# DECUS PROCEEDINGS

## FALL 1968

## PAPERS AND PRESENTATIONS

of

Digital Equipment Computer Users' Society

Maynard, Massachusetts

FALL
1968

# PAPERS AND PRESENTATIONS

of

Digital Equipment Computer Users' Society

# FALL SYMPOSIUM

DATA ACQUISITION AND CONTROL
EDUCATION , INTERACTIVE SYSTEMS
BIOMEDICINE

DECEMBER 12, 13, 14, 1968

JACK TAR HOTEL
SAN FRANCISCO, CALIFORNIA

# CONTENTS

DATA ACQUISITION AND CONTROL

INTERACTIVE SYSTEMS

*This paper was not presented at the meeting but was submitted for publication.

# PREFACE

The Fall 1968 DECUS Symposium broke all records with an attendance of 390. Extending the meeting to two-and-a-half days also was a deviation from the norm of two-day meetings. Sessions included presentations in Interactive Systems, Data Acquisition and Control, Education, Biomedicine, Hardware, FOCAL, and workshops in software systems for the PDP-8, PDP-9, PDP-6/10 and LINC-8. Available during the meeting was a PDP-9 and a LINC-8 for demonstration and use by the attendees.

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, these papers are listed in the appendix. 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 and author index.

Special thanks to Meetings Chairman, Professor Philip Bevington and to all Session Chairmen and speakers for their support and participation.

# DATA ACQUISITION AND CONTROL

# A PDP-8/S AS A PROCESS CONTROLLER FOR MANUFACTURE OF TANTALUM THIN FILM T-PAD ATTENUATORS

H. D. Marshall & R. L. Siegel
Western Electric Company
Allentown, Pennsylvania

## ABSTRACT

A PDP-8/S forms the nucleus of a complex anodizer tester for the manufacture of T-pad attenuators. This paper describes the basic problems of anodizing and testing tantalum resistors and the design consideration of hardware and software to meet this task. The hardware coverage in this discussion is limited to basic descriptions of the peripheral equipment to allow a more thorough treatment of the software logic.

The Electronic Industry continues to introduce more complex devices every year. The manufacture and testing of these devices have become more time consuming, complicated and expensive. The relatively recent development of the "small" computer has facilitated the design of more sophisticated manufacturing and testing equipment, thus, reducing the device cost. The intent of this paper is to discuss a method of adjusting and testing tantalum thin film attenuators using a Digital Equipment Corp. PDP8/S as the process and test coordinator.

The scope of the discussion is the familiarization of the reader to the problem of anodizing attenuators, the necessary hardware development, and the marriage of the hardware to the software.

## DEFINING THE TASK

It was desired to create a system to anodize and test tantalum thin film "T" attenuator pairs with values of 8.0, 4.0, 2.0 and 1.0 ± 0.01 db and 0.8, 0.4, 0.2 and 0.1 ± 0.01 db with input and output impedances of 300 ± 2% ohms.

The electro-chemical process of anodizing is beyond the scope of this paper, however, a few basic facts will be discussed to establish the tasks expected of the hardware.

The resistive value of tantalum thin film can be increased by covering an area of tantalum with electrolyte (a mild acid solution) and passing current from the tantalum to a probe suspended in the electrolyte. Tantalum oxide, an insulator, forms on the tantalum film as the film reduces in thickness during anodizing. (See Figure 1) Resistors can be anodized to a predetermined value by alternating between measure and anodize circuitry increasing the anodizing voltage slightly during each cycle. Thus, several important requirements for the hardware are defined, i.e., an adjustable anodizing voltage source, an alternating measure-anodize system and fixturing* to mechanically attach the measuring and anodizing probes to the tantalum circuits.

The above method may be used to adjust single or multiple sets of resistors and when many resistors or sets of resistors are to be adjusted simultaneously it becomes apparent that a "controller" and "memory" must be employed.

A "T" attenuator (or pad) is simply three resistors connected in a "T" configuration and can be easily duplicated by a tantalum thin film configuration as seen in Figure 2. A "T" attenuator is a device used to reduce power or voltage to some desired level and is usually inserted in a line of a given impedance and does not alter the impedance of the line. In this case, the insertion impedance is 300 ohms. Figure 3 shows an attenuator inserted in a 300 ohm line and cutting into the line at either dotted line 300 ohms would be measured in each direction. Therefore, a properly constructed attenuator inserted into a line will require the same amount of power from the source as that required before insertion.

There are only three discrete resistor values to define a T-attenuator of a given input-output impedance and power or voltage loss. If the input and output impedances are desired equal, $R_1$ must equal $R_2$ (Figure 2). Then $R_1 + R_2$ called $R_{series}$ and $R_3$ called $R_{shunt}$ form a balanced attenuator. By assigning values to $R_{series}$ and $R_{shunt}$ a "T" attenuator is fabricated with a defined loss and its input and output impedance balanced. To calculate the necessary values of $R_{series}$ and $R_{shunt}$ to produce a given attenuation and impedance (or visa versa) see Appendix 1.

Imagine $R_{series}$ and $R_{shunt}$ in Figure 3 are adjustable resistances. Increasing $R_{series}$ increases the loss ratio and increasing $R_{shunt}$ decreases the loss ratio. However, increasing either $R_{series}$ or $R_{shunt}$ increases the input impedance of the attenuator. These facts must be arranged into a workable anodizing plan to define hardware and software design parameters. Let us think through the necessary steps to adjust an attenuator, for example, to 8.00 ± 0.01db at 300.0 ohms ±2%.

The first step is to separately adjust $R_{series}$ to the calculated value needed for an 8.00db attenuator at 300 ohms. During the adjustment of $R_{series}$, control the balance of $R_1$ and $R_2$ of which $R_{series}$ is composed. With $R_{series}$ adjusted to the calculated value and $R_1$ equal to $R_2$ there is only one

*Fixturing, although in this application is both complicated and important, is primarily a mechanical engineering problem and will not be discussed.

$R_{shunt}$ resistive value which will exactly define the attenuator as 8.00db at 300.0 ohms. Therefore, $R_{shunt}$ must be separately adjusted to the proper value.

The method for performing this $R_{shunt}$ adjustment is to insert the attenuator in a circuit as shown in Figure 3 for measuring loss ratio, i.e., the ratio of one half of the power supply voltage* to the voltage across $R_{load}$. Then, anodize $R_{shunt}$ up in value causing the insertion loss to lower to the specified limit as also shown in Figure 3. If, as previously proposed, $R_{series}$ was accurately adjusted and $R_{shunt}$ adjusted to the correct limit, the attenuator is now 8.00 db at 300.0 ohms. Notice, with this scheme it is not necessary to measure the input-output impedance during adjustment. Precise selection of the 300 ohm R source and $R_{load}$ resistors (in the measuring circuit) and adjustment of $R_{series}$ and $R_{shunt}$ to the exact calculated values guarantees the correct input-output impedance of the attenuators.

## HARDWARE CONSIDERATIONS

The basic task has been defined and the actual attenuator circuits to be processed and tested are shown in Figures 4 and 5. Notice in Figure 4 the pairs marked 1, 2, 4 and 8 are the 1, 2, 4 and 8 db pads and in Figure 5 the pairs marked .1, .2, .4, and .8 are the 0.1, 0.2, 0.4 and 0.8 db pads. In later discussion, the top four pads in each circuit are referred to as "Top" or "Upper" pads and the bottom four pads are referred to as "Bottom" or "Lower" pads. This describes their physical location on the circuit. Also, the attenuators in Figure 4 are referred to as "High" pads and in Figure 5 are referred to as "Low" pads which describes their electrical value. Therefore, the proposed equipment must be capable of processing and testing either High or Low pads of the values presented above (all with input and output impedance of 300 ohms). To spice the requirements even more, there are nine identical circuits on each ceramic substrate, eight pads on each circuit and essentially two resistors ($R_{series}$ and $R_{shunt}$) on each pad or 144 total anodizations per substrate. A layout of the substrate can be seen at the bottom of Figure 6. The upper portion of Figure 6 is the block diagram of the hardware necessary to adjust and measure the substrate. Each pad must be serviced by 5 fixture contacts (three for measuring, one for $R_{series}$ anodize and one for $R_{shunt}$ anodize); and remembering there are 72 pads per substrate means 360 contacts must be made at the substrate surface. Also recall that no anodizing contact may touch any part of the tantalum circuit. The contacts are brought down in open ceramic areas near the resistors to be anodized and are immersed in the electrolyte paste which has been carefully screened on each associated resistor** and includes the open area. With reference to Figure 6, the fixturing pins contact each pad of either the High or Low value substrate and the Matrix connects only the 18 identical pads to the Pad Selector at a given time (i.e., with regard to Figure 4 first connecting 18 - 1 db pads until adjusted then 18 - 2 db pads, followed by 4 db and 8 db pads in order determined by the controller). The Pad Selectors normally connect the appropriate 5 leads from each pad to its associated anodizer; however, by controller selection any 2 pads may be connected to configuration selectors for Ratio measurement. Notice, the hardware is divided essentially into two identical branches. The reason for the hardware division is simply the fact that the Ratiometer operates slowly compared to the controller. The controller is essentially servicing two separate hardware facilities; i.e. the left side (Figure 6) adjusts "Top" pads, the right side adjusts "Bottom" pads and the controller alternates between them.

To summarize the hardware: The pins contact the attenuator circuits, the Matrix selects a column of similar pad values to be operated on, the Pad Selector connects all pads to its associated Anodizer or selects a pair of pads and connects them to the Configuration Selectors, the Configuration selectors connect the pads to be tested in a Zin, Zout, or insertion loss circuit configurations, the Ratio meter reads analog ratios and converts them to binary coded decimal for controller comparison to stored values.

## THE OPERATION

A block diagram of the actual equipment is seen in Figure 7 and the following discussion describes the tasks done by the hardware when accessed by the controller (or computer). Each of the 18 anodizers has its own device number and device selector. The anodizers are controlled by placing a word in the accumulator and sending this word through an associated device selector, flip-flops, and relay drivers to cause the anodizer to do any of the following operations:

1. Generate a voltage step increase on the output of the anodizer and anodize the associated resistor to this new voltage level (the duration of anodizing times are preset by interface hardware).

2. Operate on "Series" or "Shunt" resistors as determined by the software program.

3. Set any Anodizer to "slow" or "fast" anodize as determined by the proximity of the preselected desired value.

4. Discharge any anodizer when the desired value is attained.

5. Change from anodize to measure mode for series or shunt (this transfer is executed by the Pad Selector under the control of the anodizing-board device Selector).

The equipment Start and Stop buttons (not the start and stop buttons on the computer) are used to start and stop the processing of the T-pads. Both of these buttons are under control of the same device selector and allow the computer to determine if either of the buttons has been depressed. The start button is enabled only when the fixture is in the raised position and the stop button which is always enabled is also connected to the computer interrupt buss.

*Loss Ratio (db) = $20 \log \frac{V_R \text{ Load}}{1/2 \text{ (V Supply)}}$

Samples in Appendix 1

**Isolation dams of asphalt are temporarily screened on the substrate to restrict the electrolyte from touching the gold contact areas and to separate the $R_{series}$ and $R_{shunt}$ portions of the tantalum.

The Matrix allows the system to process any one of four columns of T-pads from the fixture. Each column consists of 18 pads with 5 leads per pad. The three measuring leads of each pad are connected to the Pad Selectors and the two anodizing leads are connected to each of the 18 anodizers. The computer controls the column to be selected or reset by placing a word in the Accumulator and sending the word through the Matrix device selector and 10 amp drivers to operate the matrix.

There are two fixtures associated with this equipment. One fixture is used for high value pads and the other fixture for low value pads. (The High and Low substrates differ geometrically, consequently, need separate pin configurations) Both of the fixtures are parallel wired to the Matrix and the program allows only one fixture to be used at a time. The computer takes an initial predetermined resistance reading to identify whether a high or low substrate has been contacted by the raised fixture then, utilizes this information to initialize the proper limits for processing. The computer also controls the lowering of the fixture by using an extra pulse from one of the anodizer device selector boards.

The pad Selectors normally connect each T-pad to its associated anodizer, however, the computer may select a pair of T-pads in a given column and connect them to the Configuration Selector for measurement during programmed control.

The computer controls the Configuration Selectors by placing a word in the accumulator and sending this word through a device selector, flip-flops, and relay drivers to each Configuration Selector. The Configuration Selectors are operated in parallel to setup the following ratio measurement circuits:

1. Series Resistance and Series Verify (Series Verify determines if the series measuring pins are contacting the substrate)

2. Insertion loss

3. Input impedance

4. Output impedance

5. Shunt Verify (determines if the shunt pin is contacting the substrate.

There is a series resistor balance circuit associated with each configuration selector. Each balance circuit is always energized but is only used when the configuration selector is measuring the series resistance of a T-pad. The balance circuit sets a latching relay on each associated anodizer board to produce nonlinear anodizing of the series resistor with respect to the shunt resistor. This is accomplished by forcing a voltage gradient across the series resistor of a pad during anodizing and causes the lower resistance half of the series resistor to anodize faster. This process tends to equalize the attenuators input and output impedances. The output signal from each balance circuit is fed through the pad selectors to the associated anodizer board. The timing for this balance routine will be covered in the later discussion of the Digital Ratiometers.

There are two Digital Ratio Voltmeters (DVM's) used in conjunction with the configuration selectors to make the actual ratio measurements. The upper DVM is used with the upper configuration selector and the lower DVM with the lower configuration selector.

Each DVM is controlled by the computer through its two device selectors. The BCD output from the DVM's are handled as "double precision" numbers, each half with a separate device selector. The computer can also sense when either DVM has completed a reading. This sensing is done through the device selector that transmits the least significant portion of the BCD output. The other device selector which transmits the most significant portion of the BCD output is also used to cause the DVM to start a reading. When a DVM reading is required, the first step that occurs is an 8 ms. delay for the system to settle down and then a 15 ms. read command to the DVM. It is during this 15 ms. read command that the balance circuit associated with that DVM is also commanded to make a balance measurement. Both the 8 ms. and 15 ms. delay and read durations for each DVM are under hardware control.

Nine inker solenoids are located under each fixture and operate if any of the 8 T-Pads on a circuit have failed final measurements by placing an ink mark on the underside of the faulty circuit on the substrate. Both sets of inkers are operated in parallel from the computer but either set of inkers is disabled by raising the opposite fixture. The computer controls each inker by placing a word in the accumulator corresponding to the inker to be activated. This word is sent through the inkers device selector, flip-flops, and solenoid drivers to energize the proper inkers.

The software program was written to combine the previously described peripheral devices into a attenuator processing and testing system.

## ALL PADS ANODIZE LOGIC FLOW (FIGURE 8)

The main anodizing program is stored from location 200 to location 3377 and on part of the zero page. The program is started at location 0200 and a command to drop the fixture is issued. The sections of the program for "Verify" and "Measure" are then initialized. The Verify portion will check each of the 72 T-Pads to determine if each of the 3 measuring pins per T-Pad are making connection. The measuring portion will measure each T-Pad for input impedance, output impedance, and insertion loss. The computer will wait for a prepared substrate to be raised in the fixture and the program to be started. When the program is started, the computer takes an initial reading to determine whether the substrate is a "high" or "low" value and stores this information. The switch register setting will direct the program to either measure the pads or verify, anodize, then measure the pads. Assume for this example the latter is desired. The verify limits are set up and the first pad is tested for "verify series" and "verify shunt." The computer makes a comparison between the DVM readings and stored limits to verify low series or shunt pin contact resistance. If verification is negative, flashing lamps afford a choice to "override" or "drop the fixture" and restart. When verify is satisfied, a set of limits are selected based on the initial high - low readings previously stored.

An appropriate Matrix column counting location is set to (1). The configuration selectors are set for series measurements and the anodizers are set up for series anodize. The computer tests to find if the last matrix column has been finished. Since it has not, the matrix is set to column one, and both DVM's are commanded to read. The interrupt is then turned on and if no devices are asking for service, a delay bit is rotated in the accumulator. The interrupt subroutine will service only one DVM at a time. When one DVM has been serviced, the subroutine alternates to service the other DVM and stores a count until 18 measurements have been taken (which constitutes one pass on all the anodizers.)

The devices that may call the interrupt are the Upper and Lower DVM, parity error, equipment stop button, or the teletype. If a parity error occurs, the computer is automatically shut down. The equipment stop button causes all processing to stop and the fixture to drop. If the teletype reader or punch calls, it is completely ignored sending a "clear" pulse through its device selector. There will be approximately 30 ms. from the time a command is issued for the DVM's to read until a DVM calls the interrupt. Since this is the first reading, only the upper DVM will be serviced. If the lower DVM calls the interrupt first, the interrupt subroutine will ignore it and wait for the upper DVM. When the upper DVM finally asks for service, it s reading will be gated into the computer and a double precision comparison will be made against a stored fast-anodize limit. If the reading is less than the fast limit, that anodizer will be set to fast-series-anodize and charge. (A 100 ms. anodize pulse will occur) If the reading is greater than the fast limit, it will then be compared against a finished value limit. If the reading is less than the finished value, that anodizer will be set for slow-anodize and charge (a 100 ms anodize pulse will occur). If the reading is greater than or equal to the finished value, a "finished" register will be incremented and the anodizer discharged. In any case, the next circuit will be set to measure and the Upper DVM commanded to read. Control is then given to the interrupt subroutine which will now accept only the Lower DVM. When the Lower DVM asks for service, the same type of processing occurs and the reading is compared against the proper limits to check for fast-anodize, slow-anodize or discharge. The program now checks to see if 18 pads have been measured. The program investigates whether 18 pads have completed series anodize, or a specified time limit has been exceeded. If all 18 pads are finished (or the allotted time exceeded) the process is then set up for shunt-anodize, insertion loss measurements, and a new set of limits are used. If all 18 pads have not been anodized, the next Lower pad is set to measure and control is again given to the interrupt subroutine. This process is continued until all 18 pads are finished (or timed out). The shunts of the T-Pads are anodized following the same process as described for the series anodize except different limits are used. When all of the shunts have been adjusted to a specified insertion loss ratio, the Matrix Column Register is incremented. The program again sets up for series-anodize. If the Matrix Column Register is not (4), the process of anodizing series and then anodizing the shunts is continued until all four columns have been completed. At this time, the interrupt is turned "off" and the measuring subroutine is given control. The measuring routine has the capability of rejecting circuits for insertion loss to tolerances of ± .01db, .03db, .05db, or .07db (determined by previous setting of the switch register) and output impedance of 300 ohms ±2%.

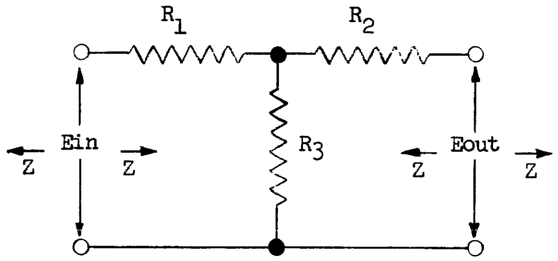
A circuit (8 pads) is to be rejected if any pad fails any test limits. Therefore, any time a pad is out of limits, one of the nine associated circuit registers are incremented. This information is used later to operate inkers in the fixture to mark circuit failures. However, the proper setting on the switch register will prevent inking of rejects and command teletype print out of insertion loss, input and output ratios. At this point, the fixture will release and is ready for the next substrate.

Included in the computer program are 16 routines which aid in the general maintenance of the equipment. These additional programs allow an operator a fast examination of approximately 90% of the machine hardware and in many cases describe on the teletype the location of an existing problem.

This equipment cost approximately $78,000 including engineering and design and will process and test a complete substrate in approximately one minute.

## APPENDIX 1

### DETERMINATION OF RESISTANCE VALUES FOR A BALANCED T-ATTENUATOR NETWORK



K = Ein/Eout, this ratio is the desired voltage ratio corresponding to the desired attenuation in db. $(K > 1)$

$$\text{LOSS (db)} = 20 \log_{10} \text{Ein/Eout}$$

$$R_1 = R_2 = Z \frac{(K - 1)}{K + 1} \text{ and } R_3 = \frac{2ZK}{K^2 - 1}$$

Example: calculate 8.00db, 300 ohm Zin, Zout Pad.

$$8.00 \text{ db} = 20 \log \text{Ein/Eout}$$

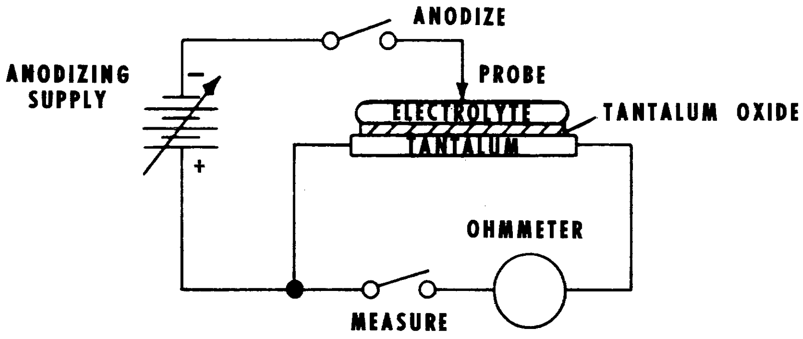
$$\text{Ein/Eout} = \text{antilog } \frac{(8.00\text{db})}{20}$$

$$\text{Ein/Eout} = \text{antilog } (.400)$$

$$\text{Ein/Eout} = 2.5119 = K$$

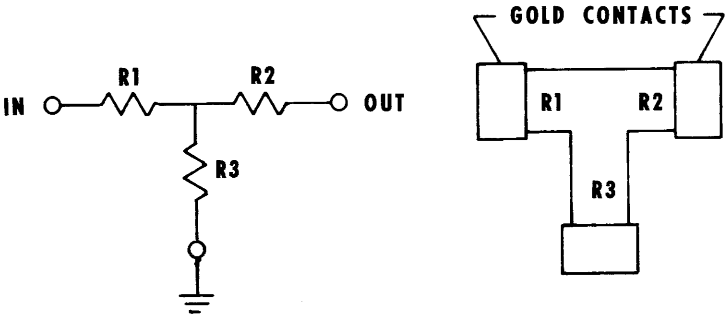
$$R_1 = R_2 = 300.0 \frac{(2.5119-1)}{2.5119+1}, \quad R_3 = \frac{2(300.00)(2.5119)}{(2.5119)^2 - 1}$$

$$R_1 = R_2 = 129.155 \qquad R_3 = 283.845$$



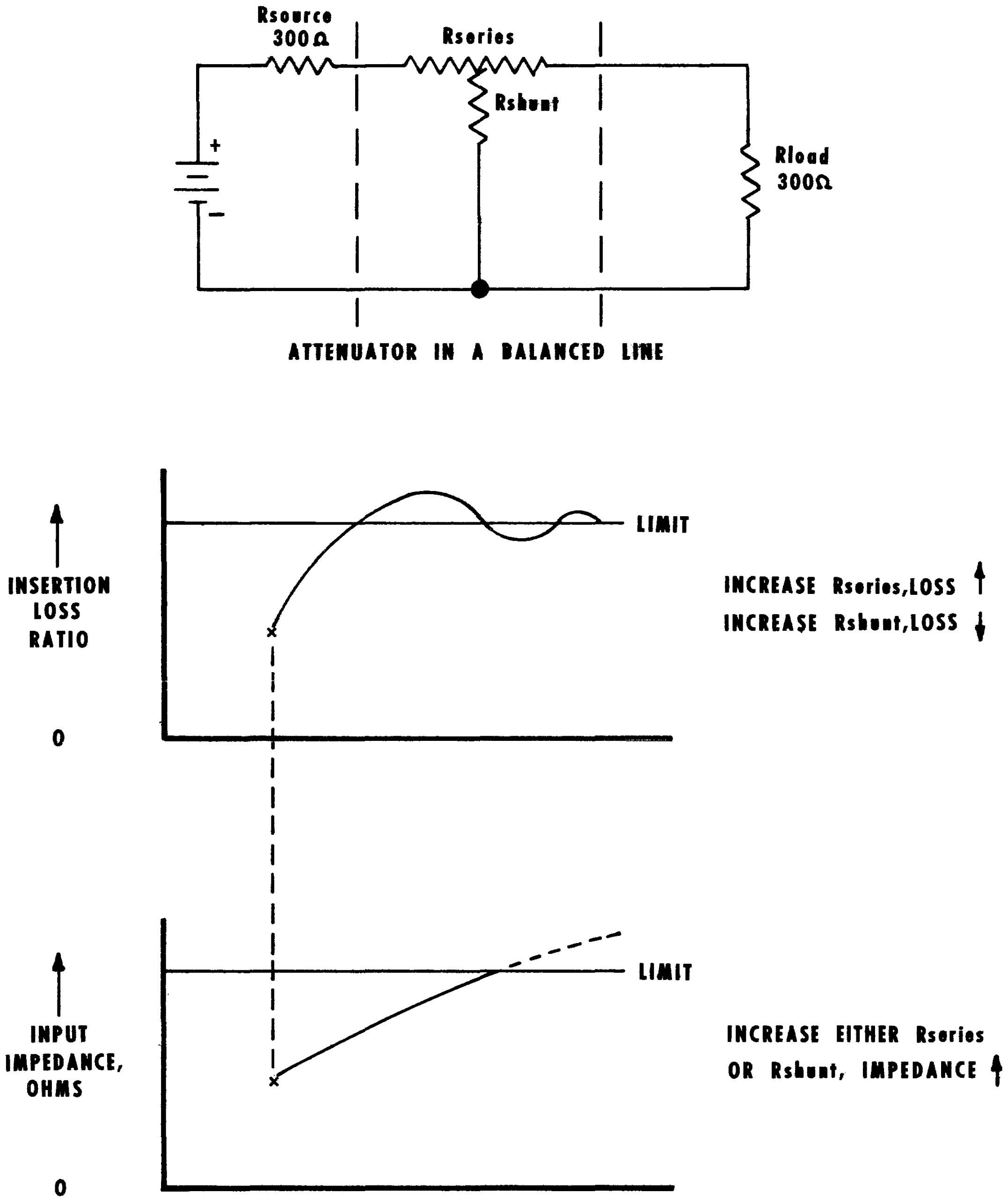
SIMPLE ALTERNATING ANODIZE–MEASURE SCHEMATIC

FIGURE 1



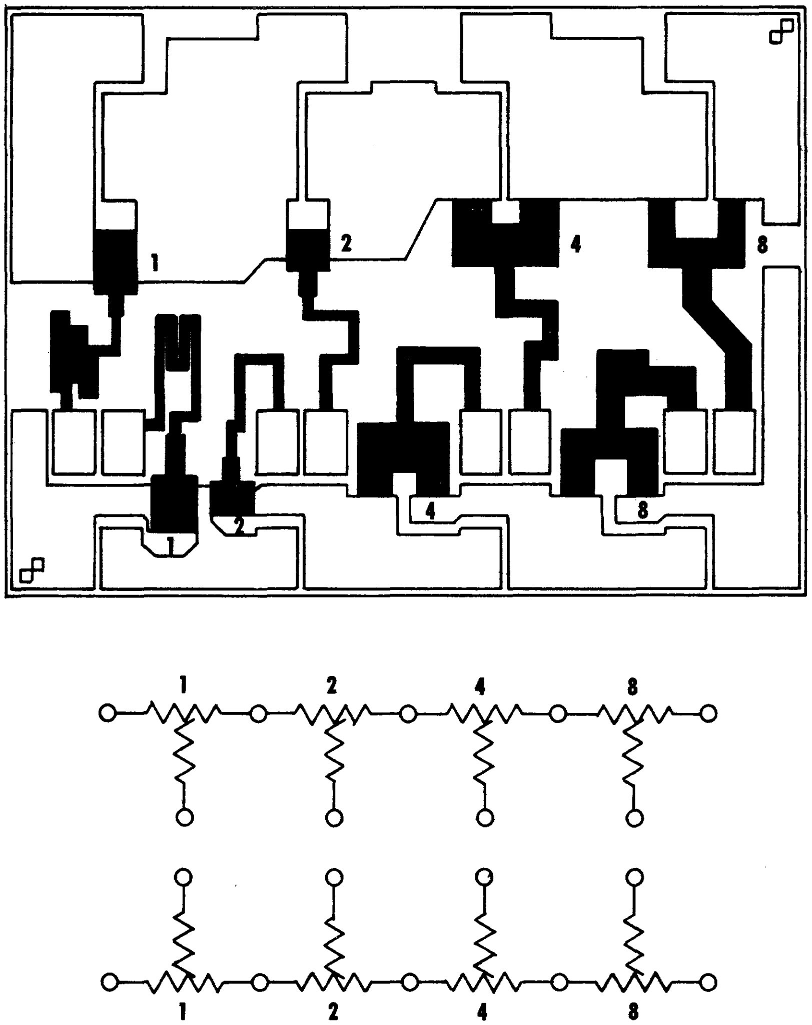
"T" ATTENUATORS

FIGURE 2



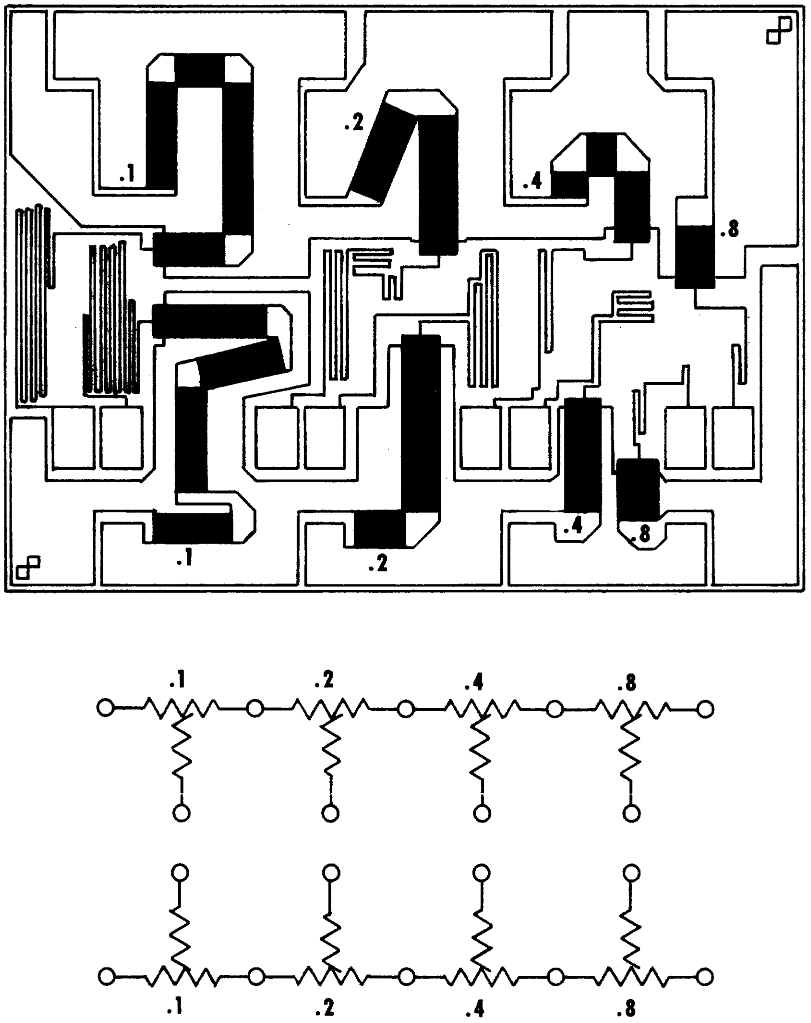
IMPEDANCE AND INSERTION LOSS
AS FUNCTIONS OF Rseries AND Rshunt

FIGURE 3



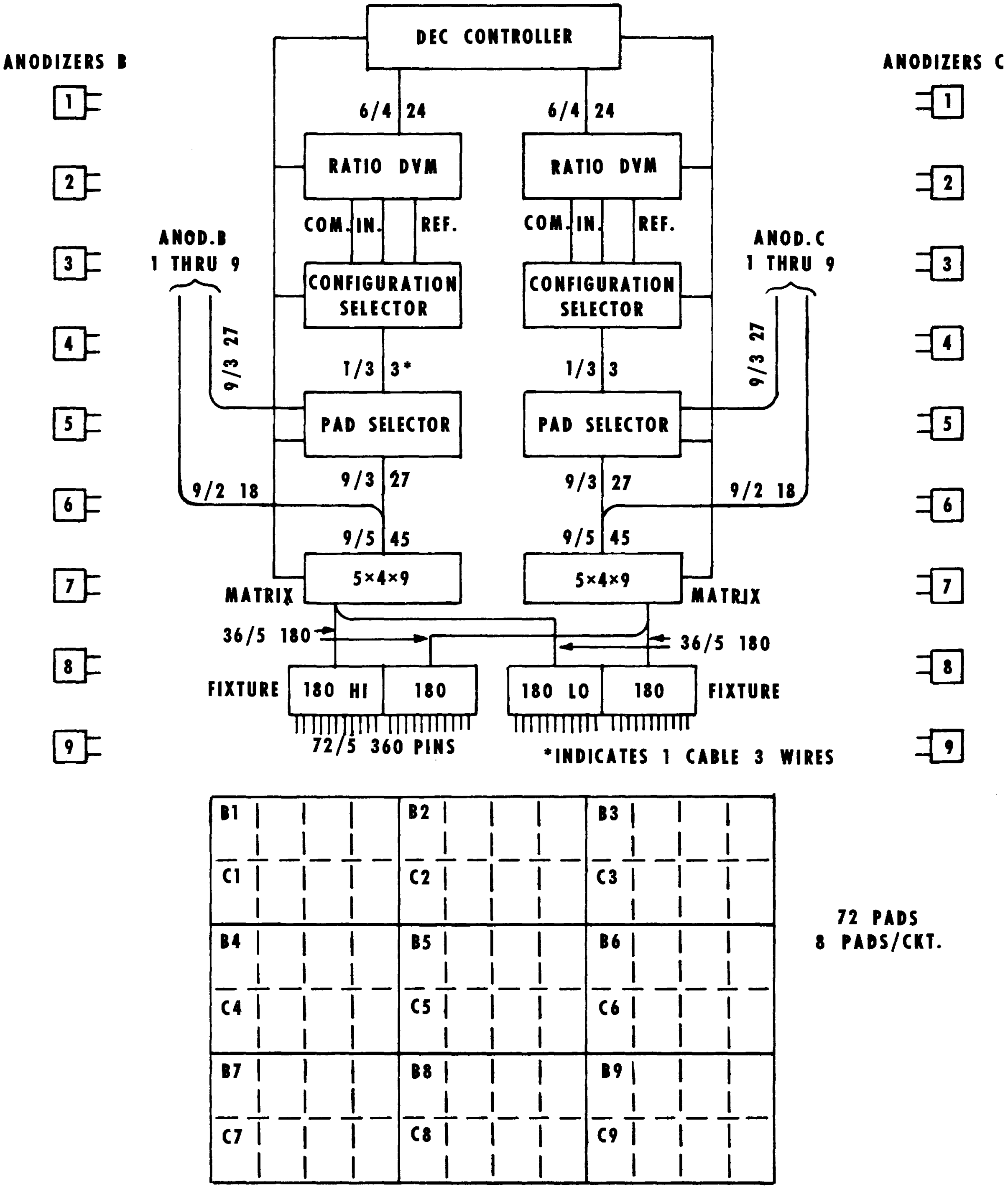
F TYPE IN-BAND CIRCUIT
(15 db)

FIGURE 4



**F TYPE IN-BAND CIRCUIT**
**(1.5 db)**

**FIGURE 5**



## ANODIZER–TESTER BLOCK DIAGRAM

## FIGURE 6

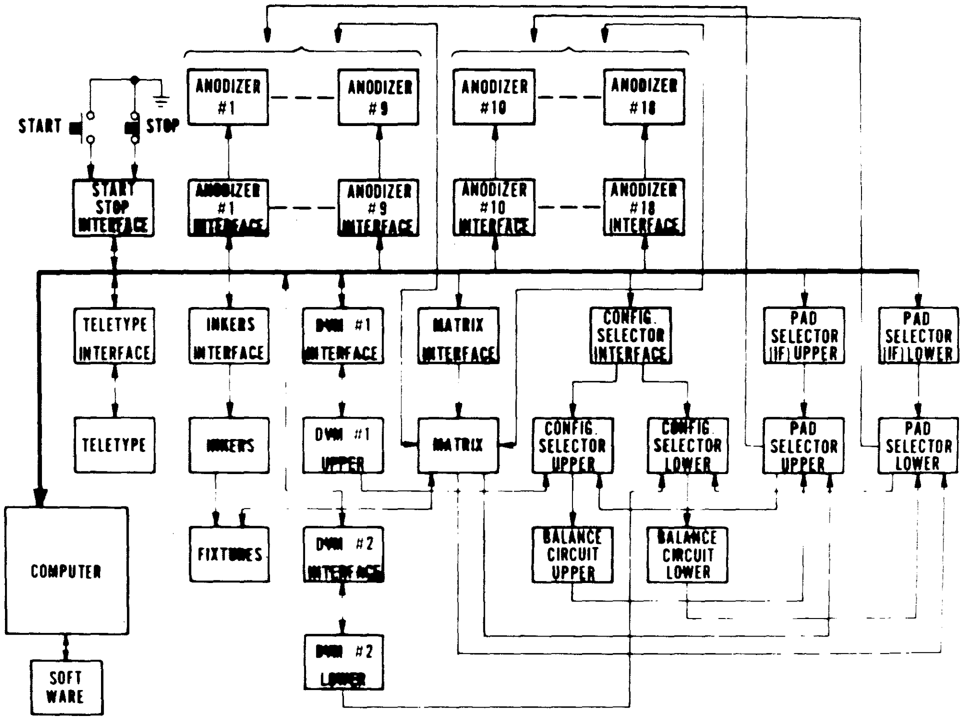


FIGURE 7

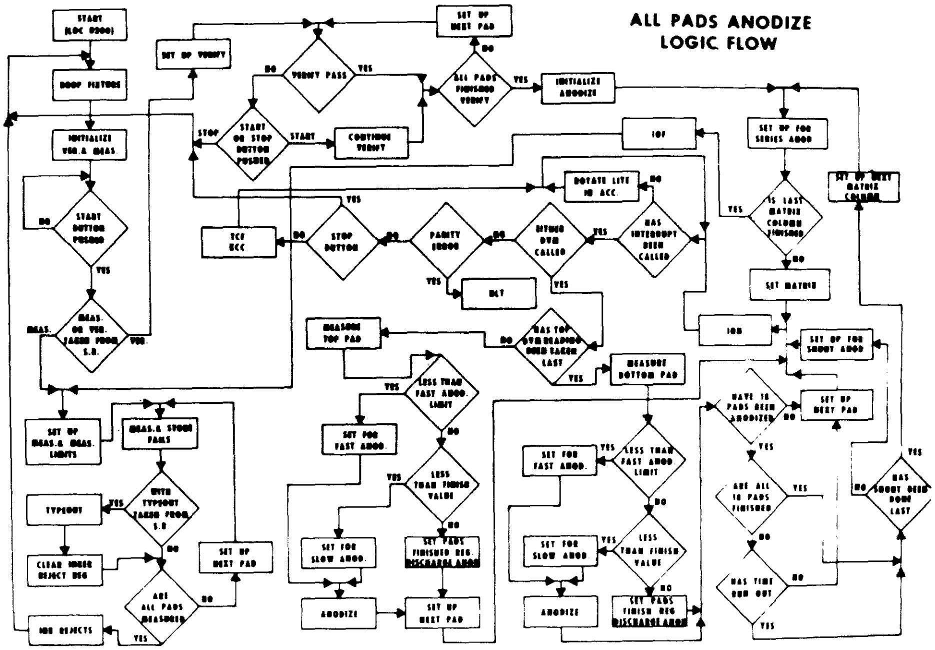

ALL PADS ANODIZE LOGIC FLOW

FIGURE 8

# THE GASCHROM-8 SYSTEM

By: Bradley Dewey III and Gary Cole
Digital Equipment Corporation
Maynard, Massachusetts

In many research and quality control laboratories in the chemical process industries, multiple gas chromatographs are relied upon to supply chromatograms which are analyzed and used for quantitative analysis.

These instruments are often running simultaneously and on multiple shift bases. Characteristic problems involved in this sort of wholesale chromatography are many. Often there are many peaks of interest, whoze size varies over many orders of magnitude; these must be integrated and mormalized or compared to some standard.

Noise is an ever present and varying quantity. Baselines drift and confuse the area determination process and apparent sensitivities. Fused peaks create the need for judgement of area allocation between two or more peaks. Column aging and other variables cause calibrations to change which requires a determination of up-to-date component response factors.

All these difficulties are costly. They lead to wasted time, money, and personnel. A large number of people is required to analyze and quantitate the chromatograms. A long time interval is present between sampling and reporting. Accuracy is limited. Integration and area allocation are seldom as accurate as the chromatographic instrument. Finally, manual data analysis, with its associated lead times causes a low utilization of the expensive chromatographs. The GasChrom-8 system has been designed to solve many of these problems. It is a dedicated, computer based system for acquisition and analysis of up to 22 chromatographs, simultaneously. The system automatically accomplishes the following:

1. It collects all input signals from up to 22 instruments.
2. It allows parallel operation of a strip chart recorder and its associated attenuation switching.
3. It calculates peak areas and peak retention times.
4. It allocates area of overlapping peaks.
5. It corrects for baseline shifts.
6. It calculates component concentrations.
7. It identifies peaks by name and elution time.
8. It applies appropriate response factors.
9. It types a complete analysis report.

This system is completely automatic except for sample injection.

The program is divided into four discrete sections. FIRST, the monitor. This operates the ADC, multiplexer, chooses gain ranges for the amplifier, operates the local operator's console and the teletypes. SECOND, the foreground program. This processes the chromatograph data; forming and integrating peaks and storing peak data on the disk in a chained array. It incorporates:

1. digital signal filtering
2. threshold logic peak detection
3. automatic baseline determination
4. shoulder detection.

THIRD is the report generator. This processes stored peaks and produces analysis reports ready for typeout at the end of a chromatogram. It incorporates the following:

1. baseline correction
2. overlapped peak resolution
3. relative retention time calculation
4. peak identification
5. raw area reporting
6. area normalization
7. adjustment by response factors
8. internal standard computation.

FOURTH is the conversation mode. This allows entry, deletion, modification, and reporting of methods for analysis. Operator prompting and validity checks are incorporated. In includes:

1. a library facility for maintaining up to 100 user developed analysis methods
2. automatic calibration of analysis methods

This software package therefore affords dynamic interpretation of gas chromatography data. It's variable sampling rates from 3 points per second to 60 points per second enables capillary columns to be run with ease. Its variable sensitivity to peaks may be set vastly different at differing points in the chromatogram through <u>four</u> separate sensitive software controls over amplitude threshold, minimum peak araa, minimum peak time, and sensitivity to shoulders.

Identification of peaks is very flexible due to the user creation of the compound table portion of the analysis method. The expected elution time and compound name are entered first. A tolerance is then specified. This is a figure in seconds which provides a time window around the expected elution time. The largest peak in this window is given the compound name. Windows may be overlapped for closely neighboring peaks. The response factor (which may be updated at any time by calibrating the method) and calibration weight of the component in the standard are then entered.

The complete computation may be of four types. Area normalization, internal standard, external standard, or calibration Each report format includes a printout of tolerance in ± %. This number reflects where, in the expected time window, the peak actually appeared. When examined frequently, this is an excellent index of column aging and general repeatability of the chromatograph system.

The calibration is used to calculate, or update response factors on a ssandard sample on demand. Once determined and made a part of the analysis method automatically, it will not again be changed until such time as another calibration is requested. Therefore, a bum sample will not ruin a carefully created method.

GasChrom-8 makes chromatography date more meaningful and sees features which the human analyist cannot. Both accuracy and repeatability become instrument limited, not analysis limited. Shoulders not even visible on the strip chart recording will have their accurate component percentages reported. Overlapping peaks will be repeatedly reported with the proper area allocation and correct baseline will always be accurately computed.

The GasChrom-8 is a very powerful system. The basic hardware includes an 8K, PDP-8/I, extended arithmetic element, a 32,000 word disk, 12 bit ADC, a 9 gain range programmable amplifier and individual isolation amplifiers.

Up to 5 teletypes may be operated, independently and remotely from the basic system. Teletypes and the individual chromatographs may be run remotely as far as 1500 feet from the main system without modification.

These characteristics are due to the specifications of the analog circuitry. Depending on whether a X1 or X10 isolation amplifier is selected, the full scale range of the system is one microvolt to one volt or ten microvolts to 10 volts.

The imput amplifier is differential and has a one megohm 60 db at D.C. Common mode voltage is 300 volts.

The most useful part of the entire system is the user created method of analysis of which 100 may be stored on the system's disk. This includes the compound table described previously. There are only two parameters which must be set and held constant for the entire chromatogram; they are report type and sampling rate. All other functions, or software controls, and there are eleven, may be different at any point in the run. In the enter method (conversational mode) these function codes are all placed at a selected time, in seconds, from the beginning of the run and are weighted to provide the desired sensitivites -

Code 0 terminates the run and generates the report.
Codes 1 and 2 are used to start and stop data acquisition providing "no interest" windows
Code 3 starts the automatic baseline determination
Code 4 is the amplitude threshold level for peak detection.
Code 5 sets sensitivity to shoulders.
Code 6 sets the peak detection examination time.
Code 7 sets a time, after peak crest at which a <u>peak</u> tailing peak is terminated.
Code 8 allows unique points in time to be chosen as baseline points.
Code 9 determines the area threshold for peak detection.
Code 10 allows the status of three optional relays to be changed for control of the chromatograph. These may be used for such things as flushing the column.

Handling of unknowns may be handled in three ways. An unknown, in the GasChrom-8 system, is a component which is not listed in the compound table. They may be ignored or an arbitrary response factor of one may be assigned or the response factor of the most recent named component may be applied.

The reference peak to be used to scale the time frame of the run is also user selected. Its search zone may be made as wide as neighboring peaks allow. Each peak elution time is scaled by this reference peak before it is identified by the compound table. This is another control over the effect of column aging.

The GasChrom-8 analysis report may be used very nicely with the strip chart recording when one is desired. Aside from cross referencing thru elution times, each component in the report is coded with two of three letters.....B V S denoting the type of peak where B represents baseline, V - valley, and S - shoulder. So, a peak labeled SV begins at a shoulder and ends in a valley.

Report formats have also been modified for user convenience such as one case where a quality control lab wanted to supply the process operator with only name and weight percentage. These are relatively easily accomplished. Also, the system is expandable with extra core for up to 64 chromatographs running.

Therefore, the requirements of even the largest laboratory may be met.

# COMPUTERS AND NUCLEAR MAGNETIC RESONANCE SPECTROSCOPY

By: Charles P. Spector and Bradley Dewey III
Digital Equipment Corporation
Maynard, Massachusetts

Since the mid 1950's utilization of NMR has been growing at a high rate in the field of Analytical Chemistry as it has been recognized as a very powerful tool in qualitative analysis. Fortuitously, among the nucleii which exhibit the phenomenon are the most critical in organic chemistry, are isotopes of hydrogen, carbon, fluorine, and phosphorus. NMR now is a routine analytical procedure used by organic chemists in analysis, and confirmation of structure.

The NMR spectrometer applies a large magnetic field normal to another alternating magnetic field. This produces a macroscopic magnetic moment, in the sample, which will induce a voltage in a coil. The effect of the precessing nuclear magnetic moments, in resonance, may be measured as a change in impedance of the coil. Although the nuclear magnetic moment of a given nucleus is primarily determined by the external field, small differences, caused by interactions of shielding electrons and nucleii, cause slight shifting, called chemical shifts and spin-spin coupling. This spectoral information is what the investigator must acquire and analyze.

However, there are significant problems both in acquisition and analysis. In many cases, one is faced with a very low signal to noise ratio which can be so bad as to preclude spectrum recognition. This clearly calls for signal averaging on repetitive scans. Historically, hard wired signal averagers have been used. However, general purpose computers, and dedicated computer systems have been applied to the problem very profitably.

The Lab-8 system is such a dedicated system which may also be used as a general purpose computer.

The basic Lab-8 system is a powerful signal averager which incorporates a full scale general purpose digital computer to provide desired flexibility and versatility. The standard package includes a 4K PDP-8/I computer and an AX-08 laboratory peripheral.

The AX-08 consists of a 9 bit analog-to-digital converter, sample and hold, analog multiplexer, oscilliscope interface, amplitude discriminators, clocks and a power supply. An AA01A digital-to-analog converter is added to the basic system for controlling the sweep of the spectrometer.

In both the AX-08 and the AA01A, by eliminating the duplication of modules required, cost was decreased substantially without loss of capability.

The AX-08 will accept the signal from the NMR spectrometer both as signal information and through the synchronization input so as to trigger the sweep. This will usually be set up with the discriminators so that the TMS peak will trigger the scan. Since the Lab-8 system works on a rotating buffer concept, no information will be lost. The AX-08 will also control an oscilliscope and the recorder through a three part routine of the spectrometer for analog output of the averaged spectrum.

The software for the Lab-8 system will allow signal averaging, using the conversational mode for setting up parameters. The conversational mode in this case is used in conjunction with the oscilliscope. Questions and answers are displayed on the face of the scope. The scope asks the question then displays the answer that the experimenter feeds in on the teletype on the face of the scope.

There are two basic signal epics which are inherent in the software of the Lab-8 system. The total number of data points in the system is 1024. These may be used for one average or two averages, where the second average would be a higher resolution region of interest sweep. These are accomplished simultaneously. User sets parameters of these epics by choosing the number of data points, the sweep speed, and the dealy from the synchronization pulse. After the parameters are set up a summary appears on the oscilliscope displaying the beginning of the sweep, the sweep rate, and the end time of the sweep in microseconds, milliseconds, or seconds.

Another advantage inherent in the use of a general purpose computer for signal averaging is the fact that by adding more core memory, the number of data points may be increased for yet higher resolution. With the option XR, which is recommended for the AX-08 used for NMR, the region of interest is displayed as a brightened part of the spectrum on the scope. At any point during the averaging process by using one key on the teletype either the raw signal or the accumulated average may be viewed on the scope. Averaged spectra are available for output in three ways:

1. The first is the already mentioned display on the oscilliscope.
2. The second is a readout on the recorder of the spectrometer.
3. The third is a digital output on the teletype of any or all places in the spectrum, chosen by the user, and marked by him by two available calibration cursers adjusted on the face of the scope. He may get information from each data point between the cursers or he may choose to find the integral of the portion of the wave form between the two cursers and, of course, he can get punched paper tape output of the data at the

same time. This feature of integration is extremely valuable so far as final reporting on the NMR experiment is concerned.

There are two other aspects which recommend the Lab-8 as a signal averager for NMR· Along with the averaging capability of the general purpose computer capability, there is the ability to create user program to enhance the operation of the spectrometer itself. Also, pulsed or trangent NMR can be done with this basic system with some hardware addition. The first of these two uses could encompass instrument alignment, at present a somewhat tedious operation. Three other areas which could be aided significantly by using the computer are data collection, data enhancement, and spectrum synthesis.

Instrument alignment includes, for purposes of this discussion, calibration about the TMS peak and field homogeneity control. The computer could be used for calibrating about the TMS peak by scanning repetitively and recommending to the chemist which controls should be adjusted and in what direction. Homogeneity control can become extremely important in running a sample with a very low signal noise ratio such as might be found in work with carbon 13 or with phosphorus where literally thousands of cans may be required. In these cases all resolution can be lost if homogeneity is not held. Here the computer could be used to touch up the homogeneity controls so as to maximize the resolution obtained from the averaged spectra. The second major catagory of data collection could be eased by eliminating frequent changing of paper while making exploratory scans. It should be possible with the computer to simplify this process considerably with a CRT display. This implies using the computer to examine single scans as well as to the averaging. Even in doing a single scan therefore use of the spectrometer is very much simplified.

There is yet another potential method of signal averaging using the computer. The computer could operate the scan on a step basis. This would consist of setting the instrument to a fixed field value (assuming field frequency is held constant) taking an arbitrary number of samples at that point and averaging them, and then stepping to the next point. At the end of one scan each point would represent an averaged result which would be displayed as generated on the oscilliscope. There are many advantages to this approach and no additional hardware is required to synchronize the scans with each other. A spectra with a low signal to noise ratio, a single scan would permit the determination of whether data, was in fact, present in the spectrum. This point is particularly critical with nucleii other than hydrogen or fluorine. The primary disadvantage of the stepping approach to signal averaging is the unavoidable presence of low frequency noise components in the output of the spectrum. We haven't been able to find out very much about the characteristics of these low frequency components, whether they do exist to a serious extent therefore whether they can cause severe distortions in the spectrum. This will be investigated. If they are predictable however they could be removed by computer analysis.

Another significant aid in using a computer for NMR could be storage of spectrum on paper tape or a disk while an attempt is made to improve the resolution of the instrument and thereby improve the spectrum. If such improvement proves unsuccessful further analysis could be done with the digitized spectrum which was previously stored.

In most applications of NMR spectroscopy the chemist has a pretty good idea of the spectrum he ought to obtain from the sample he put in. He can run into two factors which make this identification difficult however in the area of data enhancement. First of these is the resolution of the instrument. It is affected both by the frequency of the instrument and the strength of the field and also by the homogeneties mentioned previously. As a result of low resolution two peaks may be combined into one widened peak or more likely into a broadened peak with a shoulder.

Sensitivity both relative to the noise level and relative to each point is used as the measure of the heighth of the peaks. In a spectrum of low sensitivity it may be difficult to distinguish peaks from the noise or it may be difficult to differentiate the heights of peaks from each other.

Sensitivity and resolution are in a sense mutually exclusive variables. The primary purpose of time averaging is to increase sensitivity by using multiple scans to average out random of noise and thereby removing it. On the other hand, the averaging procedure reduces resolution by arbitrarily dividing a spectrum into one thousand channels (or more as chosen) thereby limiting the resolution to one part in a thousand. Worse yet, successive scans cause a certain small amount of smearing of the spectrum because of nonhomogeneities. There are different ways of enhancing resolution sensitivity in a spectrum resulting either from a single scan or from average multiple scans. One technique involves the application of point by point weighting functions to the spectrum in order to enhance distinctions between peaks. Another technique is the application of the second derivative weighting. This may also be used to enhance the differences between peaks or to bring out separate peaks where only a small shoulder may exist. Both of these techniques enhance resolution at the expense of sensitivity; that is, the height differences between the peaks become less meaningful. Another potential technique for computer application is the removal of fast passage ringing, an effect observed when spectrum is scanned too fast and trangents occur at the end of the peaks. It should be possible to subtract a function from the spectrum to eliminate this ringing which would have the effect of increasing resolution. It is one method of improving resolution which is not necessarily of the expense of sensitivity. Precise NMR work requires the accurate determination of chemical shifts and coupling constants. Programs have been written for larger computers which take the chemical shifts and coupling constants for the positions of the peaks and spectrum in search libraries of NMR spectra stored on bulk storage and match the sample spectrum against the library spectrum.

However, it is often very difficult or impossible to determine the chemical shifts and coupling constants by looking at the spectrum. For second order spectra, a spectrum synthesis is commonly run. Given a set of chemical shifts and coupling constants one can compute a theoretical spectrum. One approach to the calculation of coupling constants and chemical shifts for a second order spectrum would be to assume a set of shifts and coupling constant values and compute a theoretical spectrum to be compared with an experimental one. On a second pass the chemical shifts and coupling constants would be adjusted until the theoretical spectrum matches the experimental one. At this point one has determined the proper chemical shifts and coupling constants. This could be most helpful. A totally different area for computer applications to NMR would be the pulsed NMR applications. Pulsed NMR can shorten the time required for acquiring information to orders of magnitude in some cases. Also signal to noise enhancement can be improved by factor 10. However, the data from the pulsed NMR experiment must undergo a fourier transformation to look like the conventional spectrum obtained by high resolution NMR. With a dedicated computer system, pulsed NMR could become more valuable than the scanning type where the computer would be involved in doing a transformation, storing the resulting spectrum, and repeating this at short time intervals adding each transformed spectrum to those totalled before.

In this case, again the computer is making the instrument a more valuable piece of equipment.

# REAL TIME ACQUISITION AND DISPLAY OF MASS SPECTRA*

P. D. Siemens
Lawrence Radiation Laboratory, University of California
Livermore, California 94550

## ABSTRACT

A program package has been developed to perform real time data acquisition and display from a mass spectrometer. In this particular case the data acquisition routine performs multisumming-scaling, but with minor changes the package could do signal averaging or pulse height analysis.

Through a keyboard monitor, the operator has complete control of the experiment with a variety of commands available to him. Among these are commands which provide for: control of the data acquisition, real time log or linear displays, data output on paper tape, teletype, DEC tape or Calcomp, and data reduction (peak stripping and the calculation of isotope ratios).

In our Laboratory, we are just completing the tune-up procedure on a new mass spectrometer. This spectrometer will be used to determine isotopic ratios of trace amounts of heavy elements. A small computer was chosen to be the data acquisition device for several reasons, some of which will be explained in this paper.

By way of introduction, this spectrometer is a three-stage instrument with two magnetic sections and one electrostatiç section. Ions, produced by a hot filament, are sorted by their mass/charge ratio in the magnetic sections, while the electrostatic section functions as an energy selector. The magnetic field is swept with a triangular wave over a range of a few hundred gauss, with the center field ranging from essentially zero to 12 kilogauss.

Each half of the sweep is divided into 128 time slots or channels (each time slot is generally about 2 msec long, corresponding to a sweep rate of approximately two sweeps per sec). During each channel time, the number of ions striking the detector are counted. If, during one sweep, the number of ions counted for each channel time are stored in sequential core locations, the resulting array is a spectrum whose peaks represent the abundances of the various isotopes present in the sample. The signal-to-noise ratio of this spectrum is poor, however, so a signal enhancement technique is used in which repetitive sweeps of the spectrum are summed together (commonly called multi-summing-scaling).

A block diagram of the computer system and spectrometer interface is shown in Figure 1. The triangular reference sweep for the magnet sweep control is generated by a 12-bit up-down counter and digital-to-analog converter. The counter is driven from a free running clock that is not under computer control, but the possibility does exist, and is practical, to give the computer control of the sweep generation. A strobe pulse generated every 32 clock pulses is used to load the contents of the scaler register into the buffer register and to cause a program interrupt notifying the processor that data are available. The scaler register is constructed from 100-MHz emitter coupled integrated circuits, but at the present time, the double pulse resolution of the system is limited by the pulse width from the electron multiplier. In addition to these functions, a sync interrupt is generated once per sweep to insure that the spectrometer and the data acquisition program never get out of sync.

As the software for this mass spectrometer system was formulated, several requirements became clear. First was the need for a good, efficient data acquisition routine. Secondly, the operators of the machine required a very good real-time display capability in order that they might be continually aware of how well the spectrometer was functioning. Finally, it became apparent that all control should be exercised through a keyboard monitor. Experience has shown that there are fewer problems if the operator of a laboratory computer system never touches the front panel switches of the computer.

Since the users of this system range in computer experience from fairly complete knowledge to none at all, a simple method of operation is mandatory. Conflicting with simplicity is the requirement for a very flexible software system. This flexibility was gained by giving the user control of as many functions as possible. For instance, 99 times out of 100, a spectrum is allowed to accumulate until a channel reaches 100,000 counts, at which time the accumulation is automatically halted. However, to prevent operator frustration in that 1 case out of 100, this maximum count limit can be easily changed by operator command.

Once the program has been loaded and started, no direct interaction with the computer front panel is called for. All functions of the program are controlled through a keyboard monitor. However, channeling all control information through the teletype imposes a few problems in programming. First of all, even the best typists make mistakes, so the code must have the capability of allowing

*Work performed under the auspices of the U. S. Atomic Energy Commission.

error correcting from the keyboard. Perhaps more important than this is that the code have error checking capability, and gracefully reject commands that are in error. Secondly, the code should allow the user to change his mind. Any operation, once started, should be capable of being stopped when the user desires. For instance, if the user started a type out of the accumulated data (a process that takes approximately five minutes) and on the basis of the first few channels concludes that the spectrum is not good, there is no reason why the program should force him to wait while the rest of a useless spectrum is typed. In short, one of the important aspects in writing a program that the user will gladly use is to provide him with complete control over every feature of the experiment. The highly touted flexibility of a digital computer is lost unless the software also supplies this flexibility.

The primary task of the software, of course, was the acquisition of the mass spectra data from the spectrometer. A flow chart of the acquisition routine is shown in Figure 2. As can be seen from the flow chart, there are two entry points to the acquisition routine; the sync pulse entry is used once per sweep to synchronize the beginning or ending of the accumulated data mode, and the data flag entry point is used 255 times per sweep. The data are stored in two arrays, the real time data in RT (single precision) and the accumulated data in CHAN (double precision). The data are continuously deposited in the real time array, which stores the current data from the spectrometer. Only one half sweep of data is stored (128 data points) and this is being continuously overwritten by incoming data. A display of this array allows the operator to continuously monitor the performance of the spectrometer, and is the primary display used during a tune-up procedure.

When the STATE flag is in the ACCUMULATE DATA mode, the data are added to the CHAN array, building the desired mass spectrum. Note that when the user commands a start or stop in the data taking, STATE is set to a 1 or 2 respectively. The acquisition routine then synchronizes the start or stop of data acquisition with the beginning or ending of the sweep.

For reasons which are unimportant for the purposes of this paper, the data acquired during the up and down half-sweeps are treated as two separate spectra. If the data are stored sequentially in-core as they come from the spectrometer, the display of the data would show the down-sweep spectrum as the mirror image of the up-sweep spectrum. Normally, however, one would desire to see the low mass data on the left of the display. This was accomplished by having the acquisition routine store the data from the beginning of the data array up for the up-sweep and from the end of the data array down for the down-sweep.

One final problem with the acquisition process should be mentioned. The data from the spectrometer appear at an absolute synchronous rate, placing quite a restriction on the length of time spent in any interrupt service routine.

The display, of course, is one of the major reasons for using a computer in this application. What makes the computer-driven display better than the previous display used? First of all, the computer display is entirely flicker-free, while the previous analog displays were driven at the same rate as the magnetic sweep—1 to 2 sweeps per second. Secondly, scaling of the linear display is under keyboard control. The full scale value of the linear display can be varied from a value of $2^{22}$ to $2^{3}$ in powers of two. Third, a flicker-free logarithmic display with or without decade lines is available. The data compression afforded by this type of display has proved to be invaluable, but the price paid for the logarithmic display is very minimal, as the routine occupies less than 20 locations and approximates a logarithm in 35 μsec.[1] Finally, the computer-driven display is capable of expanding a selected number of channels so the user can examine them in more detail. With this expand mode, the user can set a marker at any desired channel, and then expand plus or minus N channels on either side of the marker. In addition, there are two commands to move the marker at approximately one channel per second to either the right or the left and stop when desired.

The display routine, diagrammed in Figure 3, is essentially the main program, as other routines are entered only to service interrupts. The display loop is coded for maximum speed, because in striving for a flicker-free refresh rate of 256 channels every 30 msec, one must remember that every cycle in the loop accounts for more than 1% of the refresh time. For a flicker-free display of 256 channels, then, the loop to display one data point must consist of less than 100 machine cycles. One of the techniques used to achieve this speed is in the area of decision branches. As can be seen from Figure 3, there are three decisions to be made within the main display loop. These decisions could be made by putting the appropriate flag in the accumulator and then executing a skip instruction, but this procedure requires a minimum of three machine cycles. However, the decision can be done in single cycle by depositing the appropriate JMP instruction (this is essentially address modification) in the decision location. This technique alone saves more than 2 msec in the display of 256 channels of data.

Basically, this program is input-output oriented. It can be viewed as a collection of asynchronous program segments that can make conflicting input-output service requests. To resolve these conflicts, a basic input-output processor was written to schedule the response of the requests in an orderly manner.

A request for I/0 service is made by transmitting to the Service Request Routine a Service Request Block (see Figures 4 and 5). The Service Request Block consists of three words: a data word to convey needed information to the requested routine, a device number specifying the I/0 device, and the starting address of the desired service routine. The Service Request Routine stores this block of data in the Service Request Storage Queue, and then transfers control to the Service Start Routine. This routine sequentially examines Request Blocks in the Storage Queue, and by checking the appropriate Device Busy Flag (a software flag) determines if the requested device is already engaged. If it is busy, the service start routine examines the next entry in the queue. If it is not busy, the appropriate Device Busy Flag is set, and the data word is transmitted to an I/0 service initialization subroutine designated by the starting address in the request block. When this requested service has been completed, the End

[1] P. D. Siemens, "A Fast Approximation Method for Finding Logarithms." UCRL-71344, 1968. Also to be published in DECUSCOPE, Vol. 7, No. 5.

Service Routine is entered, which clears the Device Busy Flag and then transfers control to the Service Start Routine. In this way, maximum throughput to the I/0 devices is achieved.

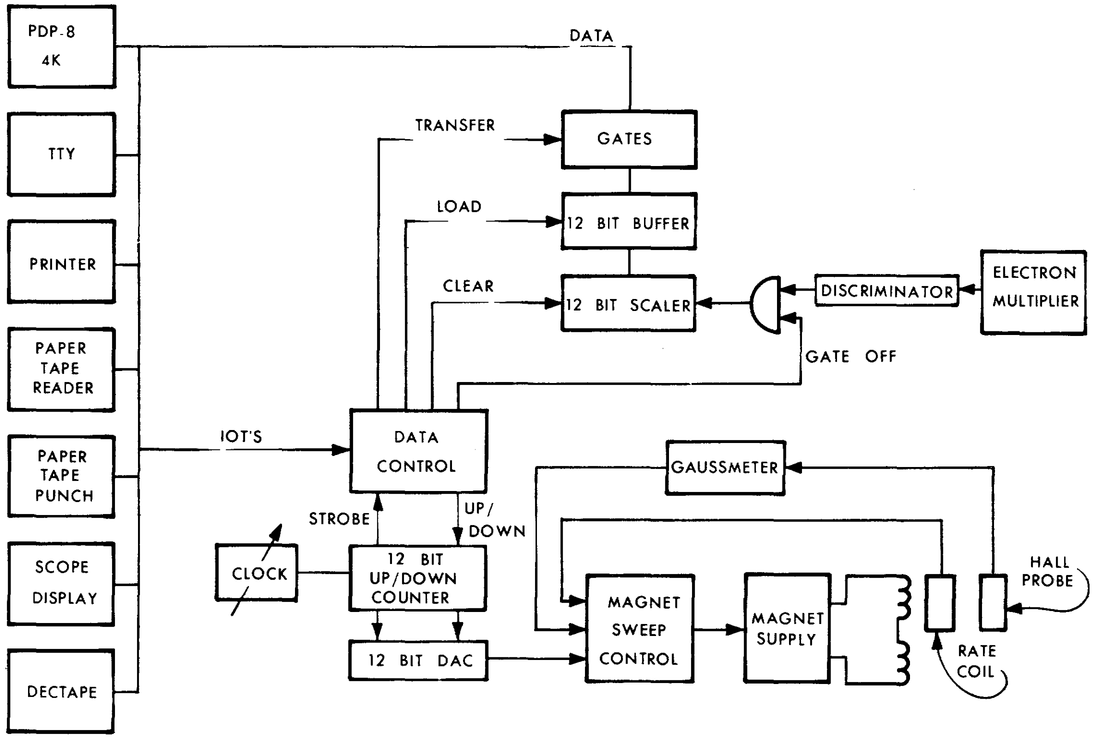
There are several other I/0 operations that have not been mentioned yet. For instance, a spectrum can be punched or read from paper tape, and written or read from DECtape. Also, a spectrum can be printed on a printer (now an ASR 33 teletype, but soon a 100 character/sec printer). Here again, an effort was made to present the data in a very understandable format. To achieve this, the data and channel number were listed in a four-column format with leading zeroes suppressed.

It should be noted that the display and I/0 routines are fairly general, and whether the function that is performed is multi-summing scaling, signal averaging, or pulse height analysis depends on the data acquisition routine.

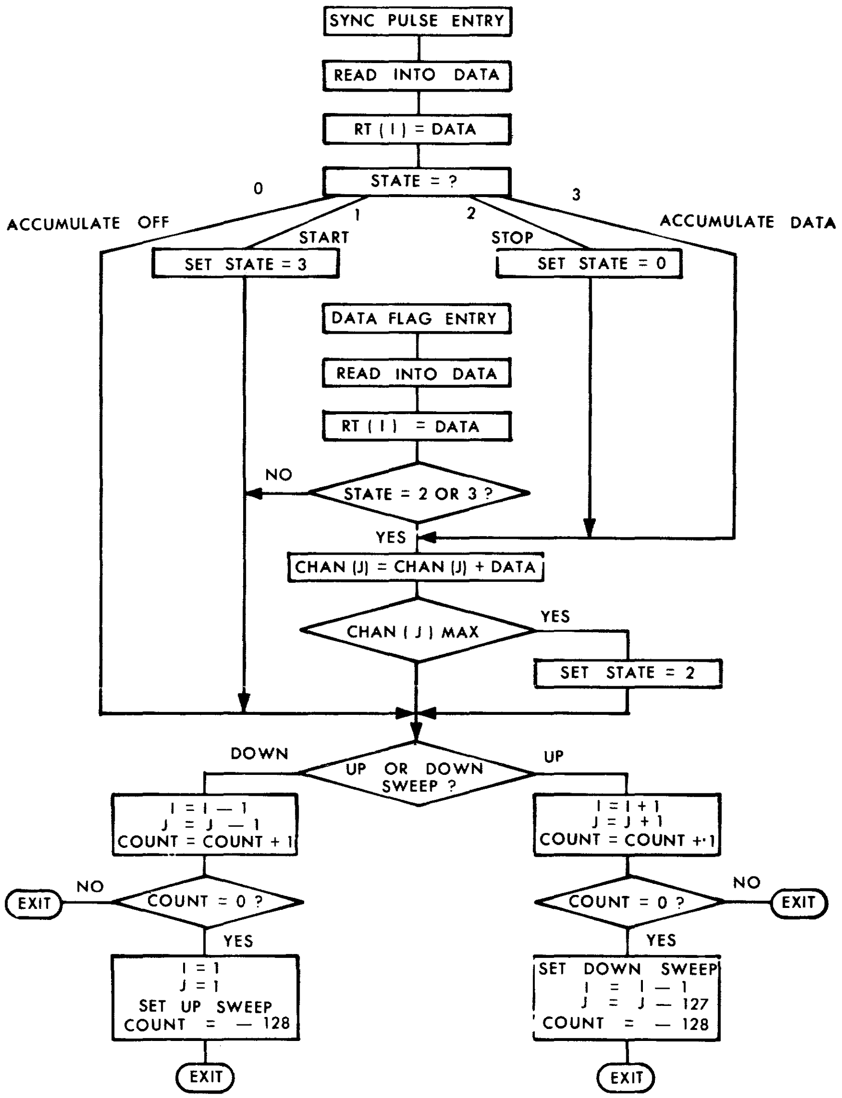
While much useful data can be obtained with these routines, the software development is not yet complete. Yet to be integrated into this package are the on-line calculational routines to furnish the user with the isotopic ratios immediately after a spectrum has been accumulated.

Acknowledgments

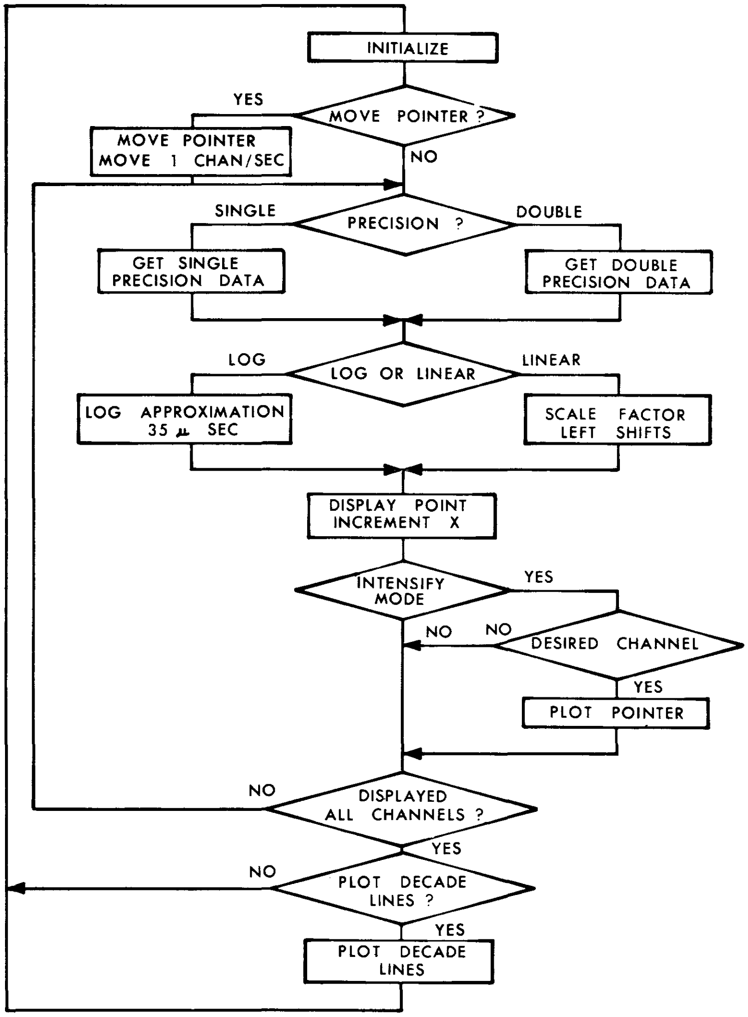
Acknowledgments are due to R. L. Peterson for his constructive criticism of the software and the design of the spectrometer interface, and to R. J. Dupzyk for furnishing the users point of view on the operation of the system.



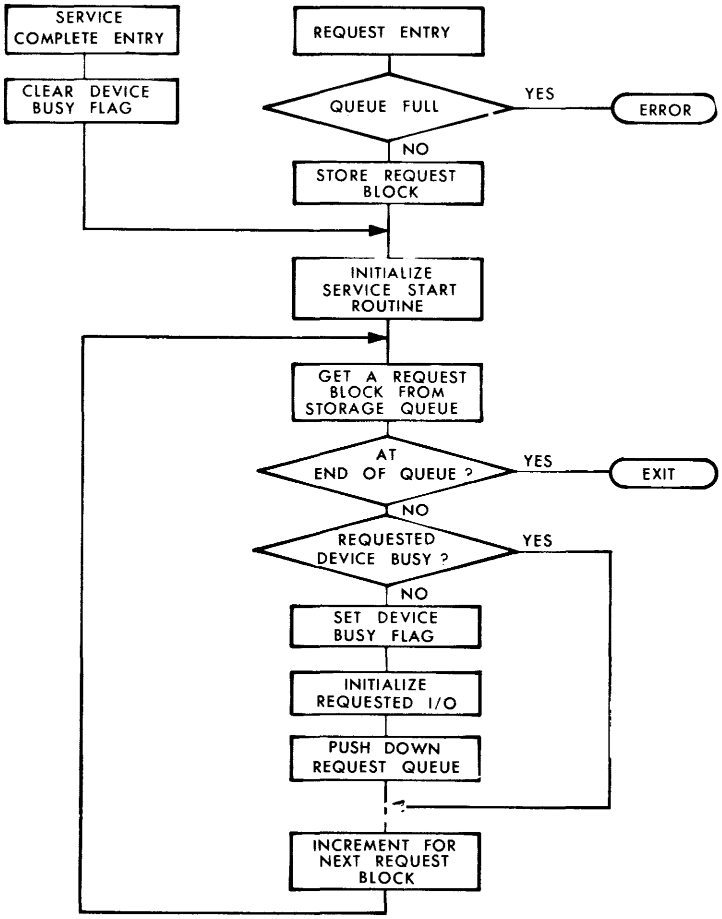
Siemens - Fig. 1



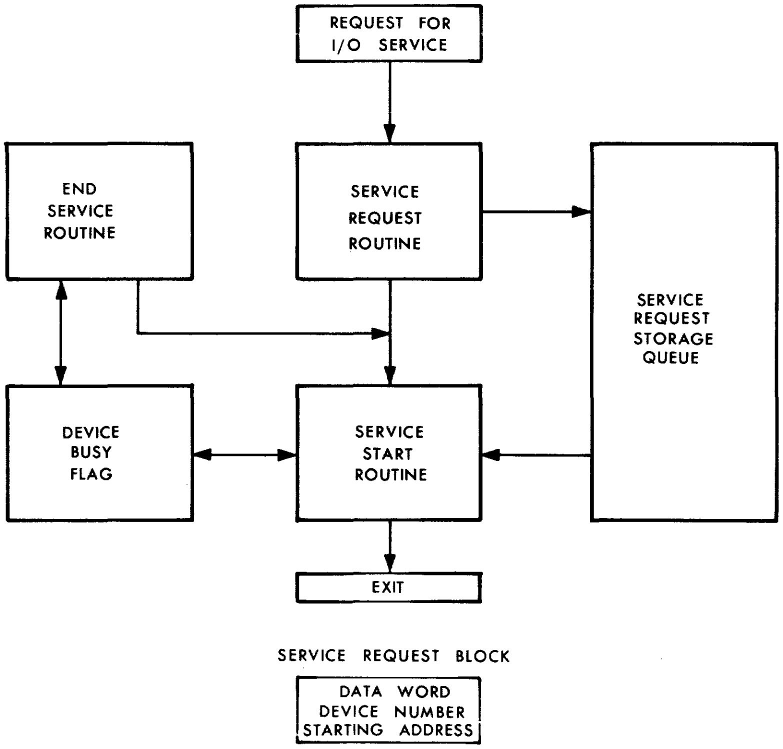
Siemens - Fig. 2



Siemens - Fig. 3



Siemens - Fig. 5



Siemens - Fig. 4

# ON-LINE DATA REDUCTION FROM CARY 14, 15, AND 60 SPECTROMETERS[*]

Martin S. Itzkowitz[†] and Barrett L. Tomlinson[‡]
Lawrence Radiation Laboratory, University of California
Berkeley, California

## ABSTRACT

A 4K PDP 8/S computer with ASR-33 teletype has been installed in this laboratory as the heart of an on-line data-reduction system for the Cary 14, 15, and 60 spectrometers. Data flow is from the spectrometer through a set of Datex mechanical encoders, through an interface designed to our specifications by Berkeley Scientific Laboratories, through the computer, and onto the teletype both as printout and binary punchout. The software system includes a rapid averaging algorithm to eliminate high-frequency noise, a sliding thirteen point least-squares curve fitting, a fully buffered I/O system, and a versatile monitor which virtually eliminates the possibility of unrecoverable operator error.

## HARDWARE

A 4K PDP 8/S computer with an ASR-33 teletype has been installed in this laboratory as the heart of an on-line data reduction system for Cary 14, 15, and 60 spectrometers. An interface, designed and built by Berkeley Scientific Laboratories, has been connected to the computer I/O bus so that the computer can sense the status of six input devices. Each of the six devices is assumed to be a set of twelve binary switches. A special set of two switches is connected to device 1 of the interface such that a change in status of either of these switches causes the interface to set the program interrupt flag for the PDP 8/S. Currently a Datex 20-bit shaft encoder, type 22-300-14, which is mechanically linked to the wavelength counter of the spectrometer is connected as devices 1 and 2. The two-switch Gray code of device 1 is connected to a two-bit encoder in the Datex shaft encoder, so that any change in the wavelength encoder status causes the program interrupt flag to be set. A Datex single-turn, 12-bit encoder, type 03-005-1, which is mechanically linked to the pen of the spectrometer, is connected as device 3. The pen and wavelength encoders are part of the standard Cary Digital System, and may be connected interchangeably with that system. Device 4 of the interface is connected to a panel of 12 switches.

