user, in an intermediate language. It was this approach that was chosen, and DPL is a language to encode the large computers data base for transmission to the display computer.

LANGUAGE CRITERIA

A set of criteria was established that the display language was to meet. Following is a list of these criteria:

Console Operation

The language was to be so constructed that the user could write programs in the language, at the console of the display computer, and each statement was to be meaningful. In other words, the interpreter for the display language would have to have an incremental mode of operation.

Stored Program Control

The language was to be such that sequences of instructions in the language could be executed under some form of loop control. In addition to having immediately interpreted instructions (console operation) other instructions could be used in a stored program computer mode.

Basis on DEC-338

There was to be a certain amount of hardware dependence, but this was to be kept to a minimum. The basic assumption was that the display screen had an (x,y) addressable screen.

338 Hardware Features Available

Certain 338 hardware features were to be made available to the user in a generalized form if possible. These were to include at least, the ability to use the pushbuttons, the blink feature and the light pen of the 338 system.

Basic Primitives in Definitinal Mode

There was to be a small set of basic primitive, with which the user was to be able to define basic elements. These were to include at least text, points, lines.

Constructable Higher Level Structures

Given any set of the basic primitives, one must be allowed to group these together under a common name. From then on, the name could be used instead of the list of primitives. At the user's option, the defined structure should be relocatable and hence, usable as a subroutine anywhere on the screen. This feature was to be available at any level of definition.

Computation

There was to be some amount of computational ability available, at least at the level of definition (i.e. 12 bit arithmetic). More powerful arithmetic was to be available from the higher level processor.

Logical Operations

There was to be a number of logical operations, transfer of control, etc., to allow non-sequential execution of stored programs. The decisions were to be based on both user generated and computed conditions.

Light Pen Sensitive Structures

It should be possible to make structures light pen sensitive. The user should be able to specify where (what processor) to transfer to under the condition of a light pen "hit".

Structure Manipulation

For any structure that was defined, the user should have the ability to search down the structure. He should be able to look for a particular element and select it if he desires. He should be able to designate them as selected or distinguished elements and operate on them.

Generated Symbols

There should be a process available to allow the user to generate names of objects and use the generated names. The user should have access to the names as easily as names generated explicitly by program statements.

Input Output

There should be an input/output facility for the language to allow communication in the following directions:

LARGE COMPUTER	→	DISPLAY COMPUTER
LARGE COMPUTER	→	DISPLAY USER
DISPLAY COMPUTER	→	LARGE COMPUTER
DISPLAY COMPUTER	→	DISPLAY USER
DISPLAY USER	→	DISPLAY COMPUTER
DISPLAY USER	→	LARGE COMPUTER

SPECIFICATION OF THE LANGUAGE

Modes of Operation

In the current version of DPL, there are two modes of operation, reactive typewriter and stored program.

Modes of Operation

↙ ↘

Reactive	Stored Program
A: Point (0000, 0000)	•Instruction •/And, Instruction

In the reactive typewriter mode, as each instruction is read in from the teletypewriter, it is decoded and executed. If the instruction is preceded by

(a) .

(b) ./NAME,

then the instruction is stored away for later execution. If the instruction is preceded by (b) above, then NAME is the name of that instruction. Transfer of control instructions (defined later) transfer to named statements.

STATEMENT TYPES

DEFINITIONAL
    primitive
    recursive

LOGICAL
    transfer of control
    logical test

GENERATED SYMBOLS

ARITHMETIC
    Algebraic
    Logical

COMMAND

STRUCTURE SEARCH

INPUT/OUTPUT

There are seven basic statement types in DPL, and each is broken down into one or more sub classes. Each will be explained below. Most of the statements have the following form:

NAME:OPERATION<MODIFIER>(OPERANDs)

However, only certain statements have all of the above options.

Varbles and Names

In the definitions below, we will use the following terms:

TERM	INTERPRETATION
NAME POINTNAME DISPLAYGROUPNAME	Any string, 1-5 characters
VARBLE	A varble is an expression which evaluates to a octal value of x, such that, $0 \leq x \leq 7777$ There are a number of different VARBLES, and they will be discussed below in VARBLES PART II and PART III. The first type of VARBLE is the constant. NNNN where $0 \leq N \leq 7$

Definitional Statements - Primitive

POINTNAME:POINT(VARBLE1,VARBLE2)

This defines a point on the screen with name POINTNAME at (x,y) coordinates (VARBLE1,VARBLE2)

LINENAME:LINE(POINTNAME1,POINTNAME2)

This defines a line between the points POINTNAME1 and POINTNAME2 and the name of the line is LINENAME.

NAME:DISPLAYGROUP(LINENAME,DISPLAYGROUPNAME...)

This defines the set of objects in the operand field named NAME. Any number of items may appear in the operand field.

TEXTNAME:TEXT(any ASCII text)

This defines the text string in the operand field assigning the name TEXTNAME.

CONSTANTNAME:CONSTANT(VARBLE)

This defines the structure with name CONSTANTNAME, to have the value of VARBLE and class constant which can be used in arithmetic operations described below.

Definitional Statements - Recursive

NAMEA:DISPLAYGROUP(NAMEA,....)

This redefines displaygroup NAMEA so that the new elements specified in (NAMEA,....) will be added to the elements in the displaygroup. The name of the displaygroup can appear anywhere in the operand list and as many times as desired. If the displaygroup was originally a relocatable subroutine, then multiple occurrences of that name may result in relocated pictures for each occurrence.

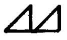
LAB:LINE <RELATIVE> (A,B)   FIGURE DEFINED

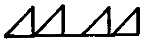
LBC:LINE <RELATIVE> (B,C)

LCA:LINE <RELATIVE> (C,A)

LAC:LINE <RELATIVE> (A,C)

TRI:DISPLAYGROUP(LAB,LAC,LBC,LCA,LAC)

TRI:DISPLAYGROUP(TRI,TRI) 

TRI:DISPLAYGROUP(TRI,LAC,TRI) 

Modifiers to Definitional Statements

A modifier to a definitional statement appears in the form:

NAME:OPERATION <MODIFIER> (OPERANDS)

The modifiers are:

<BLINK>

This will result in the defined structure blinking at a rate of 2 times a second, when displayed.

<UNINTENSIFY>

This will result in the defined structure appearing at intensity 0 when it is displayed.

<RELATIVE>

The syntactic effect of this statement is to remove all beam positioning information from the operands in defining the new structure. The semantic results are that the new structure consists of displacement vectors, rather than lines between fixed points.

For example:

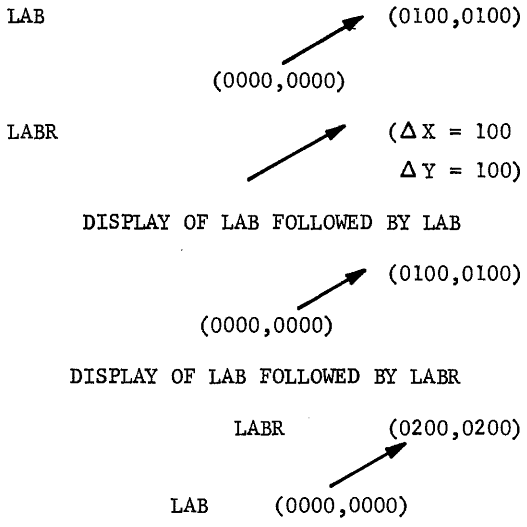
A:POINT(0000,0000)

B:POINT(0100,0100)

LAB:LINE(A,B)

LABR:LINE< RELATIVE> (A,B)

FIGURE CONSTRUCTED



DISPLAY OF LAB FOLLOWED BY LAB

DISPLAY OF LAB FOLLOWED BY LABR

ARITHMETIC STATEMENTS

NAME:PLUS(VARBLE1,VARBLE2)

This assigns to name NAME the class constant, and assigns to it the value VARBLE1+VARBLE2

NAME:MINUS(A,B)

This assigns to name NAME the class constant and assigns k to it the value VARBLE1-VARBLE2

NAME:TIMES(A,B)

This assigns to name NAME the class constant, and the value VARBLE1*VARBLE2

NAME:DIVIDE(A,B)

This assigns to name NAME the class constant and assigns to it the value VARBLE1/VARBLE2

NAME:AND(VARBLE1,VARBLE2)

This assigns to name NAME the class constant and assigns to it the value VARBLE1 $\wedge$ VARBLE2

NAME:OR(VARBLE1,VARBLE2)

This assigns to name NAME the class constant and the value VARBLE1 $\vee$ VARBLE 2

Command Statements

:EXECUTE(NAME,...)

This instruction clears the screen of all previous contents, generates and executes the display file associated with the items in the operand field.

:APPEND(NAME,...)

This operates like the EXECUTE command above, but does not clear the screen before displaying the specified operands. This is used for adding to the contents of the screen.

In both of these instructions, one may apply the modifiers as indicated in the section on modifiers applicable to definitional statements. For example:

:EXECUTE< BLINK> (NAME,...)

Logical Statements

(a) TRANSFER OF CONTROL STATEMENTS

:GOTO(PROGRAMNAME)

This transfer control unconditionally to program statement PROGRAMNAME. It does not save the current program location.

:CALL(PROGRAMNAME)

This transfers control to program statement PROGRAMNAME. It pushes down the current program location.

:RETURN

This pops up the address of the last call encountered and returns control to it, plus 1 statement. Upon executing a top level RETURN, control returns to the teletype.

(b) LOGICAL TESTS

:IF(VARBLE1.EQ.VARBLE2)S1

If the value of VARBLE1 equals the value of VARBLE2, then statement S1 is executed, else S1 is skipped, S1 can be any legal DPL statement.

:IF(VARBLE1.LT.VARBLE2)

If the value of VARBLE1 is less than that of VARBLE2 then statement S1 is executed, else S1 is skipped.

:IF(VARBLE1.GT.VARBLE2)S1

If the value of VARBLE1 is greater than that of VARBLE2 then statement S1 is executed, else S1 is skipped.

VARBLES Part II

In addition to the originally defined varble, octal constants, we have additional varbles that correspond to the various system elements. These are given below:

VARBLE	VALUE	INTERPRETATION
<XCPT:POINTNAME>	The X coordinate of point POINTNAME	
<YCPT:POINTNAME>	The Y coordinate of point POINTNAME	
<PB:n>	The condition of push-button n =1 if on =0 if off	
<CLCPT:NAME >	The class of structure NAME The following classes are already defined: 1 = POINT 2 = LINE 3 = DISPLAYGROUP 4 = STORED PROGRAM 5 = GENERATED SYMBOL 6 = GENERATED SYMBOL 7 = CONSTANT 10 = TEXT 11 = PUSHDOWN 400 = BLINKED STRUCTURE 1000 = RELATIVE STRUCTURE 2000 = UNINTENSIFIED STRUCTURE	

Example

```
:IF( <PB:n >.EQ.0001):GOTO(PROG)  (executed if
                                   PBn is on)
```

NAME:PLUS(<XCPT:A >, <YCPT:A >) (sets NAME to be $A_x + A_y$)

```
:IF( <CLCPT:A >.EQ.0002)S1  (executed if A is a
                             line)
```

Micro Definitional Instructions

To correspond to the above variables, we have instructions that allow one to selectively access the X and Y component of a point and the class component of a structure:

```
NAME:XSET(VARBLE)
```

The X component of point NAME is set to the value of VARBLE

```
NAME:YSET(VARBLE)
```

The Y component of point NAME is set to the value of VARBLE.

```
NAME:CSET(VARBLE)
```

The CLASS component of structure NAME is set to the value of VARBLE.

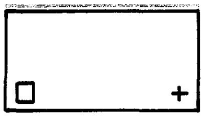
Tracking Facility

There is available to the user one basic primitive for tracking such that the user will not have to process the light pen activity himself. This process is called by a statement of the following form:

```
NAME:POINT( <TRACK> )
```

The procedures that occur are as follows:

(a) The following is appended to the screen:



(b) The user can track the square with the light pen to the desired position.

(c) Finally, when the user has finished tracking, he touches the target (+) with the pen to end the tracking sequence.

(d) The coordinates of the upper left corner of the box are assigned as the coordinates of point NAME.

Alternatively

```
NAME:POINT( <CTRAC >)
```

will operate as <TRACK >does except that the tracking square is not repositioned to the lower left hand corner before commencing tracking. This feature is useful for generating continuous figures.

Generated Symbols

A facility for generating names of symbols is available to the user. It is called by the following statement:

```
NAME:SYMBOLGEN
```

This assigns to NAME, the name of the generated symbol. All future uses of NAME are equivalent to using the generated symbol. One can have any number of names representing generated symbols, and the same symbols can be used over again to represent a new generated symbol.

Example

```
./P1,A:SYMBOLGEN
.A:POINT( <TRACK >)
.B:SYMBOLGEN
.B:POINT( <CTRAC> )
.C:SYMBOLGEN
.C:LINE(A,B)
.D:DISPLAYGROUP(D,C)
```

```
        .:EXECUTE(D)
        .:GOTO(P1)
```

This program will read in two points, from the light pen and assign them to generated symbols A and B. Then it will define a line in terms of A and B called C, also a generated symbol. Finally it appends to the current definition of D, the new line C, and displays it. If we had not used generated symbol features, we could only define one line, display it, and re-define it.

Structure Search

With the ease of generating symbols that is available with the symbolgen feature, a handle is needed on all of the generated symbols. In addition to this, we also want a generalized process that will enable to search a structure selectively, for a particular element. For this purpose, the following instructions have been defined:

```
        :PUSHDOWN(NAME,...)
```

This defines structure NAME to be of class pushdown. That is, when a definition is put into it, the previous definition is saved and the new one is added to the top. Redefinition of other structures in general, results in the loss of the previous definition. The above statement is used in conjunction with the following statement:

```
        :LET(NAME1)BE(NAME2)
```

This pushes down onto NAME1 the definition of NAME2. If NAME1 is empty, then this definition is the only definition of NAME2. If NAME1 already stood for something then the previous definition is pushed down and the new one added to the top.

```
        :FOR(NAME1)∈ (NAME2)DO(NAME3)
```

This allows NAME1 to index over the elements of structure NAME2. The element that is used is changed (indexed further up in the structure) each time NAME3 is encountered until NAME2 is emptied.

For example, we want to select out of structure Z, all the lines that it consists of and group them under displaygroup LINES. If there is a displaygroup in Z we want to further index down this displaygroup so that LINES will have in it all the lines that compose the structure.

```
        :PUSHDOWN (B)
        :LET(B)BE(Z)
        ./P1,:FOR(C) ∈(B)DO(P2)
        .:IF( <CLCPT:C >.EQ.002):GOTO(LINE)
        .:LET(B)BE(C)
        .:GOTO(P2)
        ./LINE,LINE:DISPLAYGROUP(LINE,C)
        ./P2...
```

Text Variables

It is desirable to display as a result of computation some variables such as number, text strings, etc. To do this, the following variables may appear in text strings only:

VARBLE	VALUE
<OCTAL: NAME>	The ASCII equivalent of the variable NAME, converted into an octal number. The value of X when the statement is defined is used.
< TEXT:NAME >	The ASCII string denoted by the structure NAME is inserted into the text string.
< C/R >	Returns the beam to the beginning of this line but displaced one line lower than the original line.

Input Output Varbles Part III

To allow the user to enter information from the teletype the following VARBLES are also available:

VARBLE	VALUE
<READ:TEXT>	A text string is read in from the teletypewriter assigned as the value of this VARBLE up to the input C/R.
<READ:OCTAL> <READ:DECIMAL>	Reads in a 4 digit number and converts it into a 12 bit binary number according to the specified conversion.

Light Pen Sensitive Structures

It is desirable to allow the user to denote certain structures as being light pen sensitive and if the light pen ever touches them when displayed, to transfer to a special routine. This is done by the appropriate trap time program at the time the light pen hit occurs.

```
        :TRAP(< LPHIT:NAME1> )(NAME2)
```

This will trap to processor NAME2 if the light pen ever touches the structure NAME1. NAME1 can be of class line, text, or displaygroup.

```
        :INHIBIT (< LPHIT:NAME1 > )
```

This will inhibit interrupts on this structure so that it will become light pen insensitive.

```
        :ENABLE
```

This will enable all traps that are currently defined to be enabled again. When a light pen hit occurs, all traps will be disabled until this instruction is executed, to prevent continuous trapping on one condition.

```
        <LPEN >
```

This is a VARBLE that will represent the name of the object that caused the light pen hit if a trap occurs because of light pen hits.

NOTE: During the processing of these traps the old address of the program is saved and can be returned to by executing the RETURN operation.

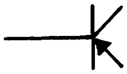
EVALUATION AND CONCLUSION

DPL was originally designed to serve as a description language for a higher level data structure. It was intended that the display attribute of the structures would be specified in a DPL like language. DPL was extended in a number of ways over and above this primitive definitional level. This was to give the user an effective way of using the display computer without necessarily having the support of a large scale computer system.

Toward these aims DPL was a very successful effort. Because of the facilities of the language even relatively sophisticated problems are easily handled in a small number of statements. One can easily, and in a few minutes, construct both useful programs and interesting graphical structures.

On the other hand, experience with DPL has shown its inadequacies for forming the basis of a really power graphic system. Because of the lack of ability to establish relationships between data elements the structure constructed, while graphically complex, have simple data bases. Hence the DPL data base does not really reflect the structure involved.

For example:



It is easy to define structures that look like the above. However, the DPL data base would allow connectivity at only two points, the beginning and the end of the structure, and hence, this structure would not truly be represented by the data base. It appears that a much more complicated data base is really needed for powerful graphics. We would for instance like to say that this is a PNP transistor, of type... The base is connected to terminal Al or circuit K3, etc.

DPL is a useful tool to those who are able to accept the limited data base upon which it builds its structures. It will allow its users from the teletype to define, display and redefine graphical structures. Towards this end DPL is a useful tool to the user of a display computer.

Johnson, Timothy E. "Sketchpad III -Experimental Graphical Communication with a Digital Computer," ESL Labs. Department of Electrical Engineering, Massachusetts Institute of Technology, May 1963, AD# 406-855.

Lang, C. A., Polansey, R. B., Ross, D. T. "Some Experiments with an Algorithmic Graphical Language," ESL Lab., Department of Electrical Engineering, Massachusetts Institute of Technology, August, 1965.

Lindsay, R. K. Pratt, T. W., Shaves, K. M. "An Experimental Syntax Directed Data Structure Language, Rand Corp., April 1965, AD# 614782.

Mann, W. C. "Language Facilities for Man, Machine, and Relational Data," Office of Naval Research, Information Systems Branch RPT #1062, April 1967, AD# 651-973.

Ross, D. T. "Notes for Lectures on Graphical Communication", ESL Lab. Department of Electrical Engineering, Massachusetts Institute of Technology, June 1965.

Sutherland, I. E. "Sketchpad - A Man Machine Graphical Communication System", Massachusetts Institute of Technology, June 1963 AD#404-549.

Sutherland, W. R. "On-Line Graphical Specification of Computer Procedures", Lincoln Lab., Massachusetts Institute of Technology, Group 23 - Report 405, May 1966, AD# 639-734.

Wessler, Barry. "TRAC D", Masters Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, May 1967.

STRUCTURE DEFINITION

Primitive

```
NAME:POINT(0000,0000)
NAME:LINE(POINTNAME,POINTNAME)
NAME:DISPLAYGROUP(NAME ...)
NAME:TEXT(...)
NAME:CONSTANT(0000)
```

Recursive

```
NAMEA:DISPLAYGROUP(NAMEA...)
```

ALGEBRAIC

Arithmetic

```
A:PLUS(B,C)
A:MINUS(A,C)
A:TIMES(B,C)
A:DIVIDE(B,C)
```

Logical

```
A:AND(B,C)
A:OR(B,C)
```

Figure 1

LOGICAL

Transfer of Control

```
:GOTO(NAME)
:CALL(NAME)
:RETURN
```

Logical Test

```
:IF(A.LT.B)SI
:IF(A.GT.B)SI
:IF(A.EQ.B)SI
```

NAME GENERATION

```
A:SYMBOLGEN
```

COMMAND

```
:EXECUTE(NAME1,NAME2...)
:APPEND(NAME,...)
:DELETE
```

Figure 2

MICRO DEFINITIONAL

```
A:XSET(B)
A:YSET(B)
A:CSET(B)
```

Defined Classes

1-	POINT
2-	LINE
3-	DISPLAYGROUP
4-	PROGRAM
5-	GENERATED SYMBOL
6-	GENERATED SYMBOL
7-	CONSTANT
10-	TEXT
11-	PUSHDOWN
400-	BLINKED STRUCTURE
1000-	RELATIVE STRUCTURE
2000-	UNINTENSIFIED STRUCTURE

Figure 3

STRUCTURE SEARCH

```
:PUSHDOWN(NAME)
:LET(NAME)BE(NAME)
:FOR(NAME)=(NAME)DO (NAME)
```

LIGHT PEN

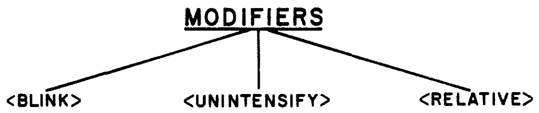
```
:TRAP(<LPHIT:NAME1>)(NAME2)

:INHIBIT(<LPHIT:NAME1>)

:ENABLE
```

<LPEN>
Contains Name of structure hit by light pen during last interrupt

Figure 4



MODIFIERS

<BLINK>   <UNINTENSIFY>   <RELATIVE>

BLINK--BLINKS PICTURE
UNINTENSIFY--DISPLAYS PICTURE AT 0 INTENSITY
RELATIVE--ALLOWS DEFINITION OF SUBROUTINES

```
NAME: DISPLAYGROUP<BLINK>(...)
NAME: DISPLAYGROUP<RELATIVE>(...)
NAME: DISPLAYGROUP<UNINTENSIFY>(...)
```

Figure 5

SPECIAL VARIABLES

SYMBOL	VALUE
<PB:N>	=0 if push button N off =1 if push button N on
<XCPT:A> <YCPT:A> <CLCPT>	X,Y,and CLASS components of a structure. X and Y are defined for points only
<OCTAL:NAME> <DECIMAL:NAME> <TEXT:NAME>	Used in TEXT instructions to encode computed variables into text strings
<CR>	In text string, positions beam to beginning of new line
<READ:TEXT> <READ:OCTAL> <READ:DECIMAL>	Reads in from console Teletype, the appropriate value

Figure 6

THE FASBAC SYSTEM - TIME DIVISION MULTIPLEXING AND THE PDP-8

G. E. Friend
Consultant, Remote Processing
and
Paul J. Bell
Consultant, System Design
University Computing Company
Dallas, Texas

ABSTRACT

Hardware modifications and software techniques for the efficient utilization of the 680 Data Communications System as a low-speed-line multiplexer for the University Computing Company's FASBAC Remote Access System are described.