## SOFTWARE

A library of programs has been written to acquire and process data from Cary spectrometers using the above system. Called "SUPER SPECTRUM," the operating system features fully buffered teletype input/output[1] and a monitor program which permits control to be transferred at will between the various subprograms. Program control is by means of one-letter commands given from the teletype keyboard, and by settings of the computer-console switch register. Provision has been made for future expansion of this operating system. Routines may be added or deleted from the monitor calling program by means of one-word patches. Compatibility with the current program requires that future additions to the system use program interrupt and use "JMP ." rather than "HLT" instructions for delays. Teletype input is achieved by calling the subroutine IN, which returns with the next character in the teletype input buffer in the accumulator. Teletype output is done by calling the OUT subroutine with the character to be printed in the accumulator. The output buffer is emptied and the input buffer is cleared upon entry to the monitor.

Upon entry, the program first calls for the starting and ending wavelengths, the wavelength increment between data points, and the interval over which averaging is to be performed. These and various scaling and identification parameters are entered via the keyboard. Additional control information is entered in the switch register. The keyboard is continually examined, and control is transferred to the monitor whenever the character "←" is typed. From the monitor any of the following routines may be called:

1. Baseline start
2. Spectrum start
3. Data read (from paper tape)
4. Data punch (to paper tape)
5. Backup and retake part of spectrum
6. Clear baseline is zero
7. Echo keyboard
8. Print pen encoder
9. Print wavelength encoder.

The existing program is designed to average the spectrometer pen reading 4096 times or for a specifiable wavelength interval, whichever is less, and store the result internally as a 12-bit integer. The averaging interval is centered about the wavelength to which the averaged value nominally corresponds. The averaging procedure is repeated at specified wavelength intervals within a specified wavelength range. The averaging algorithm used

[*] This work was done under the auspices of the U. S. Atomic Energy Commission.

[†] Speaker.

[‡] Present address: Department of Chemistry, University of Oregon, Eugene, Oregon.

is a modified "stable averaging,"[2] where the average M after taking m readings is

$$M_m = M_{m-1} + \frac{f(m) - M_{m-1}}{2^N},$$

where

$$2^{N-1} + 2^{N-2} < m < 2^N + 2^{N-1} + 1,$$

and where f(m) is the value of the $m$th reading of the signal. Stable averaging is used rather than simple arithmetic averaging, because it can be programmed to execute more rapidly and because it attenuates noise almost as efficiently. The average is computed in double precision using integer arithmetic (24 bits); however, only the most significant 12 bits are kept for later use. Data may be stored as either a spectrum or a baseline, and baselines and spectra may be taken in any order.

Using the output buffering programming, data may be printed on the teletype concurrently with data acquisition, at the experimenter's option. Several output options are available. In addition to printing the wavelength and raw data, a sliding 13-point least-squares average may be applied to the difference of the spectrum and baseline raw data. The resulting smoothed data may be multiplied by a specifiable constant and printed. The difference between the smoothed value and the scaled difference of spectrum and base line raw data may be computed and printed.

The algorithm used in the sliding 13-point least-squares averaging is due to Savitzky and Golay.[3] It is of the form

$$q_0 = \frac{1}{a_7}[a_6(p_{-6} \pm p_6) + a_5(p_{-5} \pm p_5) \cdots$$

$$+ a_1(p_{-1} \pm p_1) + a_0 p_0],$$

where $a_0$ to $a_7$ are constants, $q_0$ is the smoothed value, $p_i$ the raw value, and the signs used in all the inner parentheses are the same. The constants used in the program perform a least-squares fit of 13 points to a third-degree polynomial. By simply changing the constants by means of a patch, the program may be used to calculate the $n$th derivative using any odd number of points less than or equal to 13, or any odd number of points less than or equal to 13 may be used to fit an $n$th-order polynomial, where $n$ is an odd integer less than or equal to the number of points used in the fit.

Data may be preserved for later processing by having a paper tape punched containing the control information and the data. If the last data taken before punching is a baseline, the data tape contains the raw baseline data values. If the last data taken is a spectrum, the data tape contains values of (spectrum - baseline) E/OD, where E and OD are constants specified by the operator at the start of each baseline or spectrum.

A routine exists to read the data tapes back into the spectrum baseline data arrays, and another routine permits processing and printing of the data arrays at any time. Programs to print the wavelength and pen position for calibration and test purposes may be called.

Mistakes while acquiring data can be corrected by calling an error-recovery program which enables the operator to retake a portion of the data under carefully defined circumstances.

The program is loaded into the computer memory from a single paper tape punched in Digital Equipment Corporation binary loader format (Digital 8-2-U). It consists of four blocks of code separated by short sections of leader (200 code punches). The first block of code modifies the binary loader to read tape continuously, unless a checksum error is detected. The second block is a copy of Floating-Point Package II (as released by DEC). The third block is the Super Spectrum system, including patches to the Floating-Point package. The final code block restores the binary loader program to its original form, so that the loader halts after reading the final checksum. This means that to the user the tape acts as if it were a single binary tape, and is loaded by following the instructions for the binary loader (Digital 8-2-U).

Great care has been taken to ensure the integrity of the program during execution. To our knowledge once the program is loaded and running, and the console panel is locked, the program can not be destroyed by any operator action except turning the power off while the computer is running. The program has been used for several months without incident by about five operators, most of whom regard the programmed computer as a "black box." The program and data storage occupy all of memory.

## ACKNOWLEDGMENT

We would like to thank Dr. David McMullen for many helpful suggestions during the early stages of this work.

## REFERENCES

1. W. L. Van der Poel, Decuscope, No. 5, 21 (1967).
2. J. E. Deardorff and C. R. Trimble, Hewlett-Packard Journal 19, No. 8, 8 (1968).
3. A. Savitzky and M. J. E. Golay, Anal. Chem. 36, 1627 (1964).

# DATA ACQUISITION AND CONTROL OF A SPECTROPHOTOFLUORIMETER

Robert H. McKay
Department of Biochemistry and Biophysics, University of Hawaii
Honolulu, Hawaii

Frank Neu and Myron Myers
Lawrence Radiation Laboratory, University of California
Berkeley, California

## ABSTRACT

An interface between a spectrophotofluorimeter built in this laboratory and a standard PDP-8/S computer is described. The interface utilizes DEC modules, stepper motors for wavelength and polarizer positioning, and a Hewlett-Packard #2401-C integrating digital voltmeter for data acquisition. Light source fluctuations are controlled using a monitor on the incident light, a v-f converter, and feeding this in as an external time base for the DVM.

The instrument is capable of obtaining either corrected fluorescence excitation spectra or polarization spectra, with a data collection interval as low as 0.5 nm.

Software developed for this application is described briefly.

## INTRODUCTION

Fluorescence, the immediate (within 1n-sec) emission of radiation from a molecule or atom following absorption of electromagnetic radiation is of increasing importance in biochemistry and medicine. To be really useful, these measurements must be accurate and reproducible to 1% or better. This is more difficult to achieve in fluorometry than in spectrophotometry for a number of reasons, such as:

(1) The fluorescence is proportional to the intensity of the exciting light.

(2) The fluorescence is proportional to the fractional absorbance of the emitting species.

(3) The fluorescence is dependent on the quantum yield of the compound.

(4) The fluorescence is strongly temperature dependent.

Since both the intensity of the exciting light and the absorption spectrum of the fluorescent emitter are strongly wavelength dependent, and the quantum yield of fluorescence varies from a small fraction to nearly 1, we see the need also for a wide dynamic range in the instrument. Special demands are placed on the instrument in the 200-350 nm region, since here both the light source intensity and the phototube response are falling drastically, however, it is precisely in this region that many biological compounds absorb and fluoresce.

Of the many types of fluorescence measurements, only two will be discussed here, these are: (1) The fluorescence excitation spectrum (defined as the variation in the fluorescence intensity as a function of the exciting wavelength, and (2) the fluorescence polarization spectrum, defined as the variation in the polarizability (P) of the fluorescence as a function of the exciting wavelength. P is defined as $I_V-I_H/I_V + I_H$ where $I_V$ and $I_H$ are the vertical and horizontal components of the emission, viewed at 90° to the exciting light.

The relatively low values of P usually obtained (0.5 to -0.33) means that you are essentially taking the difference over the sum of 2 large numbers, and this tends to magnify the effects of noise and other errors, especially as P approaches zero. Thus we need a very stable instrument. It is especially important that the lamp energy be constant during $I_V$ and $I_H$ measurements at each wavelength.

Because of the extreme demands for precision, accuracy and stability and the tedium of doing these measurements manually, we have turned to computer control of the instrument.

## DESCRIPTION OF THE INSTRUMENT

Figure 1 shows the overall optical configuration. Light from the 900 W Xenon Arc is rendered monochromatic, a fraction of this excitation beam is passed to a monitor photomultiplier through a fluorescent screen. The purpose of the screen is to eliminate the wavelength dependence of the photomultiplier sensitivity. The beam then passes through an optional polarizer and then to the sample cell, which is temperature controlled. The fluorescence, emitted in all directions is sampled at 90° to the direction of excitation to minimize stray exciting light. The fluorescence beam passes through a rotatable polarizer, a filter to exclude exciting light and finally to a photomultiplier.

Figure 2 shows the general electronics configuration. Special circuits were designed to level shift the digital voltmeter (DVM) logic signals to DEC logic levels. All other circuits and boards are standard DEC modules. Three sets of 12 bit word gates are provided for the computer to control in reading the DVM output data. Several external interrupts are available to signal the computer for DVM "ready", wavelength limit switches, etc. Two logic bins, self-powered and capable of holding 96 flip-chip modules were required for the interface.

In order to achieve corrected excitation spectra, the monitor photomultiplier signal is converted to a frequency and used as an external time base to the DVM. This results in a constant energy monochromator at all exciting wavelengths, and, in addition, cancels out lamp energy fluctuations occurring at constant wavelength. Alternatively, the monitor signal can be integrated at the same time as the fluorescence signal, and the ratioing done in the computer.

Figure 3 shows our primary reduction program for obtaining a polarization spectrum. The final computed data will be typed and punched out or put on magnetic tape. The tape to be used ultimately to supply data to a digital plotter, or be used in other data reduction programs.

The software is organized as a set of modular subroutines linked together by several small main programs, all residing in memory simultaneously. This was done to facilitate easy program restructuring by the experimenter. Most of the main programs are diagnostics, exercising the hardware to varying degrees of complexity. These diagnostics use the relevant production subroutines to drive the external hardware. Since neither memory space nor timing are problems in this application, all programs reside in memory together, and the interrupt facility is not used. The input data from the digital voltmeter is converted to double precision for storage and computational reasons.

"Limit switch conditions" and the "DVM integration complete" status are tied to device flags. All other "timeout" conditions are timed out in the software. These include reed relay settling times on the DVM and stepping motor moving times.

Other data reduction requirements, e.g. correcting the sample spectrum for the solvent contribution, correcting the fluorescence emission spectra for the wavelength dependence of the photomultiplier tube, determining the areas of overlap of absorption and emission spectra, etc. are all much easier, faster and more accurately done with the digital system.

## ACKNOWLEDGEMENTS

This investigation was supported by PHS Research Grant No. GM-11682 and Career Development Award No. GM-8517 (to R. H. McKay) from the Institute of General Medical Sciences, N.I.H.

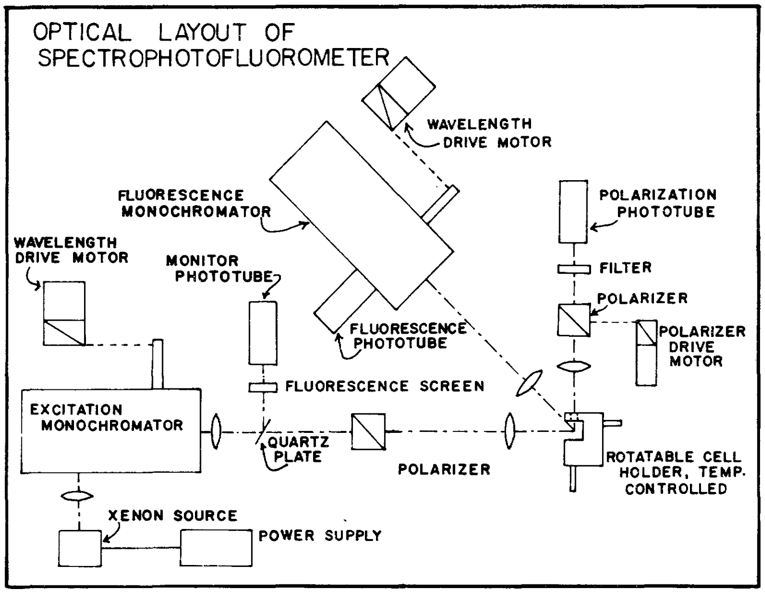


Figure 1 Optical Diagram

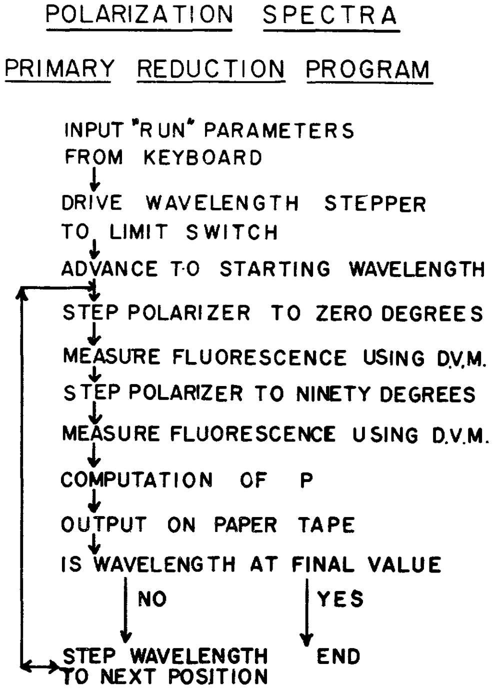

POLARIZATION SPECTRA
PRIMARY REDUCTION PROGRAM
INPUT "RUN" PARAMETERS FROM KEYBOARD
DRIVE WAVELENGTH STEPPER TO LIMIT SWITCH
ADVANCE TO STARTING WAVELENGTH
STEP POLARIZER TO ZERO DEGREES
MEASURE FLUORESCENCE USING D.V.M.
STEP POLARIZER TO NINETY DEGREES
MEASURE FLUORESCENCE USING D.V.M.
COMPUTATION OF P
OUTPUT ON PAPER TAPE
IS WAVELENGTH AT FINAL VALUE
NO
YES
STEP WAVELENGTH TO NEXT POSITION
END

Figure 3 Primary Reduction Program

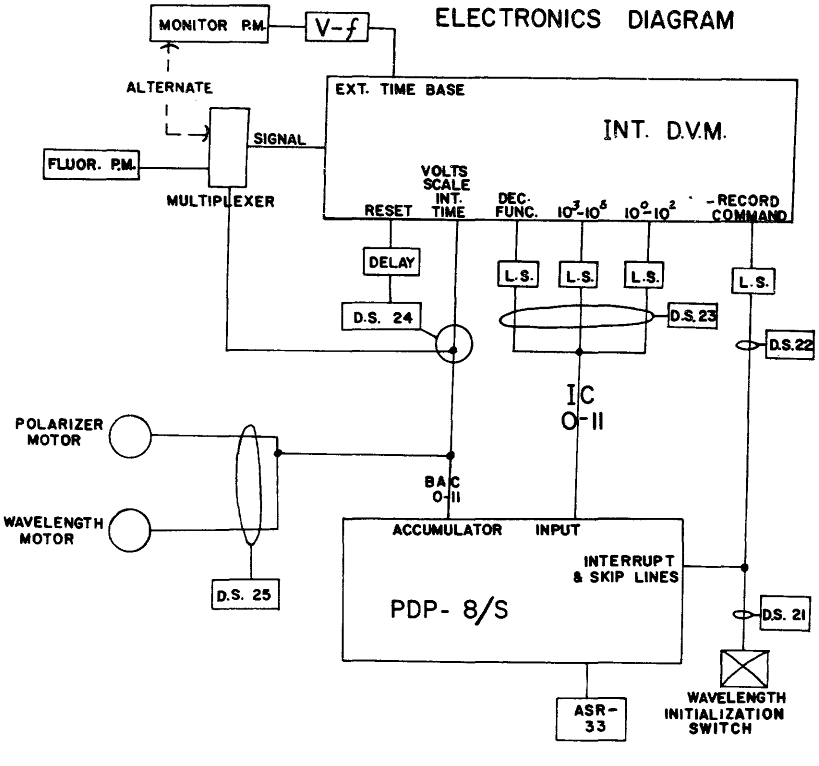


Figure 2 Electronics Diagram

# WEATHER AND A PDP-8/S

T. McGovern and R.E. Archinuk
Whiteshell Nuclear Research Establishment
Pinawa, Manitoba

## ABSTRACT

The existing meteorological system at Whiteshell Nuclear Research Establishment included several instruments for measuring wind speed, wind direction, temperature and radiation. The data were recorded on continuous and multipoint strip chart recorders. The computer system was designed to digitise and reduce the recorded data to enable a more efficient and accurate analysis of the data to be made. This would ultimately provide the meteorologist with the facility and the time for a more in-depth study, of diffusion climatology in the Whiteshell area. The approach is applicable to other meteorological systems.

## INTRODUCTION

The use of weather information and meteorological advice by industry for construction, transportation, marketing and a host of other purposes is well established and constantly increasing. Although from this standpoint meteorology is very useful to the atomic energy industry, it is the potential hazard from airborne radioactive materials and the use of meteorology to help minimise this hazard that has assumed the most importance. The dilution efficiency of the atmosphere on various gases, aerosols and particles depends essentially on wind and temperature gradients. Meteorological techniques for the evaluation and prediction of atmospheric diffusion exist and these have been applied with considerable success.

This paper offers a method of collection and analysis of meteorological data to assist in the definition of a diffusion climatology at Whiteshell Nuclear Research Establishment (WNRE), and describes some of the problems encountered when a small computer is interfaced to an already existing system[1] of instruments and recorders.

Diffusion climatology data at WNRE are obtained from meteorological instruments mounted on a 200 ft triangular steel tower (Fig. 1) and from a climatology station adjacent to the tower. Aerovane transmitters are mounted at 20 ft, 83 ft, and 200 ft levels on the tower. These are connected to three recorders in the Health and Safety Building about 700 ft from the tower. Each recorder is a dual trace recorder which records wind speed and azimuth wind direction at the three levels. Temperature is also measured at the same levels on the tower and at various heights above and below the ground. The signals are taken to the same room and recorded on multipoint recorders.

In addition to these instruments which are installed and operating, the system will be expanded in the near future to include an Anemometer Bivane and a Net Radiometer. The bivane measures wind speed and wind direction, in both the azimuth and elevation planes. The response of this instrument will allow a more detailed study of rapidly changing wind conditions to be made. The instrument will be used in several locations for unspecified periods of time and is connected to two recorders.

The Radiometer will be situated near the tower and used for solar and ground radiation measurements.

With the old system all the information was recorded graphically on strip charts. To perform statistical analysis it was necessary to digitise the charts. This was done manually by an operator, who visually estimated the mean value over a 10 minute interval once every hour, for each variable. Scaling of charts is a tedious and time consuming operation, and it was decided to install a data acquisition system which would automatically digitise the values, calculate the means, and produce output for direct input to a large computer for statistical analysis.

To provide automatic data logging and reduction a Digital Equipment Corporation PDP-8/S Computer was installed. The PDP-8/S is a single address, fixed word length, serial arithmetic computer using a word length of 12 bits and two's complement arithmetic. Cycle time of the 4096 word magnetic core memory is 8 μs. Standard features of the system include indirect addressing and facility for instruction skipping and program interruption as functions of input/output devices.

In addition to the central processor the following peripherals were installed:

(1) A standard teletypewriter for use in a conversational mode to initialize the system and to print out results.

(2) A high speed paper tape reader for use in editing, assembly and debugging programs as well as an efficient means of inputting the final program.

(3) A high speed paper tape punch for use in assembly and debugging of programs and in outputting results in a form suitable for direct input and analysis on the computer centre's Honeywell H620 Computer.

(4) A multiplexer assembled from DEC A Series

modules. Forty channels were installed originally with provision for a further twenty-four.

(5) A Preston Differential Amplifier used to give high accuracy, low noise, good common mode rejection and high input impedance necessary in conditioning the signals.

(6) An analog to digital converter (A/D) assembled from a DEC C002 Panelaid Kit. The converter accepts any analog value from 0 to -10.23V and converts it to a 10 bit digital value.

Figure 2 shows a block diagram of the computer system and Figure 3 is a photograph of the complete installation.

## SIGNAL CONDITIONING

The signals available are shown in Table 1. These signals had to be conditioned to coincide with the specification of the A/D converter. It was decided to standardize on the basis of 100 mV full scale input to the amplifier. The gain setting on the amplifier must then be about 100.

The tower wind speed signals were attenuated by a potential divider. The tower wind direction signals were obtained from transmitting synchros in the aerovanes. Receiving synchros were connected in parallel with the recorder receiving synchros and a continuous rotation potentiometer was mounted on the shaft of each synchro to convert the shaft position to a 0-100 mV dc signal.

The range of the soil temperature signals was -50 mV to +50 mV. To allow this signal to be interfaced with the computer, it was necessary to connect a 50 mV dc power supply in series with the signal to convert the range to 0 mV to 100 mV.

All three bivane signals required only a simple potential divider for conditioning.

The radiometer signal required a 10 mV dc power supply in series with the signal for the same reason as the soil temperature signals.

All signals required longtime constant filtering to remove 60 Hz noise due to cable pick-up and recorder effects.

The three tower temperature signals from 200 ft, 83 ft, and 20 ft positions on the tower were taken to a Honeywell recorder. The recorder contained additional circuitry to provide the three temperature differences associated with the three temperatures. The available signals were:

74.2 mV - 105.93 mV

equivalent to: $-45^{\circ}F$ - $105^{\circ}F$

It was required that we measure the absolute value at 20 ft level to $0.5^{\circ}F$ accuracy, and the differences $(T_{200} - T_{20})$ and $(T_{83} - T_{20})$ to $0.1^{\circ}F$ accuracy.

Experiments were carried out on the signals supplied to the recorder to examine the possibility of interaction between the recorder and the computer system. The recorder plotted one point every 15 seconds, i.e: one cycle of three absolute values and three differences took 1½ minutes.

The computer samples each of three channels associated with tower temperatures once per second. Three problems were encountered:

1) The recorder input impedance in the out-of-balance position was low enough to affect the signal from the tower. The millivolt signal fell by 5% each time the recorder moved to a new point.

2) The recorder multiplexer used mercury-wetted relays which have a make-before-break action during switching. This results in two adjacent multiplexer channels being shorted together or shorted to ground for a period of 20 ms. A spike which appeared on each signal for this period was attributed to the effect of multiplexer switching.

3) The cable transferring the signal from the tower to the computer room was 700 ft long. Noise measurements were made on the signals and it was found that 60 Hz noise exceeded our accuracy specifications. This noise could be attributed to cable pick-up and effects of recorder servo system.

Various solutions to these problems were considered:

a) Filters to remove the noise would have to be designed as 'notch' filters to keep the effects of recorder multiplexer switching to a minimum. Otherwise, the time constant of the filter would extend the period of the spike. The filters would not overcome the effect of the low impedance of the recorder.

b) Disconnecting all three signal lines to the recorder during the computer sampling period was not feasible because of the long time constant of the filters required to remove 60 Hz noise. With a long time constant filter the recorder must be disconnected well in advance of sampling to permit the filter output to reach equilibrium. This results in large errors in the recorder traces.

c) An amplifier on each of the signal lines to the recorder, with a gain of unity, would allow the computer signals to be taken from the input of each amplifier. The amplifier would be used as a buffer to prevent recorder effects from interacting with the computer signals.

An additional problem was that the A/D converter used in the system has a resolution of approximately 1 in 1000. The range of the temperature signals is $150^{\circ}F$, giving a resolution of $0.15^{\circ}F$. This is inadequate to provide difference values to $0.1^{\circ}F$ accuracy by subtraction of the absolute values.

To overcome both the effects of the recorder and the resolution limitations of the A/D converter the method shown in Figure 4 was adopted. The three temperature signals are first amplified by approximately a factor of 50 to bring them to the five volt level. Attenuators at the amplifier outputs reduce the signals to their original levels to drive the recorder. The isolation provided by the attenuators and the low output impedance of the amplifiers reduces the effects of low input impedance or short circuiting of signals by the recorder to negligible levels.

To improve the resolution, the temperature at the 20 ft level, the difference between 20 ft and 83 ft, and the difference between 20 ft and 200 ft are measured directly. Since the range of temperature difference expected is only 50°F, a resolution of 1 in 1000 corresponds to 0.05°F. The Preston amplifier following the multiplexer accepts floating, differential inputs so that the differences can be measured simply by taking a signal from each of the desired temperature signals. Proper choice of attenuation factors provides a range of 50°F for the differences and 150°F for the absolute temperature at the 20 ft level. Three dc power supplies provide zero suppression giving a final range of:

a) -45°F → +105°F corresponding to 0 mV → 100 mV for the temperature at 20 ft, and

b) -15°F → +35°F corresponding to 0 mV → 100 mV for the temperature differences.

## SAMPLING

For the wind speed and direction, tower temperatures and soil temperatures it was required to record a mean value over a 10 minute period, lasting from five minutes before the hour to five minutes after the hour. This is a standard meteorological method. The bivane measurements would be taken for short periods at irregular intervals throughout the year.

It was decided to arbitrarily assign them to the period 25 minutes after the hour until 35 minutes after the hour. These readings could be ignored, or the program changed to vary sample speed and/or period. The radiometer output was required as a mean value over the hour every hour.

To simplify programming one minute of every hour was reserved solely for data reduction and output and during this time the radiometer was not sampled. The loss in accuracy was considered negligible. Figure 5 shows a timing diagram.

A hardware clock was used for control and timing purposes. The clock was run from the 60 Hz main supply to achieve long term accuracy.

An analog mean was considered and rejected on the basis of the long time constant (10 minutes) required and the complexity of an automatic sample-hold-reset switching operation. It is much simpler to sum N samples and divide by N.

It was decided to sample each variable once per second over the defined period. This means taking 600 samples in a 10 minute interval. Obviously, in the case of temperature, 600 samples would not be necessary - especially below the soil. By taking all sensors at the same rate the program is very much simplified.

## SOFTWARE

It was originally hoped that we could use DATAK (for data acquisition) to minimize programming effort. DATAK is a DEC program which "permits a complex program-controlled data acquisition system to be adapted to a particular experimental environment through the use of a highly sophisticated and precise pseudo code. The necessity of extensive machine language programming to meet a particular data acquisition requirement is thereby eliminated"

Unfortunately, DATAK has never been completed and it was necessary to use machine language for the program. Two thousand five hundred instructions of machine language have been written.

The system operated in real time, using the program interrupt facility to allow external conditions to interrupt the computer program. The interrupt could be caused by the clock, the teletype - writer, or the fast punch. When a program interrupt request is made the computer completes execution of the instruction in progress before acknowledging the request and initiating another routine. The interrupt program is responsible for identifying the signal causing the interruption, for removing the interrupt condition and for returning to the original program.

The hardware clock supplied a train of pulses at the rate of 10 Hz. A program was written which included counters to keep count of seconds, minutes, hours and days. The program also contained a subroutine to convert days to months and days; and to keep track of the year, taking into account leap years. The clock interrupted the computer program every 100 ms to test which subroutine the program should be in and whether a change was necessary. Figure 5 shows the manner in which the program was subdivided. At five minutes past each hour, the results were processed and a table (including the time and date) was printed on the teletypewriter. A digital record was output from the fast punch at the same time.

Since each individual sample could not be retained in store until the end of the sample period because of the limited size of the memory, the mean value had to be computed during sampling. Considering the simple case of taking all channels between clock pulses; i.e. in 100 ms, then during period A (Figure 5), it is necessary to sample 18 channels in 100 ms or sample and compute a mean value every 5 ms. A program was written based on a paper in DECUSCOPE[2], to find the running mean using the formula:

$$M_N = \frac{N-1}{N} M_{N-1} + \frac{1}{N} S_N$$

Where: $M_N$ = Mean of N samples
N = Sample Number
$S_N$ = Nth Sample

This program took 30 ms to run on the PDP-8/S and would obviously not fit the requirement.

A much faster method of obtaining the same result was found by storing a running sum instead of a running mean. Since 600 x $2^{10}$ is less than $2^{24}$ the sum of six hundred samples can be held in two 12 bit computer words. The division by 600 to obtain the mean can be carried out just prior to outputting the data.

The program to obtain the running sum is as follows:

```
STORE;   Ø
         CLL
         TAD I DTLO  -  Add C(A) to low word
         DCA I DTLO  -  Store in low word
         SZL         -  Test for overflow
         ISZ I DTHI  -  Yes, incr. high word
         ISZ DTLO    -  No, carry on
         ISZ DTLO    )
         ISZ DTHI    )  Update pointers
         ISZ DTHI    )
         JMP I STORE )
```

This took less than 0.5 ms to execute on the PDP-8/S.

The arithmetic consisted of dividing by the number of samples taken and converting the result to engineering units (mph, °F, etc.). This involved multiplication factors and scale shifting.

An interesting program was developed to obtain the mean value of wind direction. This reading can only be interpreted for small changes in the direction of wind. Wind direction measurements are made on an angular scale, north being defined as 0°, east as 90°, south as 180° and west as 270°. If the wind direction is fluctuating about a mean value of north, individual readings will be either near zero or near 360°, and a simple summation will not give a correct sum or mean. For example, if one measurement is 10° and a second is 350° the mean is 180°, an obviously incorrect result. Vector addition could be used to produce the correct value but a simpler method was developed. The real difference in direction between successive readings is always less than 180°. If the measured difference in direction is greater than 180° we assume the direction has shifted through north. The following technique is used to obtain a correct sum.

The first measurement of the set is used directly as the first term of the sum, and is also saved in a separate memory location (A) for later use. For each subsequent reading in the set, the new reading is compared with the first reading and if the absolute value of the difference is less than 180° it indicates that the direction has not shifted through north. In this case, the new reading is simply added to the sum. If the absolute value of the difference between the first reading and the new reading is more than 180° we assume a shift through north has occurred. If the new reading is larger than the first reading, we subtract 360° from the new reading and add the result to the sum. If the new reading is smaller than the first reading we add 360° to the new reading and add the result to the sum. These calculations are carried out on each reading in the set.

Because of the possible addition or subtraction of 360° from each reading the final sum may be one of the following:

a) negative: in this case the program adds 360 x 600 to the sum, producing a positive value and a positive mean.

b) positive and less than or equal to 360 x 600: this gives a mean between 0° and 360° and no further action is required.

c) positive and greater than 360 x 600: this results in a mean of more than 360° so the program subtracts 360 x 600, again giving a mean between 0 and 360°.

Figure 6 is a flow chart of the program.

## CONCLUSIONS

The digital output eliminates the time consuming labour of chart scaling and the delay between information being obtained and analysed. The computer system replaces the eight recorders producing 25 traces continuously.

The paper tape can be directly input to the Honeywell H620 Computer in the computer centre and used for statistical studies over long periods of time. These studies are necessary to assist in the definition of a diffusion climatology for WNRE. For example, graphs of mean hourly vertical temperature difference, annual and temporal wind roses (polar co-ordinate frequency distribution diagrams) and monthly cumulative frequency distribution of inversion days can be plotted automatically.

The system was programmed entirely in machine language since no other alternative was available. While machine language programming results in efficient use of memory storage, it is difficult and time-consuming to implement. There is a need for higher level languages to reduce the effort in developing on-line, real time systems. DATAK appeared to be a step in this direction and it is unfortunate that it was never fully developed.

Costs of computer hardware and particularly memory storage are dropping rapidly and it is becoming more attractive to install additional hardware to reduce the programming effort. Higher level languages that provide both mathematical facilities as well as the ability to handle data acquisition functions in real time are urgently needed.

One function of the Assessment, Computing and Instrumentation Branch of WNRE is to design and develop special small computer systems for other branches where these are not available on the market. It is often found that once a user becomes familiar with a system, he becomes aware of the possibilities of developing the system further. To allow the Instrumentation group to concentrate on the development of new systems, we decided to have a member of the Meteorology Section trained in machine language programming.

Already modifications are being considered to increase the usefulness of the computer and the efficiency of meteorology data analysis. A 'spot check' feature is being planned to allow the meteorologist to read any sensor or any instrument attached to the computer immediately. This partially eliminates the need for recorders. On the wind direction measurements, it would be useful to have the facility for examing the variations in direction over a few minutes to provide information on gusting. This would be possible with a short program.

Ambient radiation is at present monitored at five stations, all situated at a two mile radius from the plant. By the use of telemetry this information could be transmitted to the meteorology laboratory and input directly to the computer. This information would then be correlated with meteorological information, e.g. wind direction, to identify and quantify diffusion parameters.

The Meteorological Service of Canada (M.S.C.) have shown an interest in the project. Since they have many similar climatology and atmospheric pollution interests it would appear that this approach would enable information to be obtained and immediately transferred via telephone lines to a data centre. This would allow analysis and prediction to be based on very up-to-date information. The result would be a faster, more accurate and detailed assessment of current air pollution conditions. Diffusion measurements may also be used in the future for the study of propagation of plant and animal pathogens, which would be very useful in a country so involved with agriculture.

## REFERENCES

1. The Micrometeorology of the Whiteshell Nuclear Research Establishment at Pinawa, Manitoba, Pre-Operational Survey Report, A. Reimer, AECL-2620, Atomic Energy of Canada Limited, November 1966.

2. A Survey of Methods for Computing Means, Standard Deviations, and Correlation Coefficients on the PDP-8 Computer, August E. Sapega Decuscope Vol. 6 No. 3, 1967.

## ACKNOWLEDGEMENT

The authors wish to acknowledge the very direct assistance in the problems of programming and commissioning the system given by E.E. Wuschke, Head of Instrumentation, and in the many discussions on the design held with him. In addition, the authors wish to thank A. Reimer of Environmental Control for his many helpful suggestions and discussions on the climatology aspects of the system and for his unlimited patience throughout the commissioning period.

## SIGNAL CONDITIONING

| SOURCE | SIGNAL | INTERPRETATION |
|---|---|---|
| Tower Wind Speed | 0 - 10.56 V | 0 - 100 mph |
| Tower Wind Direction | Synchro | 0 - 360° |
| Tower Temperatures | 74.2 mV - 105.93 mV | -45°F - 105°F |
| Soil Temperatures | -50 mV - +50 mV | -50°C - +50°C |
| Bivane Speed | 0 - 0.21 V | 0 - 50 mph |
| Bivane Azimuth Direction | 0 - 5.4 V | 0 - 360° |
| Bivane Elevation Direction | 0 - 1.0 V | -50° - + 50° |
| Radiometer | -10 mV - 40 mV | Not Available |

TABLE 1 Signal Conditioning

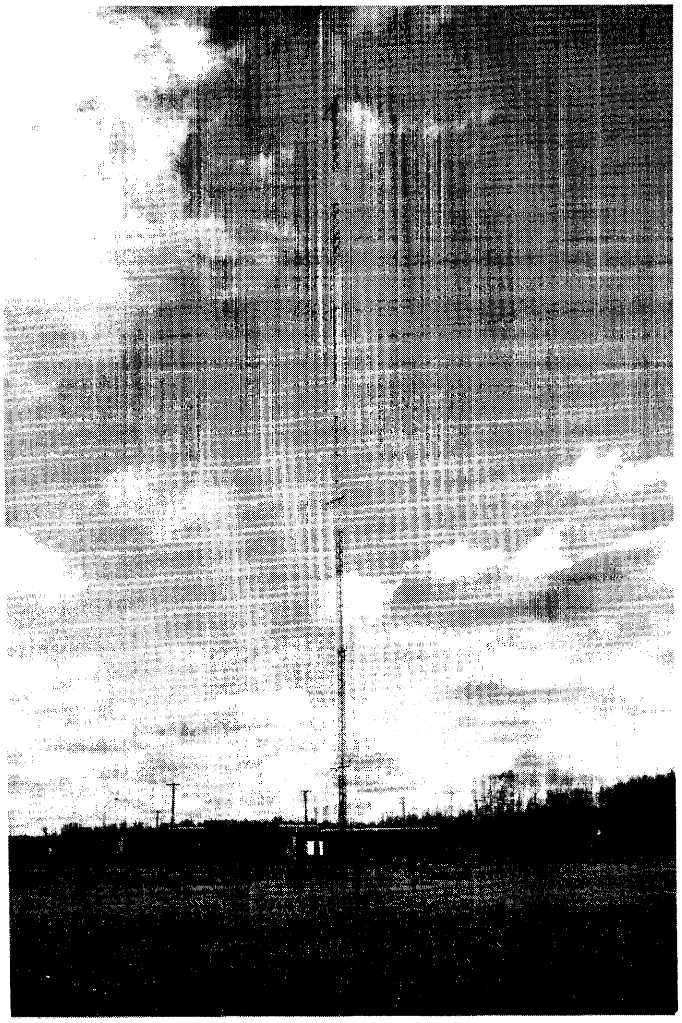


Figure 1 Meteorology Tower

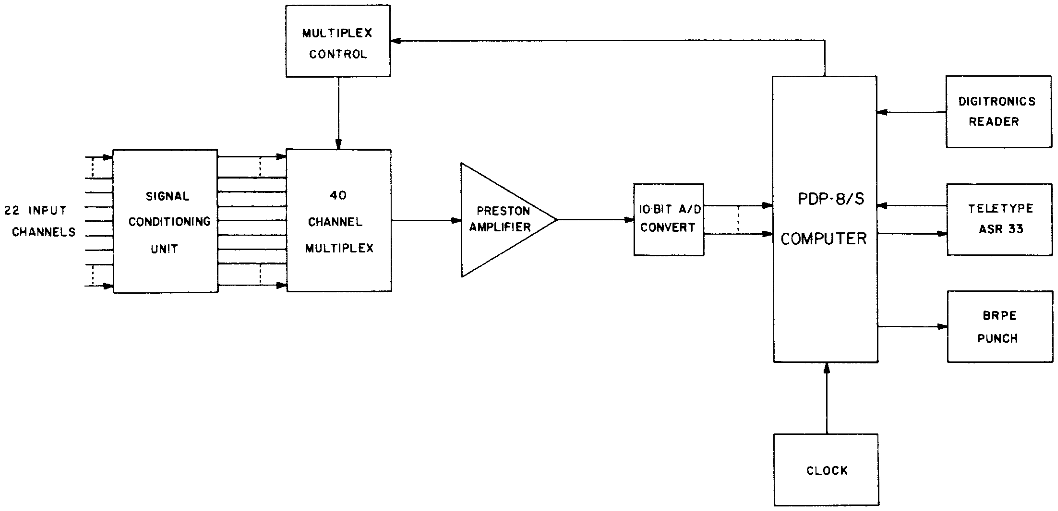


Figure 2 Computer System Block Diagram

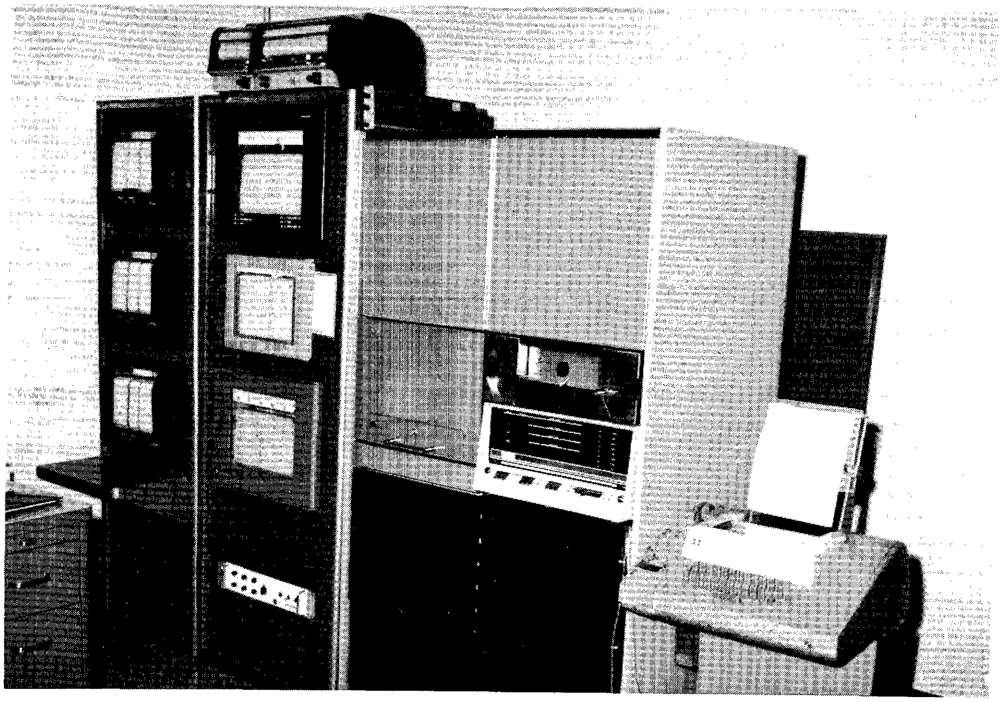
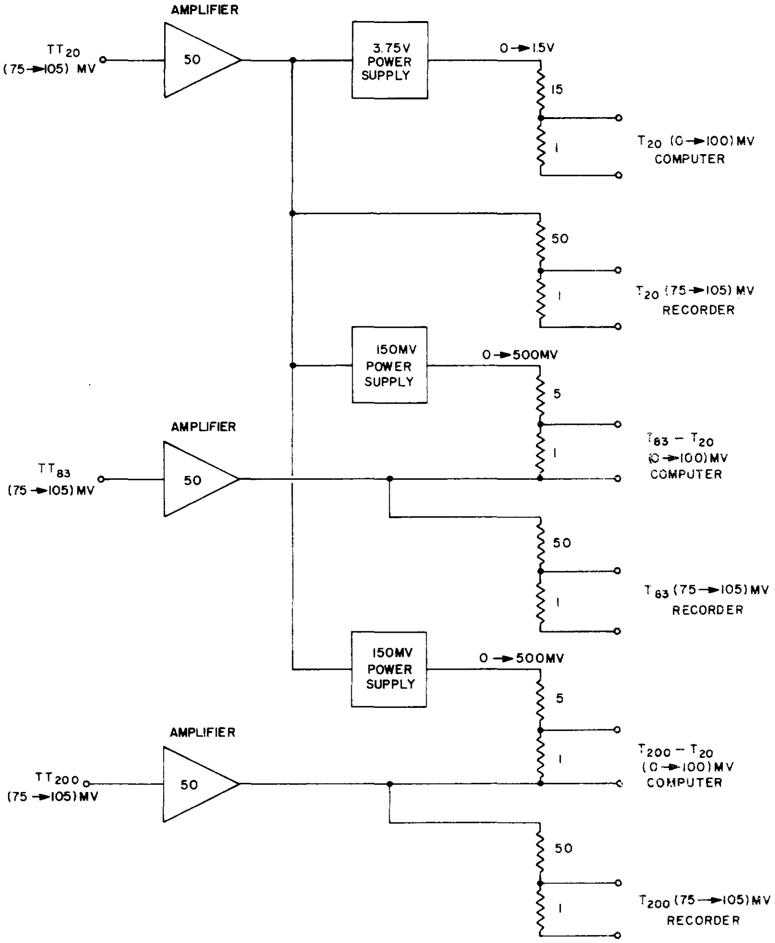
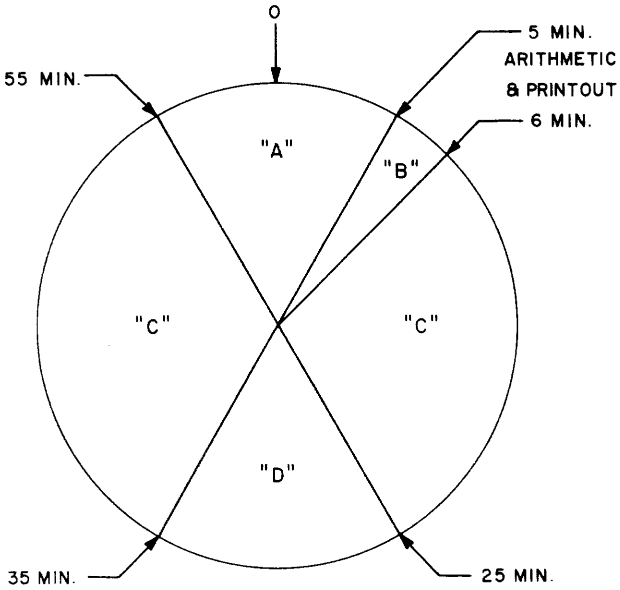


Figure 3 Computer System and Recorders



AMPLIFIER
$TT_{20}$ (75→105) MV
50
3.75V POWER SUPPLY
0→1.5V
15
1
$T_{20}$ (0→100) MV COMPUTER
50
1
$T_{20}$ (75→105) MV RECORDER
150MV POWER SUPPLY
0→500MV
5
1
$T_{83} - T_{20}$ (0→100) MV COMPUTER
AMPLIFIER
$TT_{83}$ (75→105) MV
50
50
1
$T_{83}$ (75→105) MV RECORDER
150MV POWER SUPPLY
0→500MV
5
AMPLIFIER
$TT_{200}$ (75→105) MV
50
1
$T_{200} - T_{20}$ (0→100) MV COMPUTER
50
1
$T_{200}$ (75→105) MV RECORDER

Figure 4 Tower Temperature Conditioning



| Sensor | Sample Period |
|---|---|
| Wind Speeds | A |
| Wind Directions | A |
| Tower Temperatures | A |
| Soil Temperatures | A |
| Bivane | D |
| Radiometer | A+C+D |

Figure 5 Timing

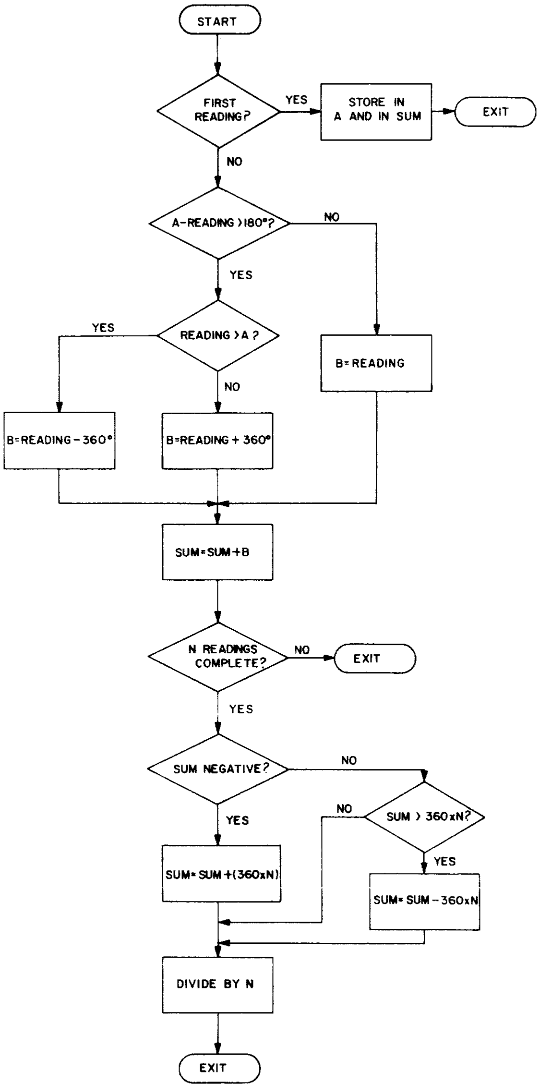


Figure 6 Wind Direction Flow Diagram

# A FLEXIBLE DATA ACQUISITION AND CONTROL SYSTEM UTILIZING A PDP-8/S

G. E. Stokes and D. R. Staples
Idaho Nuclear Corporation
Idaho Falls, Idaho

## ABSTRACT

A multi-scaler data acquisition system with sensing devices and feedback control to the experiment has been designed around a PDP-8/S computer. This system has been used on a number of experiments with a variety of control requirements. In each case the configuration was integrated into the experimental setup with a minimum of hardward changes. The computer interface includes four 12-bit scalers, a real time clock, a 10-bit ADC, a 6-bit relay divider, pulse generators for driving pulsed motors, and a 10-bit digital-to-analog converter.

## INTRODUCTION

Recent work at the Materials Testing Reactor (MTR) created a demand for data collection systems and control functions on several experimental facilities. The nature of the experiments required the use of the data acquisition and control functions on a periodic basis. The availability of limited funding dictated the use of an inexpensive system to satisfy the needs of the various applications. A system has been designed around a PDP-8/S general purpose computer that satisfies the above requirements and it was constructed for less than $15,000. A block diagram of the system is shown in Figure 1.

The system consists of four major functional areas, the PDP-8/S computer and a real time clock, a data input section, an output section, and a man-machine communication section. The computer with the interface installed is shown in Figure 2. The logic functions used in the interface consisted of M series logic from Digital Equipment Corporation, and a number of functional modules that were built here at MTR (level converters, device selectors, indicator drivers). M series logic proved the most inexpensive even with the extra cost of the additional modules. The system is in use at the MTR and has been very satisfactory in a number of applications. It has been used to acquire data and control the sequencing and positioning of samples on a scandium filtered beam. During data acquisition, corrections for detector efficiency and unwanted background signals were made. The system was used on a crystal spectrometer to drive a scanning device and acquire the data associated with this device. It has also been programmed to control the speed of the fast chopper rotor. It has been used in feasibility studies on other experiments to assist in designing instrumentation for those experiments. In each case the system has proven to be flexible and is easily programmed for a specific application. A description of the system follows.

## COMPUTER AND CLOCK

The computer has 4096 12-bit words of core storage and an 8 microsecond memory cycle time. The interface is connected to the computer through the same input bus as the teletype. The clock (Figure 3) consists of an 8-bit binary scaler fed by an RC type variable oscillator. The oscillator is tuned to 1 KHZ for the majority of our applications. When the clock overflow flag is set the scaler is read into the computer accumulator by an IOT and the accumulated time is stored in the computer memory. The scaler overflow flag can be set by the overflow of one of the four high order bits of the clock scaler. This bit is switch selectable from the front panel. The clock is used to keep track of elapsed time and to control the gating of the data scalers.

## DATA INPUT SECTION

The data input section is made up of the data scalers and an analog to digital converter (Figure 4). The data scalers are four 12-bit 6 MHZ binary scalers. The scaler inputs will accept pulses from -3 to -30 volts in amplitude. One of the scalers is wired to the overflow flag so that the overflow of one of the four high order bits of the scaler sets a flag. When this flag is set the input to the scalers is gated off while the scalers are read serially into the accumulator and the contents stored in the computer core. The overflow flag can be set by the clock scaler or scaler 0 of the data scalers. These functions are switch selectable from the front panel. When the clock is used to gate the scalers the system will accept counting rates to 140 KHZ without overflow of the 12-bit data scalers. This counting limitation could be raised by increasing the frequency of the oscillator driving the clock. A clear IOT clears all the scalers and the clock. Another clear IOT to the overflow flag gates on the clock and scalers. The ADC is a 10-bit, 100 KHZ successive approximation device. It is used to handle signals from thermisters and thermocouples. The ADC is controlled and read by computer IOT's.

## OUTPUT SECTION

The output section is shown in block diagram in Figure 5. There are facilities for driving two pulsed stepping motors, for selecting one of six relays and for pulsing three pulsed relays. The stepping motors are pulsed by an IOT from the computer, one pulse for each IOT issued. The relay register is loaded from the accumulator by an IOT and the appropriate relays are opened or closed according to the state of the register bit they are connected to. The three pulsed relays are used to drive devices that require a latching and then a release. The relays that are presently used have holding times up to one second. The output section is used for control functions that are needed at the experiment.

## MAN MACHINE COMMUNICATION SECTION

The last section to be described and perhaps the most useful one for the experimenter is the section that allows communication with the machine (Figure 6). This section has a digital to analog converter for driving analog devices and a group of function switches for operator convenience in communicating with the machine. The DAC was designed for use with a storage scope but has successfully been used to drive an RM-503 and a tektronix 335 oscilloscope. The display was good on the non-storage scopes when less than 256 points were displayed. The display worked well with the focal package for the PDP-8/S. The function switches have proven very useful for people that are not familiar with the computer. The state of the switches are read into the accumulator by an IOT. The program can then check the switches and branch to the appropriate function. There is a flag that can be set by a push button on the front panel. This flag interrupts the machine so that the function switches can be checked at the operators command. The computer and control panel are shown in Figure 7. The switches can be labeled for various functions depending upon the program in the machine.

## CONCLUSION

The system just described has proven of great worth in the experimental group at the MTR. The use of the function switches have enhanced the usage of the system by non-computer oriented people. The system was inexpensive to build but has demonstrated a high degree of flexibility. The programming is done by many of the experimenters themselves. The data are stored in a two word format. The data count is stopped every 16 milliseconds in a high counting rate experiment and the data stored in the scalers are transferred to computer storage 0.3 milliseconds. The count is then automatically resumed. The storage in the machine in two word format takes 5 milliseconds and then the machine is through until the next 16 millisecond interval has elapsed. The system has been very reliable.

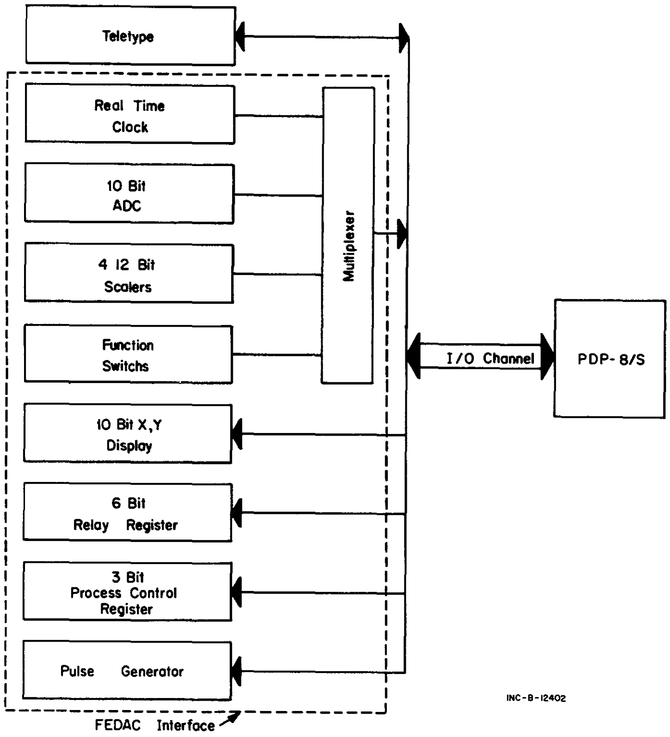


Figure 1. A block diagram of the data acquisition and control system.

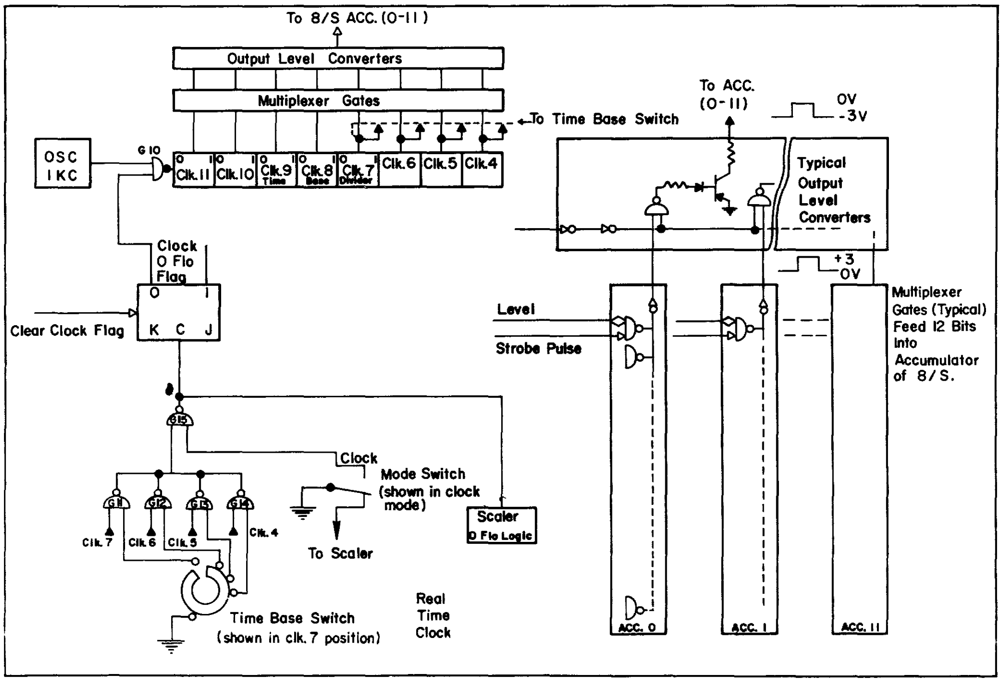


Figure 3. A block diagram of the real time clock and the multiplexer gates.

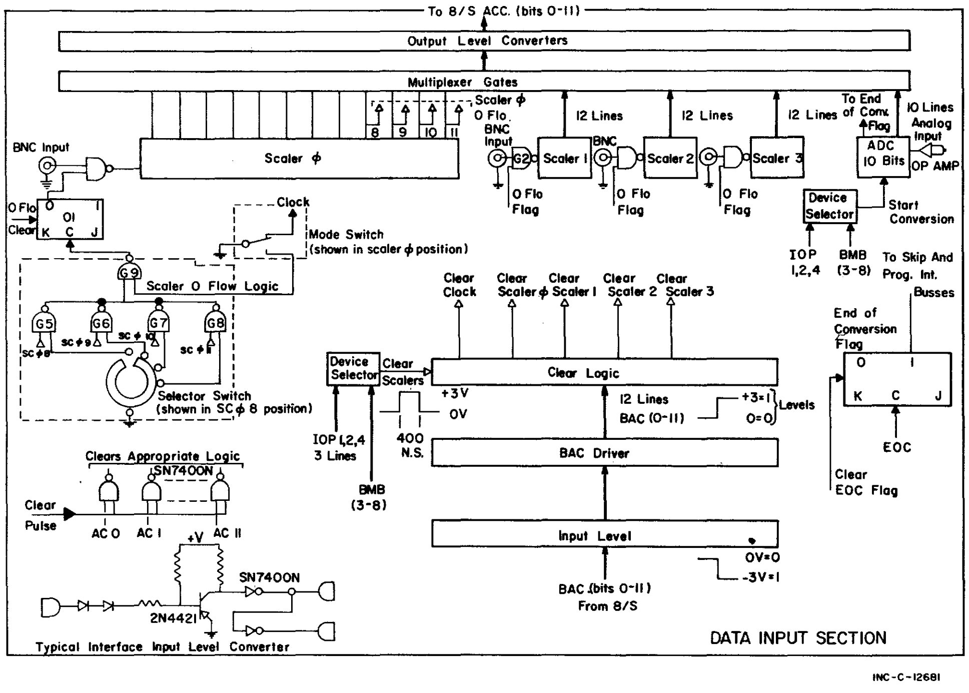


Figure 4. A block diagram of the input section of the interface.

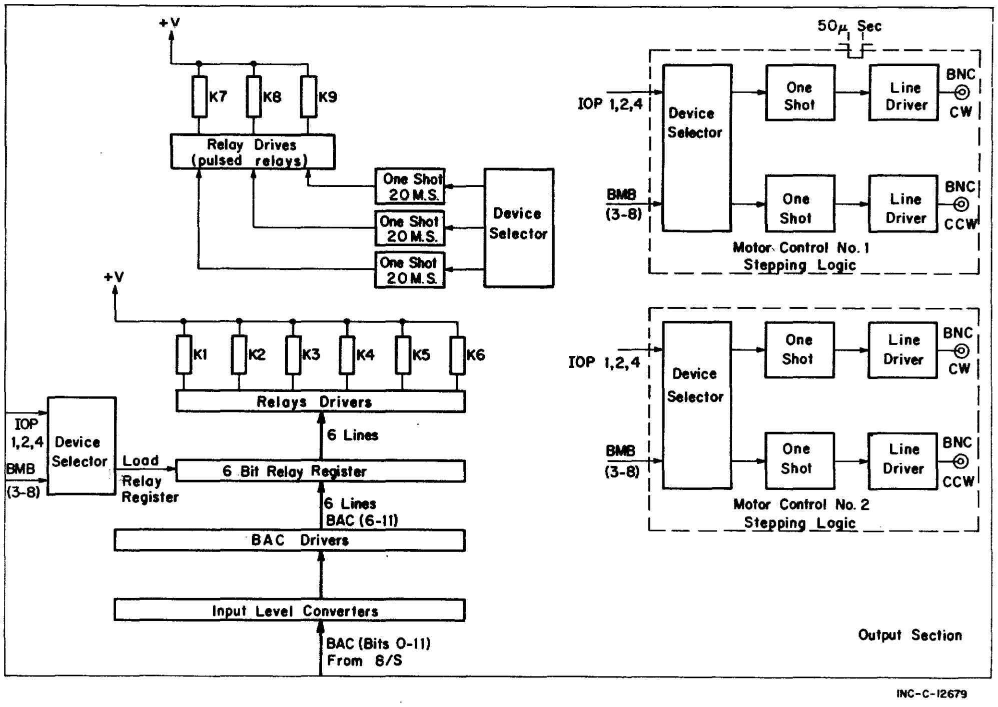


Figure 5. A block diagram of the output section of the interface.

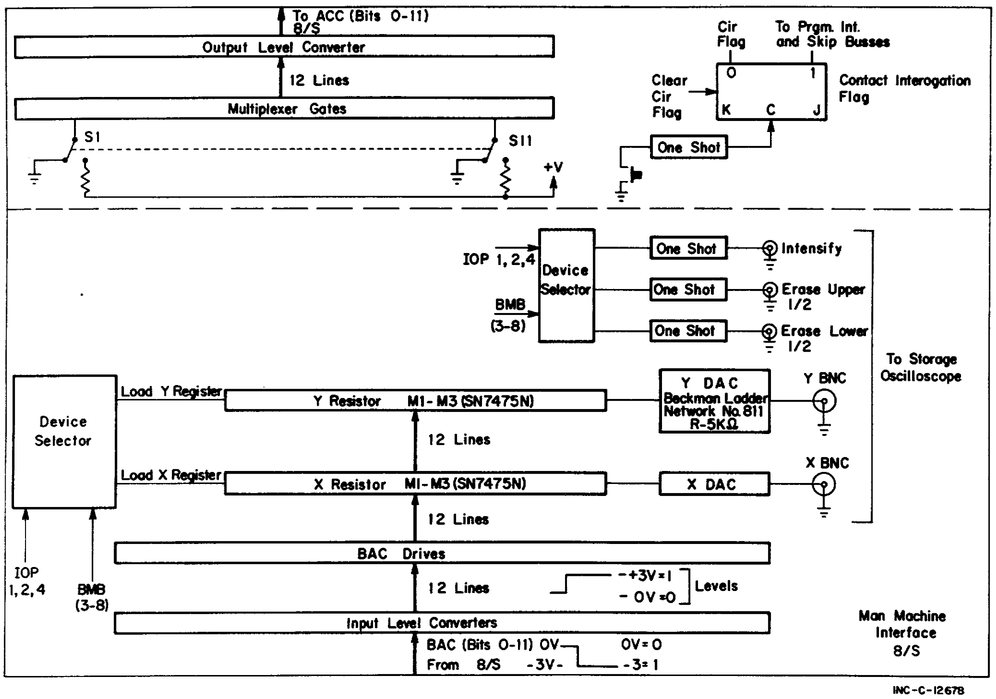


Figure 6. A block diagram of the man-machine communication section of the interface.

# A MODULAR ANALOG DIGITAL INPUT OUTPUT SYSTEM (ADIOS) FOR ON-LINE COMPUTERS

R. W. Kerr, H. P. Lie, G. L. Miller, and D. A. H. Robinson
Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

## ABSTRACT

A system is described which is designed to permit the user of an on-line computer to achieve a desired hardware configuration with a high degree of flexibility. The system uses modular plug-in units inserted in bins which are interconnected by a common two-way analog and digital data bus. The system is designed for a PDP-8 computer and can be expanded, as needed, to handle up to 60 modules.

The modules, which were designed to provide a simple, low cost means of analog and digital measurement and control, are of four types. These are a digital input-output register, a scaler, a relay bank, and a power supply. The design of the modules takes advantage of the versatility of the computer in a number of ways, including provisions to allow the computer to check each module for proper operation.

Several system configurations are described. Included are examples of the use of the system to automate such things as the testing of spacecraft experimental hardware, and the measurement of Hall effect coefficients over an extended temperature range.

## 1. INTRODUCTION

The most important single feature that allows a computer to be employed in a broad range of calculations is the fact that one can, by programming, in effect restructure the machine to perform the desired computational task.

It is not possible to retain the same degree of flexibility in on-line systems because of their need to be connected to specialized external hardware. Primarily for this reason the majority of on-line computer systems that have been constructed in the past have been designed to perform a predefined class of specialized operations. This situation is analogous to that which existed before the invention of the stored program machine when a computing device would be constructed to perform each new special task.

The system described here is the result of an effort to obtain a reasonable degree of flexibility in on-line computer controlled environments and is based on careful considerations of the factors that tend to limit such flexibility. The consequences of such problems in previous systems has been evidenced by difficulties of adding or reconfiguring hardware and interface equipment. Such difficulties have reduced the potential versatility of many existing systems in which the effort required to implement useful changes is uneconomic and such changes are therefore only rarely made.

It is possible, however, by the use of appropriately designed modular units interconnected by a common two-way analog and digital data-bus to obtain the desired degree of flexibility and power. Section 2 of this paper outlines the general consideration involved in the design of such databus systems, while the remainder of the paper describes the implementation of these ideas for a specific small computer, namely a Digital Equipment Corporation PDP-8.

## 2. DESIGN CONSIDERATIONS

Of the many schemes whereby equipment may be connected to a computer perhaps the simplest division is between "radial" and "bus" systems. In the former, the interconnecting cables can be thought of as radiating like the spokes of a wheel to connect the computer to each external unit, while in the latter each external unit is connected to a common "highway," "party line," or "bus" cable system. In a certain sense this distinction is artificial since in the last resort even a radial connection is handled on a bus basis once the signals enter the computer hardware proper. However, the distinction is a valid one for the domain of equipment external to the computer itself and can serve as a starting point for comparing different systems.

The greatest single advantage in a radial system, from the user's point of view, is the fact that external equipment need only be plugged into a suitable connector to be on-line with the computer. The outstanding disadvantage, however, is that such systems are relatively inflexible and can become expensive if large numbers of external units are required. Furthermore, the user may become overly dependent on the computer vendor and his instrument division since only their equipment is automatically interfaced with the machine. In bus systems, on the other hand, the organization is different in that each external unit is connected to a common set of cables which carry address, data, and status information to and from the computer.

A feature of such bus systems that is worth noting is that the interfacing operation between the computer and the external world occurs only once, namely between the computer and the data bus. It is therefore possible to design such systems so that the same, or different collections of on-line equipment can be connected to many different computers by changing only the bus-to-computer interface.

The differentiation into radial and bus systems is indicated by the node labeled 1 in Fig. 1. Other important decisions follow at other nodes in this diagram and it is the purpose of this section of the paper briefly to indicate the major considerations involved at each branch. In order to forstall any misunderstanding it may be well to point out that though a particular design path was followed in the PDP-8 system described in the remainder of this paper, it is by no means claimed that the resulting system is ideal for every applications. As will become clear the design of any system is dependent on numbers of factors, major ones being, for example, the total size of the system envisaged (i.e., number of input and output units together with information on their spatial separation), and whether the system is to be employed by a single user or time shared by several noninteracting users simultaneously. Another important consideration is of course input/output speed and data-rate. Interestingly enough, however, in a number of actual on-line experimental environments that we have considered, it turns out that the bus approach imposes only a small time burden on the system. In the last resort this arises from two causes, first because operations proceed sequentially inside the computer, requiring a certain time to service each external unit, and second because external units are themselves often quite slow (e.g., ~50 μs conversion time for a typical 12 bit nuclear physics ADC). The result of this is that if reasonable care is excercised in the design of the bus system the access and I/O time for external units can become quite small compared with the sum of the device operation and computer servicing time. Obviously it is always possible to envisage situations where this is not the case, but we believe them to be only a small subset of most on-line situations. In those cases where I/O speed becomes an unavoidable limitation, e.g., CRT displays, it is usually worthwhile to consider the use of a separate dedicated piece of equipment (such as a disc with suitable DAC's, etc. for display) to perform the critical function at all times.

Returning to the general considerations indicated in Fig. 1, the next question is one of system size. It is taken for granted that the user's hardware will consist of some form of modules which plug into bins (see for instance the European standard IANUS system or the ADIOS system described here), and the question is whether there will be one bin or several. This decision involves questions of how multiple bins are to be addressed, and what will be their physical separation and distance from the computer. This latter point is highly important though easily overlooked. Its importance can be seen in the following way. If the cable length is long then the cables must be terminated to avoid reflection. The logic levels employed for modern microcircuits are typically 0 and +4 volts. For 50 ohm terminated cable this means 80 ma/bit. Since on-line systems usually employ common grounds between analog and digital hardware it follows that the transfer of a 16 bit word can involve a current pulse at over 1.2 amperes in the ground return. The noise and crosstalk implications of this are obvious. (A new data bus system presently being designed at Bell Telephone Laboratories for a multi-user environment using SDS Sigma computers circumvents this problem by using balanced current-driven twisted pair. No such extreme steps were necessary for the relatively small PDP-8 system described in this paper.)

If multiple simultaneous users are envisaged it is advantageous to employ a buffer unit between each bin and the data bus. This has a number of advantages for large systems, not least being the ability to provide logical buffering between bins to prevent one user from wiping out another by, for instance, unplugging a module. This decision is shown at node 3 in Fig. 1.

Again in large multi-bin environments it can be advantageous to provide a bin address with unit sub-addresses within each bin, (this is the route followed in both the European lANUS and BTL Sigma system designs).

At node 5 a difficult choice must be made regarding the extent to which the bus cables are shared by time, or other, multiplexing arrangements. The advantage of multiplexing lies in its ability to reduce the number of cables in the system. The disadvantages are reduced I/O speed and added complications to unit hardware and system programming.

The way in which external units signal the computer via the interrupt system is also one of central importance. In this connection the major choices lie, as indicated at node 6, between using a single common interrupt line, or of employing a hierarchical or priority system. The latter can be organized two ways, either by using a separate physical interrupt wire from each external unit to the computer, or by connecting the external unit interrupts in head-to-tail fashion whereby priority is defined by position in the chain. Neither of the latter systems is well suited to a flexible data bus system designed to accommodate a wide variety and number of external units since each time a change is made in the configuration of modules, numbers of separate physical wires must also be re-routed.

It will be appreciated that the foregoing discussion of general questions is of necessity superficial, though we believe it to indicate most of the major hardware considerations involved. Without going too deeply into details of software and logical design one other question regarding module addressing must be raised. This is the issue of what we term "generic" addressing and its importance can be seen with a simple example. Suppose the on-line system involves a number of external devices which must be turned on and off in exact time synchronism. Since any bus system is by definition sequential, in that only one set of address lines is used, it is not at first clear how this can be achieved. A solution to the problem can be provided by allowing units to recognize more than one address. Each unit or module recognizes its own unique address and having been so addressed one of the commands to which it can then respond is to enable recognition of another address. Such other addresses are termed generic addresses and they can be common to many different units. In this way the computer can issue generic commands which apply simultaneously to any subset of external modules, allowing them to operate in exact time synchronism.

By way of concluding this section on general design considerations it may be illuminating to consider a number of questions that can be asked regarding the logical organization of any on-line computer system. Does it, for example, require special timing, logic and drive circuits to be added to an external unit before it can talk to the computer? If the answer is yes then the chances are that considerably less experimental innovation will be carried out with the on-line hardware than would otherwise be the case.

Another important point to bear in mind is the ability of the system to check itself. Can the computer tell what units are connected and whether they are operating? This feature can be very important in systems employing many modules.

An area that is outside the scope of this account is that of programming, but it is obvious that the hardware and software of any on-line systems must be harmoniously designed. Less frequently considered from the outset, however, is the question of the ease of debugging the operation of the entire on-line system. Our experience with the present system has shown the extreme desirability of being able to "force" external equipment to well defined conditions, by hand, as a check in debugging programs and hardware. This also is therefore a point to consider in comparing system configurations, how difficult is it to debug the hardware-software interaction in preparing programs for the on-line system?

A corollary to this point is the related one of investigating the degree to which the computer is able to exercise external equipment. In this connection it has been found extremely useful to prepare programs which operate all the computer-accessible features of a module sequentially. This approach allows convenient debugging of modules as they are produced since an operator can examine repetitive waveforms at his leisure, proceeding sequentially through a series of test conditions under programmatic control.

A final point involves the provision of an analog measurement capability within the data bus system. This has been found to be most useful in the PDP-8 system described here, and comprises a shielded twisted pair in the data bus cable connected to a central 12 bit ADC at the computer. In conjunction with suitable external modules this furnishes the ability to both measure and provide analog levels. Together with the digital capabilities of the system this provides a combined digital and analog capability which encompasses a broad range of applications.

## 3. THE DATA BUS

A diagram of the data bus system chosen for a PDP-8 and a single-user environment is shown in Fig. 2. The logic of the operation of the data bus closely parallels that of the PDP-8 computer. Commands are sent to a particular module by placing the module address on the address lines and activating the instruction lines. The three instruction pluses in this system occur at 1 μs intervals and are .4 μs in width. Because one of the system rules is that the modules operate with instruction pluses of any length greater than ∼.2 μs, there is no restriction on the maximum length of the cycle. (Thus the system may also be used with other computers of lower speed than the PDP-8 provided a suitable computer to data-bus interface is constructed.)

The type of operation performed with each instruction pulse has been standardized as follows.

| Instruction Pulse | Operation |
|---|---|
| IOP1 | Augmented Instructions &<br>I/O Skip |
| IOP2 | Input to computer |
| IOP4 | Output to modules |

Since it was useful to have many more than three instructions, a system for deriving a set of augmented instructions during IOP1 is used wherein the six low order bits of the computer output lines are each interpreted as a separate instruction. The six high order bits have been reserved to be used in coincidence to obtain 64 additional augmented instructions, should such a need arise in the future. IOP2 is used to generate all inputs so as to relax the requirement on the fall time of the input pulses which must be clear before the next instruction is executed.

A module requests attention from the computer by energizing a common interrupt line until it is serviced by the computer. The computer then interrogates the modules to determine which one is requesting service. (Since a priority interrupt system might be desirable when the system is interfaced to a different computer, four additional lines in the data bus have been provided, which may be used in this manner.)

The I/O skip line provides a means for a module to respond to interrogation by the computer. An affirmative response is signalled by energizing this line, which causes the PDP-8 to skip an instruction. If the system were connected to a different computer, the I/O skip line could set a status bit in the machine.

The distribution of the analog input lines to the module bins is performed with shielded twisted pair. The interface contains a parametric amplifier in a configuration which converts the single ended 0 to -10V range of the analog to digital converter in the PDP-8 to a +10V to -10V differential system with good dynamic common mode rejection.

Since the logic of the bus system is compatible with the PDP-8, the interface is used simply to provide the level shifting and buffering that is necessary to communicate directly with the integrated circuits in the modules.

The interface unit converts the negative logic levels of the PDP-8 to standard microcircuit levels as used in the data bus system. In addition the interface input buffers provide noise filtering and an input threshold which can be varied in order to investigate noise margins. Tests have shown the data bus system capable of operating with cable lengths of more than 100 feet.

The data bus consists physically of 48 miniature coaxial cables, together with one shielded twisted pair, which interconnect the required number of module bins. Within the bins, the data bus loops through twelve 50 pin connectors into which the modules connect upon insertion into the bins.

## 4. THE PLUG-IN MODULES

Four general purpose modules were designed to operate in conjunction with the computer to assist in the operation and control of experiments and in the acquisition of the resulting data. Figure 3 shows one of each type of module installed in a module bin. A modified NIM power supply located at the rear of the bin provides local power for the modules.

The construction of all the modules is similar to that of the scaler, which is shown in Fig. 4. The use of integrated circuits on a single special purpose board results in considerable reduction of cost and size over the more usual technique of using general purpose logic boards. The cost of the more complex modules is approximately $300 each.

In order to simplify programing and debugging, the module addresses are defined by small plug-in cards, visible in Fig. 4, which may be changed at will. Removal of a unit for repair may thus be performed by switching its address card to a new module. A brief discussion of the structure and operation of each of the modules is given in the following four sections.

### 4.1 Register

The register provides a general purpose interface for digital devices. It is capable of accepting a 12 bit word from an external device and inputting the word to the computer. It can also accept a word from the computer and present it, with buffering, to the outside world. The block diagram of this unit is shown in Fig. 5. Table I lists the commands for the unit, most of which require no explanation.

The voltage levels for input and output are standard NIM and integrated circuit levels. The use of jam transfer makes resetting unnecessary, and allows alteration of selected bits of the register without even a momentary change in the other bits.

The use of master-slave flip flops as buffers permits the register to be used as a hardware bit-reformatting device by connecting the outputs of the register to the inputs in the desired sequence. The word to be reformatted is sent out to the register and then read in as external data. This feature also permits use of some bits of the register as input and some as output, since by connecting a bit output line to the corresponding bit input line one makes the value of the bit independent of the Load Register from External Unit command.

The most serious stumbling block in interfacing an external device to a computer is not the compatibility of the input/output levels, but the necessity of establishing logical communication between the devices and the computer in a simple way. In the register the "freeze" circuitry allows the computer to command a device to remain stable while being read. The interrupt circuitry allows the device to request service from the computer. A busy line informs the unit of the status of its request.

### 4.2 Relay Module

The logic of this module is shown in block diagram in Fig. 6. It contains twelve single pole double throw high speed reed relays each capable of switching 3 amps. It is used in conjunction with a register module to handle signal and power levels which are inconvenient to handle electronically.

The relay driver commands are shown in Table II. The Test Unit Ready command allows the computer to test for the presence of the module.

### 4.3 Scaler

This module comprises a 12 bit binary ripple counter and the necessary logic to permit the module to function on the PDP-8 data bus system. In operation the module sends an interrupt to the computer for every 4096 input pulses. The system records the number of these interrupts and therefore functions as a scaler modulo 4096. At the end of the counting period the fractional count remaining in the scaler is added to the previously recorded total. Figure 7 is a logical block diagram of this unit. A unique feature in this design is the use of a two address command structure. The first of these is the generic address and the second the unit address. By use of the generic address the computer can execute any one of three commands and have all scalers sharing that address respond simultaneously. The command structure for the unit is shown in Table III. Most of the commands are self-explanatory. While Increment operates in front of the input gate, Preset, which also increments, operates at all times. Enable Generic and Disable Generic permit the generic commands to be obeyed or ignored.

The overflow and saturate logic allow the computer to serve as the high order portion of the scaler in the following manner. When the high order bit of the scaler overflows, its overflow flip flop is set and a program interrupt is sent to computer. The computer then initiates a search using the Test Overflow instruction and thereby ascertains which module interrupted. Should a second overflow occur in a scaler before the previous one has been recorded by the computer then the saturate flip-flop is set. The computer can test this flip-flop, and thus either ascertain that the scaling has been performed without error or take appropriate action to insure correct scaling.

A rear panel switch connects the output of the most significant bit to either the overflow detecting circuitry or to a front panel connector, thus allowing the use of a second scaler module to form a 24 bit scaler if desired.

A discriminator is located at the input to the module and its level is adjustable from -5 volts to +5 volts. A front panel lamp indicates the status of the input gate.

A three position switch allows manual setting of the unit in either the start or stop condition, or returns this control to the computer.