FASBAC AND UNIVERSITY COMPUTING COMPANY

The FASBAC system being developed within the Technical Services Division of University Computing Company is a computing system which gives the customers of U.C.C. simultaneous, remote access to the UNIVAC 1108 equipment at our Data Centers via keyboard devices.

Technical Services Division has two principal businesses. They are:

1. a computing service bureau business using UNIVAC 1108 equipment, and

2. a contract programming operation which produces proprietary software running on the 1108 and available to our customers. The contract programming operation also writes programs for other types of equipment, but the principal emphasis is on the development of proprietary application programs for the UNIVAC 1108.

The revenue generated by a large-scale computing system is more or less a function of the number of people who access the equipment in a given period of time. This is the economic foundation of the current generation of multi-programming, multi-processor equipment. With this equipment, more than one user may have access to the machine at one time. The FASBAC project, then, seeks to increase the load factor on our existing equipment, and to develop new applications for that equipment.

FASBAC PROJECT GOALS

Our initial goals are:

1. to provide our customers with a remote file inquiry, update, and batch job initiation capability, and

2. to provide our programming staff with a remote access program development and debug facility.

Our experience with previous time-sharing systems indicates that this latter area is one of rapid pay-off, since it helps reduce programming time, and programming time is about the most expensive single commodity in the computer business.

THE FASBAC SYSTEM IN BRIEF (Fig. 1)

The FASBAC system configuration comprises the following equipment:

1. Teleprinter devices in customers' offices, or somewhere about their sites. These teleprinter devices may be connected either over a hard wired circuit, or over the public switched telephone network, to the communications multiplexers.

These communication multiplexers are D.E.C. 680 Data Communications Systems, about which more later. The multiplexers are connected to the FASBAC system executive processor, the system controller, via either an Interprocessor Buffer or a High-Speed Data Line Interface, modems, and a voice grade private wire telephone circuit

The executive processor is in turn connected to the UNIVAC FASTRAND II mass storage device and to the UNIVAC 1108 computer, via a proprietary multiplexor.

Since the functions of the executive processor have been covered in another paper (THE FASBAC REMOTE ACCESS SYSTEM by Dan W. Scott), we will concentrate on the detailed functions implemented in the system front end, the 680 Data Communications System.

THE 680 DATA COMMUNICATIONS SYSTEM (DCS) (Fig. 2)

The 680 DCS performs the following functions:

1. It receives data from and transmits data to the low-speed (less than 300 bauds) teleprinter devices connected to

the PDP-8 computer via a dataset interface, Serial Line Multiplexor, and a Serial Line Interface.

2. It performs message buffering and local editing.

3. It transmits messages to and receives messages from the PDP-9 via either an Interprocessor Buffer or a high-speed Data Communications Interface.

4. It handles the remote (dial-up) terminal line discipline, answering and hanging up the phone checking for disconnected lines, etc.

5. It handles program and hardware fault detection and recovery.

The low-speed data line interface program may accommodate mixed speed lines with varied character lengths and start-stop configurations. Message buffering is handled on a "line" basis, where a line may consist of a fixed number of characters, or a variable number of characters, not exceeding some arbitrary number, terminated by a carriage return or other meta character.

Local editing of the incoming text consists of handling character delete and line delete functions and trapping continuous BREAK characters.

Remote terminal line discipline includes the detection of incoming calls, the assignment of data channels to varied incoming customers, the monitoring of the data line condition, and the orderly termination of calls and initialization of channels for reuse by the next customer.

Fault detection and recovery procedures include examination of the data lines for faults and initiation of program reload in the case of catastrophic program failure.

DATA COMMUNICATIONS SYSTEM DESIGN CONSIDERATIONS

Some of the considerations which affect the design of the DCS program include:

1. A high instantaneous memory load, which may reach 97% of the available memory time for short periods.

2. A high interrupt rate due to the design of the serial line multiplex unit, with its attendant short interrupt cycle.

3. A large number of incoming lines, requiring the service of several lines at each interrupt.

4. A "hidden" asynchronous memory load applied by the Interprocessor Buffer, which operates on the 3-cycle data break.

A simplified algorithm for calculating the average memory load is shown as Figure 3. This algorithm calculates the memory load which must be sustained in order not to lose incoming data. It does not show the load necessary to do the local editing, line buffering, and other housekeeping tasks.

680 DCS LIMITATIONS

There are a number of fairly serious limitations inherent in the use of the 680 DCS as a line multiplexor. Some of these limitations are:

1. A limited addressing capability, including the lack of an index register.

2. A small instruction repertoire.

These limitations are inherent in computers with short word lengths, and the PDP-8 design is perhaps the best possible compromise in the use of the available bits between addressing capability and instruction repertoire.

SOFTWARE APPROACH (Fig. 4)

The FASBAC 680 DCS program attempts to meet the functional requirements for a multiline interface by a combination of hardware and software techniques. The 680 DCS program is organized on a multi-priority basis with six modules, each of which is entered in order of its priority, and is not exited until it has nothing further to do. These six modules are programmed in two groups.

The first group operates with the program interrupt system disabled in what we will call "Real Time", and is entered upon the occurrence of any hardware interrupt. There are two modules in this group, the Line Service module, and the Other Interrupt Service module.

The Line Service module checks the flags set by the various clocks in the Serial Line Multiplex unit. If any of these flags are set, the appropriate lines are sampled for input data, complete input characters are collected and put in an input character queue, and waiting output data is transmitted. The exit from this module is back to the beginning of the module, and this process is repeated until no further clock flags are found.

The Other Interrupt Service module is then entered to test for the occurrence of interrupts from such devices as the Interprocessor Buffer, console teletype keyboard/printer, etc. Any such interrupts are serviced by resetting the flag, saving any volatile data and making an entry in a table of subroutine calls (which is normally filled with NOP's). Exit from the Other Interrupt Service module is made to the location at which the original interrupt occurred.

The second group of modules operates with the interrupt system enabled in what we will call "Spare Time".

There are four such modules. The highest priority is given to the message buffering of incoming characters queued by the real time line service module, and to the furnishing of output characters to this module. Second priority goes to the execution of tasks set up by the real time Other Interrupt Service module. Any such tasks, when they are entered, erase themselves from the table of subroutine calls and, when they are completed, exit to the highest priority spare time module.

After all spare time character input-output service and spare time interrupt service is completed, the third spare time module is entered. This module is a table of lower priority tasks set up by other, higher priority tasks. When completed, these tasks also exit to the spare time character I/O service module. If no spare time tasks are waiting, a diagnostic and fault locating procedure is initiated.

This modular approach increases program running time and memory space requirements, because of the many necessary queues and pointers. However, it does accomplish the objective of avoiding instantaneous overloads in the memory load of the processor (load leveling).

THE HARDWARE APPROACH

The standard D.E.C. furnished hardware configuration has been modified by us, or to our specification, in several respects to meet some of the special requirements of the FASBAC system. These modifications are in the areas of:

1. automatic failure recovery

2. dataset control

3. additional machine status displays, and

4. the addition of an index register.

The program failure recovery subsystem senses program halts or their logical equivalent and automatically loads a wired bootstrap program into PDP-8 memory. This bootstrap program then reloads the main program and sets it running.

The dataset control subsystem senses and controls the condition of Bell System 103 Series datasets, or equivalent. It senses the Ring and Carrier Status leads from the dataset, and controls the Data Terminal Ready and Request to Send leads to the dataset.

Indexing (Fig. 5) in the PDP-8 is accomplished by cheating, since all of the 12 bits in the PDP-8 word are already used. It is not possible to specify explicitly indexing on memory reference instructions. Our approach is to use implied indexing. The register used for the index is the line select register, which is part of the 685 Serial Line Multiplexor. This 7-bit register is inclusively-ORed with memory address bits 5-11 if certain other logical conditions are present. These conditions are that the index register is enabled, not inhibited, and that the DEFER cycle is in progress. The limitations of this technique are obvious, but it is quite adequate for the addressing of the many tables found in a communications multiplexor, and quite essential for the successful operation of this one.

A gate to turn the register on and off is required, since not all indirect memory references need to be indexed, and the inhibit function is necessary in order to restore the state of the index register on/off switch when exiting the interrupt handler. Suitable instructions are provided to operate these functions.

The machine status display additions consist of bringing the 7 bits of the index register, the two bits of the 689 ADF group counter, the 689 ADF enable flip flop, and the index register on/off flip flop to the lamp positions normally used for the MQ display.

SOME OBSERVATIONS

It won't be possible to make a detailed functional critique of the system in a few paragraphs, or even in a book. However, we would like to pass on some of our more important conclusions concerning the use of this equipment and these programming techniques for the low speed line multiplexing functions.

The first is that the development effort put into the improvement of the PDP-8 addressing capability, by the addition of the indexing facility, has been highly rewarded. The program is much shorter than it would be otherwise, with a concomitant reduction in the number of instructions executed.

The second conclusion is that, for a program as large as the FASBAC 680 DCS EXECUTIVE, a PDP-8 Assembler running on a larger, faster machine is worth many times its cost, by reducing turnaround time and frustration. We developed such a PDP-8 Assembler, running on the UNIVAC 1108, and we have found it extremely helpful. This assembler is available as a proprietary program from University Computing Company.

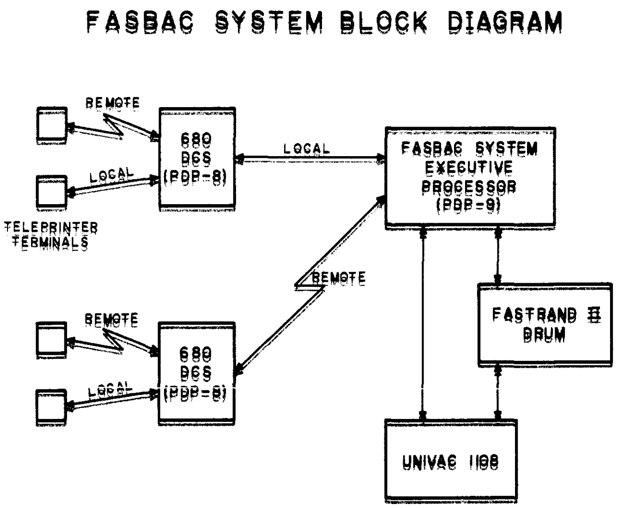


Figure 1 = FASBAC System Block Diagram

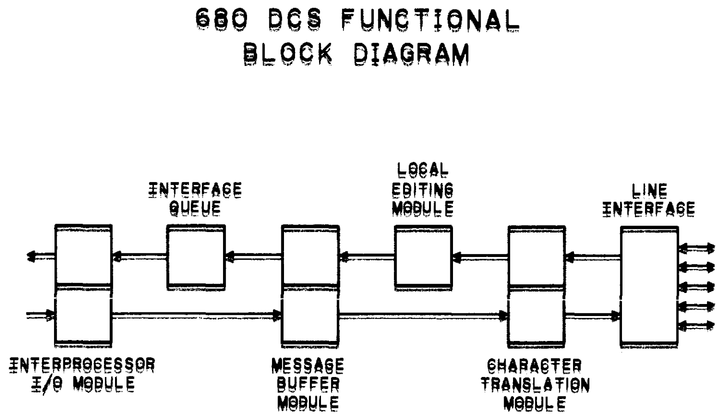

680 DCS FUNCTIONAL BLOCK DIAGRAM
INTERFACE QUEUE
LOCAL EDITING MODULE
LINE INTERFACE
INTERPROCESSOR I/O MODULE
MESSAGE BUFFER MODULE
CHARACTER TRANSLATION MODULE

Figure 2 = 680 DCS Functional Block Diagram

680 DCS MEMORY LOADING CALCULATION

$S_1 = 8*(T_1\ M + T_2)$

$T_2 = (8.38*10^{-6})* \sum_{K=1}^{M} N_K$

$L = 100* \left[(S_1+S_{2M}) \left(\sum_{K=1}^{M} R_K/M\right)+R_I\right]$

WHERE:

S_1 = BIT INPUT SERVICE TIME, SEC/BIT

T_1 = INTERRUPT SERVICE TIME, EXCLUDING TTI, SEC/INTERRUPT

M = NUMBER OF DIFFERENT LINE TYPES (SPEEDS)

T_2 = TTI EXECUTION TIME, SEC/INTERRUPT

N_K = NUMBER OF LINES OF TYPE K

S_2 = BIT-TO-CHARACTER ASSEMBLY TIME, SEC/BIT

R_K = BIT RATE OF LINES OF TYPE K, BITS/SEC,

R_I = INTERPROCESSOR BUFFER TRANSFER LOAD ($\leq$ .1 SEC/SEC)

Figure 3 - 680 DCS Memory Loading Calculation

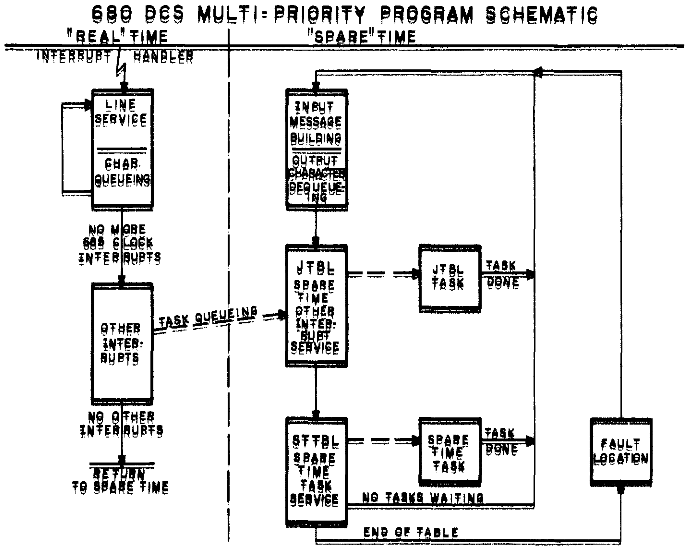


Figure 4 - 680 DCS Multi-Priority Program Schematic

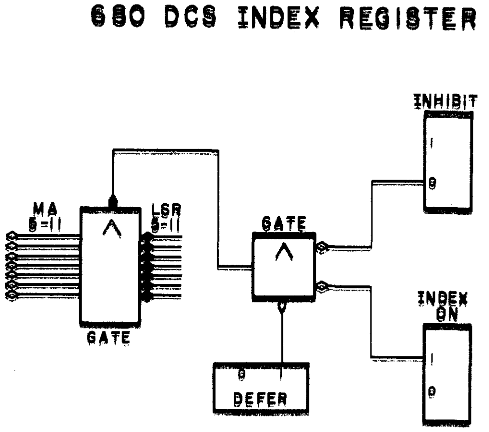


Figure 5 - 680 DCS Index Register

A REAL-TIME MULTIPLE TASK EXECUTIVE PROGRAM WITH A BUILT-IN CONSOLE UTILITY PACKAGE FOR PDP-8/S AND PDP-8 COMPUTERS[1]

C. D. Martin, Jr.   R. L. Simpson
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37830

ABSTRACT

An executive routine was developed for the PDP-8/S and PDP-8 computers. This routine schedules process control tasks in real-time and establishes operating priorities. The program (including the utility package) occupies about one-third of a 4096-word memory block and accommodates eight major control tasks. The only hardware addition to the standard computer configuration required by the executive is a real-time interrupt.

INTRODUCTION

If a digital computer is to be used effectively for process control, a means must be provided for scheduling various programs that the computer is to execute and for assigning an operational priority to each program. As an example of the diversity of the programs, the program having the highest priority in a system might be one for scanning the analog inputs to the computer, digitizing them, checking them against prestored limits, initiating the printout of messages alerting operators that signals are out of limits, and storing the digitized values in locations accessible to other programs. The program having the next highest priority might be a control algorithm for examining the digitized analog signals, comparing them against desired values, and initiating a control output to minimize the difference. These might be followed by other programs such as one for printing periodic logs and one for writing the digitized data on magnetic tape for further processing by a larger computer.

Scheduling the execution of the programs, maintaining priorities, and coordinating the use of input-output equipment by several programs are accomplished by a program called a real-time executive. Various actions within the system are triggered by hardware interrupts from input-output equipment and from a real-time clock. A real-time clock gives the time of day while an internal computer clock synchronizes the computer operations.

MEMORY REQUIREMENTS AND SYSTEM OVERHEAD

The real-time executive was developed for the PDP-8 and 8/S computers to schedule the execution of eight different computer programs, or tasks, at desired time intervals. Input-output functions are coordinated so as to eliminate conflicts between tasks. The system can be expanded to accommodate more tasks. An initial limit of eight tasks was selected, because the computer used for developing the system had a memory of only 4K words (see memory map, Fig. 1). The executive system now requires about 576_{10} or 1100_8 words of memory, and the on-line--off-line utility package occupies about 640_{10} or 1200_8 words of memory, leaving the remainder of the memory for system tasks.

The system operates in response to a 60-hertz interrupt in the PDP-8 computer and a 10-hertz interrupt in the PDP-8/S computer. System overhead for the PDP-8/S computer is about 8%, leaving 92% of the available time for task execution. The overhead for the PDP-8 computer is about 3%.

REAL-TIME SCHEDULING

Real time is maintained in the system in two memory words, the cycles counter and the minute counter. In response to a clock interrupt, the cycles counter, which is set initially with the negative number of cycles per minute, is incremented by one count. When the count becomes zero, the minute counter, which is set initially with the negative number of minutes in a day, is incremented. By use of these counters the time of day can be calculated for log purposes. Every minute, when the cycles counter becomes zero, the counter is initialized with the negative number of cycles in a minute. At midnight, when the minute counter becomes zero, the minute counter is reset with the negative number of minutes in a day.

Several tables are used for internal control and status indication of the system. In every table each entry corresponds to a given task. The tasks are numbered 0 to 7 in decreasing order of priority; that is, task number 0 has the highest priority, and task number 7 has the lowest priority. All tables are arranged in this manner to make it easier to index all tables with a single pointer, which is called the task count. The task count can be added to the starting address of any table to form the address of the table entry for the particular task. If no task is being executed the task count is 10_8. The tables are as follows:

1. The status table indicates the status of a task at a particular time.

2. The location table contains the entry point of each task.

3. The period table contains the negative number of minutes between consecutive executions of a task.

4. The offset table controls executions of tasks to optimize scheduling.

[1] Research sponsored by the U.S. Atomic Energy Commission under contract with the Union Carbide Corporation.

5. The cycles and minutes rundown tables are counters for the time between executions of tasks.

6. The printer queue table contains the address of the first ASCII character to be printed for each task.

7. The floating-point-package queue table is used to store the return address to the task after the floating point package has been used.

STATUS TABLE

The information bits in each word of the status table are assigned as follows:

1. If bit 0 is a 1, the task has been actuated for execution.

2. If bit 1 is a 1, the task is waiting for the completion of an input-output function.

3. If bit 2 is a 1, the task has been interrupted.

4. If bit 3 is a 1, the task is waiting for the floating-point package.

5. If bits 4 through 11 are all a zero, the task is not scheduled to be actuated by the clock.

If a task is to be scheduled by the clock, at least one of these bits must be set at a 1.

A task can be actuated for immediate execution by setting bit 0 to a 1 by the keyboard utility package or by another task. The execution of a task is begun or the execution of a previously interrupted task is resumed as a result of a scan of the status table which is scanned after every clock, keyboard, and printer interrupt, and after completion of a task. The scan is always started with the entry corresponding to the highest priority task (task 0). The first task found whose bit 0 is a 1 and whose bit 2 is a zero is executed from the address in the location table. Bit 2 will always be a 1 if either bit 1 or bit 3 is a 1, but not vice versa; that is, a task waiting for the completion of an input-output function or the floating-point package must have been interrupted. If bit 2 is a 1 and bit 3 is a zero, the registers are restored and execution is resumed from the interrupted address. If bit 3 is a 1 (waiting for the floating-point package), the floating-point-package busy flag is checked; if the package is busy, the task count is incremented and the next task is checked. If not busy, the busy flag is set, the registers are restored, and execution is resumed from the interrupted address.