### 4.4 Programmable Power Supply

The primary purpose of this module is to allow the computer to supply adjustable voltages to external devices. As shown in Fig. 8, the computer controls the power supply voltage by causing rotation of a motor driven ten turn potentiometer which serves as an inexpensive analog memory. The series regulators, which operate by re-regulating the bin power, are built either as positive or negative supplies, and furnish 0 to 10 volts with overcurrent protection from 10 to 250 ma.

The control commands for this unit are shown in Table IV. The computer controls the unit by operating the potentiometer while simultaneously monitoring its output voltage, thus becoming part of a servo loop.

The front panel dial attached to the potentiometer indicates the supply voltage directly while also permitting manual setting of the voltage. In addition, the module may be used as an analog input from the operator, since the computer can, in effect, read the dial setting. Connection of the potentiometer shaft to other rotating equipment could also permit the computer to cause controlled motion in the external equipment should this be desired.

## 5. SAMPLE APPLICATIONS

The system has been used to control, and process data from space experiments; to run nuclear analysis displays using DAC's; and as the data acquisition and control center for an automatic Hall-effect measuring system.

Figure 9 is a block diagram showing how experiments are connected into the system. Note that the computer output can control the experiment via the data bus. Computer input can store digital outputs from the experiment via the data bus and make analog measurements via the analog bus and computer ADC.

### 5.1 Testing of a Satellite Charged Particle Spectrometer

A simplified block diagram of a satellite particle detector experiment is shown in Fig. 10. Particles incident on the semiconductor detector assembly deposite their charge in one or more detectors. Coincidence logic applied to detector outputs determines the particle type, while the linear system sorts the energy of each particle type into one of five consecutive energy ranges.

Sixteen different particle identifying modes and the five channel energy ranges are controlled by the digital outputs from the spacecraft sequence clock, in such a way that each mode lasts for approximately 10 seconds. For in-flight calibration the experiment contains a test pulse generator and two internal sources, each activated by certain states of the sequence clock once every six hours.

When tested in thermal vacuum in the laboratory by the computer system, outputs from a register unit simulated the sequence clock and thereby controlled the experiment modes. Calibration modes were arranged to alternate between the test pulser and internal source modes for every complete sequence of experiment modes, interspersed with complete sequences of no excitation. In this way a calibration cycle was repeated once every 25 minutes, so large amounts of calibration data were processed in a relatively short time. The sequences of no excitation were useful for the observation of noise counts. Scalers were used to accumulate the five channel outputs and to transfer the counts into the computer for printout.

Temperatures were also recorded. The outputs of temperature sensors were switched onto the analog bus using a relay driver and register combination, and were measured by the computer ADC.

Although not used in this particular test, it would be appropriate to use programmable power supply units in a test of this kind to investigate the effect of varying power supply voltages on circuit performance.

### 5.2 Automated Hall-Effect Measurements

An ion implantation laboratory is in operation and many implanted diode samples will require extensive electrical testing. Each sample is expected to go through several stages of annealing, and following each stage measurements will be made to evaluate Hall coefficients, specific conductivity, carrier concentration and carrier mobility, over the temperature range 2°K to 300°K. Figure 11 shows a simplified block diagram of the electrical system.

A large number of voltage measurements from contact to contact are required at each temperature of interest, and the temperature stability at each measurement point must be carefully controlled.

Figure 12 is a flow chart showing the main steps in the computer controlled system. After the sample is mounted and ready to be lowered into the cryostat, the program starts with a comprehensive check of the hardware and a "FAULT" printout is generated indicating the nature of any malfunction. An "OK" printout indicates the satisfactory completion of each test.

When the hardware test is completed the sample is manually lowered into the cryostat. The program continues by welding the sample contacts to ensure good connections and then checking that the voltage drop across each is within acceptable limits. The weld current and the contact test current paths are selected, by using relay units, so that they flow in the appropriate direction (depending on the diode junction type) and through any desired contact.

The next step is the measurement of a complete set of diode characteristics. These are made at several values of current, taken from a table in memory. A first set of Hall measurements is then made, coefficients are processed and printed out. A manual decision is then made whether the Hall properties exhibited by the sample make continuation of the measurements worthwhile.

In continuing the test, the operator types in the upper and lower limits of temperature range and the increments at which measurements are to be made, and opens the liquid helium valve on the cryostat. The computer selects the temperature values by interpolation from a table in memory, starting with the lowest specified temperature. The required temperature control is provided by the setting of a programmable power supply whose output controls the power applied to the sample heaters.

The computer proceeds in a similar way to control and check the remaining experimental conditions, as indicated in Fig. 12.

The measurement of each of the many voltages is accomplished by using relay units as multiplexers at the input of a digital voltmeter. Two register modules are used to receive the DVM digital data and transfer it to the computer via the data bus. One of these registers is used to trigger, and also to recognize the end of each DVM measurement, signaling to the computer that data is ready for input.

## 5.3 Other Applications

Many uses other than those already described have been considered. An example is that of automated sample liquid scintillation counting in which programmable power supplies can provide levels defining pulse-height windows. In conjunction with scalers this provides pulse height analysis, while relay/register combinations can excise electromechanical control.

The system may also serve as a versatile, economical alternative to large multiparameter pulse height analyzers when used in conjunction with pulse analog-to-digital converters, and fast digital-to-analog converters for CRT displays.

When connected to an engineering breadboard, the system has been used as a versatile programmed pulser and circuit tester.

## 6. DISCUSSION

The point can be made, and with justification, that computer manufacturers realized years ago that peripheral hardware was best handled on a bus basis, which is all that is being achieved with the system described here. This is quite true. The differences that arise with on-line systems are primarily those of degree (with the exception of analog bus facilities) rather than those of kind. One example will suffice to make the point. The present system might be required to handle 60 scalers all counting at ~1 MHz (e.g., a data rate of $6 \times 10^7$ bits/second) with the subsidiary requirement that various subsets of them be gated on and off in exact time synchronsim. Such requirements are not encountered with standard computer peripherals for which supervisory control and timing can always be exercised in a logical sequential manner.

The major point being made here is really a different one, namely how to design a modular system with a small number of different types of modules to encompass a large number of on-line tasks. While examples of such tasks are endless it is hoped the outline of the rationale of the design, together with the sample applications given, demonstrates the flexibility that such an on-line modular analog-digital system can provide.

## 7. ACKNOWLEDGMENTS

It is a pleasure to acknowledge the many contributions of others to the work presented here. Notable among these, have been E. H. Cooke-Yarborough and his associates at AERE Harwell in discussions of design philosophy and in providing information on their IANUS system,[1] R. Stensgaard of the University of Aarhus in all phases of the work on Hall effect measurements, and W. L. Brown of Bell Telephone Laboratories for constant encouragement and support .

TABLE I. REGISTER MODULE COMMANDS

| IOP | DATA BIT | COMMAND |
|---|---|---|
| 1 | 6 | Test Interrupt |
| 1 | 7 | Generate Interrupt from Computer |
| 1 | 8 | Disable Interrupt |
| 1 | 9 | Enable Interrupt |
| 1 | 10 | Set Freeze Output |
| 1 | 11 | Load Register from External Unit |
| 2 | - | Load Computer from Register |
| 4 | - | Load Register from Computer |

TABLE II. RELAY MODULE COMMANDS

| IOP | COMMAND |
|---|---|
| 1 | Test Unit Ready |
| 2 | Enable Relay Drivers |
| 4 | Disable Relay Drivers |

TABLE III. SCALER MODULE COMMANDS

A. UNIT ADDRESS COMMANDS

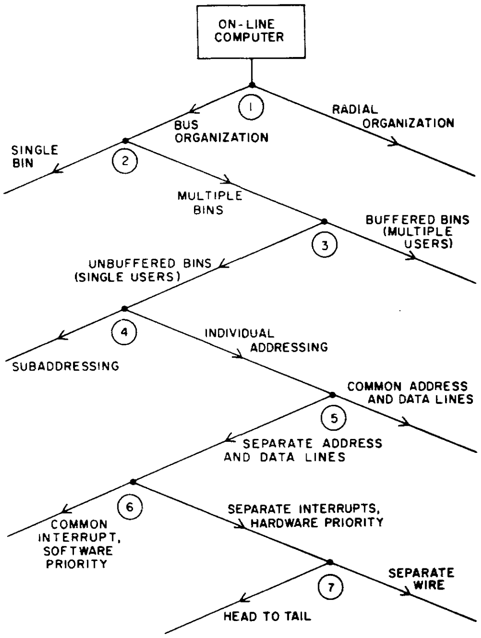
| IOP | DATA BIT | COMMAND |
|---|---|---|
| 1 | 6 | Test Overflow |
| 1 | 7 | Test Saturate |
| 1 | 8 | Disable Generic |
| 1 | 9 | Enable Generic |
| 1 | 10 | Increment |
| 1 | 11 | Clear |
| 2 | - | Load Computer from Scaler |
| 4 | - | Preset |

B. GENERIC ADDRESS COMMANDS

| IOP | COMMAND |
|---|---|
| 1 | Start Scaling |
| 2 | Clear |
| 4 | Stop Scaling |

TABLE IV. POWER SUPPLY MODULE COMMANDS

| IOP | DATA BIT | COMMAND |
|---|---|---|
| 1 | 11 | Motor Off |
| 1 | 10 | Measurement Off |
| 1 | 9 | Measure Current |
| 1 | 8 | Measure Voltage |
| 2 | | Motor Counterclockwise (Lower Voltage) |
| 4 | | Motor Clockwise (Raise Voltage) |



ON-LINE COMPUTER
1
BUS ORGANIZATION
RADIAL ORGANIZATION
SINGLE BIN
2
MULTIPLE BINS
BUFFERED BINS (MULTIPLE USERS)
3
UNBUFFERED BINS (SINGLE USERS)
4
SUBADDRESSING
INDIVIDUAL ADDRESSING
COMMON ADDRESS AND DATA LINES
5
SEPARATE ADDRESS AND DATA LINES
6
COMMON INTERRUPT, SOFTWARE PRIORITY
SEPARATE INTERRUPTS, HARDWARE PRIORITY
7
SEPARATE WIRE
HEAD TO TAIL

Figure 1 Logic tree indicating the major decisions involved in the design of a data bus system.

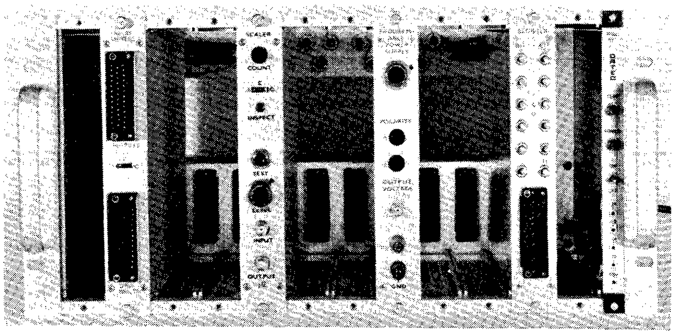


Fig. 3 - Photograph of one modified NIM bin containing one of each of the four modules.

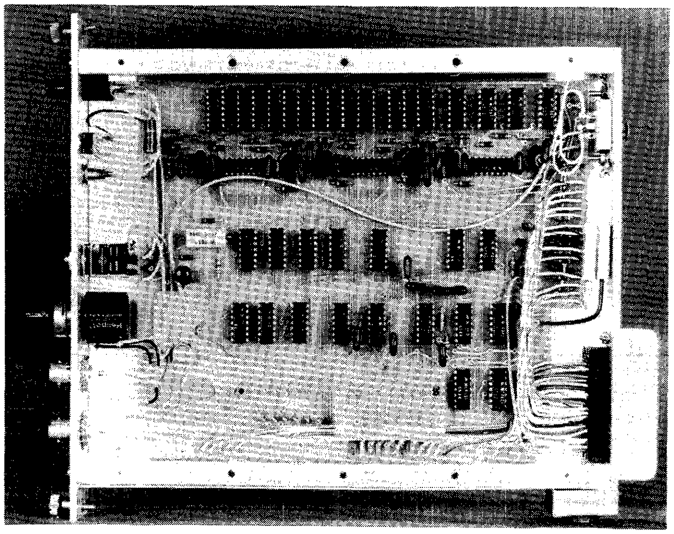


Fig. 4 - Photograph of a scaler module, showing the location of the plug-in address cards at the lower center of the printed circuit.

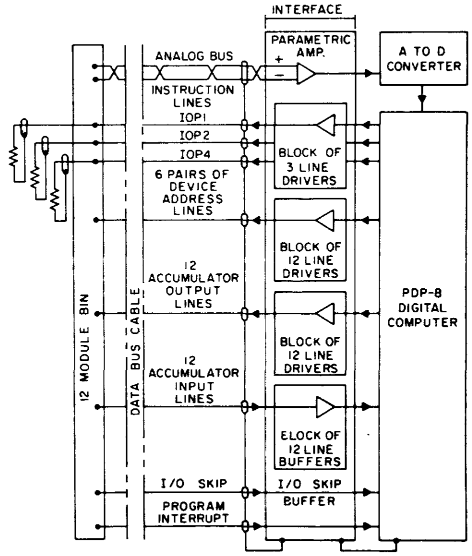


Fig. 2 - Simplified block diagram showing the computer interface and data bus cable.

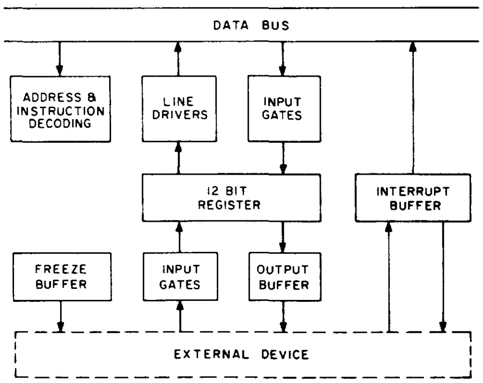


Fig. 5 - Simplified block diagram of register module.

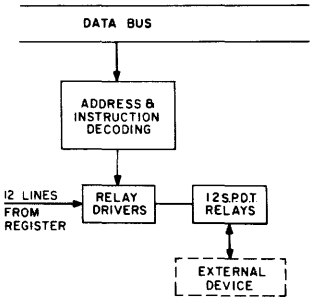


Fig. 6 - Simplified block diagram of relay driver module.

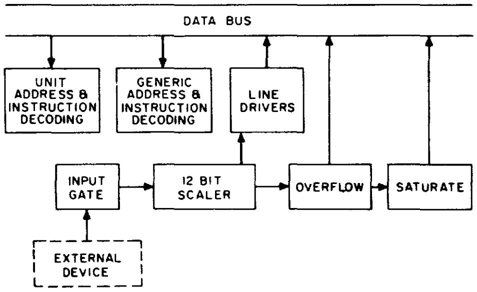


Fig. 7 - Simplified block diagram of scaler module.

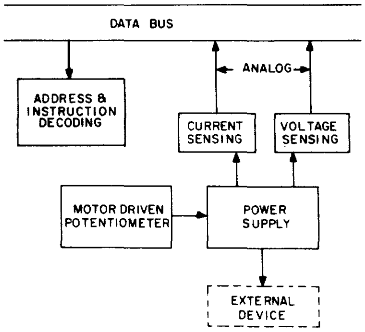


Fig. 8 - Simplified block diagram of power supply module.

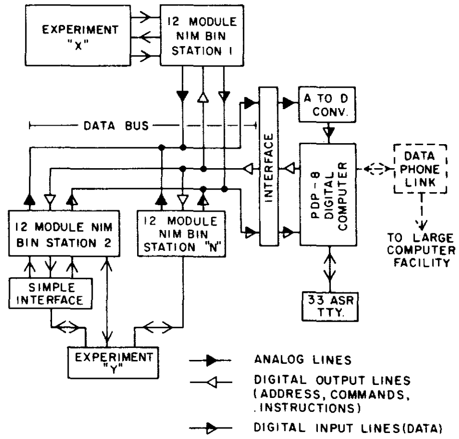


Fig. 9 - Simplified block diagram showing three bins connected to the data bus.

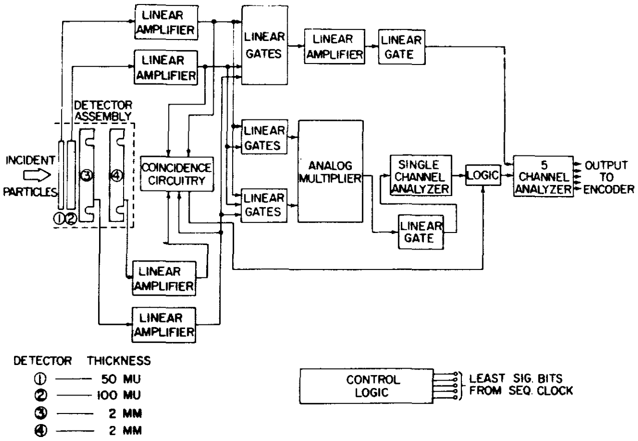


Fig. 10 - Simplified block diagram of a satellite experiment.

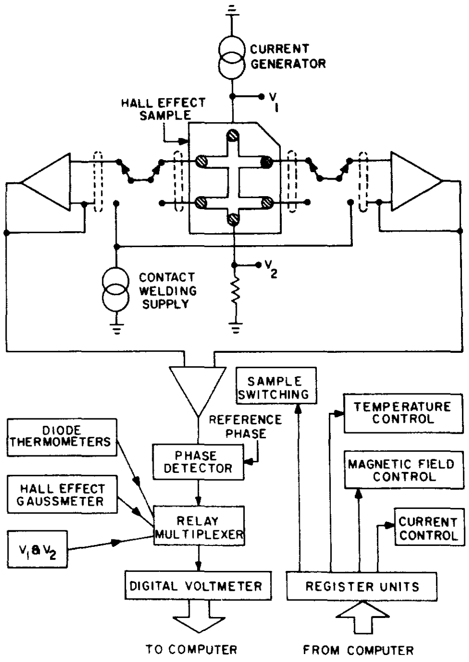


Fig. 11 - Electrical block diagram of Hall effect measurement system.

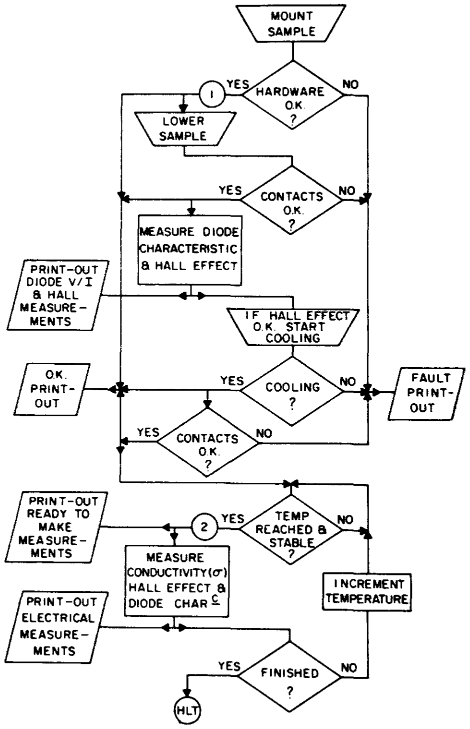

MOUNT SAMPLE
HARDWARE O.K ?
YES
NO
1
LOWER SAMPLE
CONTACTS O.K. ?
YES
NO
MEASURE DIODE CHARACTERISTIC & HALL EFFECT
PRINT-OUT DIODE V/I & HALL MEASURE-MENTS
IF HALL EFFECT O.K. START COOLING
COOLING ?
YES
NO
OK. PRINT-OUT
FAULT PRINT-OUT
CONTACTS O.K. ?
YES
NO
TEMP REACHED & STABLE ?
YES
NO
2
PRINT-OUT READY TO MAKE MEASURE-MENTS
MEASURE CONDUCTIVITY(σ) HALL EFFECT & DIODE CHAR$^{c}$
INCREMENT TEMPERATURE
PRINT-OUT ELECTRICAL MEASURE-MENTS
FINISHED ?
YES
NO
HLT

Fig. 12 - Simplified flow-graph of Hall effect measurement.

## REFERENCES

1. A Standardised Data Highway for On-Line Computer Applications, I. N. Hooton and R. C. M. Barnes AERE UKAEA Harwell, U.K. AFIPS Proceedings-Fall Joint Computer Conference 1968 pp. 1077-1087.

# QUANTIFYING MUSCULAR ACTIVITY IN THE ANALYSIS OF HUMAN SPEECH

## A method of processing electromyographic (EMG) data in the analysis of speech production

Stan Hubler
Phonetics Laboratory, UCLA*
Los Angeles, California

## ABSTRACT

An interface is described which improves the quality of data processing in a LINC-8 computer by preprocessing analog input data using an analog integrator. The interface can be used not only on the LINC-8 but also on other digital computers equipped with multiplexed analog to digital converters.

## INTRODUCTION

Human speech production is complex. It requires lots of brain power and muscular coordination. After years of study, relatively little is known about how the brain initiates speech commands to the muscles and little is known about the exact mechanism of muscle movement. Phoneticians hope that by understanding the activity of muscles in speech they can make reasonable inferences as to how the brain is organized and how it stores and produces speech. The implications of this knowledge are far reaching. Short term benefits can be the correction of speech defects. Long term benefits can be the unravelling of some of the brain's mysteries. But a lot of groundwork remains to be laid. For example, how do you go about understanding the mechanism of speech production? Phoneticians believe that the brain must store some speech data and then use a set of instructions or a program to put the data together into recognizable speech such as words, phrases, sentences, etc. This is as opposed to the brain's having stored all possible data and merely playing it back. So one can immediately ask the question, "What size or sizes are the basic data units?". This question and others are being attacked in many ways. One method is to correlate the activities of the speech muscles with the speech generated. We can't answer the data size question completely by this method but we can at least establish that the unit is either syllable sized or larger or, if they are smaller than the muscle patterns of speech are not the right place to look for them (Fromkin, 1966).

## METHODOLOGY AND PROBLEM

A brief summary of the data collection/analysis methodology used in this laboratory follows. More detailed descriptions are available in Hirano and Ohala (1967) and Hirano et al. (1967). The software has been described by Harshman and Ladefoged in the Fall 1967 DECUS PROCEEDINGS.

Currently, we measure muscle activity by inserting a pair of wires into a muscle and the electrical signals which result from muscular action (electromyographic signals or EMG). (See Figure 4) It turns out that these signals, which are voltage pulses or spikes, are individually about the same size. The number of these spikes per unit time is roughly proportional to muscular effort except for extreme muscular effort -- when the amplitude must be taken into account as well because the spikes become superposed. In our system (Figures 1, 2, 3) we record these signals, then later play them back through the computer, for analysis. But there are some hurdles to overcome in order to get this data into the computer: Speech events of interest may last up to several seconds. If all the muscle spikes are stored (there may be thousands), a great amount of memory will be used. Fortunately, only the relatively slow changes in the muscle activity are of interest, so only the envelope of the activity is required. Previously, this was entered into the computer without preprocessing and, after averaging the envelope was enhanced by a software smoothing technique, was only partially successful. The difficulty was there was a computer generated artifact which appeared similar to noise. It appeared that this artifact could be eliminated if a kind of temporary storage or smoothing device could be installed external to the computer circuitry. An added advantage of this device would be a relaxing of the software requirements and a resulting increase of available core.

## SOLUTION

A computer synchronized, analog integrator was developed to accomplish the temporary storage/smoothing function. It turned out that the synchronization feature was easy to accomplish, since it required a simple modification to the LINC-8 computer. This modification should be reasonably easy for other computers using multiplexed analog to digital converters. A true integrator was chosen over an RC type in order to avoid having optimizing the integrator for a particular sampling frequency.

## DESCRIPTION OF THE INTEGRATOR SYSTEM

(See Figures 5, 6, 7) Raw EMG signals are rectified by a linear full wave rectifier. The output of the rectifier drives the true integrator, which sums the signals in such a way that both an increasing number of signals per unit time and increasing signal size (amplitude) cause the summed output to increase. This process continues for a time determined by the analog to digital converter sampling frequency, which in turn, is determined by the software. Almost immediately after the computer samples the output of the integrator it is discharged to zero (within 15 microseconds) and resumes summing. In this manner the rectifier-integrator combination act as an analog memory and provide a smooth transmission of the analog data to the analog to digital converter.

## HARDWARE REALIZATION

(See Figure 8) As might be expected, certain embellishments to the original idea are necessary so that the hardware is compatible with the real world. An output amplifier follows the integrator and acts as a buffer between the integrator and the normal computer input and, additionally, conditions the output to meet the normal input requirements of the computer. An input preamplifier increases the input impedance from 50 k ohms to about 1 megohm, provides for an optional additional gain of 10 and acts as a low impedance driver to the linear rectifiers. An overload detection circuit warns the user that the integrator has reached its upper limit. This detection circuit triggers a special lamp driving circuit which holds the lamp on long enough to be seen. The overload indicators are necessary since only one channel can be monitored by the computer's oscilloscope during a given averaging run. A input potentiometer has been added to permit timely adjustments in case of integrator overload and to provide an adjustment capability for unadjustable input devices.

All the circuitry has been built on DEC-compatible printed circuit cards which were installed in the spare receptacles in the Data Terminal Panel. Switches, knobs, and lights were mounted on the front of the Data Terminal Panel. Extensive use of integrated circuit operational amplifiers keeps the overall cost of the interface low.

Schematics of the circuitry are available on request from the Phonetics Lab., Linguistics Dept., UCLA, Los Angeles, California 90024.

## COMPUTER MODIFICATIONS

(See Figure 9) Four DEC flip-chip pulse-gate equipped monostable multivibrators (one per data channel) were installed in and connected to the computer. The outputs of these circuits provide triggers to the circuitry which resets the respective integrator to zero. There are two inputs per gate: a voltage level which indicates that the computer is sampling that particular channel, and the 1.9 microsecond pulse which enables the sampling of inputs to the analog to digital converter of the digital computer.

## SUMMARY

A hardware interface which enhances the processing of electromyographic speech data has been described. This interface simplifies programming by easing the software requirements both by eliminating the software rectifier and by presmoothing the data. In addition, the interface saves some core and eliminates a sampling artifact.

The interface provides an output proportional to both the amplitude and number of EMG signals during a given integration/sampling interval.

## ACKNOWLEDGEMENTS

I would like to thank Larry Grant for his excellent work in construction of the interface, Willie Martin for his very good work in modifying the computer and Richard Harshman, Shizuo Hiki and John Ohala for their most helpful ideas, criticisms and suggestions for this paper.

This work was supported in part by the National Institute of Health.

* Presently at RCA, Van Nuys, California

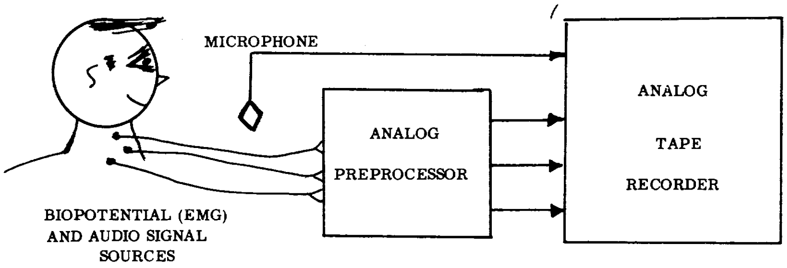
## REFERENCES

1. Victoria A. Fromkin (1966), Neuro-muscular specification of Linguistic Units, Language and Speech, 9-3, 170-199.

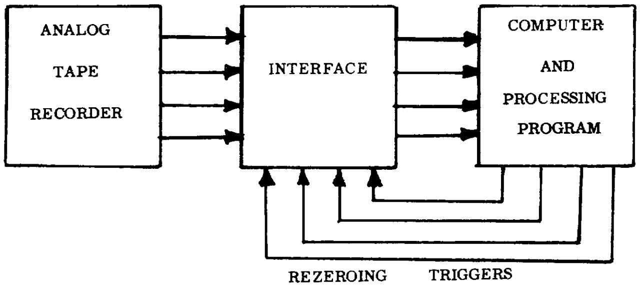
2. Richard Harshman and Peter Ladefoged (1967), The LINC-8 computer in speech research, DECUS PROCEEDINGS, Fall 1967.

3. Minoru Hirano and John Ohala (1967), Use of hooked-wire electrodes for electromyography of the intrinsic laryngeal muscles, Working Papers in Phonetics Seven, Phonetics Laboratory, UCLA, November, 1967.

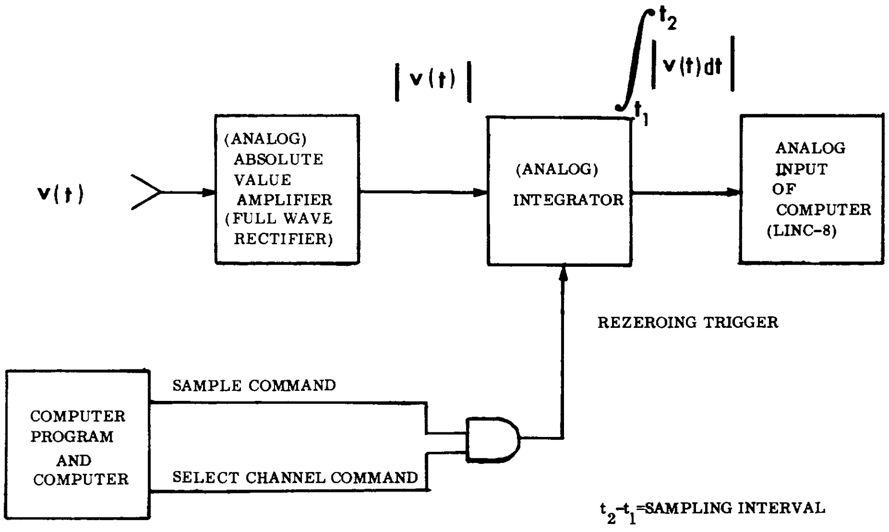
4. Minoru Hirano, John Ohala, and Tim Smith (1967), Current techniques used in obtaining EMG data, Working Papers in Phonetics Seven, Phonetics Laboratory, UCLA, November, 1967.



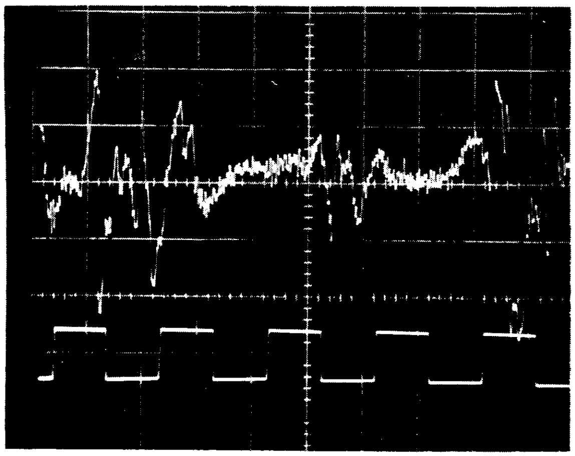
1. DATA COLLECTING SYSTEM



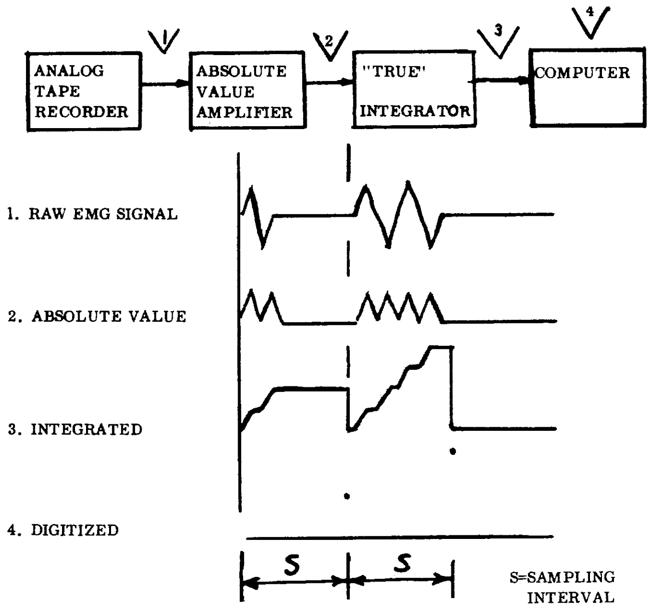
2. DATA PROCESSING SYSTEM


v(t)
(ANALOG) ABSOLUTE VALUE AMPLIFIER (FULL WAVE RECTIFIER)
|v(t)|
$\int_{t_1}^{t_2} |v(t)dt|$
(ANALOG) INTEGRATOR
ANALOG INPUT OF COMPUTER (LINC-8)
REZEROING TRIGGER
COMPUTER PROGRAM AND COMPUTER
SAMPLE COMMAND
SELECT CHANNEL COMMAND
$t_2$-$t_1$=SAMPLING INTERVAL

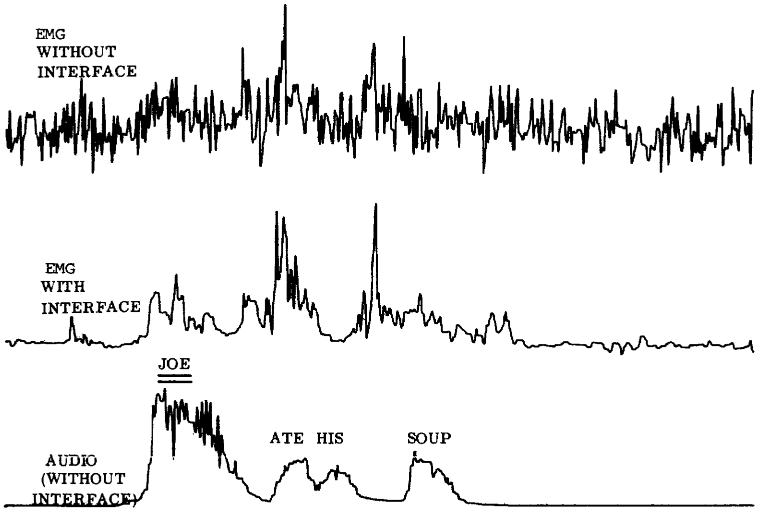
3. INTERFACE SIMPLIFIED FUNCTIONAL DIAGRAM



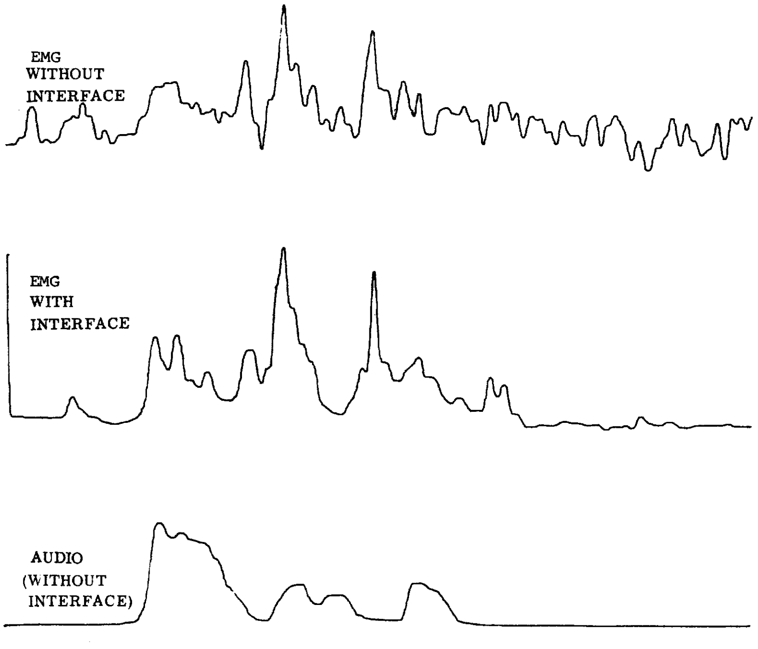
4. RAW ELECTROMYOGRAPHIC SIGNALS
50 microvolts/cm 2 milliseconds/cm



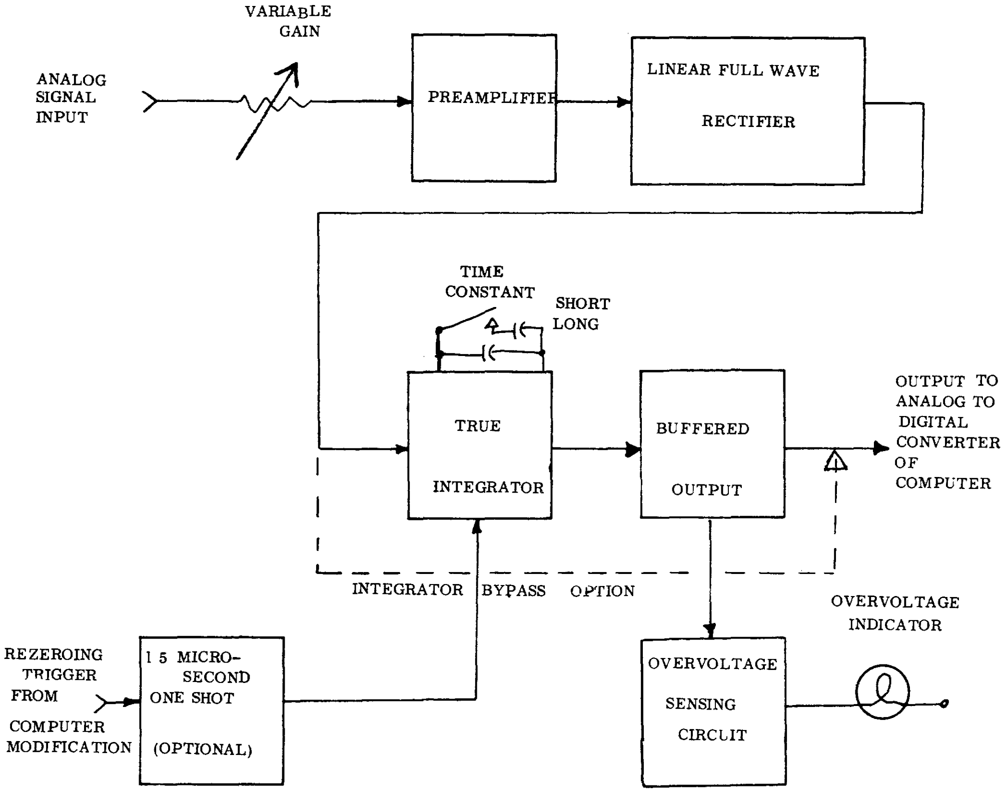
5. INTERFACE WAVEFORMS



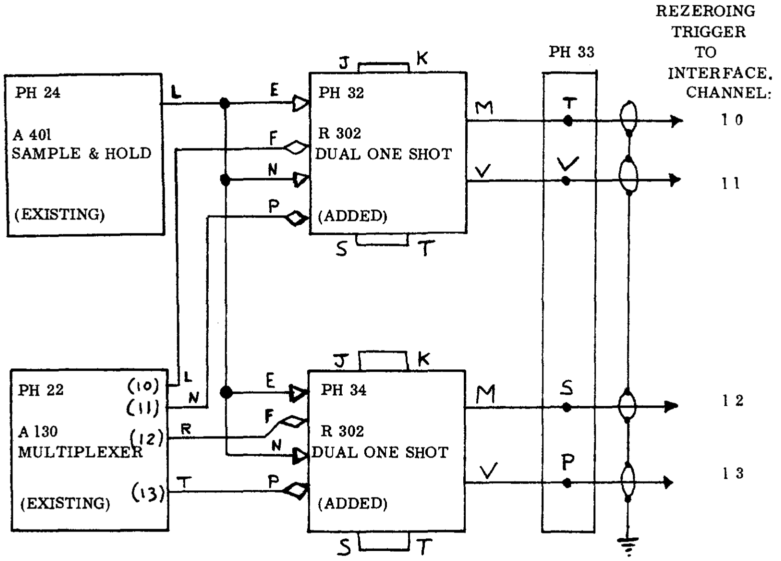
6. THE AVERAGE OF 15 SAMPLES OF " JOE ATE HIS SOUP" (AFTER PROCESSING)



7. SAME AS FIGURE 6 BUT WITH SOFTWARE SMOOTHING


VARIABLE GAIN
ANALOG SIGNAL INPUT
PREAMPLIFIER
LINEAR FULL WAVE RECTIFIER
TIME CONSTANT
SHORT
LONG
TRUE INTEGRATOR
BUFFERED OUTPUT
OUTPUT TO ANALOG TO DIGITAL CONVERTER OF COMPUTER
INTEGRATOR BYPASS OPTION
OVERVOLTAGE INDICATOR
REZEROING TRIGGER FROM COMPUTER MODIFICATION
1 5 MICRO- SECOND ONE SHOT (OPTIONAL)
OVERVOLTAGE SENSING CIRCUIT

8. INTERFACE BLOCK DIAGRAM (FOR ONE CHANNEL)



9. COMPUTER MODIFICATIONS FOR THE LINC-8 (FOR FOUR CHANNELS)

# LEXIGRAPH: AN INTERPRETIVE LANGUAGE FOR GRAPHICS

Dr. Daniel M. Forsyth
Psychology Department
University of Vermont
Burlington, Vermont

## Abstract

LEXIGRAPH is an interpretive language which places a display system and various peripherals in the hands of researchers unfamiliar with low-level languages. The interpreter accepts as input a "script" from paper tape or DECtape. The user may specify the display of text or arbitrary figures defined in the script. A wide range of script commands have been implemented. Presentation and inter-presentation times are controlled (with milli-second accuracy), and chains of displays may be generated which run off without intervening instructions. When display segments (texts or figures) are grouped in lists or strings, attributes of the individual segments (intensity, origin, etc.) may be varied selectively. Subject responses may be recorded via a response box tied to the information collector, from a Teletype, or by way of a light pen. Acceptable responses or response patterns are defined in the input script. Logical testing of responses is provided for, and full branching capabilities are included. Data (e.g., name of response key and reaction time) are recorded automatically and stored on DECtape and may be punched onto paper tape for off-line listing at the conclusion of an experiment. The script can also direct the opening and closing of selected bits in the relay buffer. This may be used, for example, to control a remotely located audio tape recorder, recording subject responses at arbitrary intervals.

LEXIGRAPH is a programming language designed to ease the task of utilizing the computer in certain classes of psychological experiments. It was originally planned as a system which would facilitate the preparation of experiments utilizing a display to present textual material to subjects (hence the name), It has been modified to permit convenient representation of two-dimensional figures, and to provide for control of peripheral devices in addition to the display scope.

The system provides, among other capabilities:

1. Simple specification of textual material and figures for display
2. Flexible control of the material displayed (e.g. timing, location, intensity, etc.), and of other peripherals (e.g. relay buffer).
3. Automatic recording of data collected during an experiment.
4. Conditional statements and program branching.
5. Program size limited only by DECtape capacity (i.e. automatic overlay).
6. A reasonably natural programming language.

The program operates in an 8K PDP-4 with at least two DECtapes, 340 display with character generator and vector mode, and millisecond clock. It is implemented in the context of the Harvard Center for Cognitive Studies Operator System and Scope Editor (DECUS PDP-4-14 and PDP-4-15). The system permits programs to be entered directly from the teletype at a remote location via the Scope Editor. The DECtape files produced by this operation are read by the LEXIGRAPH system. Thus experiments may be written and debugged (and subjects run) without leaving the scope room.

As an interpreter, LEXIGRAPH scans the source program, immediately performing the operations commanded there. Interpretation and execution is performed on a statement by statement basis. Statements in LEXIGRAPH may establish the value of a variable, manipulate a stimulus parameter, define a figure, initiate an operation, etc. A program for LEXIGRAPH, consisting of a series of statements, is called a script. Scripts are maintained on DECtape. At any given time, only a portion of a script will be in core. Statements are processed serially, the script buffer being refilled from DECtape, until a branching statement is encountered. Branching statements refer to line numbers which appear in the script (similar to FORTRAN lines). As LEXIGRAPH maintains a record of line numbers which have been encountered and their location on tape, branching can be either forward or backward. Thus looping is permitted under control of either internal or external parameters (e.g. the value of a program variable or the condition of a response device).

No attempt will be made in this paper to fully specify the LEXIGRAPH language, but selected statement types will be examined to indicate the nature of the system. All material which might appear in a script 'as is' will be presented in full caps. Arguments will be presented as 'arg1, arg2' etc.

The basic manipulanda of the system are texts and figures. They are specified simply:

TEXT argl (THIS IS A SAMPLE TEXT.)

FIGURE argl (40,20/BO,-20/40/20)

The arguments are used by statements specifying an operation, as in displaying a figure. Figures are compounded from a series of vectors specified by doublets, where the sign has the usual significance and a doublet preceded by a 'B' indicates a blanked (invisible) line. The figure above, for example, would appear as a pair of parallel lines. Texts and figures have an origin whose value is established by the statement:

ORIGIN LINE argl, TAB arg2

The scope is treated in this sense as a typewriter with tabs set 10 character spaces apart. Refinements of origin can be made, in text specifications, with spaces, and for figures, with blanked lines. Once an origin has been specified it remains as an internal parameter until changed by a new origin statement. Texts and figures are assigned the origin which is current at the time they are scanned. The origin is an attribute of the figure or text (one of several) which can only be changed by the command:

SET TEXT argl ORIGIN LINE arg2

This command would have the effect of setting the origin of text 'argl' to line 'arg2', tab 0 (the value of tab if not specified in the argument).

When the interpreter encounters text and figure declarations, it assembles an internal representation (scope buffer) and stores it in the main buffer area. This is a dynamic storage area in which is stored all material that must be preserved across statements. Scope buffers, line number locations, variable names and values, and other material is maintained in this area in a threaded list structure. If the interpreter, in looping, encounters a figure or text which has already been entered in storage, it will be ignored by the scan. The programmer may optimize storage by selectively removing material which is no longer needed, by issuing the command:

CLEAR TEXT argl,.....argn

Similar clear statements might be formed for other items in storage. Removal of items from the buffer is accompanied by automatic garbage collection so that there are never any holes in the storage area. This is accomplished by compaction and re-threading (scope buffers must be preserved as intact, although variable length, records).

Texts and figures which have been declared may be displayed by using the command:

DISPLAY TEXT argl, arg2,.....argn

DISPLAY FIGURE argl

Any number of texts or figures may be displayed simultaneously by providing a string of arguments as illustrated above. More complex forms of the display command are available, e.g.:

DISPLAY TEXT argl, arg2/FIGURE arg3, arg4

Whenever a display command is encountered, the interpreter sets up a job list with pointers to all the scope buffers involved. This insures that in executing the display command, the usual slow speed of interpretation is eliminated.

Often, in preparing stimulus material for psychological experiments, it is advantageous to organize a small number of elements into a large number of lists. This can be done in LEXIGRAPH by defining a

QUEUE argl = TEXT argl, arg2,.....argn

The queue may then be displayed:

DISPLAY QUEUE argl

As opposed to the simultaneous display of material by compounding arguments in a simple DISPLAY command, a queue is displayed one member at a time.

It is necessary at this point to consider how a display may be terminated. The display statement itself merely sets a process into operation, but does not specify when it is to stop. For a number of reasons, in the LEXIGRAPH system the terminating condition is established as a property of the individual text or figure. In the current version there are three types of conditions that can be used to signal the end of a display:

1. The subject has pressed an acceptable response key (i.e. an acceptable code has been found in the information collector).
2. The clock time alloted a display has expired.
3. By indicating the appropriate text with a light pen, a correct terminating condition has been attained.

These terminating conditions are specified in the script by a mode statement, e.g.:

MODE TACHISTOSCOPE

MODE KEYPRESS

These, together with additional mode statements, are used to set an internal program parameter; a text or figure, when encountered, will be tagged according to the mode in effect at the time. Associated with the two examples given above are specific values, e.g.:

KEY 1,2,3

TIME argl, arg2

The key statement establishes which of 14 keys on a response box must be pressed to terminate a display which is in key mode. The time statement sets the values of two internal program parameters, an 'on' clock and an 'off' clock. As with origins and modes, these values become part of the internal representation of a text or figure. It is possible to modify these values (using a SET command) after a scope buffer has been created.

Variables may be introduced:

```
ABC = 10

ABC = ABC + 10
```

line numbering:

```
LINE 100
```

branching statements:

```
GOTO LINE 200
```

and conditionals:

```
IF ABC = 50 GOTO LINE 75
```

It is possible to substitute for a real number either the value of a variable, or a random number:

```
DISPLAY TEXT 15

DISPLAY TEXT ABC

DISPLAY TEXT RANMOD 20
```

All the above are legitimate. In the second case, that text would get displayed whose label equalled the value of ABC. In the third case, a text would be displayed whose label would be some number between 1 and 20, selected randomly.

While the instruction set is not exhausted by these examples, enough material has been presented to indicate the major capabilities of the LEXIGRAPH system. A very short example may serve to illustrate the wav a script might appear:

```
BEGIN

/COMMENTS FOLLOW INITIAL SLASHES

MODE TACHISTOSCOPE

TIMES 1000,1000

/SETS CLOCKS TO ONE SECOND EACH

ORIGIN LINE 10

TEXT 100 (

      THIS IS ONE TEXT)

TEXT 200 (

      THIS IS ANOTHER)

DISPLAY TEXT 100,200

END
```

Data collection is automatic during the execution of the program. Key presses are recorded by key number and response latency (with millisecond accuracy), along with other relevant information. The value of a variable may be recorded on command, and information may be specified which will control the listing of the data after the experiment is completed, e.g.

```
PRINT THIS IS PART II OF THE EXPERIMENT
```

would cause all except the PRINT command to appear in the output.

LEXIGRAPH in its current form (version II) has been operating in the Computer-Based Laboratory of the Harvard Psychology Department for approximately one year. It has proven valuable in experiments in psycholinguistics, short-term memory, concept formation and in other areas, and has greatly facilitated use of the laboratory by individuals without experience in using a computer.

### Acknowledgment

This work was supported by Grant GM 15258, and was performed while the author was Director, Computer-Based Laboratory, Psychology Department, Harvard University.

# A MULTI-STATION DATA ACQUISITION AND CONTROL SYSTEM

Thomas H. Rau and Howard W. Borer
The Dikewood Corporation
Albuquerque, New Mexico 87106

## ABSTRACT

The system is designed to acquire and accumulate data from three sources and operate upon that data for purposes of product control and acceptance. The discussion includes system design and control, data acquisition, and data reduction. The "fresh-start" system creation, optional interrupt processes, restart capabilities, report generations, and system "shut-down" procedures are also discussed.

## SYSTEM DESIGN AND CONTROL

The Dikewood Corporation, operating within design specifications from Sandia Corporation of New Mexico, has designated the hardware and designed the software for this control system, and implementation of the system is presently progressing. The discussion of this system is logically divided into three sections: system design and control, system operation for data acquisition, and system operation for data reduction. Only a generalized theory of operation for each section is treated here; detailed questions should be addressed to the authors at The Dikewood Corporation.

The hardware components of the system include a PDP-8/I central processor with 4K memory, four DF32 disks and controller, an incremental PEC magnetic tape unit, a Kleinschmidt printer, a CR8I card reader, and an ASR-35 teletype. The function of each of these units will be discussed when they are considered. The system interfaces three product testing units (hereafter referred to as PT's) which are used as test data transmission devices; however, each unit is capable of both sending and receiving information from the central processor. Each PT is programmed by paper tape to perform groups of tests on the product. The data accumulated by the three PT's is then examined and evaluated, rather simply in real time, but more extensively later during the data reduction phase of operation. The evaluation is accomplished through the use of reports which were designed to point out problem areas and to control product acceptance.

The software system design provides three types of initiation: a completely new start, a complete system load from a previously created magnetic tape, or a simple restart from a previously operating condition. In any case, the system will allow itself to be modified before entering the actual execution phase. This modification is made possible through the use of a resident system monitor which is either loaded from paper tape (as in case A above) or loaded from disk (as in cases B and C). The system monitor contains its directive coding, a single pass Phoenix assembler, a binary loader, and disk read/write routines. Execution of this monitor will allow immediate entry into the execution monitor, the assembly of routines and punching of tapes, and/or the loading of previously punched binary tapes. It should be noted that the sole purpose of this system monitor is to facilitate the debugging and modification phases of system implementation; once satisfactory operation is achieved, it would be no longer required and could be removed from the system.

Upon completion of all or any of the options made available by the system monitor, control passes to the execution monitor which controls the operation of the data acquisition, data reduction, and optional modes of the system.

Each operational day is broken down into three distinct portions: the data acquisition section is concerned with data monitoring and accumulation (approx. 16 hours), and the data reduction and optional sections are concerned with report generations which take place during the remaining portion of the day. Since each data reduction phase concerns itself with only that data taken during the previous data acquisition phase, it would seem obvious to allow the execution monitor to pass control directly to the data acquisition mode; however, there are some optional functions within data reduction which could be of great use prior to data acquisition, and it is for this reason that the monitor will query the operator at the beginning of the day as to which of the three modes he wishes to enter.

## DATA ACQUISITION

There are four types of interrupt in the data acquisition mode of system operation: PT, teletype, Kleinschmidt, and card reader. With the exception of servicing the keyboard interrupt which initiates a "shut-down" of data acquisition, time of service is of paramount importance. The interrupt routine is entered immediately after each interrupt is received and saves the location and cell contents necessary to assure correct return after the interrupt is satisfied. The routine also determines the source of the interrupt and directs control accordingly. Priority is internally established to set all PT's at an equal level, the teletype second, and the card reader last. As soon as a Kleinschmidt interrupt is received, it is turned off and execution is resumed. Since a real-time activity summary is the only item which is allowed to be printed during data acquisition, and since that printing involves only a single character at a time, we can effectively "ignore" this interrupt. The card reader will process cards only when the system is free from any PT considerations.

The bit configuration presented at the initiation of a PT interrupt is held only for one second and may degenerate after that time, therefore it is mandatory to complete all computations involving these bits within that time. However, a further restriction exists in the fact that interrupts may occur from each of the three PT's simultaneously. This requires that all computations be completed within 333 milliseconds. To allow for adequate safeguards, this time has been further reduced to 300 milliseconds of processing; all other interrupts are "closed out" and will not be recognized until the processing is completed.

All data arrays and counters must be initialized prior to the start of data acquisition due to the fact that, for purposes of data reduction, only daily data is retained. An in-core buffer is maintained for each of the PT's: each data buffer is composed of an ID block and results of all tests conducted in a group. Since data is presented from the PT in 48 bit patterns, the ID (consisting of sixteen words) must be collected by means of four samplings of the 48 bit register. The remaining portion of the buffer is filled four words at a time as each test is completed and the entire buffer is copied onto the disk at the end of a test group.

The processing of the PT data must necessarily accomplish three tasks; the first of these consists of properly identifying the source of the incoming data; that is, which of the three PT's is sending data. This is accomplished by testing each of the three PT flags in order. Since it has been established that processing of an interrupt will be completed within 300 milliseconds, it is possible to test each flag every time an interrupt is received without establishing a bias for the first PT. The second task involves the generation and checking of all system flags and counters. One or many of these are updated or initialized each time an ID is received, each time a test measurement is received, each time a group is completed, each time some signal is to be returned to the PT, and each time a buffer is dumped onto disk. All flags are stored for each PT due to the fact that there is no relation between the activity of one PT and the processing occurring on either of the other two. The third task requires completion of an entry into the real-time activity buffer. This buffer is filled whenever appropriate information is received and is printed, one character at a time, whenever the system is not concerned with other interrupt conditions.

Upon receipt of a keyboard interrupt, the data acquisition phase must accomplish completion of all test groups currently operating and then set up a restart capability for possible use when all options have been completed. Successful completion of the "shut-down" procedure must be verified by an operator-inserted code before data acquisition is abandoned. Once verification is received, control is returned to the execution monitor and the appropriate data reduction or optional program is then loaded and executed.

## DATA REDUCTION AND OPTIONS

As previously mentioned, the operational day is broken down into three distinct parts: data monitoring and accumulation, report generation, and options. This section deals with the latter portions of the day. The time allotted to these parts is approximately 8 hours, since another data acquisition period will follow.

There are two kinds of reports to be printed during this time: mandatory and optional. Mandatory reports consist of a daily failure list, a statistical summary of the data, a product summary, and tape copy functions. Each of the reports is generated automatically by a separate program which resides on disk. The loading and execution of each program is handled by the data reduction monitor. When a particular report is desired, its respective program will be loaded from the disk and executed, with control returning to the monitor program in order to process the next necessary program in the same manner.

The first operational step is to request that the operator mount a magnetic tape on the transport. Following a message to that effect, the four mandatory report programs are loaded and executed in order by the monitor program. After these functions are complete, a check is made to insure that the tape is ready. The monitor then controls the creation of the three tapes: an IBM 360 compatible tape containing the data taken during the last data acquisition phase, a complete data history tape, and a system tape to be used in the event the system and its associated data are destroyed.

Upon completion of this tape generation, all of the mandatory tasks have been accomplished and the next step is to satisfy any optional requests which the operator may make. A list of possible reports and their respective code numbers will be available to the operator. He will enter the code number of the program to be executed and the system will then load the program from disk, execute it, and return to accept additional requests. Some of the optional programs are: an individual test summary, an individual group summary, an individual unit summary, various test analyses, changing and printing of test limits, and a software/hardware program check. When the operator indicates that he has no further requests, he has three options at his disposal. He may request that the system be turned off to enable hardware diagnostics or other problem programs to be run; he may set the system in a "wait" state in preparation for the next data acquisition period which is probably only a few minutes away; or he may choose to continue data acquisition in which case initialization is skipped and data acquisition would proceed as before. This latter fact will probably occur only when the data acquisition process was interrupted to perform some optional tasks.

To summarize then, the execution monitor exercises complete control over each of the three subsystems (data acquisition, data reduction, and optional programs), allowing each to terminate under program control or at the discretion of the operator. Upon termination of any subsystem, it is possible to enter any of the other two, thereby affording complete flexibility to the system.

# VIDAC
# A DATA ACQUISITION SYSTEM

Noel P. Lyons, Robert W. Skyles
VIDAR Corporation
Mountain View, California

## ABSTRACT

VIDAC is a data acquisition system program for the non-sophisticated computer user. Its features include flexibility of sub-routine usage, a linking loader, and an easily modified executive routine.

## VIDAC

VIDAC is a real-time programming system developed for operating a processor-controlled data acquisition system manufactured by VIDAR Corporation. VIDAR is a manufacturer of a broad range of test instrumentation and data handling equipment. One of VIDAR's major lines is a group of digital data acquisition systems or data loggers.

In surveying the uses of our data logging products, it became obvious that the majority of the data being recorded was subsequently analyzed on a central computer. In 1967 VIDAR first combined a data logger with a small computer in one integrated system.

At that time, a survey of available equipment indicated that while many computer manufacturers offered A-D converter front-ends, none of these provided the sensitivity and noise immunity required for direct measurements of typical industrial/scientific transducers such as thermocouples, strain gages, load cells, etc. Subsequent discussions with DEC led to the development of the DEC AF04 Scanning Digital Voltmeter for the PDP-8 and PDP-9 families of computers. As far as we know, this was the first commercial A-D front-end for a computer developed by an instrumentation manufacturer.

Concurrent with this effort, VIDAR embarked upon development of the VIDAR 5206 - a proprietary data-acquisition system using a PDP-8/S to bring the power of a small computer to the data acquisition customer. Our product planning indicated that the software system would be as critical as the hardware system. The total system would be used by customers skilled in their field of instrumentation, but having little or no knowledge of computer programming. Typical applications for this class of equipment included:

Test monitoring (by a jet turbine manufacturer)

Product Test (by a consumer electronics manufacturer)

Fuel Cell Research (by NASA)

Geological Research (at a mid-western university)

Process Supervision (in a glass manufacturing plant)

Thus, it became obvious that a "total system" approach to hardware and software would be required. Using such an approach, we can examine the hardware and software parameters which such a total system should have:

### Hardware Parameters:

1. Sub-microvolt sensitivity for direct measurement of T/C's, strain gages, etc.
2. High noise immunity (true integrating measurements) to withstand noisy industrial environments.
3. High common-mode voltage rejection, to allow for long leads to transducers.
4. Fast and accurate, to match modern computer speeds.
5. Modular, to allow each user to configure his system to meet his exact requirements with a minimum of special equipment.
6. Reliable. Neither the customer nor the vendor ever profits from a service call.

### Hardware Results

The Resulting System Shown in Figure 1 can:

1. Measure down to 0.1uv.
2. Offer true integrated measurements, period selectable, from 1-2/3ms to 166-2/3ms.
3. Furnish up to 160db of common-mode rejection at common-mode voltages up to 500v.

4. Random Access, from 10 to 1000 channels at speeds of 50 channels/second.

5. Figure 2 shows the various optional equipment that can be added to configure systems of 10 to 1000 input channels, optional measurements of DC and AC voltages, frequencies, time intervals, and resistance. The PDP-8 family of processors offers complete control of random channel-by-channel measurement selection, function and speed.

Software Parameters

1. Control experiment or test in process.....

2. Acquire the data, - both controlling the measurement system and bringing the data into the processor.....

3. Manipulate the data, with full arithmetic and logical capability.....

4. Output results to recording peripherals, display panels and control outputs.

5. The software system must be so structured that users who know their data acquisition task, but not computer programming, can readily combine the hardware and software to apply the full power of the "total system" to their instrumentation problems. Thus, software should be written in a data acquisition oriented language with the most common tasks pre-programmed by subroutines.

6. Typical applications to be easily programmed on a 4K PDP-8 family computer.

Software Results

The result has been the development of VIDAC--a real-time assembly-language programming system for the VIDAR 5206. The system is built around a relocatable assembler, a linking loader, and a data-acquisition oriented subroutine library.

After much study, an assembly level language was chosen instead of a compiler level language. Real-time data acquisition and manipulation problems are intimately associated with what is happening in the computer. Compiler level languages were all found to be too far removed from the real-time world to be easily adapted or extended to real time processing.

A second factor was the ease in which logical functions and manipulations can be handled by an assembly language. VIDAC concentrates on the logical not the mathematical manipulation of data. In addition, the efficiencies of speed and space available to the non-expert user with assembly language were an added bonus.

VIDAC is not an assembler in the strictest sense in that each mnemonic instruction generates only one machine instruction. VIDAC uses a fleshed-out version of PAL III with the addition of 19 convenient pseudo operations. An automatic paging feature eliminates the novice programmer's problems with only 256 directly addressable locations. VIDAC programs are written as if 4096 locations were directly addressable.

The output of the VIDAC Assembler is a relocatable binary tape. The VIDAC Linking Loader automatically links together the user's program and the VIDAC system subroutines. This allows the programmer to use any VIDAC system subroutine or any subroutine he has created without the usual problems of off page references and subroutine locations.

The VIDAC software package contains over 50 prewritten subroutines to handle the control, acquisition, manipulation and recording of data on a real-time basis. A typical data acquisition system will use about 30 of these routines.

The main subroutine in the VIDAC library, referred to as the VIDAC System Module, is capable of performing all the tasks common to the majority of data acquisition and control applications. The VIDAC System Module is a real-time monitor program capable of handling three distinct types of program interrupts associated with real-time processing and control. These interrupts represent the most demanding aspect of the tasks which the System Module must perform. At the first level, there are a large number of program interrupts which go back and forth between the selection and measurement front-end, the Teletype, a software clock, and the various I/O peripherals. Next, the System Module itself generates program-independent time interrupts that allow unconditional jumps to pre-selected program segments at preset time intervals. Naturally, this type of interrupt requires saving the state of processor, including the contents of the Floating Point Accumulator, to allow restoration at a later time. Finally, the System Module must service Priority Process Interrupts in which the user may, on a priority basis, interrupt the program and request a reading of individual input channels from the Teletype keyboard.

The system module contains all the subroutines necessary to randomly select and measure any sequence of input points. These subroutines fully control the data acquisition hardware, including the many optional hardware modules.

Additional subroutines allow the user to easily establish a multilevel real-time interrupt sequence to easily handle the variety of timing requirements of various data acquisition routines.

In addition to handling all I/O and program interrupts, the VIDAC System Module also contains a functionally complete set of Floating Point Arithmetic Subroutines which handle 8 decimal-digit numbers over the range of $10^{-38}$ to $10^{+38}$. The speeds of the Floating Point operations vary between 8 and 15ms.

Experience has shown that these times are quite compatible with real-time data acquisition and manipulation requirements. Floating Point subroutines eliminate number system scaling problems that would otherwise plague a novice programmer.

The balance of the VIDAC Subroutine Library contains additional routines for integer arithmetic, data conversion for non-linear functions, utility routines for optional peripherals such as the DEC DF32 disc memory, field conversion routines with FORTRAN-like format specifications, and other miscellaneous logic routines.

Another important aspect of the VIDAC system is that the large library of relocatable subroutines allows a simple logical programming framework to be followed for each data acquisition problem. This framework is directly related to the real-time problem, with minimum consideration of the computer. Figure 3 is a listing of a complete user program to measure and type out 200 channels of data. This example demonstrates the basic programming framework that can be used with VIDAC to set up much more complicated data acquisition and manipulation tasks. Figure 4 contains segments of various user's programs showing the ease of accomplishing various other real-time functions and adding them to the basic programming framework.

Thus, the task of programming a highly complex and sophisticated real-time hardware system has been reduced to a straightforward sequence of writing an assembly-language program using the problem-oriented VIDAC language (over half of a typical such program consists of "calls" to library subroutines), assembling the program via the one-pass VIDAC Assembler to a relocatable binary tape, loading this program together with the VIDAC System Module and any other library or user subroutines required, and pressing START.

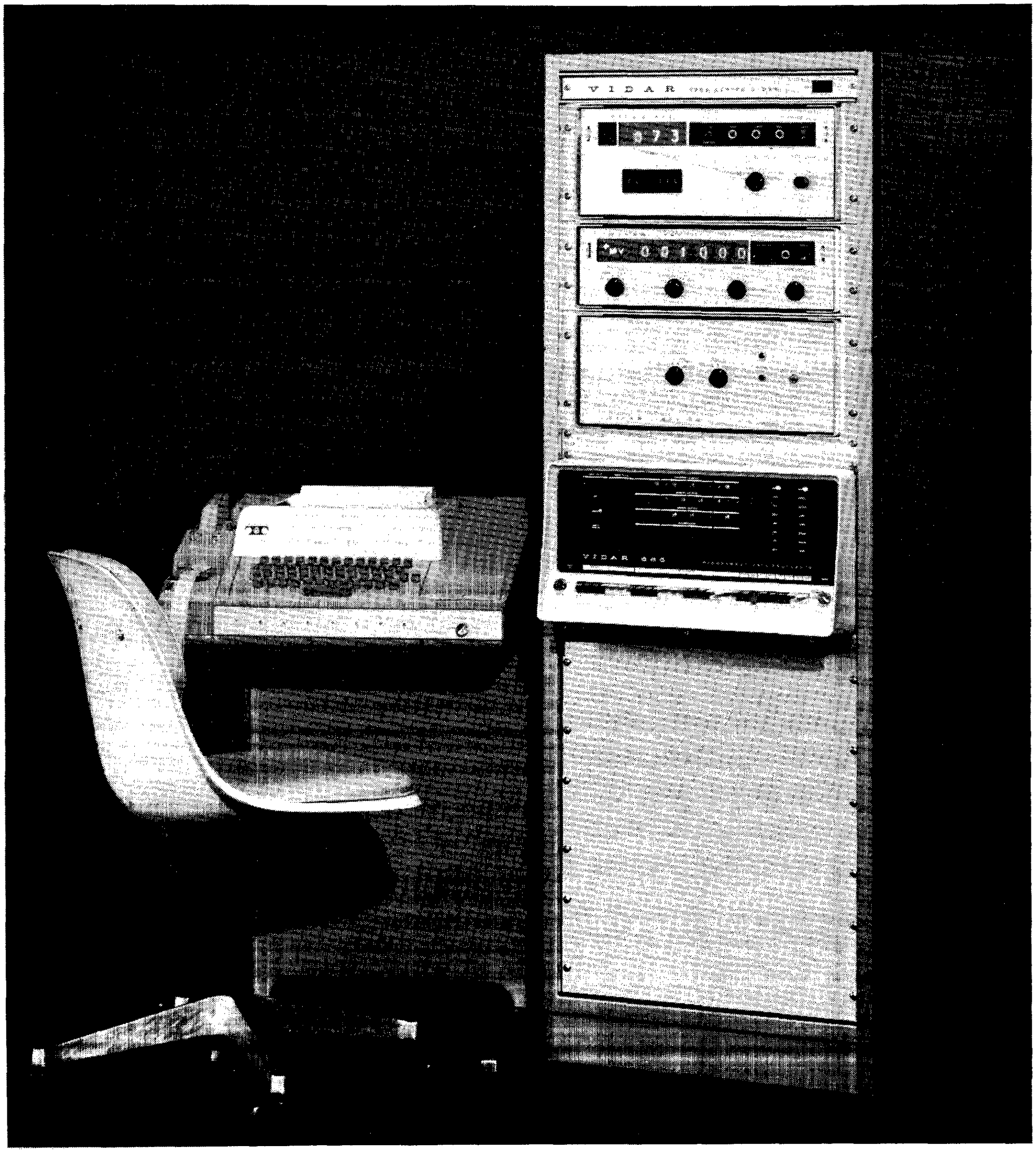


Figure 1 - VIDAR 5206

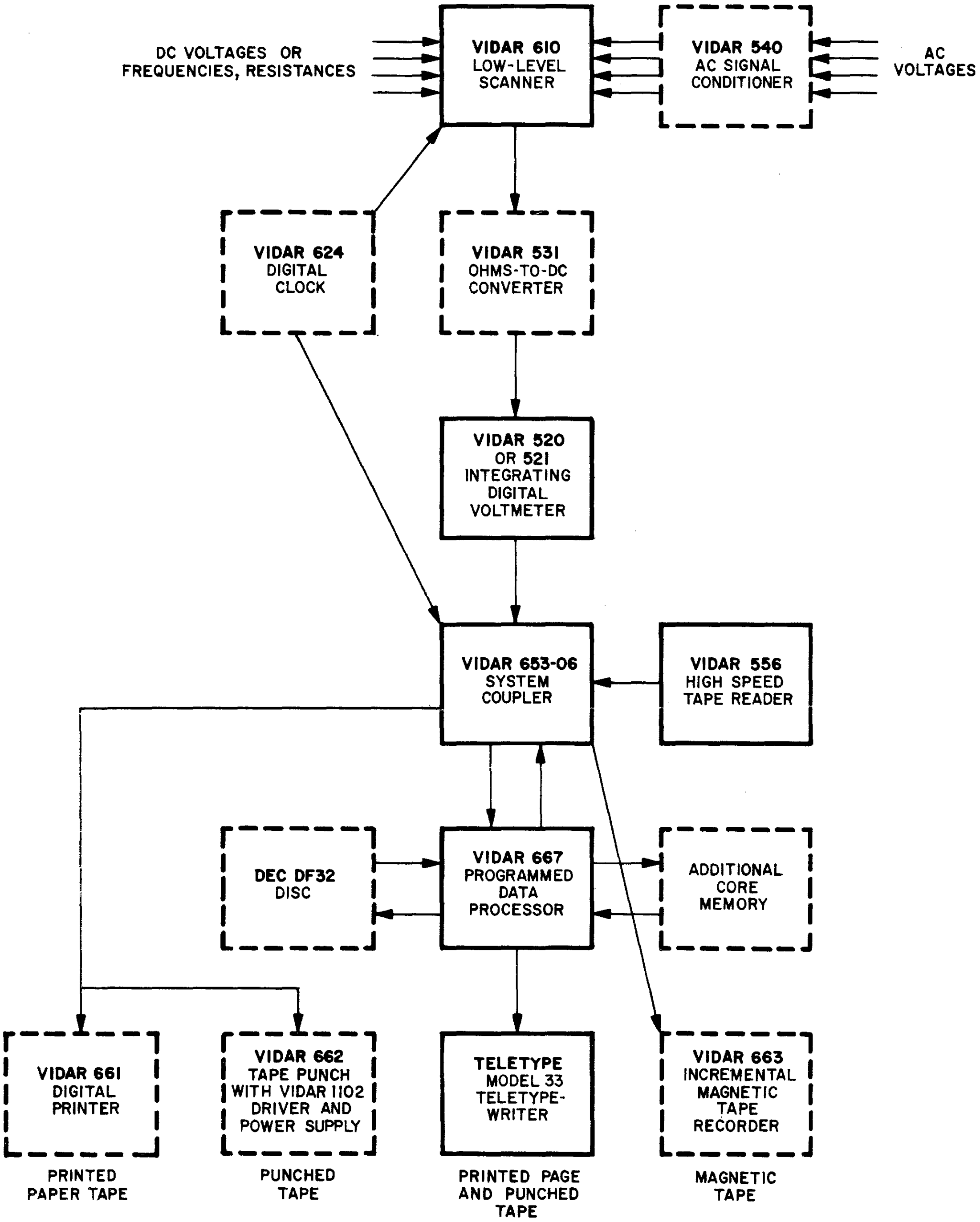

DC VOLTAGES OR FREQUENCIES, RESISTANCES
VIDAR 610 LOW-LEVEL SCANNER
VIDAR 540 AC SIGNAL CONDITIONER
AC VOLTAGES
VIDAR 624 DIGITAL CLOCK
VIDAR 531 OHMS-TO-DC CONVERTER
VIDAR 520 OR 521 INTEGRATING DIGITAL VOLTMETER
VIDAR 653-06 SYSTEM COUPLER
VIDAR 556 HIGH SPEED TAPE READER
DEC DF32 DISC
VIDAR 667 PROGRAMMED DATA PROCESSOR
ADDITIONAL CORE MEMORY
VIDAR 661 DIGITAL PRINTER
VIDAR 662 TAPE PUNCH WITH VIDAR 1102 DRIVER AND POWER SUPPLY
TELETYPE MODEL 33 TELETYPE-WRITER
VIDAR 663 INCREMENTAL MAGNETIC TAPE RECORDER
PRINTED PAPER TAPE
PUNCHED TAPE
PRINTED PAGE AND PUNCHED TAPE
MAGNETIC TAPE

Figure 2 - VIDAR 5206 Block Diagram

```
*
*  VIDAC SAMPLE USER PROGRAM 2
*
*  PROGRAM SCANS AND TYPES OUT CHANNELS 0-199
*
        ENTRY   SAMPL2
SAMPL2: CALL    0,\OP    INITIALIZE THE SYSTEM MODULE
        TAD     =-D200   MINUS NUMBER OF CHANNELS
        DCA     DCTR     TO A DOWN COUNTER LOCATION
        DCA     CHAN     SET FIRST CHANNEL TO ZERO
        CALL    1,VIDAR  START INTEGRATION OF CHANNEL ZERO
        PAR     CHAN     CHAN IS THE BASE OF THE PARAMETER VECTOR
*
*  THE FOLLOWING IS A LOOP EXECUTED FOR EVERY CHANNEL
*
LOOP:   ISZ     CHAN     ADD ONE TO CHANNEL NUMBER
        CALL    1,VIDAR  START INTEGRATION OF NTH CHANNEL
        PAR     CHAN
        CALL    0,VRSLT  GET THE RESULT FROM THE N-1 TH CHANNEL
        DCA     CH       N-1 IS IN THE ACCUMULATOR SO SAVE IT
        CALL    1,\STO   THE RESULT IS IN THE FLOATING AC SO SAVE IT
        PAR     RSLT
        CALL    0,\TTO   AC IS CLEAR SO TYPE A RETURN-LINE FEED (CODE 00)
        CALL    2,IOUT   PRINT THE CHANNEL NUMBER
        PAR     CH       FIRST ARGUMENT IS ADDRESS OF NUMBER TO PRINT
        PAR     IFMT     SECOND IS ADDRESS OF FORMAT WORD
        CALL    2,FOUT   NOW TYPE THE RESULT
        PAR     RSLT     FIRST ARGUMENT IS THE ADDRESS OF NUMBER TO PRINT
        PAR     FFMT     SECOND IS ADDRESS OF FORMAT WORD
        ISZ     DCTR     CHECK THE DOWN COUNTER TO SEE IF MORE CHANNELS
        JMP     LOOP     THERE ARE MORE CHANNELS LOOP AGAIN
*
*  ALL THE CHANNELS HAVE NOW BEEN PRINTED
*
        CALL    0,\CK    WAIT FOR ALL PRINTING TO CEASE
        HLT              THEN STOP THE COMPUTER
        JMP     SAMPL2   RESTART PROGRAM IF OPERATOR HITS CONTINUE
*
*  THE FOLLOWING IS ALL THE VARIABLE STORAGE
*
DCTR:   BSS     1        DOWN COUNTER FOR COUNTING CHANNELS
        IFF     4        PARAMETER VECTOR
CHAN:   BSS     1        CHANNEL NUMBER
        PAR     1        FUNCTION 1 - DC
        PAR     7        RANGE 7 - AUTORANGING
        PAR     2        RESOLUTION 2 - 16.6 MS
CH:     BSS     1        CHANNEL RETURNED FOR PRINTOUT
RSLT:   BSS     3        FLOATING POINT RESULT
IFMT:   PAR     0400     INTEGER FORMAT WORD FOR AN I4 FORMAT
FFMT:   PAR     1206     FLOATING POINT FORMAT FOR AN F10.6
*                        NOTE THAT 12 OCTAL IS 10 DECIMAL
        END
```

Figure 3  Typical Program

```
*
*   SELECTED AUTOMATICALLY EVERY 5 MINUTES
*
         ENTRY SAMPL5
SAMPL5:  CALL     Ø,\OP    INITIALIZE I/O
         CALL     2,SETIM  SET THE TIME ROUTINES TO GO TO LOCATION
         PAR      =D6Ø     LABELED WORK IN 6Ø SECONDS
         PAR      WORK
*
*   FOLLOWING IS CALLED AUTOMATICALLY EVERY 3ØØ SECONDS (5 MINUTES)
*
WORK:    CALL     2,SETIM  SET UP SO WE WILL RETURN HERE IN 5 MINUTES
         PAR      =D3ØØ
         PAR      WORK
         CLA               FIRST CHANNEL
         DCA      CHAN
         TAD      =-D1ØØØ  NUMBER OF CHANNELS TO AVERAGE
         DCA      INDX
         CALL     Ø,\CL    CLEAR FLOATING POINT ACCUMULATOR
         CALL     1,\STO   SAVE IT IN THE SUM
         PAR      SUM
```

```
*
*   NOW COMPUTE THE AVERAGE
*
         CALL     1,\FAD   PLACE THE SUM IN THE FPAC
         PAR      SUM
         CALL     1,\FDV   DIVIDE BY THE NUMBER OF CHANNELS
         PAR      TEN
         CALL     1,\STO   AND STORE THE RESULT
         PAR      AVG
*   NOW PRINT OUT THE RESULTS
*
         CALL     1,TYPE   PRINT "HIGH"
         PAR      TXT1
         CALL     2,FOUT   PRINT OUT THE HIGH RESULT
         PAR      HIGH
         PAR      FMT1
         CALL     1,TYPE   PRINT "LOW"
         PAR      TXT2
         CALL     2,FOUT   PRINT OUT THE LOW RESULT
         PAR      LOW
         PAR      FMT1
         CALL     1,TYPE   PRINT "AVERAGE"
         PAR      TXT3
         CALL     2,FOUT   PRINT OUT THE AVERAGE
         PAR      AVG
         PAR      FMT1
         CALL     Ø,\CK    WAIT FOR ALL I/O TO COMPLETE
         HLT               AND HALT
         JMP      SAMPL3   EXECUTE PROGRAM AGAIN IF OPERATOR HITS CONTINUE
```

Figure 4  Typical System Timing and Data Manipulation Routines

# EXPO - A FLEXIBLE PDP-8 DATA-ACQUISITION PROGRAM

Bruce Arne Sherwood
California Institute of Technology
Pasadena, California

## ABSTRACT

EXPO is a PDP-8 program which reads various kinds of data from experimental apparatus, optionally logs data on magnetic tape, and accumulates one - or two - dimensional histograms of selected variables; these histograms may be displayed on the teletype or scope, simultaneous with data acquisition. From the keyboard the user defines what variables are to be histogrammed, and under what conditions; variable names are symbolic and numerical parameters are decimal. Also from the keyboard the user may call for teletype or scope output with some control of format. Because of its flexible user-oriented input-output, EXPO has proven to be very useful in debugging and utilizing complex apparatus in a high-energy physics experiment; it is likely to be useful in similar experimental situations in science or engineering. The paper includes a useful general discussion of interrupt handling on the PDP-8.

## INTRODUCTION

### How to Read This Paper

The section on "Basic Structure" gives an overview of the progr^m EXPO. "Input-Output Interrupt Handling" should be of general interest to the serious reader, whereas "On-line Analysis" is perhaps of more special interest to experimenters as a suggestion of useful on-line compilation-like techniques and as a guide to the actual use of EXPO, which will be available from the DECUS program library. An appendix on operating instructions details the use of the various keyboard control options.

### Required Hardware

EXPO requires a PDP-8 with 4K of core, hardware multiply/divide, and a teletype; also needed is an interface to read experimental data. A magnetic tape unit for data-logging is optional. A standard scope, while optional, is extremely useful. EXPO will also handle as an optional device a point-plotter which shares the scope's x - and y - buffers. (The plotter should interrupt when ready, but the scope shouldn't.)

### Program Size

EXPO occupies cells 0-7177$_8$ (with a few holes), and the rest of core is used for magtape buffers and histogram storage. In many applications it will be possible to have a considerably larger histogram area; this will be discussed later.

## BASIC STRUCTURE

For those familiar with the terms, the basic strueture of EXPO may be described as a multiprogramming scheme with different tasks connected to the various interrupts; the foreground consists of tasks which read data, log data on magtape, and handle other external devices, while the background consists of a routine which samples the incoming data and constructs histograms from this data. This succinct description is offered as orientation to some readers; the technical jargon of multiprogramming will not be used in what follows.

EXPO handles a situation common in many experimental arrangements: Data from some apparatus arrive at a high rate, either continuously or in bursts, and must be logged on magtape (for later analysis) as efficiently as possible to avoid excessive "dead-time" (time when no data can be acquired because the computer is busy logging previous data). At the same time, rather extensive on-line analysis of this data is desired in order to monitor the quality of the acquired data and to suggest possible changes in data-taking strategy. This on-line analysis should not itself create large dead-time; first priority must be given to acquiring and logging data for off-line analysis. This implies that the on-line analysis be performed on only a fraction of the data when data rates are high; this should of course be an unbiased sample.

Efficient data-logging is achieved by EXPO through double-buffering: The data-reading routine stores directly into one of two magtape buffers. As soon as one buffer is full its transfer onto magtape (through data-break in our case) is triggered; while this record is being written, the data-reading routine stores into the other buffer. In this way there will be significant dead-time only if data rates exceed tape-writing speeds.

For the on-line analysis of sample data the analysis routine must have access to two parameters in the data-reading routine: the pointer which shows where data was most recently stored in the tape buffers, and a flag (i.e., a cell whose contents are 1 for "set" or 0 for "reset") which indicates that some data have been read and are available for sampling. When the analysis routine is idle with no data available to work on (for example, before the first data arrive), the routine simply waits in an infinite loop asking vainly whether the flag is set. (See Fig. 1.) When a data interrupt occurs this loop is interrupted while the data reading routine stores new data into the tape buffer, advances the pointer to be ready for the next data interrupt, sets the flag, and exits from the interrupt. This brings us back to the loop, where it is now found that the flag is set, whereupon the flag is reset and data located by the pointer are moved (copied) from the tape buffer to a "sample buffer". (After moving, the flag is checked. If set, the flag is again reset and the data moved using the new pointer value. This is repeated until the flag stays reset during the move, insuring that the data in the tape buffer weren't changed during the move.) Once the data are in the sample buffer the analysis routine may spend a long time digesting the data, including making histograms, during which time many more data interrupts may occur with attendant data-logging; these data are lost to the one-line analysis but saved for off-line analysis. When the sampled data have been fully analyzed, the analysis routine returns to the loop, asking whether the flag is set for more data.

The important thing to notice is that, with the exception of one instruction to set the flag, the data-reading routine carries none of the burden of the sampling procedure; hence it is freed to perform the vital data-logging function as efficiently as possible.

It may be useful to point out that just as the data-reading routine is initiated by a data interrupt, so the on-line analysis is initiated by a software pseudo-interrupt, the condition of the flag being set. This pseudo-interrupt is triggered by the data-reading routine (by setting the flag). If it should be desirable not to waste time idling in the flag-test loop, it would be easy to connect the analysis routine to an actual hardware interrupt. All that is needed is an external flip-flop, connected to the interrupt bus, which could be set by a command from the data-reading routine. (The connection to the interrupt bus should be gated to permit the analysis routine to disable this interrupt once acknowledged.)

In addition to the tasks performed by the data-reading, data-logging, and on-line analysis routines, there are other tasks initiated by interrupts from the teletype (keyboard and typer) and from the plotter. The keyboard interrupt actually may be thought of as many different interrupts, since depressing different control characters initiates different tasks.

In the next sections sufficient details of this overall scheme are given, with reference to the program listing, to permit the reader to use and modify the program. The program listing will be available through the DECUS library. EXPO is written in the language of the Lawrence Radiation Laboratory assembler, DECUS 5-13. This is similar to but less flexible than PAL; however, because assembly is performed on a large computer, a large symbol table is permitted and a very useful cross-reference listing is provided. In addition to the listing there will also be available a BIN paper tape and the source card deck.

## INPUT/OUTPUT INTERRUPT HANDLING

Fig. 2 shows a more detailed flow diagram of the way EXPO handles input/output through interrupts from the various external devices. Initially the program is started at DRONE, address 1200. DRONE enables the interrupt and goes into an infinite loop waiting for data to histogram. When an interrupt occurs, a search is initiated to determine which external device was responsible; this is done in INTSCH on p. 200 (page 200 is the page whose initial address is $200_8$) without disturbing the AC (accumulator) or L (link). Once the device has been identified, the AC, L, and return address (contained in cell 0) are saved in cells unique to the device. Each device saves the registers in its own stack, which permits another device to interrupt before the first device-handling routine is finished, without losing the original return address to DRONE. The only other necessary precaution is to prevent the same device from interrupting its own interrupt-handling routine and thus overwriting the saved register stack, which would induce an infinite loop. In the case of the teletype (both keyboard and typer) and the plotter, the interrupt device flag is cleared before the ION is given, thus preventing a second interrupt from the device. In the case of the magtape and data, the ION isn't given until servicing is complete, since these devices must be handled quickly without interruption. In Fig. 1 the notation "return" includes restoring the saved registers. (If no device is identified, INTSCH goes to NOINT at 4760, which rings the teletype bell in warning.)

### Data

If a data interrupt occurs, a check of a "tape" bit in the data is first made to see whether to log this event on magtape (Fig. 2). If so, DATA (p.3600) reads from the data interface into the tape buffer (and triggers tape output if the buffer is now full), then sets the "move" flag. The "move" flag (MOVEFL) tells DRONE to move newly acquired data from the tape buffer to the sample buffer for histogramming. If the new event is not to be logged on tape and the sample buffer is empty, DATA reads directly into the sample buffer and sets the "sample" flag (SMPFLG), which tells DRONE that the sample buffer contains data to histogram. This is slightly more general than the simplified scheme discussed above and shown in Fig. 1, where for clarity it was assumed that all events would be logged. Note that a full tape buffer may hold many events, while the sample buffer holds only one.

If DRONE during its infinite loop finds the "sample" flag set, UNPAK (p. 5000) unpacks the compact sample buffer data into as many separate variables as are contained in the data. For example, one 12-bit work might unpack into two six-bit pulse heights; another word might unpack into twelve separate logical variables, each with value 0 (false) or (true). After unpacking, the sample buffer is no longer needed, so the "sample" flag is reset, enabling DATA to reload the sample buffer. Then IF (p. 5200) checks whether selected variables fall into the ranges previously defined as permissible by the user; if so, HIST (p. 3200) histograms this event in those correlations previously defined by the user and then returns to DRONE.

If DRONE encounters the "move" flag set, it sets the "sample" flag to prevent DATA from reading into the sample buffer. Next it resets the "move" flag and moves data from the tape buffer to the sample buffer. After moving the data DRONE asks whether the "move" flag has been set by DATA during the transfer. If so, a new attempt is made to transfer the data, since the tape buffer may have changed part way through the transfer. When the "move" flag is found still reset after the moving, the transfer is successful and unpacking of the sample buffer begins.

Typer

Typer output is performed from buffers containing character strings. Histogram output requires considerable computation to set up each line of type, and this is done one line at a time so that the buffer may be small. DRONE in its cycle checks whether histograms are being typed out and, if so, whether the present line has been fully typed. If needed, DRONE call TMXOUT (p. 5600) to set up the next line and initiate output. The actual character-by-character typing under interrupt control from the buffer is handled not by DRONE, but by TRIGR (p. 600) and by TYPT (p. 1000); these routines will be described in detail later. During typing the interrupt is off for only 40 microseconds per character.

Scope and Plotter

Suppose for the moment that the system includes a scope but no point plotter. If DRONE in its cycle sees that the scope is being used, it calls a scope-handling routine to set the scope's x - and y - buffers and intensify to display the next point; then EXPO goes back to DRONE. Hence while the scope is in use a new point is plotted each time DRONE completes a cycle; the entire scope display with all its points is repeated continuously to provide a steady picture. One of the scope-handling routines is TITLEI on p. 3000 which writes character-string titles. (These strings have the same format as those used for typing out; see description below.) TITLEI furnishes successive characters to OSCHAR (p. 1400) which plots them as a 3 x 5 matrix of points.(OSCHAR is a Lawrence Radiation Lab subroutine.) The character patterns are stored on p. 2400, one character to two words; only 15 of the 24 bits are meaningful. The other main scope routine is POINTS (2300) which plots a column or row of histogram data as a graph, labeled by TITLEI. The vertical scale of the graph is determined by the switch register setting, whose decimal equivalent appears as part of the labelling; this scaling feature is very useful. (The rather time-consuming DIVIDE routine (cell 6543) which sets up the scaling factor is entered only if, after plotting the last of the points, the switch register is found to have changed from its previous value. In that case the next entire display cycle will use the new scale factor.)

The point plotter must be handled somewhat differently. It uses the same x - and y - buffers as the scope but differs vastly in its response time: the scope can plot $10^5$ points/sec, whereas the plotter can complete 5 points/sec. Hence the scope is ready every time DRONE comes around and should not be tied to the interrupt bus, but the plotter must interrupt when ready. If the plotter is used, DRONE does not enter the scope routines; these are entered instead from a plotter interrupt as shown in Fig. 2. A plot command issued after x and y are set causes the plotter to move the pen to the new location, plot, and cause an interrupt upon completion. In order to make a plot there must first be a scope display running; then a keyboard command sets the necessary flags (PLTFLG, PLFLGI) so that the next complete series of scope points goes onto the plotter. After one pass through all the points the plotter is disengaged and EXPO reverts to the fast scope display.

Keyboard

Initially EXPO waits in DRONE for instructions from the keyboard. Depressing a key causes an interrupt which is handled by KEYBD (p. 400); see Fig. 3. If the CTRL key is depressed together with a letter from A - Z (but not M; CTRL-M = 215 = carriage return), KEYBD types out the letter preceded by an asterisk and computes a jump to one of the control routines; otherwise KEYBD merely echos back the character to permit writing comments. The control routine addresses are listed starting at 444; unused letters correspond to KBDRET ( at 552) which is the return from the keyboard interrupt.

Some control routines, such as CLEAR (clear histograms and event counters), do not require further information from the user to perform their tasks. In that case, upon completion of the task the routine simply jumps to KBDRET to exit. Other control routines, such as HALT (halt after N events), require additional input: in this case, a four-digit decimal number N specifying the number of events to be histogrammed before halting. To acquire this needed data, HALT (cell 4734) calls GETNAM (p. 4400); see Fig. 3. GETNAM modifies KEYINT so that future keyboard interrupts will take an abnormal JMP* KEYINT (at cell 407) into the GETNAM routine at GTNM (cell 4412), where the new character is read and added to the input string; then exit to KBDRET is taken to await the next character. A space or CR (carriage return) signals GETNAM that the input character string is complete, in which case instead of exiting through KBDRET the GETNAM subroutine restores KEYINT to its normal value and finally returns to the HALT routine. There the input data is used to complete the task of the HALT control routine; the decimal input string is converted to octal, made negative, and used to initialize an event counter. Then return is made through KBDRET. The next keyboard interrupt will take the normal path through CNTRL (cell 410).

Magnetic Tape

TRGTAP ( p. 6600) sets up the calling sequence to the tape handler RECORD (p. 7000). The non-standard magtype interface was designed and built at CalTech; transfers occur through 3-cycle data break, and the word count and address registers are at 7776 and 7777. Tape commands are given by issuing a 6734 instruction for unit 2 (or 6714 for unit 1) together with appropriate bits in the accumulator to specify write or read, rewind, set ready, etc. Completion of transfer on read or write generates an interrupt which is identified in INTSCH by a 6721 instruction (6701 for unit 1). A few other IOT's are used to check parity, etc. RECORD is entered with AC bits to specify read or write, unit 1 or 2, interrupt on or interrupt off operation. TAPE, the main subroutine within RECORD, is also occasionally called directly by EXPO for unusual operations such as backing up over a record. The subroutine RECORD is included in the listing as an aid to determining what kind of tape - handling routine other PDP installations will need; some of the programming techniques used in RECORD may also be of general interest.

It should be pointed out that DATA (p. 3600) calls TRGTAP in such a way that tape output is double-buffered with consequently very little dead-time due to logging the data on magtape.

Before calling TRGTAP, DATA call TWAIT (at 1144) to check whether the previous tape operation has been completed. If so, TWAIT returns immediately. If the operation has not been completed, TWAIT waits for completion with the interrupt off, then simulates a magtape interrupt to service the tape, after which return is made to DATA.

### More On Typing Out

There is a special problem associated with the typer which must be explained in order for the solution to seem relevant. Suppose EXPO is typing out a histogram, one line at a time as described above, when EXPO discovers upon reading a new event that it must type out an error message. Moreover, suppose the error is such that EXPO should continue acquiring data rather than waiting for completion of the histogram output line. In order not to lose the error message it is necessary to link the new character string (the error message) to the old string (the histogram line) in such a way that the error message will be typed automatically when the old line is finished. Note that in general these two strings are not stored contiguously in core.

The problem of linking successive strings together is handled by the subroutine TRIGR (p. 600). Single character typeout from a string under interrupt control is performed by TYPT (p. 1000); when TYPT has typed the last character of the string it checks the chain of links generated by TRIGR to see whether another string must be typed.

The calling sequence for TRIGR is as follows:

```
          JMS    TRIGR
LINKØ            Ø          to link character strings.
                 ADRESØ     address of first character
                            of string.
```

The first call to TRIGR finds ACTFLG reset (equal to zero), indicating that the typer is unused. TRIGR sets ACTFLG (to 1), saves the address of LINKØ in LO* (link zero address) and LL* (last link address), initializes the string pointer ADDR to ADRESØ, and transmits the first character of the string (which must not be a special character such as carriage return). Then the last link, in this case LINKØ, is set to 1 and TRIGR is finished.

After one-tenth second the teletype has typed the first character, the one transmitted by TRIGR. This completion causes a typer interrupt to TYPT which turns the interrupt on after merely saving registers and clearing the typer flag. Using the pointer ADDR, TYPT gets the next character in the string, transmits it, and returns. Each succeeding character is typed in this way by TYPT until $ is found, signalling the end of the string; this causes a jump to DONE (cell 1044). Here the contents of the original link LINKØ are examined (indirect through LO*). If there has been only one call to TRIGR, LINKØ still contains 1, which tells TYPT to reset ACTFLG and LINKØ (to zero) to signal completion, and typing is finished.

On the other hand, suppose during typing of the string stored at ADRESØ there comes another call to TRIGR:

```
          JMS       TRIGR
LINKL               0
                    ADRESL
```

In this case TRIGR finds ACTFLG set (typer active) and must therefore link this new string to the old one. Remember that LINKØ contained 1 while the ADRESØ string was being typed out, and LINKØ was reset to Ø on completion. TRIGR now changes LINKØ to contain not 1 but the address of LINKL. The last link address, LL*, is also set to the address of LINKL. Then when TYPT reaches DONE it finds in LINKØ (indirect through LO*) not 1, but the address of LINKL, the next link in the chain of calls to TRIGR. TYPT resets LINKØ to show that the first string is finished, puts the LINKL address in LO*, and initializes the string pointer ADDR by getting ADRESL from the cell LINKL + 1. Then the first character of the new string is transmitted. In this way an arbitrary number of calls to TRIGR will be linked together, and the character strings will be typed out in order.

If it is desired to type out a string completely before proceeding to other tasks, the following sequence should be used:

```
      IOF             Turn off interrupt to prevent
      JMS    TRIGR    simultaneous entries into TRIGR.
LINKØ        Ø
             ADRESØ
      ION             Interrupt on to await completion.
      TAD    LINKØ    Get link.
      SZA CLA         Skip if link = Ø; string has
                      been typed.
      JMP    *-2      Not done yet, go back and wait.
      NOP             All finished.
```

The characters in the strings are packed two to a word; these are 6-bit characters formed by subtracting octal 24Ø from ASCII codes. Octal 76 and 77 are special 6-bit characters: 76 induces TYPT to issue a carriage return and line feed, while a word 77xx tells TYPT to type xx number of spaces. A dollar sign $ (244-24Ø = Ø4 in 6-bit form) terminates a string. The first character of a string must not be a special character (76,77,Ø4) since no checking of its nature is performed before transmission; hence to type only a carriage return the string must consist of ØØ76 Ø4ØØ, which will type an extra space before giving the CR and LF.

For typing single characters there is a special routine TYPE (cell 504). TYPE takes the character and constructs an appropriate character string in TT (cell 543), then calls TRIGR. TYPE is used to echo text input from the keyboard. To avoid certain timing problems that can arise if keyboard text arrives too rapidly, TYPE does not echo if the previous character has not finished typing out: TYPE first checks whether its previous request to type string TT has been completed and, if not, simply throws away the character. Otherwise the chain of TRIGR calls would contain an infinite loop, since the string TT would be linked to itself. (It might be better to have TRIGR do this checking.)

## ON-LINE ANALYSIS

Having described above the input/output handling, we now turn out attention to the details of the on-line analysis part of EXPO. As mentioned briefly above, UNPAK (p. 5000) unpacks the compressed data for an event, IF (p. 5200) determines whether the data fit the criteria previously specified by the experimenter and, if the event is acceptable, HIST (p. 3200) increments the histograms specified by the experimenter. These histograms together with a few overall event counters (number of data interrupts, number of events analyzed on-line, etc.) may be examined at any time by the experimenter in order to monitor the performance of the experimental apparatus and, in case of malfunction, to isolate the trouble. The great power of EXPO in monitoring and trouble-shooting derives from the fact that the permissible IF criteria and HIST specifications are quite general, and may be created or changed rapidly and easily from the teletype keyboard. This is possible because the sample buffer is unpacked into separate variables whose locations are specified by a dictionary of four-character names contained in EXPO; keyboard reference to these mnemonics permits on-line compilation of the IF and HIST specifications.

The compressed data contained in the sample buffer may be of several types, which are unpacked by UNPAK in different ways. One 12-bit word of compressed data may represent twelve "logical" variables, each of which has the value 0 or 1. Another word may consist of two 6-bit "analog" variables resulting from analog-to-digital conversion. Both logical and analog variables are single-valued variables; to each variable corresponds one number specifying its value for the current event, 0 of 1 for the logicals and a range 0 to 63 for the analogs.

In high-energy physics experimentation there is another useful kind of variable, the multiple-valued variable; this too is handled by UNPAK. This kind of variable may or may not be useful in other fields. These variables arise naturally when describing the passage of several particles through an array of particle detectors (called a "hodoscope") designed to determine the spatial location of the particles. If we number the hodoscope detectors (1,2,3,...N) we can specify a location by giving the number of the detector struck by a particle. If there can be more than one particle, we must give a list of detector numbers; that is, the variable representing the hodoscope is multiple-valued. In the compressed data an array of twelve detectors would be represented by a 12-bit word whose individual bits specify whether the corresponding detector has been struck (1) or not (0). If this word were 010001000001 ( = 2101 octal), then UNPAK would produce a list of detector numbers (2,6,12) and a count of 3 particles, and this information would be stored in an area referenced by the name of the hodoscope. UNPAK will handle longer hodoscopes by crossing word boundaries; in the distributed version of EXPO there are two hodoscopes names MHA and RHA of 15 and 32 detectors respectively.

There are two dictionaries of variable mnemonics, one for single-valued variables and one for multiple-valued variables. The dictionary of single-valued variables is found in SNAMES (p. 6000). Suppose for simplicity there were only three such variables, with mnemonics FRST, SCND, and THD. Then the structure of SNAMES would be

```
6000   6000   SNAMES   PZE   *

6001   4662            ALF   FR

6002   6343            ALF   SC     first half of name

6003   0064            ALF    T

6004   6364            ALF   ST

6005   5644            ALF   ND     second half of name

6006   5044            ALF   HD

6007   0000   FRST     PZE          UNPAK stores into

6010   0000   SCND     PZE          these cells

6011   0000   THD      PZE
```

(The alphanumeric characters are 6-bit characters formed by subtracting 240 octal from the ASCII codes.) When the user types SCND in defining the IF event criteria or the HIST histogram specifications, the subroutine SEARCH (at 4461) looks for SC in the first part of SNAMES and when found checks that ND is in the corresponding second part of SNAMES. EXPO then knows that UNPAK will place the value of SCND in cell 6010; that is, the address of the variable SCND is 6010, and this address is used in the compilation.

If a mnemonic is not found in the SNAMES dictionary SEARCH looks in MNAMES, the dictionary of multiple-valued variables starting at cell 636.

The structure is similar to SNAMES:

```
636       0636   MNAMES   PZE        *

637       0055            ALF        M    first half of
                                          name
640       0062            ALF        R

641       5041            ALF       HA    second half of
                                          name
642       5041            ALF       HA

643       0000   MHA      PZE             UNPAK places here
                                          the number of
644       0000   RHA      PZE             particles which
                                          hit MHA and RHA

645       0647                  MHMULT    addresses of
                                          lists
646       0654                  RHMULT

647-653          MAMULT   BSS        5    up to five parti-
                                          cles allowed in
654-660          RHMULT   BSS        5    each hodoscope
```

When SEARCH finds MHA (say) in MNAMES it acquires not only the address 643 but also the address 647 where the list of traversed detectors will be built up by UNPAK. This supplementary address (the list location) is set to zero if the name is found in SNAMES; hence the histogramming routine may determine what kind of variable is in use by checking whether this supplementary address delivered by SEARCH is zero or non-zero. In EXPO this information is in fact referred to as KIND-the kind of variable.

## The IF Function: Event Selection

The occurrence of a data interrupt is usually determined by hard-wired circuitry outside the computer. It is often essential in trouble-shooting or monitoring to analyze on-line some special subset of the data (without of course disturbing the logging of all the data). This need is met by the IF routine (p. 5200), which checks each event against a compiled list of criteria in IFLIST (4102-4137). If all criteria are met, the event is analyzed; otherwise the event is ignored. Even if it were permissible to stop normal data-taking by modifying the hard-wired circuitry, it would be difficult to set up with hardware the complex criteria which may be easily compiled into IFLIST.

The IFLIST compilation is performed by IFIN (p. 4000). IFIN reads specifications from the teletype keyboard of three types: coincidence, veto, and analog. The coincidence requirement is that a named variable be non-zero, veto requirement is that a named variable be zero, and the analog requirement is that a named variable have a value which lies within a given range. These various types may be combined, thus producing a very complex event specification.

IFIN is entered by a keyboard interrupt generated by depressing the CTRL key and hitting the I (for IF). IFIN then responds by typing COIN (for coincidence) and waiting for a list of names terminated with a $. The addresses of these variables (i.e., the cells stored into by UNPAK which correspond to the names) are stored in IFLIST, and the number of these is specified by the coincidence counter CCNT (cell 65). Next IFIN requests veto requirements by typing VETO; the corresponding addresses are added to IFLIST and VCNT (cell 66) keeps count of how many there are. Finally IFIN types ANAL to request analog specifications of the form SCND 5 29 FRST 0 3, etc.; that is, the variable SCND must have an unpacked value between 5 and 29 inclusive, FRST must lie between 0 and 3 inclusive, etc. The number of analog requirements is specified by ACNT (cell 67), and adresses and ranges are stored in IFLIST. A fairly elaborate IF assignment is given as an example in the appendix on operating instructions.

Note that the IF specification consists of the logical AND of the various requirements: All of the coincidence variables must be non-zero and all of the veto variables must be zero and all of the analog variables must fall within the prescribed limits; otherwise the event will be ignored. Often it is useful to have an OR capability; that is, be able to analyze two or more types of events in a single data-taking run. EXPO does provide a limited OR capability: HIST constructs two-dimensional histograms (arrays), and it is often possible to arrange that the various columns of an array correspond to physically distinct types of events.

## The HIST Function: Histogramming

Once an event has passed the scrutiny of the IF routine, it may be analyzed. The only type of analysis incorporated in EXPO is that of making two-dimensional histograms of any variable against any other variable (including itself); this is done by HIST (p. 3200). One-dimensional histograms may be made by using a special dummy variable as one of the two variables in a two-dimensional histogram. Which variables are histogrammed is controlled by information in the histogram assignment list ALIST (261-376), which is compiled by the ASSIGN routine (p. 4200) in response to a CTRL - A keyboard interrupt (see appendix on operating instructions). The region of core above the resident program is a scratch area used for histogram storage; the size and number of histograms are limited by the size of the scratch area and by the length of ALIST, which can hold the defining information for six histograms.

The histogramming routine HIST uses ALIST to determine which variables are to be histogrammed, the array size, the relevant storage locations, and what kind of variables they are: the "KIND" entry in the list is zero for single-valued and non-zero for multiple-valued variables. To compile ALIST, ASSIGN initializes the BASE pointer to the beginning of the scratch area, then calls HSTINP twice to acquire from the keyboard row and column defining information for a two-dimensional histogram. For row and column, HSTINP reads a variable name, such as RHA, calls SEARCH to locate it in the dictionaries of single and multiple variables, and when found saves in ALIST the address of the variable and the KIND. Then HSTINP reads the minimum, maximum, and binning factor desired and adds these to ALIST. The binning factor is actually the number of AC right shifts made to the data when histogramming in order to lump 1, 2, 4, 8, or 16 adjacent bins together. Also added to ALIST is the row or column length, computed from

$$\text{length} = \frac{(\text{maximum-minimum})}{2^{(\text{binning factor})}} + 1$$

After twice calling HSTINP, ASSIGN multiples together the row and column lengths to determine the amount of scratch area required for this histogram. If this extends beyond the end of the scratch area, an overflow message is typed and ASSIGN waits for the histogram to be redefined; otherwise the BASE pointer is advanced to the next available cell in the scratch area, and the next histogram may be defined. ASSIGN finally terminates upon receipt of a $ from the keyboard or after hav ing six histograms defined; the number of histograms ($\leq$ 6) is contained in NHST (cell 76).

The sample assignment given in section 3) of the appendix on operating instructions is:

```
MHA 1 15 0 NMH 1 5 0

RHA 5 28 2  Z  0 0 0 $
```

(Additional description of this keyboard input is given in the appendix.) These definitions would generate in ALIST (cells 261 -) the entries shown below.

| | | | |
|---|---|---|---|
| 261/ | 7427 | | base address; where $H(MHA_{min},NMH_{min})$ will be stored. |
| | 643 | | address of MHA value after unpacking an event. |
| | 647 | | KIND; nonzero for multiple-valued; address of vector list. |
| | 1 | | minimum. |
| MHA | 17 | description | maximum (15 decimal). |
| | 0 | of 1st | binning; number of right shifts. |
| | 17 | histogram | row length. |
| | 6222 | | address of NMH value after unpacking an event. |
| | 0 | | KIND; 0 for single-valued. |
| | 1 | | minimum. |
| NMH | 5 | | maximum. |
| | 0 | | binning. |
| | 5 | | column length. |
| | 7542 | | base of second histogram; (7427 + 17*5) octal. |
| | 644 | | |
| | 654 | | non-zero for multiple-valued. |
| RHA | 5 | description | |
| | 34 | of 2nd | (28 decimal.) |
| | 2 | histogram | |
| | 6 | | $(28 - 5)/2^2 = 5$ truncated; |
| | 6224 | | $5 + 1 = 6$. |
| | 0 | | |
| Z | 0 | | 13 decimal cells required to define a histogram. |
| | 0 | | |
| | 0 | | |
| | 1 | | |

Note that after unpacking an event, cell 643 ("MHA") contains the number of entries in the list of struck detectors found in the list starting at cell 647. For ease in referring to the "struck" count as a single-valued variable, the contents of MHA (cell 643) are copied by UNPAK into NMH (cell 6222), which is in the SNAMES dictionary.

It should be mentioned that HIST doesn't store into a histogram if the minimum-maximum bounds are exceeded.

Overflow Handling

With the short PDP-8 word length it is important to be able to handle overflows encountered during histogramming; it is assumed that only a few cells will overflow. In order to save space there is an overflow "associative memory" in OLIST (2734 - 2777), created by OVFLOW (p. 2600) in response to a call from HIST. OVFLOW is entered with the AC containing the address of the cell that overflowed (i.e., reached a count of 4096 decimal). If this is the first time this cell has overflowed, this address is added to the first half of OLIST and the corresponding location in the second half of OLIST is set to one (to count one overflow). Also, NFLOW (cell 2730), the number of overflowed cells contained in OLIST, is incremented. If in the future the same cell should overflow again, OVFLOW will find its address already entered in the first half of OLIST and will increment the corresponding overflow counter in the second half of OLIST. (Naturally NFLOW is not incremented in this case). When typing out a histogram, for each element OLIST is scanned in order to type out the "double-precision" result. Since this is time-consuming, the overflow information is ignored in scope displays.

From the above description one sees that a count of 4096 would result in 1 in the high-order associative memory and 0 in the original histogram cell in the scratch area. Actually, because it made the double-precision binary-to-decimal conversion on output seem simpler, OVFLOW performs one further function. The histogram element which has just reached zero through overflow is set to 96 decimal; thus the full double-precision result is 4000x (high-order count) + (low-order count), rather than 4096x (high) + (low). (Note that after the first overflow, a histogram element will overflow every 4000 more counts, since it has a "head start" of 96 each time.)

If OLIST is full, overflow in a new cell cannot be treated normally. In this case the location of the overflowed cell is merely typed out.

Histogram Output

At any time, including during data-taking, the present contents of any histogram may be examined either by typing out on the teletype or by displaying on the scope. Type-out is initiated by a CTRL-P (P for print) keyboard interrupt, followed by a histogram specification similar to the format used to assign the histogram originally. Printouts of selected regions are permitted, and row and column totals are given; details are given in the appendix on operating instructions. Scope display is initiated by a CTRL-S (S for scope) keyboard interrupt, followed by specifications similar to those for print-outs. The display is in the form of a labeled one-dimensional histogram with scale determined by the switch register setting; one may display any portion of one row or one column of the two-dimensional histogram in core. (Again, see details in the appendix on operating instructions, including use of a point plotter.) If data is coming in one sees the counts rise dynamically on the scope, which is fun.

Since the interrupt is always on during the scope display and is off for only 40 microseconds per typed character (or 400 microseconds per second at 10 characters per second, which is 0.04% of the time), histogram output creates negligible dead time. There may be some small loss in the fraction of incoming data which is analyzed on-line, which usually is unimportant.

The subroutine MXA (MX for matrix or histogram; p. 5400) is called following a request for either printing or scope display. MXA in turn twice calls HSTINP (the same routine is used by ASSIGN) to read, from the keyboard, information defining the desired histogram and region for output. Then ALIST is inspected to find out where this histogram lies in core. Once the histogram is located, TMXOUT (p. 5600) controls printout or POINTS (at 2300) controls scope display of the histogram contents. Labels or titles are set up by MXTITL (p. 2000) for printing and SCTITL (at 6251) for scope display.

When printing, overflows are taken into account by MXWORD (p. 3400): Before printing a histogram cell, the overflow list OLIST is scanned to see whether this cell overflowed. If so, the double precision result is used. In either case, leading zeroes are suppressed by LDZERO (at 1466).

## MISCELLANY

There are two keyboard control functions related to magtape handling which are not discussed in the appendix on operating instructions: RUN (at 6706; initiated by CTRL-R) and ZEND (at 2157, initiated by CTRL-Z). RUN begins a new data run: First it asks for a run number from the keyboard, then searches the present data tape for a double end-of-file (EOF) and backs over the second EOF, ready to write a title as the first record of this new file. Next the routine RTYSTR (p. 1600) reads a title of up to 70 characters (terminated by $) from the keyboard into the tape buffer, which is then written onto the magtape. The first three PDP words of this title record contain the run number, a 1 to indicate this is a title record (the data records to follow have a 0 here), and nonsense (data records have here 240x, meaning 24 octal words per event and x events in this physical record, which may be 0, 1, 2, or 3); the physical record is 75 decimal words long. RUN also initializes pointers and counters for storing data into the tape buffers and for controlling switching between the two buffers.

ZEND terminates a run: The incompletely filled tape buffer presently in use is written out, then two EOF's are written and the second is backed over. The tape is now in the proper position for RUN to find a double EOF for the next run, since RUN actually looks for an <u>empty</u> file to indicate a double EOF.

CONV10 (4600) converts decimal keyboard input to octal. The sequence to read a four-digit number, convert, and store in A is:

```
    JMS  GETNAM
    JMP  DONE          (abnormal; $ found)
    JMS  CONVIO        (normal return)
    DCA  A
```

If the four-digit input is greater than 4095 decimal, A is set to 7777 octal. The four-digit input number is terminated by a space or CR; if more than four digits are typed before a termination the last four are taken, which provides a simple way of correcting typing errors.

The subroutine BCD (4655) converts an octal number in the accumulator to four BCD characters packed two per word in BCHI and BCDLO (cells 147 and 150). BCD is a DEC subroutine.

COUNT (3750) and DCOUNT (1354) are identical subroutines used for making double-precision tallies of the number of data interrupts, the number of events which passed the If test, etc. COUNT is used by the analysis task. Since a data interrupt might catch DCOUNT in use, DCOUNT would have to be reentrant to be used by DATA; hence the two tasks have two identical (non-reentrant) subroutines.

READAT (661) reads ten 12-bit words from a pulse height analyzer (PHA), each of which consists of two 6-bit pulse heights, and an additional ten 12-bit words from an external buffer storage unit (referred to as BS1 in the program). Obviously READAT must be tailored to specific needs and hardware configurations, just as the dictionaries SNAMES and MNAMES must be.

In some cases more and/or larger histograms would be useful. If magtape is not used (i.e., the printed histogram information is all that is needed), the magtape handling routines and buffers above 6600 are unnecessary and the scratch area may be extended down: change BEGSCR (cell 4274) from its present value of 7427 to 6600 and change cell 3607 from SNA CLA (7650) to CLA (7200) to prevent DATA from calling the tape routines. As long as the magtape isn't in use the routine RTYSTR (which needs a title from the keyboard to write on tape) isn't needed either. Hence ALIST can use the space from 1600 to 1754 (or 1777 if the characters strings used by IFIN are moved elsewhere, say to the old ALIST area), which will permit 8 rather than 6 histograms to be defined: change ALIST* (cell 71) from 261 to 1600 and change NHSMX (cell 4303) from 6 to 8.

If magtape is used it is much more difficult to get more room; reassembly will certainly be required. Some suggestions are probably in order. There are certainly routines which could be removed at a rather small sacrifice. In particular, the scope-labeling routines (but not POINTS) including the character patterns while useful are perhaps the least essential pieces of EXPO. Next might come RTYSTR; the tape will lack titles but not data. ASSIGN and IFIN could reside in the scratch area, to be overwritten once they have compiled ALIST and IFLIST; all that is lost is the ability to change these compilations without reloading the program. As described in the appendix on operating instructions, the routines which handle scope and printer histogram output permit the user to type the output specification in an order reversed from the original ASSIGN order (and hence reversed from the order in ALIST). This permissiveness leads to lengthy coding in several parts of EXPO: loss of this facility would only affect typer output format, as described in the appendix.

## APPENDIX: OPERATING INSTRUCTIONS FOR THE PDP-8 PROGRAM "EXPO"

0. Turn off data-acquisition interface to prevent data interrupts.
1. Load program and start at 1200.
2. I for IF. Type CTRL-I (i.e., hold down the CTRL key and type I).

   Computer will respond with COIN for "coincidence", inviting you to type the names of those variables you wish to be non-zero in the events to be histogrammed. You type for example (_ means space, CR means carriage return)

   C1_V1_NMH_SH7_$

   This means counters $C_1$ and $V_1$ must fire, there must be at least one particle in the multiples hodoscope, and shower counter 7 must have a non-zero pulse height. The dollar sign terminates the string. Note that each variable name, <u>including the last</u>, must end with a space or CR.

   The computer will next type VETO. You type the names of those variables which you want to be zero in order for the event to be histogrammed.

For example,

```
V2_GCS(CR)
RHG2_$
```

This means counter $V_2$ didn't fire, the south gas Cerenkov counter discriminator didn't fire, and the RHG2 (RH > 2) discriminator didn't fire.

Finally the computer types ANAL, asking for "analog" requirement. You define minimum and maximum values of variables; e.g.,

```
NRH_3_5_RHA_17_29(CR)
SH6_13_19_$
```

For an event to be histogrammed, you have required that there be 3 to 5 particles in the rear hodoscope, all of these within the range of counters 17 through 29; in addition shower counter 6 should have a pulse height between 13 and 19 inclusive.

The computer now responds with ENUF to indicate completion of the IF definition. If you want to change this definition, retype CTRL-I.

3. A for ASSIGN. Now assign the histograms to be constructed. Type CTRL-A. Then type, for example,

```
MHA_1_15_0_NMH_1_5_0(CR)
RHA_5_26_2_Z_0_0_0_$
```

For those events satisfying the criteria of the IF definition, two histograms will be made. The first is of the multiples hodoscope (lumped in bins of $2^0 = 1$ counter from counter 1 to 15) versus the number of particles striking the multiples hodoscope. The second histogram plots the rear hodoscope from counter 5 to 26 lumped in bins of $2^2 = 4$ counters (i.e., 5-8, 9-12, 13-16, 17-20, 21-24, 25-26), versus Z, a special variable which always has value zero to facilitate making histograms which are effectively one-dimensional. Note that if any particles in the rear hodoscope lie outside the 5-26 range, none of the counters will be plotted. This is just part of the multiple-particle problem. After you type the dollar sign, the computer types ENUF to show the assignment is finished. It will also type ENUF after the definition of the sixth histogram; 6 histograms is maximum. If you type a name wrong (i.e., RAH instead of RHA) the computer will respond with a ?; then you may retype RHA. If you define a histogram which would overflow the scratch area set aside for histograms, the computer types OVFL. You may then retype the last definition. If you want to change everything, just retype CTRL-A and start over.

4. C for CLEAR. To clear all histograms and event counters, type CTRL-C.

5. Turn on the data-acquisition interface to permit data interrupts. Events will now be processed.

6. H for HALT. If you wish to halt after N events have been histogrammed (passed the IF test), type CTRL-H followed by N, a decimal number less than 4096. This may be done at any time, even while running. If you do it after N events have already been histogrammed, the computer will halt after N + 4000 events. The teletype bell will ring to signal the halt. Stop the computer, turn off the data-acquisition interface and restart in 1200.

7. Output may be made even while collecting data, although on the teletype the last line will have events collected over a longer time than for those of the first line. This is due to the fact that only a one-line buffer is used for teletype output.

8. P for PRINT. To print out a histogram, type CTRL-P, followed by MHA_3_12_X_NMH_2_3_Y_, for example. Here X may be anything, and Y is zero if no titling is desired. If Y is non-zero you will get

```
NMH MHA 3  4  5  6  7  8  9  10

 2      _  _  _  _  _  _  _  __ row sum (3-10)

 3      _  _  _  _  _  _  _  __ row sum
        column sums (2-3)

NMH MHA 11  12
 2      __  __  row sum (11-12)

 3      __  __  row sum
        column sums       Grand total
         (2-3)
```

If Y is zero only the lines starting with NMH MHA are omitted. If you type

RHA_10_12_0_Z_0_0_1_ you get

```
 Z    RHA   10   11   12

   0        __   __   __  sum

                sums      totals
```

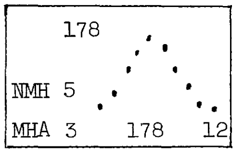
If you type in the other order

Z_0_0_0_RHA_10_12_1_ you get

```
     RHA    Z   0

         10     _
         11     _   sums
         12     _
                sum  total
```

If you make a mistake, type $ and then CTRL-P to start over.

9. S for SCOPE. To display on the scope, type CTRL-S, followed by MHA_3_12_0_NMH_5_5_1_ (same format as for CTRL-P). This will give a scope display like



Here the "178" is the decimal conversion of the switch register setting, which controls the vertical scale.

You can in this way display any slice of a histogram. If no title is requested you get only the points. Once you have a display you like, you may turn on the plotter with the "STANDBY" switch in the middle position and type CTRL-G, whereupon whatever is on the scope will go on the graph plotter.

10. E for EVENTS. Type CTRL-E to get event counts:

```
TAPE
MOVE  Tape + nontape = number of interrupts
                    acknowledged.
NMOV
NTAP
PRYN  Sample + sample moved = number of
NPYN  events tested by IF statement.
SMPL
HST   Histogram = number of events which
SMMV  passed the IF statement.
```

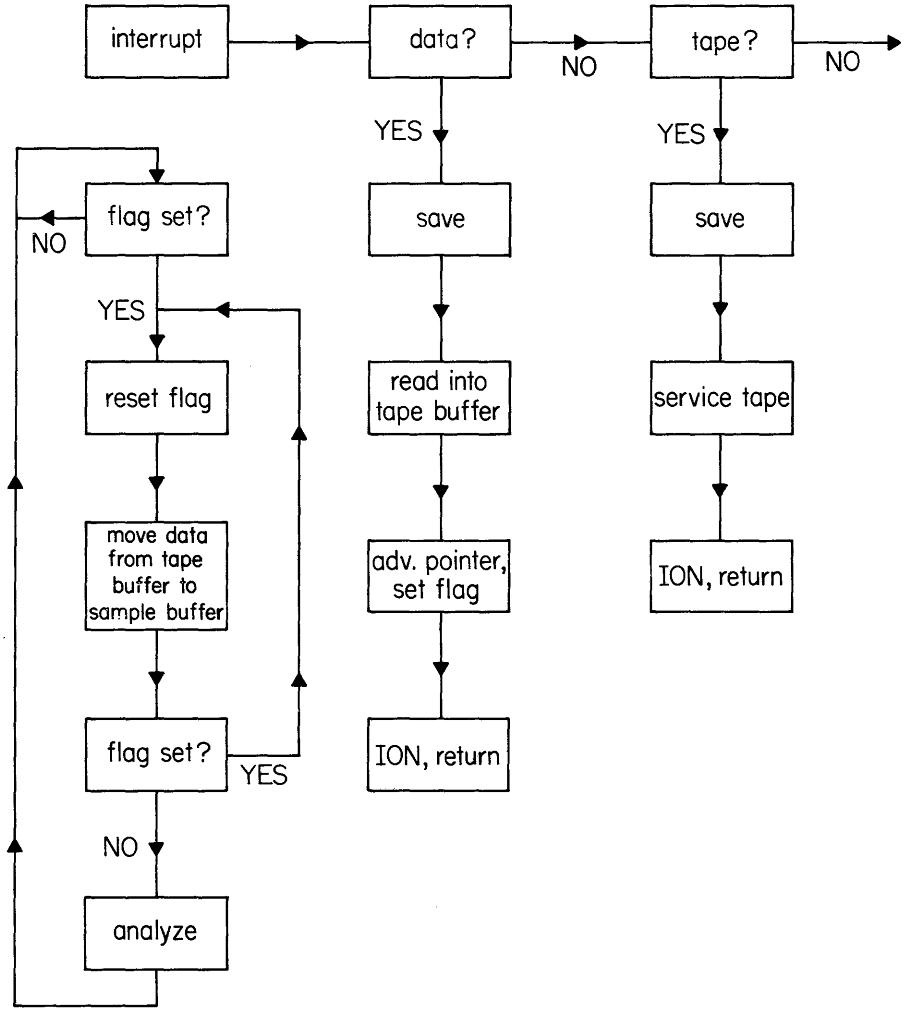


Figure 1 Simplified flow diagram of the basic interrupt handling and on-line analysis.

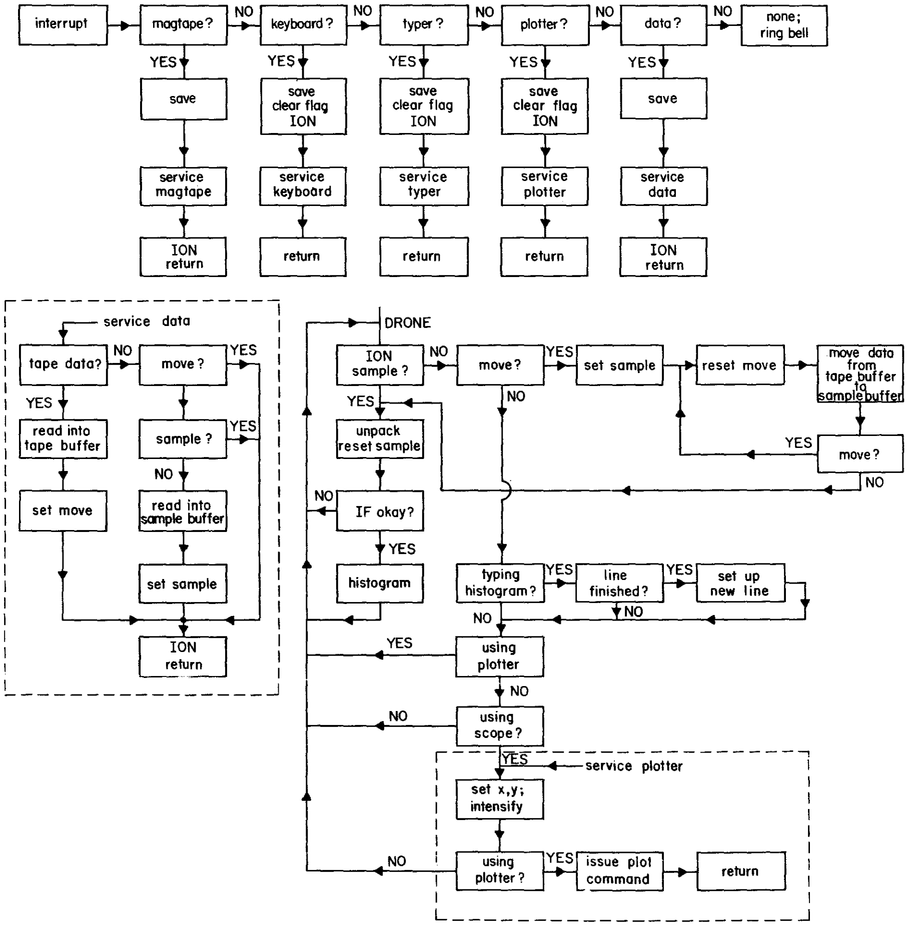


Figure 2 Detailed flow diagram of input/output interrupt handling. "RETURN" includes restoring the saved accumulator and link, followed by an indirect jump to the saved return address.

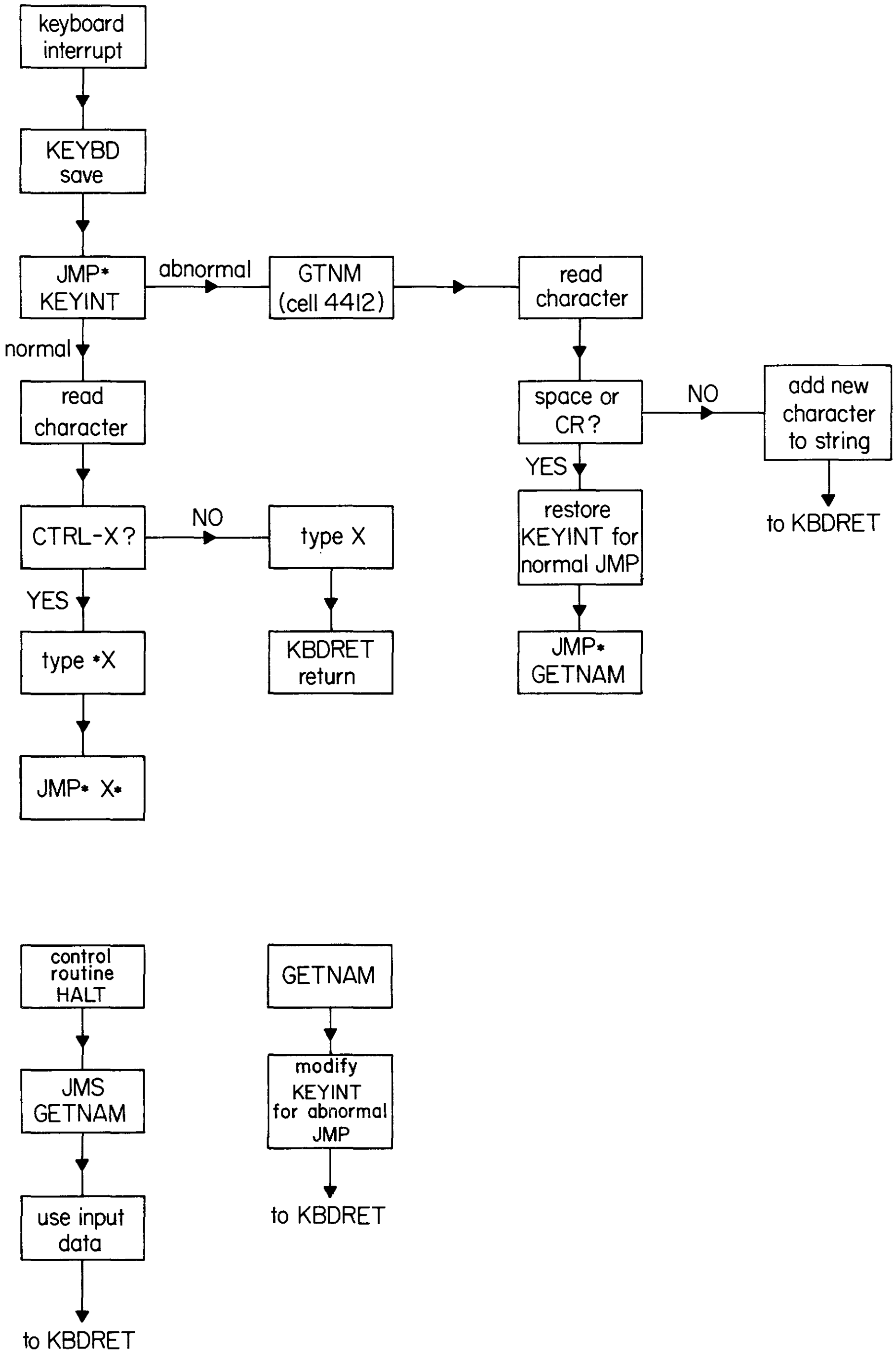


Figure 3  Flow diagram for keyboard control functions.

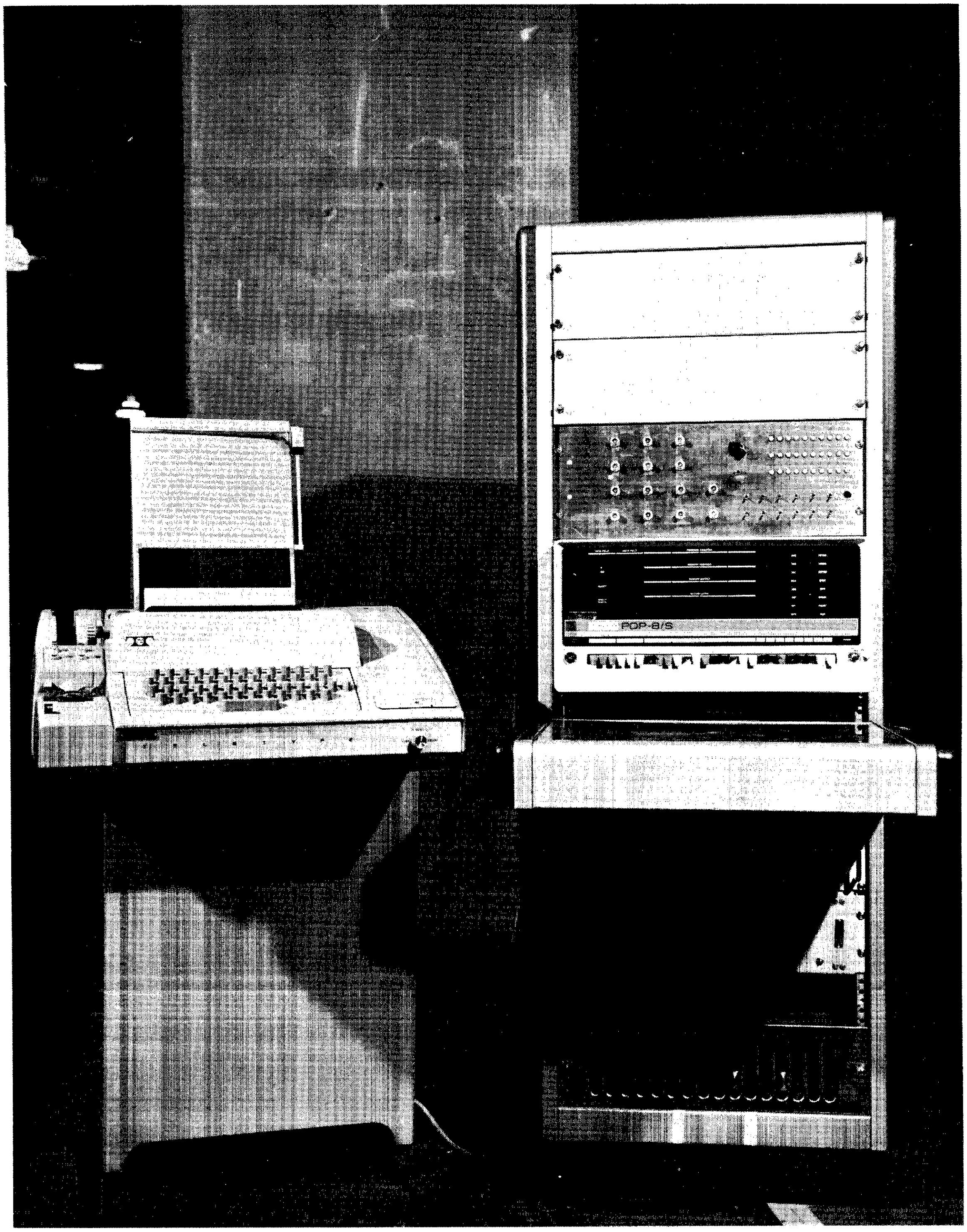


Figure 2. The completed interface and computer installed in an instrument rack.

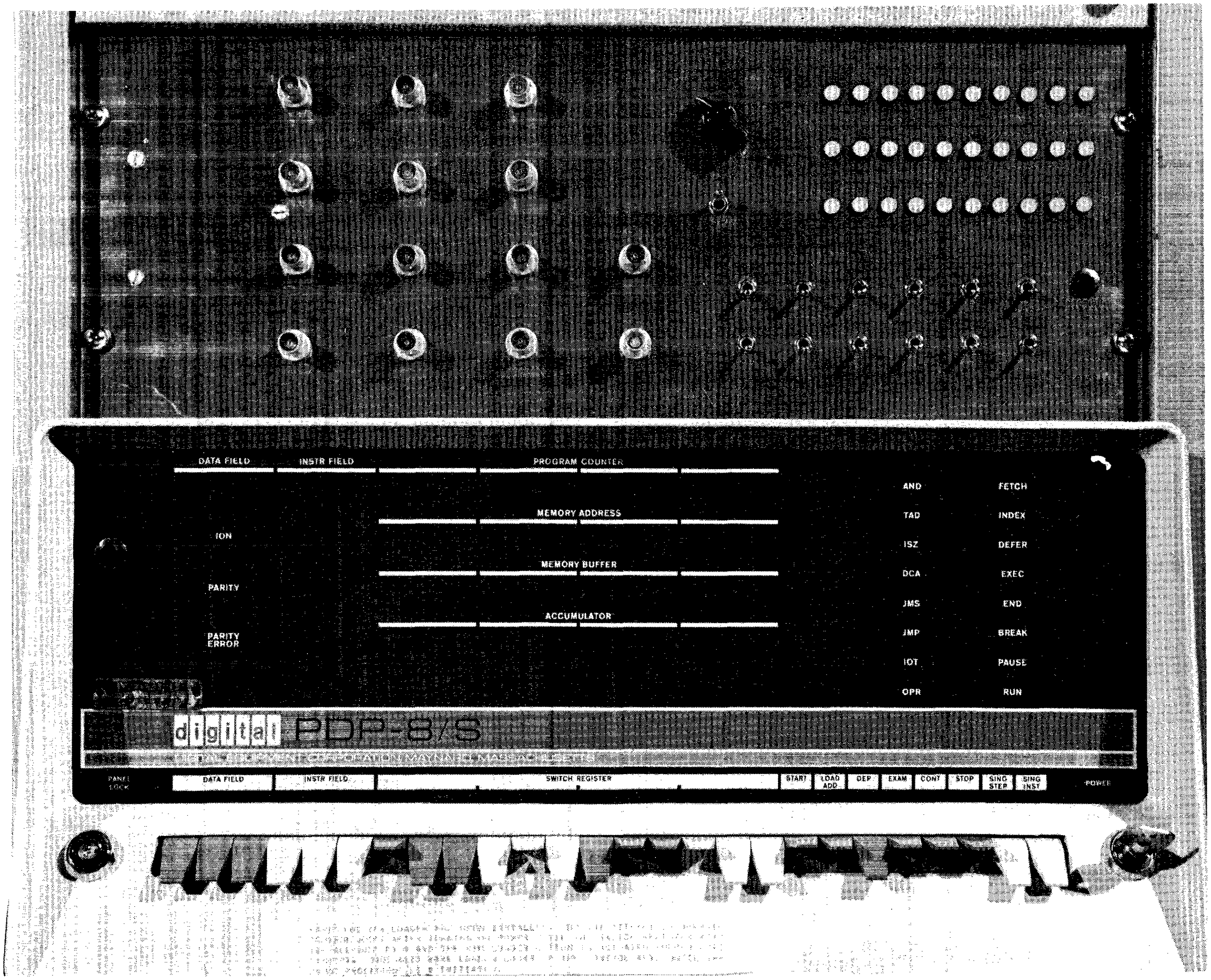


Figure 7. A closeup of the front panel showing the function switches and the indicator light panel.

# SCHLEP

Michael Greenblatt

University of Pennsylvania

Philadelphia, Pennsylvania

## ABSTRACT

SCHLEP is a program which was written to work in conjunction with magnetostrictive read-out spark chambers. Its main purpose is to help maintain these spark chambers and other devices which are the apparatus used in this High Energy Physics experiment. The experiment that it will participate in, is the study $K^o$ long lived meson decay into $3\pi$ mesons. It also has the ability to do some on-line analysis of some of the data.

As a start let me first describe the hardware of which the program makes use. The basic machine is a PDP-9 with Extended Arithmetic Element, 2 Dec-tapes units and a control and a 34-H Display. The rest of the non-standard I/O equipment has been built at the University of Pennsylvania. This includes an IBM compatible tape unit interface, and an interface that transfers the spark chamber coordinates into the computer. Also, we have an alarm which can be turned on and off. I would like to describe the tape units that we have. One is a Kennedy Incremental that is "souped" up to run at 1kc. We plan to use this in the back up mode. The tape unit that we plan to use as the main unit is manufactured by Peripherial Equipment Corp., Model No. 4820-7 and it reads or writes at a maximum of 20kc. I would, also, like to describe in brief some of the new instructions that were added to the interfaces. The first and probably most unique is a program interrupt at word count overflow from the tape units. This gives the program the opportunity to block the records on tape without the use of a large buffer. This is important when you only have 8192 words of memory This leaves room for more programs by keeping the output buffer small. Of course the efficient use of an instruction like this depends on a high and constant input rate. Another instruction that was added is a "skip on program interrupt active" this allows the program to handle waiting interrupts without returning to the interrupted program first.

The main purpose of "Schlep" is to help maintain approximately 20 spark chambers and 40 scintillation counters that are the basic components of this experiment. One of the ways that it does this, for spark chambers, is to compute straight lines from the spark coordinates and from these points calculate histograms of the spark position in each chamber. These histograms will have a resolution of about 1 inch per bin. By looking at this set of histograms the physicist can see if the chamber or the read-out device, the wand, is working properly. The way this information is presented is by plots on the CRT The physicist can tell if these devices are working properly by looking for a loss of symmetry in the histogram, i.e., a hole in the middle of it. Another plot of the spark chambers is a histogram of the residuals. A residual is the difference between a computed spark position and a real spark position. This histogram would show if there is a translation of the spark coordinates in the spark chambers. It can also be used as a measure of the resolution of this wand or spark chamber.

We also have a histogram of the number of times a particular scintillation counter fires. This would be a quick check of whether a counter is functioning properly. The program also will monitor voltages from several power supplies used in the experiment. The maintenance value here is obvious. It is, of course, when one of these voltages change beyond limits the physicist can be notified immediately.

Besides its maintenance value the program could also be used as a filter of the data. There is a natural phobia involved here, that is, of course, the fear of throwing away data that may initially look bad, but may be useful in some sense. Excluding this, the program has the capability of making some simple cuts on the data. The first of these would be incorrect "Time of flight" for the k meson. Others make use of the line finding portion of the program. These would be a restriction on the number of lines in the front chambers. In our case we would require at least two. We could also ask if these two lines have an intersection point within a given area. The aperture of the magnet can also be used as a cut by requiring that the lines go through, not hit the sides. There is probably some other requirements such as trigger which we could use, but since we have no plans at present to cut the data I will leave this up in the air.

The initial objective of buying the PDP-9 and then investing time in writing programs is to increase through-put of data.Our input record size is 172, 12 bit words. From this I expect to be able to "prune" about 30%, this 30% would be data which is outside the active region of the chamber. Using the above numbers , our spark chamber interface time would be 1.72 milliseconds. This record is then to be written on IBM compatible tape. The time required for this would be 18 milliseconds. Somewhere in between the time the record comes into the computer and the time it goes out to the tape unit, we have the program. The program as I pointed out before reduces the data and moves it into another buffer, this operation will take about 30 milliseconds. We are, therefore, through-put limited to about 30 events per second. It is most probable that this present experiment will run at 10 events a second or less.

The program itself is divided into four distinct parts those being 1) Supervisor and I/0 devise handlers 2) Display routines 3) Throughput 4) Compute. The supervisor, and in my case I use the name loosely, is basically a revised DDT which makes use of Program Interrupt and the Extended Arithemetic Element. One of the things that were changed in the basic DDT, is the output to the paper type punch. These were punched in a punched in "Funny Format." What is done now, is the operator has the choice of punching in "RIM" mode or in "Hardware Read-in" mode. These I found to be much more useful. The reason being that one is able to write any type of program, which may or may not pertain to Schlep, and get an easily useable paper tape. Another useful option in Schlep is the Dectape. What I have done is divide each Dectape into 10 files. The operator can use any of these files to write out core or just a portion of it. The routines are also available to Schlep itself. At present we have no plans for online storage of data, but the program is set up so that by setting a few locations and calling a subroutine one can easily write data on different files or just successive blocks. Probably one of the most useful aspects of having a real time DDT is the ability to debug background programs while the experiment is running, that is, taking data. This would be useful when I or someone else would like the program to compute a new variable or even add a new display routine. This may sound like it could be very dangerous because of having an undebugged program in core with Schlep running. For those of you who think so, I agree. I have taken some precautions to help prevent anything too terrible from happening. The program can be set in a mode so that no data entered from the teletype could be entered into the region of Schlep which is finished and debugged. In other words, one is not able to change core locations where the program resides. But, one can only type information into a data section of core. One always has the ability to examine any location in memory via the teletype, but unless he knows a keyword, he would not be able to change any locations except those in a data region. For instance an example of the data that he would be able to change, is number of bins for a particular type of histogram, or even the range of this histogram, and other such parameters. Another useful feature of the supervisor is the ability to dump an event on the teletype in decimal, at the same time plot this event on the CRT. This is very useful at the beginning of an experiment for one would like to see what were the actual coordinates of the sparks in the chambers Any time during the running of the experiment one can shut down the data taking by hitting a key on the keyboard. One can also terminate a run by use of a macro type instruction entered via the teletype. In summary the supervisor provides I/0 device handlers, a macro facility, a simple minded memory protection, and mnemonics for core locations and instructions. I will conclude the section on the supervisor with a short description of the macros available to the operator through the console teletype.

1) PUNCH$ - punch a paper tape in RIM loadable format by first setting locations. PBGN$ - first location to be punched and PLST - last location to be punched.

2) DKTAP$ - write or read from or into core locations from unit 1. One must first set DBGN$ - first location DLST - last location RWBIT - 0 - read into core, 1 write from core onto defined file. FILE - the file on which the data resides.

3) RIM$ - read in a RIM loadable tape.

4) DUMP$ - dump on the teletype in decimal, core locations defined in TBGN$ - first location to be dumped TLST - last location to be dumped.

5) STOP$ - this is the instruction which causes Schlep to go to its end of fun routine.

6) WORD$, NOT$, SYMBO$, ADDRE$ - these are the standard DDT macros which are explained in the DDT manual.

7) SCON$ - this sets the program in a mode such that it types out the event on the teletype and displays it on the CRTbit. 16 controls the acceptance of the next event.

8) SCOF$ - turns off the last option

9) D$ - get the directory from dectape and print it on the teletype.

10) OPEN$ - this allows the operator to examine and change any location in memory.

11) CLOSE$ - this macro prevents the operator from changing any of the "protected" region of core.

The next section of the program is the display. This is the portion of the program which displays, histograms, pictures of the event, and counter voltages. The display section is divided into many subroutines which perform small functions. The reason that it was done this way, is because this modular approach helps make many subroutines available to other parts of the program. It also would make it easier to add new displays to the program. To choose the display that one wants to see is just a matter of "flicking" one of the AC-switches. A macro which pertains to the display position of the program is called DMES$. This allows the operator to enter a message into a buffer which is displayed in the top portion of each display. This can be used to leave messages to other operators or can be used to label a certain display. The hardware that is used with the display programs is a 34-H display unit, the small Tektronix scope with attached camera, and a Hewlett-Packard 1300A. Both scopes have the same display. The display programs are of the lowest priority. This means that we do not display anything until we have finished with the higher priority items, these being, throughput which is the highest priority, and the compute section, which is the next highest in priority. A great deal of effort was put into the display programs to make the display easy for the physicist to interpret. For example, in the histogram displays a base line with half bin. This helps in the statistical analysis of the histogram. In the spark chamber display routines the chambers themselves are marked off, so if he wishes he could calculate roughly the kinematics of the event. Also, in the display of the spark chambers each scintillation counter is defined by its edges, and when that particular counter fires it is filled in with dots. One would like to be able to hold a histogram or picture of event on the scope without having it change each time a new event comes. He has this ability by using another AC-switch. It is, also, conceivable that the physicist would like to look at one of the histograms on a log scale. He again has this ability through another AC-switch. Another thing that may be useful is the ability to see the histogram doubled in the x-scale or the y-scale, again by the use of the AC-switches he can do this. I might point out that the histograms of the spark position in each chamber are displayed with the bottom half of the screen display, the x-coordinate and the top half is committed to displaying y-coordinate. In concluding the description of the display section of the program I will list the assignment of the AC-switches.

| | |
|---|---|
| AC0 | display 2 histograms of the wands |
| AC1 | display time of flight of the k-meson and $\gamma$ ray |
| AC2 | display 2 histograms of chamber residuals |
| AC3 | display the x-view of the event |
| AC4 | display the y-view with the rear left chamber |
| AC5 | display the histogram of counter firings |
| AC6 | display the rear right y chambers with switch AC4 |
| AC7 | display the voltages |
| AC8 | freeze the buffer |
| AC9 | display the previous on log scale |
| AC10 | double the x-scale for histograms |
| AC11 | double the y-scale for histograms |
| AC16 | get next event, in association with SCONS |
| AC17 | turn off the audible alarm |

If any part of Schlep could be considered its heart, the throughput section would be it. The throughput section is naturally divided into two parts, those being input and output. The input section is called the scanner section after the device which transfers the data into the computer. Both the input and output section of the program makes use of a word in memory called FLGWRD. The bits in this word are used to indicate the condition of the I/0 devices, buffers and the throughput programs. These octal equivalents are as follows:

| | |
|---|---|
| 1 | INITIAL INPUT BUFFER PULL<br>SCANNER BUFFER (SB) |
| 2 | STORED SCANNER BUFFER NOT EMPTY (SSB) |
| 4 | WANT RAW DATA |
| 10 | DATA WAIT IN SB |
| 20 | HALT AFTER END OF RECORD |
| 40 | P.T. READER BUSY |
| 100 | COMPUTE BUSY |
| 200 | PAST END OF TAPE |
| 400 | P.T. PUNCH BUSY |
| 1000 | GET AN EVENT |
| 2000 | DECTAPE BUSY |
| 4000 | TELETYPEWRITER BUSY |
| 100000 | NOT ASSIGNED |
| 200000 | REGISTER 10-12 SAVED |
| 400000 | NOT ASSIGNED |

The throughput, is naturally broken into two parts. The first part is the SCANNER, and the next part is the MAG-TAPE. We will first describe the flow of the scanner or read in portion. The first signal that we get is an interrupt called ER (EVENT READY). This causes a program interrupt, and we are transferred to a routine called SCAN. In this routine we first find out if the SCANNER BUFFER (SB) is empty. If it is, we initiate the transfer of the data into the SB. If the buffers are all full we set a flag and return to try it again, at some later time. Assuming that we find an empty SB we then transfer the data into the SB via the data channel. At this time we get an interrupt when either a word count overflow or an EVENT COMPLETE (EC) occurs, which causes a transfer to a routine called SCANI. The first thing we do here is see if the SCANNERS have underflowed or not transferred as much information as we expected. If this does happen we type out a message on the teletype which reads "SCANNER UNDERFLOW". We then set a flag in FLGWRD which tells the rest of the program that the SB is full. After this we find out whether we are going to use raw data or condensed data. If we are going to condense the data we do this in the SB. The last thing that we do is move the SB if possible to the STORED SCANNER BUFFER (SSB). If it was impossible to move the data we set a flag so that we may try again later. Once there is some data in the SSB we then try to write this data out on magnetic tape. The first thing we do is find out in which buffer this data exists, and then set up this address and word count in the appropriate registers. At this point we initiate a transfer to the tape unit. The decision of whether to end the record with an EOR or to continue is made at the time of a word count overflow. At the beginning of each record there is the internal event number, the word count and the starting address for this record. The records will be recorded on tape with unknown logical record size. The blocking factor is also unknown. this seems to be a ridiculous way to do things, but the experiment is going to be analyzed on a 3rd generation computer and the facilities to handle these type of records exist. The 2nd generation computers can also be made to do this, too, by use of tricky traps.

The compute section of the program is divided into two parts. These being, compute all things that are unrelated to line finding, and compute all things that make use of line finding. In the order of time, the items unrelated to line find are computed first. These would include histograming of counter voltages, histograming of counters that fired, and histograming of the time of flight of the k meson and associated gamma-ray. The rest of the compute section is devoted to line finding and its brand of histograms. The line finding program is fairly simple minded in that it only tries to fit a straight line through several points. Since there is three groups of spark chambers and we have 2 views for each group, then we can break them up into six units. These units or views are worked on by the program one at a time. Schlep then analyzes each view one at a time and finds as many lines as it can. These lines are then used to compute the spark position in each chamber. If this spark position falls within a given range the coordinate is accepted. We then continue this process for all the views. As each line is found and accepted, the coordinates of these lines are then used to compute histograms of spark position. These histograms would also give us the relative efficiency of each non-choice chamber. These relative efficiencies are as important as the histogram itself, for they will give us a feeling of how well the chamber is working. We also use the difference between the real spark position and the computed spark position to plot another histogram. This is, as I have mentioned before, called the residual. When the track finding is done for a certain event the program transfers to the general dispatching routine. I will describe this in the next section, but before I go on let me first show the method that is now used in track finding. I say "now used" because we are in the process of changing to a faster technique.

1) get the coordinates of the choice chambers
2) with these calculate the slope and intercept

$$m_\gamma = \frac{\Delta Z}{\Delta C} = \frac{Z_2-Z_1}{X_2-X_1}$$

$$b_\gamma = Z_o - m_\gamma X_o$$

3) calculate the points in the other chambers

$$X_i = \frac{Z_i - b_\gamma}{m_\gamma}$$

4) if there are enough points to make a line, then histogram the points and the residuals.

In describing all of the previous sections of the program we talked about many conditions which were left hanging in the air. These conditions will be taken care of when the program transfers control to the dispatcher. The dispatcher is a small routine whose main purpose in life is to help keep all other programs happy. The way it does this for the scanner routine is to see if there is any data waiting to be condensed or to be moved. If there is it transfers to the appropriate routine. It does the same type of thing for the magnetic tape. It asks if there is some data waiting to be written, and if there is it transfers control to the magnetic tape programs. The next thing on its list is the compute programs. It again sees if there is data waiting to be computed if there is then it calls the compute program. If there is no data to be computed it then starts to display. So we can see that this program somewhat defines the priority of the system.

In writing this program for the PDP-9 computer I have noticed a few things which I think we might have done a little differently. For example, I think that when you are running an online experiment with many I/O devices that it would be very advantageous to have the Automatic Priority Interrupt. This would help speed many of the programs up. It would also allow for a much neater and more powerful system. Another item that I think would be very useful is a drum. A drum that could be used for program storage, and more important display storage. The program storage advantage is obvious it would allow for a simple time sharing system on-line to an experiment. The display ability of a drum would allow much more time and core to be committed to on-line analysis of data, and in many experiments this could be done in real time.

So, in conclusion I think that Schlep is a powerful program for doing a magneto-strictive spark chamber experiment but it is only a beginning. There are many things that could be added in the hardware and the software to make the program even more powerful.

# A PROGRAMMED CONTROL AND INSTRUMENTATION SYSTEM FOR A NUCLEAR REACTOR

J. R. Kosorok
Battelle Memorial Institute
Pacific Northwest Laboratory
Richland, Washington

## ABSTRACT

A DEC PDP-7 has been interfaced with the instrumentation and control system for the High Temperature Lattice Test Reactor. The test reactor's electric heating system can heat it to 1000°C, so its graphite moderator is blanketed with pressurized nitrogen to inhibit oxidation. The digital computer directly controls the nitrogen and heating systems, and provides operational aids for the nuclear system. The central processor has 8K words of core storage and utilizes 3 DECtapes for bulk storage. In addition to the interface for over 100 analog and 190 digital inputs, two unique features are a three color, alpha-numeric display and two six decade analog-to-digital converters for measuring flux. The software system is composed of a highly integrated set of routines to monitor computer operation and provide operations assistance for the test facility.

## INTRODUCTION

A small digital computer (DEC PDP-7) has been interfaced with the instrumentation and control system of the High Temperature Lattice Test Reactor (HTLTR)*. The computer, the interface and the instrumentation and control system make up the Programmed Measurement and Control System (PMACS), which controls the heating and cooling loops, aids nuclear reactor operations, and provides data acquisition and logging.

The reactor facility is used to take nuclear data at temperatures to 1000°C, which will provide support for the design of high temperature power reactors. The reactor itself consists of a ten foot cube of graphite with a removeable central section--all enclosed within an insulated, gastight container. It is electrically heated with graphite heater bars supplied with 384 Kw. Temperature measurements are taken from thermocouples and moveable high precision resistance temperature devices (RTD's). The reactor blanket is a recirculating nitrogen atmosphere at low pressure. Controlling and monitoring devices associated with the gas system are gas blowers, digital stepping valves, binary (open-closed) valves, flow transducer elements, gas chromatographs and moisture monitors, and low precision RTD's. The flux level is measured by digitized signals from enriched BF3 ion chambers and is controlled with horizontal control rods. There are four vertical safety rods.

The complete programmed system is composed of a highly integrated set of executive control and utility routines and system application programs, which are coded in machine language. Because of the close interdependency between the computer, the interface and the hardware of the reactor system itself, Fortran is not used for on-line programs. The programming and computer memory requirements sometimes were responsible for actual hardware and interface modifications, and the converse was true. A general description of the system programs is almost a description of the instrumentation requirements for the HTLTR. Briefly, the requirements are: (1) monitoring nuclear flux and automatically calculating reactor period and power; (2) accurately measuring the pressure of the reactor atmosphere and the temperature of the moderating graphite; (3) heating the reactor and maintaining a desired temperature; (4) cooling the reactor and operating vital equipment within safe limits; (5) detecting the off-on or closed-open status of mechanical devices such as valves and doors; (6) accurately measuring the position of all horizontal control rods and completing the operator-computer control with respect to operational safety requirements. (In addition, the capability must exist to monitor the vertical rod status at all times.); (7) logging, displaying, and storing all measured data and binary inputs in meaningful, easy-to-read form for reactor operators and the physics staff; (8) providing the capability to automatically shut down the reactor without operator intervention, and (9) allowing operation communication with the computer. For example, the operator must be able to ask for heat control, turn heat control off, demand various outputs (demand displays or logging on typewriters), and move the control rods.

## CONCLUSIONS

The programmed control and instrumentation system is extremely flexible, because some members of the operating staff know how to program and can maintain the system programs without requiring outside help. When the original programs were being written, there was very close cooperation between the programmers and the operating staff--this is a necessity for a real-time system. Three members of the operating staff know how to program, while three others have lesser knowledge about programming, or computers. Their lack of knowledge about computers has in no way hindered the effectiveness of the latter group of operators. The fast response of the computer has required the programmers to carefully consider such physical problems as contact bounce, and noise on power and signal lines. Although any kind of problem with the PMACS seldom occurs at present, noise is often the cause when a

*Operated for the United States Atomic Energy Commission by Battelle Memorial Institute, Pacific Northwest Laboratory, Richland, Washington.

problem arises. The Programmed Measurement and Control System (PMACS) cost approximately the same as a comparable analog hardware system, so economy can not be claimed as a feature of the installation. However, the on-line basic data analysis with the results automatically logged and continuously displayed to the operator in engineering units makes a more efficient and safer operation. There is less chance for human error during the operations and data handling.

## MEASUREMENT AND CONTRÔL HARDWARE

The basic computer is the PDP-7, manufactured by Digital Equipment Corporation. The central processing unit has the extended arithmetic option and is supplied with an 8K word core. Computer peripherals consist of a high speed paper tape reader, a high speed paper tape punch, three DECtapes, two Teletype KSR 35's, and the necessary circuitry to read 18 digital inputs (one watch channel) into the information collector and to decode computer words for generating control pulses.

The system has two oscilloscope units which can be run in the normal mode or can be used as storage scopes. With the interface developed by Battelle-Northwest two different point plots can be presented under program control. One of the units is also used as a backup for the three-color display.

The PDP-7 was also supplied with a 16 channel automatic priority interrupt (API). There are two API modes: (1) A single instruction mode which, upon interrupt, increments a core counter and returns control to the interrupted program. (2) The second mode is the regular multi-instruction mode which allows subroutines to be entered and executed. The interrupt service routines allow interrupts from lower channels only when they have completed their function and restored the proper registers.