OFFSET, MINUTES RUNDOWN, AND CYCLES RUNDOWN TABLES

The offset table indicates the period in negative number of cycles for a task having a period less than 1 min.

Every entry in the cycles rundown table is incremented every clock cycle to keep the next execution time current with the clock. When an entry in this table becomes zero, the corresponding entry for the same task in the minutes rundown table is incremented. If the corresponding entry too becomes zero, the task is actuated (bit 0 in the status table is set), and the minutes rundown table is reset from the corresponding entry in the period table. The entry in the cycles rundown table is reset to the corresponding entry in the offset table if the period for that task is less than 1 min, otherwise the entry in the cycles rundown table is set to the negative number of cycles per minute.

PRINTER CONTROL

The printer queue table has an entry for each task. The pointer to the task that is using the printer is called the printer busy flag, and it is loaded with the task number. When a task calls for a printer output, the location of the first ASCII character to be printed is stored in the printer queue table entry for that task. Once the output for that task is started, the address is incremented every time a character is printed, and the contents of the incremented address are examined for a negative number. When a negative number is found, the entry in the table is zeroed, bit 1 in the status word for that task is cleared, and the table is scanned for a nonzero entry to start another output on the printer. If no requests are in the printer queue table, the printer busy flag equals 10_8.

INTERRUPT HANDLER

The following list describes the interrupts that the real-time executive is set up to handle (Fig. 2):

1. Low power--The response to this interrupt is to store the contents of the accumulator, of the link, and of the program counter (return address), to store a "jump to a restart routine" instruction in cell zero, and to halt to wait for the power to be restored. When power is restored, the computer is automatically started at cell zero. The "jump to a restart routine" instruction is executed, and the restart routine restores the contents of the accumulator and the link, and jumps to the next instruction in the program that had been interrupted.

2. Clock--The response to this interrupt updates the time-of-day clock (cycles and minutes counters), checks for zero in the minutes counter (indicates midnight), and increments the cycles rundown entry in each task. Since this incrementing could result in the actuation of a higher priority task (having a higher priority than the task interrupted by the clock), the contents of the registers for the interrupted task are saved and are restored before execution of the interrupted task is resumed.

3. Keyboard--This interrupt occurs when there is a character in the teletype output buffer awaiting transfer to the accumulator. This character is checked to determine if it is an "Alt Mod" (alternate mode) character. If it is, the utility package is actuated for immediate execution. If it is not, the character is stored, and the "input-output in progress" bit in the status word for the utility package is cleared so that execution of the task will be resumed after the status

table has been scanned and no task is found for execution that has a higher priority.

4. Printer--This interrupt occurs when the printer has typed a character and is ready to type another. The response procedure was explained in the description of the printer queue table.

5. Parity--Only the PDP-8/S computer is equipped with this interrupt as a standard hardware feature. Since the result of any parity error will soon become evident, the parity errors are counted by incrementing a parity counter and an attempt is made to resume execution of the interrupted task.

The analog-to-digital converter (ADC) interrupt was removed from the interrupt buss on the PDP-8/S computer. All analog-to-digital conversion is done in a task, because a check for an ADC flag requires 38 μsec, but conversion (using the AD8S) requires only 20 μsec. Also, since contents of the registers are saved after every interrupt, the PDP-8/S computer requires more than 1180 μsec and the PDP-8 computer requires more than 75 μsec to save registers. Thus the removal of the ADC interrupt results in a significant time saving.

REGISTER SAVE

To maintain continuity during execution of a task, the contents of eight "registers" are saved when execution of a task is interrupted, because it might happen that a task with a higher priority than the one in execution will be actuated. When execution of the interrupted task is resumed, the task must first be restored to its previous state. The registers saved are the AC, the link, the return address in the interrupted task, and core memory locations 0016, 0017, 0020, 0021.

GENERAL CONSIDERATIONS

When a task is completed, the task return to the executive must be through a task completion routine which clears bits 0 through 3 in the status word for that task and then scans the status table for another task awaiting execution or waits for an interrupt.

All input-output operations must be scheduled by the executive program. When scheduled, all messages are output intact; that is, once a character string is started, all of that string will be printed before another character string will be started. All keyboard input is processed through the utility package.

The floating-point package was changed so that it could be used by a task through the real-time executive.

CONSOLE UTILITY PACKAGE

By use of the console utility package the operator or programmer can communicate with the computer either on-line or off-line; that is, he can type a mnemonic code which will set up the following operation: store into the memory from the keyboard, read and punch binary paper tape, obtain an octal dump, actuate a program to be executed only once ("one shot"), clear defined parts of the memory, set defined parts of memory to a specific bit configuration, and disable the keyboard. Before any instructions or information can be stored in the memory from the keyboard or paper tape, the limits of the storage must have been specified. This prevents arbitrary storage and destruction of any program outside the "legal" limits.

The utility program checks all characters typed from the keyboard to determine that the characters are the proper type (alphabetic or numeric) for their position in the instruction. If they are not, the teletype bell will ring, and the character will not be accepted. For example, letters are accepted only as the first two characters, then a comma, and then only octal numbers. Usually, the utility package is run as the task of highest priority, but it can also run at any other priority level.

The console utility package functions with their mnemonic commands are given in the following list:

AR,XXXX,YYYY	Defines the first and last addresses of the legal core storage area.
CL,XXXX,YYYY	Clear to zero from XXXX to YYYY.
GO,XXXX	Go to XXXX and execute as lowest priority (one shot) task.
MW,XXXX,YYYY, ZZZZ	Sets ZZZZ into the core from XXXX to YYYY.
PT,XXXX,YYYY	Punches binary tape of core from XXXX to YYYY (inhibits all other teletype message outputs from tasks).
RT,XXXX	Reads binary tape; location can be offset by XXXX$_8$ words (integer number of pages).
TI,XXXX,YYYY	Store into location XXXX the the contents YYYY.
TO,XXXX,YYYY	Type out the contents of XXXX through YYYY.
FI	Disables the keyboard input to all characters except "ALT MOD" and makes all core locations illegal.

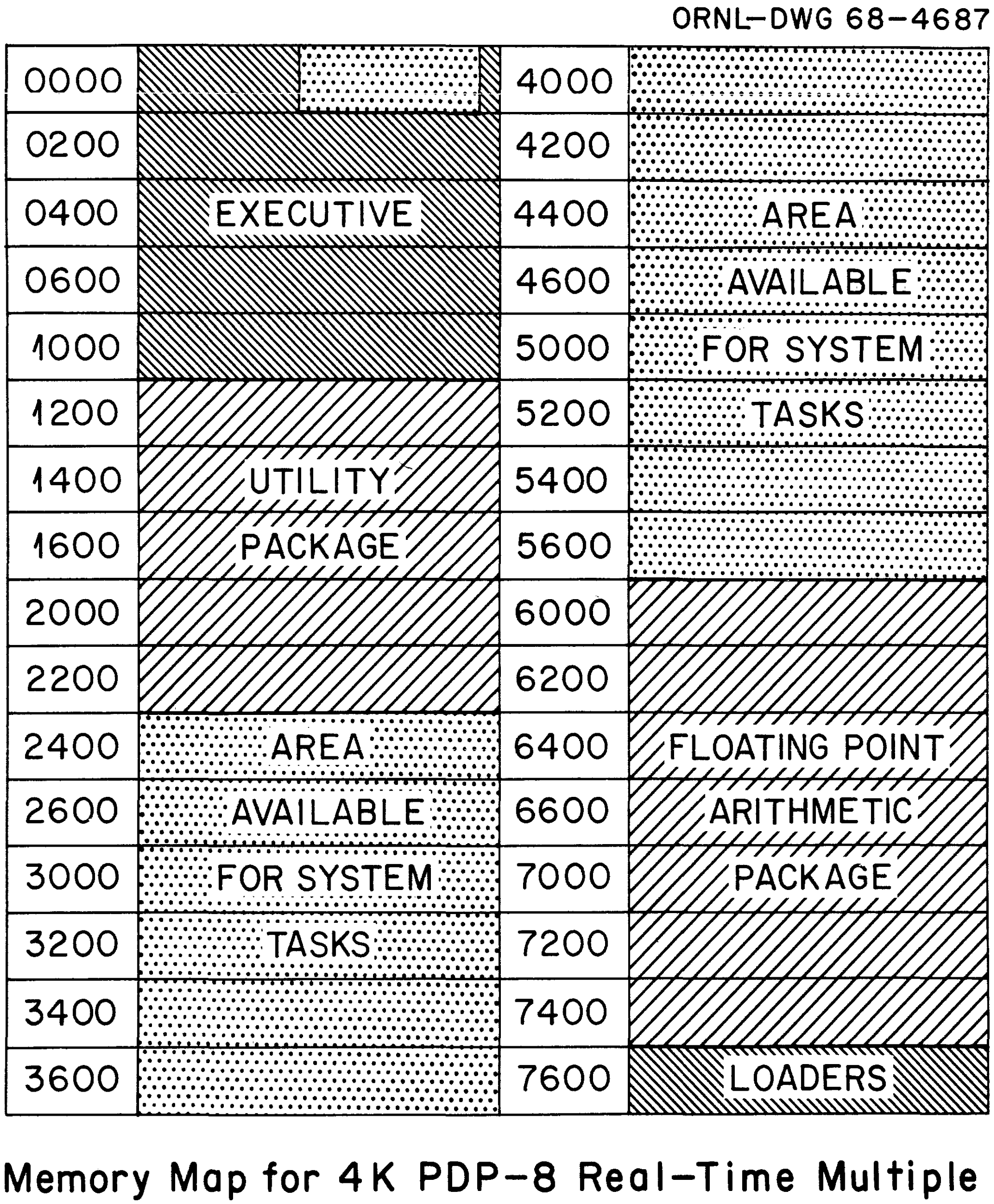
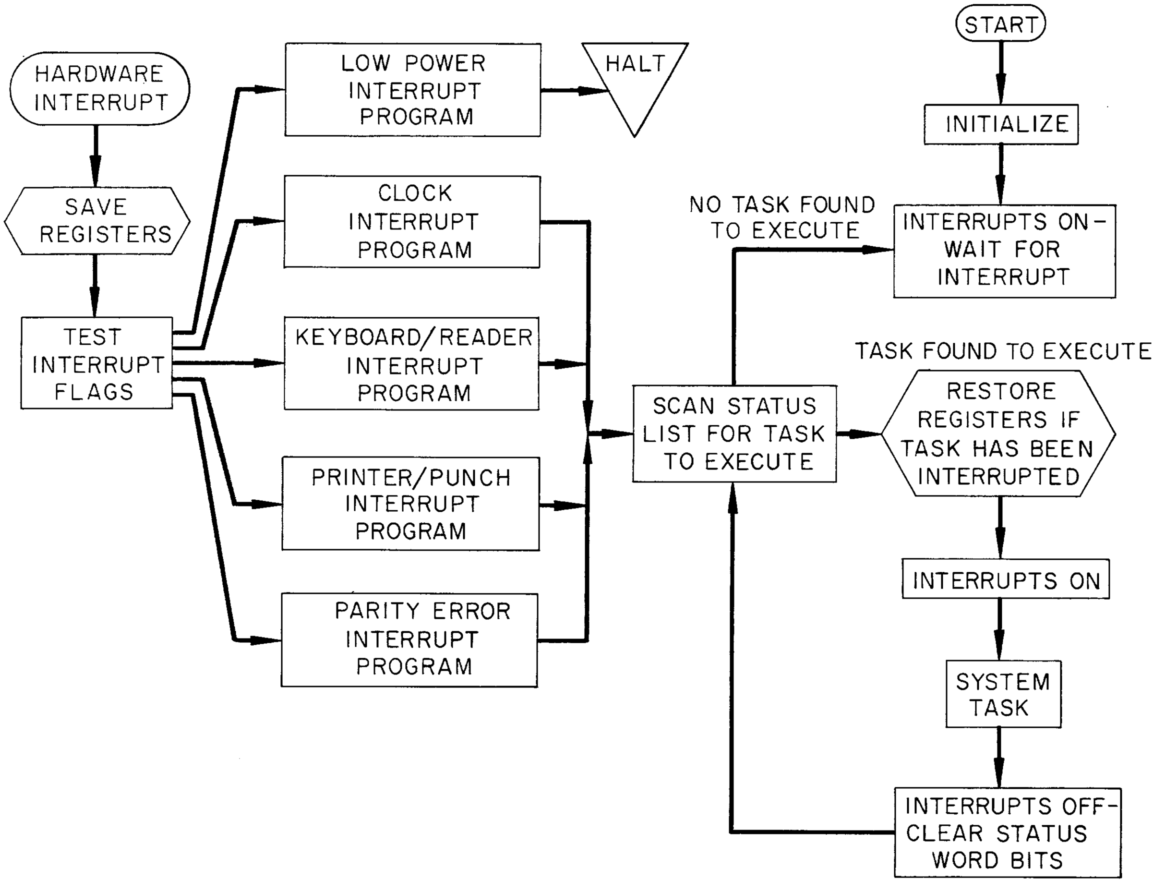


Figure 1 Memory Map for a 4K PDP-8 Real-Time Multiple Task System

ORNL-DWG 68-4688


START
INITIALIZE
HARDWARE INTERRUPT
SAVE REGISTERS
TEST INTERRUPT FLAGS
LOW POWER INTERRUPT PROGRAM
HALT
CLOCK INTERRUPT PROGRAM
KEYBOARD/READER INTERRUPT PROGRAM
PRINTER/PUNCH INTERRUPT PROGRAM
PARITY ERROR INTERRUPT PROGRAM
NO TASK FOUND TO EXECUTE
INTERRUPTS ON - WAIT FOR INTERRUPT
TASK FOUND TO EXECUTE
SCAN STATUS LIST FOR TASK TO EXECUTE
RESTORE REGISTERS IF TASK HAS BEEN INTERRUPTED
INTERRUPTS ON
SYSTEM TASK
INTERRUPTS OFF- CLEAR STATUS WORD BITS

Schematic Flow Chart of Real-Time Multiple Task Executive for PDP-8/PDP-8S Computers.

Figure 2

FOCAL*

Rick Merrill
Digital Equipment Corporation
Maynard, Massachusetts

Abstract

A new small computer language called FOCAL has been designed and written at DEC to be used in Formulating On-Line Calculations in Algebraic Language. This paper is a discussion of how size (3K), power (14 functions), and flexibility (several options) were achieved in designing an easy-to-use language and in programming it for the PDP-8 family of computers.

*This paper was not received for publication.

A PDP-6 LANGUAGE FOR SIMULATING COMPLICATED BIOCHEMICAL SYSTEMS

Johnson Research Foundation
Department of Biophysics and Physical Biochemistry
University of Pennsylvania
Philadelphia, Pennsylvania 19104

ABSTRACT

A language for simulating biochemical systems composed of complex sets of chemical reactions is described. This is written in FORTRAN IV; a machine-independent version of it has been prepared, but is appreciably more powerful when set up for on-line interaction, which is presently done with a PDP 6 including card reader, printer, and scope display. The input is in the form of chemical reactions and associated numbers, on cards; output in tabular and graphical form. The principal mathematical operation is the solution of differential equations representing the time behavior of the chemical concentrations, but alternative mathematical treatments are being added. A number of applications of this language will be described.

For about ten years the author and his associates have been engaged in simulation of complex chemical systems.[1,2] The mathematical operation involved is primarily the solution of differential equations by numerical methods, but the actual input language is that of chemistry. Digital computer programs have been developed to convert chemical reactions into differential equations, solve them by numerical methods, and edit the results into tabular or graphic form.

This type of work is commonly done with analog computers, which are particularly well adapted to solve differential equations. Why then bother with a digital computer, in particular with a digital the size of a PDP-6? There are two principal reasons for this: the number of differential equations to be considered, and their behavior.

To represent a biochemical system of any complexity it is usually necessary to solve at least dozens of differential equations, and sometimes even hundreds. Furthermore, these differential equations are often quite non-linear, and as a result are beyond the capabilities of any but the largest analog computers.

The coefficients which are required to make these differential equations represent biochemical reality, (especially if real enzymes are involved), are such as to make the equations very badly behaved. Sometimes it is impossible to scale these numbers into an ordinary analog computer. When they are used in differential equations on a digital solved numerically computer it requires tens of thousands of integration steps to reproduce a smooth curve that superficially looks as if it might reasonably require a few hundred. Hence the need for a large and fast digital computer. Specialized analog computers have been or are being built which may help with this problem, but they are not generally available. Hybrid computers might be useful here.

A series of programs have been prepared, being revised from computer to computer, to meet this need. These have accepted as input some form of chemical reaction accompanied by initial conditions, and have produced output in the form of graphs of the differential equation solutions against time, and/or tabulations of results (usually of the state of the simulated system at any time). Such programs also have the capability of acting as differential equation compilers. They have sometimes been accompanied by auziliary routines, which have been able to do things as complicated as conducting pattern recognition studies on the results of a systematic single enzyme simulation. The general format for all of these is that a generator program accepts the chemical equations and produces a routine to solve the corresponding differential equations; this routine then starts from input initial conditions, and solves the equations to yield graphical and tabular output.

The fundamental algorithm for converting chemical reactions to differential equations is as follows: Multiply together the concentrations of all the chemicals mentioned on the operative (usually left) side of a reaction, times the appropriate rate constant; and for each substance appearing or disappearing in the reaction, these products (called fluxes) are summed, with proper sign, to yield the time derivative of the concentration of that substance For the reaction A + B = C + D (the equals sign indicates reversibility) there are the fluxes:

$$K_1(A)(B) = \text{flux } 1$$

$$K_2(C)(D) = \text{flux } 2, \text{ and}$$

$$\frac{dA}{dt} = \text{flux } 2 - \text{flux } 1.$$

Two types of modification to this algorithm are provided. It is possible that in a reaction more than one molecule of a given substance may appear or be consumed. In this case it is necessary that the flux for that reaction be multiplied by an appropriate coefficient (called stoichiometry) when being summed into the differential equation. Sometimes it happens that a substance (such as a catalyst) can help control the rate of a chemical reaction without appearing or disappearing in it at all; here it is necessary to have the concentration of such a substance multiplied into the flux, which is not included in the sum for the derivative of that substance.

Once the differential equations (invariably with respect to time) have been compiled and are ready for solution, their initial conditions are inputted and the equations themselves are then solved with the simplest possible method, the first-order Euler method. This choice of differential equation solving method may sound strange, as higher order methods usually work better. However, with the particular numbers to be found in realistic chemical or biochemical systems, first-order methods seem to perform better then higher-order methods. This is not dependent on things like roundoff error; indeed, it has been observed in calculations with desk calculators as well as with computers of widely differing word-length, used by a variety of people. Usually a great accuracy is not required of the differential equation solver; few biochemical systems can be measured with an accuracy of better then one per cent, and often the error is much greater.

The newest version of this language has just been implemented on the PDP-6. Actually it is a specialized and more powerful version of a language which is as machine independent as possible and is described in detail elsewhere[3]. This language is written in FORTRAN IV, and is as consistent as possible with the requirements of the FORTRAN IV compilers of most large computers. The only respect in which it is not fully machine independent is that machines with short word lengths (32 bits) require the arithmetic to be done in double precision; the 36-bit word of the PDP-6 appears to be long enough to permit single precision operation.

In effect this constitutes a two-pass compiler: in the first pass chemical reactions are converted to differential equations, written in FORTRAN: in the second pass these are compiled by the FORTRAN compiler, along with some subroutines whose dimensioning is dependent on the size of the system being studied. These are then loaded together with a subroutine library to produce an operating program, which is commonly used fairly often before requiring revision. This language is intended primarily for batch processing machines, and is completely card and printer oriented. It is possible to add user supplied subroutines at a number of points to perform services that the language in general is not explicitly intended for; thus far these appear to have been very much on an individual problem-dependent basis. This program set up for a simple test system occupies 17k of core on the PDP-6, requires one DEC tape for the program itself, and the equivalents of card reader input and printer output.

Running simulation in a batch mode, however, is not the most efficient way of doing simulation. In fact the principal reason why our facility has a PDP-6 is to permit easy interaction between a user and the machine. Accordingly a PDP-6 specific version of this computer-independent program has been prepared, which permits much greater flexibility in user interaction. It will accept the same input as the computer-independent version, and perform the same calculations, but the user interaction is quite different. This version, in addition to the usual teletype and DEC tape, requires 25k of core, the scope, and the equivalents of card reader and line printer.

To make this PDP-6 version a variety of graphical output and teletype subroutines have been added to the machine-independent language; more may be added later. It is expected that the primary input will still be on cards, but the user has the ability to change or add input from his teletype, when the machine is running in time-sharing mode, and to see the partial results as they are calculated, either on teletype, on the printer, or on the scope display. The user may in particular start from a given situation and modify it as partial results are obtained.

The principal graphic output (when graphic output is desired) with the PDP-6 is a graphical display scope as exemplified in Figure 1. This is relatively expensive in terms of core storage (8K), but not in terms of computer time. Substantially the same display normally appears on the printer or a printer substitute (usually "forthe permanent record"), and may also be put out on the plotter. While one can if necessary follow a simular picture on the printer as it is produced, this is definitely inferior, both because there is less resolution and because the most recent part of the picture is still hidden by the printer ribbon. The scope can show up to 8 curves of concentration against time simultaneously, the curves being identified by their line quality (which does not show up well in the photograph, Figure 1) and specified by cards in the initial condition input. A mechanism is provided whereby the user may delete curves that are not currently useful to him, before or during a calculation. Graphical output may also be stored on a scratch tape for further processing.

Availability of a teletype on-line, especially in conjunction with graphical output permits the user to do the following things:

(1) Specify when he wants a printout of the state of the system being simulated (the current rate constants, fluxes, and concentrations and the derivatives and second derivatives of the latter). This is put out on the printer.

(2) Specify when he wants a subset (designated by card input) of the above, which may go out either on the printer or the teletype.

(3) Repeat this calculation, with initial condition changes specified by teletype.

(4) Go on to the next calculation specified on cards, with or without initial condition changes from teletype before starting.

(5) Make changes in the computation in progress and then continue. This includes changing concentrations, rate constants, etc., and also changing the permissible length of the calculation, the maximum and minimum integration step sizes, etc.

(6) Any reasonable combination of the above, as well as stopping all computations. This combination of capabilities permits the user to specify a set of models on cards, work with each of these (modifying it) until he is satisfied with it, and then go on to the next.

It is hoped to extend the convenience features so that this will be nearly as convenient to use as an analog computer. As a possible alternative input arrangement, we have added a series of potentiometers to our scope display (Figure 2). When all the appropriate software is written, it will be possible to use these as input devices to this program, probably by having each potentiometer be a multiplying factor for some number initially specified to the routine with cards.

In the simplest possible application this could replace a teletype input at the beginning of a calculation, (and probably does not offer much advantage over it). However, it is sometimes necessary to continuously vary an input throughout a calculation. Thus far this has been done mostly to simulate the effect of some of a model's environment which cannot be readily modeled by the set of differential equations which the computer is working. An example is shown in Figure 3, which is part of an ongoing study on an oscillating glycolytic system. This shows an enzyme velocity profile as a function of time, which might be imputted in this way. Here the user can by suitably twiddling the potentiometers, evolve such a curve by setting the correct momentary value needed to make things come out right and having the computer keep track of what he has been inputting. If necessary one may go part-way back through the calculation (there is provision to save designated states of the system and start from there) and repeat this input operation with slightly different potentiometer settings to get a fit. This operation would seem to involve sufficient pattern recognition so that it is more easily done, at least in the early stages, by a man looking at a scope than by any automatic optimization routine.

Some flexibility is specifically built into the program to compensate for the differences in behavior between the card reader and a DECtape assigned as a card reader. Mechanisms are provided to in effect tell the program about its environment, which the monitor will not do for it. The card reader particularly differs from a DECtape in the way it is buffered, and if one is to read part way through a deck of cards, do some computation and then read more, different arrangements are necessary in the two cases.

It is possible that this language may be extended in a higher-level direction, so as to make the individual calculations part of automatic optimization techniques, for example, or even to have processes that may be described as artificial intelligence. An example is in trying to fit a given experimental curve by a given simulated mechanism; the computer may find the best fit, decide that it is not adequate and that it deviates in a certain way (e.g., by having the computed curve be too low at the longer time intervals) and then revise the mechanism so as to adjust for this and go back and fit it over again. It is unlikely that this can be made completely independent of human intervention in the near future, but such a process probably will be more efficient if the computer and the human are cooperating than if either is doing it alone.

The input format and some of the aspects of machine independence are the result of the experience of a number of people, and the contribution of Dr. E. M. Chance II to both of these should be particularly acknowledged. This work was supported by grant FR-15 from the National Institutes of Health.

References

1. Garfinkel, D., Rutledge, J.D., and Higgins, J.J. Comm. Assoc. Comput. Mach. 4, 559 (1961).

2. Garfinkel, D., Simulation of Biochemical Systems, in Computers in Biomedical Research, Vol. 1, ed. by B. Waxman and R.W. Stacy. Academic Press Inc., New York, (1965) p. 111.

3. Garfinkel, D., Computers and Biomedical Research (in the press).

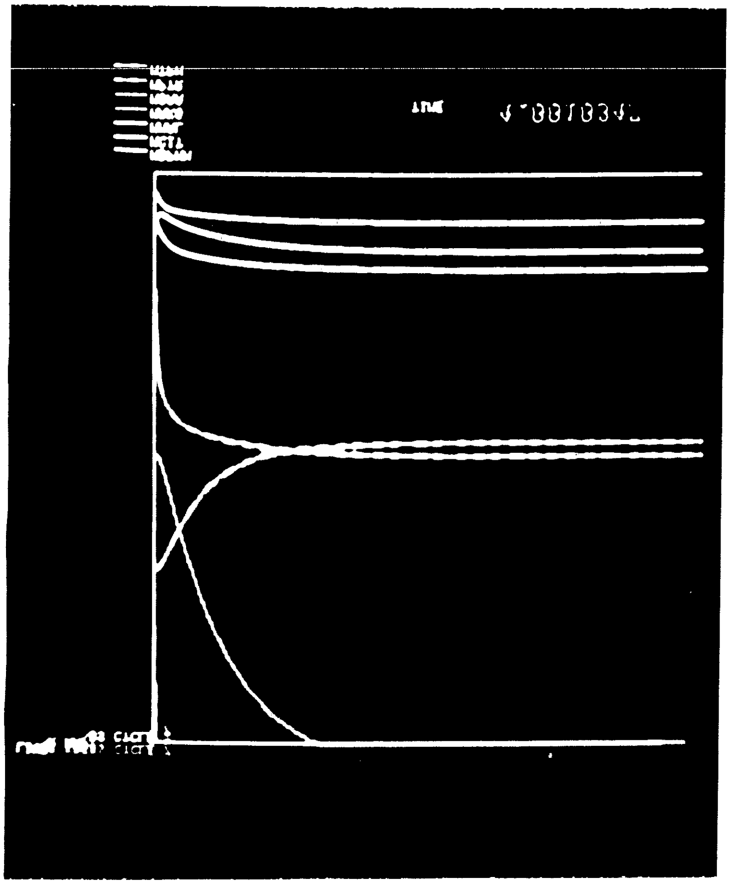


Figure 1 Photograph taken from the 340 Scope Display showing concentrations as a function of time. The lines are distinguishable by their detailed fine structure, which is difficult to photograph.

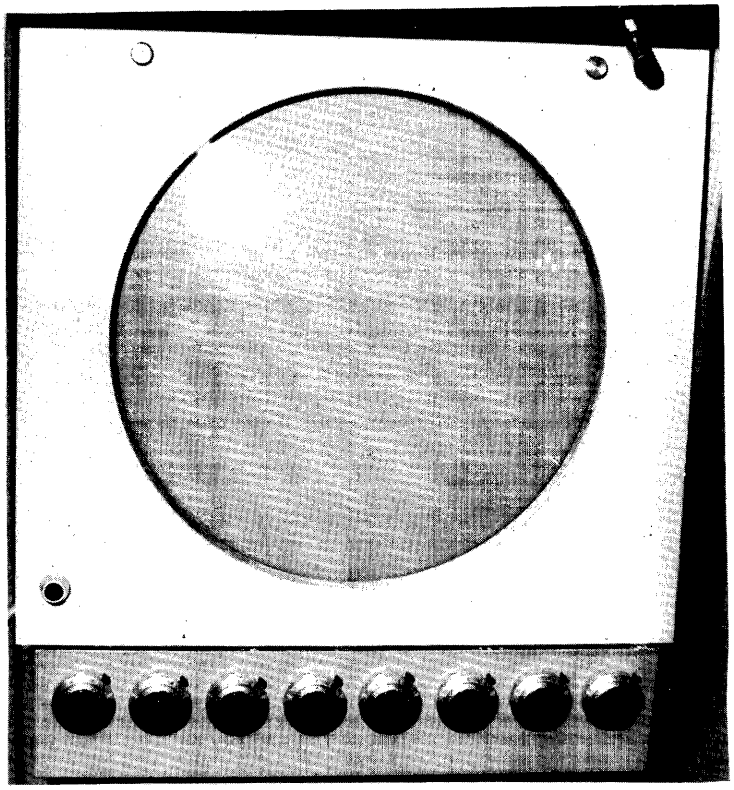


Figure 2 Potentiometers mounted on the scope display unit.

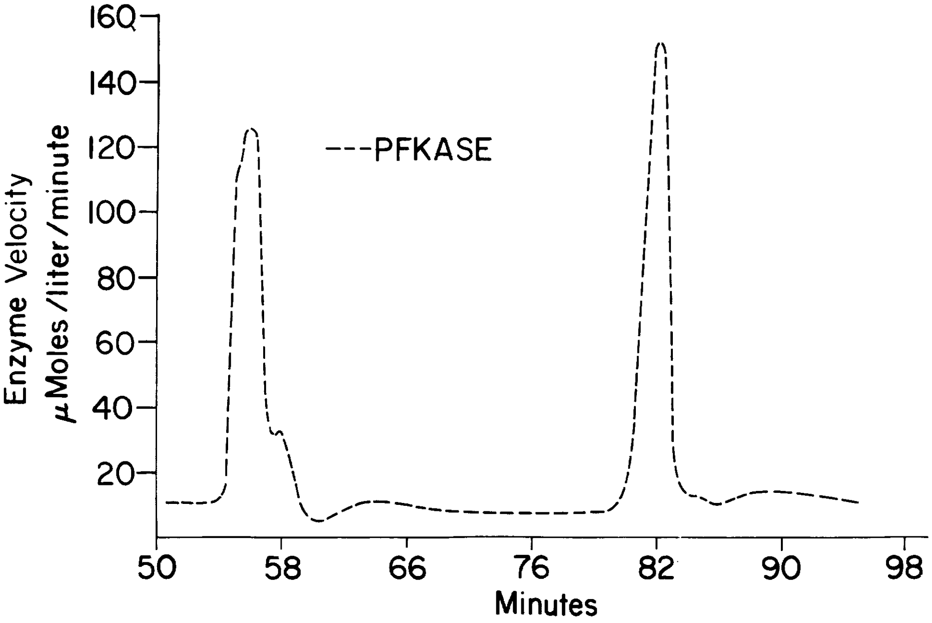


Figure 3 A profile of enzymatic activity as a function of time (phosphofructokinase in an oscillating glycolytic system), as an example of a type of input which may be inserted with the scope potentiometers.

A GENERAL LANGUAGE FOR ON-LINE CONTROL OF PSYCHOLOGICAL EXPERIMENTATION

J. R. Millenson
Dept. of Psychology, University of Reading
Reading, England

ABSTRACT

A problem-oriented language is being developed for on-line process control of psychological experimentation. The language consists of nested blocks of simple English statements familiar to every experimental psychologist. The function of this language is to produce an Automated Contingency Translator (ACT) which samples and updates a number of independent time-shared experimental environments 60 times a sec. Experimental procedures are mapped by the ACT compiler from the English statements into a probabilistic finite state network in list structure format. An independent operating system (which in the PDP 8, 4K version overwrites the compiler) then executes the list structure automata: that is, runs the experiments, records and retreives data and admits low priority background programs in any available dead time.

Psychologists have made extensive use of computers for data reduction and for simulation studies of behavioral processes. They have been, with some notable exceptions[1,4,15,17] somewhat more diffident in applying computers directly to a third major class of problems, namely the laboratory control of their experiments. With the appearance of the small, fast, accessible, and relatively inexpensive general purpose machines this reluctance on the part of psychologists can no longer be principally ascribed either to economical or instrumentation difficulties.

If we look closely at the situation we discover that for problems such as statistical analysis and data reduction, flexible routines and sub-programs for building a variety of special programs have been standardized in the algebraic-like languages of FORTRAN and ALGOL. Similarly in the fields of artificial intelligence and simulation of human problem solving by heuristic methods, well established special purpose interpreters such as IPL-V, LISP, and COMIT provide the investigator with a convenient language to formulate and manipulate his problems. In the field of laboratory control, however, the absence of any such general problem-oriented languages[5] is conspicuous. Psychologists who wish to use the small computer to program experiments are generally obliged to learn a complex and unfamiliar code to communicate with their machine. Many are deterred by this language barrier;even for those who surmount it, the low-level machine or assembly languages that they have had to learn prove a poor vehicle for easy expression and creative exploration of new procedures. There have been attempts[4] to use existing problem-oriented languages to do the job. But these problem oriented languages are oriented for the wrong problem, and thus whileFORTRAN, for instance, makes algebraic formula translation and manipulation a routine exercise, it provides far less obviously a natural language for the logical structure of process control applications.

The key to a natural language for psychological procedures is an adequate theory of those procedures. While there have been a few attempts in the past to formulate the independent procedural variables of psychology into a unified framework[8,9,14] these attempts have contained important restrictions and indeterminancies which limit their general application to all procedures of the field. Recently, however, A. G. Snapper and his associates[6,16] employed the concepts of cybernetic machine theory[2] to demonstrate what amounts to a proof that all behavioral procedures can be formulated as finite state automata. They exploited this discovery to write a very general program for the PDP 8 computer to process control in real time a variety of animal conditioning procedures associated with schedules of reward and punishment.

Neither Snapper's group, nor Marlowe[7], who also seems to have seen the implications of finite state automata theory for a general problem-oriented psychological process-control language, went so far as to evolve a general language for framing the procedures of psychology. Marlowe explicitly noted the practical difficulties in developing such a language and speculated as to whether "the effort required...might offset any gain made by using such a programming language" (p. 10). There is reason to believe that this conclusion was overcautious. In what follows, the aims and lexical-grammatical properties of a compiler written for the Digital Equipment Corporation PDP 8 family, purporting to establish such a language, are described.

General Aims

From the outset it appeared that five major conditions would have to be satisfied by the language. Firstly, the language was to be a perfectly general problem-oriented one, oriented to the particular problems arising in on-line process control of behavioral experiments. It must be able, without any ad hoc additions to its basic form, to describe, and therefore given the support hardware, carry out the procedures of nearly all experiments ever done in experimental psychology. Only in that case could the kind of generality that would permit the investigator to use the language as a conceptual model or vehicle for creating his procedures of the future be assured. The utility of these new procedures, far more than its ability to simulate the procedures of the past is likely to constitute the ultimate justification of the language. It was thus apparent that the language should provide no special constraints for any one area of psychology, even though at present certain areas (for example, automation of conditioning techniques, recording of neuro-electrical phenomena) might be more obviously computer oriented than others.

A correlary to this first condition was that since the language was to provide a variety of investigators with differing backgrounds and theoretical dispositions with a general tool, the language must not possess a bias towards any one particular method of analysis within psychology. Thus it should be possible for any psychologist of whatever persuasion to be able to describe his procedures in this language. The general nature of Snapper et al's contribution assured that since all psychological procedures could be reduced to a state diagram, in principle this would indeed be the case. But the actual language that was to map that state diagram to a computer data structure must still contain as few idiosyncratic theoretical connotations as possible. For instance, while the language would clearly evolve with the ubiquitious terms of stimulus and response, it must in no way commit the investigator to a reflex psychology. Stimuli and responses, or if the investigator prefers, environmental situations and behavioral repetoires, are simply a natural and operational way to talk about the changes in environments and behavior patterns of living organisms that make up psychological experiments.

Secondly, parameter modification had to be integral, simple, yet powerful. As I[10] and others[17] have pointed out elsewhere the special promise of the high speed digital computer in controlling psychological procedures lies in its ability to adjust rapidly to very intimate behavioral properties of the subject. That adjustment consists of modifications, depending on the moment to moment status of those behavioral properties, not only of quantitative parameters but possibly even the very structure of the procedure.

Thirdly, the fundamental features of the program had to be viable with a minimum hardware configuration (e.g., a PDP 8/S with 4K of core and a single teletype) so as to make the computer a feasible economic proposition in even small laboratories. At the same time, the program should possess sufficient flexibility to expand easily its command features so as to exploit the additional hardware of fortunate users with extra core, back-up memory, auxillary teletypewriters, display scopes, analogue converters, and so forth.

Fourthly, however complex the hardware input/output interface might in reality turn out to be at circuit level, it must appear to the psychologist-programmer wishing to control it as simple as possible:e.g., with PDP 8 machines, a simple 12-bit parallel word. Thus the program, unlike ALGOL, but like FORTRAN, would standardize its input/output instructions by assuming a standardized interface ~~terminal~~. Thus, complex microprogrammed machine input/output commands as such need never be seen directly by the programmer.

Fifthly, finally, and perhaps most important of all, the vocabulary, syntax, and grammar of the command language must be as close as possible to a simple English of everyday usage not unlike that which the investigator might employ in describing the procedures of his experiment to a colleague. The ability to use the language should therefore require almost no learning, it should be as insensitive to syntactical formalities (e.g., critical spacing, correct spelling, etc.) as possible.

Implementation in situ

A typical multi-access environment in which the computer might be expected to act as a process controller and data recorder for psychological laboratories is shown in Fig. 1. A PDP 8/S central processor is shown there directing the procedures and recording the data from four experimental stations. The system shown is slightly expanded from minimal configuration, containing an additional output teleprinter used exclusively for printing out critical results of the experiments, and a hi-speed paper tape punch for outputting selected aspects of the raw data.

In psychological experiments in environments of the sort shown in Fig. 1, a key role of the computer is to provide automatic control of the relations or contingencies that the experimenter desires to hold between selected aspects of the behavior of the subjects, and subsequent presentation and maintenance of selected changes in the subjects' environments. The language that is to be described for this general task amounts to an Automated Contingency Translator, hence its mnemonic name, ACT.

The associated I/O hardware for each of the stations of Fig. 1 must eventually terminate in two 12-bit words. One of these words registers the outputs (responses) from the subject and is read into the main arithmetic register of the computer, the accumulator. The other word consists of an input bit configuration and is strobed to the subject from the accumulator. A typical configuration for a rat subject in a conditioning experiment appears in Fig. 2. Only a portion of the two distinct words are used, 8 bits for R outputs and 6 bits for stimulus inputs. Learning the octal number system to designate the behavior and environment events that he wishes to control is very nearly the only specialized computer knowledge that the psychologist is obliged to acquire. The use of octal labels for stimulus and response events is an important way of simplifying the program to meet, in a small machine, all the aims described above. Thus, R115 set in Fig. 2 corresponds to the closing of a switch by the experimenter and a certain value (17) of a 5 bit analgoue voltage taken from electrodes attached to the skin of the subject. S5 corresponds to a compound stimulus: the presence of a 30 cps tone and the presence of a small "houselight" in the subject's chamber. In the protype installation for testing ACT these stimulus outputs represent -24 v to ground levels, and the inputs represent switch closures. Nevertheless the language of ACT itself is completely independent of how assertion voltages come to be on the R input lines, and what work one chooses to do with the assertion voltages the computer puts on the S output lines.

Writing in the language of ACT is expedited by first drawing a modified state diagram of the desired procedures. The units of these state diagrams are the state, shown as a rectange or a rectangular solid in Fig. 3, labeled with the actual environmental conditions; and the transitions from one state to another shown as the vectored lines in Fig. 3 with actual time or response values as labels. States may be (1) nested, as shown by the representation of boxes within boxes; and (2) they may be organized into multiple sets, or planes, of states as shown by the independenceof the top portion of the figure from the bottom portion. Not shown in the pictures is the ability to modify at each occurrence of a unique state the value of variable parameters of the experiment, and the execution of a variety of data retreival routines such as PRINTING, TAPING, DISPLAYING and so forth. These additions to the purely procedural contingencies relate ACT to the automata oriented REACTION HANDLER of Newman[12], an intriguing correspondence since Newman's program was designed for an entirely different task environment, that of expediting communication between on-line users and graphical light-pen displays.

To program his experiment the psychologist first works out a state diagram of it by listing the various sequential conditions and the temporal or behavioral events that will cause one condition to change to another. Then he connects up his structure with vectored lines corresponding to the contingency logic of the experiment. He then adds any special states he may need purely for recording purposes, or any states used for trapping the occurrence of rare or special experimental results. Then, referring to a representation of his input/output words (c.f., Fig. 2) he assigns numerical values to the states, the behavioral response events, and the times.

What are his restrictions on such labeling? Firstly ACT is a completely synchronously driven system. An external clock which in the prototype operates at line frequency (60 hz) restricts resolution of updating events to 60 times a second. Thus the shortest duration of a state is approximately 16.7 msec. (This is of course not a constraint of the language per se, only of the particular implementation of it. In the PDP 8 or 8/I a far faster clock would be practicable.) Secondly, the values of states must correspond to actual octal equivalents of 12-bit numbers. State 9 is thus ambiguous; response 72413 is too large.

Once in a state diagram, the procedure is ready to be mapped to the basic English of ACT. The vocabulary of ACT is shown in Fig. 4. It will be

SYMBOLS

S1.2	S̸	V(J)
R4	←	+
U55	(tab)	-
I,J,...N	&	=

ENGLISH WORDS

WHEN, WHILE, GIVE

IF
AFTER
FOLLOWING

GO, THEN, GO TO

PROBABILITY

Figure 4 Legal ACT I symbols and words.

observed that the with exception of only a few special symbols, the vocabulary consists of simple English words, S, U, and V letter abbreviations for states, R abbreviations for responses, and the integers I through N for integer variables. In order to distinguish states that are associated with the same environmental conditions (e.g., the same octal output word) but which occur at different points in the procedure and are therefore associated with different experimental conditions, a "point" followed by a unique number, less than or equal to 77_8 is used as a way of distinguishing such states. The ordinal number to the right of the point has no significance except to provide a unique arbitrary label.