Two hardware modifications were made to the computer: (1) Core space and programming capabilities were increased by allowing the use of core locations 1 - 7, in addition to $10_8$ - $17_8$, as auto indexing registers. (2) Input typewriter reliability could be checked by allowing an optional change over between duplex and simplex operations. When the programming system is in control, the keyboard program prints the actual character received. If the program is not entered, depression of a key does not cause printing. This is called the duplex mode. Simplex mode is the normal typer mode with or without computer control. If the command typewriter should fail, the logging typer can replace it.

The process-computer interface was fabricated by Astrodata and consists of the following:

Analog calibration unit, multiplexer, analog-to-digital converter (ADC) and two amplifiers. Either the low-or-high-gain amplifier can be selected by computer command. There are 128 channels which are scanned approximately once every second.

Two flux digitizer units with a current-to-voltage amplifier and a voltage to frequency converter in each. Each amplifier has 12 gain ranges providing a full scale range of $10^{-11}$ to $10^{-4}$ amperes. The computer reads the digitizer outputs and controls the range settings and calibration inputs of zero and ten per cent of full scale.

System clock which uses a two-megacycle crystal-controlled oscillator. From this circuitry, there is derived the .1 second interrupt (wired into the API system) for the basic program timing loop, which results in periodic program execution and the pulsing of activity sensors, or deadmen, every .1 second. For example, the activity sensors must keep the safety circuit made up, if the facility is operating in a nuclear mode.

Rod position counters which allow the program to keep track of the horizontal rod counters.

Alpha-numeric display unit which accepts BCD data from fixed computer core locations by stealing computer cycles. The three color (blue-green-red) unit allows character generation of 26 alphabetic and 10 numeric characters, as well as, the minus sign and period. The unit will display 420 characters including spaces and color change indicators. The 420 characters, which are in a 20 x 21 matrix, require only 140 core locations in the computer memory.

Digital outputs providing 35 contact outputs used to open and close relays for saturable reactors, gas system valves, warning horns, moisture monitors, gas blowers and process water pumps.

Stepping motor outputs which drive the 9 horizontal rods at program changeable rates, and the 7 valves and 4 vertical rods at fixed rates.

Ten analog output channels with digital to analog converters. These are used for controlling the heater power and for driving the oscilloscopes.

Safety circuit which makes up flip-flops and deadmen. Functional commands are provided to turn on or set the logic level for both PMACS safety circuitry channels.

These devices making up the process-computer interface are under control of the computer program, or it reacts to them in response to their interrupts. To the computer, the many devices in the Astrodata hardware look like one peripheral. In this peripheral there is a register whose contents are decoded to signal the device action. The program tests if this Computer Data Register (CDR) is busy or not. If not busy, the program executes a transfer command which transfer a control word from the PDP-7 accumulator to the CDR. The control word in the CDR contains device identification and action.

## MEASUREMENT AND CONTROL SOFTWARE

All the HTLTR operations staff became familiar with the original software in varying degrees, and a few of them also contributed to the design and coding of some of the applications and maintenance programs. After the completion of the original software package, the operations staff took over the task of maintaining it. Examples of recent modifications to the programs will be discussed after they are briefly described in their original form. A rather lengthy report,cited under "Acknowledgements", gives complete details of the original programs.

The programs were designed to satisfy basic operations needs, rigid safety requirements, the availability of computer core and magnetic tape storage, and the requirements imposed by hardware and electro-mechanical reactor equipment.

To allow the operators to interrogate the PMACS system for reactor information at any time and initiate reactor control or actions leading to control, over 150 separate Teletype commands were made available. For those types of operator requests (via the Teletype) which demanded some sort of action such as moving rods or opening gas system valves, the programs themselves permitted the action if certain necessary conditions were met. This involved the examination of many program parameters. In the case of the 14 different logs (temperatures, pressures, raw flux, etc.) the logging programs had to reference many core locations. Other types of programs had to have access to other program inputs and outputs and parameters. To avoid duplication of program constants and common reference parameters and tables, a large equivalence system and constant pool was devised--for use by all programs, whether permanent core residents or transient (scratch area) programs.

The logic and flow of the functional programs reflected the many years of experience of the operating personnel. If a certain condition demanded a particular programmed action and even if this condition was almost sure never to happen, the programmed instruction set still had to exist. If there were not safety requirements, a program to move the control rods would have been three instructions long. There were numerous examples of this sort. In addition, the programs were written to guard against operator errors--even if safety was not the main issue.

When the reactor was in the nuclear mode at a set temperature, there were 20 major programs which completely used all 8K of core. In addition to the core residents, 26 additional programs were written--programs having a combined instruction set which was approximately 10,000 words long. Counting tables, constants and programs, there were about 18,000 programmed words. Only four of the above mentioned 26 additional programs were completely independent programs and operated mostly off-line. The rest of them formed an integral part of the programmed system. The core capacity was doubled by chain linking various operational programs and by the use of a core scratch or overlay area which was used by 17 programs. By operator request or at fixed intervals, these on-line programs were retreived one at a time from magnetic tape, stored in the scratch areas and executed. The scratch area was then free for another program.

All programming was done in machine language and was very compact in order to conserve core. Also, all calculations were done using fixed point arithmetic. Machine language was most efficient for handling the complex interrupt system and the special machine instructions required for the unique computer-process interface. The program package was a multi-programmed system to the extent that several programs could be at different levels of execution because of I/O or other delays. For example, a program could be waiting for a valve to re-position.

Table I outlines the original HTLTR software.

Table I. Software complement.

Executive Control and Service
Timing Control
Input-Output
Limit Check and Alarm
System Application
Maintenance

The first three program groups could probably be considered the system Monitor. As will be seen, the Monitor performs the usual computer system control, with the details dependent on this particular installation. The "System Application Programs" handled problems unique to the operations in the HTLTR facility, while the "Maintenance Programs" were used for calibration and routine maintenance checks of hardware components. In addition to the on-line and real-time programs, there were programs for reading in and starting them, and for shutting down the PMACS so that the computer could be run off-line or shut off.

Executive Control and Service Programs

The "Executive Control and Service" programs are listed in Table II. Because they were either part of the Monitor or used by many other routines, these programs were all permanent core residents.

Table II. Executive Control and Service Programs

Executive Control
Analog-Digital Service
Microtape Service
Binary-BCD Conversion
Message Queuing and Buffering
Units Conversion
Temperature Averaging

Executive Control Program-The functions of the Executive Control Program were to: (1) Listen for keyboard commands from the operator and call the Keyboard Executive Program when a command was detected. (2) Print out any error messages and control the message queue table. (3) Handle Microtape requests for program loading or data logging. (4) Interrogate the program on-off status table and allow execution of programs that were in core and ready for execution. (5) Start execution of the program which updated the CRT display. The program mentioned in (5) was executed after all programs in (4) had progressed as far as possible without entering a waiting loop.

Analog-to-Digital Converter Service Program-The Analog-to-Digital Converter Service Program was an interrupt service routine which was entered each time a digital conversion was completed. A total of 128 input channels were converted once every 1.28 seconds. The converted raw counts were stored in a current value table, where they were available for all function programs. The routine switched back and forth between the low gain and high gain amplifiers depending on the analog signal level in a particular channel.

Microtape Service Program-This was an interrupt service routine, which was entered when a block number was encountered, a data word had been read or written, or when an error was detected. It was initialized by the "Executive Control Program", which actually selected the proper tape unit and set the search forward mode. Three tape units were used: one for program storage, which was only read in real-time, and two others used for storing and retrieving data.

Binary - BCD Conversion-Two routines were used for converting between teletype coded decimal numbers and their binary representations. The binary to decimal conversion program was called with the binary number and the binary point location as arguments. The result was a 13 entry table with six integer numberals and six fraction numerals separated by a decimal point. The decimal to binary converter started with a table of coded integer and fraction numerals and gave the binary representation for the integer part in the AC register and the fraction in the MQ register.

Message Queue and Buffer Programs-Programs could output messages of variable lengths to the communication typer using these routines which were intermediate to the final output routine. Messages longer than three characters, including spaces, were packed three characters to a word in tables at least two computer words long. The program for queuing messages was called with the length of the message and its location as arguments. As each message was taken from the queue it was unpacked and transferred to the output program. For messages of three or less characters, a program for queuing was called with the actual message as the argument.

Units Conversion Program-The engineering units conversion routine could be called by all programs, except the interrupt service routines. The calling program supplied the conversion program with the core address of the conversion equation index. The conversion program used this information to find the correct raw number from the analog-to-digital converter, which had been stored by the ADC routine into a current value table. This raw count was converted to the particular prescribed units by using the proper programmed equation, and the converted value was then returned to the calling program. All calculations were done using fixed point arithemetic. Conversion equations were written to supply engineering units from the ADC words obtained from the following transducer types: (1) low precision and high precision resistance temperature devices; (2) two types of thermocouples; (3) Hall effect devices for measuring heater power; (4) flow transducers for measuring nitrogen flow; (5) pressure transducers for measuring nitrogen pressure; (6) a current transducer for measuring current in the gas blower motor; (7) valve position transducers; and (8) rod position transducers.

Temperature Averaging Program-Temperature averaging was included as a service program because the results were used in the three-color display, in certain typewriter logs and in some system application programs. Averages were obtained for thermocouples grouped near the top and bottom, for thermocouples on each side of the reactor, and for thermocouples in the reactor core.

### Timing Control Program

The "Timing Control Program" was entered on a 0.1 sec. interrupt, and performed the following functions: (1) Called the nuclear check programs whenever the reactor was being operated in the "nuclear mode". (2) Monitored the two analog channels which corresponded to the high precision RTD inputs. The monitoring was necessary because the two RTD's had 10 different ranges, which could be selected by the computer. (3) Every 10 seconds, turned on the Limit Check and Alarm Program. (4) Turned on the Gas Purity Check Program, one of the System Application Programs once every second. (5) When time, updated the time of the day (hours and minutes) and the day of the year. (6) Incremented, every 0.1 second, auxiliary time counters used by the gas-heat programs. (7) Read and stored the horizontal rod position counters ten times a second. The digital feedback counts were stored in a table and thus were available for use by other programs. The digital counts were converted to readings in inches for all nine horizontal control rods (HCR) and then stored in a table for use by the display program. (8) Checked to see if any HCRs or vertical safety rods (VSR) were selected and if so, entered the associated rod monitor routines, which were System Application Programs (9) Shut off any horn relays after a 4-5 second delay if they had been turned on. For example, when the nuclear program made up the safety circuit, the safety circuit horn had to be blow. (10) Called in the analog-digital calibrate program at regular intervals.

### Input-Output Programs

These programs, which are listed in Table III, provided the sequences of computer instructions necessary to drive the input and output peripherals used during on-line operation. The one input device used by the operator was the command typer. Limited output in the form of concise responses to operator commands also occurred on this teletype. Hard Copy logs appeared on a second teletype unit with identical storage of the logs on a Microtape. The three-color display presented real-time operating information.

Table III. Input-Output Programs.

Keyboard Executive
Display·
Tape-Core Buffered Output
Internal Data
Logging

Keyboard Executive Program-The Keyboard Executive Program was the essential link between the operator and the total programmed system. It allowed the observation of reactor measurements and the pre-operational checks on computer inputs and outputs and permitted the initiation of the Application Programs for the heat, cool and nuclear systems.

Display Programs-These programs controlled the presentation of permanent and transient information on the three-color display. Real-time logs displayed information about the constituents in the reactor atmosphere; valve positions and gas flow and pressures; heater power and thermocouple temperatures; and readings of horizontal control rod position from the analog position transducers.

The states of contacts could also be displayed in 18 bit groups. Any one of the real-time logs could have been displayed and continuously updated.

Tape-Core Buffered Output Programs-Using a tape-core buffer allowed the teletype output buffer to accumulate data faster than the teletype could print. For example, if several logs were requested, it could take several minutes for them to be typed and there would have been a considerable time lag between the first and last log if the log data were collected at the teletype speed. With the buffer, however, if a log was being typed and another one was requested, the logs were stored onto magnetic tape, and they were then read into the typing buffer from the tape.

Internal Data Program-This program, which was entered from the Executive Control Program every time the Executive loop was executed, caused transfer to tape of any data which may have been generated by the heat and nuclear system programs. This data was not normally available from an operating log.

Logging Programs-Fourteen different logs were a available on the typewriters. The majority of these log programs printed analog process data on the logging teletype.

One of the Logging Programs typed on the command teletype the identification, raw counts and engineering units for a single analog channel requested by the operator. He might request typing of any number of channels (one at a time) and could deactivate the log at any time.

Another Logging Program allowed the operator to request the contents of any address in the core memory during the real time operation of the system. These numbers were also typed on the command teletype.

### Limit Check and Alarm Program

This program was a permanent core resident and was routinely executed once every ten seconds. However, the program was also executed immediately after the reactor safety circuit was tripped by something external to PMACS in order to identify the source of the safety circuit trip to the operator. The primary function of this program was to examine PMACS inputs for off-normal conditions and to indicate these conditions to the operator using the color display and the logging teletype. The inputs checked were binary values from switch contacts and analog values of reactor gas constituents.

### System Application Programs

There are three major systems in the HTLTR facility: (1) Cooling, (2) Heating, and (3) Nuclear. In Table IV are listed the Application Programs related to these systems. The Gas-Heat Program handled the manual and automatic control of the cooling and heating systems; the Nuclear Program was concerned with flux measurement in the nuclear system; the Gas Purity Program determined the amounts of different types of gases in the gas (nitrogen) system; the Reactor Rods Program aided in positioning the horizontal control rods and vertical safety rods in the nuclear system; and the Scram Program identified the cause of an automatic shutdown of the nuclear system.

Table IV. System Application Programs.

Gas-Heat System
Nuclear System
Gas Purity
Reactor Rods
Scram

Gas-Heat System Programs-These consisted of 11 distinct segments on four magnetic tape links, becuase the coding was almost twice as long as the assigned core locations. Each of the different segments roughly corresponded to the state of the gas and heat systems during execution of the program segment.

Valves could be moved on command, or could be set on automatic control to maintain operator requested conditions in the gas system. In the heat system the operator could set heater powers or could request automatic temperature regulation at any temperature to 1000°C. Checking was performed by the programs to only allow valve positions, heater powers, and automatic controller setpoints which were within the physical limits of the equipment. The basic automatic feedback control algorithm was a discrete version of a continuous two-mode controller (proportional-plus-integral). Many modifications were introduced to allow for the physical characteristics of the systems. For example, different error signs had to be allowed for control of water cooled heat exchangers, as well as, for control of the pressure in the reactor containment vessel. Although there were many different types of measured variables and different ranges, automatic control was achieved with one time shared control program for the gas system and one for the heat system.

Nuclear System Program-The following program functions were performed for both flux digitizer units in the PMACS: (1) Determined if operational safety requirements were satisfied before allowing the nuclear program initialization and the "making up" of the safety circuit for the nuclear mode of operation. (2) Found the proper digitizer range, or made sure the digitizer was "on scale", and kept the digitizer on scale whenever the reactor was in the nuclear or core load mode. (3) Calculated a low accuracy period ten times each second and compared it with the lower period limit set by the operator. An automatic shutdown would occur after a predetermined number of low periods appeared in succession. (4) Every second calculated period and power values, and compared these against operator set limit values. If the period was too short or the power too high an automatic shutdown occured. (5) Every eight seconds performed a high accuracy period calculation, stored the result on magnetic tape, and displayed it on the CRT.

Gas Purity Program-This permanent core resident was executed once every second. Its primary function was to read the signals from moisture monitors and gas chromatographs and then calculate the quantities of gas contaminants in the nitrogen atmosphere. The peaks in the analog output from the chromatographs were found by the program, but the operator had to synchromize the chromatograph self timer and the program. The program monitored the accuracy of the synchronization by the location of the nitrogen peak. Moisture content was calculated directly from the outputs of moisture monitors every four minutes.

Reactor Rods Program-With this program the operator was able to manipulate the horizontal control rods and the vertical safety rods safely, because it allowed only predetermined sequences of operations. The program did not initiate rod motion automatically, but merely related operator specified movements to predefined safety criteria and movement sequences.

Scram Program-The Scram Program identified the cause of an automatic shutdown. Shutdowns caused by the programmed system could have been the result of the program bug of executing a zero operation code, or off-normal conditions as signalled by switch contacts and analog signals. An automatic shutdown resulting from conditions external to the programmed system was signalled by a hardware interrupt. For example, if the reactor door, which makes contact with a limit switch, was opened, an automatic shutdown would occur; a too short period could cause a shutdown; or the operator could push the "Shutdown" button, which triggered the hardware interrupt.

### Maintenance Programs

These programs, which are listed in Table V, differed from a memory checkerobard test, for example, in that they were part of the on-line program system.

Table V. Maintenance Programs.

Valve Calibration
Analog-Digital Calibration
Rod Maintenance
Digitizer Maintenance

Valve Calibration Program-The operator requested this program to calibrate one valve in either the full open or closed positions using limit switches. Calibration of valves was necessary in order to determine the actual position of a valve stem or to maintain agreement between the digital drive motor pulse count and the analog position feedback signal.

Analog-Digital Calibration Program-This program checked for proper operation of typical analog-to-digital converter channels which contained a low level voltage, a high level voltage, a low precision RTD, and a high precision RTD.

Rod Maintenance Program-This was used when the facility was in the non-nuclear mode for making up the safety circuit and then driving a control or safety rod for testing, calibrating or repairing. Response times of the independent safety circuits could also be determined.

Digitizer Maintenance Program-The program permitted the operator to read the neutron flux without making up the safety circuit, to check zero offsets, and to make adjustments to the digitizer units.

### Initialization and Shutdown Programs

Before the real-time programs were loaded into core and their execution started, the PMACS hardware was routinely tested to a limited extent for malfunctions, and then it was initialized for the real-time programs. Initialization was performed by program, or a message was typed requesting the necessary operator actions. In addition to the hardware initialization, the queues, switches, data tables and flags for the resident programs were also initialized.

A shutdown program provided messages and programmed actions to assure the operators that the PMACS was in a safe condition and could be left unattended.

## OPERATING EXPERIENCE

The operator has a typewriter keyboard and a single spring loaded toggle switch to operate most of the equipment in the reactor systems. The list of over 150 acceptable commands, which he can type, is too large for an operator to easily memorize, but he can memorize the routinely used ones and refer to a convenient tabulation of the others. As an example, the operator moves control rods by enabling the rod drivers for specific control rods with typewriter commands and then moves the rod or rods by pressing the toggle switch. At any time a conventional push button switch, the reactor shutdown button, can be used to disable the rod hold power supply and shut down the reactor instantly. Using a teletype for controlling the reactor systems has not been too unwieldy.

Training for the operating staff consisted of reviewing and actually writing programmed functions, and participating in the final debugging. The operators performed the design tests, which provided them with further experience.

Failure in the computer mainframe has caused no unscheduled process shutdowns, however failures in the computer peripherals and in the computer-process interface have occurred.

HTLTR has been operating over a year as an experimental reactor. During this period both the PMACS hardware and software have been modified, with all programming being done by the HTLTR operations staff. All modifications are reviewed and approved prior to their final incorporation in the Programmed Measurement and Control System. Also there have been some additions to the process equipment external to PMACS which will not be considered here. The majority of changes in the PMACS software and hardware were made in three general areas: (1) Programs were added to make up for deficiencies in the original program package, or to add capabilities that had not been though of for the original system. (2) Programs were changed to remove bugs that showed up during routine operation. (3) Changes were made to improve the operation of the reactor facility.

In some cases the program deficiencies were recognized at the completion of the original system, but were not included because of time limitations. An example of this is a tape retrieval program for logging information from Microtape. Another added program continually checks that the central processor is not locked in a loop which is smaller than the Executive Control Loop. This insures the operator that all programs, which have been requested and turned on, are being executed. As another example, in order to allow unattended operation of some parts of the HTLTR facility with the PMACS active, an annunciator and typewriter have been installed in an adjacent reactor facility. When a malfunction occurs, the computer sounds the alarm and types a description of the malfunction.

Modifications in the system were made to simplify the operator's job of operating the facility for experimenting, as well as programming and maintaining equipment. One change that was made was increasing the use of the three-color display for programming and for checking out the hardware. Several changes were made to make the keyboard a less cumbersome means for commanding the PMACS operation. For example, the Keyboard Executive Program was expanded to recognize more mnemonics, and some mnemonics were shortened. Accuracy of some calculations was improved. The gas chromotograph was removed from the PMACS system, because of the difficulty of keeping track of the gas peaks as a function of time. Gas chromatographs have been under computer control, and the one at HTLTR could probably be controlled also. However, the operations staff felt that proper control of the chromatograph would take too much core storage that would be more valuable for other programs.

Modification of the software to continually improve the operator's capability for controlling the process was one of the prime arguments for including a computer in HTLTR's instrumentation system. That this original argument was sound, has been proven already in the short operating life of the facility.

## ACKNOWLEDGEMENTS

This paper is based on work performed under United States Atomic Energy Commission Contract AT(45-1)-1830. The author wishes to acknowledge other Battelle-Northwest staff members who participated in the writing of the original report* from which most of this paper was abstracted. He particularly thanks R. G. Clark, D. D. Lanning and other members of the HTLTR operating staff who provided information about HTLTR operating experience.

*Computer Programs for Operation of the High Temperature Lattice Test Reactor (HTLTR), Volumes I and II (BNWL-651), December, 1967 by L. D. Coffin, S. J. Simmons, J. R. Kosorok, C. E. Haines and W. C. Roberts.

# THE USE OF THE PDP8 IN AUTOMATED TEST SYSTEMS AT SANDIA LABORATORIES

Glenn Elliott
Sandia Laboratories
Albuquerque, New Mexico

## ABSTRACT

The PDP8 series of computers is being used frequently in automated test systems at Sandia Laboratory. Several system configurations are presented briefly. Emphasis is placed upon the solution of general problems arising from their use in a large organization of design engineers and in applications where delivery-schedule time scales are very short. General interfacing methods are discussed and a semimodular approach is shown. Sandia's method of obtaining programs from an engineering organization where no programming talent previously existed is presented.

The author is a member of a group of about 50 engineers and technicians at Sandia Laboratories responsible for the design of test systems. These test systems are used in development and acceptance testing and for quality verification of components used in the AEC's weapons programs.

Typically, a requirement for a system is not known until late in the tested item's development process. This results in a very tight time scale being placed upon the test-system designer. The procurement of parts for the systems are routinely on a "crash" basis, with the frequent burning of large quantities of "midnight oil."

One of the difficulties in such an operation is that the designers tend to fall into familiar ruts and change does not come easily. This could result in bypassing new and more effective ways of accomplishing the jobs at hand. With the advent of the small, inexpensive digital computer came the basic question: Could these new machines be used to an advantage in our test systems? Of course, the answer was definitely affirmative, but some problems were apparent which would need solutions within our framework of doing business.

1. Could our design engineers be trained to understand and apply digital computers?
2. Could we interface the computers with the proper transducers and control circuitry necessary to do the complete testing job?
3. Could we find qualified programming help?

We stepped gingerly forward, ordered our first PDP8, and started our first application. This was about 18 months ago. Today we have purchased 23 of the PDP8-series computers, and their use has become almost routine.

Before we discuss the solution to the previously mentioned problems, let us look briefly at some of these applications.

## INERTIAL SWITCH TESTERS

PDP8/S's have been used on three inertial-switch testing programs. Two of these programs involve testing of rather conventional slug mass devices, and the other a device based upon Sandia's new Rolamite principle. Typically, the computer controls a centrifuge to follow some precise acceleration versus time function by implementing a digital-to-analog converter. Readings are taken of acceleration levels, contact resistance, high-voltage leakage currents, and several other variables. The test operation, data taking, and presentation are entirely under the control of the PDP8/S, and are completely automatic.

## TIMER TESTER

A precise, multichannel electronic sequence timer is tested, using a test system based upon a PDP8/S. This project involved the interfacing of the computer with a number of high-quality commercial electronic timers.

## JUNCTION BOX TESTER

At first glance, the use of a digital computer for testing of a simple thing, such as a junction box, would appear to be an unlikely application. However, the inspectors and production engineers could easily be overwhelmed by the quantity of data from loop resistance tests and leakage-current tests on multipath junction boxes. The computer removes this problem by only presenting out-of-limits readings, and then, on a lot basis, quality summary information such as X bar and sigma.

## THERMAL BATTERY TESTER

A thermal battery is a one-shot device. Once the heat-producing chemical reaction is complete and the battery has produced its current and voltage, the device is discarded. This means that thermal-battery test systems must be reliable. A production engineer tends to be rather unhappy if a thermal battery is fired without obtaining useful data.

The PDP8/S computer controls a binary-coded-decimal resistor bank to produce a rapidly and precisely controlled varying load. Output voltages and currents are measured periodically.

## EXPLOSIVE POWDER DISPENSER

A PDP8 is used as a process-control computer to control the dispensing of small quantities of explosive powder to a high degree of accuracy. Naturally, the system is completely automatic, being located in a remote concrete blockhouse.

## MASS PROPERTIES LABORATORY

A PDP8 is used in a time-sharing mode to record data from two balancing machines and several moment-of-inertia devices. Three teletypes operate into an executive program to permit the operator to enter independent variables and receive output results.

## RADAR TESTER

A PDP8/I with discs and IBM compatible magnetic tape is used to assemble data taken from four sophisticated radar testers. This is the subject of another paper presented at this meeting by Mr. Tom Rau.

In reviewing these examples of application, one can see that the computers have been put to a wide variety of uses. A number of engineers have been involved in the design and checkout of these systems. It is interesting to note also that these engineers are a part of an organization where two years ago very little computer background existed.

The first problem tackled was the training of personnel. A few individuals were sent to the DEC programming school in Maynard. But this proved to be somewhat ineffective, since the curriculum of the school assumes considerable prior knowledge of computer technology and programming. Upon receipt of the first PDP8, one man was assigned to become familiar with the machine, with the intent that he conduct classes for others. This was done with 10 individuals in the first class. Two of these were subsequently sent to the Maynard course and reported it to be of little value after taking our own in-house training. The next training sessions included members of the Sandia Educational Division, who are trained engineering instructors. More recent classes have in turn been taught by them. To date, over 200 have taken our in-house computer course. Attendance by supervisory and managerial personnel at recent sessions has been heavy.

The second big problem undertaken was the interfacing problem. This, in retrospect, proved to be less of a problem than originally thought. Fortunately, our engineers talked to each other, and informal standards soon evolved. DEC logic boards (R and W series, primarily) are used, and have proved to be satisfactory. The digital engineer certainly cannot be bothered by problems residing within the logical building blocks, and the DEC equipment fills the bill in that respect.

In our applications, typically, a number of relays are operated by the computer under program control. A technique, using either the W103 interface board or a combination of R107's, B117's, R203's, and W061's, was originated early and has not varied greatly. Typically, schematics of earlier jobs are red-marked for the current job and sent to drafting for rework and to the fabrication shops for wiring of the card racks.

Obtaining effective programming aid proved to be a more difficult problem. We have determined that it is absolutely necessary for the design engineer to be a capable programmer. This is necessary for him to understand his system in all its aspects. But to prevent his being burdened by the intricate details of programming, it is desirable to have a professional programmer working with him. Originally, it was thought that any programmer qualified on any modern computer would suffice. This did not prove to be totally true. With some programmers, thoroughly experienced in using highly advanced software, the gap was just too great for them to communicate on a hardware-oriented level with our engineering staff.

Recently, we have specified programmers with engineering background as those qualified for our jobs. This service has generally not been available within our own house, and we have gone to outside programming houses with good results.

In conclusion, then, it would appear that we at Sandia Laboratories have just begun to scratch the surface in our application of small digital computers. We anticipate a rapid growth in the use of the computer in our business in the immediate future.

## Acknowledgment

This work was supported by the U. S. Atomic Energy Commission.

# COMPUTER CONTROL OF HYDRAULIC TESTS

Lyndon A. Thomas
Research and Productivity Council
Fredericton, N.B., Canada

## ABSTRACT

This paoer describes the use of a PDP-8/S computer in obtaining the characteristics of hydraulic rotating machinery. The aim of this work is to provide a raoid and accurate method for carrying out such tests.

Some of the interface problems, methods of handling the data and orogramming techniques which are peculiar to this tyoe of system are discussed.

A programming language, which makes use of macros and an ooerating system, has been written providing the user with a oowerful test control and data acquisition system.

## Introduction

The problem of obtaining the steady state operating characteristics of hydraulic pumps, motors and transmission systems can be considered at three basic levels:

1. All the work, including orocessing test data and olotting graohs, to be done manually.

2. Setting uo the test conditions manually, but using a data logging system to collect the test data for further orocessing by comouter.

3. Using a comouter to set up the test conditions, to suoervise the collection of test data, to orocess this information and to outout it in the required form.

If only a few results are required at fixed ooints on the characteristic curve of the device under test, then a manual system will normally suffice.. However, if a comolete set of characteristics is required, then to perform the tests manually is a lengthy, error-prone orocess.

The object of the test rig described here is to provide a system capable of determining and olotting the characteristics of a hydraulic transmission system quickly and accurately.

Figure 1 illustrates a tyoical set of graphs which may be required for a motor test. Not all of these may be needed for any one test, and many other characteristics are also possible.

## Design Criteria

The following are some of the considerations which were taken into account during the design of the system:

### 1. Flexibility

Since many different kinds of test are carried out on one test bed, system flexibility is essential.

### 2. Accuracy

In the absence of an international testing procedure, tests are carried out according to principles laid down in a draft proposal being considered by the British Standards Institute. This proposal will probably be adopted in a modified form as an international standard, and is therefore used as the basis for this work. This proposal calls for accuracies of the order of .5% in all measurements and this figure is used as the maximum permissible error for this system.

### 3. Simultaneous Readings

The proposal mentioned above states that all readings should be taken simultaneously. This is not possible when testing manually, but can easily be achieved in a computer system.

### 4. Manual Capability

It is sometimes desirable to run tests manually, when no programme exists for the required test and only a few spot checks are required.

## 5. Ease of Programming

An easy method of programming tests had to be provided so that non-standard tests could be programmed easily and with a reduced risk of error.

## Description of System

Figure 2 shows the system when set up to carry out tests on a variable stroke hydraulic motor.

The three input voltages $V_1$, $V_2$ and $V_3$ can be derived from manually set potentiometers or from the computer through digital to analog converters. These voltages control the pump output flow, motor torque and motor stroke respectively. It can be seen that in all three cases the control loop is closed outside the computer. This is done to provide system stability when running under manual control. It can be shown that this does not limit the flexibility of the system when running under computer control.

Consider the loop controlling the piston of the linear actuator which sets the pump stroke. The amplifier output current "i" is given by

$$i = K(V_1 - V_{FB}) \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

where $V_{FB}$ is the voltage fed back by the potentiometer and K is the amplifier gain.

If the computer output voltage is $V_1'$ instead of $V_1$ such that:

$$V_1' = K_1V_1 - (K_1-1)V_{FB} \quad \ldots\ldots\ldots\ldots (2)$$

Then

$$i = K\left[K_1V_1 - (K_1-1)V_{FB} - V_{FB}\right] \ldots\ldots (3)$$

$$= K\ K_1(V_1-V_{FB}) \ldots\ldots\ldots\ldots\ldots\ldots (4)$$

and the effective amplifier gain is now modified by the term $K_1$ and becomes $K.K_1$.

Similarly by setting

$$V_1' = V_1 + V_{FB} \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

the amplifier output is given by

$$i = KV_1 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6)$$

which means that the feedback loop has been effectively opened.

Therefore, closing the loop physically does not impose any restriction on the system when running under computer control.

## Measurement of Variables

Signals to be monitored are converted into voltage levels or pulse trains. Voltage levels are multiplexed into an analog-digital converter. Pulse trains are monitored by using the pulses to gate a 10 Mc/s clock, and the clock pulses are counted on a 23 bit binary up counter with a final flip flop to detect overflow. Time interval measurement is used because the pulse trains have a low repetition rate and a long sampling time would be required if a fixed time interval pulse counting technique were used.

Measurement of flow is carried out by means of a turbine flowmeter. This type of meter gives a number of pulses (N) for each revolution of the turbine. The number N is equal to the number of blades on the turbine. It is necessary to divide the output pulse train by the factor N before gating the clock, so that one complete revolution of the turbine is timed, and errors due to inaccuracies of the blade positions are therefore eliminated.

## Software

Figure 3 illustrates the computer core map during normal operation of the computer.

The binary loader has been modified so that in the event of a checksum error when a user's programme is being read in, the user area in core is cleared to "halt". This minimizes the possibility of a programme running wild, which could cause damage to the system.

The operating system consists of a floating point package, I/0 subroutines and all subroutines needed for test purposes. This includes routines to perform "do loops", conditional branching, control functions, XY plotter control and so on. In fact, the only instructions which the user is allowed to incorporate in a programme are macros and entries into the operating system subroutines. This further reduces the possibility of damaging the hydraulic equipment when running a programme which has not been debugged.

The supervisor is an independent programme which is entered at regular time intervals by pulsing the programme interrupt line with a clock pulse. When the supervisor is entered, the test is frozen for a short period and all of the system variables are monitored to ensure that they fall within safe predetermined limits. The supervisor is able to initiate an alarm or a system shutdown if it detects an "out of limit" condition, but will allow the test to continue if all conditions are satisfactory.

## Data Handling

There are three sources of data to be considered:

1. The keyboard
2. A/D converter
3. Binary counter

Instead of having three different input macros, a common "read" instruction is used as follows:

Any user programme begins with a series of data definition statements such as:

```
KBD   A  (source - keyboard)
BIN   B, 1, 230 (source-binary counter)
AD    PI, 1, 526 (source-A/D converter)
```

These macros will be expanded as:

```
        A, Ø
           Ø
           Ø
        ØØØØ (data source indicator -
               keyboard entry)

        B, Ø
           Ø
           Ø
           Ø
        4ØØØ (data source indicator -
               digital entry)
           1 (digital channel number)
         23Ø (transducer number, for use
               with calibration table)

       PI, Ø
           Ø
           Ø
        3777 (data source indicator -
               analog entry)
           1 (multiplexer channel number)
         526 (transducer number)
```

The instruction "READ A" is interpreted as a jump to the read subroutine with the address of the variable "A" following the subroutine jump instruction. The read subroutine will then check the data source indicator and enter one of three subroutines depending on the indicator value. If the data source indicator shows that the source is the A/D converter, the following sequence is initiated:

1. The multiplexer channel is examined and the correct channel is selected.

2. The A/D converter is sampled a predetermined number of times, and the arithmetic mean value calculated.

3. A stored table is consulted for the transducer numbers concerned. This table provides calibrated data corresponding to the result obtained from the above averaging process. This calibrated data is given directly in engineering units. By using linear interpolation between the points in this table, it is therefore possible to correct for nonlinearity in the transducer and to convert the result into engineering units in one step.

4. Finally, the result is stored in the allocated location.

The averaging process mentioned above is, in fact, a simple numerical filter used to filter out unwanted ripple in the variable being measured. The characteristics of this filter are shown in Figure 4 for samples of 1, 3 and 10.

The filter output $h(\Delta t,f,n)$ is related to the sinusoidal input $g(t)$ by the relationship

$$h(\Delta t,f,n)=g(t_1+\frac{n-1}{2}.\Delta t).A(n,x)$$

where $A(n,x)=\dfrac{\sin n\pi x}{n \sin\pi x}$

and $x = f.\Delta t$

$f$ = frequency of $g(t)$

$\Delta t$ = sample interval

$n$= number of samples taken

The derivation of this relationship is trivial.

It can readily be seen that all frequencies between $f=0$ and $f=\frac{1}{\Delta t}$ can be attenuated, and when the sample number becomes large, the filter becomes a narrow band comb filter. All frequencies above $f=\frac{1}{\Delta t}$ can be attenuated by means of a suitable low pass electronic filter if necessary. Using the graph shown in Figure 4, it is therefore possible to determine the sample number necessary to attenuate ripple if the ripple frequency is known.

## Conclusion

Using the system described here, it has been possible to reduce the total time required to carry out comprehensive tests on hydraulic rotating equipment, and to maintain a high degree of accuracy. At present, system variables are controlled by means of a simple finite difference equation, and the operator has to specify three constants which determine loop gain, error reset and damping. An alternative "adaptive" system is being planned to eliminate the need for this, so that the operator will not be concerned with changes in the system performance when the device under test is changed.

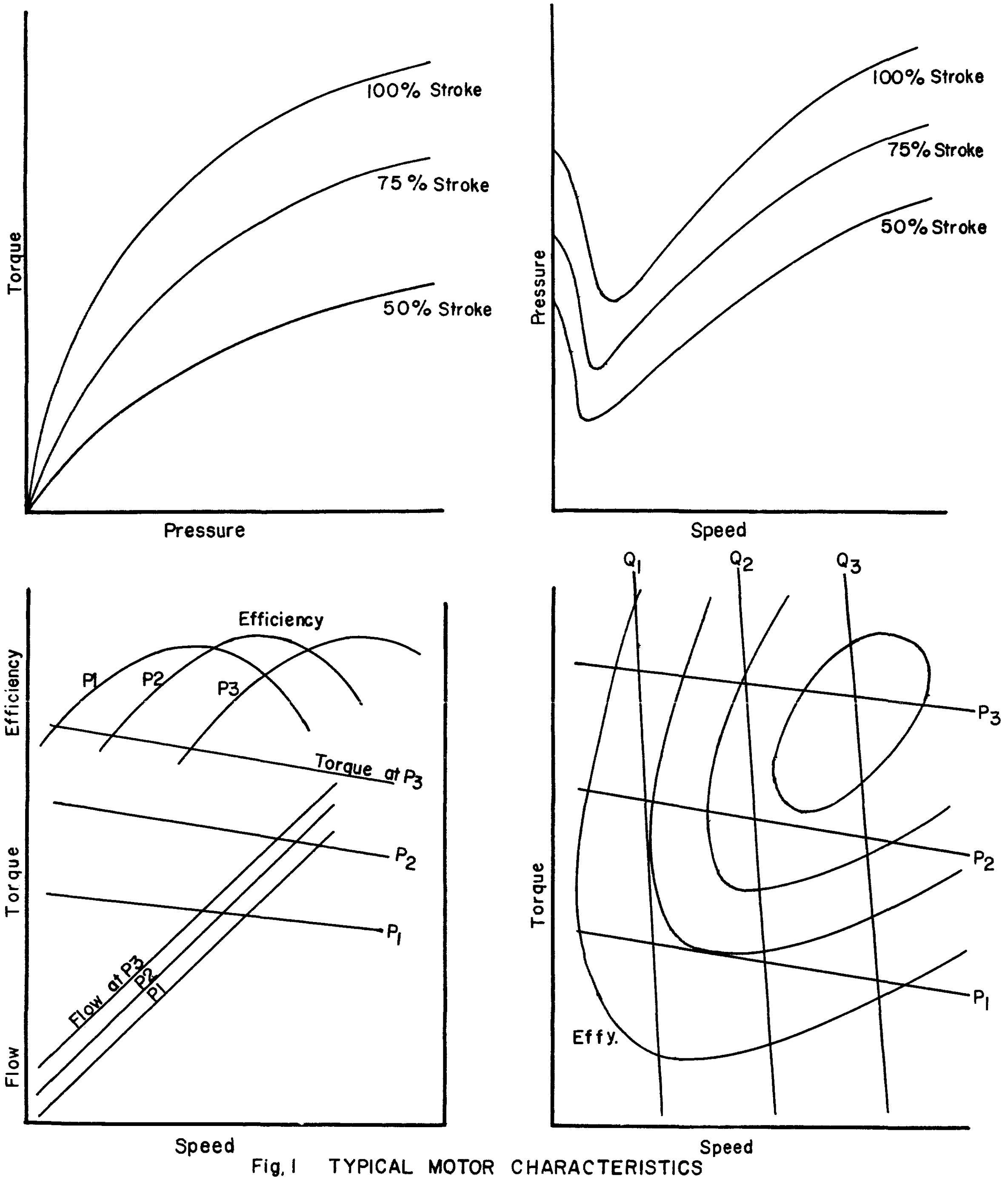


Fig. I TYPICAL MOTOR CHARACTERISTICS

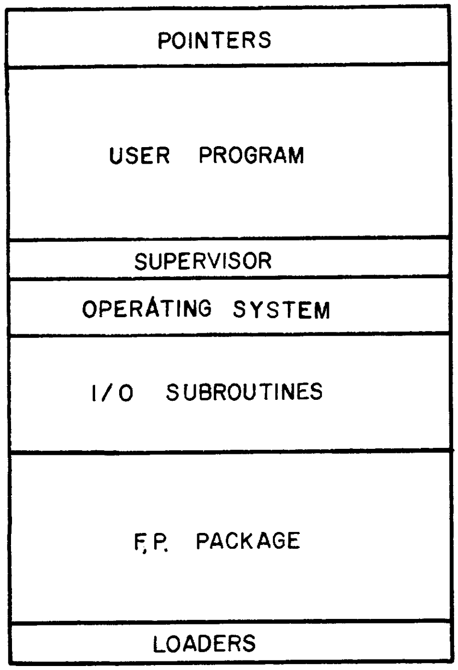


Fig. 3 CORE MAP

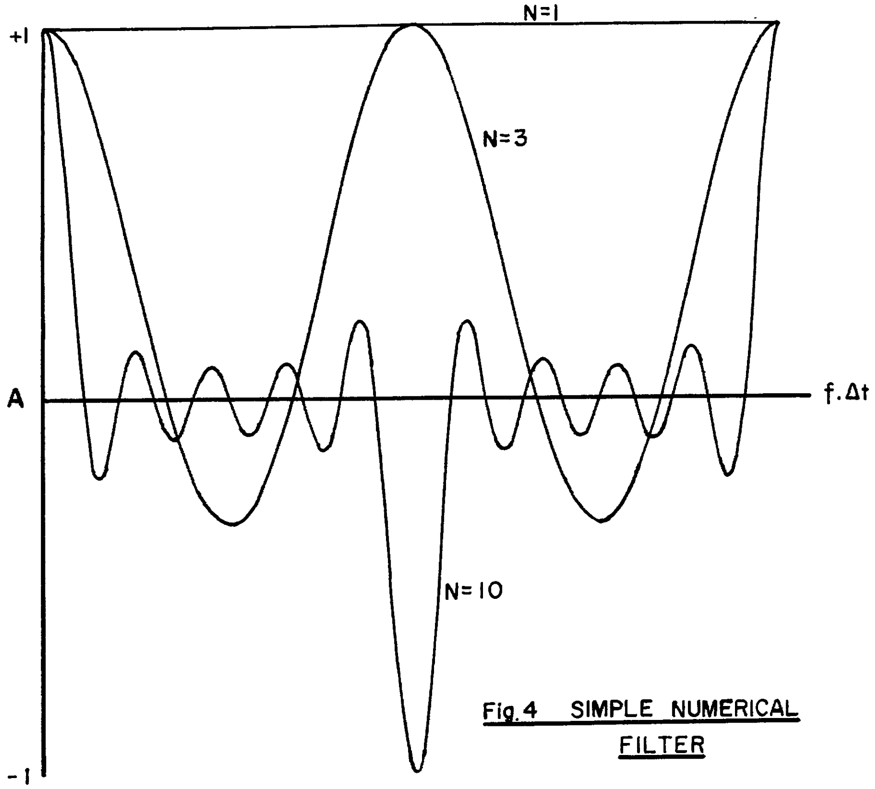


Fig. 4 SIMPLE NUMERICAL FILTER

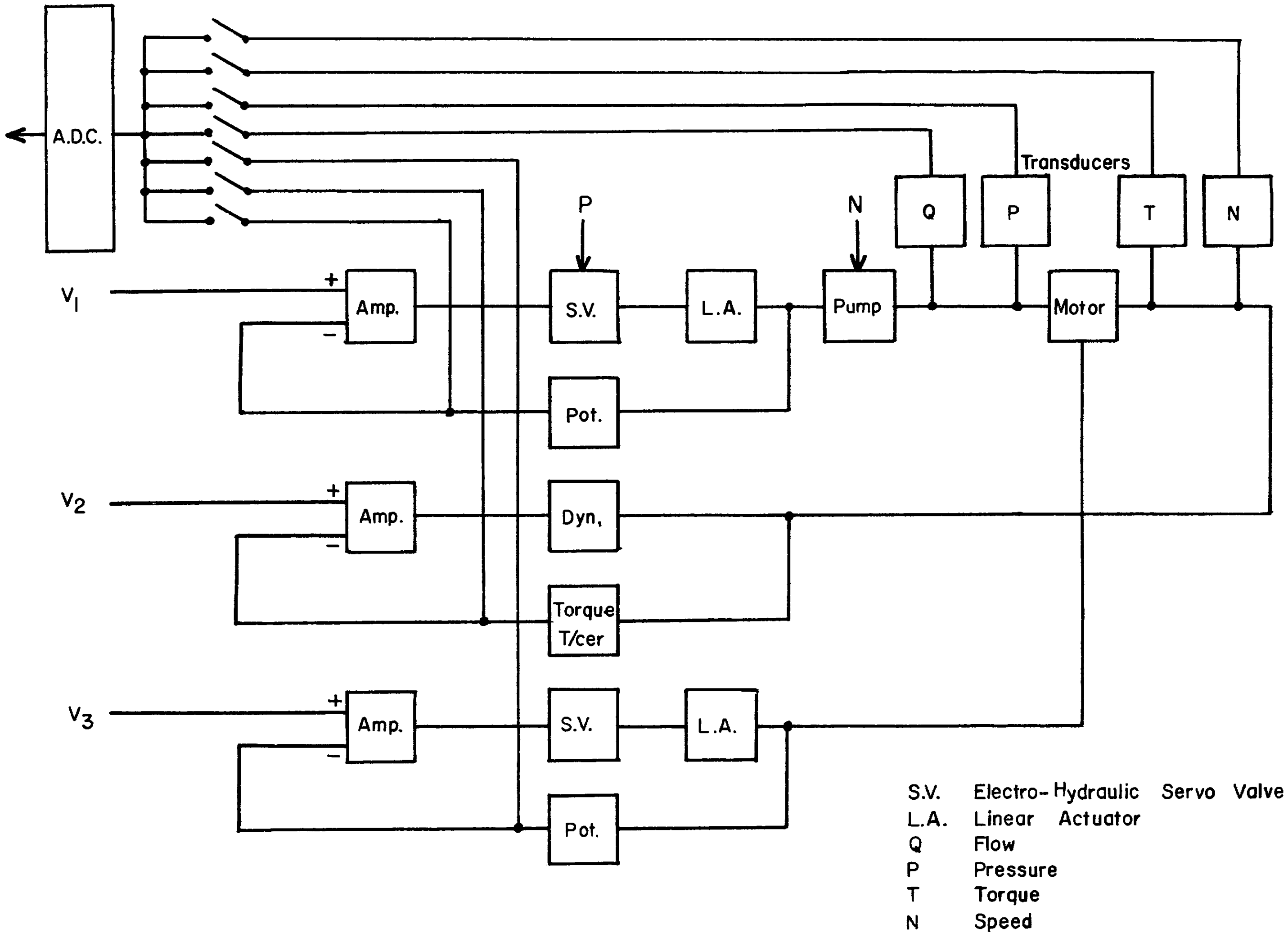


Fig. 2 MOTOR TEST

# QUEUE STRUCTURES IN A PDP-9 DATA ACQUISITION SYSTEM

Barry L. Wolfe
and
Sol B. Metz
Information Control Systems, Inc.
Ann Arbor, Michigan

## ABSTRACT

The paper describes the methods of implementation of a data acquisition system in a manufacturing enviroment on an 8K PDP-9 with DEC tapes. The system maintains production unit counts and updates production schedules, logs processing unit state changes, reports exception conditions, and processes real time inquiries. The system utilizes queue structures for in-process data in order to conserve and dynamically allocate core storage.

## INTRODUCTION

The primary function of a data acquisition system in a manufacturing enviroment is maintenance of production records. For the system being considered, this requires maintaining:

1. counts of the number of production units passing by each of two counting points on each processing unit;

2. the identification and the schedule for each of four production assignments for each processing unit;

3. the identification of the three operators assigned to each processing unit;

4. records of the durations of each of nine operational modes for each processing unit.

The system is implemented on an 8K PDP-9 with power failure protection option, three DEC tapes a CR01E card reader, multiple teletypes, a DS02BA data scanning unit, and a DR01B light driver unit.

Production unit counts are derived from the data scanning unit input. The passage of a production unit through a counting point complements a flip-flop. Comparison of the flip-flop with its previous state indicates whether a production unit has been produced. Processing unit operational modes are input through the data scanning unit. Production assignments, schedules, and schedule changes may be input through the teletypes and the card reader. Operator assignments are input through the teletypes. Interrogatives and commands are input and replies are output through the teletypes. Exceptional operating conditions are reported on the teletypes and, by means of the light driver unit, are reported on a light at each processing unit. Data is logged on punched paper tape and backed-up on DEC tape.

The system is core resident and occupies in excess of five thousand locations. In addition, production records for each processing unit require in excess of forty words, the bulk of which is DEC tape storage. The system is completely protected against power failure. All input and output is under interrupt control. Processing unit scanning is initiated by fixed interval clock interrupts, but processing unit scanning is fully interruptable. The system maintains clock time.

## SYSTEM CAPABILITIES

Each processing unit is a member of a class and a subclass. The number of processing units, subclasses, and classes is variable; the number of processing units in a subclass and subclasses in a class is variable. Four production assignments are maintained for each processing unit. A processing unit has nine operational modes. One of these modes indicates the processing unit is inactive. If the processing unit is in any of the other eight operational modes, it is active; its mode indicates which of the four production assignments is active and whether production units are to be counted.

The system recognizes and reports two kinds of assignment completion conditions. An overflow condition exists when a processing unit's production assignment has been fulfilled. Production units with the same identifications may be assigned to multiple processing units. If this situation exists for processing units in different subclasses, it is assumed that the assignments represent different stages of completion of the production units; if this situation exists for processing units in the same subclass, it is assumed that the production units are functionally identical. A second kind of overflow can exist. When the sum of the assignments of functionally identical production units has been produced, overflow conditions exist for the functionally identical assignments. Either type of overflow may also occur as a result of schedule decreases. Each class of processing units is serviced by one teletype. When an overflow occurs the condition is reported on the teletype servicing the class. When an overflowed assignment is active, a light on the processing unit is on. When schedule changes or schedule increases cause either type of overflow condition to terminate, the condition change is reported on the teletype and the overflow light is turned off.

Teletype reports indicate the start of production of an assignment. When a processing unit enters counting mode and was last in counting mode for a different assignment, a report of the entered mode is made on the class teletype.

Data logging occurs for each mode change, production schedule change, production assignment change, operator assignment, and for console teletype initiated changes to the system's real-time clock. The system maintains an operator shift schedule for each class. When shifts change the class is logged. When a processing unit enters counting mode and has not been in counting mode since its shift change, a report of the entered mode is made on the teletype for the class. Each teletype may accept commands for shifting its class or any processing unit in its class. Data is logged on DEC tape. When the paper tape punch is available, the data is copied from DEC tape to paper tape.

The system recognizes additional teletype commands to enable and disable selected overflow lights. The system also recognizes teletype interrogatives which cause typeouts of:

1. the operators of a processing unit;

2. the status of a processing unit;

3. the status of specified modes of a processing unit;

4. the status of every processing unit is a subclass;

5. the status of every occurrence of a specified production assignment in a subclass.

A teletype will only accept commands relevant to processing units in its class; however, the console teletype will accept commands for any processing unit. All interrogatives are acceptable from any teletype. In order to suppress an item of teletype output, a one character command is entered from the keyboard while the output is occurring. When available core storage reaches critically low levels, further keyboard input is temporarily prohibited; the prohibition is indicated by failure to echo. If teletype output is pending and user has taken more than two minutes to type a command or an interrogative, the system takes the teletype from the user.

## SYSTEM IMPLEMENTATION

Faced with this hardware complement and system capability and the design objective of maximizing the potential for system expansion and processing unit additions, the decision was made that the system would be core resident and production records would be DEC tape resident. However, DEC tape access speed was judged insufficient for random access to production records. A scheme was devised for batching tasks requiring access to a production record. With this scheme production records are accessed sequentially and when a production record is available the batch of tasks requiring it is processed.

One production record exists for each processing unit. For each production record, four words are core resident and forty-five words are DEC tape resident. Which assignment's production information is in core is a function of the mode of the processing unit. The objective is that information which must be accessed most frequently be in core. The information in core includes the status last reported by the data scanning unit, the status of the overflow light, the production unit counts for the active assignment, and the production schedule for the active assignment. Two pointers are also part of the in-core production record. One of these is the queue-head for the batch of tasks requiring access to the DEC tape production record. The other points to a production record item which is shared by functionally identical production assignments.

Core available for task queues is divided into fixed length blocks of consecutive words. These blocks are strung into a garbage queue. A garbage queue-head points to a block which in turn points to another block, and so on. When a task is recognized, the block which the garbage queue-head points to is removed from the garbage queue. This is done by replacing the pointer in garbage queue-head by the pointer contained in the removed block. The information which describes the task is then placed in the block. For some tasks multiple blocks are necessary, in which case multiple blocks are removed from the garbage queue. The block describing the task is placed in the task queue of the appropriate processing unit. The pointer currently in the queue-head is placed in the block being inserted and a pointer to the block being inserted is placed in the queue-head. If multiple blocks are required, the first block is queued in the usual manner and a word in it is used as a queue-head to point to a queue of additional blocks. When the task has been completed, the procedure is reversed: the block is removed from the task queue and restored to the garbage queue. If the task required multiple blocks, the queue of additional blocks must also be restored to the garbage queue. The process of removing the task is actually part of the task, that is, the task defines its disposition.

The new end of the queue is the end which contains the most recent addition, that is, the end to which the queue-head points. It is generally desireable to process the queued tasks in the same order in which they were generated. To access the oldest block in a queue, it is necessary to step through the queue until the oldest block, designated by a special end-of-queue code in its pointer, is found. The procedure for insertion and removal of blocks from any part of a queue is simple but is fastest for the new end of the queue. This speed facilitates rapid queuing of tasks generated in real time. Tasks are subsequently executed at leisure.

Tasks often arise which require access to many production records. In this situation, the task is self perpetuating which means that its execution causes it to be moved to the queue for the next production record. A task may also be capable of creating additional tasks during its execution. Before creating a task, the system checks a count of available garbage blocks. If the maximum of the number of blocks which may be required during the task's execution is not available, creation of the task is deferred until its requirement can be met. If the blocks are available, they are reserved by reducing the garbage block count. When the task is being executed and conditions do not require the creation of all of the additional tasks, the reservations are cancelled by incrementing the gargage block count.

Tasks often require access to teltypes. If the teletype is not available, the task is left pending until the required teletype is available. In this manner the queues also stack teletype output. Each teletype has a single line buffer used for both input and output. Associated with a teletype buffer are several indicators. One of these indicates whether the buffer is currently assigned to a task. Another indicates how many tasks have bids pending for the teletype. When a task which may generate teletype output is created, a bid is made for the teletype. Before the task is executed, the system determines if the teletype is already assigned to a task. If it is assigned, any other task requiring it is left pending. If it is not assigned, it is assigned to the task requesting it, and its bid count is decremented. When the task to which the teletype is assigned releases it, it becomes available for reassignment. A teletype is available for input only when it is unassigned and its bid count is zero.

The core resident portion of the production record contains a pointer to production information which is shared by functionally identical production assignments. Master schedules must be maintained to show the sums of the individual schedules of functionally identical production assignments. The master shcedules reside in core but exist only for assignments which are active. Many core production records may point to the same master schedule. When an assignment becomes active, the system determines whether its master schedule is in core. If it is not, a task is created to generate a master schedule.

When a production schedule ceases to be active and is not functionally identical to any other active production assignment, the block of storage used for its master production schedule is released. Master production schedule core blocks which have been released are strung into a garbage queue. An active master production schedule contains a count of the number of active assignments pointing to it. When an assignment ceases to be active, the count in its master production schedule is decremented. If the count becomes zero a task is created to release the master production schedule block.

## CONCLUSION

Batching tasks makes it possible to maintain production records in auxiliary storage. The number and size of production records can be expanded greatly without necessitating design changes to the system. With core free from large amounts of production information, it is available for the addition of system features. Queuing dynamically allocates available storage to the tasks and production records which require it. Because this is done, arbitrary limits are not imposed on the allocation of storage.

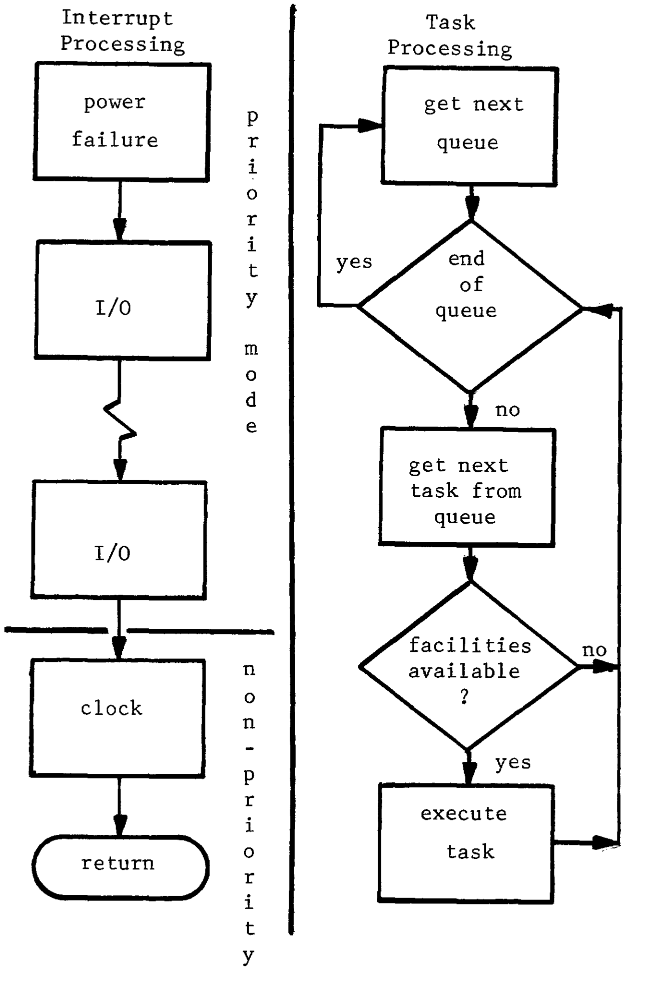


Figure 1 Program Flow

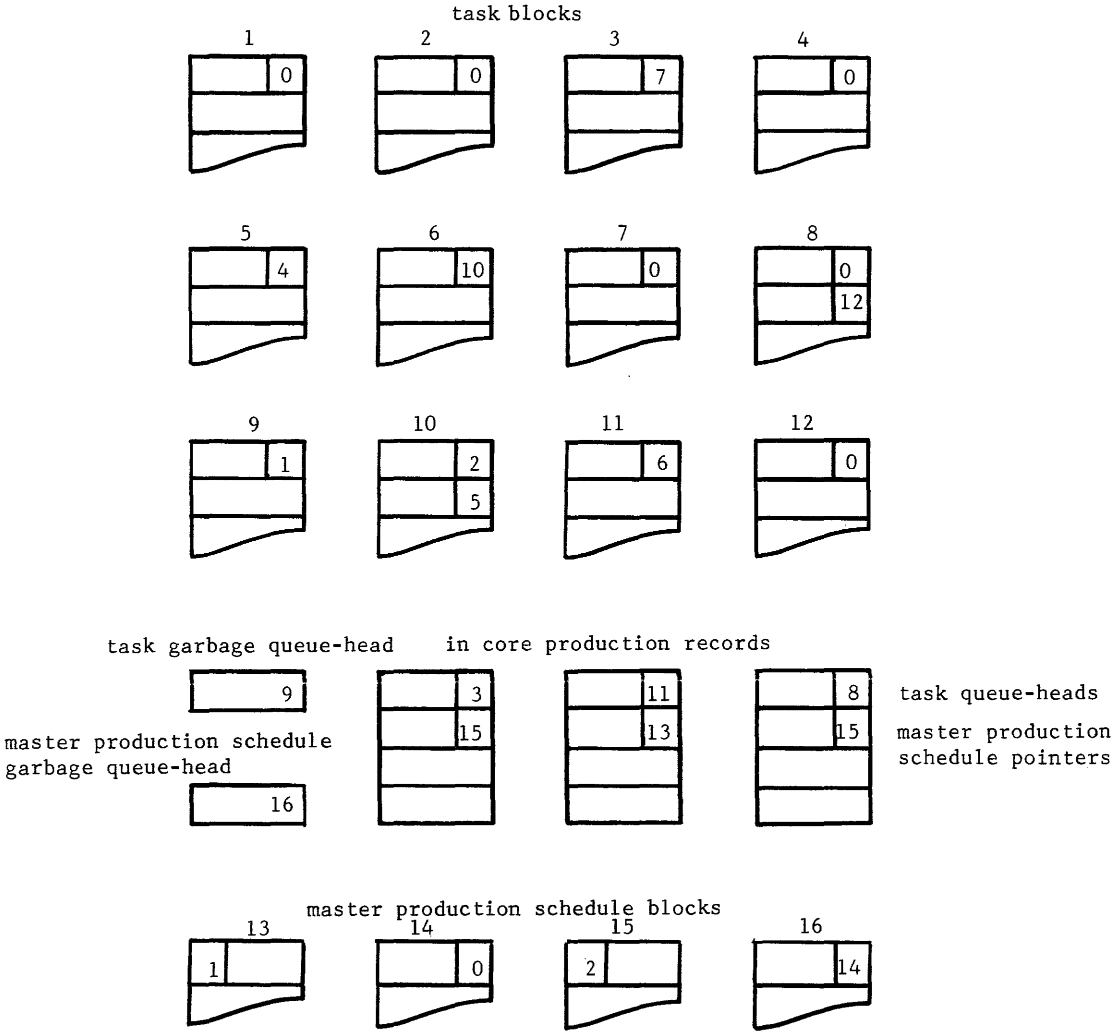


Figure 2 Queue Structures

# COMPUTER CONTROLLED TIRE FORCE ANALYSIS SYSTEM

Richard A. Cabot
S. Sterling Company
Southfield, Michigan

## ABSTRACT

A machine control system has been developed for a 4K PDP-8 computer, model ASR-33 teletype, and DF32 disc file to control, analyze, and alter the force characteristics of passenger car tires. The system provides many of the characteristics of large scale time-sharing systems while avoiding much of the programming complexity required in such systems. The control features of the system include the ability to execute operator initiated "macro" commands in an online mode or in a batch mode. The analysis features of the system range from storage of digitized data records on the DF32 disc file to the computation of Fourier coefficients using the recently developed Fast Fourier Transform techniques.

## INTRODUCTION

When I first became involved with this system, I was overwhelmed with the feeling that I was unknowingly taking part in a huge practical joke. The S. Sterling Company was seriously responding to a request from the General Tire and Rubber Company for a system that would grind rubber from the edges of brand new passenger car tires! As a car owner this was exactly what I diligently tried to avoid (rotation, balancing, etc.) and yet General Tire was intending to grind rubber off tires before I even got a chance to buy them.

It soon became clear, however, that the process of tire grinding, called tire correction by the industry, is an ingenious method developed over the past several years by test engineers at General Tire to alter the force characteristics exhibited by a tire when placed under a load equivalent to a passenger car. By removing a few thousandths of an inch of rubber from specific spots on the circumference of the tire, its ride is significantly improved. It is natural to wonder why these spots, or "force bumps," are not avoided during the production of the tire, thus eliminating the need to carry out post-production tire correction. The reason lies in the literally hundreds of variables involved in the tire molding process. Understanding the myriad of interrelationships between these variables, let alone controlling for their effects on the final force characteristics of the tire, is currently impossible. Although the process of tire correction had been shown to be conceptually sound through a previous, strictly analog machine, there were still many research questions to be answered. It was also necessary to test the practicality of correcting tires in production quantities. Thus the S. Sterling Company was confronted with the problem of developing the electronic hardware and programs to control a tire correction machine to be operated in both a research and production mode by test engineers with no prior real-time computer experience.

## HARDWARE

Figure 1 provides an overall view of the hardware involved in the system. The electronics cabinets contain the following equipment:

| | |
|---|---|
| 1 | Digitec digital voltmeter (slide wire type) |
| 5 | Dana amplifiers |
| 5 | BLH signal conditioners |
| 8 | Datronic signal conditioners |
| 9 | Moog differential amplifiers (servo controllers) |
| 14 | Strip chart recorder outputs |
| 1 | PDP-8 computer with 4K core, extended arithmetic element, memory parity, and power failure options |
| 1 | DF32 disc file with one surface |
| 1 | Texas Instrument 16 channel A/D converter-multiplexer |
| 1 | Operator's control panel with a dial indicating current RPM, switches indicating rotation direction, tire size, and whether the tire is mounted or unmounted. |

The axle of the load drum contains six load cells (three at each end) that provide the means of measuring the various major forces of a tire. When the load drum is brought in against a tire, the load cells produce outputs equivalent, when summed in pairs, to the radial, lateral, and tangential forces exhibited by the tire. The summing is carried out in analog form, at the load cell, for the radial and lateral forces and in digital form, in the computer, for the tangential force. All resulting force signals are conditioned and amplified by the BLH signal conditioners and the Dana amplifiers before being brought to the A/D converter as input for the computer. The force signals are also available to the digital voltmeter as a visual aid for the operator during calibration checking. Once in digital form the force signals do not require further conversion (other than the summing of tangential) since they have been calibrated so that one binary digit represents one pound of force.

The grinder carriages require the greatest amount of control of any component of the system. The two carriages are mirror images of each other. During the actual grinding of a tire, the carriages move laterally across the face of the machine to a position over the outer edges of the tire. In order to grind a tire with regard to its force characteristics, it is first necessary to keep the grinder wheels (see Figure 2) a small constant distance from, and directly over, the edges of the tire. It is important to realize, at this point, that the physical roundness (called runout) of a tire is not directly related to its internal force characteristics. It is possible, therefore, for a tire to be perfectly round and yet have an extremely variable force trace. The converse is also true as long as the tire runout is within industry quality control limits. Thus, maintaining a small constant distance between the grinder wheels and the tire edges first requires continual monitoring of both lateral and radial tire runout. The electronics used to perform this function include Datronic signal conditioners and Moog differential amplifiers that receive input from two LVDT's mounted on each of the runout probes shown in Figure 2. The lateral changes in runout are known to be gradual throughout a tire revolution and thus are monitored in closed loop, causing the corresponding lateral movement of the particular grinder carriage via two Moog servo valves. The radial changes in runout require more precise control due both to the abruptness of the changes and also the physical separation of the runout probes and the actual grinder wheels. As in the lateral control, the radial runout LVDT's also produces a signal that ultimately controls the Moog servo valves positioning the particular runout probes, but now two additional signals are required to be received by the computer, delayed in time, and used to position the actual grinder wheels. These additional signals are provided by two potentiometers, one for each carriage, whose case is attached to the given grinder carriage and shaft attached to the runout probe. Thus, when the probes change position corresponding to a change in radial runout, the potentiometer readings change. The computer receives the runout potentiometer output from both the left and right grinder carriages via the A/D converter.

The positioning of the grinder wheels is accomplished by the two slides shown in Figure 3. The reader should also observe from this figure that the runout probe is extended to the edge of the tire (the left grinder carriage was disconnected for the photograph). The readings received from the runout potentiometers are delayed in the computer and sent to the runout slides via two D/A converters whose values ultimately control two Moog servo valves (one for the runout slide on each grinder carriage). Since the force slides ride on top of the runout slides, a change in the position of the runout slides causes, by physical necessity, a corresponding change in the relative position of the force slides. Thus, by sending delayed runout values from the computer to the runout slides, the grinder wheels will follow radial runout at a known, small, constant distance from the tire. At this point the only components of the grinder carriages that have not been actively affected are the force slides. These slides, which, due to their physical position on top of the runout slides move with tire runout, are used to drive the grinder wheels into the tire at the points exhibiting force "bumps." The reader should recall that the force characteristics of the tire are captured by the load cells within the axle of the load drum (180° from the grinder wheels). Thus, the readings from the force axis to be used for grinding are delayed in the computer, modified to represent grinder depth, and then sent to the force slides via two D/A converters. The delay for both the runout values and the force values is kept independent of RPM by using the 128 tooth gear clock attached to the motor drive shaft (see Figure 3).

The decision as to which force axis to use in grinding, the criteria for choosing the points to be ground, and the number of revolutions to attempt grinding a single tire, are specified by the test engineer in a parameter table described in the following section.

## PROGRAMMING

### General

From the previous section it is clear that the mechanical complexity of the system alone warrants a fair amount of control work from the computer. More importantly, however, since the machine was to be used in both a production mode and a research mode, the software had to allow the "hands-on" flexibility required in the research environment and also the speed and simplicity-of-operation required in the production environment. And, finally, both these objectives had to be accomplished in a manner that would allow system operation to be carried out by test engineers with no prior real-time computer experience. In order to achieve these goals a good deal of time was initially spent (away from the computer) developing the basic system structure.

The structure evolved from picturing the total task of measuring and correcting a tire as a set of linked subtasks. Each subtask consists of a logically complete unit, such as fetching a tire from the conveyor and loading it in the machine, recording data from the tire, correcting the tire, performing a Fourier Transform, saving data on the memory disc, entering operating parameters, etc. The subtasks were developed as separate programs whose execution can be commanded by the test engineer via the system teletype. The major benefit of this approach is that the order in which the subtasks are linked is now at the operator's discretion. Obviously, certain sequences of commands are impossible to execute, such as attempting to correct a tire before a tire has been brought into the machine, or attempting to list data before data has been recorded. This problem is automatically handled by the system, however, by monitoring each of the operator's command requests and typing an error message if a command is requested that requires information not yet obtained.

Another benefit of this approach results from the autonomous nature of each subtask. Since each subtask (command) is a separate program, new commands can be added to the system with minimum difficulty. Thus, as the testing methodology expands, the system can also expand to incorporate the new techniques.

Although the decision to subdivide the total task into linked subtasks provides maximum operating flexibility, it also carries with it the inherent disadvantage of requiring the operator to physically enter each command. When the system is used in a production mode versus a research mode, such a disadvantage becomes overpowering. This problem, however, was foreseen during system development and avoided by allowing the operator to compile a sequence of up to 16 commands to be executed whenever required. Additional commands were added to provide operator control over the sequence, e.g., begin using the sequence, pause during the use of the sequence, restart the sequence, etc. During execution of such a sequence the operator may still define commands, and thus, in effect, cause the temporary insertion of a command in a previously defined sequence.

A command is entered via the teletype keyboard by typing a colon(:) followed by a command message. Since the system only uses the first two leters of the command message to determine what action has been requested, the operator may abbreviate commands to two letters or may type a complete message. A command is considered to be defined when the return key on the teletype is depressed. The entry of a command may occur at any time during system operation. If a command is being executed when a new command is entered, the system will, if possible, execute both commands concurrently. If, due to logical or space constraints, it is not possible to execute both commands concurrently, the command presently in execution will be completed before the new command is carried out.

There are currently 25 commands available to the operator of the system, many of which are shown in Figure 4. This figure also demonstrates the compilation of a run sequence.

The system was organized in a way that attempted to minimize the dead time between the execution of commands. Clearly, when the operator is manually entering commands that are to be executed as soon as their definition is complete, (not using a previously compiled run sequence), the dead time between commands is irrelevant. However, when operating in a production mode, with commands being taken from a run sequence, the more rapidly the commands are executed the higher the production rate. The techniques used to accomplish this objective are described in the following section.

### Internal Structure

The memory of the computer was divided into two segments: a resident segment and a background segment. The resident segment consists of the system monitor, (including a disc referencing routine and the table containing any compiled commands), an interrupt processor, a circular buffer for all teletype output, a teletype output processor, and a common region containing the parameters required by various commands during their execution. The background segment is continually overlayed, from the disc, with the programs corresponding to the commands requested by the operator. All output generated is placed in the output buffer with each location containing two trimmed ASCII characters. It should be noted that it is necessary for the actual trimmed characters to be placed in the table, versus, in the case of a title, a pointer to a string of characters located within the command program being executed, since the command program is very likely to be overlayed before its output is completed. In order to avoid an overburdening of the disc transfer capability, the system was designed so that the occurrence of an interrupt from the tire correction machine never requires a disc transfer for its processing. Thus the command programs that examine the sequence of interrupts to, for example, bring a tire from the conveyor and load it in the machine, are also permanently located in the resident segment. Since these programs respond to interrupts generated by a relatively slow moving mechanical process, most of the time these programs are in execution is spent waiting for the next interrupt. In order to take advantage of this, the system was designed to allow the execution of other command programs while waiting for the mechanical operations to be completed. Steps 11-14 of the run sequence of Figure 4 show an example of how to take advantage of this feature. Since the great majority of command programs require a disc transfer, a significant amount of design time was spent in developing the technique to initially store and subsequently retrieve information from the disc. For various reasons, the command programs are generally not stored in a single sequential block of memory, but rather in several noncontiguous blocks. The number of words in each block is never the same within a given command program and seldom the same across the various command programs. Finally, the various command programs generally do not begin at the same starting address. The above constraints lead to a design in which the disc image of each command program contains its own loading information. The first two words of the image indicate the number of blocks and the starting address. Each of the subsequent blocks is preceded by a count of the number of words in the block and its loading address. Since each of the control words must be processed by the computer before the information to follow can be loaded, the storing of the entire disc image in sequential disc locations would cause several needless disc revolutions to pass during the transmission of the entire image. To avoid this problem, each group of control words and each program block is padded with several trailing blank words. The blank words are not transmitted from the disc to core, but rather serve as time gaps during the transmission, to allow the disc routines to carry out the required control processing.

## RESULTS

Although the system has not been in existence long enough to effectively evaluate its contribution as a research device, there are certain observations that can be made. The test engineers have found system operation easily learned. With a single 15 minute practice session they are prepared to attack the problems confronting them in their research. The feature of the system that they felt best contributed to their relative ease in learning was how the system apparently adjusted itself to the operator's level of sophistication. Initially the test engineers would type complete command messages but soon began abbreviating commands to the first two letters. They would also request for the printing of every bit of data available from a tire versus those results that were of actual concern. As they became more enthusiastic in the use of commands they also began making more logical errors, e.g., requesting the printout of the results of a calculation before requesting its computation. Since the system "looks" for this type of error, and, when discovered, prints an error comment, the operator tended to be irritated at himself versus bothered by the system interference.

Of course, all of this is strongly colored by the results produced by the system. If tires were not being corrected, no amount of flexibility of operation could make up for it. Figure 5 shows the results of grinding an 8.25 x 14 tubeless two-ply passenger car tire. The relative increase in high frequency components after grinding has been attributed to the grinder wheels currently being used. The main initial objective, however, is to reduce the difference between the maximum point and minimum point (called peak-to-peak). The data shown in Figure 5 reflects a reduction in peak-to-peak from 37 pounds to 14 pounds.

## ACKNOWLEDGMENTS

There were two other members of the S. Sterling Company directly involved with this system. Mr. John Gentelia designed and built the computer interface. After finding and repairing the minor errors in his own work he spent many invaluable hours consulting for the designers of the other portions of the system. Mr. Thomas Wilkins worked with me on the design and coding of the programs. Without his enthusiasm and willingness to work limitless hours, the system would still be in coding phase.