STATEMENTS

In ACT there are six different classes of legal statements utilizing this vocabulary. Examples of these classes are shown in Fig. 5. (1) Initialization of the integer variables by parameter assignment where the meaning should be self-evident. Such assignments are limited to the range $-2048 < I < +2048$. (2) Declarations of fixed or variable states in S, U, or V state sets. Thus, writing a declarative statement beginning with WHEN, GIVE, or WHILE followed by a state identifier amounts pictatorially to drawing a box. (3) Transition statements, of which there are three types. Writing one of these transition statements is equivalent to drawing a line away from one state rectangle to another. The six examples of transitions illustrate different features of the vocabulary. Response transitions begin with IF and designate a fixed (R2), variable (R(K)), or analogue $(10 < R < L)$ behavioral output for initiating a change in state. Time transitions are straightforward, with the upper limit being 72 hr, the lower limit being 1 unit (of time--determined by the clock) and variable times (being single precision integer)

```
                    INTEGER ASSIGNMENTS

I = 4
J = 7
N = 7772

                       DECLARATIONS

WHEN S1
GIVE U12.5
WHILE V(J)

                        TRANSITIONS

IF R2 GO TO S44
IF 29 R(K) GO TO V16.8
IF (10<R< L) GO TO S5
  AFTER 2 MIN GO WITH PROBABILITY =3/32 TO S4
  AFTER K UNITS THEN U2
    FOLLOWING J S55.2 THEN GO TO S1.Ø

                  PARAMETER MODIFICATION

WHEN S1  [N = N + 4]
WHEN S1  [J = K & SWITCH REGISTER]

                      DATA RETREIVAL

WHEN S1  [PRINT "EXPERIMENT FINISHED %#]
WHEN S1  [PUNCH 2]

                        RECORD DATA

1  IN S1: R1 LATENCY
2  IN S3: R1 COUNT
3  IN S5.1:R4 SUM
4  IN U(K): S2 COUNT
5  IN V16.7:V16.7 TIME
```

Figure 5   Six classes of legal ACT statements

restricted to 16.7 msec. units specification.

The first time transition (line 4 above under TRANSITIONS) illustrates an additional feature of ACT, namely its ability to describe probabilistic procedures. Probabilities of the form A/B where A must be greater than or equal to B, and B must be a negative power of two, less than or equal to 2^{-7}, may be specified for any transition. Probabilities so specified produce bernoulli distributions of events. In the prototype they are hardware generated by the peaks of a noisy diode counting in a 7-bit shift register.

The third class of transition, shown as the last line under TRANSITIONS in Fig. 5, is a second order transition in which an R or Time transition from any given state to another, at some point in the state diagram, can trigger yet another transition for a different state at a different level or in a different state plane. Second order transitions are essential for communication between state planes, and for making possible the changes in a procedure after a certain number of organism determined events, such as numbers of stimulus presentations of a given sort.

(4) Parameter modification form a fourth group of legal statements. These consitute in ACT I simple three-operant ALGOL assignment statements written immediately after a state declaration and enclosed within square brackets. In ACT I the operators are limited to addition and subtraction ("&" means to add the switch register) and only a single such statement may be written per state declaration.

(5) Data retreival statements shown just lower in this figure are also written at state declaration time. They make it possible to type messages and to retreive various data stored by the five recording directives shown below them. The numerals after PRINT or PUNCH refer to the line numbers of (6) the record statements.

The manner by which these statements are combined into a program is straightforward. The statements are written one statement per teleprinter line, sequentially and in a block form suggestive of ALGOL. A given state is first declared, its integer parameter modifications or data retreival orders enclosed within square brackets, and on subsequent lines all the possible transitions for that state are listed one by one. Another state is then declared and its transitions listed; and so on. State diagrams seem to most easily be constructed and read from left to right; so that the top of the page of ACT statements will generally correspond to the left side of a state diagram. For legibility the transitions of a state are indented one tab stop in from the declaration statement. Nesting of states within states is accomplished by maintaining a one-to-one relation between tabular indentation of the declared state to degree of nesting.

An example of a simple complete program to control the scheduling of a 3-second food reward to a confined pigeon subject for each key peck on a plastic disk only after a minute of non-reward has passed, appears in Fig. 6. The program is initiated by a special non-printing character (WRU). The compiler (in 4K versions loaded by the user, in 8K versions called down by monitor) then prints out an installation identification followed by a new line and the word EXPERIMENT. The user types in a comment identifying his experiment, followed by a carriage return character. The ACT compiler then asks for a number for the station where the experiment is to be carried out. (This number is none other than the W103 device code for the station.) Upon receipt of this number the compiler will respond BUSY if the station is already in use, or if not busy with the approximate number of lines available for program. The user then types any initialization lines, the first declaration (S1), its transition, the next declaration (S1.1), its transition, and the third declaration (S3) and its transition. Any desired comments are preceded by the "/" symbol. Finally a $$ terminates compilation.

Not shown in Fig. 6 are occurrences of any one of 45 compiler diagnostics. These advise the user of syntatical (mispellings, illegal combinations of symbols, spurious characters, and so forth) or semantic (attempts to direct transitions from one state plane to another, or from one level of nesting to another, exceeding the upper limit of probabilities, failing to declare a state that is referred to in a transition line and so forth) errors. Syntactical diagnostics occur immediately upon receipt of the illegal character. Semantic diagnostics occur only at the termination of the current line.

The completion of the program leaves a data structure corresponding to the state diagram resident in core. The prototype system has 2K of core available to accomodate eight independent stations and at any one time, all eight may be occupied with programs of moderate complexity.

In practice a developed program would be stored on paper tape, and entered whenever it was desired to run. In the 4K version once all desired programs are resident, an operating system to execute the data structure is loaded overwriting the compiler, and thenceforth the teletypewriter serves only to print data messages; or if the data rates are low, to punch in coded form the significant events tagged with their relative times of occurrence. There is only one diagnostic at run time: "AVAILABLE UPDATE TIME EXCEEDED" followed by the station last completed in the queue. The example of Fig. 6 servies to illustrate the simplicity and naturalness of the language format for the problems for which it is directed.

An important feature of the language is its ability to incorporate new increasingly sophisticated design features without affecting the format of previous programs. Thus, the obviously desirable ability to have compiler and executive in core together with a monitor to schedule their priorities along with other perhaps unrelated background programs is a feature being developed for 8K systems. Users will type in new programs from one teletype while the central processor continues concurrent control of other experiments. Yet these and further refinements will affect only the power of the language system, not its format.

Conclusions

In summary, the language that has evolved permits multi-access control of up to eight independent laboratory environments. The language includes facilities for modification of experimental parameters which, while simple, are powerful enough to permit the experimenter to produce procedures whose quantitative properties and/or qualitative structures can adjust to subtle on-going characteristics of the input behavior of the subjects. The language provides primitive facilities for retreiving the raw data of the experiment as it is being generated in a form suitable for later input to a conventional off-line data analysis program. An important feature of the language is that its very nature is such as to preclude the possibility of any user overwriting any other user. Thus the language permits trouble-free time sharing of the central processor. Finally, as should be obvious, although ACT was written with the psychologist principally in mind, the language is a very general one. Whenever some aspect of an environment is to be controlled as a function of inputs from other aspects of that environment, as in chemical process control, machine tool control, biomedical monitoring, and other industrial and scientific applications[13], ACT may provide a convenient method of implementing computer control by non-professional personel.

ACKNOWLEDGMENTS

I thank J. D. Keehn, C. D. Webster, D. P. Hendry, W. Lennon, and D. Taylor for numerous suggestions towards the development of the language described.

References

1. Blough, D. The reinforcement of least-frequent interresponse times. J. exp. Anal. Behav., 1966, 9, 581-591.

2. Gill, A. Introduction to the theory of finite-state machines. New York: McGraw-Hill, 1962.

4. Hendry, D. P. and Lennon. W. On-line programming II, in Symposium, Automation of behavioral experiments, Washington D.C., (APA) September, 1967.

5. Husky, H. D. An introduction to procedure oriented languages, in F. L. Alt and M. Rubinoff (Eds.) Advances in Computer, New York: Academic Press, 1964.

6. Knapp, J. Z., Kushner, H. K. and Snapper, A. G. Finite automata: a notation system and methodology for modeling and experimental control. (unpub), 1967.

7. Marlowe, L. A general purpose programming system for the LINC computer. DECUS Biomedical Symposium Procedings, June 1967.

8. Mechner, F. A notation system for the description of behavioral procedures. J. exp. anal. Behav., 1959, 2, 133-150.

9. Millenson, J. R. Principles of behavioral analysis. (Chap. 6) Macmillan: New York, 1967.

10. Millenson, J. R. Maintaining avoidance behavior in the guinea-pig by an automated contingency translator. Alcoholism and Drug Addiction Research Foundation (Toronto) Project No. 168, Substudy, 1967.

11. Millenson, J. R. An automated contingency translator ACT I, A computer system for Process control of psychological experimentation. Psychonom. Bull, 1967, 1, 17.

12. Newman, W. M. A system for interactive graphical programming. Imperial College (London) Computer Technology Group report 67/7, October 1967.

13. Savas, E. S. Computer control of industrial processes. New York: McGraw Hill, 1965.

14. Schoenfeld, W. N., Cumming, W. W. and Hearst, E. On the classification of reinforcement schedules. Proc. Nat. Acad. Sci., 1956, 42, 563-570.

15. Shimp, C. P. Reinforcement of least-frequent sequences of choices. J. exp. anal. Behav., 1967, 10, 57-65.

16. Snapper, A. G., Kadden, R. M., Knapp, J. Z., and Kushner, H. K. A notational system and computer program for behavioral experiments. DECUS Biomedical Procedings, June 1967.

17. Weiss, B., and Laties, V. G. Reinforcement schedule generated by an on-line digital computer. Science, 1965, 148, 658-661.

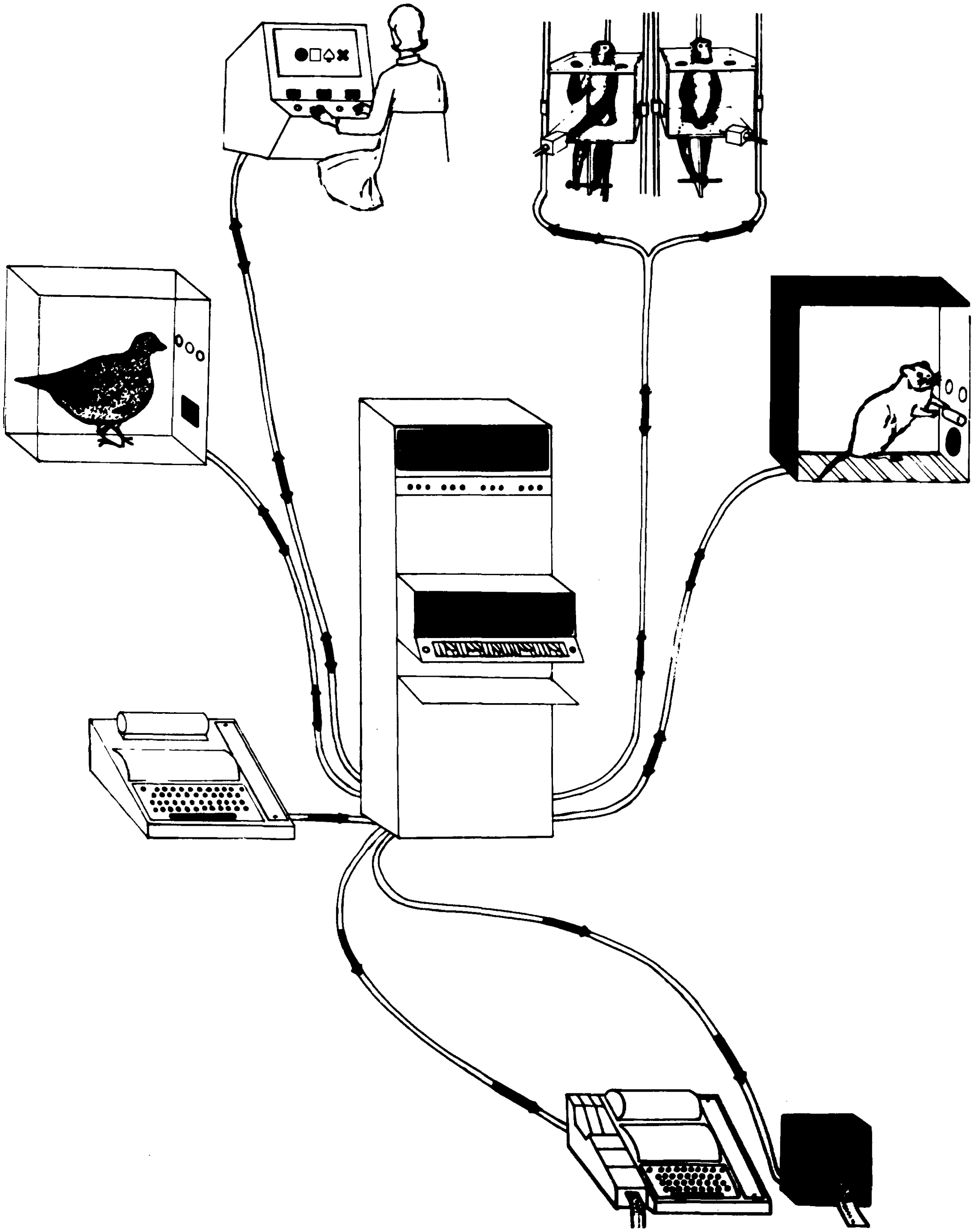


Figure 1   A typical multi-access psychological environment controlled by a rack-mounted PDP 8/S central processor. Four experimental stations, one input teletype (left), an input/output teletypewriter (bottom), and a high speed punch are shown connected to the central processor.

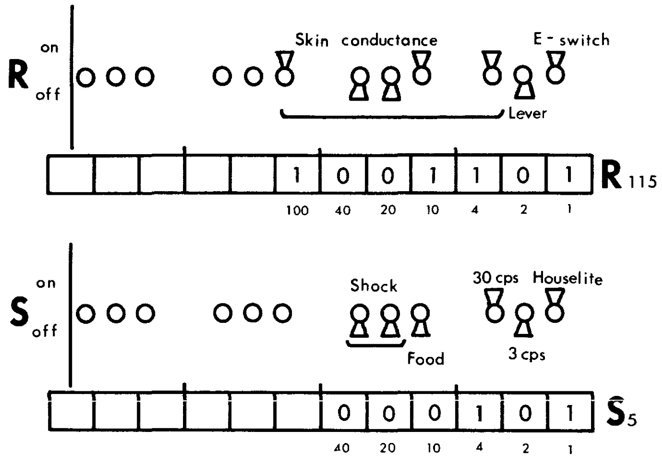


Figure 2 12-bit S-input / R-output words compared for illustrative purposes with two analagous banks of toggle switches. Only 8 bits (switches) of the R-output word and only 6 bits (switches) of the S-input word are in use.

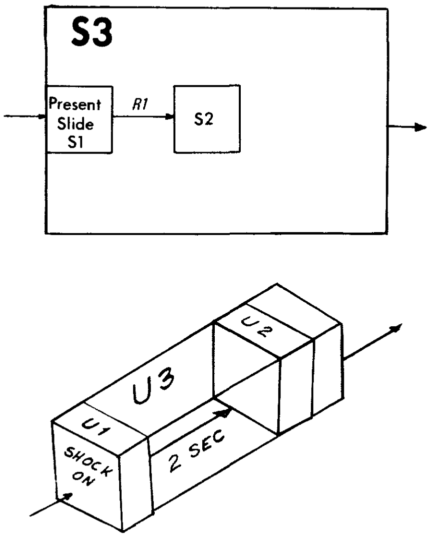


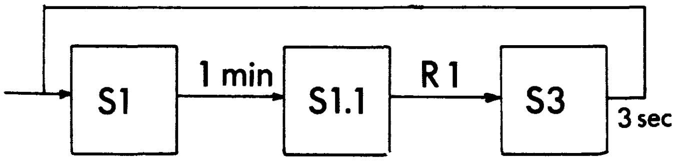
Figure 3 Two nested independent fragments of nested state diagrams for representing the contingencies of behavioral experiments.

```
U.READING PSY PDP8/S ACT I COMPILER
EXPERIMENT   PIGEON FIXED INTERVAL (FI 1')
STATION      5
PROCEDURE    (0077 LINES  AVAILABLE)
 1  /S1=HOUSELIGHTS ONLY      S3=REINFORCEMENT
 1  GIVE S1
 2     AFTER 1 MIN GO TO S1.1
 3  WHEN S1.1
 4     IF R1 GO TO S3
 5  WHEN S3
 6     AFTER 3 SEC GO BACK TO S1
 7     $$

RECORD
 1  /RECORD ROUTINES NOT YETWRITTEN

0071  LINES STILL AVAIL
   PROGRAM ACCEPTED  00004096 MIN  29 APRIL 68
```

Figure 6 A simple program typeout and its associated state diagram. Lines typed by the computer are underlined.



A TAPE STORE FOR BIOCHEMICAL DATA

Harold W. Shipton
University of Iowa
Iowa City, Iowa

ABSTRACT

A means for recording low data rate signals on an inexpensive tape recorder and a program to read these data is described. The technique is especially suited to biochemical applications.

Signals from many biophysical instruments (e.g., amino-acid analysis) are characterized by their low data rate and by the relatively long duration of an experimental run. Traditionally the output from such devices is displayed on a strip-chart recorder having a rebalance time of the order of one second. The data are inherently suited to analysis by digital computer and have been so treated by a number of workers. Operations such as integration and base line drift correction are tedious to perform by hand but are easily implemented on a small scale laboratory computer; a machine such as a LINC can easily handle the output from many photometric devices. The major difficulty in use of computers for this type of application results from the difficulty of entering the data into the machine in an economical and orderly fashion. The LINC, in common with some other machines, is designed for on-line real time operation and if it can be dedicated to data collection, no serious problems of interfacing arise. In most laboratories, however, the computer is in use for other projects and economy dictates some form of intermediate storage for the biochemical data. (In some circumstances the "interrupt" feature can be used. However, other programs running on the machine are not always written in interruptable form; this applies especially to the assembly program used for software development.) Two systems of intermediate storage suggest themselves: punched paper tape or an off-line incremental magnetic recorder. In practice if either of these solutions is used, a separate recorder is necessary for each instrument because there are formidable problems of data organization in any feasible time sharing system.

This paper describes a technique for recording such slowly varying signals on inexpensive tape recorders of the type used as office dictating machines.

There are many methods for transforming low frequency analog data so that it can be recorded as an audio frequency signal; in the most common the data signal modulates the frequency of an audio oscillator. On replay this signal is demodulated and filtered so that the original data are recovered. This method suffers from two major disadvantages. Firstly, if the DC component of the signal is to be recovered accurately, the tape speed must either be maintained within close limits or relatively complicated feedback techniques must be used to compensate for variations in it. Secondly, the tape motion is continuous during recording an extravagance which must be paid for not only in terms of tape consumed but also in terms of replay time.

To overcome these limitations and to achieve a highly reliable but inexpensive store the scheme outlined in Figure 1 has been used. A clock is used to produce pulses at rates compatible with the analog signal data rate (for example in amino-acid analysis this is approximately 1 pulse per 5 seconds). These are accumulated in a time register which can conveniently be a conventional binary counter. Provision is made for manually clearing the counter at the onset of each experimental run.

The analog signal from the strip-chart recorder is converted to digital form by means of an 8 bit shaft encoder with which is associated a storage register (R). Initially, the signal value is loaded into R and at each clock pulse the contents of R are compared with the contents of the shaft encoder. If they match (i.e., there has been no change in the value of the input signal) no further action is taken, but if the contents have changed, the following sequence of events is initiated. The data register is updated to contain the new shaft encoder value. The tape recorder start switch is activated, and a preset delay circuit (one shot) is fired. This delay is long enough to allow tape motion to become well established. At the conclusion of this delay period a scanning oscillator examines the contents of each section of the time and data registers so that a serial 20 bit word appears at the output of the gates. Signal and data bits account for 17 bits; the additional bits are used for control purposes and to separate the time and data portions of the word. During the entire scan period a level is present which serves to control the FSK oscillator (see below). Data are recorded on the tape by frequency shift keying (FSK) an audio generator. At the end of the data block tape motion is stopped and the system remains quiescent until a further change in signal level occurs.

The frequency shift keyer is of an unusual design (Figure 2) in that the audio oscillator runs at $8f_0$ where f_0 is the carrier frequency; this is then divided down by a 3 bit binary counter which can be reset to 0 at the count of 3, 4 or 6 according to which of three gates are enabled. The outputs are further divided by a binary counter so that frequencies of f_0, $\frac{4}{3f_0}$, and $\frac{2}{3f_0}$ are available. These tone pulses are associated respectively with "space", "binary 0," and "binary 1" of the data word. For most small tape recorders f_0 = 5khz is a convenient and easily recorded choice of carrier.

Several variations on the above system have been tried in differing circumstances, but will not be described in detail since they are readily implemented by standard electronic techniques. They include a means for shortening the data word by omitting that portion which contains the signal magnitude and using a single bit to indicate the direction of change. The function is reconstructed in the decoding process by a method akin to Lebesque integration.

The electronic circuits have not been fully described since it is unlikely that others will wish to duplicate them exactly. The system is readily constructed from TTL logic and for those who require further information a set of prints can be obtained from the Bioengineering Resource Facility, University of Iowa, Iowa City, Iowa 52240.

Playback is accomplished by running the tape continuously and feeding the output signal into an analog channel of a LINC computer. This permits high reliability decoding and reconstruction of the original data.

In essence the program which examines the signal measures the frequency of the carrier by counting zero crossings for unit time and from this computes the permissable limits of f_l and f_h. Thus the tape speed need only be constant for the time required to transfer one data word (typically 150 milliseconds) and even in recorders which have no capstan drive this is easily achieved. Since many cycles of audio signal are counted during each data bit an occasional drop out is tolerable. Several versions of the decode program are in use. A flow diagram (Figure 3) of the complete program and of the frequency measuring sub-routine (Figure 4) are appended.

Summary. The system has been found to be a useful substitute for incremental methods when the data rates are low and extensions of the method show promise in several biomedical applications.

It is a pleasure to acknowledge the detail design work of Mr. John W. Emde of the Bioengineering Resource Facility of the University of Iowa. This work was supported by Grant #GM15907 from the National Institutes of Health.

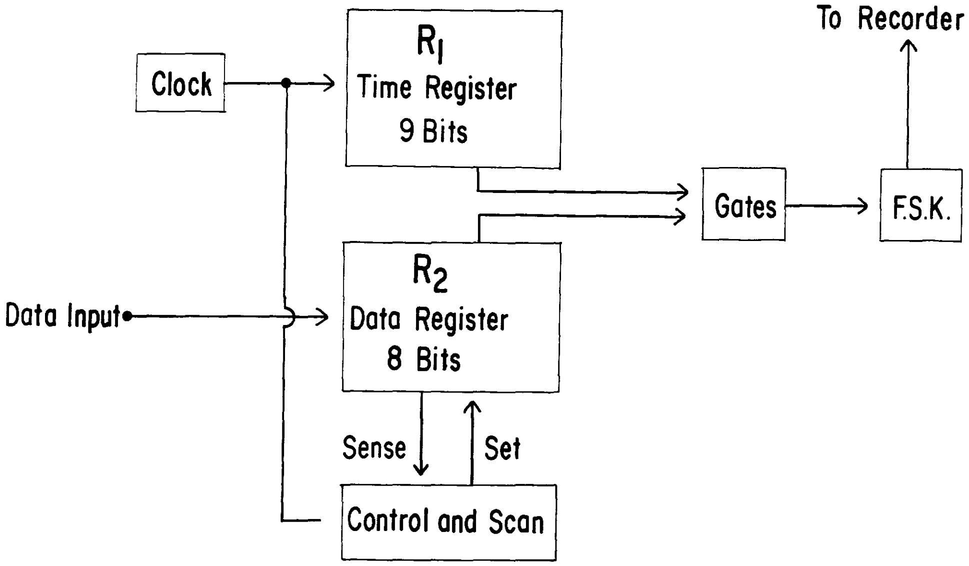


Figure 1 Block Diagram of Basic System

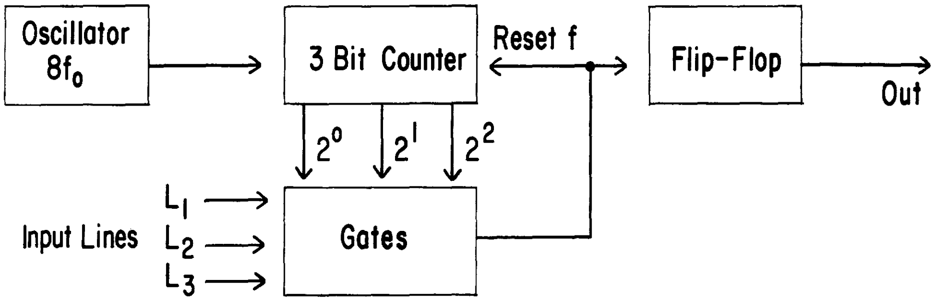


Figure 2 Frequency Shift Keyer

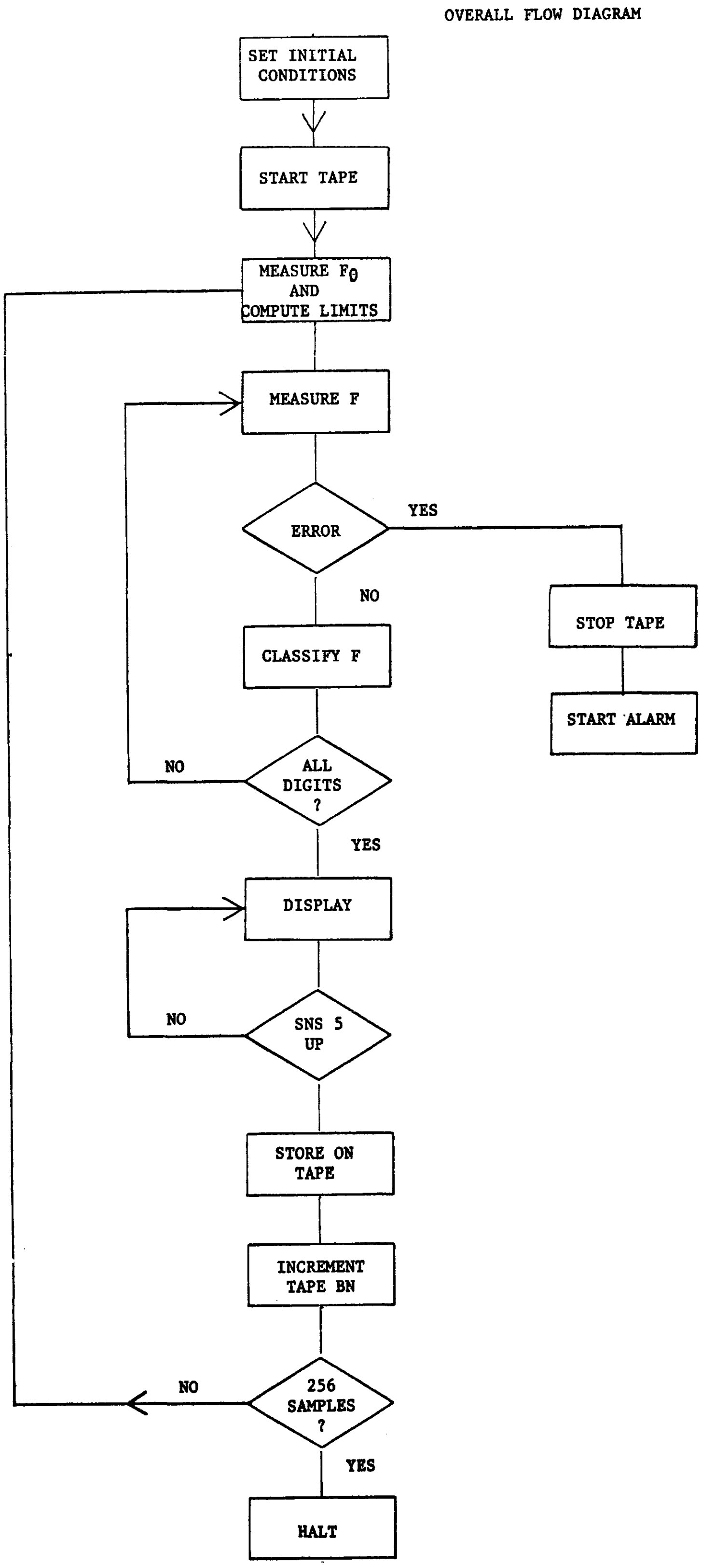


Figure 3 Overall Flow Diagram

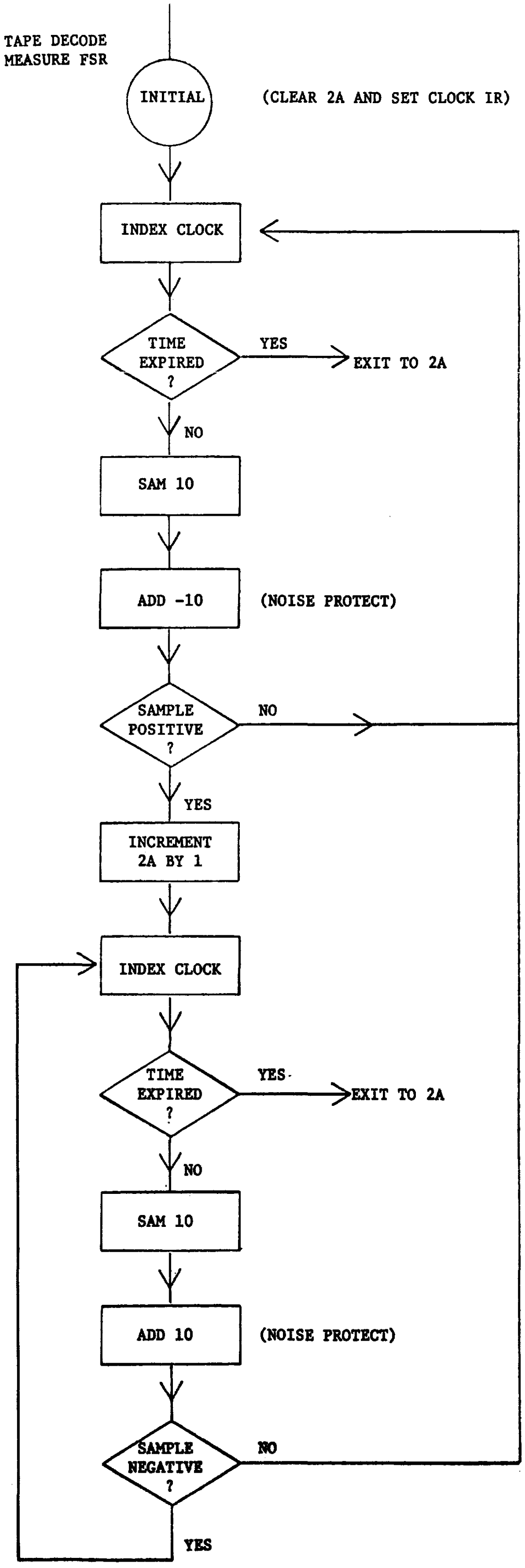


Figure 4 Frequency Measuring Sub-routine

AUTOMATIC SPECTROPHOTOMETRIC IDENTIFICATION OF BACTERIA

Alan Ferry and Richard Martin
QEI
Winchester, Massachusetts

ABSTRACT

Present research at the Office of Naval Research is reported on a project to automatically identify bacteria using only spectrophotometric information. The numbers to be extracted from the measurement are fluorescence to phosphorescence ratio, emission spectrum peak wavelengths, and parameters of phosphorescent decay. A procedure for resolving components of a multiple exponential function is discussed.

This report describes some on-going research into the feasability of extracting enough numbers from luminosity measurements on bacteria to develop an identifying dictionary. In the functioning "automatic bacteria identifyer", a dictionary lookup would be made by the same computer that processes the luminosity data (actually micro-spectrophotometer output), and the type or class of bacteria in the sample would be printed out. With the idea that it would be desirable to some day have a number of these devices available, minimum apparatus configuration has been made a design consideration.

In cases when there is an outer non-membraneous layer to a bacterium, it is usually cellulose. Therefore, most re-emitted light arises from surface membrane lipo-proteins. Photon emission from amino acids, the structural units of proteins, is always from singlet or triplet decays; and, in these experiments, is always from the lowest energy level of an excited state to the ground state. The half-life of a singlet state is of the order of 10^{-15} seconds, and the light emitted by radiative decay from singlet to ground is referred to as fluorescence. Since the half-life of the triplet state is much longer (order of 10^{-3} to 10^{-1} seconds), the associated emission (phosphorescence) continues for a much longer time.

A spectrum obtained under continuous illumination is the sum of fluorescence and phosphorescence. If the illuminating beam is chopped, the recorded light is largely phosphorescence. The ratio of the integral under the fluorescent spectrum to that under the phosphorescent spectrum can then be calculated and becomes one number for the bacterium signature.

Peak positions in the spectrographs are the emission frequencies and will be a set of numbers contributed to the dictionary. We hope to handle the problem of resolving multiple peaks using a technique suggested by Dr. Ian Bush of the Medical College of Virginia. The leading edge of the last peak in a superimposed group would be functionally generated, point by point, from the decaying edge. When the entire peak has been resolved in this way, its position would be determined and the peak would be subtracted from the composite. If the composite were a double peak, the remainder would be a single peak, the first one. If there were more than two, the procedure would be repeated until one peak remained.

To be able to measure the phosphorescent emission at all requires special apparatus consideration. Because of the long lifetime of the excited triplet state, non-radiative transition, especially thermal ones, compete so favorably with photon emission that virtually no light is given off at room temperature. Reduction of the thermal energy in the sample is brought about by making the measurements at low temperature with the culture maintained as a glass rather than as a suspension in liquid media. Liquid nitrogen temperature, 77°K, is being used as standard.

The "glow" in response to a flash of light decays as a multiple exponential of the form $\sum_{i=1}^{n} A_i \epsilon^{-\alpha_i t}$ $n < 10$, and if the decay parameters could be determined, another set of numbers would be available. In keeping with plans for a "minimal configuration", we originally began the project with PDP-8S's. It has become clear, however, these will not be adequate, and we plan to be using 8I's with DECTape. Our procedure for examining the decay information begins with an interrupt trigger from the light source signaling the 8S to begin reading one datum through its A/D converter every millisecond. To implement the data input a 1 kc real time clock was installed. It was desired to have available the inherent stability of a crystal clock, and, since the 8S is too slow to efficiently use the slowest crystal oscillators, it was necessary to include a down-counter network to obtain a 1 kc clock pulse rate. Advantage was taken of the R202 modules necessary for the down-counter to create the following clock instructions:

1. Skip on clock flag set.
2. Clear clock flag and connect to interrupt.
3. Clear clock flag and disconnect from interrupt.

Two alternative averaging techniques are employed for noise reduction. By one procedure, which is technically more difficult but gives better results, identical runs are repeated and the data points are averaged across runs. Another method smoothes a single run by sampling every millisecond and storing the average of n ($4 \leq n \leq 32$) data. That this arithmetic mean is inappropriate for exponential functions is a valid objection, and a geometric mean calculation should be used. This improvement, however, wi&1 require the faster PDP-8I system. In any case, there are stored at this point a thousand numbers representing a multiple exponential function we wish to resolve into components.

An example of a double exponential of the form

$A\epsilon^{-\alpha t} + B\epsilon^{-\beta t}$ is shown in figure 1A.

This function was generated in place of the described data acquisition to eliminate the noise for demonstration. Operations described below for estimating the parameters of one exponential component then stripping it from the multiple are applied by the program as though the data had been sampled from the micro-spectrophotometer. In Figure 2 is shown, indicated by crosses, a semilog plot of the same double exponential function. The program operates on the log transformed data in groups of fifty beginning at the right and proceeding left ten at a time. For each fifty points the best straight line by least squares is calculated. When the last slope calculated differs from the first by a factor greater than m, the process halts and the average slope and intercept of all the calculations in the series are printed. The factor m is a specifiable operating parameter whose best value must be experimentally chosen with consideration for whether it is more desirable to chance missing a component or picking up an extra one. When the slowest decaying component has been estimated (in this example B expβt), shown as a solid line in Figure 2, it is stripped from the composite leaving the other components as a residue as shown in 1B. The least squares calculations are repeated on the log transform of the residue (dots in Figure 2) and another component (broken line in Figure 2) is found. Since the example was a double exponential, the procedure ends. The slope of the semilog plot (log I (t)) of a multiple exponential at any point can be expressed as

$$Y(t) = \frac{\sum_{i=1}^{n} A_i(-\alpha_i)\epsilon^{-\alpha_i t}}{I(t)}$$

and this expression could be used to correct each estimated component after others have been found. We have not found yet, however, that such a refinement is necessary.

We would like to mention two other approaches to the problem of resolving multiple exponentials. A least squares fit to the complete, untransformed,

function $\sum_{i=1}^{n} A_i \epsilon^{-\alpha_i t}$ could be made, but to apply this method conveniently, the number (n) of components should be known in advance.

A different approach, which we have not tried, is based on the fact that the Fourier transform of an exponential function has peak heights that are measures of the exponents.

BIBLIOGRAPHY

1. E. L. Bell and R. Garcia, "Fitting Multi-Component Exponential Decay Curves By Digital Computer", SAM-TR-65-59, August 1965.

2. M. B. Danford, "Some Problems on the Use of Negative Exponential Curves in Biology", SAM-TR-65-4, March 1965.

3. A. B. Callahan and S. M. Pizer, "Applicability of Fourier Transformation Analysis to Biological Compartmental Models", in Patte, Edelsack, Fein, and Callahan, NATURAL AUTOMATA AND USEFUL SIMULATION, Spartan Books, Wash., D.C., 1966.

4. S. V. Konev, "Fluorescence and Phosphorescence of Proteins and Nucelic Acids", (trans. from Russ.), Plenum Press, N.Y., 1967.

ACKNOWLEDGMENT

Supported in part by Office of Naval Research, BSD, Medicine and Dentistry Branch, N0014-68-6-0372, NR304-501. Reproduction in whole or part is permitted for any purpose of the U.S. Government. We would like to thank Dr. K. Hoerman and Dr. A. B. Callahan for their direction and comments.

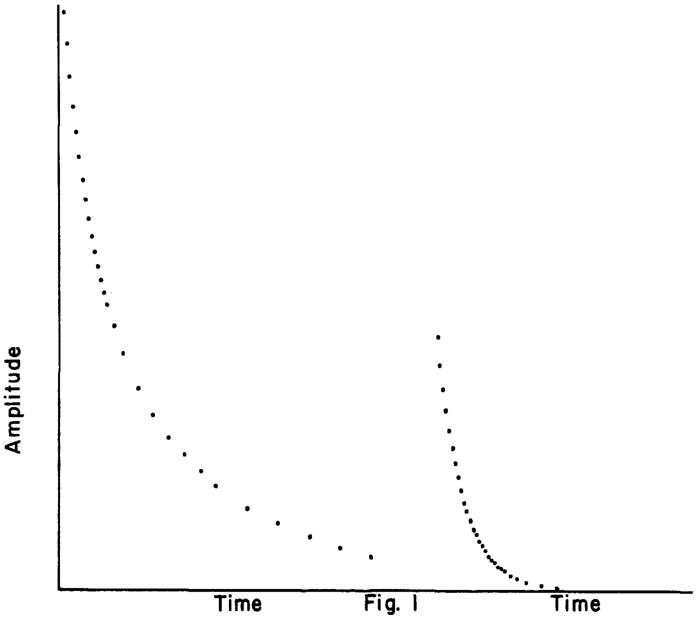


Fig. 1

A. Multiple Exponential B. Residue

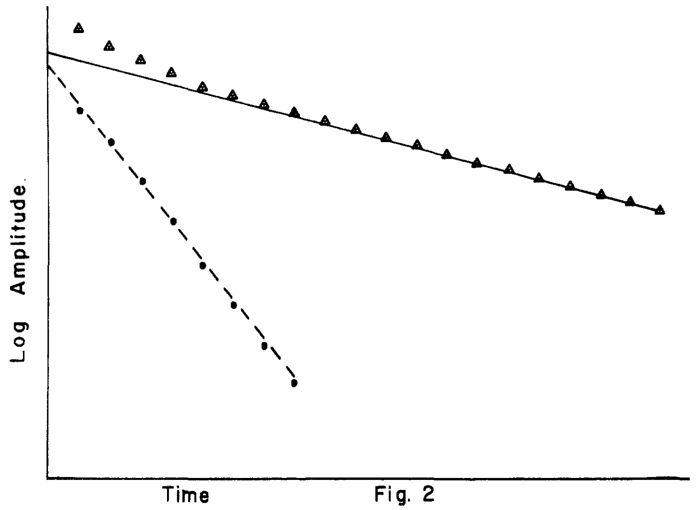


Fig. 2

APPENDIX 1

DECUS SPRING 1968 SYMPOSIUM PROGRAM

Bellevue Stratford Hotel
Philadelphia, Pennsylvania

April 26 and 27, 1968

FRIDAY - APRIL 26

General Session - Terrace Room (Morning)

Chairman: Prof. Philip R. Bevington, Stanford University

8:30-9:45 Registration, 18th floor lobby

10:00 Opening - Prof. Philip R. Bevington

Welcome - Prof. John W. Carr, III, Chairman of the Graduate Group Committee for Computer and Information Sciences, Moore School of Electrical Engineering, University of Pennsylvania

10:30 A STIMULUS-RESPONSE PROGRAM FOR HOTEL ROOM INVENTORIES
David W. Roberts, London, England

The policy aims fulfilled by the program are listed and an outline of the original booking system is given. The difficulties of this system are highlighted and the new system described.

The action of a single function is detailed and the development of a single message is traced through seven phases in response to user requirements. The details of a useful method of handling dated information are given.

The system has been extended to two other hotels.

11:00 THE UCC FASBAC PROJECT
Dan W. Scott, University Computing Company, Technical Services Division, Dallas, Texas

Remote large-volume input and output facilities are available from the University Computing Company UNIVAC 1108 Computer Utility. The FASBAC text editing system now under development will be described. This new conversational system facilitates setting up runs for the UNIVAC 1108 via remote Teletypes in the customers' offices.

12:00 Lunch - Rose Garden

SESSION I (Afternoon)

Chairman: Thomas Day, University of Maryland

1:15 ON-LINE ANALYSIS OF WIRE SPARK CHAMBER DATA
Phylis F. Niccolai and Robert H. Bicker, Carnegie-Mellon University, Nuclear Research Center, Saxonburg, Pennsylvania

A PDP-7 background/foreground mode of time sharing for on-line analysis of wire spark chamber data will be discussed. The background mode analyzes a representative sample of wire spark chamber data in the time available between interrupts from the wire spark chamber logic, an interface between the wire spark chambers and the PDP-7. The foreground mode logs data from the wire spark chamber logic onto magnetic tape. The experimenter is given control of the cyclic operations via the Teletype keyboard and may interrupt the program at any time to retrieve the output from the background mode. The particular analysis required from the background mode must be decided prior to load time and selected from the programs stored on DECtape. This program will be discussed as applied to pion absorption by light nuclei experiment at the Carnegie-Mellon University Nuclear Research Center.