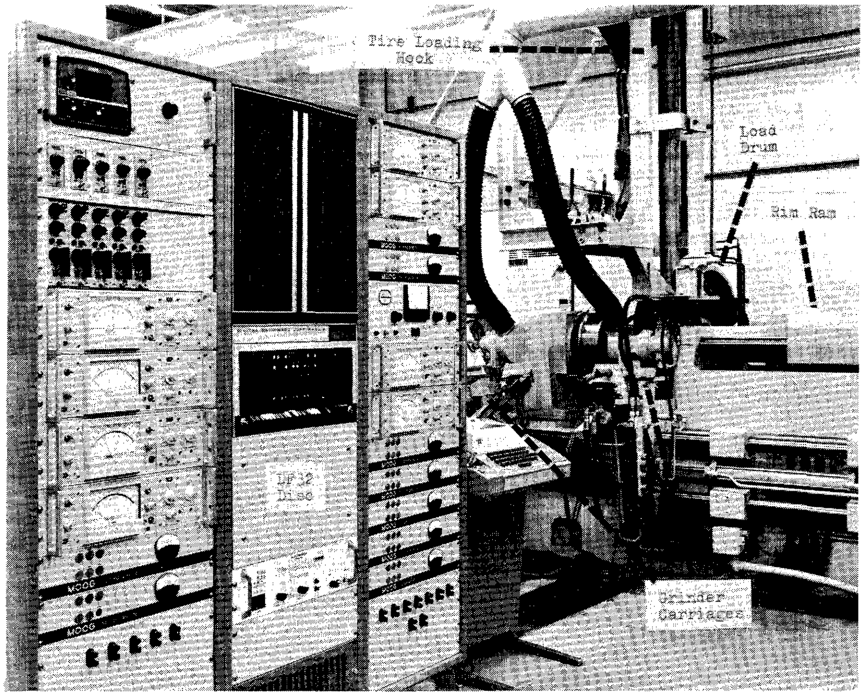


FIGURE 1 GENERAL TIRE PHOTO 2 2 8 1 3

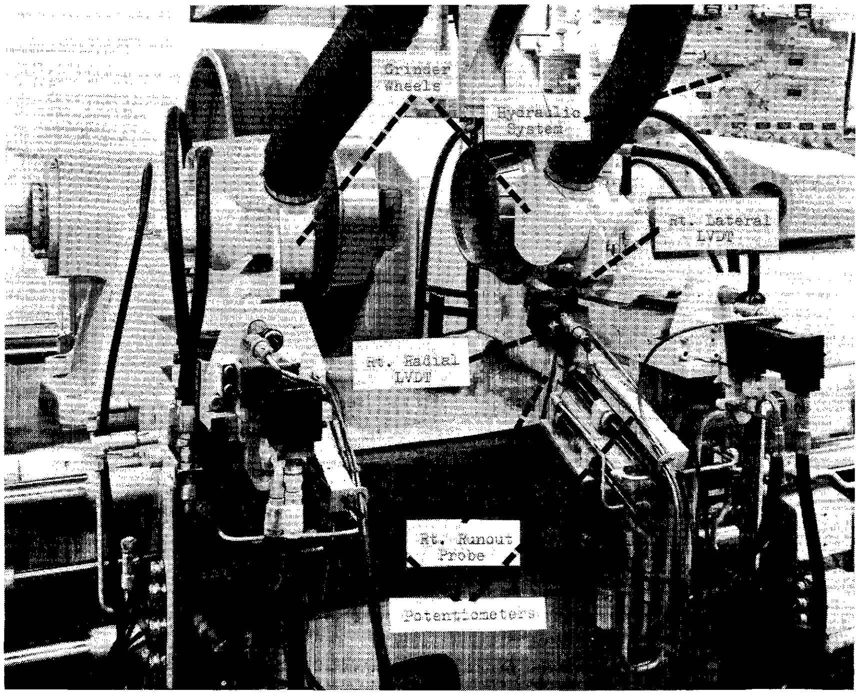


FIGURE 2 GENERAL TIRE PHOTO 2 2 8 1 3

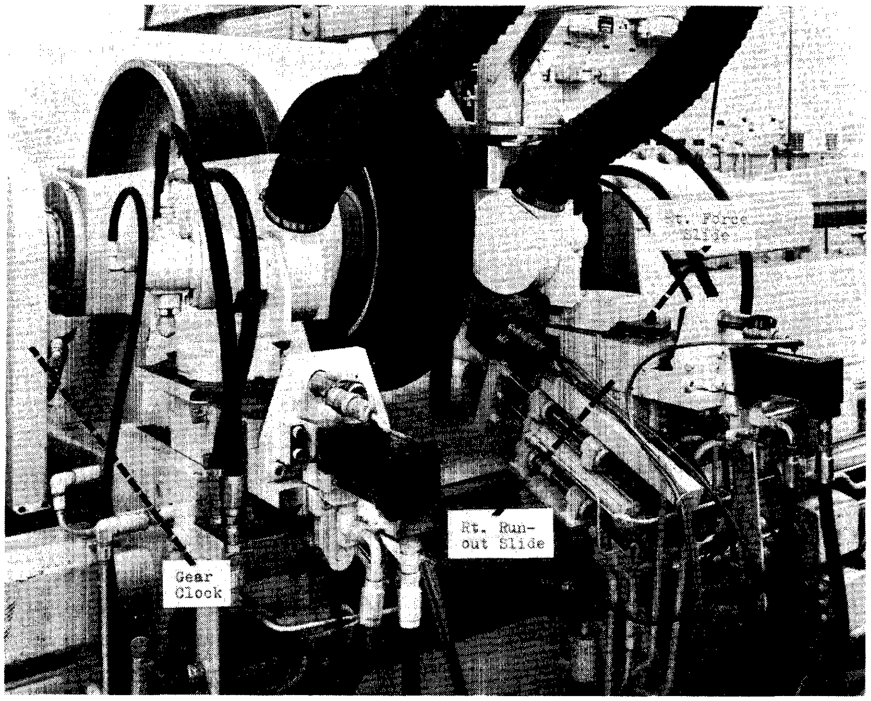


FIGURE 3 GENERAL TIRE PHOTO 2 2 8 1 3

```
:DEFINE
       REGION= RUN SEQ.
       START= 1

 STEP COMMAND
   1  :GET A TIRE FROM THE CONVEYOR
   2  :WARM-UP THE TIRE
   3  :RECORD UNIFORMITY DATA
   4  :GO TO SPECIAL HANDLING SECTION IF THE TIRE IS REJECTED,IE.
      STEP=15
   5  :BASIC STATISTICS FOR THE DATA COLLECTED
   6  :LIST THE INFORMATION IN
      REGION= DATA
      START= 1
   7  :CORRECT THE TIRE
   8  :...(CONTINUE)--IGNORING THE PROBLEM OF AN "UNCORRECTABLE"TIRE
   9  :RECORD DATA AFTER ATTEMPTING TO CORRECT
  10  :GO TO SPECIAL HANDLING SECTION IF THE TIRE IS REJECTED,IE.
      STEP=15
  11  :GET THE NEXT TIRE
  12  :BASIC STATISTICS FOR THE CURRENT TIRE
  13  :LIST THE POST-CORRECTION DATA IN
      REGION= DATA
      START= 1
  14  :GO BACK INTO THE PROCESSING LOOP STARTING AT
      STEP=2
  15  :SAVE DATA BROM THE NONUNIFORM TIRE ON THE DISC
  16  :GO BACK,BUT SKIP THE CORRECTION OF THE TIRE,IE.
      STEP=1
 17...NO MORE
```

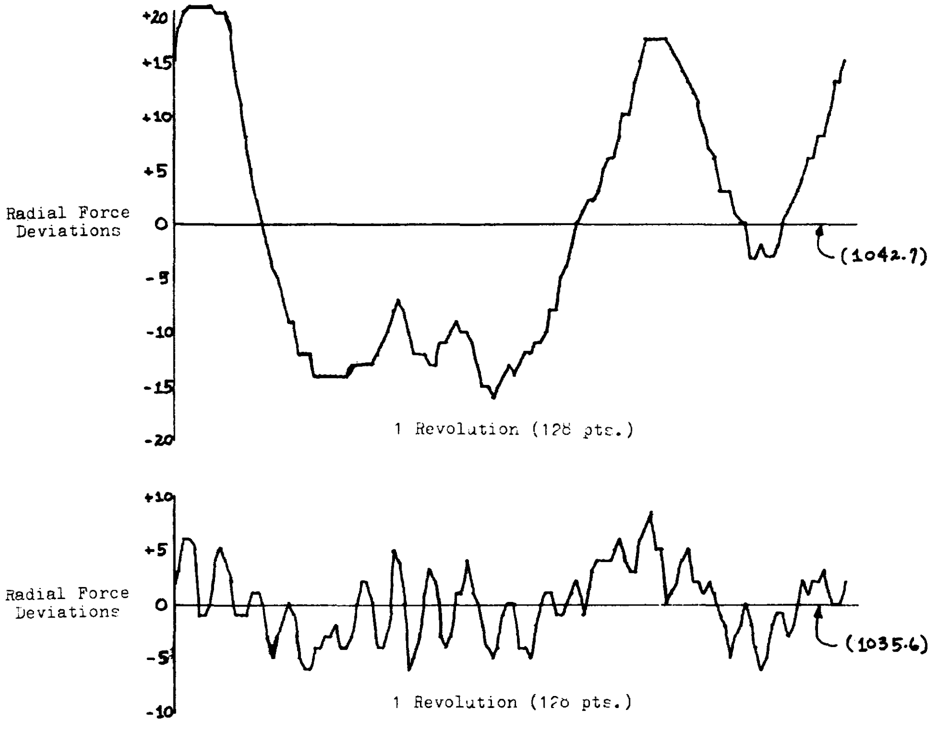

+20
+15
+10
+5
Radial Force Deviations
0
-5
-10
-15
-20
1 Revolution (128 pts.)
(1042.7)
+10
+5
Radial Force Deviations
0
-5
-10
1 Revolution (128 pts.)
(1035.6)

FIGURE 5

# INTERACTIVE SYSTEMS

# A SEISMIC DATA ANALYSIS CONSOLE

P. L. Fleck
Lincoln Laboratory,* Massachusetts Institute of Technology
Lexington, Massachusetts

## ABSTRACT

A software system for a PDP-7 digital computer with a 340 display has been designed to process seismic data. The system permits quick visual inspection of digitized data and allows easy application of powerful programs which operate on the digitized data or on the results of previously used programs. Some operations which can be performed are: epicenter location, beamforming, magnitude, complexity and spectral ratio computation, filtering, autocorrelation, Fourier transformation, sonogram generation and automatic event detection. A human operator is in the processing loop, inspecting the output at each step before applying the next. This system has greatly increased the speed and efficiency of much of our seismic data processing.

## GENERAL

There is a trend for data acquisition and other field systems to use digital magnetic tape as the recording medium in lieu of analog devices such as chart, film or analog tape recorders. This trend is likely to increase as cheaper and more reliable digital tape recorders become available and processing by digital computers becomes more popular. Digital tape recording has some large advantages over most analog recordings, viz. multichannel capability, precise timing, increased dynamic range, improved accuracy,** and compatibility with digital computer input devices. Experimenters are reluctant to use digital systems, however, because of the increased cost and complexity concomitant with these systems and also because it is difficult to visually examine the data. This paper describes a system which not only circumvents the latter difficulty, but makes standard data processing schemes that require the full power of a digital computer easy to use, apply, and interpret.

Our particular hardware configuration is shown in Fig. 1a. It consists of two identical PDP-7s, each with a 340 display, 32 K of core memory, an EAE, and an API. They share a three million word drum and six 570 seven-track IBM compatible tape drives. One of the PDP-7s can be connected to the M.I.T. Computation Center via a 40.8 kilobaud phone link. The software system described in this paper is self-contained in each PDP-7 and does not use the link to the 360 system. That is, we have, in effect, two data analysis consoles that can be run simultaneously.

The Lincoln Data Analysis Console is a computer-oriented system which is designed to analyze seismic data from the Large Aperture Seismic Array (LASA).[1] (It can be used to analyze any data which has been digitized onto magnetic tape in any one of the several LASA formats.)[2] This system is designed to do many of the standard data processing tasks more efficiently and accurately, and yet be quicker and easier to use than a "batch processing" computer of the type that is found in most modern data centers. The system displays the input data or the computer output on the 340 quickly and in an easily interpretable manner to a human data analyst who can alter the data or give new commands to the computer via a fiber optics light pen, adjustment knobs, or a teletypewriter keyboard (see Fig. 1b).

This man-machine system puts a man in the feedback loop to do pattern recognition and to make decisions based on his experience as a trained seismologist or geophysicist, and to do other tasks that are very difficult for a modern electronic computer. The computer is in the loop at a place where it can function with an efficiency equal to or better than that of a human.

One of the difficulties of processing geophysical data in large computing centers arises from the large amounts of both input data that is required and output data that is produced. With data from a large seismic array such as the Montana LASA, the input might typically be a reel of magnetic tape and the physical output might be 25 feet of 32-channel chart recording, of which only 25 constants are useful output (which may be the input, along with the original data tape, to the next step in the processing). Each iteration or pass through the computing center takes time, generates a lot of superfluous data,

* Operated with support from the U. S. Advanced Research Projects Agency.

** Off-the-shelf digital systems claim over 80 db dynamic range with 0.01% accuracy, while it is difficult to get more than 60 db and 0.1%, respectively, from analog recordings. It is interesting to note that the dynamic range and accuracy of digital systems are not limited by the recording method per se, but by the analog-to-digital converting processes.

and can introduce error in each two-way digital-to-analog and analog-to-digital conversion process (e.g. errors in reading the chart recorder and transcribing to punched cards).

The Lincoln Data Analysis Console is designed to eliminate these difficulties by presenting to the operator quickly and simply only that data which he desires to see. This data is presented in a quasi-analog manner, i.e., it appears analog to the eye and can be manipulated by analog devices (e.g. light pen and knobs), yet the full accuracy of the digital values is retained for computation because the data never leaves the computer. For example, the operator can easily scan the input data tape to insure it is not only good, but it is indeed the data he desires to have processed. Useless data is not put onto paper, but it is readily available for perusal if need be. Only after the data has been reduced, or after the operator decides he really wants it, is a permanent hard copy produced in the form of chart recordings, 'scope photographs, teletype copy or punched paper tape.

The system is organized so that the operator can have all the processing programs that have been written and debugged available literally at his fingertips. This ability to process the results of a prior processing without punching data cards and submitting for another pass through the computing center saves time. Typically, operations that take several days via other methods can be done in a few minutes on the analysis console.

## THE MONITOR SYSTEM

The operation and control of the console are done by a master program which is called the MONITOR system. This program always resides in the computer core memory and controls the calling and execution of all the console system programs which are stored on magnetic tape or a drum as a program file. The MONITOR program always knows which programs have been used and hence which physical parameters have already been determined. The MONITOR saves all these parameters so that further processing will have access to them. Any processing that tries to use parameters which have not yet been determined will not be executed by the MONITOR until they have been determined. The MONITOR program is the only program that must be manually read into the computer. This is done once at the start. A self-protect feature makes it impossible to be destroyed by any of the software.

All commands to the MONITOR are input to the computer via the teletypewriter keyboard in a one-character mnemonic code. The MONITOR will fetch the proper program from the program file, transfer it into the core memory and execute it All the programs are arranged so that they return to the MONITOR automatically. Control can be returned to the MONITOR prematurely by typing R (for return) on the keyboard.

## THE DISPLAY SYSTEM

The console display system is the basic program from which most of the other programs are called. It is designed to make digitized seismograms accessible to a seismologist in a quasi analog manner to which he is accustomed. This program displays the seismograms (earth motion vs time) on the 'scope with six knobs controlling the parameters of the display.

The six knobs control the horizontal and vertical gain of the seismograms, the vertical separation between the seismograms, the horizontal and vertical position of the seismograms, and a clipping level. Figure 2 shows some typical examples.

The operator can use this program to scan the data visually and pick a portion of any seismogram for further processing by other programs. This picking is done by pointing the light pen at any waveform which causes it to be saved in a "reference buffer." The contents of this reference buffer will be displayed at the top of the screen with a vertical cursor appearing at the exact point in the waveform that was touched by the light pen. This "reference" trace is unaffected by most of the knobs. Thus one can, for example, move all the traces, one by one, alongside this reference by adroit manipulation of the horizontal and vertical position knobs to do visual alignment and correlation. This process is used to pick arrival times for the various traces to be used for beamforming and location (see below). By proper adjustments of the gains, this time picking can easily be done with a precision of one sample (50 milliseconds when using short-period data) as illustrated in Fig. 3.

## THE CONSOLE PROGRAMS

Some of the operations that can be done with the analysis console will be briefly described.* The system is designed so that additional programs can easily be added when they are written and checked out.

### Initialize

This is usually the first program that is called and executed. It reads the data tape and determines which of the several allowable formats the tape has been recorded in. (Each site may have its own tape format.) The program types out this information, along with the date and starting time of the data tape. The operator can select which data channels he wants to operate upon (there may be up to several hundred channels recorded on the data tapes) and what sampling interval should be used. The sampling can be changed by instructing the computer not to use every sample that was recorded, but to skip a fixed number of data samples from each channel. This program initializes all the other programs to use these pre-selected parameters.

### Location

If three or more arrival times are picked, either by the Display program or by manually typing them in, the best fitting plane wave that minimizes the r.m.s. error will be calculated. From this, the horizontal phase velocity and azimuth will be calculated. In addition, the

* A detailed description of all the programs is available from the author on request.

relative arrival times when this plane wave would hit all 21 sites, along with the residuals from these times, is typed out. Provision is made for deleting times from any sites that may have large residuals and repeating the calculations (If the data is from short-period seismometers, station corrections will have automatically been inserted in this calculation, and the distance from the center of LASA, the latitude and longitude of the epicenter will be computed. Also the G.M.T. time at which the portion of the seismogram corresponding to the reference cursor must have originated will be calculated, assuming a focal depth of 33 km.) Two constants giving the major and minor axes of the location uncertainty ellipse are also typed out. Provision is made for manually deleting the station corrections. Figure 4 shows the output for the event shown in Figs. 2 and 3.

Beamforming

If one or more time picks are made via the Display program or the manual type-in program, the delay-and-sum beam will be formed and placed in the reference buffer as in Fig. 5. From here, it can be filtered (see below) or used as a reference to pick arrival times from weak events. Channels that have been deleted in the Locate program (above) will also be automatically deleted from the beam.

Constants

This program displays the reference waveform on the 'scope. Two points can be selected by the light pen and the G.M.T. time corresponding to each point will be displayed to the nearest 0.1 second. In addition, the time difference between the points and the amplitude difference in millimicrons corresponding to the two points picked (assuming normal calibration for the data) will be displayed. The magnitude based on these last two quantities will be typed out on request if the distance has already been computed. Provision is made for inputting the distance if it has not been computed. If the data is from a short-period instrument, the "Q" factors for a depth of 33 km are used. See Fig. 6.

Filtering

The data stored in the computer (21 seismograms, each 600 samples long) or the reference can be filtered. Currently, one has the choice of two filters which have been found useful in the past for short-period data.

Notch Filter This is a low distortion filter with two notches in the frequency response (at 0.2 and 0.3 Hz) to attenuate microseismic noise occurring around these frequencies. Zero frequency is attenuated about 10 db and frequencies higher than about 1 Hz pass through with little attenuation or phase shift.

Butterworth Filter This is a three-pole bandpass filter with 3 db points at 0.6 and 2.0 Hz. Both zero frequency and high frequencies (– 30 db at 5.0 Hz) are attenuated, but some phase distortion occurs.

Figure 7 shows some examples of the use of these filters.

Plot

This program plots all the waveform data stored in the computer on the Sanborn chart recorders so that hard copy can be obtained. This gives a much larger scaling than photographs of the 'scope can.

Transform

This program computes and displays the autocorrelation function of the reference waveform (512 data points), which has been modified as follows:

$$f_j^{MOD} = \begin{cases} 0 & j < A, \ j > B \\ f_j^{REF} & A \le j \le B \qquad j = 1, 512 \end{cases}$$

where A and B are set to any values via the light pen.

The finite Fourier transform of either the modified reference or of its autocorrelation function (both considered periodic with a period of 512 samples) can be computed. The autocorrelation function can be modified by an adjustable rectangular window. The periodogram of this transform is normalized and displayed (db versus frequency) and it can be output onto paper tape. Figure 8 shows some examples.

Complexity

This program computes and types out the complexity of the reference trace (beam or seismogram) starting at the cursor. The complexity is defined as follows:

$$C = \frac{\sum\limits_{j = t_c + 5 \text{ sec}}^{t_c + 35 \text{ sec}} (f_j - \bar{f})}{\sum\limits_{j = t_c}^{t_c + 5 \text{ sec}} (f_j - \bar{f})} .$$

where $t_c$ is the time of the cursor that was set by the light pen, and

$$\bar{f} = \frac{1}{20} \sum_{j = t_c - 20 \text{ sec}}^{t_c} f_j$$

Sonogram

A sonogram is a display of signal intensity (represented by several different levels of brightness) as a function of frequency (vertically) and time (horizontally). Figure 9b shows the sonogram of the test calibrate signal of Fig. 9a.

This sonogram uses a bank of 50 constant bandwidth filters, each with a (sin x/x)-type frequency response, with impulse response exactly 10 seconds in duration. The filters are spaced 0. 1 Hz apart, and they cover a range of 0. 1 to 5. 0 Hz. The input to these filters is the reference trace starting 1200 samples before the data displayed in the reference buffer and continuing for 3200 samples. The sonogram time scale, therefore, covers a range of 160 seconds if short-period data is used or over 2 hours if long-period data is used.

Each output of the 50 filters is full-wave-linear rectified to form the signal intensity,* and passed through two low-pass filters in cascade. The first is an energy summing "integrate and dump" filter. It sums the output of the rectifier for 20 samples,then resets itself to zero and repeats. The second filter takes these outputs, sampled just before the dump and performs "RC" type low-pass filtering. The 3 db cutoff of this latter filter is 0. 11 Hz. The computer stores the output of this bank of 50 "RC" filters in a two-dimensional array of power vs frequency and time. The sonogram is simply the pseudo three-dimensional display of this two-dimensional array.

If, for every time element in the sonogram array, the intensity at every frequency is summed, a one-dimensional "energy profile" array is formed. This program calculates and displays this energy profile as a series of 160 fine dots at the top of the display.

If, for every frequency element in the sonogram array, the intensity between two set times is summed, a one-dimensional "frequency spectrum" is formed. This program allows the two times to be manually set via the light pen. The spectrum is displayed as a series of 50 coarsely spaced dots superimposed upon the energy profile. Both these latter displays are automatically normalized to use all the room on the 'scope available to them.

The spectral ratio[3,4] can easily be calculated by summing the proper elements in the sonogram array and taking their ratio. This is done and the result is output on the teletypewriter.

## Automatic Event Detection

This program scans the data tape and types out the time of all teleseisms encountered. The actual mechanism by which teleseisms are distinguished from microseismic and other types of interference has been discussed elsewhere.[5,6] This operation takes place much faster than real time so that one can start with this program and quickly find all the events on the data tape, then analyze each separate event by using the programs described earlier.

## Convolutional Matched Filtering

This program cross correlates the reference trace with a matched filter that is especially designed to enhance the S/N of long-period dispersed surface waves.[7] The matched filter is a sinusoid with a linear frequency modulation (chirp). The matched filter has three parameters that can be input to the program via the teletypewriter. They are: the period at the start of the chirp; the period at the end of the chirp; and the length of the chirp. The maximum allowable length of the chirp matched filter is 800 samples which corresponds to 160 to 1920 seconds depending on the sampling chosen in the Initialize program. Enough of the input reference will be convolved with this chirp to give 512 samples of filtered output, which will appear in the reference buffer. Figure 10 shows a typical example of this type of filtering.

## REFERENCES

1. Green, P. E., R. A. Frosch and C. F. Romney, Principles of an Experimental Large Aperture Seismic Array (LASA), Proc. IEEE, 53, December 1965.

2. Briscoe, H. W., LASA Magnetic Tape Format, M.I.T. Lincoln Laboratory, 18 October 1965, private communication.

3. Seismic Discrimination, Semiannual Technical Summary Report, M.I.T. Lincoln Laboratory, 30 June 1967, pg. 5, DDC 637308.

4. Kelly, E. J., A Study of Two Short-Period Discriminants, M.I.T. Lincoln Laboratory Technical Note 1968-8, 12 February 1968, pg. 12, DDC 666701.

5. Briscoe, H. W. and P. L. Fleck, A Real-Time Computing System for LASA, Proc. SJCC, Boston, Mass., April 1966, pp. 221-228.

6. Seismic Discrimination, Semiannual Technical Summary Report, M.I.T. Lincoln Laboratory, 31 December 1965 (see Section III), DDC 630559.

7. Capon, J., R. J. Greenfield and R. T. Lacoss, Long-Period Signal Processing Results for Large Aperture Seismic Array, M.I.T. Lincoln Laboratory Technical Note 1967-50, 15 November 1967, DDC 663429.

* Intensity rather than power is computed because it makes less stringent requirements on the dynamic range of the internal computer calculations.

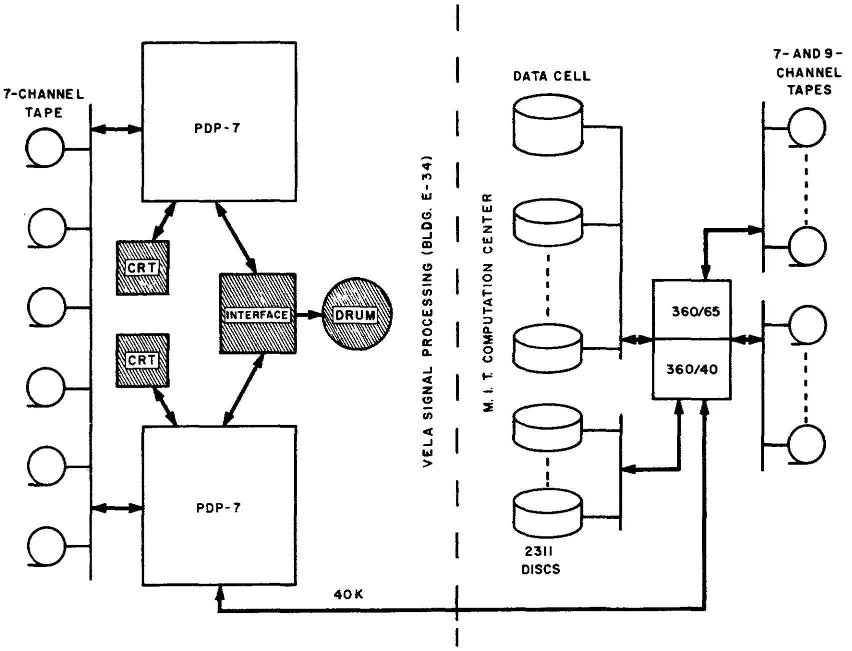


Figure 1a PDP-7 Computation Center Hardware

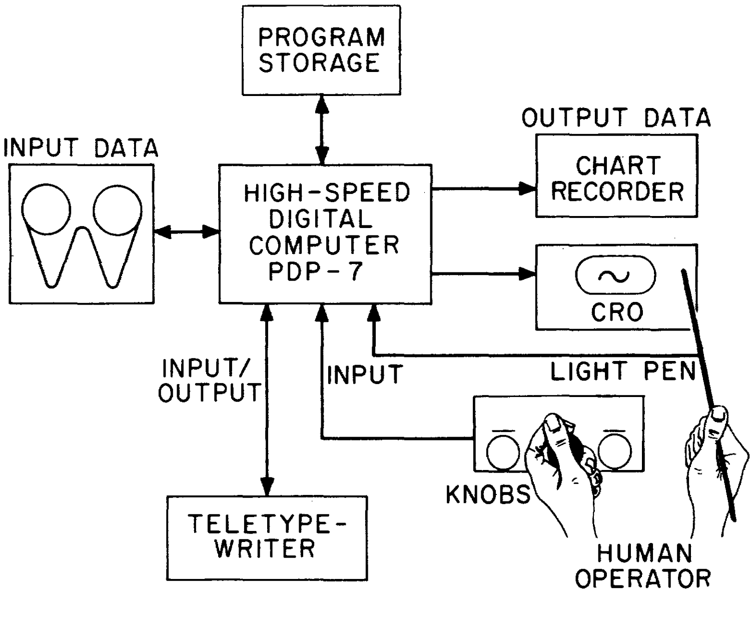


Figure 1b The Lincoln Data Analysis Console

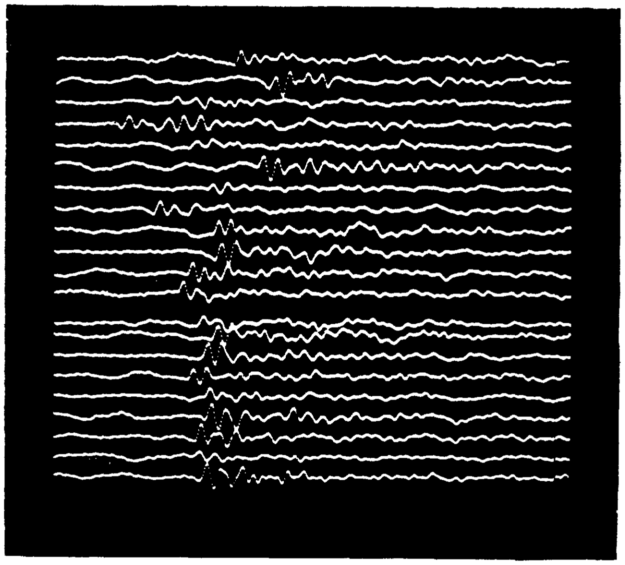


Figure 2a All 21 Seismograms

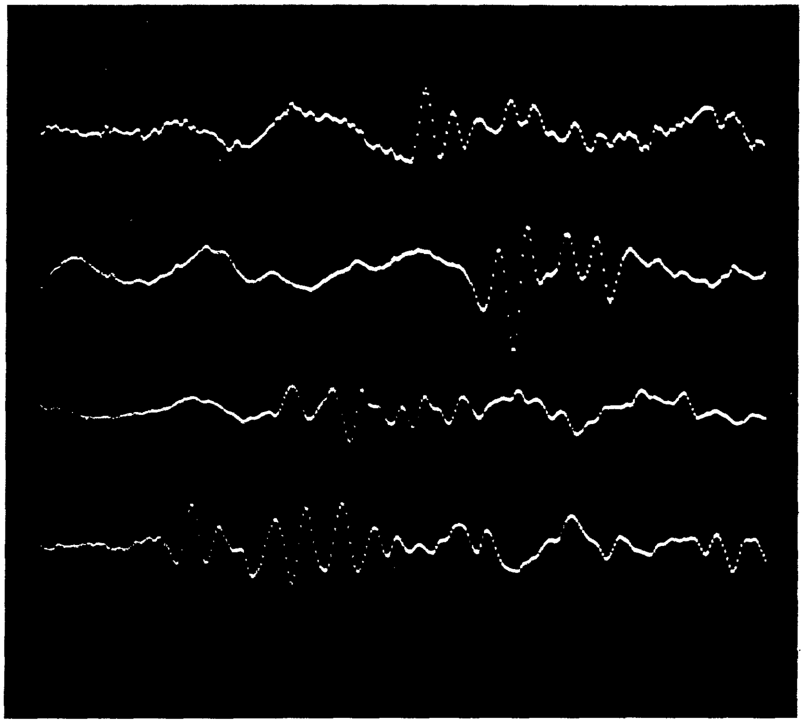


Figure 2b
The Top Four Seismograms with Greater Gain and Separation

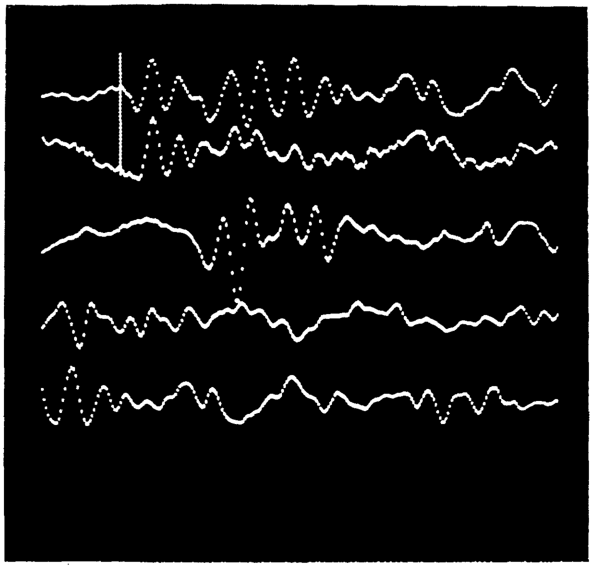


Figure 3a

Seismogram No. 4 Used as Reference and Aligned with No. 1

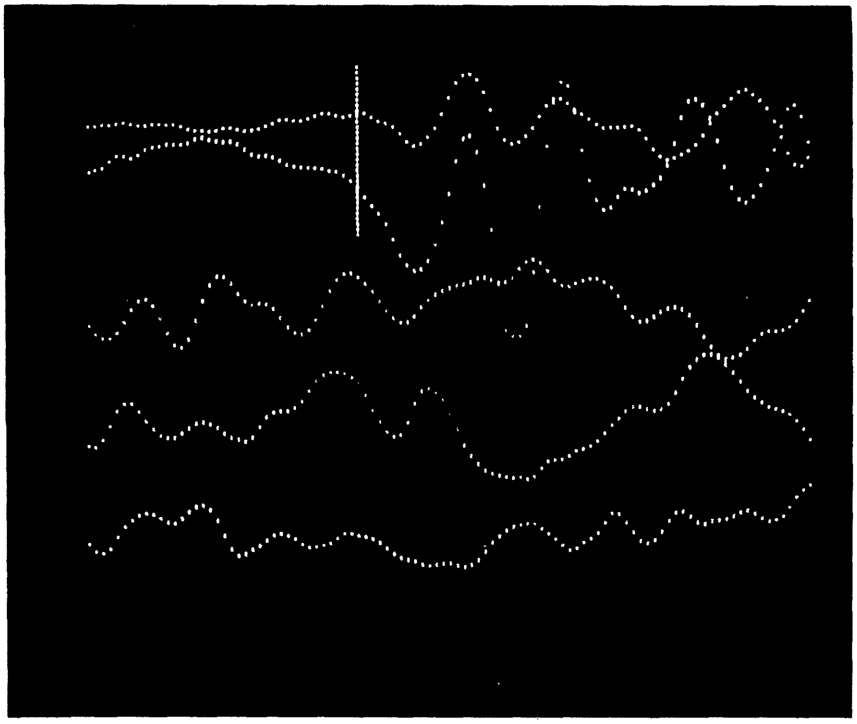
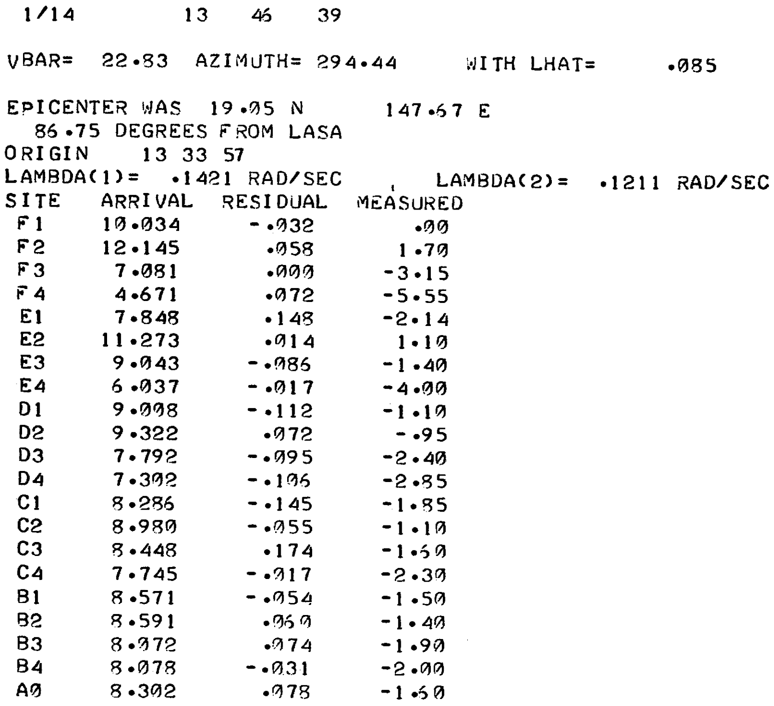


Figure 3b Same Except Aligned with No. 2 and More Gain



```
 1/14          13   46   39

VBAR=  22.83  AZIMUTH= 294.44      WITH LHAT=      .085

EPICENTER WAS  19.05 N       147.67 E
  86.75 DEGREES FROM LASA
ORIGIN     13 33 57
LAMBDA(1)=  .1421 RAD/SEC    ,    LAMBDA(2)=  .1211 RAD/SEC
SITE    ARRIVAL  RESIDUAL  MEASURED
 F1     10.034    -.032      .00
 F2     12.145     .058     1.70
 F3      7.081     .000    -3.15
 F4      4.671     .072    -5.55
 E1      7.848     .148    -2.14
 E2     11.273     .014     1.10
 E3      9.043    -.086    -1.40
 E4      6.037    -.017    -4.00
 D1      9.008    -.112    -1.10
 D2      9.322     .072     -.95
 D3      7.792    -.095    -2.40
 D4      7.302    -.106    -2.85
 C1      8.286    -.145    -1.85
 C2      8.980    -.055    -1.10
 C3      8.448     .174    -1.60
 C4      7.745    -.017    -2.30
 B1      8.571    -.054    -1.50
 B2      8.591     .060    -1.40
 B3      8.072     .074    -1.90
 B4      8.078    -.031    -2.00
 A0      8.302     .078    -1.60
```

Figure 4 Output of Locate Program

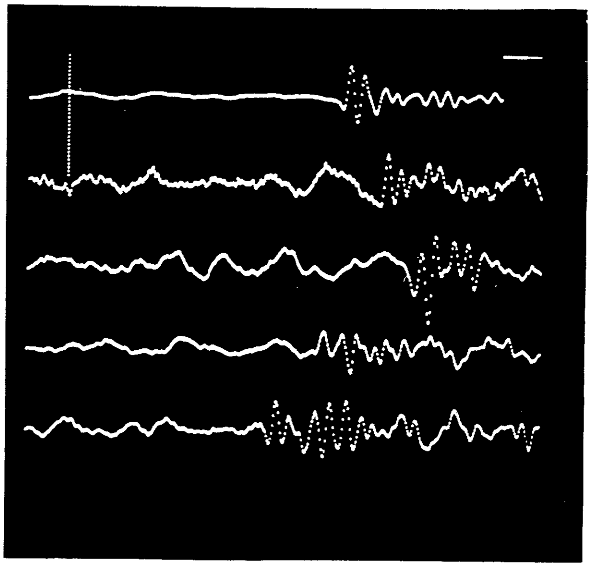
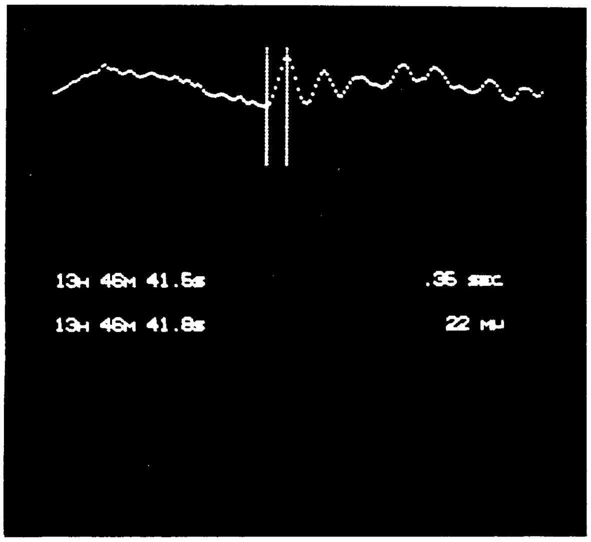


Figure 5 Beam Formed from All 24 Seismograms at Top. Notice Improved S/N



13H 46M 41.5s .35 SEC

13H 46M 41.8s 22 MU

Figure 6 Some constants for top seismogram of Fig. 2. Note that the vertical lines can be moved at will with the light pen. The teletypewriter typed MAG = 5.2.

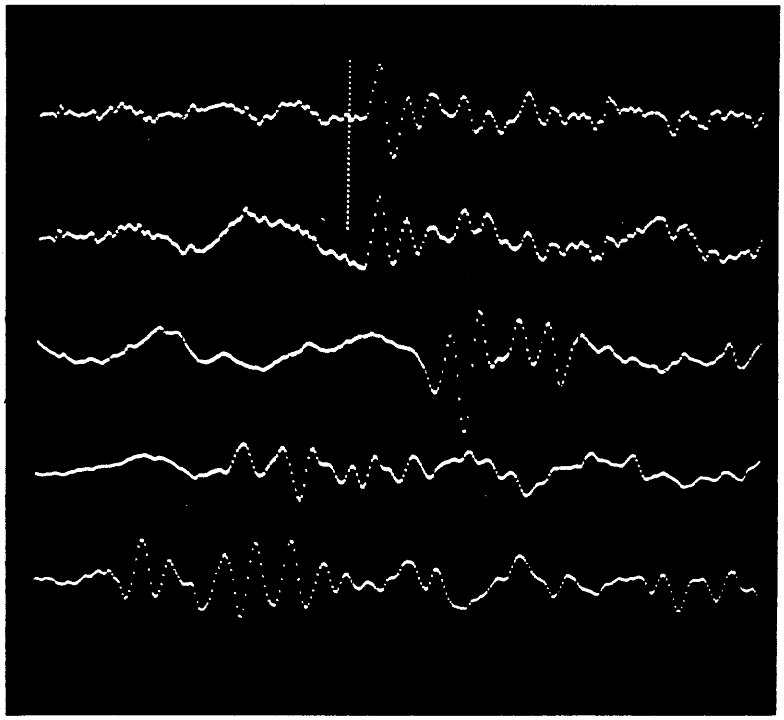


Figure 7a Top trace is notched filtered. Second trace: raw data.

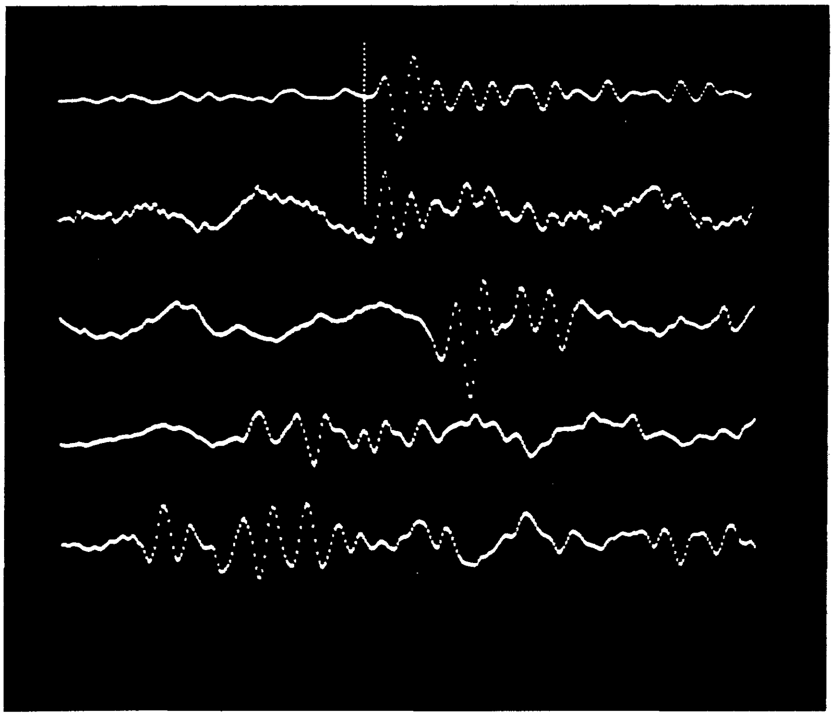


Figure 7b Top trace is Butterworth filtered. Second trace: raw data.

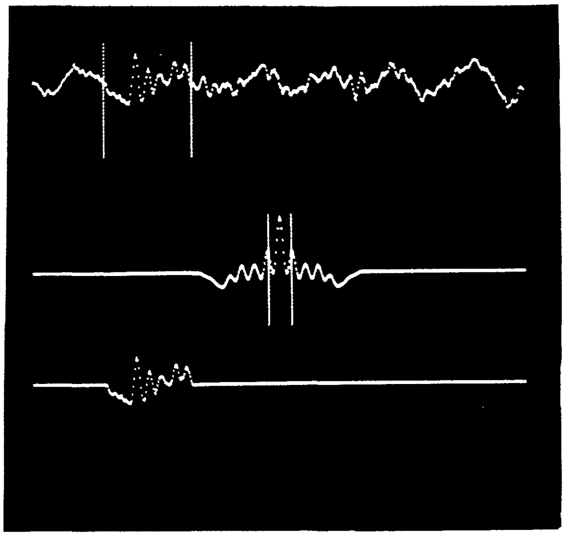


Figure 8a Top trace is input data. At the bottom, a certain portion is selected with the light pen. The middle trace is the autocorrelation function of the bottom trace.

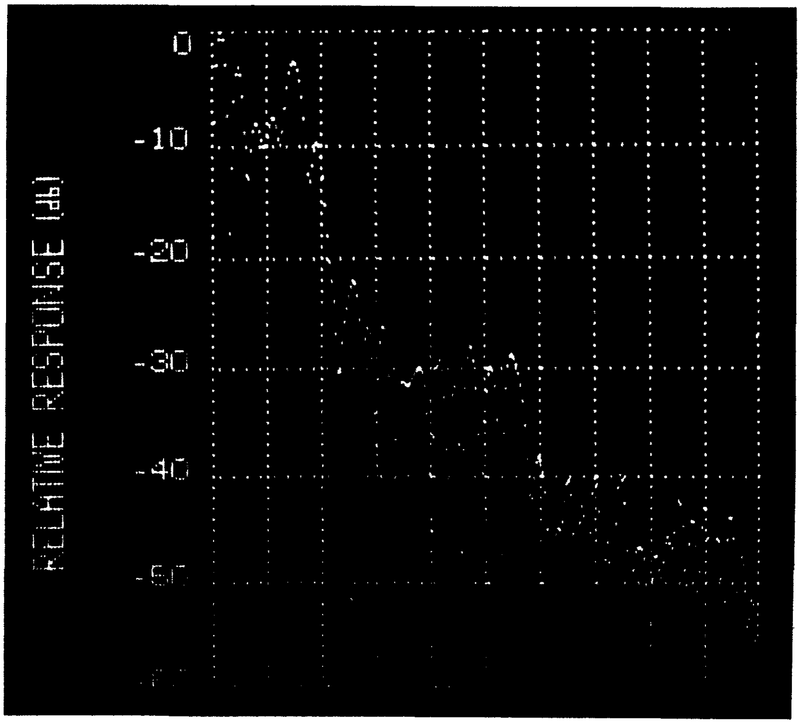


Figure 8b Periodogram of the bottom trace of Fig. 8a.

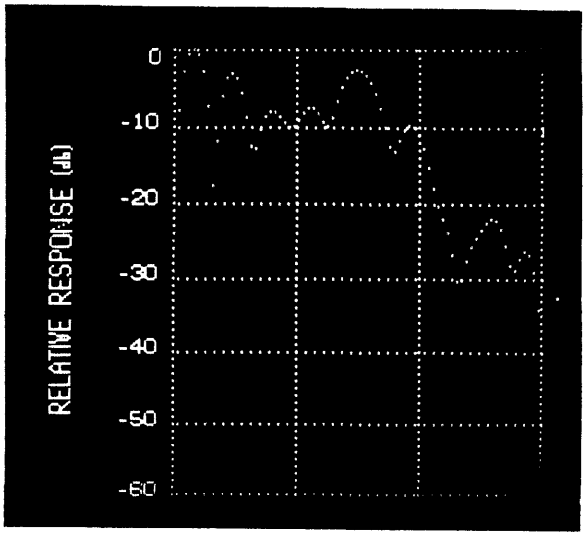


Figure 8c Same as Fig. 8b showing the first three Hz expanded.

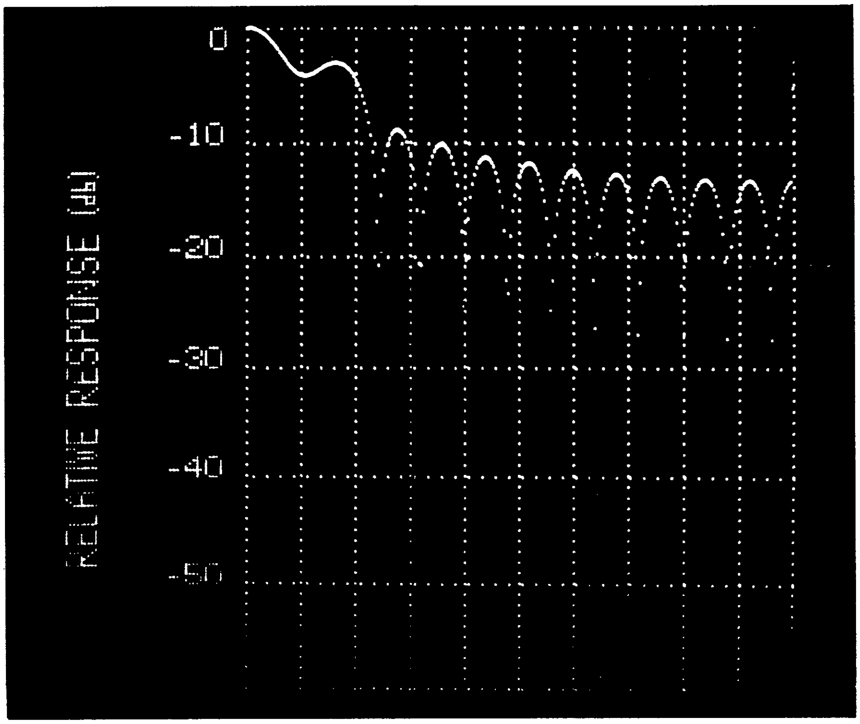


Figure 8d Periodogram of the autocorrelation function of Fig. 8a after it has been modified by a rectangular window (vertical lines in center portion of Fig. 8a).

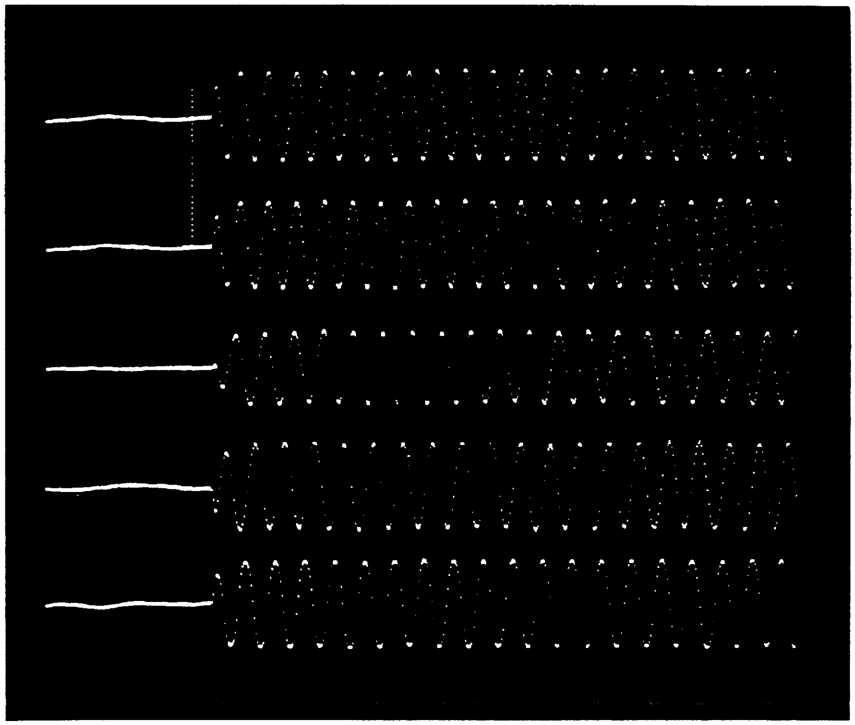


Figure 9a Calibrate signals on various channels.

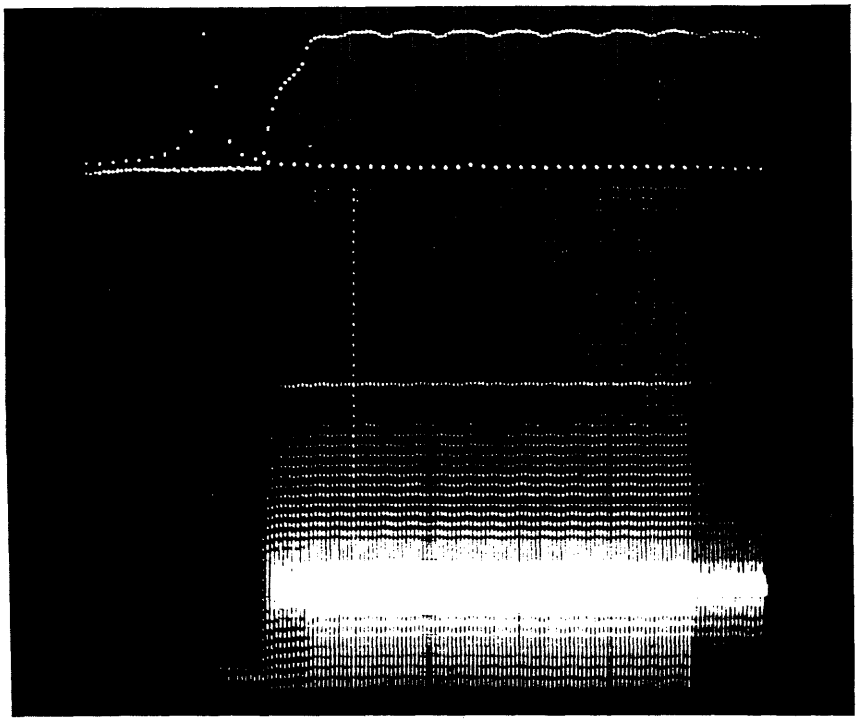


Figure 9b Sonogram of top trace of Fig. 9a. Note energy profile (fine dotted line) and spectrum (coarse dotted line) at the top. Note slight third harmonic distortion inadvertently present in the calibrate signal.

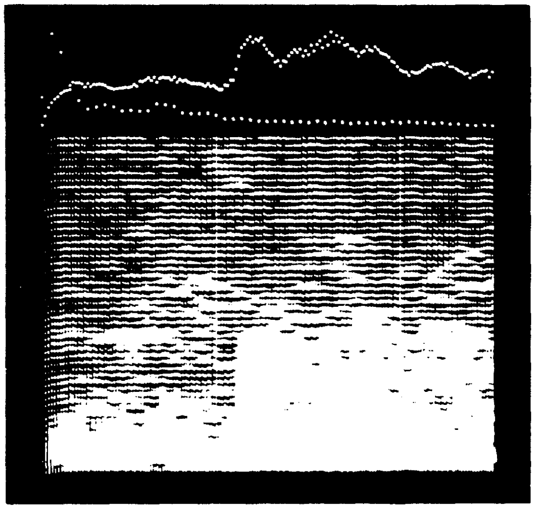


Figure 9c Sonogram of top trace of Fig. 2b.
Teletypewriter typed
SPECTRAL RATIO = 0.647.

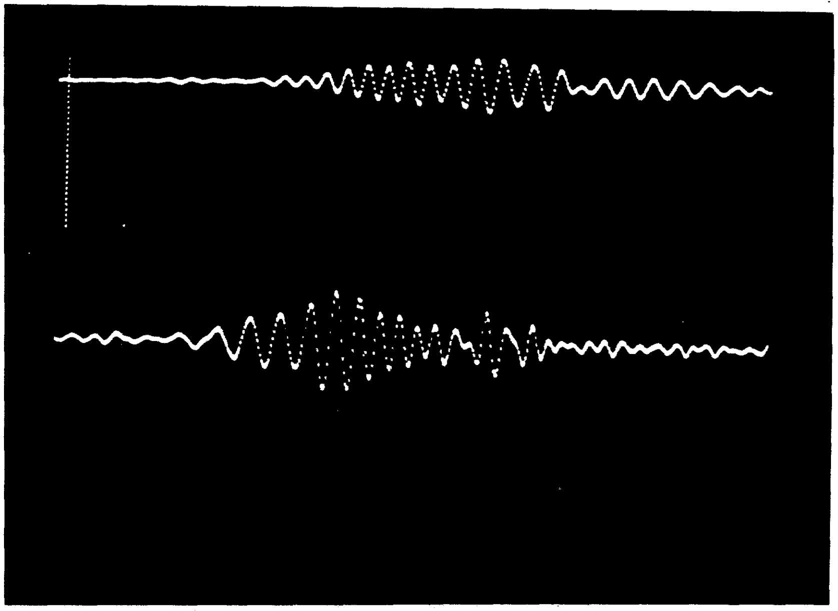


Figure 10 The top trace is the result of convolving the Rayleigh wave shown in the bottom trace with a chirp filter 400 seconds long with a starting period of 40 seconds, and an ending period of 16 seconds. The sampling interval is 1.2 seconds and the horizontal scale is 10.24 · minutes.

# COORDINATE MEASURING MACHINES AND COMPUTERS

Neale F. Koenig
Information Control Systems, Inc.
Ann Arbor, Michigan

## ABSTRACT

Digitizer applications of Coordinate Measuring Machines (CMM's) are directed toward the production of N/C tape for machining complex two and three-dimensional part configurations. This task is best performed in a man computer coalition, i.e., the man directs the CMM over the part and the computer performs the mathematical computations and translation of data to the desired tape format. In the immediate future, these tasks can be performed in a fully automatic mode whereby a surface sensing probe provides information that when processed by an interfaced computer, allow the computer/CMM to automatically digitize the part. The operator in this case takes the role of supervisor and only needs to define the limits of digitizing during the initialization phase.

In order to achieve the most cost effective (i.e., low cost, high effectivity) hardware system, a great deal of concern must be paid to the development of associated computer software. Thus, such techniques as foreground/background and priority interrupt processing must occur to effect total utilization of a small, inexpensive on-line computer interfaced to the CMM. Such a system has been developed for the production of N/C machine tapes to digitize turbine blades and is fully described in the text.

## A HISTORY OF COMPUTERIZED COORDINATE MEASURING MACHINES

Little has been done to date in producting N/C* tape from CMM's due to the general lack of adequate data processing techniques in converting the raw data as received from the measuring mechanism. This problem was due in part to the lack of a high speed integral processing system that could convert directly the input data to a finished N/C machine tool tape. With the advent of inexpensive, highly flexible digital computers, the mating of the CMM and a small digital computer essentially solved the problem of real-time N/C tape production. The first appearance of such a combination occurred about a year ago (1967) on a production basis. At that time, the applications took the expected turn of solving basic coordinate rotation and translation problems as well as numerous tolerance comparison problems. As progress would have it, attention was finally brought to the development of tape production. Several of these programs, which have been written during the past, have led to the production of tape to drive N/C machine tools. An example, which is included, represents a fairly sophisticated N/C tape production program. This program has as its main feature the ability to perform three axis contouring and thus, suggests CMM/Computer usage for producing N/C tapes for any complex three-dimensional surface which cannot be easily drafted in an engineering drawing.

The applications of programs of this sort are unlimited. Such programs are easily developed for car body design, inspection and model making. These applications can eventually be extended to the rough cutting of dies, thus effecting a great cost savings in automotive design and subsequent production.

Ship model development for production of test hulls represents another area where CMM/Computer combinations are valuable. Such things as scaling, pattern inversion, and other transformations are easy tasks for the computer to perform during data taking.

### The CMM Family Tree

Coordinate Measuring Machines have proven to be a valuable aid in the inspection of precision parts. These machines, in conjunction with general purpose digital computers, also provide the answer to direct production of N/C machine tool tapes. The inspection uses of CMM's include both manual and automated gaging of parts for hole locations, cam tolerance measurements and, in general, comparing production parts to the desired part configuration. These tasks can be accomplished with or without the aid of an interconnected computer. In the former case, the inspector performs the calculations from the optical or digital outputs of the CMM. In the latter case, the computer provides an output if an out of tolerance condition exists. Computers, in this role, allow for computer alignment of parts on the table, polar coordinate measurements and numerous other user oriented functions.

CMM's, when fully automated with feedback control systems, can be used to perform automatic inspection via the use of N/C machine tapes. These tasks can be either point to point inspections or continuous path with pre-selected inspection intervals. Once this type of inspection is utilized it becomes apparent that the CMM can be used for actual milling, provided that the material to be milled is relatively soft. The same techniques that are used to perform automatic inspection can be used in milling and point to point drilling, i.e., the input is obtained from a pre-punched N/C tape of the type used with conventional N/C milling machine tools.

*N/C - Numerical Control

Digitizer applications of CMM's are directed toward the production of an N/C tape (i.e., the production of a part). The CMM thus fills the role of N/C tape production from templates or a model of the desired part.

Again, with CMM's both computer aided and unaided techniques are used in digitizer applications:

1. Manually oriented system - the user performs all calculations for the eventual production of an N/C tape and uses an off-line tape punch machine (such as a Flexowriter) to produce the tape.

2. Computer assisted system - a more costly technique, due to required programming in which the computer performs all calculations and actually punches the tape to be used on the N/C controller. An added advantage in computer generated output lies in the elimination of mistakes encountered in manually performing the hundreds of calculations necessary for the final production of the N/C tape.

Figure 1 summarizes these many applications of CMM's both for N/C and general inspection tasks.

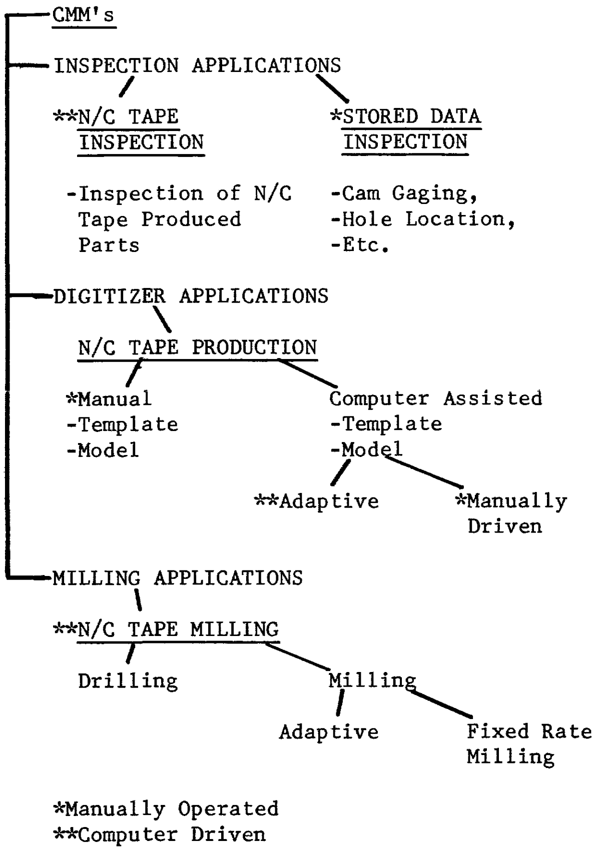


Figure 1. THE CMM TREE

Future systems include computer driven digitizer systems as shown in the tree. Such systems are already existent for inspection applications whereby the inspection machine is driven in much the same way as an N/C machine tool.

Digitizers - Manual vs. Computer Assisted

Producing the N/C tape via the manual method (see Fig. 2.) requires that the operator perform all calculations necessary to convert the X and Y information output from the CMM into the machine tool controller tape code. This code usually includes the following for each individual machine tool instruction:

N - the instruction sequence number

G - a code indicating the type of machine tool movement, i.e., circular, (CCW or CW), linear or perhaps parabolic

F - the tool feed rate, a number based on metal removal rate

X
Y
Z } - coordinates indicating the direction of movement
i    for a specific G-function (code)
j
k

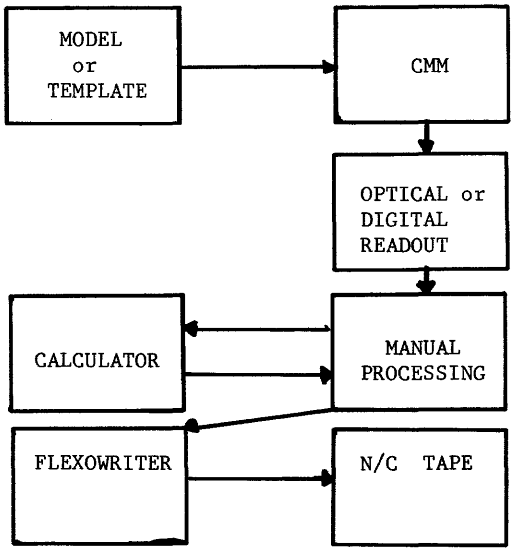


Figure 2. THE MANUAL WAY

The procedure for producing a tape manually is as follows:

1. The operator places the model or template on the CMM and aligns it as desired.

2. Data is recorded as the CMM probe scans the part. This is accomplished either by optically recording data at descrete x or y increments, or by digital strobing via a footswitch, again at descrete x or y movements. In most cases, the probe must be stopped to allow recording (in automated systems, recording is done "on the fly", i.e., while the probe is moving along the part).

3. The recorded data is "processed" manually via calculators; the result desired being the N/C tape. Optimization procedures designed to bring about minimum machine tape lengths or minimum machining time are very time consuming and thus not normally used.

4. N/C coding forms are filled out and production of the tape is then accomplished via a manual tape punch such as a flexowriter.

In the automated system, there is no operator interference with tape production except in movement and placement of the CMM probe. Figure 3 indicates how automatic production of tape is accomplished.

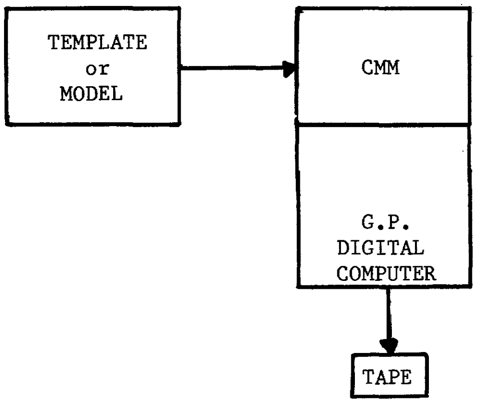


Figure 3. THE AUTOMATIC WAY

In producing the N/C tape, the operator proceeds by entering the format of the tape to be produced via computer input at the computer console. He has the ability to select any of several modes of output, including raw data printed and punched as well as optimized data punched in either ASCII or EIA code or any number of other user designated options.

Once the system is initiated, the part is placed on the table as is done manually. After placing the probe on the part, the operator depresses a foot switch when he wishes to start recording. The computer samples the input data and stores it in raw form. During the time between samples, the computer can perform optimizing and data conversion to final output format. At this point tape punching is initiated. All available computer time is utilized toward maximizing the continuous input of data and thus creating high scan speeds. During data taking, the probe is moving continuously and the data is stored in the computer memory in raw form, when this raw data is converted to output, the computer storage area becomes available for more input data shortly after the completion of the operator scanning of the part.

CMM & Computer: The Hardware Picture

Connecting a computer to a CMM requires very special hardware which, in general, appears as shown in Figure 4. This block diagram illustrates how the computer "asks" for data from the CMM at descrete points in time.

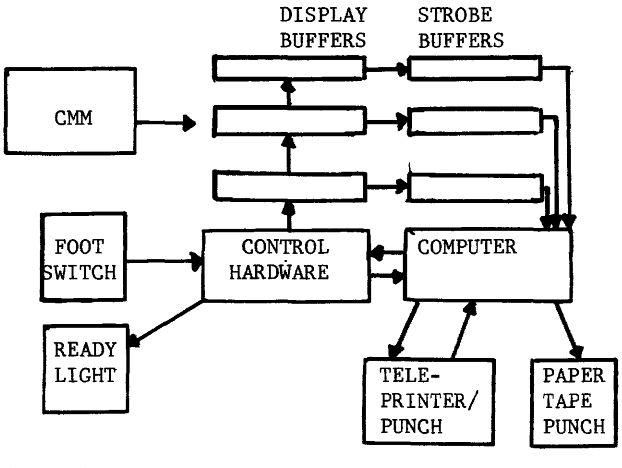


Figure 4. INTERFACING CMM'S TO COMPUTERS

In essence, data is continuously input to the display buffers (in BCD), which also supply an optical output of the probe position. On request from the computer, a signal is sent via the control buffer. This buffered data can then be transferred to the computer memory at any time desired by the computer CPU (central processing unit). Any required interaction between operator and computer is accomplished via the foot switch or the operator ready light. The ready light is turned on or off by the computer to indicate computer readiness. The foot switch is used by the operator, in a similar manner, to indicate operator readiness to the computer. The paper tape punch is used to produce N/C tape, and the teletypewriter is used in program initialization and printing instructions or data to be used by the operator.

This system provides high flexibility in a general purpose configuration. It can be used in any inspection or digitization application. Internal to the computer are various devices which aid in ease of programming and are used to maximize the total data processed. In particular, a clock interrupt of the computer is valuable to indicate when to strobe the display buffer for a data sample. This interrupt automatically halts the CPU from processing its current data and switches to an input data processing mode. When this input task is complete, resumption of the former task is effected.

Man/Machine Operation

In any application tying a computer to a machine for factory production purposes, the most significant problem encountered is that of man/machine operation. Frequently (actually in 99% of the cases) the operator of such a device as a CMM/computer knows little or nothing about the operation of computers. It is thus important to minimize those tasks which would normally require a computer operator to perform. This can be achieved via the implementation of a "HELP" system whereby the initialization portion of the software asks questions concerning the desired program operation. These questions are designed in a manner such that the operator can answer with a simple "yes" or "no". Other questions asked by the computer are answered with a numeric input, and; as format independent as possible to eliminate format errors.

Problems such as stored data overflow are eliminated via the use of the teletypewriter bell, a device that can be actuated via the computer. At such time that the program detects an eminent overflow of the stored data buffer, the bell is rung to indicate the situation to the operator. Sufficient space is allowed to give the operator time to slow down. When the buffer is emptied sufficiently the bell is once again rung--this time twice, and the operator proceeds.

The techniques mentioned above are quite valuable in achieving a minimum of problems when training an operator to use the resulting computer/CMM in an expeditious manner.

An Application, The Sheffield Dual Optimizer Program

Information Control Systems has developed an N/C tape production program for Bendix, Automation and Measurement Division, that allows the production of N/C tapes from turbine blade models. This system utilizes the Sheffield Cordax CMM and the DEC PDP-8/S computer with a high speed punch (50 characters/sec.) Software has been produced that allows maximum utilization of all available components, for this special application.

In digitizing the blade, a number of passes is made over the part in a manner similar to the N/C milling process. (See Figure 5.)

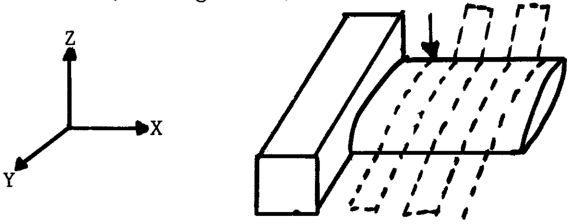


Figure 5. SCANNING THE PART

To eliminate the offset calculations for the milling tool, a probe is selected with the identical shape of the tool, and a preselected X spacing is used. This is done to achieve the required finish on the machined blade. The chord height tolerances are selected by the operator and input to the computer for optimization processing by the computer (see Figure 6.) During optimization, the computer samples and optimizes at a rate that produces a minimum length N/C tape.

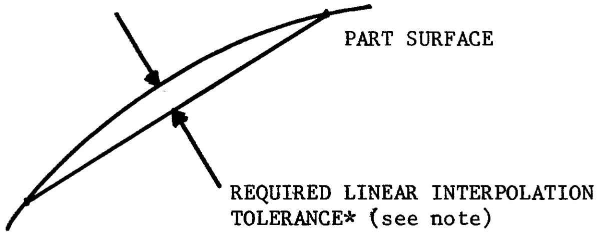


Figure 6. CHORD HEIGHT TOLERANCE

The processing program operates in foreground/background modes under 1/60 second clock interrup control. After the Cordax machine operator has keyed in program options and pressed the START button on the Cordax, the interrupt clock is enabled, and thereafter coordinate data are examined every 1/60 second until the program determines the end of a pass has been reached. This is foreground processing, and takes approximately 8 mlsec., or half of each 1/60 second interval. During this time some optimization is performed; selected points are stored internally in a memory data buffer for further optimization and subsequent output during the background processing. Background processing occurs during the second half of each 1/60 second interval and throughout the end of pass interval. When the program determines that the end of pass has been reached, the clock is disabled, allowing continuous operation in background mode until the operator again presses START. Before allowing the operator to press START however, the program makes certain that there is sufficient room available in the memory data buffer to contain one entire pass.**

This foreground/background technique allows the operator to "move ahead" of the program, i.e., he may be operating several passes ahead of the point at which the program is currently printing or punching output. The only restrictions then, on operator speed, are desired accuracy of output and core size of the computer.

A functional flow diagram of this program is shown below.

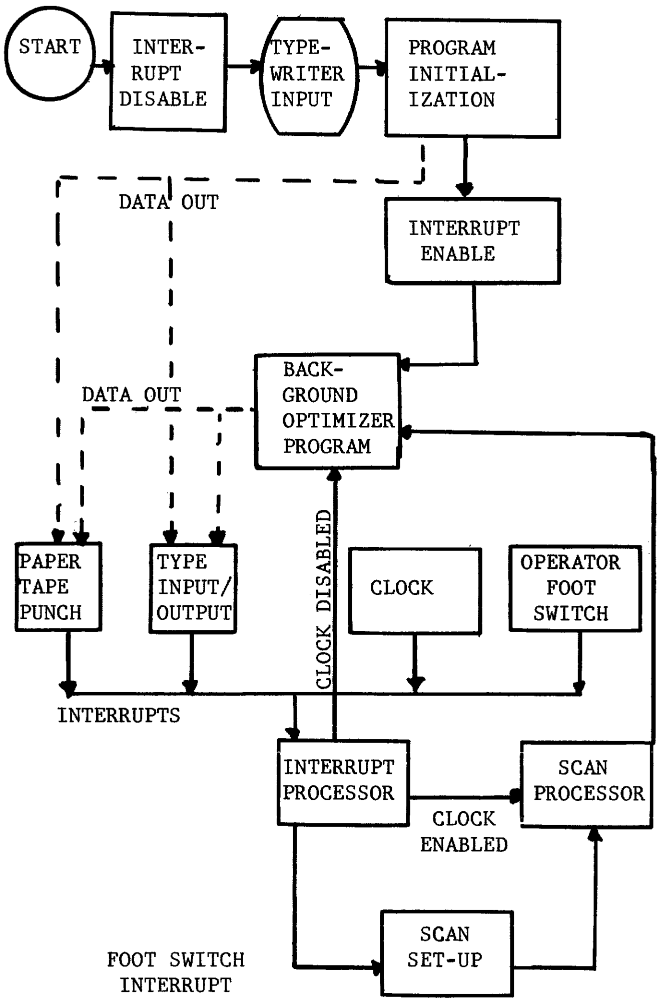


Figure 7. PROGRAM FLOW FOR TURBINE BLADE N/C TAPE GENERATOR

Clock interrupts have first priority and direct scan processing. Rough optimization occurs at this point as well as halted-probe detection (to minimize data storage) and "end of scan" detection. The computer detects "end of scan" as the probe drops off the part edge. The operator foot switch interrupt initiates scan data taking (i.e., turns on the clock), and the other interrupts return control to the background optimization routine which will continue optimization and output of data until another interrupt is detected. With all modes of operation proceeding more or less in an interconnected manner, all tasks occur simultaneously and at their individual maximum speeds. Thus, while scanning occurs, both typewriter and paper tape output is occurring. Sufficient computer storage is available to allow the operator to be several scans ahead of the tape or printer output. Input and output optimization create the minimum required raw data to be stored

*NOTE: This process can be achieved with either linear, circular or parabolic approximations to the part surface.

**A modification to the program will allow large parts to be digitized via a bell signal which signals the operator to halt scanning during the middle of a scan. When the buffer has been sufficiently emptied another bell is rung to signal continuance of the scanning process.

and a minimum amount of tape punching in output.

Applications for the Future

Applications such as the above point to the future in a vivid way. These systems are general and indicate complete man-computer interaction. The limitations lie in computer speed and memory size. Since computers are mathematically oriented, they are at their best in situations involving a large number of calculations. Optimization, coordinate translation, rotation and scaling are easy tasks for the computer, yet difficult and time consuming for the unaided operator. Thus, the following applications can be achieved as a direct result of the computer's calculating power:

1. Automatic cam alignment, i.e., the part is aligned once and the computer shifts its internal data for cam checking, no part realignment is required.

2. Several CMM's can be controlled by a single computer on a time-sharing basis. In this way, computer costs can be kept low per machine.

3. Large circle centers can be located where probe size would not allow direct measurement.

4. N/C produced parts can be checked directly from the N/C tape via computer interpretation of the tape. The computer would indicate which points to check on the printer and the operator proceeds as directed for the tolerance check. The computer would then indicate when an out of tolerance condition existed.

With the addition of servo drive motors to the probe, inspection of parts could be made fully automatic (such machines already exist). It is conceivable that the N/C tape that produced the part would also be the inspection tape.

Digitization for the production of N/C tapes could also be made completely automatic. The servo drives, in conjunction with null seeking probes,would allow the probe to automatically follow the part in much the same manner as photo electric systems presently do.

Figure 8 shows the functional orientation of a fully automatic system where the CMM is driven by the computer. The computer here performs the task of calculating step velocity inputs to the CMM. This must be done in such a way that the required tolerances of 1/1000" are maintained. One great advantage to the resultant system lies in it's adaptivity. When buffers begin to fill during data taking, a slow down of the drive motors is affected. Thus the background processor will keep track of data stored in the buffer and signal the machine drive program to slow down. All axes are coordinated via the machine drive program to achieve minimum deviation from the desired path. Any corrections necessary to achieve a display different than a direct read-out from the up/down counters of the CMM position, such as different coordinate systems (the part's, for instance) or for detection-correction,are achieved in the display drive program and updating of the display occurs as a

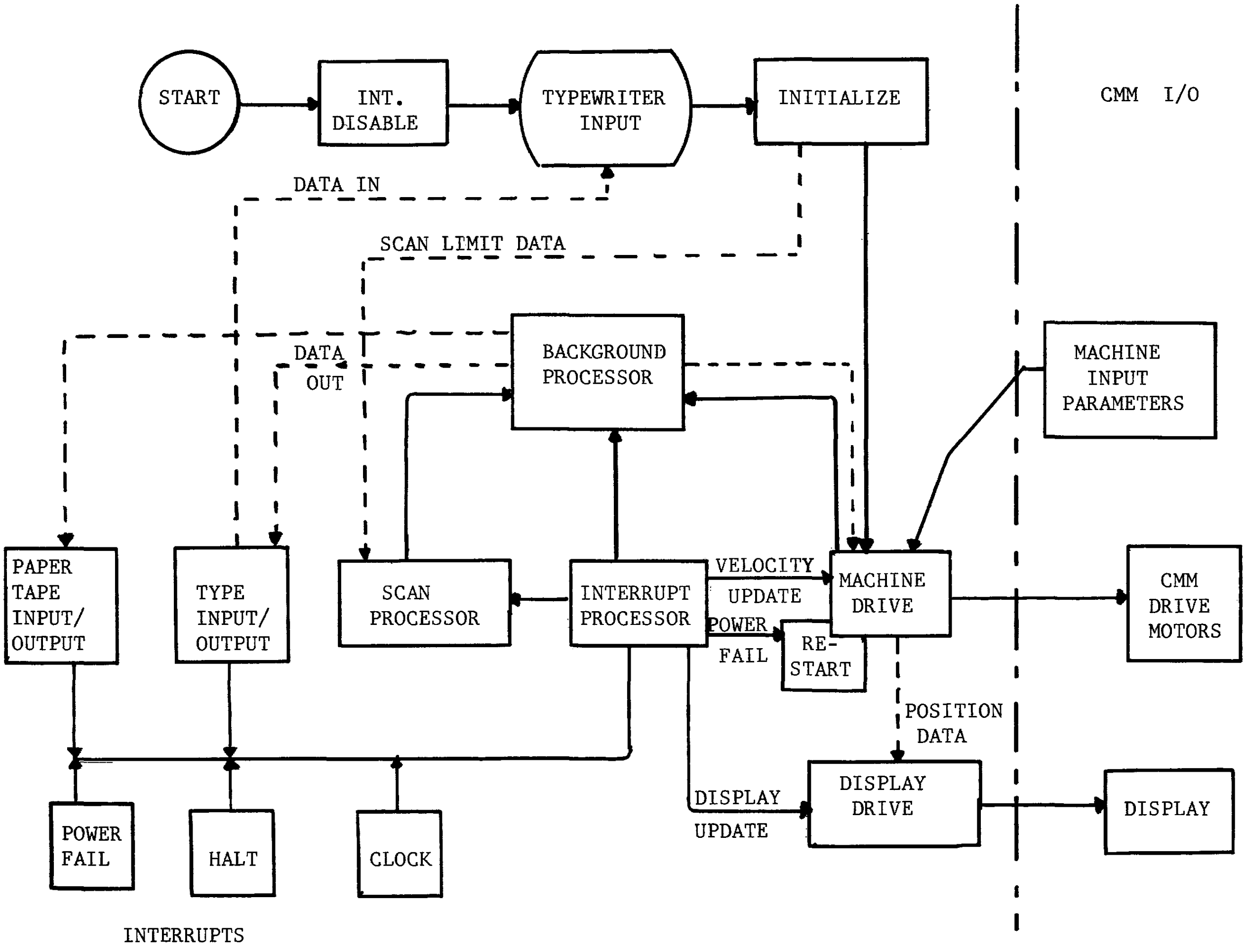


Figure 8. FULLY AUTOMATIC SERVO DRIVEN DIGITIZER (FUNCTIONAL FLOW DIAGRAM)

result. This updating need not occur any more frequently than 15 times a second due to the response frequency of the eye. The remainder of the completely automatic CMM/computer system is identical to the semi-automatic configuration previously described, with the following exception; an automatic restart capability is provided to allow restart of the motors after emergencies such as a power failure or a system failure detection. This is accomplished via a special interrupt after failure mode detect which resets parameters and restarts the motors from the zero velocity condition. If backup of the CMM position to pick up lost data is required, the restart program also handles the function.

Eventually non-contacting systems will be developed with laser interferometers which will allow in digitizing of soft materials. It is felt that at this point the limit of automation will be reached in the interfacing of computers to CMM's, the other changes will then lie in the computer software itself to bring about a very efficient man/machine system of automatic digitization.

# GRAPHICS - TERMINAL COMMUNICATIONS PACKAGE

Barry R. Borgerson
Project Genie, University of California
Berkeley, California

## ABSTRACT

The SDS-94ϕ user program communicates with the display hardware through a communications package operating between the SDS-94ϕ and a PDP-5 which shares memory and the display controller. By transferring and buffering all data and control words, the communications package handles the timing problems for the user.

With the aid of an unpluggable hardware addition, the PDP-5 runs under an interrupt monitor which handles all of the I/O for its end of the communications package.

The actual transmission between the two computers is done over a high-speed, half-duplex link and a low-speed, full-duplex path. All of the transfers over the half-duplex line are set up on the low-speed path.

## INTRODUCTION

GOCOM is a communications package, for the Project Genie GO display system, that is designed to relieve the user of the tedious task of directly programming the display hardware (see Figure 1). The two main problems encountered when trying to use the display without a communications package are the need to program the PDP-5 and the task of timing the data traffic.

Although the PDP-5 is a fairly basic computer, it is obviously desirable to avoid learning to program it if at all possible. Even if the display user took the time to learn how to program the PDP-5,·it is unlikely that he would program it very efficiently, not to mention the time that would be wasted.