1:45 PDP-8 ON-LINE DATA ACQUISITION SYSTEM FOR HIGH ENERGY PHYSICS
Paul Shrager and Larry Taylor, University of Pennsylvania, Philadelphia, Pennsylvania

This presentation will be a description of an on-line data acquisition system for magneto strictive spark chamber readouts in high energy physics. The system outputs to an incremental magtape unit and CRT display tube. The system includes a real-time clock, high speed paper tape reader, and 24-channel A-D converter for experimental parameter monitoring.

In addition to data acquisition and output data verification, simple on-line analysis is performed, including histograms showing distribution of sparks in chambers. The oscilloscope display includes a reconstruction of the elementary particle event that occurred in the spark chamber.

2:15 THE ON-LINE USE OF PDP-9 AND 360/65 IN A PROTON-PROTON BREMSSTRAHLUNG EXPERIMENT USING WIRE CHAMBERS

D. Reimer, J. V. Jovanovich, J. McKeown, D. Peterson, and J. C. Thompson, Institute for Computer Studies and Physics Department, University of Manitoba, Manitoba, Winnipeg, Canada

The system allowing conversation between two computers (PDP-9 and IBM 360/65) and an on-line analysis of a wire chamber experiment will be described. Wire chambers and scintillation counters are interfaced to the PDP-9 which is connected to the IBM 360/65 via a standard DEC high speed data link. The PDP-9 performs preliminary analyses and selection of detected events. A FORTRAN program residing in a small partition of 360/65 memory completes kinematic analyses of events accepted by the PDP-9, stores the results on magnetic tape, and returns formed histograms to the PDP-9 for visual display or graphical plotting.

2:45 AUTOMATIC FILM MEASUREMENT WITH A PDP-9

C. Drum, T. McGrath, and R. Van Berg, Department of Physics, University of Pennsylvania, Philadelphia, Pennsylvania

The University of Pennsylvania's High Energy Physics Group employs a PDP-9 to control and record digitizings from a Hough-Powell flying spot digitizer. It is not possible with the small computer to perform any on-line analysis or track reconstruction of non-predigitized bubble and spark chamber photographs. Therefore, the system relies on CRT displays and simple checking algorithms for monitoring the quality of the digitizings.

This paper describes the system in general and especially the non-standard software and peripherals.

3:15 Coffee

3:30 A COMPUTER-CONTROLLED SYSTEM FOR AUTOMATICALLY SCANNING AND INTERPRETING PHOTOGRAPHIC SPECTRA*

C. A. Bailey, R. D. Carver, R. A. Thomas, and R. J. Dupzyk, Lawrence Radiation Laboratory, University of California, Livermore, California

In analytical spectrography, the most time-consuming portion of an analysis is the scanning and interpreting of the photographically recorded spectra. A system has been devised to shorten this time considerably by using a small digital computer to control the scanning densitometer and subsequently to calculate abundances from the photographic data.

The following description applies specifically to spark-source mass spectrography; however, adaptation to other systems would be relatively straightforward. A typical photoplate from our spectrograph contains several thousand lines from as many as twenty graded exposures and represents approximately sixty-five elements. Starting with the most intense exposure, the optical transmission of each line is measured using a Grant microphotometer. These transmissions as well as the position of each line are stored in a PDP-8 computer. The computer initiates and completely controls the scanning and simultaneously converts each line position to an exact mass number from a calibration performed at the beginning of the scan. The computer is programmed to distinguish between lines and empty areas on the photoplate, and all the graded exposures of each line are recorded before the scanning continues to the next line. Backgrounds are continuously upgraded and recorded along with their adjacent line densities. After the desired area of the photoplate has been scanned, an emulsion calibration is calculated from the data stored in the computer. Then all line densities on the linear portion of the calibration curve are converted to ionic abundances. Total time involved in scanning twenty exposures on a fifteen inch photoplate is now approximately two hours.

* This work was performed under the auspices of the U.S. Atomic Energy Commission

4:00 A PDP-8 SYSTEM FOR BUBBLE CHAMBER MEASUREMENTS

John Rayner, University of Maryland, College Park, Maryland

This paper describes an on-line measuring system in which the PDP-8 is used both as an up-down scaler for an image plane digitizer and to supervise the measurer in an attempt to prevent the most common measuring errors. This error prevention is accomplished by having the program institute most of the necessary procedures through messages to the measurer on a Teletype and by elementary checking of the input data. Another aim of the system is to replace cards with IBM compatible magnetic tape as the output medium. To this purpose a Digi-Data Stepping Recorder has been interfaced to the PDP-8. It is planned to expand the system to four measuring stations in the future.

4:30 STRIP, A DATA DISPLAY AND ANALYSIS PROGRAM FOR THE PDP-8, 8/I

John Alderman, Georgia Institute of Technology, Atlanta, Georgia

This program, using the PDP-8, high-speed paper tape reader, and Type 34 display, accepts paper tape data listings and displays the result on the display unit. Some elementary computations are made on the data and are also displayed. The program is deliberately designed to be open-ended, and most users will want to add features peculiar to their own problem. Almost all functions are carried out in subroutine form, and these subroutines can be called either from the keyboard or within another subroutine.

6:30 Reception - North Cameo Room

7:30 Dinner - Rose Garden

SESSION II - EDUCATION, NORTH CAMEO ROOM (Afternoon)

Chairman: Dr. Sylvia Charp, Philadelphia Board of Education

1:15 BRINGING THE COMPUTER INTO THE HIGH SCHOOL CLASSROOM
Michael L. Doren and Karl P. Wildermuth,
Deerfield High School, Deerfield, Illinois

This presentation is geared primarily for high school teachers. Our ideas on how the PDP-8/S, in combination with an inexpensive closed-circuit TV setup, can be used to enrich concepts taught in all levels of high school classes will be discussed. Emphasis and discussion will be on the following points:

1. What concepts lend themselves to effective use of the computer.

2. What criteria should be considered in writing a FORTRAN program for use in a classroom demonstration.

3. How to effectively bring the computer physically into the classroom.

a. Slides on our computer laboratory with its closed-circuit TV facilities and two-way intercom to each of our classrooms.

b. Training of student lab assistants to help teachers make these demonstrations.

4. Discussion of strengths and weaknesses observed.

1:40 PDP-8/S IN THE HIGH SCHOOL CLASSROOM
Bud Pembroke and Dave Gilette, Computer
Instruction NETWORK, Salem, Oregon

The presentation will cover the present use of the PDP-8/S as a portable computer in several curricular areas in schools within the Computer Instruction NETWORK. The use of machine language will be discussed along with the use of CINIC as a "Load and Go" conversational compiler. CINIC "Computer Instruction NETWORK Instructional Compiler" was patterned after a subset of BASIC for the 4K core memory of the PDP-8/S. The authors will include a description of the instructions, examples of programs, and a candid explanation of advantages and limitations of this language.

2:15 PROJECT ASC - THE SMALL COMPUTER IN EDUCATION
Richard R. Karash, Richard L. Mazer,
Robert M. Metcalfe, and Clyde E. Rettig, Jr.,
MSC Associates, Boston, Massachusetts

ASC is a research project conducting investigations into applications of the small computer made possible jointly by Digital Equipment Corporation and the Massachusetts Institute of Technology. The project ASC computer (PDP-8/S) has been made available along with complete technical assistance to a number of people at M.I.T. involved in research, education, and administration. Some interesting results on the effect of this computer's availability in an educational environment, suggestions on how to optimize the benefits of such computers' services, and a few hints to improving the computers themselves for future educational applications have been collected and will be presented.

2:40 CLOSING THE EDUCATIONAL LOOP IN APPLIED MATHEMATICS
Dr. John Elder, Department of Applied
Mathematics and Theoretical Physics,
University of Cambridge, Cambridge, England

Lectures supported by demonstration have an immediate impact on students impossible with chalk and blackboard alone. In essentially conceptual areas of knowledge, such as applied mathematics, demonstrations are often impossible and the cumbersome input/output procedure of note taking and understanding after much midnight oil and personal supervision is inevitable. The educational loop can be closed right in the classroom in the following way. The lecturer is provided with a control box on which are some knobs and switches connected to a computer (housed elsewhere) and a closed circuit TV monitor(s), the camera of which is watching the computer display screen. Parameters are entered from the knobs and tasks initiated from the switches and the results are displayed in graphical form. The lecturer had continuous control over his problem parameters, and may choose settings arising from discussion in the class. Typical problems involve systems of ordinary or partial differential equations. Separate "workshops" which simulate the equivalent of a physicists laboratory session reinforce the lecture material and provide the student with an opportunity to use his initiative.

An "experimental" hybrid computer system incorporating a PDP-8, currently in use in the DAMTP, Cambridge, will be described and illustrated in a movie. A system using a PDP-9 is now being designed.

3:05 Coffee

3:15 A SYSTEM FOR PRESENTING PROGRAMMED INSTRUCTION TO THE DEAF AND HEARING IMPAIRED
K. E. Rigg and James A. Boehm, III, New Mexico State University, Department of Speech, Las Cruces, New Mexico

A digital system for presenting programmed instruction of language concepts to hearing impaired and deaf children will be discussed. The system presents controlled visual and auditory stimuli to the learner, requiring either a matching-to-sample response with four solutions or the solution of a straight four choice task. The system reinforces correct responses with a variety of visual, auditory, and primary reinforcers including pulsed pure tones, colored lights, tokens and candies.

This system is complete in that it includes the basic teaching unit, its own instrumentation, data reduction, and provisions for making programs.

3:35 A COMPUTER SYSTEM FOR ELECTRICAL ENGINEERS
Dr. David M. Robinson, Department of Electrical Engineering, University of Delaware, Newark, Delaware

Educational computer applications usually center on the problem solving capabilities of general-purpose machines. The electrical engineer is peculiar in that he must become more deeply involved in the computational system than is suggested by this casual use. His concern arises by virtue of his responsibility for the conception and design of the computer itself and for its hardware adaptation to a variety of applications. A system has been evolved which is functionally directed at the problems generated by the realization of computers or computer-like systems. This system is described and a number of typical student problems discussed. The problem examples chosen illustrate the range of levels which may be encompassed using the system, the versatility of the system and problems which may be of some general interest.

4:00 EDUCATION SUBGROUP WORKSHOP
Chairman - Mrs. Judith B. Edwards, Director, Computer Instruction NETWORK, Salem, Oregon

Welcome and Introduction of Participants
Short (10 minute) Mini-Papers Presented by Education Users

1. "The Computer and Pomfret" by William Hrasky
2. "The Computer and Teacher In-Service" by Bud Pembroke
3. Plus additional papers (to be announced)

Organizational Structure of the Education Subgroup
Round Table Discussion:

Topics: Curricular applications of small computers in education
The computer in the junior high school
What concepts should be taught?
Teacher training
Language Levels for instruction
Vocational training programs

6:30 Reception - North Cameo Room

7:30 Dinner - Rose Garden

SESSION III - TYPESETTING AND MODULES WORKSHOP, NORTH LOUNGE (Afternoon)

Chairman: Richard J. McQuillin, Inforonics, Inc.

1:15 COMPUTER TYPESETTING OF MATHEMATICAL TEXT: THE INPUT LANGUAGE PROBLEM
Richard J. McQuillin, Inforonics, Inc., Cambridge, Massachusetts

This paper presents results of some research in computer typesetting of mathematical text. In particular, attention is given to the representation of complex symbolism using a conventional keyboard. Emphasis is on how keying conventions could be established to provide an input system that is usable by the editorial staff of a publisher of such articles.

Experimental results are given based on a test sample using these conventions. The results show how the system can be utilized to computer typeset Mathematical Reviews.

An extension of the symbol representation scheme is presented, whereby complex two-dimensional mathematical expressions may be expressed in functional notation. This would lead to a typesetting language for handling complex typography at the input keyboard.

1:45 USE OF PDP-8 FOR DRIVING PHOTO-COMPOSITION MACHINES
Richard Falt, Digital Equipment Corporation, Maynard, Massachusetts

A brief look at the use of Digital's DECtape and DECtape Disc Systems to produce punched paper tape input to various photographic units.

2:45 Discussion Period

3:15 Coffee

3:30 MODULES WORKSHOP

Discussion on Computer Interfacing Techniques with "M" and " K" Series Modules by a Representative of Digital Equipment Corporation.

6:30 Reception - North Cameo Room

7:30 Dinner - Rose Garden

SATURDAY - APRIL 27

SESSION I - PDP-9 SOFTWARE WORKSHOP, TERRACE ROOM (Morning)

Chairman: Prof. Philip R. Bevington, Stanford University

10:00 PDP-9T TIME SHARING SOFTWARE -
PHASE I: MULTIPROGRAMMING
D. M. Forsyth, Psychology Department,
Harvard University, Cambridge, Massachusetts,
and M. M. Taylor and S. Forshaw, Defence
Research Establishment Toronto, Toronto,
Ontario, Canada

The PDP-9T is a PDP-9 with the addition of paging hardware, special traps, and modifications which translate IOT instructions into specific calls to the system monitor. Phase I of software development for the system permits several processes to occupy core simultaneously. All input/output is handled by the system monitor. Real-time tasks have high priority and are generally interrupt driven, e.g., are activated as necessary to process data. Background tasks such as editing, assembling and debugging are allocated time by an algorithm which seeks to keep constant the product of invocation rate and time quantum. The minimal system requires a PDP-9T with 16K of memory and 4 DECtapes.

10:30 EXTENDED MEMORY FORTRAN WITH AN
8K PDP-7*
Philip R. Bevington, Department of Physics,
Stanford University, Stanford, California

A hardware modification to the PDP-7 and a FORTRAN subroutine are described which permit the use of Extended Memory coding in FORTRAN II with an 8K memory PDP-7. Normally, this coding permits the storage of large data arrays outside the basic 8K of memory which contains the program and the Operating Time System. In the present system the extra memory is supplied by DECtape. A scratch pad consisting of several pages of 256 words each is retained within the basic 8K memory so that access to the DECtapes is relegated to transfers of blocks. Interpretation of extended memory addresses is accomplished by trapping indirect addresses outside of basic memory and using software to modify these addresses. Such a system permits the use of larger arrays for data manipulation at the expense of time required for DECtape handling. In most cases, however, improved techniques of manipulation through the use of larger and more arrays more than offsets this expenditure of time. The philosophy of design and the relative advantages and disadvantages of such a system are discussed.

*Supported by National Science Foundation

11:00 IMPLEMENTATION OF AN ON-LINE
REACTIVE (TYPEWRITER) LANGUAGE
David Z. Polack, University Computing
Company, Dallas, Texas

The language processor to be discussed is designed for use via reactive typewriter. It accepts, names, stores and manipulates character strings which may be used as names, data and/or procedure. List processing techniques are utilized in the processor implementation.

The presentation is in the form of a tutorial session, which first places the language processor within the framework of the University Computing Company's FASBAC System. Subsequent discussion will include:

1. A brief description of the language for those unacquainted with it.

2. Discussion of memory allocation in terms of the necessary coding, strings, stacks, vectors, communication zones, etc.

3. The methodology of handling various strings.

4. Dynamic "Garbage Collect".

5. Special handling of defined primitives.

6. Additional primitives not included in previous literature.

7. Discussion Period

Reference may be made to: TRAC, A Procedure-Describing Language for the Reactive Typewriter; Calvin N. Mooers; Communications of the ACM, Volume 9/Number 3/March, 1966.

12:00 Lunch - Rose Garden

SESSION I (Afternoon)

1:15 PDP-9 MONITOR SYSTEM WORKSHOP
David Leny and James Murphy, Digital
Equipment Corporation, Maynard, Massachusetts

This lecture, informal discussion period and demonstration is directed towards the design philosophy of the PDP-9 ADVANCED Software Monitor System which centers on user convenience and optimum hardware utilization.

The sub-topics will be:

1. The comprehensive, device independent, input/output programming system which includes handlers for all the standard peripheral devices.

2. The expansion and specialization capabilities of the system to utilize all central processor and standard or non-standard peripheral options.

3. The keyboard control for automatic storage, retrieval, loading execution of all systems and user programs.

4. Complete error analysis at monitor, input/output and system program levels.

5. Background/Foreground (two user time sharing) Operating System.

It is advised that the attendees prepare for this workshop by reading the Monitors Manual (DEC-9A-MABØ-D) of the ADVANCED Software System. Copies may be obtained by contacting your local DEC Sales Office.

3:30 A REAL-TIME, MULTIPLE TASK EXECUTIVE PROGRAM WITH A BUILT-IN CONSOLE UTILITY PACKAGE
C. P. Martin, Jr., and R. L. Simpson, Oak Ridge National Laboratory, Oak Ridge, Tennessee

An Executive Routine has been developed for the PDP-8/S and PDP-8 computers which schedules process control tasks in real time and establishes operating priorities. The program including the utility package) occupies about one-third of a 4096 word memory block and accommodates eight major control tasks. The only hardware addition of the standard computer configuration required by the Executive is a real-time interrupt.

3:45 FOCAL
Rick Merrill, Digital Equipment Corporation, Maynard, Massachusetts

A new small computer language called FOCAL has been designed and written at DEC to be used in Formulating On-Line Calculations in Algebraic Language. This paper is a discussion of how size (3K), power (14 functions), and flexibility (several options) were achieved in designing an easy-to-use language and in programming it for the PDP-8 family of computers.

SESSION II - PDP-8 SOFTWARE WORKSHOP, NORTH CAMEO ROOM (Morning)

Chairman: Michael S. Wolfberg, Moore School of Electrical Engineering, University of Pennsylvania

10:00 PDP-8 (DISK) OPERATING SYSTEM
Charles Conley, Digital Equipment Corporation, Maynard, Massachusetts

This lecture and discussion session is devoted to a presentation of the design philosophy of the PDP-8 Disk software. The primary features exhibited are ease of use, increased thru-put and user liberation from operator panel switch dependency.

The following topics will be discussed:

1. The philosophy behind the monitor development and the benefits to the user.

2. The user monitor commands and internal structure of the monitor, including the core requirements, limitations, extensions, and I/O device handling.

3. The standard system programs attached to the Disk system, both for 4K memory and extended memory. A complete discussion will be given describing the way programs are saved on the Disk, the general usage of the Disk as a program storage and data file storage device.

It is advised that the attendees prepare for the workshop by reading the PDP-8 Disk Software (Basic) Manual (DEC-08-SBAB-D). Copies may be obtained by contacting the local DEC Sales Office.

12:00 Lunch - Rose Garden

SESSION II (Afternoon)

1:15 DISC VERSION OF STRIP - A DATA DISPLAY AND ANALYSIS PROGRAM FOR THE PDP-8, 8/I
John Alderman, Georgia Institute of Technology, Atlanta, Georgia

A version of STRIP has been developed to take advantage of the storage capabilities of the DF32 Disc Storage Unit. Techniques of overlay generation and calling, data storage and retrieval, and programming philosophy for open-ended programs to be modified by unskilled users are described.

1:45 PDP-8 OSCILLOSCOPE DISPLAY OF MATHEMATICAL FUNCTIONS USING FORTRAN
A. E. Sapega and S. G. Wellcome, Trinity College, Hartford, Connecticut

A general-purpose program for oscilloscope display of mathematical functions will be described. Since the main program is written in FORTRAN the user need only insert the FORTRAN statement of his function in a standard location. At object time he specifies the range of the independent variable. Following a scaling computation, the scaled function is computed and a table of values generated. These are displayed on an oscilloscope by means of a binary program which is loaded at FORTRAN object time. Interactive features allow the user to re-specify the range of the independent variable to more closely examine the various ranges of the function under study.

The system described uses a PDP-8 with 4K core and a type 34D oscilloscope display unit.

2:15 A DISPLAY PROGRAMMING LANGUAGE (DPL)
Jeffrey H. Kulick, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pennsylvania

This paper presents a description of a programming language for the PDP-8/338 known as Display Programming Language (DPL) and illustrates its use. This language has been implemented by the author. DPL allows the definition of simple data structures such as points and lines and the definition of arbitrarily complex structures called Displaygroups. A class of set operators (FOR A ∈ B DO . . .) allows the user to selectively traverse a data structure. As part of the definitional language, a computational facility is available which allows the definition of structures algorithmically.

DPL operates in two modes. The first, an interactive mode, allows the user to define, display, and modify structures from a Teletype console. The second mode, known as stored program, allows the user to define a sequence of DPL commands and then execute them as a program. Decision and recursive call statements are available when operating in the stored program mode.

2:45 THE FASBAC PROJECT-TIME DIVISION-MULTIPLEXING FOR THE PDP-8
George E. Friend and Paul J. Bell, Technical Services Division, University Computing Company, Dallas, Texas

Hardware modifications and software techniques for the efficient utilization of the 680 Data Communications System as a low-speed-line multiplexor for the University Computing Company's FASBAC Remote Access System are described.