The second problem, that of timing the data, is a much more formidable one. The lead strokes have to be sent from the PDP-5 to the 94ϕ while all of the display data must be sent the other way. In addition there is the data from the other devices associated with the display and the control data to be sent. The GOCOM package runs a fork in the 94ϕ to read data, including lead strokes, from the PDP-5 as soon as it is available. Several units of the various data are bufferred in the 94ϕ so the user does not have to get the data right away.

The transmission of the data is handled in such a way that, whenever possible, the low-speed, full-duplex link is used to transmit the information. Also, whenever the high-speed, half-duplex link is used, its timing is set up by the low-speed link.

External to the SDS-94ϕ, the components are the IDI controller, the PDP-5, an unpluggable interrupt improvement for the small computer, and a software package running in the PDP-5. The software package can be separated into three major components; there is an interrupt monitor that handles all of the physical I/O and task scheduling, a basic communications package which handles most of the data and control flow between the PDP-5 and the 94ϕ, and a package to handle the generation and transmission of stylus strokes from the Rand tablet.

The user interface in the 94ϕ is through a single subroutine called DIO. This subroutine is called with different values in the A register to perform its various functions. When parameters are necessary, they are passed by the X and B registers and by the location following the calling address.

There are three versions of GOCOM. One version simply allows transmission to and from the 94ϕ and transfer of control to any location in the PDP-5. Another version allows use of all of the features of DIO except those connected with the lead package. The final version uses all of the features of DIO including those associated with the lead package.

## SUMMARY OF THE DISPLAY SYSTEM HARDWARE

The hardware external to the 94ϕ consists of a considerably modified IDI (Information Displays, Inc.) display controller, a slightly modified PDP-5 with an improved interrupt system hung on it, a Rand tablet, a keyboard, a five-key handset, a light pen, and a high-speed, half-duplex and low-speed, full-duplex communications path between the PDP-5 and the 94ϕ (see Figure 2).

Below is a brief description of the display hardware. For a more complete description see the GO[1] manual.

### Display Controller

There have been several additions and modifications made to the display controller. The most important of these is the addition of a subroutine facility. An instruction equivalent to the JMS command in the PDP-5 was added as an external modification which fools the IDI logic into thinking it is a jump (JMP) instruction.

Since the display controller shares memory with the PDP-5 CPU, it was necessary to modify the display controller so that it could not execute a JMS into certain cells critical to the operation of the PDP-5. These cells include location zero, which is the PDP-5 program counter, and cells $7740_8$-$7777_8$, which hold the link-loader code. Other changes to the basic unit delivered by IDI include frame mode changing under program control, display list match-mode control to facilitate using the match feature in subroutines, and a display halt feature that doesn't reset the display controller.

The display controller gets access to the PDP-5 memory via the data-break facility of that computer. The time required to gain access to the memory is a function of the PDP-5 instruction being executed at the time the request for a memory cycle is made. Since this time can be fairly long, a one-word buffer has been added that fetches the next word from PDP-5 core while the IDI is operating on the last one.

PDP-5 Hardware

The PDP-5 is a one-address, fixed word length, parallel computer using 12 bits, two's complement arithmetic. The memory-cycle time is 6 µs. There is just one control register, the AC. Associated with the 12-bit AC is an overflow bit, which is called the link bit (L). Shift instructions cause the AC and L to shift as a 13-bit ring register.

There have been only two modifications made to the PDP-5 itself. The most drastic one is a change which prevents it from destroying the link-loader code in cells $7740_8$-$7777_8$. This hardware is wired in the PDP-5 rack at the location which was previously wired for use as a D-to-A converter. Any attempt to modify the protected cells (with the exception of a JMS from above $7740_8$) will cause a program halt without modifying the cell. The other modification is to the teletype line. A switch has been added which allows the speed of both the teletype input and output paths to be run at either normal speed or at 16 times normal speed. When the teletype line is connected to the 94ø as the low-speed communications path, it is run at 16ø characters per second.

The standard PDP-5 has a single channel interrupt system that can be enabled or disabled. In order to locate the device that is causing the interrupt, it is necessary to poll the devices by a series of test-and-skip instructions. An improvement has been made to this by the addition of an unpluggable interrupt extension (see Figure 3). This hardware accessory was designed so that it could also be plugged directly into a PDP-8.

The interrupt extension is built around a 16-channel scanner, which scans up to 16 devices at a rate of one device per microsecond. This scanner will stop when it encounters any device that is armed and has its attention flag raised. The scanner serves the function of deciding which device gets serviced, if more than one desires servicing; and it holds the number of the device causing the interrupt.

There are several advantages the interrupt extension offers over the basic system provided with the PDP-5. First of all, it is now possible to arm or disarm individual channels under program control. Thus, even if the interrupt feature of the PDP-5 is enabled and a device has its flag up, an interrupt will not occur unless the device had previously been armed by the program. Another advantage of the extension is in the way the interrupting device is determined. Instead of polling all the flags with test-and-skip instructions, the device is now located by reading a "JMP* 6ø$_8$ + (Device no.)" into the accumulator with an IOT. If transfer vectors for devices ø-17$_8$ are placed in cells 6ø$_8$-77$_8$ respectively, then executing the above instruction will cause control to be transferred directly to the routine which is to service the interrupting device. Thus, with the aid of the extension, it takes a lot fewer cells and requires much less time to get to the appropriate interrupt-servicing routine.

Instead of having a separate data-transfer IOT instruction for each device, the interrupt extension logic has only one data input-output command. The scanner selects the appropriate device and causes the data transfer to be completed between it and the AC. Whether this instruction causes data to be input or output is a function of the device being serviced; this information is hard wired into each scanner position. All device attention flags are also reset with a single I/O instruction, with the scanner directing the pulse to the appropriate device. Besides greatly reducing the number of I/O instructions needed, this scheme has the advantage that more than one device can be serviced by the same routine (see Section on GOINT below).

## THE PDP-5 PORTION OF GOCOM

There are three major components to the PDP-5 code. The components are called GOINT, 5GOCOM, and LEAD. Besides being separate logical elements of the PDP-5 software, they are actually physically separate in PDP-5 core. In fact, the version of GOCOM that runs without LEAD simply runs without the LEAD package; no other changes are necessary.

GOINT

GOINT is the PDP-5 interrupt monitor; it handles task scheduling and performs all physical I/O. This version of INT was written by the author specifically for use with the communications package, although, as you will see, it is quite general. The basic philosophy of this package is patterned after two similar versions written previously.[2]

The heart of GOINT is a task stack onto which processes are placed when they are created or when being dismissed for certain types of I/O, and from where processes are taken to be run. Processes are put on and taken off the stack in a round-robin fashion. All processes run with equal status under GOINT. If a process wants to create a fork, it does so by executing JMS* FORK with the starting address of the new process in A. This address is placed on the bottom of the stack and the current process then continues with the next instruction. A fork can terminate itself by executing JMP* QUIT which transfers control to the top fork on the stack.

Most I/O is done by executing JMS* IO with the first cell after the call containing the device code, the second cell containing the starting address, and the third cell containing the number of words to be transferred. This same routine is used for both input and output devices. On all IO calls the process is dismissed and the next fork on the stack is started up. A process dismissed by the IO routine is dismissed in a static sense; that is, it is not placed on the task stack to circulate until the I/O is completed, but is completely removed until the interrupt routine places it on the bottom of the stack when the I/O is completed. When the word-count gets to zero for any standard I/O block, the interrupt routine will disarm the device and restart the fork requesting the I/O by placing the calling address plus four on the bottom of the stack.

When a call is made on IO, a check is made to see if the specified device is in the process of satisfying a previous request. If it is, then the address of the call is placed on the bottom of the task stack. The process requesting I/O is thus periodically started and then restacked until the device is free to service it.

For the sake of future exposition, a process that is dismissed waiting for an IO block to be completed will be called statically dismissed, while a process currently residing in the task stack will be called dynamically dismissed.

Input on the low-speed link is not handled by the IO routine. The low-speed link can present characters as close as 6.25 ms apart; and since this is actually a teletype line, the characters will keep coming whether or not they have been removed. Since most characters received on this link represent commands to 5GOCOM, it is necessary to look at each character as it arrives; that is, an IO call with word count greater than one would not be acceptable. Since it will often take longer than 7 ms to work up through the task stack, it is obviously not possible to use the IO routine for teletype-link input. This problem was resolved by having the interrupt routine for this device read characters into a 2Ø$_8$ word ring buffer. A process obtains characters by executing JMS* TTYIN which returns with the character in A if one is available. If no character is available, the fork is stacked such that the request for a character is made again as soon as the process is awakened.

There is a similar problem with the tablet. When the stylus is down, coordinates arrive at 7 ms intervals. To avoid losing points, the tablet coordinates are read into a 4Ø$_8$ word ring buffer by the interrupt routine. Thus, 2Ø$_8$ points are bufferred representing about 1ØØ ms worth of buffering which is the same as for the low-speed link input. In order to keep track of the pen position of a point when it is input, the low order bit on the y coordinate is turned on if the pen is up and off if the pen is down at the time of input. Thus, even though a point may not be removed from the buffer for some time, the pen position of that point is retained.

A process can obtain the next point by executing JMS* TABLET. This process will continue to be dismissed and restarted until a point is available in the same manner as a call on TTYIN. When a point is obtained, the process continues with the instruction following the TABLET call if it was a pen-up point or the following instruction if it was a pen-down point. The A register is zero on return for TABLET and the coordinates are left in XVAL and YVAL. These cells are also part of the display list and serve to position the cursor for displaying.

It is usually desirable to know the last pen-down coordinates in any stroke. Using the pen-up flag for this purpose is somewhat difficult since its interrupt servicing routine must search for the last value in the tablet buffer. To facilitate locating the last pen-down point, a feature has been incorporated into the TABLET routine whereby points other than the next one can be requested. Calling TABLET with zero in A will return the next point, calling it with -2 in A will return the same point as it did the last time, and calling it with -4 in A will return the previous point. Thus, whenever the first point is found for which bit 11 of the Y coordinate is a one, calling tablet with -4 in A will return the last pen-down point to XVAL and YVAL.

There is no need to buffer the high speed link because it always transfers data and hence large word-count IO calls can be used. Also, the high-speed-link input, unlike the low-speed link, will wait for one word to be read before sending the next.

All of the input devices serviced by IO are treated by the same interrupt routine, as are all of the output devices. In fact, both routines first call a subroutine which locates the file for the device and determines the next address to be read from or written in to. The input routine then executes the IOC command which inputs the data and clears the device attention flag; this is followed by an indirect store through the above address. The output routine differs only in two instructions; it does an indirect fetch followed by IOC which outputs the data and clears the attention flag and the accumulator. Both routines end by going to a common routine which first decrements the word-count and checks for the end of the I/O block and then terminates the interrupt routine in the appropriate manner.

With the exception of teletype output, any I/O in progress by any of the devices serviced by IO can be terminated by executing JMS* DSABLE followed by the file number for that device. DSABLE will disarm the device and reset its file. Resetting the file associated with a device includes setting the word count to zero so that it appears free to any process trying to use it and removing the return address from the file so that the fork that was dismissed for I/O is terminated. The reason for excepting the low-speed output link is because this routine waits until that device is not busy before closing the appropriate file. This is done to assure that certain forks are actually dismissed on the expected device and not on low-speed output.

There are a few other features of GOINT, such as a subroutine that will continue to stack a fork until the low-speed output link is free, thus allowing subroutines that use this device. But these features just represent frill and contribute little to the basic philosophy of the monitor.

## 5GOCOM

5GOCOM is the PDP-5 portion of the communications package. When this process is started up, it initializes itself and then commences to run as two forks. One of the forks is used to listen to the low-speed link, the other to the high-speed link.

The fork that is listening for input on the high-speed link spends most of its time statically dismissed. It starts out by making a request for two words and dismissing until it gets them. Upon being awakened, this fork takes the first word, which is one greater than the word count of the data block, subtracts one and stores it in the word count specification cell of another IO call. The second word is then placed in the starting address cell of the next IO call and the JMS* IO is executed. This reads the remaining words of the block and stores them starting at the specified address. After finishing the block the process is again restarted, and it simply transfers to the two-word IO call which again dismisses and waits for a new block.

The fork that listens for input on the low-speed link is the heart of 5GOCOM. All of the commands from the 94Ø are sent over the full-duplex line, and this fork interprets the commands and takes the appropriate action. Many commands are accompanied with data; this data may be in the form of a single character or a series of characters to be assembled into full words. The commands are interpreted by adding the value of the command character to a transfer to the beginning of the service routines. This method has the annoying deficiency that the number of cells used by each routine must not be changed. For if one routine needs just one more cell, then the command character values for all of the following routines must be increased by one. The obvious way to avoid this is to add the command to an indirect transfer and store a series of transfer vectors to the service routines; this method was not used, however, because of the prevailing desire to conserve space.

The commands to reset, stop, start, and reset and start the display controller are handled by simply executing the appropriate IOT and then going back and waiting for the next character. The general PDP-5 reset is accomplished by transferring control to the beginning of GOINT which will clear the task stack, the teletype input buffer, and the tablet buffer, will disarm all devices, and then will transfer to the beginning of 5GOCOM which will again reinitialize itself.

The lead-enable routine is a little more involved than those above, but it is still done entirely with the command fork. The next four characters on the low-speed link are data associated with this routine. The first character corresponds to the inner window value, the second corresponds to the outer window, and the last two are assembled as the time-out constant. Actually, the routine that checks the window expects the first character to be one less than the inner window value and the second character to be one less than the difference between the outer and inner values. This conversion is done in the 94Ø before sending to save a couple of instructions in the PDP-5. Also, the negative of the time-out constant is actually sent over the link since that is what the routine that checks for time-out desires. Next, a check is made to see if the lead package is already running. If so, no more action is taken, and the net result is simply an updating of the parameters. If LEAD is not running, then a check is made to see if BAND is enabled. If so, a flag is set to inform BAND that LEAD is to be started. The reason for doing it this way is to allow BAND to gracefully terminate itself before starting up the LEAD routine. If BAND was not running, LEAD is started from this routine by creating a new fork. The LEAD package will be explained in more detail in the next section.

There is no data associated with the BAND routine. A check is simply made to see if LEAD is running; if so, a flag is set to tell LEAD to display in rubber band mode as well as lead mode. If LEAD is not running, the BAND fork is started up. The BAND routine simply displays a line from the first stylus-down position to its present position. When the stylus is lifted, this routine will send the first and last "pen-down" coordinates to the 94Ø. This information is sent over the low-speed link, and since an identification character preceeds the data, it takes nine characters to get the information to the 94Ø. This represents the longest single block ever sent over the low-speed link.

The LEAD and BAND routines are disabled by simply resetting the LEAD enabled and BAND enabled flags respectively. The routines themselves are periodically checking their enabled flag, and they will terminate themselves in a graceful manner upon encountering a reset flag.

The lead-erase routine reads the next character which is the number of strokes to be erased, and it then calls the same erase subroutine that the LEAD package uses. If it attempts to erase strokes that have not yet been output, the call will be stacked until the appropriate number of strokes have been output. For this reason, it is a good idea not to try to erase more strokes than have been received by the 94Ø.

The routines that enable the Handset, Keyboard, Match, GO, and Display-Halt routines all work the same way. A fork is created that immediately dismisses on a one-word IO block. This, of course, arms the device; and the fork will wake up as soon as any data becomes available. For the keyboard, the fork just immediately outputs the character along with an identification code. For the Match, GO, and Display-Halt devices, their forks output the appropriate identification character followed by the word split into two characters. For the handset, the situation is a little more complex. Each time a word is read, the value is ored with the running-or value and this number becomes the new running-or word. Since there is an interrupt on every position change of every key, this algorithm will give the assembled number whenever an interrupt occurs and the word read is a zero. Therefore, whenever the word read is a zero, the running-or value is output over the low-speed link as two characters preceeded by an identification character.

The five devices above are all disabled the same way. The DSABLE subroutine is called with the appropriate file number in the next cell. This subroutine will wait until the low-speed output device is not busy, and it will then terminate the fork that is listening to the device by closing the file associated with the device. The device is also disarmed at this time. The reason for waiting for the low-speed link to be free is so that the fork that is listening to the device is guaranteed to be dismissed waiting for that device and not for teletype output.

There are six data characters associated with the routine that moves a block in PDP-5 core. The first two characters are assembled into a word that represents the origin address minus one. The next two characters are assembled into one word that corresponds to the destination address minus one. These two addresses are loaded into auto-index registers to accomplish the transfers. Finally, the last two characters are assembled into a word which equals the negative of the number of words to be moved. This control fork then proceeds to move the block by means of a three instruction loop.

The command to send a block to the 94Ø is not handled entirely by the control fork. This fork reads the next four characters and assembles them into the starting address and word count of an IO call that is to send the block. A fork is then created to perform the block output since it could take a rather long time; and the controlling fork, which must listen to the low-speed link, cannot afford to be dismissed for very long. The fork that was created first issues a link-transmit IOT instruction. This is followed by a one-word IO call for the high-speed link output device. This word sends over the word count which is required by the 94Ø. Next, an IO call is made with the starting address and word count previously assembled as parameters. Upon waking up from this I/O, the fork gives a link-transmit end command and then terminates itself.

The routine to initialize a block simply assembles the next six characters into three words which represent, in order, the starting address minus one, the initialization character, and the negative of the word count. This fork then proceeds to initialize the block.

The routine to create a process assembles the next two characters into a word which represents the starting address of the process to be created. This address is then placed on the bottom of the task stack by calling the FORK subroutine. If the operation of GOCOM is to continue, then this fork must be terminated with a 5452, which is a JMP* QUIT. In this manner, the process will be created and destroyed as a fork; and, if it doesn't take too long, GOCOM need not be adversely affected.

Finally, the two commands that change the frame mode are handled by simply inserting or deleting the special frame-mode command in the display list. A display start command is given in the return-to-normal frame mode request since this is necessary to switch the mode back after removing the special-frame-mode command from the display list.

In addition to the above commands, there are two which are invisible to the user. The first of these is the clear-to-send-lead command which is handled by executing the set-sync-flip-flop (SSFF) instruction. The lead-output fork will be dismissed on an IO call for the sync device. This command then will wake it up and the outputting of the stroke will commence. The last command that is sent to the PDP-5 over the low-speed link is one to determine when the PDP-5 has finished all previous requests. This command is satisfied simply by immediately sending a character back on the low-speed link to inform the 94Ø that 5GOCOM has finished with all previous requests.

The basic elements of 5GOCOM have now been described. It is obvious from the above descriptions that the fork that listens to the low-speed link is the heart of this package, and that the low-speed, full-duplex path is the center of the control passed between 5GOCOM and DIO.

LEAD

The LEAD package runs as two forks. One of these forks inputs the tablet coordinates, checks to see if the points fall within the window, and places them in the display list. The other fork outputs the strokes to the 94Ø. As soon as the stylus is pushed to the down position, the first "pen-down" coordinate is written into the lead buffer. Subsequent points are read and checked to determine if they fall within the window that has previously been specified by the user. Points that do not fall within the window are discarded. As soon as a point is discovered that falls within the window, it is written into the lead buffer and this point is defined as the point from which the future window checks will be made. This process is continued until the stylus is raised at which point the last "pen-down" point is written into the lead buffer and the stroke is added to the stroke stack. The last pen-down point is written into the lead buffer regardless of whether or not it falls within the appropriate window. This is done to allow closure when using very large inner windows for approximating certain geometrical shapes.

After the stylus is raised, the coordinates are continuously read but ignored except for displaying the cursor. Also, while the stylus is up, every time a pair of coordinates is read, the time-out counter is decremented. If the time-out counter gets to zero, a time-out character is sent to the 94Ø to inform the user. The time-out counter is not decremented while the stylus is down, and it is reset from the user specified time-out constant on every transition of the stylus from the down to the up position. The counter is also reset from the time-out constant if it ever hits zero. The time-out counter is set to its maximum value when the LEAD package is first enabled to avoid getting several time-out characters sent over the link before the user has a chance to start using the stylus. Since coordinates become available about every 50 ms, when the stylus is up, a time-out will occur if the stylus is up for a period of time equal to the time-out constant * 50 ms.

The lead buffer has room for $1\text{ØØØ}_8$ coordinates. This buffer is part of the display list so that the number of points that can be displayed by LEAD is $4\text{ØØ}_8$. The lead buffer is stroke oriented, and hence, there is a stroke stack which can hold pointers to as many as $5\text{Ø}_8$ strokes. When writing points into the lead buffer, if a point that is part of a stroke in the buffer is about to be written over, then the entire stroke is erased before writing the point. Thus, the fork that writes into the lead buffer automatically makes room when necessary by erasing the oldest stroke in the buffer. The erase subroutine always checks to see if the stroke to be erased has been output yet before removing it from the buffer. If it has not been sent to the 94Ø yet, then the erase request is stacked where it will continue to make the request until the stroke is sent. Since this is the same fork that reads the tablet coordinates, the cursor will stop on the screen if too long a time passes before this process can continue. The system is fast enough, however, that under normal circumstances this should never happen. If, however, the fork which reads the lead strokes in the 94Ø should die, then the cursor will stop completely as soon as enough strokes have been drawn to fill the lead buffer with strokes not yet output.

The output fork keeps a pointer to the stroke stack; whenever this pointer does not coincide with the top of the stack, then there is a stroke to be output. The first thing that this fork does upon encountering a new stroke is to determine the word count and set up the parameters for the IO calls to send the stroke over the high-speed link. Next, a lead transmission-request character is sent over the low-speed link followed by the word count split into two characters. The fork then dismisses on an IO call to the sync device. When the confirmation character comes back, the SSFF command will be issued, which will in turn wake up the fork dismissed on the sync device. Next, a one word IO call is made to output the word count, followed by one or two blocks depending on whether or not the stroke wraps around in the lead buffer. After outputting the stroke, this fork then goes back to check for more strokes. When no stroke is waiting to be output, this process is placed on the bottom of the task stack; and it continues to circulate through the stack until a stroke is entered.

## 94Ø PORTION OF GOCOM

Basically, there are three components of the 94Ø portion of GOCOM. The heart of this system is a subroutine called DIO which will handle a large number of tasks for the user when called with various values in the A register. Upon receiving an initialization call, DIO will start up a fork, herein called 9FORK, which will continue to run as long as the user program does. 9FORK is responsible for getting the lead, band coordinates, and device words out of the PDP-5 and buffering them in the 94Ø so that the user need not worry about the timing involved in accepting the data. There is a package called 5OBJ which allows the user to specify, modify, and delete both subroutine and fixed display objects. In addition, this package handles all memory management within the PDP-5 so the user need not, and in fact should not, use DIO directly to send data to the PDP-5 if 5OBJ is being used.

### DIO

The best way to describe the extent of this subroutine is to list the various calls that can be made on it. The value of A that corresponds to each call is included but the parameters are not.

| A (in octal) | Function |
|---|---|
| Ø | Reset Display |
| 1 | Stop Display |
| 2 | Start Display |
| 3 | Reset and Start Display |
| 4 | Initialize PDP-5 |
| 5 | Enable Lead |
| 6 | Disable Lead |
| 7 | Enable Band |
| 10 | Disable Band |
| 11 | Erase Lead |
| 12 | Enable Handset |
| 13 | Disable Handset |
| 14 | Enable Keyboard |
| 15 | Disable Keyboard |
| 16 | Enable Match |
| 17 | Disable Match |
| 2Ø | Enable GO-Button |
| 21 | Disable GO-Button |
| 22 | Enable Display-Halt |
| 23 | Disable Display-Halt |
| 24 | Move a Block |
| 25 | Send a Block |
| 26 | Read a Block |
| 27 | Initialize a Block |
| 3Ø | Create a Process |
| 31 | Set Special Frame Mode |
| 32 | Return to Normal Frame Mode |
| 33 | Read Current Lead Stroke in Absolute Format |
| 34 | Read Next Lead Stroke in Absolute Format |
| 35 | Read Current Lead Stroke in Relative Format |
| 36 | Read Next Lead Stroke in Relative Format |
| 37 | Read Band Coordinates |
| 4Ø | Read Handset Word |
| 41 | Read Keyboard Character |
| 42 | Read Match Address |
| 43 | Read GO Word |
| 44 | Read Display-Halt Address |
| 45 | Read Device Word |
| 46 | Initialize Basic GOCOM |
| 47 | Initialize Normal GOCOM |
| 5Ø | Initialize Lead GOCOM |
| 51 | Plot |

The calls for A from Ø through $32_8$ were covered sufficiently in section on 5GOCOM. For the calls concerning the lead strokes, absolute format refers to the pairs format in which points are generated. The relative format is not good for most types of computation, but is desirable if the stroke is to be sent back to be displayed. The reason for two types of calls for both modes is to allow the user to obtain any given stroke in both formats. This would be desirable, for example, if the user wished to perform computations on the absolute formatted data, but he also wanted to display it for awhile on the screen. The time-out referred to in Section 5GOCOM manifests itself here in that a user program will return to the calling address plus two if a stroke is available. If no stroke is available, then the user program will hang; and if a time-out character arrives from the PDP-5 before a stroke, the user program will be returned without skipping.

When reading band coordinates, the user program has the option of being dismissed until coordinates become available or of returning without skipping when there are none available. This option is determined by one of the parameters of the A = $37_8$ call on DIO. A similar option exists for the five devices which can be individually enabled or disabled. For the calls on DIO with A = $4Ø_8$ through A = $44_8$, control will be returned to the user at the calling location plus one if no word is available. If a word is available, control will be returned at the calling location plus two with the word in A. If it is desired to hang and wait for a word, then DIO is called with A = $45_8$. When any device word becomes available, control will be returned at the calling address plus two with the word in A and a code in X which identifies the device associated with the word.

The three initialization calls differ mostly in that they send different packages to the PDP-5. The only other difference is that of the lead GOCOM initialization, the user must specify the starting address and word count of a lead buffer. Besides initializing all internal buffers, flags, and pointers, these routines attach the teletype line that is to be used as the low-speed link, open the files of the two high-speed paths, ship the code to the PDP-5 and start up 9FORK. Before the code is sent to the PDP-5, characters are sent over the low-speed link which simulate a create-fork call on DIO. These characters attempt to create a fork running in the link loader. If the PDP-5 was already running in the link-loader, these characters will be ignored. If, however, the PDP-5 was running in any version of GOCOM, then it will be sent to the link-loader area. Once in the link-loader routine, the PDP-5 is ready to receive a new program from the 94ø.

Because of a 14-bit address field, the 94ø user program has an addressable space of 16K decimal. Since GOCOM is a general user interface, it will be used by many programs; some of these programs may be quite large so it is desirable to avoid keeping the $23øø_8$ words of PDP-5 code in the user space. In order to avoid this, the PDP-5 code, together with a very short program to transmit the appropriate portions of it, now exists as a subsystem which can be invoked by GOCOM. The actual sequence is to first read the address relabeling values for the subsystem along with the starting address and then use these values to create a fork which will run in an address space separate from the users. When the code has all been sent to the PDP-5, the fork terminates itself and releases all the memory it had acquired.

The final call on DIO is one which causes the display list to be plotted on an incremental plotter.

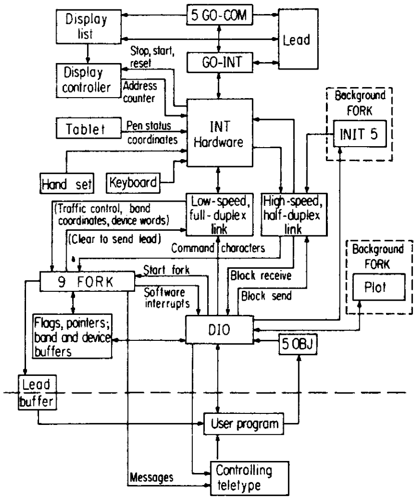


Figure 1 Block Diagram of GOCOM

The plotting is accomplished in a background fork in a similar setup to that for sending the PDP-5 code. Thus, the plotting routine and core image of the PDP-5 do not occupy any of the users space, and the user can continue using GOCOM while the plotting is going on.

## 9FORK

9FORK is the fork that listens for data coming from the PDP-5. Upon receiving lead strokes, band coordinates, or device words, 9FORK executes the appropriate interrupt to wake up the user program if it happens to be dismissed in DIO. If the user program is not dismissed in DIO, then any interrupt from 9FORK is ignored. When this fork receives the time-out character from the PDP-5, it simply enters it into the stroke table as a stroke of zero length. The same interrupt is then issued as if a stroke had just arrived, and DIO will know of the time-out by the stroke length.

## Summary

The description of the PDP-5 portion of GOCOM was given in fairly great detail. The detail was provided with that section because most people reading this paper will be somewhat familiar with the PDP-5. Since most DECUS members will not be familiar with the SDS-94ø and since the general ideas of the data and control traffic were presented in the 5GOCOM section the description of the 94ø portion of GOCOM has been presented as just an overview. If the reader would like more detail on this portion of the display package, a more extensive paper is being written for Project Genie use, and this paper can be obtained by writing the author.

## REFERENCES

1. Borgerson, B., GO, Genie Graphical Input-Output System. Project Genie Doc. R-19, University of California, Berkeley, California, June 1968.

2. Hornbuckle, Gary D., A Multiprogramming Monitor for Small Machines. Comm. of the ACM, Vol. 10, No. 5, May 1967.

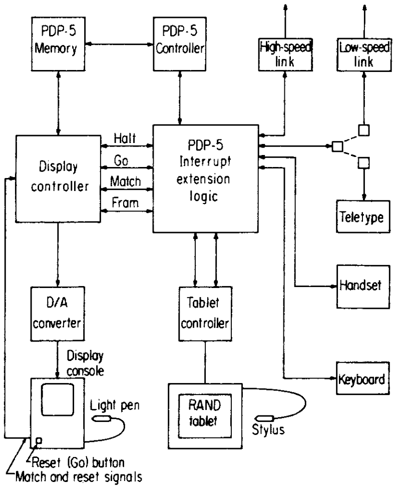


Figure 2 Block Diagram of Display Hardware

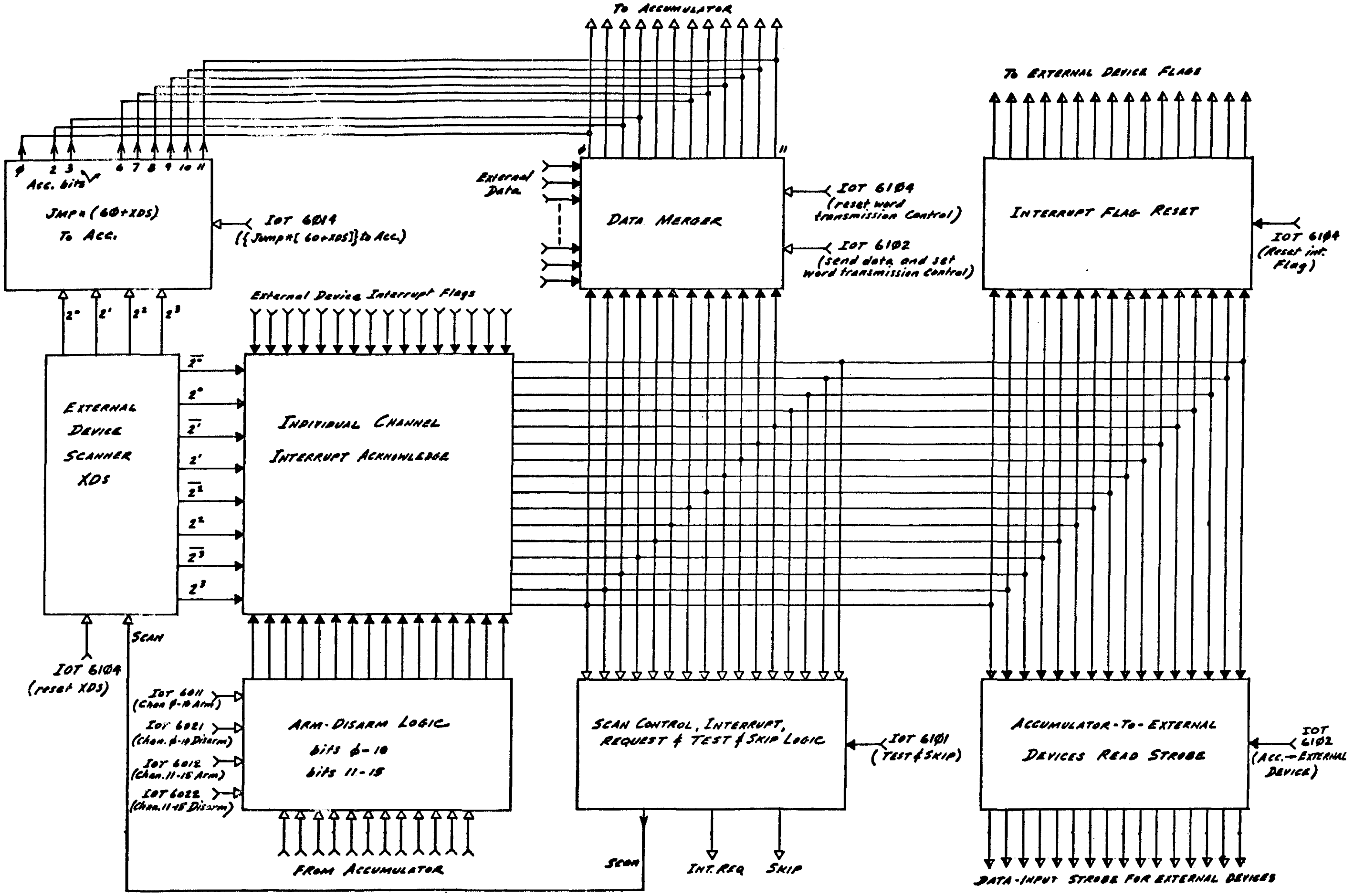


Figure 3 Block Diagram on Interrupt Extension

# GRAPHIC SOFTWARE SYSTEM USING A PDP-9/339 SUPPORTED BY A RM09 DRUM

Glen C. Johnson
Phillips Petroleum Company
Atomic Energy Division
Idaho Falls, Idaho

## ABSTRACT

A package of PDP-9 subroutines has been developed to facilitate the use of the 339 display and conserve core storage by creating display files on the RM09 drum. This package requires 2700 PDP-9 core locations, an RM09 drum, and a 339 display unit. These subroutines are both MACRO-9 and FORTRAN IV compatible and create display files in vector, text, and graphplot modes with parameters. Routines to initialize the 339 and service the light pen and function box are provided. A file swapping technique, from drum to core, permits execution of large files of display commands in a small core buffer.

## INTRODUCTION

A PDP-9 system that includes a 339 Programmed Buffer Display and an RM09 Serial Drum offers a powerful display option for large files of display commands. The basic PDP-9 features 8192 words of 18-bit core memory with a direct memory access (DMA) channel for a high rate of Input/Output transfer. The 339 is an incremental CRT with a processor that shares the computer memory. The RM09 drum, on our system, has a storage size of 131,072 words and information is transferred in blocks of 256 words.

While the display processor is reading data, the PDP-9 processor (through interrupts) is available for the transfer of display files from drum to memory. Digital data are required by the 339 display unit at a high transfer rate for visible images. The display unit is interfaced to the PDP-9 through the DMA multiplexer to meet this requirement. The drum is also interfaced through the DMA multiplexer. By controlling the transfer of data from drum to the display unit, via core memory, we can display large files of commands in a small memory buffer.

Because interaction between the display unit and the drum is rather sophisticated, it was desirable to create a graphic software system to facilitate the use of this display option. When developing this graphic system the following were our requirements:

1. Large display file capability.

2. Minimal core storage requirements.

3. Operator-Computer communication.

4. Versatility for usage.

The subroutines are FORTRAN IV and MACRO-9 compatible to provide versatility for usage. Memory storage requirements vary from 1600 to 3200 core locations, depending on the number of subroutines used. This includes storage requirements for subroutines, display buffers, and the character generator. Function box and light pen subroutines are available for communication between the operator and the computer.

## CREATING GRAPHIC FILES

Files of display commands are created on the drum with a series of calls to subroutines. These files are referred to as data sets. Each data set may have up to 1280 commands, which is 5 drum blocks. Five data sets can be constructed with this system giving a possible total of 6500 display commands, all of which can be displayed at the same time.

### Creating Data Sets

A subroutine that names the data set and initializes conditions must be called. Following this call all display commands from mode and parameter subroutines are placed in the named data set until a call is made to end the data set. An excess of 1280 commands per data set will create an error condition and will be handled by an error subroutine. Only one data set can be created at a time. Data sets can be recreated at any time to update graphic images.

### Parameters and Modes

The parameters that can be set when creating data sets include eight levels of intensity, four scales, and light pen sensitivity. Parameters can be set as desired and are in effect until changed in this or some other data set. The types of mode subroutines available for creating graphic images are graphplot, text, line, and set point. An unintensified line and set point are used to position the beam on the screen while graphplot, text, and line modes create visible images. Figure 1 is a flow chart showing the decisions a user can make to create data sets and it summarizes the subroutines available for each decision.

## DISPLAYING GRAPHIC FILES

Two methods are used for displaying data sets after they have been created. The first is a general display subroutine and the second is a special rotate display subroutine. Only one of the two display methods can be executed at a time, but they can be used alternately as desired within the same user program.

General Display

This method of display is used to display images as they have been created in data sets. Any selected combination of data sets (up to five maximum) can be displayed at any one time. This subroutine can be recalled as desired to display various combinations of data sets. A file swapping technique, from drum to memory, has been developed to load the selected data sets into a memory buffer for the 339 to process. The file swapping is serviced by the PDP-9 interrupt system and continues to operate until it is terminated.

Control is a function of this subroutine which keeps the display and drum transfer synchronous. After the file swapping and display have been initialized, program control returns to the user program. Figure 2 is a picture of four data sets being displayed simultaneously with 1024 graphplot points in each data set.

Rotation Display

This special display method was developed with this system to rotate graphplot data because of the demand at this installation to analyze data utilizing the scope. A maximum of 5120 points can be rotated past the screen with a choice of 128 or 512 points displayed. The direction of rotation (right or left) can be selected as desired and there is a choice between two speeds. Selection of speed, number of visible points, and direction of rotation is normally made from the function box. Two additional blocks or 512 display commands of non-rotating data can also be displayed allowing for grids or other information with the rotating data.

The automatic scissoring available on the 339 display unit is used by this subroutine for rotation by altering the visible sector of display. A file swapping technique is also used by this subroutine to minimize core storage requirements. The center point of this graphplot display has a higher intensity and is identified by a number displayed on the screen. This can be used to retrieve information from the plot for analysis. Figure 3 is a picture of this special rotation feature. Observe that 512 points are being displayed and the bright point identifies the 415th point in the plot.

## SYSTEM LINKAGE

The assembler and compiler generate the linkage necessary to provide a path to the subroutines and a return path to the user's program. Linkage for the handling of light pen and function box interrupts is generated by graphic system subroutines to provide paths to user service subroutines.

## OPERATOR COMMUNICATION

Communication between the operator and the computer is an important capability, and has been integrated into the graphic software system. The light pen and function box gives the operator program control during execution while viewing graphic images. Service for light pen and function box interrupts must be handled by a user subroutine. Status of buttons and light pen hits can be obtained in the user service subroutine with calls to this package.

## APPLICATIONS

Presently the most demanding use of this graphic subroutine system is for data analysis, although this is not a limitation to its potential. Other areas in which we plan to use this package are text editing, information retrieval, and management information systems.

## CONCLUSION

Development of this graphic system was desirable to facilitate the display option on our PDP-9. By storing display files on the drum we are able to display large files and minimize memory storage requirements. The ability to create graphic images from FORTRAN programs and communicate to them has made it possible to create elaborate displays quickly and easily. Figure 4 is a flow chart summarizing the utilization of this graphic system. This chart only suggests the general sequence to use the system. Between each call to this system user program effort will normally take place. Figure 5 is a listing of all callable subroutines and their functions.

## ACKNOWLEDGMENTS

I thank Wayne Black, Ric Davies, and C. Wendell Richardson for their assistance during the development of this graphic system.

| SUBROUTINE | NAME | FUNCTIONS |
|---|---|---|
| INDSP | Initialize Display | Input the character generator if needed. Locate free drum blocks for data sets. |
| BEGDS | Begin a Data Set | Set conditions to create a data set. Name the data set. |
| SETPT | Set Point Mode | Add commands to the data set to set the beam to a screen position. |
| PARM | Parameter | Add commands to the data set to set the scale, intensity, and light pen sensitivity. |
| GRAPHP | Graphplot Mode | Add a maximum of 1024 points to the data set for plotting. |
| LINE | Line Mode | Add commands to the data set to move the beam in a delta Y and delta X direction. |
| TEXTA | Text ASCII | Add commands to display 5/7 packed ASCII data or Hollerith constants. |
| TESTI | Text Integer | Add commands to display signed integer numbers. |
| ENDDS | End of Data Set | Terminate the construction of the data set being created. |
| INDEV | Initialize the Display Device | Select the data sets to be displayed and initialize the 339 display unit. |
| ROTATE | Rotate | Set up data sets for rotation and initialize the 339. |
| ROTAL | Rotate Left | Rotate graphplot data left. |
| ROTAR | Rotate Right | Rotate graphplot data right. |
| RSPEED | Rotate Speed | Select between two speeds of rotation. |
| RSCALE | Rotate Scale | Select between two scales for rotation. |
| LPSET | Light Pen Set | Generate linkage for the user's light pen service subroutine. |
| LPSEV | Light Pen Service | Read the X and Y screen position of a light pen hit. |
| LPRET | Light Pen Return | Return program control to the main program following a light pen interrupt. |
| PBSET | Push Button Set | Generate linkage for the users push button service subroutine. |
| PBSEV | Push Button Service | Read the push button status. |
| PBRET | Push Button Return | Return program control to the main program following a push button interrupt. |

Figure 5 Callable Subroutines for Graphic System

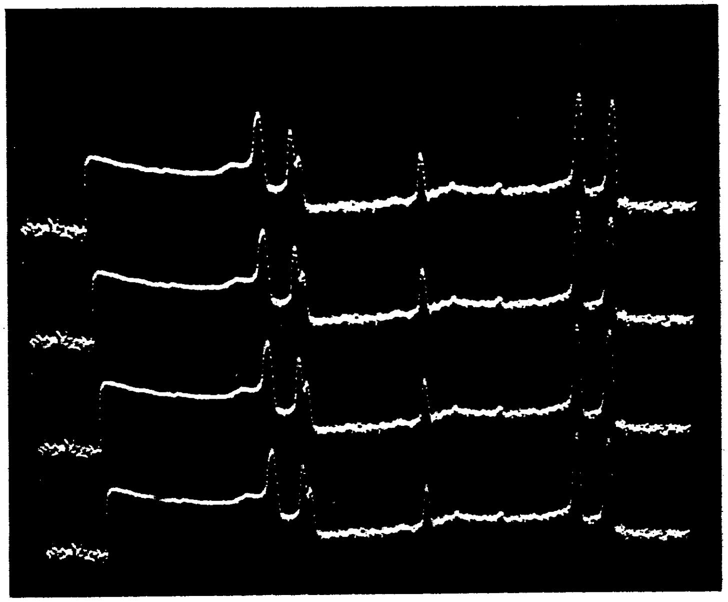


Figure 1 Creation of Data Sets

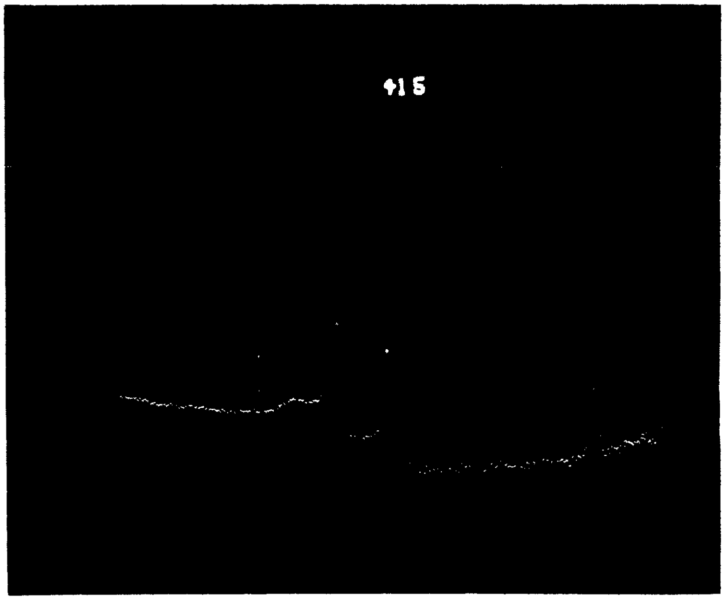


Figure 2 Graphic display of four data sets; each with 1024 graphplot points.

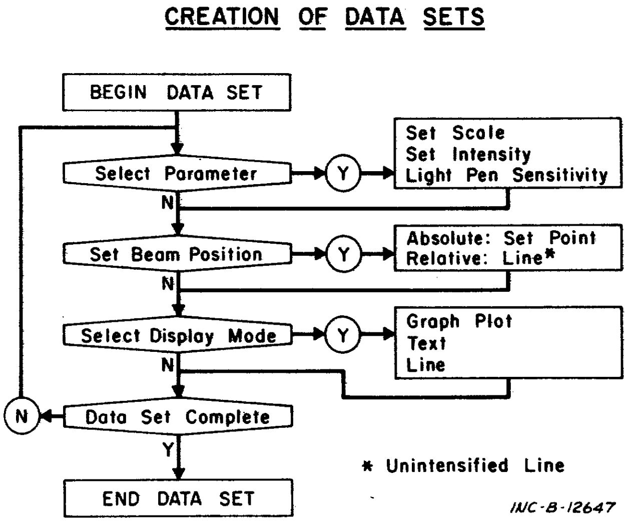


Figure 3

Graphic display using the ROTATE feature. The brighter point, at the peak, in the center of the display is identified as the 415th data point.

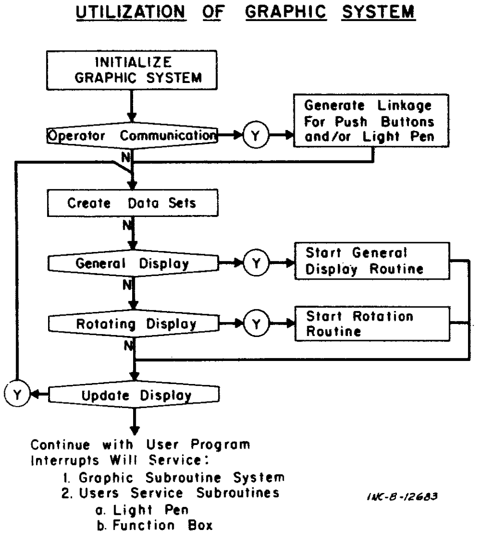

UTILIZATION OF GRAPHIC SYSTEM
INITIALIZE GRAPHIC SYSTEM
Operator Communication
Y
Generate Linkage For Push Buttons and/or Light Pen
N
Create Data Sets
N
General Display
Y
Start General Display Routine
N
Rotating Display
Y
Start Rotation Routine
N
Y
Update Display
Continue with User Program
Interrupts Will Service:
1. Graphic Subroutine System
2. Users Service Subroutines
a. Light Pen
b. Function Box
INC-8-12683

Figure 4 Utilization of the Graphic System

# REAL-TIME COMPUTING WITHIN A TIME-SHARING SYSTEM

Peter M. Hurley
Digital Equipment Corporation
Maynard, Massachusetts

## ABSTRACT

This paper describes the capability of the PDP-10 to perform on-line real-time tasks concurrently with time-sharing activity. The PDP-10 is not limited to a single real-time job, nor is it limited to running solely in a background batch mode during real-time operation. While handling several real-time jobs, such as on-line process control or data acquisition, the PDP-10 system can support a complete time-sharing service including simultaneous data processing jobs, batch jobs, and program development. Of prime importance is the consideration of the general real-time problems including high priority scheduling and real-time queues. The paper discusses the implementation of some of these real-time features and is supplemented by examples of the techniques employed at existing PDP-10 installations. The paper concludes with a description of the design goals for a multi-user real-time system which allows the running and testing of undebugged real-time jobs without degrading the perfɔrmance of other jobs.

## THE PDP-10 REAL-TIME TIME-SHARING COMPUTER SYSTEM

Real-time computer systems have traditionally been dedicated to controlling a single real-time processor, eliminating the utilization of any remaining computational power of the system by other users. Likewise, most time-sharing systems have excluded real-time functions because of the strict demands which they place on the system. For instance, when a special real-time device requests attention, its controlling program must be activated quickly. A real-time job also needs to communicate directly with its special hardware through the use of I/O instructions. Both of these demands are contrary to the general time-sharing philosophies.

The PDP-10 time-sharing system, however, was designed to allow real-time activity to run concurrently with time-sharing. The software modularity permits easy additions and alterations. The flexibility of the scheduling algorithms allows the addition of special job priority schemes. Real-time jobs can even run in a special hardware mode which permits the execution of I/O instructions. The PDP-10 system is definitely the first dual purpose real-time time-sharing computer system.

## REAL-TIME COMPUTER USAGE

The crucial aspect which must be examined when considering any real-time problem is the response requirement. Before a system can be deemed adequate for real-time work, the response requirements must be satisfied. There are three types of real-time jobs (each classified by its response requirement), the time critical job, the time important job and the time periodic job.

### Time Critical Jobs

The time critical class of real-time jobs is definitely the most demanding and the hardest to satisfy. When a signal arrives from a time-critical device, the computer must respond (i.e. start the device handling routine) within a specified period of time to avoid losing information. The card reader is a prime example of such a real-time device. Once the reader starts reading a card it cannot stop until the entire card is read. Since the reader reads a column at a time, overwriting the last one, the computer must fetch and process each card column before the next one is read.

Time Important Jobs

The time important job is very similar to the time critical jobs; the major difference being that if the computer fails to respond to the real-time request within the specified time period, nothing is lost. The paper tape reader and punch for example are time important devices. It is desirable to keep these peripherals running as fast as possible but if for some reason the computer cannot respond quickly enough, they will stop and wait for it to catch up. Likewise, if the computer fails to send the next picture to an on-line display soon enough, the display will flicker, but as long as the frequency of flickers is low, the system is adequate.

Time Periodic Jobs

The time periodic job requires that it be run at definite intervals of time providing it with a steady execution rate. An important distinction between this class and the other classes is that the computer knows when it must run the real-time job and it may be able to schedule other activities around the job. For example, a picture on a display must be refreshed every 35 milliseconds to remain flicker free. One of the time periodic functions of the system is to restart the display at least every 35 milliseconds.

It should be noted that both the time important and the time periodic jobs can be treated as special cases of the time critical class. The time important job can be treated exactly the same as a time critical job, and the time periodic job can be put to sleep for its inactive period and when it wakes it too can be treated exactly like a time critical job.

## INTEGRATING REAL-TIME FUNCTIONS INTO THE PDP-10 SYSTEM

The response requirements for real-time jobs can be segmented into three categories; immediate response (i.e. critical time period of less than 300 µseconds, priority response (response of about a millisecond), and user response. For jobs with real-time devices which require immediate response, a special service routine to control this device must be built into the monitor. This routine runs at interrupt level and allows the user to communicate with the device in the same way he communicates with a standard peripheral unit. Since the monitor is truly modular, such a routine is integrated into the system with ease.

Those real-time jobs with very long critical time periods can run as standard time-sharing jobs. However, to communicate directly to the real-time devices, these jobs can request to be placed in a privileged mode where direct I/O instructions (which are normally forbidden during time-sharing) can be executed. In fact, a privileged user can even use the priority interrupt system for brief periods of time so that uninterrupted bursts of data can be transmitted to or from a real-time device at maximum speed.

High Priority Scheduling

When a real-time job demands to be run in priority response mode, the monitor gives this job a high priority, stops the present job, and forces the scheduling algorithm to choose another job. Thus, since the real-time job has higher priority, it is run immediately. There are two kinds of PDP-10 time-sharing systems; the non-swapping system and the swapping system, each with its own scheduling algorithm. Both of these schedulers depend on a priority queue structure from which to choose the next job to run. Since both of the scheduling algorithms are table driven, a high priority real-time queue is added to either one without difficulty. When a real-time job requests to be executed (i.e. an interrupt has been received from one of the real-time devices) the monitor places this job in the high priority queue and forces rescheduling. Of course, any number of jobs can be put in this queue and are run on a first-in-first-out basis.

Under the non-swapping (10/40) monitor, all jobs are resident in core. This greatly facilitates the solution to high-priority scheduling. When a real-time job demands to be run, the monitor interrupts the present job and starts the high priority job. However, under the swapping (10/50) monitor, high priority scheduling is considerably more complex. If the high priority job has been swapped out onto the swapping device, then it must be brought in before it can be executed. To make enough room in core for this job to be swapped in, several other jobs may have to be swapped out first. Thus, even though a job has been placed in the high priority real-time queue, it might be hundreds of milliseconds before it is swapped in and started.

### Locking A Job In Core

It is often desirable to lock a job physically in its position in core. In the swapping environment this prevents the job from being swapped out and insures a faster response time. This also prevents the situation of finding the real-time job half shuffled between two places in core (shuffling can often take many milliseconds to complete). However, locking real-time jobs in core can cuase several very subtle problems. The most obvious problem is core fragmentation. If a job is locked into the middle of core dividing the remaining core into two pieces each of less than half of the original available core, there might be many jobs stranded on the disk that are too big to fit into either segment. This problem is almost completely solved if the job is locked either at the bottom or the top of core leaving one large block of free core. There is still the case when a job is still too big to be swapped in regardless of where the real-time job is locked in core. This bind can be eliminated by restricting the allowable size of the user jobs and by restricting the size and number of real-time jobs.

## EXAMPLES OF EXISTING REAL-TIME APPLICATIONS DURING TIME-SHARING

All of the previously mentioned real-time methods are being used at various PDP-10 and PDP-6 installations. The most common solution to individual real-time problems has been to build a service routine into the monitor to handle the real-time requests from special hardware. This approach is completely satisfactory whenever the service routine is small and unchanging. For instance the film advancing routine in the PEPR system at MIT runs in a real-time mode. This routine calculates how long it will take to advance to the next picture frame, starts the advancing mechanism, and puts in a request for a film stopping routine to be executed when that time has elapsed. Since the stopping of the film advance mechanism is done at a high priority level the system can continue scheduling and running the normal time-sharing users until the real-time film controller requests to be run again.

The University of Rochester has two PDP-8's doing simultaneous data collection under a non-swapping monitor. The PDP-8's act as intermediate buffers between the experiment and the main computer. When one of the PDP-8 buffers becomes almost full, it initiates a real-time job in the PDP-6 which reads and analyzes the accumulated data. This is accomplished by putting the data analysis job into a high priority queue. The monitor then halts the current job and starts the real-time job, guaranteeing it at least 17 milliseconds to complete its data transfer. This real-time queue can handle any number of jobs, scheduling each according to its entry into the queue.

The Max Planck Institut at Mulheim, Ruhr, Germany is using a PDP-10 swapping system to control many on-line gaschromatographs and mass-spectrometers. The mass-spectrometer data analysis program resides on the disk until a gaschromatograph signals that a mass-spectrometer is being activated. The mass-spectrometer control and analysis program is given the highest priority to run, cuasing it to be immediately swapped in and started. Since the data rate during a mass-spectrometer run is very fast, the analysis program is run at this high priority until the run is completed.

Stanford University, running under a swapping system, uses a time periodic real-time mode to handle their real-time jobs. During this mode of operation, the real-time jobs are locked into core and each job is scheduled on a periodic basis. This type of scheduling is crucial for simulations which require a steady time base. For example without fixed periodic scheduling the simulation of human hand movements would be jerky, and computer generated music would be off key.

The United Aircraft Corporation has a more sophisticated real-time system. The real-time jobs are placed in a high priority queue where they are guaranteed a certain percentage of the CPU time. The leftover CPU time is used for time-sharing. This installation has also solved the core fragmentation problem which arises when real-time jobs are locked into core. Real-time jobs are always shuffled to the bottom of core before being locked in place, and the monitor checks that no job will be kept out on the swapping device before it locks a real-time job into core.

## FUTURE GOALS

The problems encountered in attempting to integrate real-time capabilities into the time-sharing world are numerous, and not all of them can be solved. An application which interrupts too often or uses too much CPU time at a high priority level can render background time-sharing useless. Likewise, a real-time job which can accidently crash the system or destroy another user's area can also make background time-sharing unworkable. There must be a real-time package offering high priority scheduling and the ability to lock jobs in core. Service routines for real-time devices should not have to be built into the monitor. Real-time users should be able to chain their service routines onto the PI channels and debug them in user mode without being able to hang the rest of the system. These are the goals of the present real-time software development at Digital.

# DESIGN AND USE OF A DATABREAK DISPLAY FACILITY FOR PDP-8

E.R.F.W. Crossman, Ph.D.
Department of Industrial Engineering and Operations Research
University of California
Berkeley, California

## ABSTRACT

Our current research requires the use of contact-analog displays simulating the motion of a landscape as seen in perspective from a moving automobile or other vehicle. By employing geometrical approximations and table-look-up methods, it proved possible to generate only marginally adequate displays using the Type 34D display-controller. An improved display controller was, therefore, designed using: 1. hybrid computation,multiplying digital-analog converters,and summing amplifiers, 2. data-compression by storing 6-bit X and 6-bit Y deflections in a single word, and 3. adoption of databreak data-transfer under control of an automatic-sequence-plotting interface.

This interface permits highly-detailed, realistic contact-analog displays to be generated on line while still allowing central-processor time for performance evaluation. Hardware and software problems will be discussed.

## INTRODUCTION AND OBJECTIVES

The device described has been developed in connection with a car simulator used for automobile steering research. Its purpose is to provide a real-time display simulating objects and motions seen by the driver of a moving vehicle. This appears on a large C.R.T. screen* placed in the windshield opening of a stripped-down car body installed in the laboratory. The steering-wheel output is connected to an analog computer which solves the differential equations of the vehicle's motion and feeds analog signals representing angular heading, lateral position, and forward speed to the display system implemented by a real-time program running on a PDP-8. A "forcing function", representing the curvature of the simulated highway, is also supplied to the central processor via A/D converter channels. Ideally the experimental subject should be able to "drive" the simulator as readily as a real car. In this paper we are concerned with the display hardware and software required.

## CAPABILITY OF 34D DISPLAY-CONTROLLER

The first usable version of the highway display utilized a regular (Type 34D) display controller, with conventional-type programming. A software shift register was used to store topographical data for highway curves up to 1,280 ft. ahead, the highway being represented by a series of 256 numbers representing real-space planes 5 ft. apart. A "horizon" responded to heading changes only. Table look-up methods were used to compute the necessary perspective transformation for mapping the current topography onto the CRT screen, which was achieved by only a single multiplication (using EAE) per plane. Some 800 points could be plotted flicker-free, permitting traffic lanes to be indicated by a single point each,but no scenery or pavement detail could be included.

A very large fraction of program running time was spent in computation, and the "brightening ratio"--the fraction of time during which the CRT spot is intensified--was quite small (around 2%) yielding a poorly-lit display.

## DESIGN OF AN IMPROVED NON-AUTOMATIC DISPLAY INTERFACE (Mk. I)

Hardware developments aimed at improving display capability were therefore undertaken. It was decided to transfer the computations required for object-positioning and magnification from software to hardware, retaining pointwise image construction as a software function and packing both horizontal and vertical components of each image-point into a single 12-bit data word. This permits 2-dimensional objects of each desired type to be constructed from a unique data string in core and to be shown in any desired relative direction (i.e., screen position) and at any apparent distance (i.e., size) without digital computation. Vector generation using analog integration was rejected as being of little value for curvilinear images. No provision was made to rotate images.

Detail-generation and magnification were implemented jointly by a pair of 6-bit hybrid multipliers. Successive detail-words were loaded from AC by executing a newly defined IOT instruction. Horizontal and vertical magnification data were loaded via two standard (AA01A) D/A converters, while position data were contained in the two existing (34D) display-coordinate channels. Analog output to the CRT was provided by summing centroid-positions with

*An HP 1300A oscilloscope (8" x 10").

scaled detail-point positions in horizontal and vertical summing amplifiers. In other respects the Mark I interface functioned as a Type 34D controller.

This interface permitted landscape features such as trees, rocks, hills on the horizon, pavement surface detail, and other vehicles to be shown in cartoon form with little restriction on shape or density. Since the eye is tolerant of imperfections in compound curves such as occur in real objects, objects can be plotted convincingly in a 64 x 64 matrix, if one does not attempt to produce true geometrical forms such as circular or elliptical arcs.

The time saved by reducing the computation required for each point plotted permitted marked improvement in display realism for relatively small (c. $1,000) capital cost. However the display routine still occupied a large fraction of CPU time, and the brightening-ratio could only be raised to about 10%. A fully automatic version was therefore developed.

## DESIGN OF AN AUTOMATIC DISPLAY INTERFACE USING DATABREAK (MARK II)

Since image plotting now required only loading a single register from a single core location followed by spot intensification, an automatic loading and intensification sequence using the single-cycle databreak appeared feasible. This imposes the software restriction that detail points for a given object must be contained in sequential core locations, and also required an additional set of IOT instructions to control the automatic device. These include clearing and testing a flag to indicate the status of the automatic device, as well as supplying the databreak starting address and initiating display activity. Termination of the display cycle must be independent of CPU activity and was achieved by defining an end-of-file code and terminating the display sequence when this was encountered in the detail register.

The only direct operating penalty for making the cycle automatic was an overhead of one core location per object-type. The display flag, set on encountering the end-of-file code in detail register, was connected to the interrupt bus to free the CPU from need to monitor display activity. Compatibility with other databreak devices was achieved by assigning low priority; if Dectape or disc is in use the only result is to cause temporary dimming or flickering of the CRT display.

A description of the currently implemented (Mark II) version of the device, now termed the Databreak Display Interface, follows.

## DETAILED DESCRIPTION OF THE DATABREAK DISPLAY INTERFACE

The device permits the 2-dimensional image of an object to be generated anywhere on the CRT screen and enlarged or reduced ("zoomed") to any size. This is done by loading two data-words to locate the center of the image and two to determine its size. The image itself consists of a number of points arranged in a fixed pattern relative to one another and to the centroid. It is produced by automatically plotting a detail-string, that is a sequence of words obtained direct from core memory. The central processor is free for other work while the detail string is being plotted. Depending on the length of the string and the plotting rate in use, this may release from 5% to 95% of CPU time for non-display activity. When the detail string runs out, an end-of-file code inserted in core following the last point to be plotted causes CPU interrupt. The program is then responsible for setting a new centroid position and magnification, selecting a fresh detail string for the next object to be plotted and then initiating plotting action.

### (i) Spot Positioning

Assuming that the output terminals are connected to a CRT with symmetrical X and Y deflection amplifiers, and that a positive input voltage produces deflection upwards and to the left, the device implements the following pair of equations.

$$\begin{cases} X_p = X_o + (M_x \cdot D_x) \\ Y_p = Y_o + (M_y \cdot D_y) \end{cases}$$

where

$X_p$ , $Y_p$ = Horizontal & vertical components of spot position.

$X_o$ , $Y_o$ = Horizontal & vertical components of centroid position.

$M_x$ , $M_y$ = Horizontal & vertical components of image magnification.

$D_x$ , $D_y$ = Horizontal & vertical components of the image-point or vector measured relative to the current centroid.

The three paired input quantities are loaded into D/A registers under program control as follows.

The components of centroid position ($X_o$ , $Y_o$) are stored in 12-bit Centroid Registers (using the AA01A interface described in the Users Handbook).

C(AC) → $X_o$ on executing instruction DAL1 (61Ø1)

C(AC) → $Y_o$ on executing instruction DAL2 (61Ø2)

Centroid-position scaling is 12-bit 2's complement, i.e.

3777 → Rightmost centroid position
ØØØØ → Central centroid position
4ØØØ → Leftmost centroid position

The components of image magnification ($M_x$ , $M_y$) are stored in 10-bit Magnification Registers (using the 34D Display-control interface described in the Users Handbook).

C(AC2-11) → $M_x$ on executing instruction DXL (6Ø53)

C(AC2-11) → $M_y$ on executing instruction DYL (6Ø63)

Scaling is 10-bit (positive) binary, right justified, i.e.

1777 (AC2→11 = 1) → Full-size ($M_x$ or $M_y$ = 1.0)
1ØØØ → Half-size ($M_x$ or $M_y$ = 0.5)
ØØØØ → Zero-size ($M_x$ or $M_y$ = 0.0)

The components of the image-point vector ($D_x$ , $D_y$) are stored in two 6-bit Detail Registers.

C(SA + n) → ($D_x$ , $D_y$) under automatic control following execution of instruction DLG (6134)

where SA is the first core address of a string comprising points for the image to be displayed, and n is any integer up to 4Ø95. For further description of the plotting sequence see below.

Scaling for the image-point vector is 6-bit 2's complement, with the leftmost 6-bits producing $D_x$ and the rightmost 6-bits producing $D_y$ (see Figure 1), thus

3737 → Point at upper rightmost corner (3,3')
ØØØØ → Point at centroid (2)
4141 → Point at lower leftmost corner (5,5')
3741 → Point at lower rightmost corner (4,4')
4137 → Point at upper leftmost corner (6,6')

An image-point cannot be plotted at 4Ø4Ø (the extreme lower leftmost corner) since this is the end-of-file code. However 4Ø41 and 414Ø may be used. The deflections produced by a given image-point vector data-word are proportional to the current values of $M_x$ , $M_y$ .

Full image-point deflection equals half centroid deflection. i.e. 37ØØ in detail register produces the same deflection as 1777 in X centroid register.

### (ii) Plotting Sequence

The C(AC) at the time of executing instruction DLG (DDI. Load and GO; 6134) is interpreted as the starting address of a string of (6 + 6-bit) image-point words. These are obtained successively from core via the single-cycle databreak, loaded into the Detail Register, analog-converted and multiplied by the current values of the X and Y Magnification Registers, summed with the current values of the Centroid Registers and displayed by intensifying the CRT spot.

Sequential loading and intensification continues automatically at 13 to 25 µ-second intervals until the number 4Ø4Ø is encountered in the Detail Register. At this point the plotting action terminates, the DDI Flag is set, and CPU interrupt occurs if enabled. The DDI Flag may be cleared by executing instruction DDC (Clear DDI Flag; 6132), and its status tested by Instruction DDS (Skip on DDI Flag = 1; 6131).

In normal use the DDI Flag will be cleared at the time of initiating each fresh detail plotting sequence, using the combined instruction DCG (Clear DDI Flag, Load SA of Detail String, and Go; 6136). There is no instruction for terminating plotting action once started. If this is required the end-of-file code 4Ø4Ø (JMS 4Ø) may be placed anywhere in core. In its absence the device will display all of core repeatedly.

### (iii) Timing

As at present implemented, the following times are required for display generation.

| | |
|---|---|
| Fetch next image-point word via databreak & deflect CRT spot | 5 to 10 µs |
| Brighten spot | 5 to 10 µs |
| Wait for CPU to accept fresh databreak request | 3 µs upwards |
| Total Cycle Time = | 13-23 µs/point |

Waiting time is small except when using EAE instructions (but see below). The above timing sets an upper limit of about 77k image-points per second in continuous plotting. For objects defined by 15-point images, and assuming 30 µsec overhead per object, whis would permit some 4,444 objects to be plotted per second; at the 20 per second repetition rate which is required to provide flicker-free vision this allows 222 separate objects. Using the 64 x 64 matrix to the full, 4096[15] distinct objects can be defined.

For comparison, the 34D controller requires a minimum of about 50 µsec per point (depending on the amount of computation required to select and position successive characters) and can thus generate some 20-50 flicker-free characters, using the 6 x 4 format implemented by DECUS character-generation software. The 64 x 64 format provided by the DDI permits much more freedom for detailed construction of characters, but this is paid for by additional storage requirements.

### (iv) Lack of Compatibility with EAE Instructions

We have found that execution of certain EAE instructions while the automatic plotting sequence is running causes (with very low frequency) zeros to appear in the detail string in core memory. This produces image decimation, since points deleted reappear in the center of the image. This gradually destroys the image; users are therefore advised to avoid operating the DDI in multiprocess mode with programs using EAE.

### (v) Summary of Operating Instructions

Instruction set

| | | |
|---|---|---|
| DAL1 | 61Ø1 | Load X Centroid Register |
| DAL2 | 61Ø2 | Load Y Centroid Register |
| DXL | 6Ø53 | Load X Magnification Register |
| DYL | 6Ø63 | Load Y Magnification Register |
| DDS | 6131 | Skip on DDI Flag = 1 |
| DDC | 6132 | Clear DDI Flag (→ 0) |
| DLG | 6134 | Load Starting Address of detail string and initiate display sequence |
| DCG | 6136 | Clear Flag, Load Starting Address, and Initiate display sequence |

Data format

| | |
|---|---|
| X and Y Centroid | 12-bit 2's complement |
| X and Y Magnification binary | 10-bit (positive) |
| X and Y Detail | (6 + 6)-bit 2's complement |

End-of-file code

4Ø4Ø encountered in detail string

## MULTIPROCESSING SOFTWARE FOR THE AUTOMATIC DISPLAY FACILITY

Programming technique for the automatic display system dessribed above differs markedly from that employed with the Type 34D controller. To render display action independent of background CPU activity a communication file must be provided containing the following data for each of a specified number of objects or image-components currently being displayed--

X centroid
Y centroid
X and Y magnification (or separate data if desired)
starting address of detail string

Switch-bits may also be inserted at the (unused) bit Ø of the magnification datum locations, permitting unwanted objects to be blanked without reorganizing the whole display file. A display-maintenance subprogram, entered from CPU interrupt when the display-flag is set, iterates repeatedly through this file to provide a continuously refreshed display with brightening-ratio on the order of 80%. Changes of object selection, position and size are accomplished by the main CPU program inserting fresh data in the display communication-file at times independent of the display cycle.

This technique is effective with 10 or more image-points per object, below which point display refreshment begins to occupy a time large compared with total running time, causing reduced brightness; display refreshment will then also take an undue fraction of total CPU time away from the main program. In highway simulation this is an insignificant limitation, but for alphameric characters it might suggest unduly detailed character-drawing.

With 4 data per communication-file entry, the core requirement is 1 page per 32 objects in addition to the detail strings needed to generate object types. Complete objects such as characters could presumably be produced by combining several features each defined by a separate detail-string, but this possibility has not been tested in practice.

Using software of this type with the automatic display facility dessribed above, a highway display of 256 planes, equivalent to 1,280 ft of forward vision plus horizon, can be generated with highly realistic scenery and updated in real-time up to the equivalent to 100 mph, while still maintaining excellent display brightness. The program requires about 2 k of core.

## CONCLUSION AND SUGGESTED FURTHER APPLICATIONS

The device described permits efficient display-generation with high brightening-ratio while still leaving large amounts of CPU time free for non-display computation. The main penalty is use of additional core storage for display detail, and for a display communication file if multiprocessing is desired.

Apart from its primary purpose of permitting realistic contact-analog simulation of moving scenery, the inexpensive display system described would readily lend itself to online real-time character-generation displays for text-editing and typesetting. It would facilitate the implementation of graphical design systems, using normal or storage CRT's. Other potential applications include automatic analog waveform generation for acoustic displays, and picture-plotting with analog XY recorder

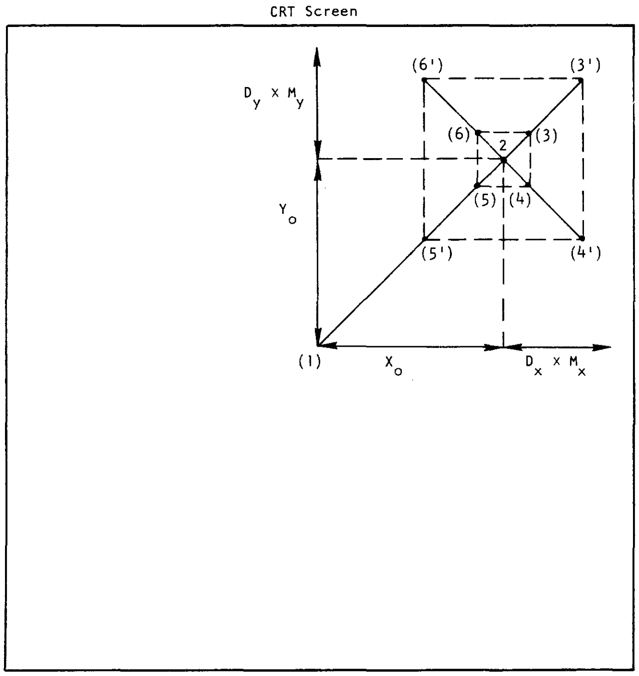
A picture-editor is currently being written to permit rapid image-generation and storage by the human artist using eyboard and/or light-pen and/or graphical tablet.

## APPENDIX

Example of a Program Using DDI

Display contents of core continuously (assuming that no location contains 4Ø4Ø). This may be used to examine all of core when program is halted. Any 7 locations may be used.

```
*n
SECOR,STA    /Set full-scale
DXL
DYL
CLA          /Set X and Y centroids to center of
             /screen
DAL1 DAL2
DCG          /Initiate display sequence at
             /SA = Ø , & clear flag
DDS          /Automatic display interface
             /continues to display core
JMP.-1       /Test flag; unless JMS 40 is en-
             /countered in core. In this case
             /halts at  n + 1Ø .
HLT
```

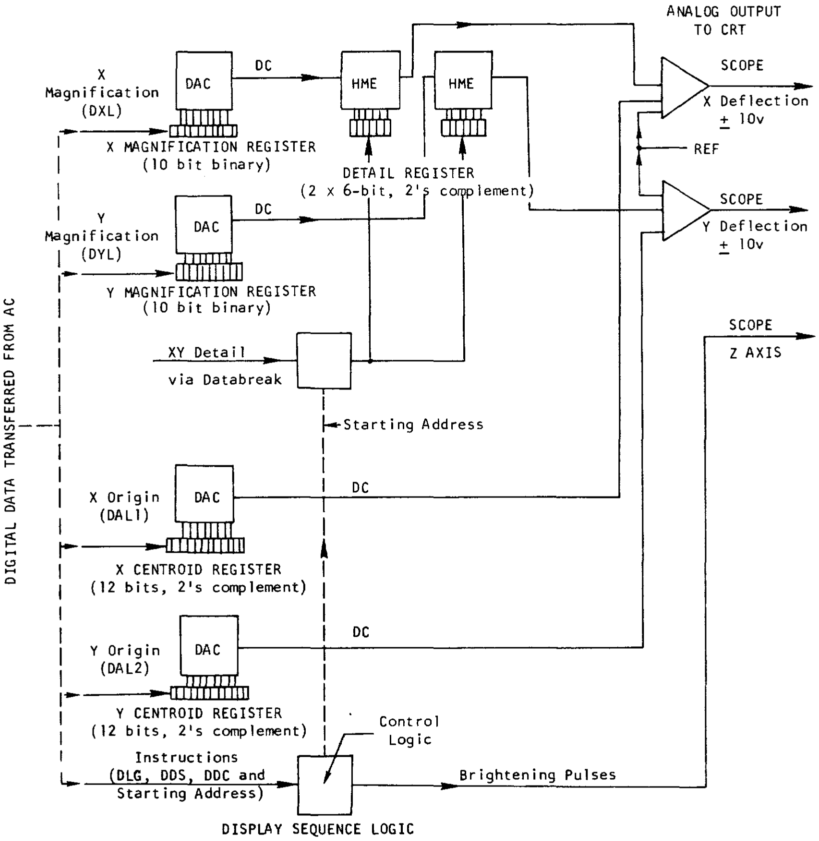


(1) Center of Screen.

(2) Centroid position set by $X_o$ , $Y_o$ with $D_x$ , $D_y = 0$ and/or $M_x$ , $M_y = 0$ .

(3-6) See Text.

Figure 1 Image Plotting with the Databreak Display Interface



DAC = Digital-to-Analog Converter

HME = Hybrid Multiplying Element

Figure 2 Hardware Schematic of the Mark II Databreak Display Interface

# FAST FOURIER TRANSFORM TECHNIQUES
# USING A DRUM FOR MEMORY EXTENSION

Ric C. Davies
Phillips Petroleum Company
Idaho Falls, Idaho

## ABSTRACT

A fast Fourier transform subroutine package which is FORTRAN compatible has been developed for a PDP-9 computer to transform any type of discrete data. A 128K RM09 serial drum is used to readily access and store the data during computation of the fast Fourier transform. A 339 display unit is used to display the original data and the transformed data separately or simultaneously for comparison. A paper tape punch option supplies the user with permanent copies of portions or of all the data.

## INTRODUCTION

The recent development of a fast Fourier transform (FFT) has brought forth new prospects in experimental work at this laboratory as it has in many other places. We have found that the PDP-9 with extended storage is capable of doing a FFT in a real time environment. In fact, the FFT written for the PDP-9, can analyze 8192 discrete data points and has capabilities of expansion.

Two main problems must be considered in the formulation of the fast Fourier transform to run on a PDP-9.

1. Core

The basic PDP-9 has 8K of core memory, but at least 32K of storage is needed for the data alone. For example, to transform 8192 data points 16384 core locations are needed for the real part and 16384 for the imaginary part of the complex result obtained from the FFT. This is because all calculations are done in single precision floating point (2 words/data point). To conserve time, a sine table is used which needs 4096 additional storage locations. Thus 36K words of storage are required to transform 8192 points.

2. Time

The PDP-9 with 1 μsec cycle time fits very well into a real time environment. However, the floating point package requires a considerable amount of time to calculate sines and cosines and to load and store the floating point accumulator. The teletype and the paper tape punch also take too much time to output any great amount of data.

## EQUIPMENT

The configuration consists of an 8K PDP-9 computer, a 128K RM09 drum, a 339 display unit, a high speed paper tape reader and punch, and a teletype.

## DEVELOPMENT

A 128K word drum extends the computer storage to the size needed to transform 8192 points. In order to use the drum as an extension of core memory, an algorithm was developed which transfers data as few times as possible and is still compatible with FFT calculations. The FFT requires $N = 2^{\gamma}$ data points, where $\gamma$ is an integer. The tracks on the drum contain 256 or $2^8$ words. Because the FFT and the drum both use the form $2^{\gamma}$ , the drum transfers can give the FFT the exact number of points needed for a calculation. Figure 1 shows the procedure used by the FFT to calculate the Fourier transform of $2^2$ data points. The first column is the original data. Each element or node is entered by a dashed and a solid line. The solid line brings a quantity from one of the elements in a previous column, multiplies the quantity by a determined value, and the product is added to the quantity brought by the dashed line. Notice that a pair of elements of one array is used to calculate the same pair of elements of the next array. This is true for any size of an array and for the complete transform. Now suppose that each of the elements in Figure 1 consists of 256 points. A pair of sets of 256 points of one array are used to calculate the same pair of sets of 256 points of the next array. This means that two sets of 256 can be read in from the drum and used to calculate the same two sets of 256 points for the next array. Once the sets of one array are used to calculate the sets of the next array, they are not used again; therefore, the new sets are written on the drum in place of the old sets.

The algorithm determines which sets of 256 data points are to be read in from the drum next. Figure 1 gives the steps used in the algorithm. If N is the number of data points, NBLK = N/256 is the number of groups of 256. Let I be the particular array being computed (the left most array is 1), then M = NBLK/2**I is the incremental difference between the two sets of a pair that are to be read from the drum. For example, if each element of Figure 1 consisted of 256 points; N would equal

1024, NBLK would equal 4, and M would equal 2 for I equal to 1. The element paired with element 1 would be element 3, since 1 + M= 1 + 2= 3. When M becomes less than 1, set NBLK = 256 and perform the FFT on the elements within each set of 256. The transform is done when M becomes less than 1 for each set of 256.

A sine table is used to save time spent in calculation of sines and cosines. The first version of the FFT subroutine took about 17 minutes for 1024 points, because sines and cosines were calculated for each data point. The table reduced the time to transform 1024 data points to about 10 minutes.