3:15 Coffee

3:30 Discussion Session

"Use of Large Computers for the Assembly of PDP-8 Programs"

SESSION III - BIOMEDICAL, NORTH LOUNGE (Morning)

Chairman: Prof. Belmont Farley, Johnson Foundation, University of Pennsylvania

10:00 A PDP-6 LANGUAGE FOR SIMULATING COMPLICATED BIOCHEMICAL SYSTEMS
David Garfinkel, Johnson Research Foundation, University of Pennsylvania, Philadelphia, Pennsylvania

A language for simulating biochemical systems composed of complex sets of chemical reactions is described. This is written in FORTRAN IV; a machine-independent version of it has been prepared, but is appreciably more powerful when set up for on-line interaction, which is presently done with a PDP-6 including card reader, printer, and scope display. The input is in the form of chemical reactions and associated numbers, on cards; output in tabular and graphical form. The principal mathematical operation is the solution of differential equations representing the time behavior of the chemical concentrations, but alternative mathematical treatments are being added. A number of applications of this language will be described.

10:30 A GENERAL LANGUAGE FOR ON-LINE CONTROL OF PSYCHOLOGICAL EXPERIMENTATION
J. R. Millenson, Department of Psychology, University of Reading, Reading, England

A problem-oriented language is being developed for on-line process control of psychological experimentation. The language consists of nested blocks of simple English statements familiar to every experimental psychologist. The function of this language is to produce an Automated Contingency Translator (ACT) which samples and updates a number of independent time-shared experimental environments 60 times a second. Experimental procedures are mapped by the ACT compiler from the English statements into a probabilistic finite state network in list structure format. An independent operating system (which in the 4K version overwrites the compiler) then executes the list structure automata: i.e., runs the experiments, records and retrieves data and admits low priority background programs in any available dead time.

11:00 TAPE STORE FOR BIOCHEMICAL DATA
Harold W. Shipton, College of Medicine, University of Iowa, Iowa City, Iowa

Signals from many biophysical instruments (e.g., gas chromatographs) are characterized by their low data rate and by the long duration of an experimental run. Operations such as integration and base line drift correction are tedious to perform by hand but are easily implemented on a small scale laboratory computer; a machine such as a LINC can easily handle the output from many photometric devices. The LINC is designed for on-line real-time operation, and if it can be dedicated to data collection, no serious problems of interfacing arise. In most laboratories, however, the computer is in use for other projects and economy dictates some form of intermediate storage such as punched paper tape for the biochemical data.

In the system to be described an entertainment or office tape recorder is used. The analog signal (y) is continuously compared with its previous value and when a significant change is detected, tape motion begins and an arbitrary time code is written on the tape. From the time values and direction of change $y = f(t)$ is reconstructed by a simple LINC program when the tape is entered off line to the computer. The program is written to that variations in tape speed or recording level have a minimum chance of introducing errors.

The system is simple, inexpensive and reliable enough to be used in routine laboratory applications.

11:30 BIOPHOSPHORESCENT ANALYSIS
Alan Ferry and Richard Martin, Q.E.I., Computer and Information Systems, Winchester, Massachusetts

If ultraviolet light is flashed on a bacterial plaque at 77° Kelvin, an optical triplet state of the ring membered ammino acids is induced and the organisms phosphoresce. Since the functional time decay of the phosphorescence is a sum of exponentials, bacterial samples may be characterized by parameters (coefficients and exponents) of the decay. Computer programs have been written to reduce noise in microspectrophotometer output sampled by the PDP-8/S and to obtain the exponential parameters by successively stripping off each component. This research and development is being supported by the Office of Naval Research, BSD, Medicine and Dentistry Branch.

12:00 Lunch - Rose Garden

SESSION III (Afternoon)

1:15 CLINICAL LABORATORY AUTOMATION PANEL DISCUSSION

Topics:

1. The benefits to be derived from Clinical Chemistry Automation.

a. With respect to improved chemistry and quality control.

b. With respect to decreasing laboratory costs, eliminating clerical errors, providing faster service, and better organization of the reported test results.

2. The dedicated laboratory computer versus the general purpose hospital data processing system for clinical laboratory automation.

3. What are the capabilities of the ideal system for total automation in the clinical laboratory?

4. How should one implement such a system?

Panel Members:

Dr. G. Phillip Hicks (Panel Leader) - University of Wisconsin

Dr. Ralph Thiers - Duke University

Dr. M. A. Evenson - University of Wisconsin

Prof. William Wattenburg - Berkeley Scientific Laboratories

Dr. Hugo Pribor - Perth Amboy

SESSION IV, PINK ROOM (Morning and Afternoon)

10:00 Discussion on developments in multiprocessor PDP-10's. Hardware configurations and programming methods will be discussed and analyzed.

12:00 Lunch - Rose Garden

1:15 PDP-6/10 SOFTWARE WORKSHOP

Chairman: David R. Friesen, Digital Equipment Corporation

Discussion of systems' software and new systems planned. A significant new PDP-10 product announcement will be made.

DEMONSTRATIONS

The following equipment will be available for demonstrations. Specific details on time, etc. will be posted at the registration desk on Friday morning.

1. PDP-8 and DEC-338
Moore School of Electrical Engineering

Use of the PDP-8 as a satellite to the IBM 7040.
Applications and systems programs for the DEC-338 Programmed Buffered Display.
Various display demonstrations on the DEC-338.

2. PDP-9
Physics Department, David Rittenhouse Laboratory

Control of Hough-Powell On-Line Flying Spot Digitizer

3. PDP-6
Johnson Foundation, Richards Building

Time-Shared System, Biomedical Applications

APPENDIX 2

AUTHOR/SPEAKER INDEX

APPENDIX 3

ATTENDANCE

Mr. James Richard Dowell
Digital Equipment Corporation

Mr. John Allen Jones
Digital Equipment Corporation

Mr. W. Meier
Applied Logic Corporation

Mr. Fredric M. Strange
Lawrence Radiation Laboratory
University of California

Dr. Bruce J. Biavati
Columbia University

Mr. William B. Bell
Columbia University

Mr. Howard Moraff
Cornell University

Mr. Erich R. Knobil
Cornell University

Mr. Steven L. Bard
U.S.A. Nuclear Defense Laboratory

Miss Ann G. Duffy
South Windsor High School

Dr. Dewey Johnson, Jr.
Equitable Life Assurance Society

Mr. James R. Masek
Electro-Optical Systems, Inc.

Dr. Mark G. Pfeiffer
La Salle College

Mr. A. J. D'Eustachio
DuPont

Mr. A. R. Briggs
E.I. du Pont Company

Mr. Donald R. Johnson
DuPont

Ms. Linda J. Shaffer
University of Pennsylvania

Mr. Norman Zamcheck
Leary Laboratory and Harvard
Medical School

Mr. Gregory G. Vereos
Perth Amboy General Hospital

Mr. John K. Moore
Cornell Medical Community
Computer Project

Dr. Roy W. Bonsnes
Cornell University Medical College

Mr. Christopher L. Cross
Carnegie-Mellon University

Dr. Henry P. Schwarz
Philadelphia General Hospital

Mr. J. N. Burns
Becton-Dickinson and Company

Mr. William Kirkham
Perth Amboy General Hospital

Mr. Norman A. Thetford
Perth Amboy General Hospital

Mr. James D. Hamm
Perth Amboy General Hospital

Miss Sylvia Blatt
N.Y.C. Department of Health

Mr. James H. Birgie
Smith Kline and French

Mr. Roy S. Taylor
Department of Defense

Mr. Richard C. Hewitt
Bell Telephone Laboratories

Mr. Howard C. Johnson
Bell Telephone Laboratories

Mr. Jeffrey H. Kulick
Moore School of Electrical Engineering
University of Pennsylvania

Mr. Robb N. Russell
Moore School of Electrical Engineering
University of Pennsylvania

Mr. Richard N. Freedman
Massachusetts Institute of Technology

Mr. Robert Berger
Bell Telephone Laboratories

Mr. Jack Harvey
Communications and Systems

Miss Maxine Paulsen
DHEW-PHS-BDPEC-NCCDC-
Heart Disease and Stroke Control Program

Mr. Ronald E. Medei
Western Electric Company, Inc.

Mr. Raymond H. Booth
Western Electric Company, Inc.

Mr. H. N. Longenbach
Western Electric Company, Inc.

Mr. Dale Hurliman
Princeton University

Mr. Paul E. Andree
Eastman Kodak Company

Mr. Alfred D. Ford
Department of Defense

Mr. Charles W. Blomquist
Oceanics, Inc.

Mr. Lewis A. Maylath
Smith Kline and French

Mr. James D. McFadden
Williamson Trade School

Mr. Michael L. Doren
Deerfield High School

Mr. Bud R. Pembroke
Computer Instruction NETWORK

Mr. William N. King
Digital Equipment Corporation

Prof. Thomas B. Day
University of Maryland

Miss Martha Sifnas
Digital Equipment Corporation

Mrs. Joan K. Fine
Digital Equipment Corporation

Ms. Jean J. Bartik
Auerbach Computer Corporation

Mr. Walter C. Janney
Arthur E. Spellissy and Associates, Inc.

Dr. Robert G. Glasser
University of Maryland

Mr. Robert R. Bishop
Digital Equipment Corporation

Mr. Lawrence Portner
Digital Equipment Corporation

Mr. Dave Friesen
Digital Equipment Corporation

Mr. William G. McNamara
Digital Equipment Corporation

Mr. John B. Wyatt, Jr.
Digital Equipment Corporation

Mr. Stanley Joseph Goliaszewski, Jr.
La Salle College

Mr. David Garfinkel
University of Pennsylvania

Mr. Richard Van Berg
University of Pennsylvania

Dr. Engelbert Ziegler
Max-Planck-Institut, W. Germany

Mr. Stephen F. Jackson
United Aircraft Corporation

Mr. Richard J. McQuillin
Inforonics, Inc.

Mr. Stanley E. Forshaw
Defence Research Establishment Toronto

Mr. Allen F. Stormont
Bell Telephone Laboratories

Mr. William C. Menke
Digital Equipment Corporation

Mr. David E. Leney
Digital Equipment Corporation

Mr. Charles H. Conley
Digital Equipment Corporation

Mr. John Margolf
Columbia University

Mr. Douglas A. Kent
Lawrence Radiation Laboratory
University of California

Mr. Robert Sundberg
Massachusetts Institute of Technology

Mr. Arthur M. Kray
Lawrence Radiation Laboratory
University of California

Mr. Louis J. Bartscher
Minneapolis-St. Paul Sanitary District

Mr. Richard L. Curtis
Aluminum Company of America

Mr. Ralph L. Bailey
Western Electric Company

Mr. Alan Ferry
Q.E.I.

Prof. Philip R. Bevington
Stanford University

Mr. Stephen G. Wellcome
Trinity College

Mr. Anthony J. Palmieri
Alphanumeric, Inc.

Dr. W. H. Highleyman
DATA TRENDS, Inc.

Mr. John B. Locke
Rutgers University

Mr. Michael S. Wolfberg
Moore School of Electrical Engineering
University of Pennsylvania

Mr. Charles Drum
University of Pennsylvania

Mr. Thomas McGrath
University of Pennsylvania

Mr. Orin C. Hansen, Jr.
Yale University

Mr. Peter W. Lucas
Yale University

Mr. Frank Hugh Byers
Mass. Eye and Ear Infirmary

Mr. Thomas J. Foley
U.S. Naval Air Development Center

Mrs. Phylis F. Niccolai
Carnegie-Mellon University

Mr. Max J. Lanzendorfer
Graphic Services, Inc.

Mr. Royal M. Gibson
Graphic Services, Inc.

Mr. Joseph Massimo
Brown University

Mr. E. W. White
Bell Telephone Laboratories

Mr. David M. Robinson
University of Delaware

Mr. R. E. Hummer
University of Maryland

Mr. Talbot Baker
Taft School

Mr. Anatole M. Shapiro
Brown University

Major Robert A. Leach
U.S. Military Academy

Mr. Michael Rossin
On-Line Systems, Inc.

Mr. William T. Lyon
Aluminum Company of America

Mr. John F. O'Donnell, Jr.
Time, Inc.

Mr. Paul M. Skern
Sweda International

Ms. Sue Kietzer
Whirlpool Corporation

Mr. John Van Drasek
Whirlpool Corporation

Mr. Redmond Sage
Whirlpool Corporation

Mr. Harv Leland
Whirlpool Corporation

Mr. Tom Allen
Whirlpool Corporation

Mr. Milton W. Petruk
University of Alberta, Canada

Mr. J. D. Dyment
Digital Equipment of Canada

Mr. G. David Singer
Night Vision Laboratory

Ms. Jean C. Rigg
New Mexico State University

Mr. David Z. Polack
University Computing Company

Mr. R. L. Simpson
Union Carbide Corporation

Mr. John Alderman
Georgia Institute of Technology

Mr. J. R. Millenson
University of Reading, England

Mr. Robert Edwin Hill
Geotec, Inc.

Mr. Wayne T. Patrick
Herald Publishing Company

Mr. Richard L. Mather
The Upjohn Company

Mr. Edward B. Corell
University Hospital, Michigan

Mr. Stephen F. Carlson
Minneapolis-St. Paul Sanitary District

Mr. James A. Field
University of Waterloo, Canada

Mr. James L. Downs
Badger Meter Manufacturing Company

Mr. James J. Murphy, Jr.
Digital Equipment Corporation

Mr. Roger A. Due
Naval Ammunition Depot

Mr. D. A. Dalby
Bedford Institute

Mr. Thomas M. Elliott
Digital Equipment Corporation

Mr. R. LaFrance
Canadian Research and Development

Mrs. Angela J. Cossette
Digital Equipment Corporation

Mrs. Jeanne M. Gabrish
Digital Equipment Corporation

Mrs. Rita M. Fryatt
Digital Equipment Corporation

Mr. J. G McHardy
University of Western Ontario Computing
Centre

Dr. Sylvia Charp
Philadelphia Board of Education

Mr. Mort Ruderman
Digital Equipment Corporation

Mr. Ray Lindsay
Digital Equipment Corporation

Mr. James G. Smith
Pennsalt Chemicals Corporation

Mr. Rolf Nordhagen
University of Oslo

Dr. Arnold H. Fainberg
Pennsalt Chemicals Corporation

Mr. David B. Dennison
Digital Equipment Corporation

Mr. Robert Fernekes
Dow-Jones & Company, Inc.

Mr. Samuel J. Wynch
RCA

Mr. Ron Ragsdale
Ontario Institute for Studies in Education

Mr. J. J. H. Park
N. R. C.

Mr. William Godwin
Educational Testing Service

Mr. Lauren Taylor
University of Pennsylvania

Mr. Arthur E. Sumner
RPA Computer Techniques, Inc.

Mr. Paul W. Knortz
Potter Instruments Company

Mr. William F. Weihner
Stanford University

Mr. Wayne G. Dengel
Digital Equipment Corporation

Mr. Garth Thomas
Ohio State University Hospital

Mr. John Elder
University of Cambridge

Mr. Robert D. McInnis
Digital Equipment Corporation

Mr. Redmond Sage
Whirlpool Corporation

Mr. J. R. Storey
Defense Research Board/DRTE

Mr. Sypko Andreae
University of California
Lawrence Radiation Laboratory

Mr. John Seuter
Stanford University

Mr. Don Barker
Digital Equipment Corporation

Mr. John C. Smallwood
Pennsalt Chemical Corporation

Mr. Frank Mina
Digital Equipment Corporation

Mr. Malcolm C. Bruce
N. I. H.

Mr. Anthony J. Stracciolini
University of Pennsylvania

Mr. Alex Marusak
Oak Ridge National Laboratory

Mr. Ronald A. Morrison
General Electric Company

Mr. John Jorgensen
Digital Equipment Corporation

Mr. David Baer Cotton
Digital Equipment Corporation

Mr. Jack J. Shrager
Federal Aviation Agency

Mr. Paul Weinberg
University of Pennsylvania

Mr. O. Choi
RCA

Mr. D. F. Battson
RCA

Mr. Frederick R. Kling
Educational Testing Service

Mr. William Kiesewetter
Digital Equipment Corporation

Mr. Richard Karash
Massachusetts Institute of Technology

Mr. Alan Ricketts
Digital Equipment Corporation

Mr. Richard F. Falt
Digital Equipment Corporation

Mr. Alfred G. Delker
Pennsylvania Hospital

Mr. Donald T. Payne
Educational Testing Service

Aloysius J. Polaneczky
Pennsalt Chemicals Corporation

Mr. Herman W. Vreenegoor
NIH

Mr. Robert Margolies
University of Pennsylvania

Mr. John A. W. Richardson
Potter Instruments Company

Mr. Norman Doelling
Digital Equipment Corporation

Mr. Philiffe Dumont
University of Pennsylvania

Mr. A. L Boni
E. I. du Pont

Dr. R. J. Kobrin
Mobil Research & Development Corporation

Mr. Joel A. Miller
LOGIC, Incorporated

Carol Sombers
Sombers Associates, Inc.

Bino Nanni
Digital Equipment Corporation

Mr. Frederick L. Hiltz
Cornell University

Mr. Ted Hess
Applied Logic Corporation

Mr. Neal Laurance
Ford Motor Company

Pauline Erickson
Digital Equipment Corporation

Mr. Rochelle Lauer
Yale University High Energy Physics
Gibbs Laboratory

Mr. Andrew L. Szilard
University of Western Ontario

Mr. William F. Carr
University of Wisconsin Medical School

Mr. Heinz-Wolfgang Kohler
University of Wisconsin Medical School

Mr. Thomas E. Spurrier, Jr.
University of Alabama Medical Center

Claudia Gail Coun
Computer Sciences Corporation

Mr. Gerald A. Masek
Presbyterian - St. Lukes Hospital

Mr. Paul E. Lund
University of California
Lawrence Radiation Laboratory

Mr. Fred R. Sias, Jr.
University of Mississippi Medical Center

Mr. G. E. Friend
University Computing Company

Mr. Dan W. Scott
University Computing Company

Mr. Robert L. Callery
Minneapolis - St. Paul Sanitary District

Mr. August E. Sapega
Trinity College

Mr. Kenneth G. Pavel
Trinity College

Mr. Adam S. Hanauer
New York University

Mr. Russell B. Ham
U. S. Public Health Service

Mr. D. Gerd Dimmler
Brookhaven National Laboratory

Carolyn A. Bailey
University of California
Lawrence Radiation Laboratory

Dr. Walter H. Moran, Jr.
West Virginia University

Mr. Harold L. Pearson
West Virginia University

Mr. George S. Cooper
LOGIC, Inc.

Miss Barbara Williams
Mobil Research & Development Corporation

Andre M. Gagnoud
University of Pennsylvania

Mr. Richard Bigelow
COSSECC

Mr. Paul Stewart Shrager
University of Pennsylvania

Mr. Rod Belden
Digital Equipment Corporation

Mr. D. W. Roberts
The Strand Hotel Limited

Dr. Bernard Migler
Behavioral Design Associates

Mr. Richard M. Merrill
Digital Equipment Corporation

Mr. Peter P. Goldstern
Digital Equipment Corporation

Mr. C. K. Stone
Digital Equipment Corporation

Bishnu Das Pradhan
University of Pennsylvania

Mr. David M. Forsyth
Harvard University

Karen A. Jackson
United Aircraft Research Laboratories

Mr. Richard Wales Macmillian
Communications & Systems, Inc.

Kay E. Rigg
New Mexico State University

Judith B. Edwards
Computer Instruction NETWORK

Alan David Loceff
Information Control System, Inc.

Mr. David Hartsig
Information Control Systems, Inc.

Mr. Allan Bellenger
University of Rochester

Belmont G. Farley
University of Pennsylvania

Mr. Harold E. Dixler
Nuclear Structure Research Laboratory
University of Rochester

Mr. Harold W. Shipton
University of Iowa

Mr. R. D. Benham
Battelle-Northwest

Mr. L. Richard Turner
NASA - Lewis Research Center

Mr. Harold R. Krall
RCA - Electronic Components Division

Mr. S. L. Cooke, Jr.
University of Louisville

Mr. Joel S. Pratt
Digital Equipment Corporation

Mr. Boyd R. Borrill
Brogan Associates Inc.

Mr. Richard E. Wyckoff
Intertype Company

Mr. P. J. Bell
University Computing Company

Mr. D. R. Reimer
Institute for Computer Studies
University of Manitoba

Mr. John Rayner
University of Maryland

Mr. David W. Gillette
Computer Instruction NETWORK

Dr. Donald W. Webert
School of Veterinary Medicine
University of Pennsylvania

Mr. Robert Snyder
Digital Equipment Corporation

Mr. Michael Edwards
Carnegie - Mellon University

Mr. V. Michael Powers
The University of Michigan

Mr. Richard J. DeJohn
Digital Equipment Corporation

Mrs. Evelyn Dow
Digital Equipment Corporation

Miss Julia Johnson
Digital Equipment Corporation

Miss Margret Noble
Digital Equipment Corporation

Mr. John J. O'Connell, Jr.
Digital Equipment Corporation

Cary W. Armstrong
Digital Equipment Corporation

Mr. John W. Carr, III
Moore School of Electrical Engineering
University of Pennsylvania

Mr. Richard Gruen
Digital Equipment Corporation

Mr. D. McArthur
Hahnemann Medical College & Hospital

Dr. William P. Boger
Wayne, Pennsylvania

Mr. Kirstein
Institute of Computer Science

Mr. Spencer Lauer
West Haven, Connecticut

Mr. James Pitts
Digital Equipment Corporation

Mr. Malcolm Sheinker
Digital Equipment Corporation

PRINTED IN U.S.A.