The table is formed by dividing 2π into N (number of data points) equal angles and taking the sine of the angles within the first quadrant. The table need contain only the sines in the first quadrant. The rest of the sines are obtained from those of the first quadrant by determing which quadrant the desired sine is in and giving it the correct sign (+ or -). The cosines are found by reading the sine table in reverse order and assigning the correct sign. The table saves time, but not enough.

Further analysis of the procedures used in the program showed that a great amount of time was being wasted in loading and storing the floating point accumulator of the FORTRAN package. The routine to load and store the floating point accumulator checks to see if single or double precision numbers are being used. The routine then does the appropriate conversion in order to do all calculations in double precision. Because our numbers are all single precision, a special routine was written to load and store the floating point accumulator. The time to transform 1024 points was reduced from 10 minures to 2 minutes.

Figure 3 shows the Fourier transform of the gamma-ray spectrum shown in Figure 2. In our applications, the FFT is used as a smoothing algorithm. By eliminating some of the higher frequencies of the transform and doing the inverse FFT, the gamma-ray spectrum is smoothed, i.e. statistical fluctuations are removed. Figure 4 shows the gamma-ray spectrum and its transform displayed together.

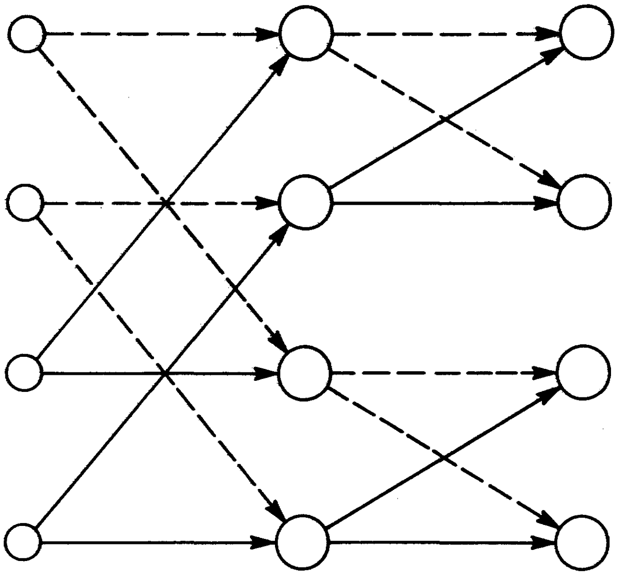
A 339 display unit is used as an output medium because the teletype and the paper tape punch are too slow for our applications. The original data and the transform are displayed separately or simultaneously in a 1024 word core buffer by interchanging the data from drum to core. This gives a fast and efficient way of comparing the data (see Figure 4). Hard copies are obtained by taking pictures of the scope or by punching out desired portions of the data.

## CONCLUSION

The PDP-9 with a drum is adequate for doing fast Fourier transforms in a real time environment. The drum gives the additional storage needed to process large amounts of data. The 339 scope gives immediate output and limits the amount of information that is needed in hard copy.

## ACKNOWLEDGMENTS

I would like to thank W. W. Black for his assistance in the initial setup of the FFT routine and for his continued interest.



$N = 2^{\gamma}$ pts.

NBLK = N/256 = # groups of 256

I = array

M = NBLK/2** I = difference between corresponding blocks.

Figure 1 Fast Fourier procedures

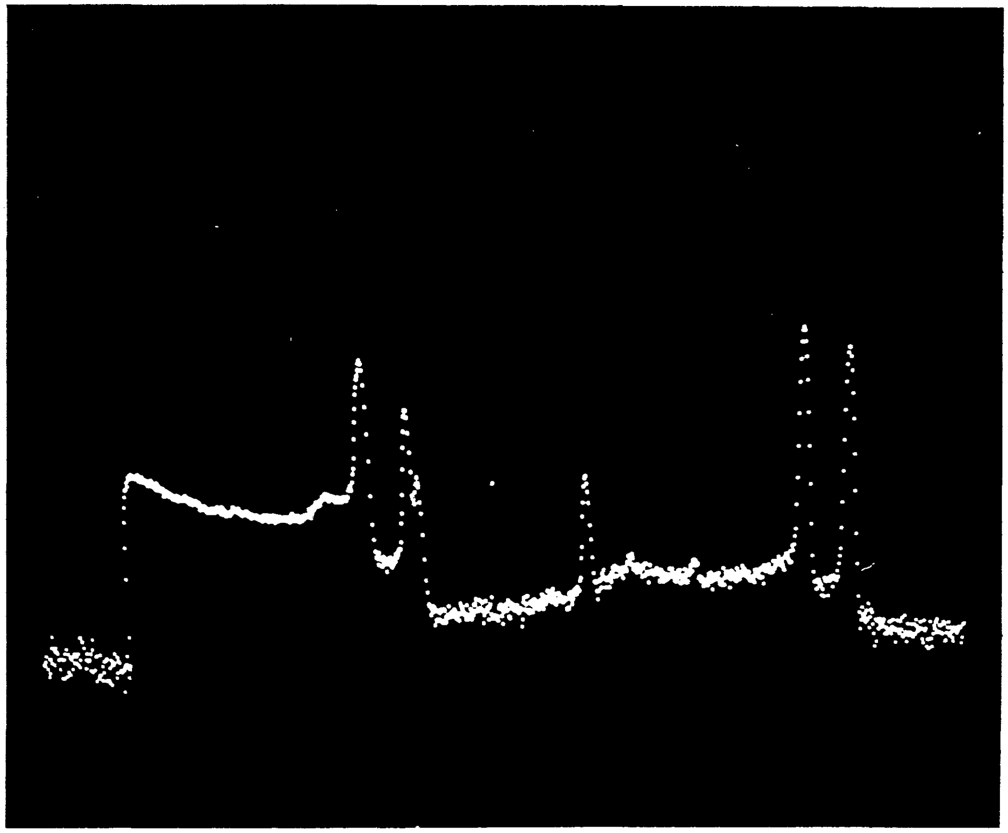


Figure 2 Gamma-ray spectrum

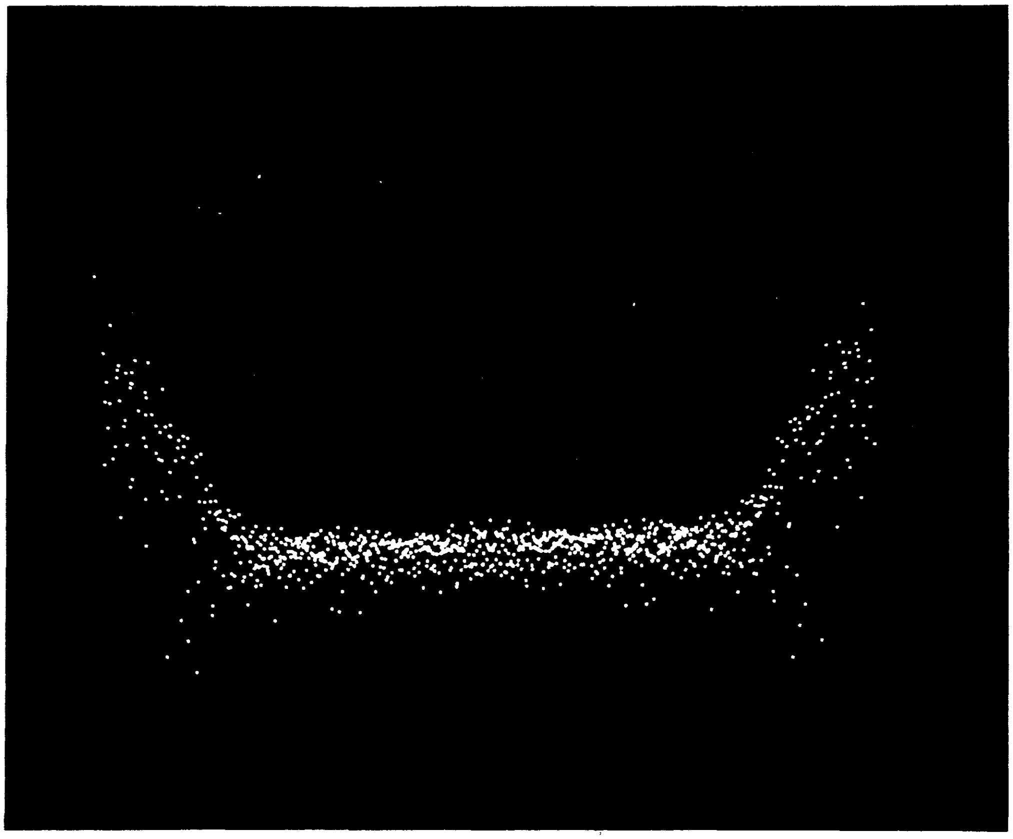


Figure 3 Fourier transform of gamma-ray spectrum

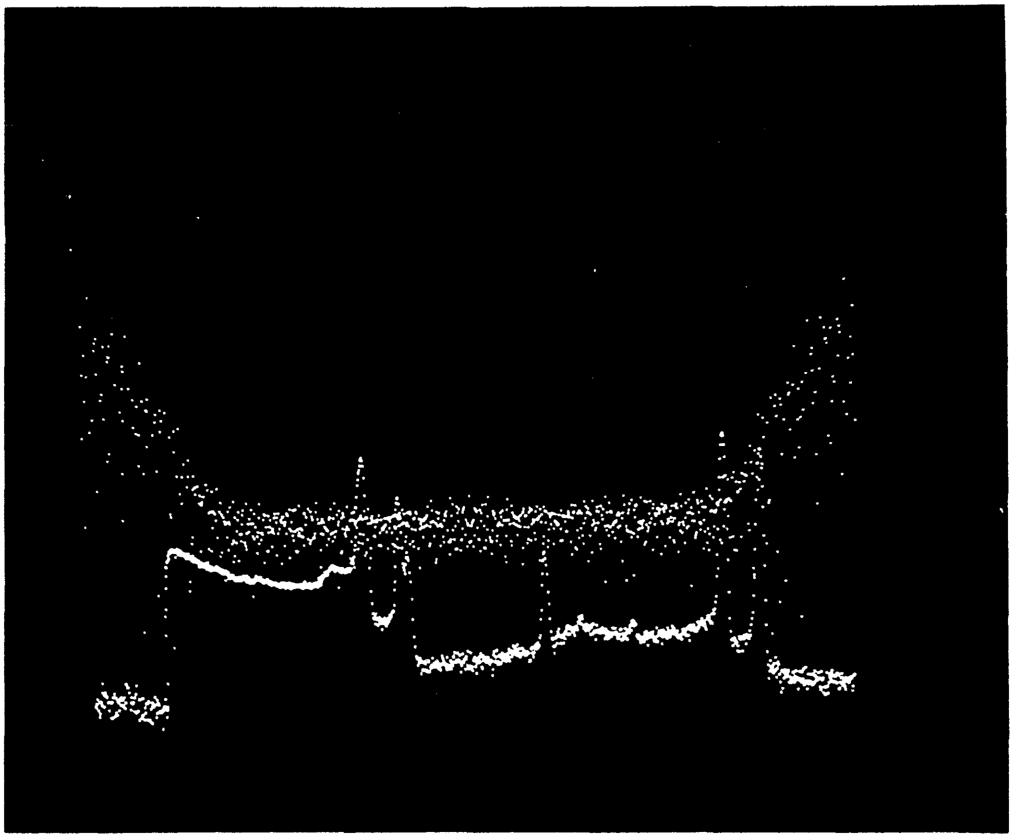


Figure 4 Gamma-ray spectrum and its transform

# INTERACTIVE DEBUGGING UNDER SIMULATION

David J. Waks
Applied Data Research, Inc.
Princeton, N. J.

## Abstract

Debugging programs for small computers is hindered by the lack of adequate memory, proper hardware, and peripheral equipment on the machine on which these programs are ultimately to be run. This paper proposes that comprehensive simulators for small computers be developed explicitly for interactive debugging and be run on larger computers with adequate memory, peripherals and hardware to completely check out the program written for the small computer. This technique has been used at ADR for over two years in debugging large, real-time PDP-8 programs on a PDP-7.

## I. The Debugging Problem

Getting a program from its pencil-and-paper form to a working program on a computer requires the application of assembly and debugging techniques. Much has been written about assemblers; this paper will deal primarily with the often neglected subject of debugging.

It will be worthwhile to start with some of the history of assembly and debugging techniques. Before operating systems came along, this was usually done by assembling the program from cards or paper tape, producing a separate object program deck or tape; then loading the object program into memory and checking it out using the console of the computer as the primary debugging aid, stepping through suspicious program sequences and using "break point" switches, tested by the program, to stop the program at selected places. These manual techniques were aided by octal patches, which permitted changing the program without reassembling; core dumps, which dumped the entire contents of memory on to the line printer to present a static picture of the state of the machine at a given time; and trace routines, which provided a dynamic picture of a small number of registers. Later, these were supplemented by selective trace and dump techniques which provided the programmer with the ability of selecting suspicious portions of the program and dumping selective portions of memory as pre-selected places were reached during execution.

When operating systems were introduced on larger computers, and "closed-shop" operations (in which the programmer is not present while his program is being run) became the rule, we started using so-called "load and go" systems in which the various routines of the program were compiled or assembled, loaded into memory, and executed without any possible intervention from the programmer. In this context, core dumps became almost the only technique available to the programmer - with the possible addition of selective dump and trace techniques, at a high cost in machine time. The programmer was forced to preplan the debugging required in each run, since no provision was made for him to intervene during execution. The argument in favor of "closed-shop" over "open-shop" operation seemed to be that open-shop operation uses the programmer better; closed-shop uses the computer better; on large machines, at least, closed-shop always won.

When small computers came along and became popular, we reverted to console debugging, which had always seemed the best approach to programmers. Now, however, we had the added problem that core dumps (the most effective debugging technique) could rarely be obtained in the absence of a line printer. So debugging programs, such as DDT, were written to permit the programmer substantial interactive capability while debugging. With the advent of dedicated small-computer systems, the normal approach seems to be to debug a program on the computer on which it will eventually be the dedicated program. But debugging programs on small computers generally raises these three major problems:

1. Interference between the debugging tools and the program being debugged. This interference takes two major forms - I/O and memory interference.

The communications devices used by debugging systems such as DDT are inevitably the same ones used by the program being debugged. Thus the teletype in DDT is used by both systems. Its state (teletype flag up or down) is indeterminate when moving from DDT to the program being debugged and back. This interaction becomes a particular problem when attempting to debug programs designed to operate under interrupt. DDT does not operate properly in such a context. Moreover, it isn't really possible to design a DDT-type debugging system which will.

The object program typically cannot afford to release as much space as DDT requires. This can be partially compensated for by debugging the program a section at a time; but this is not really satisfactory since system integration, with no room left for DDT, usually requires the most intensive testing to uncover bugs - and the bugs, once encountered, are typically the most difficult to pin down and fix.

2. Peripherals - For all the nice features offered by interactive systems such as DDT, a core dump has always been the best means to track down a large class of bugs - those which have caused the program to enter some forbidden state; most often by executing data as instructions or by clobbering an

instruction by referring to it as a data operand. Moreover, a core dump provides the means for the programmer to get away from the machine environment and find a quiet place to sit down and think.

But most small computers lack any kind of output printer other than a Teletype, which is a singularly unattractive device for producing a core dump (note the printing of a formatted core dump of 4096 words, including decoding of instructions to make them reasonably readable, takes at least an hour on a Teletype - and it won't survive long with such usage). In addition to the lack of a line printer, most small computers, particularly in dedicated applications, lack the high-speed paper tape reader so critical in loading (and, particularly, re-loading) a program and some sort of bulk storage such as DECtape, magnetic tape, or disc which can save a great deal of time by providing checkpoint images of core at the end of a debugging session.

3. Console - The consoles of all small computers - and the PDP-8 family is a particular case in point - are very unsuitable for debugging. In attempting to debug a program, one desperately needs the ability to stop a program as soon as it enters some forbidden state (and before it clears all of core to zero); one desperately needs to find out just which instruction is referencing location 1234. Most small computers do not provide any provision on the console for setting such address stops. They also rarely provide detection of invalid instructions.

A second problem is that there are many registers available to the program which are not visible to the user (for example, the teletype flags and associated buffer registers). When debugging, this information is often very valuable. Besides, even those registers that are displayed are impossible to change without clobbering the program state and making it impossible to continue execution.

## II. Simulation as a Solution

It should be clear, then, that small computers leave a lot to be desired as proper debugging environments. What alternative is available? We believe that the best way to debug programs written for small machines is under interactive simulation on a larger machine. Simulation without interaction forces the programmer to preplan his bugs; finding each bug requires a shot at the machine with an interval of several hours a day between. Interactive simulation, on the other hand, permits the experienced debugger to ferret out his bugs substantially faster than either console debugging or in a closed-shop situation. With the proper debugging tools, in an environment specifically created for debugging, his job can be made much easier.

About two and a half years ago, faced with a large quantity of undebugged PDP-8 code, we designed a simulator for the explicit purpose of debugging our PDP-8 programs. The simulator was initially built to simulate an 8K PDP-8 with high-speed paper-tape reader and punch and teletype. It was designed to faithfully simulate all program-accessible registers, simulating these registers in such a way that a program operating under the simulator could not tell, except by timing, that it was not running on a PDP-8. The simulator was written to run on the PDP-7/8 configuration at our Research Computing Center (configuration shown in Figure 1). The simulator was debugged by running the DEC MAINDEC's until the program ran properly - which turned up a number of bugs in the PDP-8 User's Manual as well as those of the simulator.

Perhaps the best way to describe the simulator is to illustrate it by a conversation - a transcript of a teletype printout on the PDP-7. Numbers to the left of the teletype lines correspond to notes below. The conversation and footnotes are contained on the following pages.

As we have seen, the simulator provides the debugger with the following facilities: (a) set PDP-8 memory either in its entirety or in part to any desired value; (b) load programs in whole or in part from paper tape or DECtape; (c) start and stop programs; (d) examine and modify any hardware register or memory location; (e) set and delete any number of instruction and/or data breakpoints in PDP-8 programs; (f) transfer memory between the PDP-7, the PDP-8 and other PDP-8/I's and PDP-8/L's; (g) obtain a formatted core dump on the line printer of any or all of memory; (h) turn on and off cycle counters which permit the timing of PDP-8 programs.

The PDP-8 simulator has been substantially expanded since its original creation. The current version fully supports an 8K PDP-8 with teletype, high-speed paper-tape reader and punch, four DF-32 platters (simulated on DECtape), and a line printer. The PDP-8 interrupt system is fully simulated, including the vagaries of interrupt operations on a multi-field PDP-8. Virtually no restrictions are placed on the programmer during coding and debugging. The simulator is designed in such a way that additional devices can be added quite easily; if a hardware-software implementation project requires custom hardware on a PDP-8, code to simulate the device is appended to the simulator so that the software for the system can be debugged before the hardware is built.

The simulator has been used to debug programs ranging from an accounting system for the PDP-8/S, a real-time mass spectrometry system (described in a previous DECUS paper[1]), The Engineering and Scientific Interpreter[2], and a complete television station Central Control Room automation system.

Many people by now have contributed to the development of this simulation approach. Particular credit is due to the following members of the Control Systems Division - Michael P. McCarthy, who helped conceive the idea; Roger Frye, Robert Supnik, and Barbara Slaughter, who with the author are responsible for the code.

## References

1. M. L. Cramer and D. J. Waks, "Data Acquisition and Analysis of High-Resolution Mass Spectra in Real Time," DECUS Proceedings, Fall 1967.

2. D. J. Waks, "ESI-Conversational-Mode Computing on the PDP-8/S," DECUS Proceedings, Fall 1966.

## Notes on the Conversation

(1) Our PDP-7/8 operating system, based on DEC's DECSYS, uses GA("Go Ahead") to mean "What program do you want me to load?".

(2) "I want to load SIM8!".

(3) The simulator has a large number of named locations which operate á la DDT - they are "opened" by typing the name of the location, and may be changed by typing a new value when they are open. The contents of FILLER are used in the FILL command following.

(4) Fill all of core with the contents of FILLER (0000).

(5) Set locations 1000 through 1777 to 7777.

(6) Load into memory, from DECtape, the binary program file called ESIX.

(7) Start ESIX running at location 5400.

(8) This is ESI running under simulation.

(9) We decide to stop the program execution so we type WRU, which simulates the PDP-8 "stop" button. This is done in such a way as to prevent interference between the dual uses of the teletype.

(10) To request the contents of the PDP-8 console panel registers.

(11) We decide to investigate the loop in operation at the time of our WRU.

(12) We observe that the loop is waiting for the keyboard flag; investigating it we discover that it is currently zero.

(13) On the other hand, some device flag must be up since QANYINT (the logical OR of all device flags) is 1.

(14) The teleprinter flag is a 1. Let us see what happens if we change it to a 0.

(15) GO simulates the PDP-8 "continue" button.

(16) We hit a few keys and nothing seems to be happening so we type WRU to investigate the situation.

(17) The program is waiting for the teleprinter flag, which we set to 0 earlier. Let's set it back to 1 and see what happens.

(18) Set an instruction breakpoint at location 200 (used in the execution of the SET verb).

(19) The breakpoint occurs before the instruction is executed. We can examine the instruction before it is executed.

(20) Reset the breakpoint.

(21) Set "data breakpoint" in all of bank 1.

(22) Reset all breakpoints.

(23) Transfer the contents of simulated PDP-8 memory to the on-line PDP-8. Simulated memory is not affected.

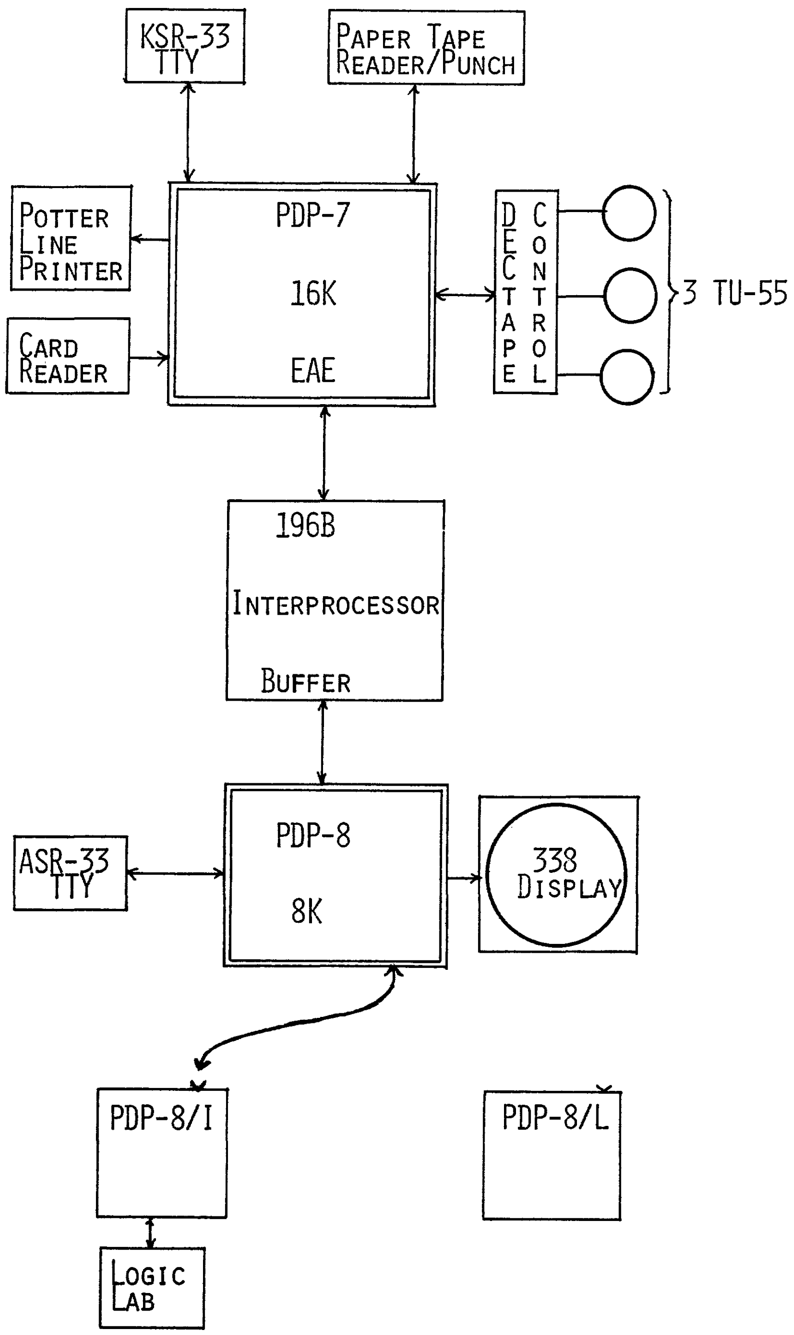
(24) We run into trouble operating on the on-line PDP-8. We bring memory back from the PDP-8 to the PDP-7.

(25) We dump all of core on the line printer, formatted for ease of reading.

(26) We dump the entire contents of simulated memory and all simulated registers on to DECtape so that we can correct the program when we come back after studying the core dump.

```
1    GA
2    SIM8!
3    FILLER/ ØØØØ
4    FILLALL/
     FILLER/ ØØØØ 7777
5    FILL/ 1ØØØ-1777

6    TLOAD/ESIX!
7    START 54ØØ/

8    ←TYPE SQRT(2).
        SQRT(2) =          1.414214
     ←
9    "WRU" AT Ø6Ø32
10   PANEL/ DBR = Ø  IBR-PC = Ø6Ø32  L-AC = Ø ØØØØ
11   6Ø32/ 6Ø31
     Ø6Ø33 \ 5232
12   KF/ Ø
13   QANYINT/ 1
     RF/ Ø
     PF/ Ø
     LPF/ Ø
14   TF/ 1 Ø
15   GO/
16   "WRU" AT Ø42Ø1
     PC/ Ø42Ø1
     DBR/ Ø
     AC/ Ø324
     L/ 1
     42ØØ/ 1715
     Ø42Ø1 \ 6Ø41
     Ø42Ø2 \ 52Ø1
     Ø42Ø3 \ 6Ø46
17   TF/ Ø 1
     TYPE "NOW ESI IS RUNNING".
     NOW ESI IS RUNNING
     ←
     "WRU" AT Ø6Ø32
18   BKPT 2ØØ/
     GO/
     SET A=Ø.

19   INST BKPT AT ØØ2Ø1
     2ØØ/ 11Ø5
     GO/
     ←
     "WRU" AT Ø6Ø33
20   RESET 2ØØ/
21   DBKPT/1ØØØØ-17777

     GO/
     SET A=2+2.

     DATA BKPT AT Ø6275 IN 1Ø33Ø
     GO/

     DATA BKPT AT Ø6313 IN 1Ø33Ø
22   RESET/1ØØØØ-17777

23   SEND/
24   RECEIVE/
25   DUMPALL/
26   CHKPNT/
     6D
```

ADR RESEARCH COMPUTING CENTER



# AN EXECUTIVE FOR A REMOTE INTERACTIVE GRAPHICS TERMINAL

C. G. Conn, P. T. Hughes
Computer Sciences Corporation
Huntsville, Alabama

## ABSTRACT

This paper describes the development of an executive system for a PDP-9/339 used as a graphics terminal remoted to a triple processor UNIVAC 1108. It includes the design of a higher-level interactive programming language which is processed interpretively by the executive system. This language allows the programmer to monitor, direct, and respond to operator actions at the scope and to communicate with the terminal hardware or software. This executive system handles all I/O, interrupts, allocation of free storage, tracking, and display file management.

## I. INTRODUCTION

The executive system described in this paper was designed on an 8K PDP-9 with two DEC tapes and a 339 Display. The computer is connected through a 637 communications interface to a triple-processor UNIVAC 1108 over a voice-grade phone line. The development of this prototype executive was undertaken as a feasibility study for the design and implementation of advanced graphic display terminal software.

Design criteria for the system included:

1. Ability to define and manipulate graphic data structures;

2. High level of program-operator interaction with good response time;

3. Machine-independent interactive graphics programming language with maximum flexibility; and

4. General purpose, readily expandable executive system.

## II. LANGUAGE STRUCTURE

As an interactive programming language, IPL is structured to permit a dynamic exchange between display operator and program under execution. It was designed for use as a general purpose graphics language, independent of the display hardware available. Syntactically a higher-level language, IPL relieves the programmer of the usually laborious programming task associated with graphics applications. At the same time, it retains a level of flexibility comparable to assembly language with respect to program-operator communication. The language is generated in the central site computer and executed interpretively at the remote terminal.

IPL is a simple, event-oriented grammar, consisting of structuring rules, instruction codes, and the logical operators AND, ØR, and NØT. Instruction codes are of two types: conditional, used in forming Boolean strings; and action, used in performing tasks. A source program consists of a series of statements followed by an END command. Each statement consists of an optional conditional string (to be evaluated as a Boolean expression) preceded by IF and terminated by THEN, and an action string terminated by a period. Any statement may be preceded by a statement number to be used in a logical transfer action command or to establish a logical register for arithmetic operation. The general format of a statement is:

n IF conditional string THEN action string.

If the conditional string evaluates TRUE, the actions in the action string are executed, otherwise they are ignored. If the conditional string is omitted, the actions are unconditionally performed.

The conditional string consists of any number of conditional instruction codes separated by logical operators. The codes are processed from left to right with no operator hierarchy inherent. The action string is also composed of any number of action instruction codes separated by a concatenation operator. The action codes are processed left-to-right in order of occurrence. Execution of a statement terminates upon detection of a period.

Initially the hardware interface to the central site computer was unavailable. In order to develop and checkout the IPL processor, a teletype was used to simulate IPL program input. To reduce preprocessing (translation of the source code to interpretive format) and typing time, an abbreviated syntax was adopted in which punctuation replaced operators and string terminators. Sematically, the language is the same. For example, a typical statement such as

n IF c AND c ØR c ØR c AND NOT c THEN a a a.

appeared for teletype input as

n c, c/c/c,-c = a,a,a.

where

n: statement number
c: conditional instruction code
,: logical AND operator (conditional) or concatenation operator (action)
/: logical ØR operator
-: logical NØT operator
=: conditional string terminator
a: action instruction code
.: statement terminator

There are at present 100 instruction codes incorporated in the language: 84 action, 16 conditional. The codes are grouped according to function in the following classes:

1. Arithmetic - performs basic computational functions.

2. Auxiliary - performs logical transfer of program control, non-sequential single-statement executions, establish state parameter values, and assorted housekeeping functions.

3. Character - performs basic character manipulation within text strings and handle teletype input.

4. Definition - defines basic display elements: point, line, etc.

5. Light button - defines, displays, and manipulates light buttons.

6. Light pen - enables and disables light pen and control of associated functions (blink, copy, move, scale, etc.).

7. Positioning - controls tracking and point (screen location) designation.

8. Response buffer - controls entries into the buffer and transmission to central site computer.

9. Text-editing - controls cursor and text (line) manipulation.

The instruction set is open-ended. Because of the interpretive design of the processor, new instruction routines are very simple to add.

Features intrinsic to the system include a tracking routine, cursor, push-down program label stack, and automatic manipulation of display items through a bit mask. Tracking is done with a displayed cross and the light pen. The position of the tracking cross is always available to the program. Operator repositioning of the cross can be programatically constrained and/or tested. The cursor is a unique character which may be positioned by the program and/or by the operator's use of the physical push buttons. Its principal use is in text-editing and character manipulation. The push-down, pop-up label stack provides automatic alternate execution paths somewhat like an ASSIGNED GO TO in FORTRAN. Utilizing this feature, the programmer may establish a number of execution levels within the program and provide for response to operator actions without regard to the current level of execution. At any given time, the top entry on the stack automatically reflects the level of program execution. In addition, a prompting message is associated ~~with~~ each stack entry providing a method of program communication to the operator. Since this message is displayed on the screen whenever the entry is brought to the top of stack, it can be used to:

1. Give the operator instructions;

2. Inform the operator of the level of program execution; and

3. Verify to the operator that he is performing the desired function.

A 12-bit mask associated with each display item controls functions commonly desired with reference to a display item; i.e. hide, delete, blink, copy, move, etc. Identification information is associated with each item defined for the display. This ID includes a definition mask, each bit of which corresponds, in a set format, to one of the above functions. Setting a mask bit to one enables the associated function for that item. In addition, a state mask variable is maintained whose value may be set by the IPL program. When an item is picked with the light pen, the definition mask associated with that item is AND-ed with the program state mask variable; functions associated with the bits set as a result of the logical product are automatically performed on that item. In this manner, an item definition mask establishes a set of permanently allowable manipulations for that item, while the program state mask provides dynamic control over the functions to be performed by the system at any given time. Other mask functions allow recording of item identification in the response buffer, designation of an item as a light button, and control of item light-pen sensitivity.

Use of the above features together with the IPL instructions which control them provides the programmer with a high degree of interaction and program flexibility in design of a graphics application.

## III. EXECUTIVE STRUCTURE

The terminal executive is designed to execute interpretively an IPL program; to maintain a display file structure and process operator light pen, push button, and teletype actions as dictated by the IPL program; and to handle communications with the central site computer. Figure 1 illustrates the communications paths between the terminal and central site. IPL source code is pre-processed in the UNIVAC 1108 and transmitted to the PDP-9 under control of the user's FORTRAN program. Buffers containing the data bases for figures to be displayed may also be transferred to the PDP-9 by the FORTRAN program. This data after being converted to 339 display format is inserted directly into the display file by the display manager. Response buffers are transmitted from the PDP-9 to the 1108; this buffer is the only method of communication from the IPL program being executed in the remote terminal to the central site computer.

The terminal system is interrupt-driven and as seen in Figure 2, consists basically of the idle loop, the interrupt processor with its associated handlers including the IPL interpreter, the display file manager, and the dynamic storage allocation program. The display file is composed of entries (obtained from free storage) which are forward and backward chained. Permanent file structure consists of the header and trailer blocks as illustrated in Figure 3. System display features such as the cursor, stack prompting message, and amount of free core remaining are contained in the header. The trailer consists of a STOP code and the transfer command (JUMP) to the top of the file. Entries are forward-chained through the display JUMP commands; each entry contains the identification information for the displayed item and a subroutine call (PJUMP) to the buffer containing the commands to display the item (see Figure 4). An item is positioned relative to the preceding one by the x- and y- positioning commands in the entry unless the clear co-ordinate, clear sector bits command is included in the mode word. In the latter case, the item is positioned (x and y) from the origin. The STOP/BLINK word is used to blink an item or to provide fast pen response. It normally contains a BLINK OFF command.

The STOP code at the end of the display file provides an interrupt to the mainframe. The IPL interpreter gains control to execute the IPL code string as a result of this interrupt, if since the last interpretive pass:

1. The tracking cross (if active) has been repositioned;

2. An item has been picked with the light pen;

3. A key has been hit on the teletype; and

4. A light button is down (state is set on).

Normally, interpretive control is possible at the end of each display cycle (single execution of the display file). By use of appropriate IPL instructions display STOP codes can be inserted in each display file entry; the interpreter may then assume control before execution of each displayed item rather than only at the end of the entire file. For large display files with many entries this latter technique provides quicker tracking response.

The interpreter itself controls the tracking routine and executes the IPL code string. The code string is processed one statement at a time:

1. The conditional string is evaluated with the Boolean result stored in a state parameter; if no conditional string is present, the state parameter is set to TRUE.

2. If the state parameter is set to TRUE, the action string is executed; otherwise, it is ignored.

3. The next statement is retrieved for processing in the above manner.

Interpretive execution terminates upon detection of the IPL STOP instruction code. Instruction codes appear in the code string as addresses of the routines performing the corresponding functions. An instruction code is executed by transferring control to the address contained in the code string. The interpretive control logic has common return points for conditional and action instruction routines. Conditional routines return the Boolean result of evaluation in the accumulator. To allow unlimited nesting of non-sequential single statement executions, a push-down execution stack of program address pointers is built by the IPL control transfer commands. Upon detection by the interpretive control logic of an IPL statement terminator, control is transferred to the code string address at the top of the execution stack; this address entry is then removed from the stack. If the execution stack is empty, sequential statement processing resumes. A logic flowchart of the interpretive control program is given in Figure 5.

Three interrupt handlers deal with operator actions at the remote terminal; light pen strike, teletype keyboard input, and physical push buttons. To avoid multiple light pen strikes, the manual interrupt button must be depressed before the system registers a strike. Interrupts resulting from light detection by the pen are ignored until the manual interrupt occurs. Automatic functions called for by the display item bit mask (See Section II) are performed in the light pen strike interrupt handler. A teletype interrupt inputs a single character from the teletype keyboard buffer provided keyboard input has been enabled in the controlling IPL program. Six-bit characters are packed three per word into a character buffer. Instructions may be input and dynamically executed from the character buffer under control of the IPL program. The physical push-buttons are used to move the displayed character cursor. As long as the button remains down, the cursor moves at a standard rate in the direction indicated by the button.

The 637 communications handler controls all full-duplex transmission for the PDP-9. Eight-bit characters are input and output on an interrupt basis allowing concurrent execution of an IPL program. As shown in Figure 2, IPL code strings and data buffers are received; response buffers are transmitted. Upon receipt of a new code string, the old one (if any) is flushed from the system. The address of the new code string is made available to the interpreter and an initial execution pass is forced. The display file is <u>not</u> cleared. Data buffers are translated into display commands. The identification information associated with the data buffer is used to construct a display file entry (see Figure 4) which is inserted in the display file by the display file manager.

The response buffer may be transmitted to the central site computer as a result of an explicity IPL command or a buffer over-flow condition. Transmission clears the buffer; if a buffer over-flow condition occurs, the item causing the overflow is not transmitted but is stored in the cleared buffer after transmission has been initiated.

The standard transmission sequence consists of an output request containing the ID and length of the data buffer to be transferred followed by the data buffer itself. The data is not transmitted until the request is acknowledged by the other computer. A "no acknowledge" received after transmission of a data buffer means a parity error was found by the receiving computer and the data buffer is retransmitted.

Communications are controlled using three chains:

1. Receive: Composed of output requests from central site. Each entry contains the request ID, the address and length of a buffer obtained from free storage for the data to be received.

2. Transmit-active: Composed of PDP-9 output requests, "acknowledges" and "no acknowledges". Transmission occurs from the top of this chain. Requests for output to the central site are placed on the end of the chain as they occur.

3. Transmit-inactive: Composed of PDP-9 output buffers which have been transmitted. They are saved pending receipt of an "acknowledge" from the central site computer.

For the format of chain entries, see Figure 6. The 637 handler has two main section - Receive Control (RCØNT) and Transmit Control (TCØNT). As output requests are received from central site they are placed on the receive chain and acknowledged. When a data buffer is received it is stored in the buffer address contained in the receive chain entry with matching ID. The entry is removed from the receive chain when the data buffer is successively received without parity error. The type code in the request designates the executive routine which is to receive the buffer; i.e. an IPL code string goes to the interpreter. Control buffers; i.e. "acknowledge" or "no acknowledge", received from central site are processed by:

1. "Acknowledge" to output request - transmission of data buffer on top of transmit-active chain is initiated.

2. "Acknowledge" to data - entry removed from transmit-inactive chain and data buffer returned to free storage.

3. "No acknowledge" to data - entry transferred from transmit-inactive to transmit active chain.

TCØNT transmits the top entry of the transmit-active chain. Following transmission, if the entry is an output request it is converted to a data entry and left at the top of the chain. Transmission of the data will be initiated by receiving an "acknowledge" to the output request from the central site. If it is a data buffer, the entry will be transferred from the transmit-active chain to the transmit-inactive chain after the data buffer itself has been transmitted.

## IV. CONCLUSION

At the present stage of development, the system as described meets the design criteria outlined in Section I. The graphic support software is operational; final checkout of the transmission handler is dependent on the completion of the EXEC VIII operating system by UNIVAC.

A UNIVAC 1558 will be interfaced to the PDP-9 to improve terminal display capability. The modularity of the executive system will allow this addition with minimum modification to the system program; i.e. coding of another display manager and expansion of the interrupt routine. Because of this modular construction and the interpretive design of the IPL processor, both the terminal executive and the IPL language can be expanded as necessary in response to application program and hardware requirements.

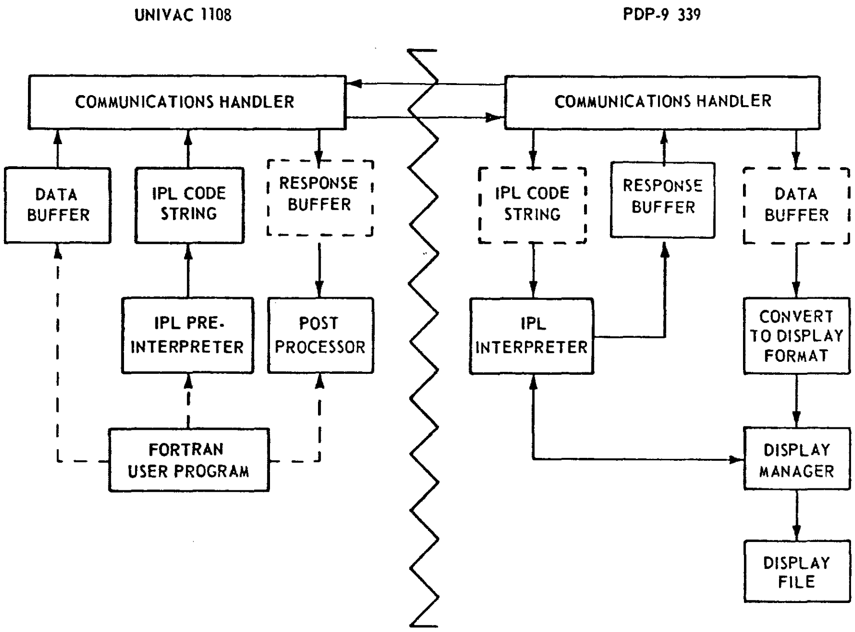


Figure 1 COMMUNICATIONS

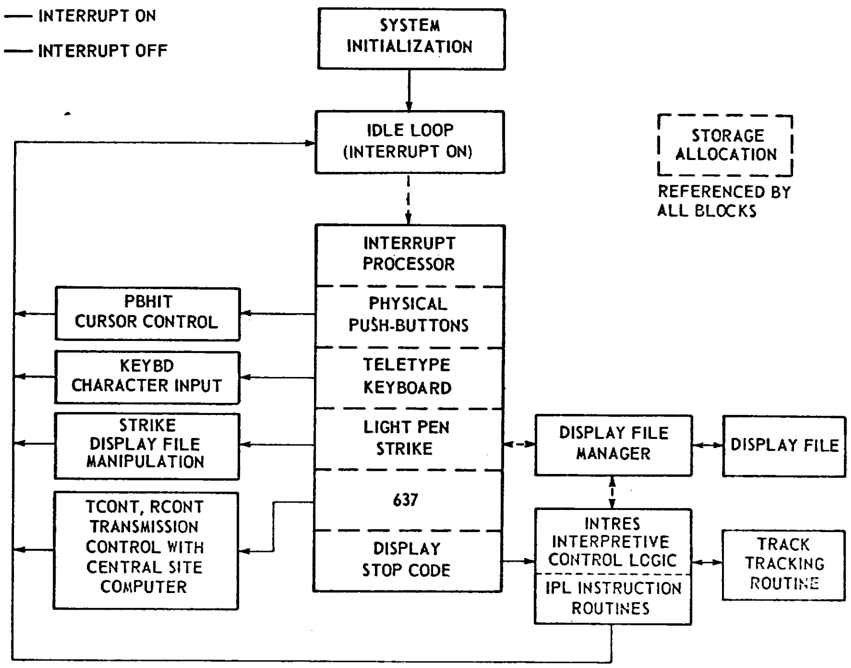
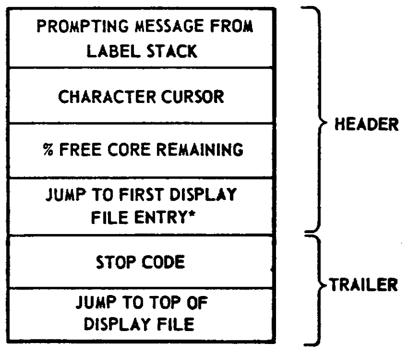


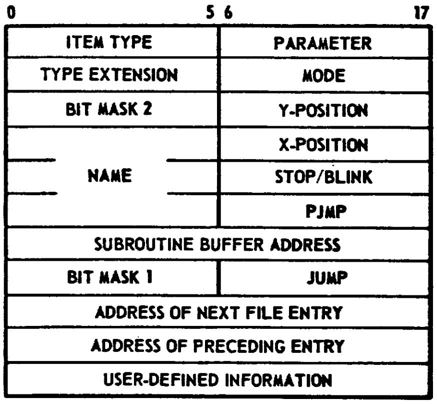
Figure 2 TERMINAL EXECUTIVE


PROMPTING MESSAGE FROM LABEL STACK
CHARACTER CURSOR
% FREE CORE REMAINING
JUMP TO FIRST DISPLAY FILE ENTRY*
STOP CODE
JUMP TO TOP OF DISPLAY FILE
HEADER
TRAILER

PERMANENT FILE

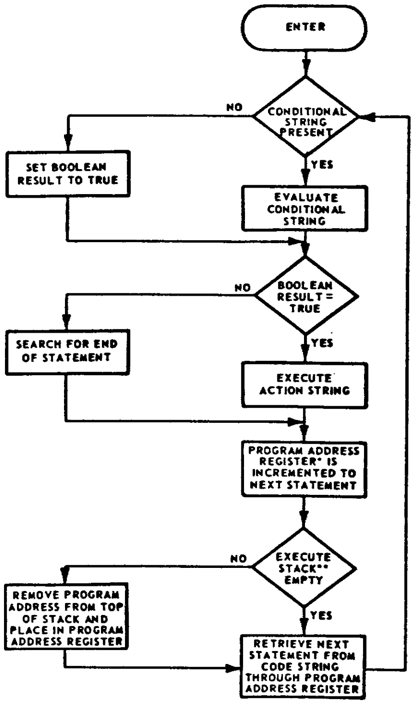
* INITIALLY NO ENTRIES EXIST - JUMP IS TO STOP CODE.

Figure 3 DISPLAY FILE FORMAT



| 0 5 | 6 17 |
|---|---|
| ITEM TYPE | PARAMETER |
| TYPE EXTENSION | MODE |
| BIT MASK 2 | Y-POSITION |
| NAME | X-POSITION |
| | STOP/BLINK |
| | PJMP |
| SUBROUTINE BUFFER ADDRESS | |
| BIT MASK 1 | JUMP |
| ADDRESS OF NEXT FILE ENTRY | |
| ADDRESS OF PRECEDING ENTRY | |
| USER-DEFINED INFORMATION | |

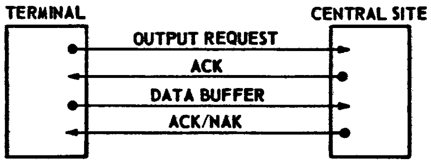
Figure 4 DISPLAY FILE ENTRY FORMAT


ENTER
CONDITIONAL STRING PRESENT
NO
YES
SET BOOLEAN RESULT TO TRUE
EVALUATE CONDITIONAL STRING
BOOLEAN RESULT = TRUE
NO
YES
SEARCH FOR END OF STATEMENT
EXECUTE ACTION STRING
PROGRAM ADDRESS REGISTER* IS INCREMENTED TO NEXT STATEMENT
EXECUTE STACK** EMPTY
NO
YES
REMOVE PROGRAM ADDRESS FROM TOP OF STACK AND PLACE IN PROGRAM ADDRESS REGISTER
RETRIEVE NEXT STATEMENT FROM CODE STRING THROUGH PROGRAM ADDRESS REGISTER

* PROGRAM ADDRESS REGISTER CONTAINS ADDRESS POINTER TO LOCATION IN CODE STRING

** EXECUTE STACK IS BUILT BY INSTRUCTIONS CALLING FOR NON-SEQUENTIAL SINGLE STATEMENT EXECUTIONS

Figure 5 INTERPRETIVE CONTROL LOGIC



THE SAME PROCESS IS FOLLOWED IN REVERSE
FOR TRANSMISSIONS FROM CENTRAL SITE
TO THE TERMINAL

RECEIVE CHAIN: OUTPUT REQUESTS FROM CENTRAL SITE.
FORMAT OF CHAIN ENTRY:

| ID |
|---|
| ADDRESS OF INPUT BUFFER FROM FREE STORAGE |
| LENGTH OF INCOMING DATA BUFFER |
| CHAIN POINTER |

TRANSMIT-ACTIVE CHAIN: OUTPUT REQUESTS, DATA, ACKs, NAKs FROM PDP-9. TRANSMISSION OCCURS FROM TOP ENTRY ON CHAIN.

TRANSMIT-INACTIVE CHAIN: DATA FROM PDP-9 HELD PENDING ACK FROM CENTRAL SITE.

FORMAT OF CHAIN ENTRY:

| CHAIN POINTER |
|---|
| MODE/TYPE* |
| ID |
| DATA BUFFER ADDRESS** |
| DATA BUFFER LENGTH** |

| * MODE: | TYPE: |
|---|---|
| CONTROL | OUTPUT REQUEST<br>ACK<br>NAK |
| DATA | IPL CODE STRING<br>DATA STRUCTURE<br>RESPONSE BUFFER |

** FOR ACK OR NAK ENTRY THESE TWO FIELDS ARE ZEROED.

Figure 6 TRANSMISSION

# EDUCATION

# A LIMITED MULTI-TERMINAL SYSTEM FOR CAI

David A. Ensor
Department of Computer Applications
The Ontario Institute for Studies in Education
Toronto, Ontario, Canada

## ABSTRACT

The paper describes a CAI multi-terminal system being implemented on the PDP-9. The OISE configuration is briefly outlined and the Author Language is discussed in addition to its processors and the ancillary programs.

## THE CONFIGURATION

Figure 1 shows the OISE configuration omitting two QUIKTRAN terminals located within another department. Most major processing is done at other installations via panel truck.

The CRT's shown are both storage mode devices and a system compatible handler has been written for textual display; the 1004 handler is underway. In addition to our open shop keypunches we have access to a keypunch service and are hoping to retrieve much of the time at present spent keying in and listing programs at the console teletype. The 1004 will be shared with our job processing group who use it for remote job entry to a commercially operated Univac 1108 in addition to reproducing and listing card decks. This is not expected to produce any insuperable scheduling problems.

The disk is on order and eagerly awaited, as is the disk handler, as many facets of the detailed system design will not be finally resolved until we have some operating experience with the disk software.

## BACKGROUND TO THE PROJECT

In the fall of 1967, working in cooperation with Mr. David Stansfield of this department, I developed the CAN CAI language and implemented a source language interpreter for it on the CGE time sharing system. This language is described in an OISE publication "A User's Guide to CAN". This facility has been used internally for introducing graduate students to CAI and is also in use at three project locations in schools. Although the response times are very bad (often in excess of 10 seconds) it has served two purposes; it has taught us much about what we would like to do, in other words provided a concrete basis for comment and discussion; secondly it has served to acquaint a wide cross-section of the educational fraternity with the basic concepts and rationale of CAI and the production of hidden multi-choice programs.

The principal criticism of CAN is that it does not allow very much freedom to the author; basically he may ask questions and branch on the subject's response. The program cannot access the scores or any other "historical" data while it is executing, and about its only advantage over a stand-alone teaching machine is that it allows open-ended questions rather than option lists (there is a PDP-10 implementation that contains certain conditional branching instructions). Although an easy language to learn it is tedious to write as the lines are sequentially numbered and these line numbers constitute the branch addresses. The language has no computational power whatever.

In addition to our experience with CAN we were also faced with a number of research proposals which required for their execution the production of conversational programs controlling a variety of devices (teletype, CRT, juke box, random access slide projector, etc.) and also demanding complex branching structures based on numerical and logical data accumulated on-line. To code these in Macro-9 would be prohibitive in time even if all of those originating the proposals were skilled assembler language programmers.

### Language Design Objectives

The new language was to be a generalized author language for writing hidden multi-choice programs and contain some computational power; it was not to favour any particular instructional strategy and was to allow the coding of certain programs communicated to the author. In addition it was to be simple enough to be readily taught to people other than computer professionals.

## THE LANGUAGE

Figures 2 and 3 list the instructions and their operands.

There are 42 variables (R0-R20, S0-S10, C0-C9) of which R0 and S0 have special meaning (time of day in seconds and number of keywords found last match respectively). Programs cannot alter these special variables. Otherwise the R stores are general working storage, the S stores are uniquely associated with the subject and the C-stores are uniquely associated with the program (they are its first ten words). The S & C stores are read in at the start of a run and updated on backing store at the end of the run. In this way limited profile building and adaptive programming experiments may be performed. The time of day is the only pseudo-random facility available to a program.

Constants may be signed or unsigned integers up to 99999, octal up to 777777 (written as Onnnnn), or up to three characters written as "aa"). Octal is right-aligned, characters left aligned in six-bit trimmed ASCII. Both are zero padded. (The limits are clearly implementation restrictions).

A string (as distinct from a text string) consists of up to six variables or constants linked by arithmetic or logical operators. The valid operators are + addition, - subtraction, * multiplication, / division, # modulus with respect to, & logical AND, . logical OR. All computation is performed according to the laws of integer arithmetic and overflow (out of 18-bit signed range) produces low-order results. Division or modulus with respect to zero produces zero. The valid relations are EQ, NE, GT, LT, GE, LE. Mode switches currently available are: A ignore characters not alphanumeric or space, B ignore spaces, K search for anticipated response as keyword (B off) or keystring (B on), and S use Demerau's algorithm for matching (similarity).

Program names must be valid PDP-9 file names. Text strings may contain any character represented in the six-bit trimmed ASCII code but the character @ has special meaning. In the textual operand of an ANS instruction it delimits substrings and may be considered as an OR operator. In a DIS or REC instruction it may be used to define the start of the text string (else leading spaces will be ignored) or to denote a special function. These include TAB, vertical forms control, sounding the TTY bell, outputting variables in integer, octal, or character form, and the output of either the answer buffer or the subject name.

## LANGUAGE PREPROCESSOR

This program, coded in Macro-9, is working at the time of writing but is not exhaustively tested. It acts as a pseudo-compiler translating the source into an interpreter format. Each page or block of output is self-contained; all references are resolved to 18-bit addresses (high order 10 bits page number, low order 8 bits relative word within page) and all constants appear within the instruction referencing them. No attempt has been made to form literal pools. Each instruction contains in its first word the instruction code in the low order six bits. End of page is marked by an instruction first word of all 1 bits and no instruction may extend over a page boundary. Instructions start and end on word boundaries. Page length was chosen at 252 words in order that complete pages might be handled under the Advanced Software System as single records.

A modified one-pass technique is used; as the source program is read the listing (if requested) is generated and error diagnostics are produced, interpreter format code is assembled, and a reference table is formed. The reference table consists of three words per entry; two containing the label in six-bit and the third containing either the object program address of the label or, if it has been found only as an operand, zero. During this phase the interpretive format code contains, for each address field, the address in core of the third word of the label entry. The code and reference table contend for the free core pointed to by .SCOM+2 and +3, a scratch area on DEC tape being used in the event of overflow. (The program aborts if the reference table overflows free core).

On finding an END pseudo-instruction the source input is closed and the reference table is scanned for undefined references; these are reported on the listing or error stream along with the program length in pages. Unless output has been requested the program reinitializes.

If the output switch has been set on the program starts resolving the references from the reference table addresses contained in the object code and writing the final version to the output device (DECtape) in maximum size IOPS binary records, one page at a time. Any undefined references are thus set to zero and this is trapped at run time. (Location 0 is C0). A switch is available to copy the C stores from a previous version of the program already on the output device into the new version at the start of the first phase.

The principal disadvantage of this technique is that undefined references are reported without any indication of where they were referred to; however the approach minimizes I/O (and processor) time and makes source input on cards or paper tape a more reasonable proposition than with a two-pass system. Alternatively it is possible to write the processed version of a file back onto the same tape as the source (a special extension is used). The preprocessor is highly modular and a new operation code which has the same operand format as an existing one can be added by inserting one six-bit constant and two JMP instructions, each at the end of a table.

## RUN-TIME SYSTEM

### The Scheduler

The present scheduler is not multi-access and is being used solely to test the other components mentioned below. Ultimately three versions of the scheduler are envisaged; a single terminal version, an LT09 (five terminal) version, and a line concentrator version. No decision will be made on expansion beyond five terminals until the system has been operational for some time.

The scheduler maintains a line control block (LCB) for each terminal. This core resident block is used by application programs for storing resume information and for communication with the scheduler. The common area contains a line status flag word, resume time of day, resume address (service routine entry), required block number (device block number of a program page while running a program), page address in core, keyboard input buffer and text output buffer (both 75 characters). The remaining area is used as required; during program execution it contains the R, C, & S stores, program address of the next instruction, current page number, a return address stack (currently 4 words long), the time-out branch address, the BRK address, mode switches, program base page block number, student name, student record address, and a twelve word scratch pad.

The scheduler also maintains a pool of 256 word buffers for allocation for disk I/O. In a five terminal system there will be five of these buffers permanently allocated one per line; with more lines they will be dynamically allocated and freed before and after line service. Before granting service to a line the scheduler checks as to whether the required disk block indicated in the LCB (if any) is in core; if not it is read in under scheduler control before the line is granted service. In practice this will be done by a one-stage look-ahead; before granting service to a line the scheduler will check whether it can also initiate a disk transfer on behalf of the next line to receive service.

Eight line states are recognized from the LCB flag word:-

1. Under service,
2. Ready for service,
3. In disk I/O
4. Waiting for disk,
5. Waiting for the first of 6 & 7,
6. Waiting time interval completion,
7. Waiting line I/O completion,
8. Inactive (not connected).

Log tape output is achieved by a subroutine which adds a line number code to the message. Double-buffering should ensure that it will not be necessary to wait for physical transfer to the device.

The scheduler is entered on I/O completion, service routine exit (always in the form of a request) and time interval end. The requests are disk read, disk write, line read, line write, idle, timed line read. It will be noted that an implicit disk read can be generated on any other request by altering the required block number in the LCB before issuing the request.

Inactive lines have a service routine entry address of the sign-on routine and the line connect status is checked once per second (a new IOT to do this is under investigation). Unexpected disconnection of a line causes it to be reset to inactive and a message to be logged. Note:- none of the application routines described below is reentrant, but each is serially reuseable from any entry to any exit, all resume information being held in the LCB. This poses a problem, yet to be fully overcome, if it is wished to bump a line for holding the CPU for too long. It is not possible just to save the active registers and restart later at the same point.

## Sign-on

This application routine asks the subject for name and validation number; if a corresponding student on-line record can be found control is passed to the select routine, otherwise the line is disconnected (again the implementation of an IOT is in hand).

## Select

The student on-line record is stored into the LCB. This 40 word record contains, in addition to identification information, the S stores and a variety of status and routing information. A student may be enrolled in up to four suites (each with an arbitrary name) having next-program pointers updated dynamically by the NEX command. The status indicator word denotes him as an "instructor", "free agent", allows him to work cyclically (one program from each suite). An instructor may examine and modify other student on-line records the first two letters of whose validation code are the same as his and may examine program C stores in addition to having the same right as a free agent to list the on-line program directory and run any program from it. Dependent on the setting of the status indicator word the select routine will determine the alternatives (if any) and offer them to the student. All the functions available except executing a program reside within the select routine itself. If a program is chosen (or is mandatory) the application program section of the LCB is set up, start of program logged, and control is passed to the interpreter.

Select is also entered from the interpreter at end of program to reset the routing pointers if required, log completion, and initiate new action or sign off.

## The Interpreter

At the time of writing the interpreter is fully coded and being tested under the pilot single terminal scheduler. Its actions are fairly well defined by the language and the LCB format. On gaining control it retains it until it detects an out of page branch, requires line I/O, finds an IDL, STP, PAS, or NEX instruction or detects an abort condition. At every opportunity it protects itself and the system, and attempts to keep the program going wherever possible. Thus there are default options for every error which can escape the preprocessor (variable out of range on a TRE instruction, time limit less than 1 second or greater than 2 minutes, branches to zero, invalid special functions on DIS & REC instructions, etc).

Although it is highly modular in concept it has been coded in one routine to save time and space on linkages. The principal problem foreseen at this time is that of infinite loops containing no I/O; as noted above there is no system means of escaping from these at this time.

## BATCH-TIME JOBS

## Log Tape Sort/Explosion

The log tape contains student information which must be sorted first by line number and then, if it is possible that the same student may have used more than one line in one day, by student. In a full system the log tape will also contain student history dump requests, program updates, program listing and C store print requests, and enrollments (on-line deletion of students is allowed). Each of these message streams must be isolated.

## Student File Update

The student master file is merged with the student update file produced from the log tape and the student on-line records from the disk to form the new student master. Listings are produced as directed by either the update stream or the operator.

### Program File Update

The master program file is merged with the program update file extracted from the log tape and listings again generated as required.

### Object Program Update

Phase 1: the C-stores from disk are merged into the object program master file. Completion of this phase frees the disk space used by the on-line system.

Phase 2: using a control stream generated from the log tape and/or by the operator, source programs are run through the preprocessor from the program master file onto the object update file.

Phase 3: the object master file is merged with the update file from phase 2 to produce the new object master.

### Disk Loader

A large contiguous space is allocated on disk, and the student on-line records extracted from the student master and written out; in so doing a required program index is formed from the next-program pointers. This index is merged with a control stream defining other programs to be loaded (i.e. standard library programs) and program loading commences from the object master file. A directory is built up and after completion of program loading the directory is written out and a disk loading map produced. This map is the basis for a very short control stream to the on-line system when it is loaded.

## SYSTEM SUMMARY

Although certain experiments which we wish to carry out using the language will be available to only one terminal because of the device configuration required, the system should in general permit a number of terminals to run concurrently on the same or different programs. Certain critical limitations are imposed to make this more easily achieved.

Each terminal has a core block in which all of its status and resume information is held and this block is of fixed length (currently 128 words); the program itself resides on disk and may not be altered at any time during a run, thus only one copy is required. The need for roll-out is eliminated and the critical factor on the flexibility of the system is the size and use made of the core resident block. All data created during the execution of a program is logged rather than updating an on-line file and only very limited on-line modifications to data are performed.

For the scheduler to allocate a line for service it need only ensure that the required program page is in core; were the program pages executable core there would be several problems in doing this. The PDP-9 has no paging or relocation hardware and most CAI work is done in system subroutines necessitating the resolution of transfer vectors. Direct solutions to these problems do exist, but an interpretive system has been adopted in the interests of simplicity. The scheduler/service routine interface is simply the address of the line control block.

A source language interpreter would, however, be prohibitive in time (particularly searching for reference labels) and core thus the source program is translated by a batch program into an intermediate form known as interpreter object code; all the references are resolved and the first word of each instruction is in a standard format. This gives relatively fast entry to the appropriate system subroutine and affords direct access to the target instruction on any branch.

No provision has been made for the on-line updating of programs but a facility allowing the logging of updates and print requests for processing in batch mode has been incorporated in the design and the system has flexible student routing and history building facilities.

Despite its limitations it is hoped that the system will provide a viable low-cost base for research and development in the broad field of "CAI".

OISE CONFIGURATION    NOVEMBER 1968

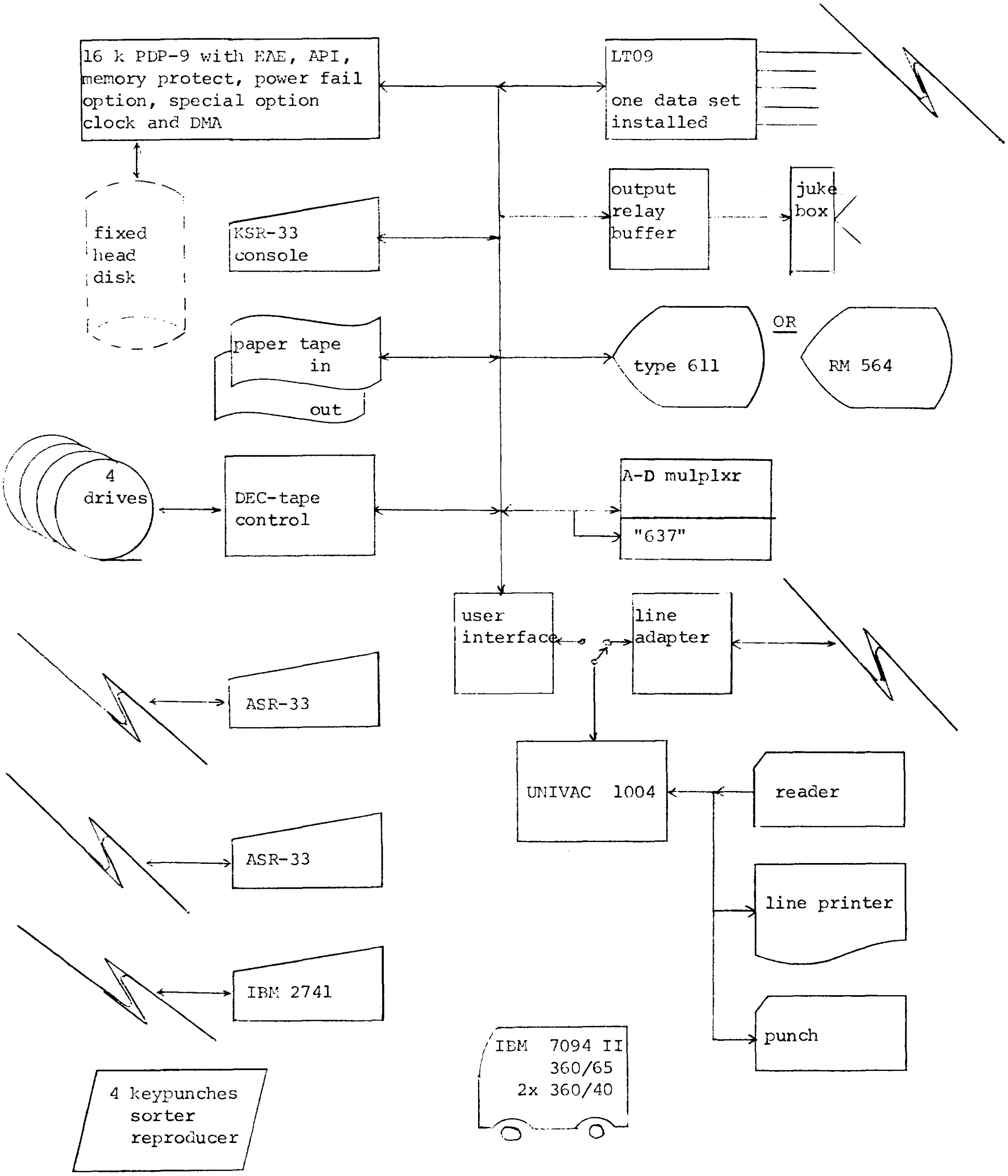


Figure 1        Configuration

## LANGUAGE (1)

### Line or card format

Reference label b̸ operation code b̸ operands b̸ comment OR

* comment

### Pseudo-operation

END terminates program ; must be the last statement of every program

### Operation codes

| mnemonic | operands | function |
|---|---|---|
| PAS | program name | the next statement obeyed is the first instruction of the named program |
| NEX | program name | program execution ends ; the suite pointer is updated to the operand name |
| STP | none | program execution ends |
| RET | none | the next instruction obeyed is that pointed to by the bottom member of the return address stack |
| JMP | {reference label} | the next instruction obeyed is that with the operand reference label; if none the next instruction in sequence is skipped. |
| BRK | {reference label} | if the subject replies BRK to any input cue the next instruction to be obeyed will be that with the operand reference label; if none the BRK will be ignored. |
| CAL | {reference label} | the address of the next instruction is placed at the bottom of the return address stack and the next instruction to be obeyed is that with the operand reference label; if none the next instruction is obeyed and a next-instruction pointer inserted in the stack. |
| INC | variable name, variable name, ... | one is added to each specified var. |
| DEC | as above | one is subtracted from each specified variable |
| NEG | as above | each specified variable is negated |
| CLR | as above | each specified variable is zeroized |

Figure 2 Language (1)

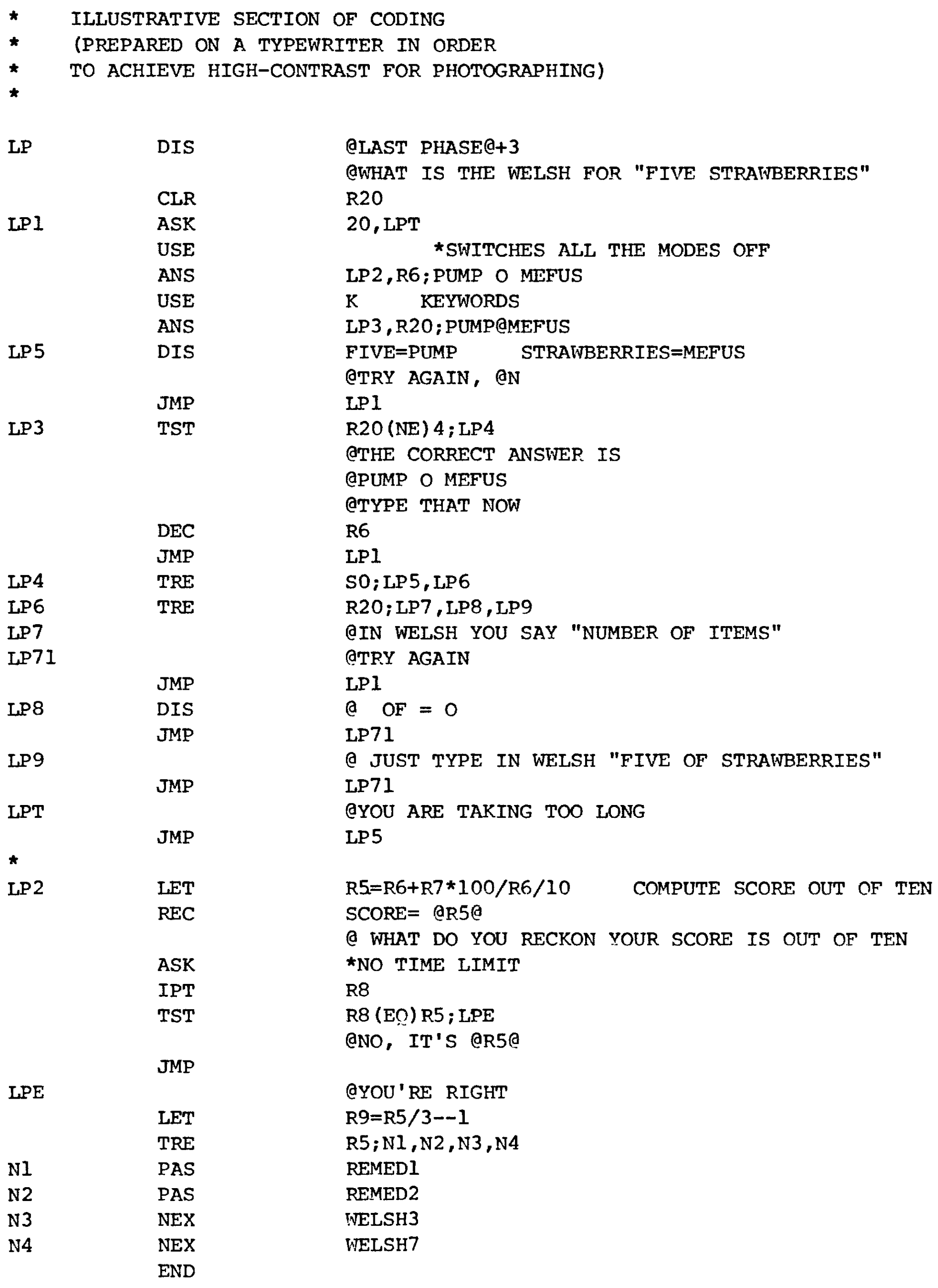
## LANGUAGE (2)

Operation codes (continued)

| mnemonic | operands | function |
|---|---|---|
| IPT | variable name, variable name, ... | packs from the answer buffer into the specified variables; signed or unsigned integers are converted to binary and other characters are packed into six-bit left-aligned zero padded. Space is used as a delimiter and not input. |
| DIS | text string | the operand text is displayed at the terminal |
| REC | text string | the operand text is logged |
| ASK | {(variable name<br>constant), reference label} | the subject is cued for input; if the optional operands are present and a reply is not completed in the number of seconds indicated by the first operand the next instruction to be obeyed is that with the operand label. |
| USE | {mode mnemonic,mode mnemonic, ...} | the mode switches specified are set on, all others are set off. |
| TRE | variable name; reference label, reference label, ... | the next instruction to be obeyed is that with the operand reference label whose positional value in the list is equal to the value of the specified variable. |
| ANS | reference label {,variable name}; text string | the answer buffer is checked against the text string according to the modes in effect. If no match is found the next instruction is obeyed else a JMP is made using the operand reference label and the specified variable, if any, is incremented. |
| LET | variable name=string | the value of the string computed left to right is substituted for the current value of the variable. |
| TST | string(relation) (variable name<br>constant);reference label | if the relationship is true a JMP is performed using the operand reference label. |
| IDL | (variable name<br>constant) | the next instruction is not obeyed until the number of seconds specified in the variable/constant have elapsed. |

The graphics { } indicate optional operands.

Figure 3 Language (2)



```
*    ILLUSTRATIVE SECTION OF CODING
*    (PREPARED ON A TYPEWRITER IN ORDER
*    TO ACHIEVE HIGH-CONTRAST FOR PHOTOGRAPHING)
*

LP          DIS           @LAST PHASE@+3
                          @WHAT IS THE WELSH FOR "FIVE STRAWBERRIES"
            CLR           R20
LP1         ASK           20,LPT
            USE                  *SWITCHES ALL THE MODES OFF
            ANS           LP2,R6;PUMP O MEFUS
            USE           K     KEYWORDS
            ANS           LP3,R20;PUMP@MEFUS
LP5         DIS           FIVE=PUMP     STRAWBERRIES=MEFUS
                          @TRY AGAIN, @N
            JMP           LP1
LP3         TST           R20(NE)4;LP4
                          @THE CORRECT ANSWER IS
                          @PUMP O MEFUS
                          @TYPE THAT NOW
            DEC           R6
            JMP           LP1
LP4         TRE           SO;LP5,LP6
LP6         TRE           R20;LP7,LP8,LP9
LP7                       @IN WELSH YOU SAY "NUMBER OF ITEMS"
LP71                      @TRY AGAIN
            JMP           LP1
LP8         DIS           @  OF = O
            JMP           LP71
LP9                       @ JUST TYPE IN WELSH "FIVE OF STRAWBERRIES"
            JMP           LP71
LPT                       @YOU ARE TAKING TOO LONG
            JMP           LP5
*
LP2         LET           R5=R6+R7*100/R6/10     COMPUTE SCORE OUT OF TEN
            REC           SCORE= @R5@
                          @ WHAT DO YOU RECKON YOUR SCORE IS OUT OF TEN
            ASK           *NO TIME LIMIT
            IPT           R8
            TST           R8(EQ)R5;LPE
                          @NO, IT'S @R5@
            JMP
LPE                       @YOU'RE RIGHT
            LET           R9=R5/3--1
            TRE           R5;N1,N2,N3,N4
N1          PAS           REMED1
N2          PAS           REMED2
N3          NEX           WELSH3
N4          NEX           WELSH7
            END
```

Figure 4          Typical segment of code

# CONVERSATIONAL BASIC ON THE PDP-8 LINE

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

## ABSTRACT

This paper will concern itself with the use of CINET-BASIC in the classroom. It will include sample problems and a discussion of the variations between this BASIC and other existing BASIC languages. CINET-BASIC (Computer Instruction NETWORK's BASIC) was written using FOCAL's subroutines for the standard PDP-8 series with 4K memory and ASR-33 Teletype.

After a six-month study and planning period, the Computer Instruction NETWORK came into being in May of 1967. One of the outcomes of the planning was a decision to offer to our students in Oregon a survey of the levels of programming, starting with machine language and ending with a compiler language. It was further felt that being a survey, we would attempt to teach concepts rather than concentrated details. In order to achieve an understanding of those fundamental concepts without a great deal of bit chasing, we restricted our machine language to relatively simple subsets of instructions. In the same manner, we felt that there were important concepts that were fundamental to most compiler programming and that these should be emphasized, eliminating as many of the extraneous restrictions as possible from instruction.

We therefore attempted to find a language that had a minimum of extraneous and superfluous compiler requirements - one that would give a minimum turnaround time with on-line debugging for hands-on use by students. We also felt that it should be a language that could be readily used as a tool by the different high school classes and reflect recent development in computers and compilers. The language we chose was BASIC, and during the summer of 1967 Dave Gillette and I wrote a conversational compiler called CINIC, patterned after BASIC, for the PDP-8. with a 4K memory. This language was severely limited by the lack of functions and no list capabilities. There were plans for extensive revision and rewriting in the summer of 1968. However, with the advent of DEC's interpretive language, FOCAL, we decided that the easiest approach might be to cannibalize this and use those subroutines that would apply.

This was the approach used, and we now have a much stronger version of the language with CINET-BASIC than we did with CINIC. As in most things, we paid a price in one area to make a gain in another: we lost speed to gain strength. The interpretive approach is about ten times slower than the compiler. However, for our needs, this was a minor price to pay in comparison to the value gained.

We feel that we now have a language that satisfies our original requirements.

A brief description of the language follows:

Statement Numbers

For a stored program each statement must be preceded by a statement number (1 through 2047). You may type statement numbers in any order. The program will run in numerical order.

Variables

Variables in CINET-BASIC are denoted by any single letter. Thus A, B, X, and N are variables. A variable may also have a subscript. The subscript is designated by immediately following identifiers with the subscript value enclosed in parentheses.

Examples:

```
A(3)
M(I)
```

Variations from Other Basics - Only single subscripts allowed (range ±2046). Subscript may be computed; however, only the integer value within the accepted range will be used.

Examples:

```
A(I+2)
X(3*4-3)
```

Numbers

Input - Numbers may be typed in with or without signs. They may be typed in fixed point, floating point or exponential form. However, no more than 6 significant digits may be given.

Output - The printout of numbers will be given in floating points unless the value exceeds the range of $10^{-5}$ to $10^{9}$. Numbers that exceed this range will automatically be printed in exponential form.

Examples:

```
1.5
-2035.
0.3200000E11
```

Instructions

Following each statement number is an instruction. The instruction name must be followed by a space. Correct spacing in CINET-BASIC is important. However, in general, correct spacing can be achieved if the user will use the normal conventions of spacing between words and not spacing in algebraic expressions. Following is a list of instructions with examples and explanations.

### Print

The print instruction commands the computer to Output information.

If print is followed by a letter or series of letters separated by commas or semicolons, the values presently assigned to these variables will be typed. If print is followed by any series of characters enclosed within quotation marks, the enclosed characters will be reproduced exactly.

Examples:

```
    100 PRINT A
```

If "A" were first computed to be 5, the Teletype would type:

```
    5.
```

```
    50 PRINT "AMOUNT IS", A
```

The Teletype would type:

```
    AMOUNT IS 5.
```

The print instruction may also be used to evaluate one or more expressions.

Examples:

```
    1000 PRINT 2-5, SQR(64)
```

2-5 would be evaluated and output as -3. Then the square root of 64 would be found and output as 8. 14 spaces are reserved for each numerical value typed. The output would appear as

```
    -3.           8.
```

The print statement will normally give a carriage return and line feed at the completion of typing. This carriage return, line feed, may be suppressed by ending the statement with a comma or semicolon.

Examples:

```
    20 PRINT "WELL!"

    30 PRINT "HI ",

    40 PRINT "THERE"
```

The TTY would type the following, suppressing the carriage return and line feed, after the "HI ".

```
WELL!
HI THERE
```

The print instruction can be used to give line spacing.

Example:

```
    1000 PRINT
```

The TTY would give only a carriage return and line feed.

Variation from other Basics - There is no distinction made between a comma and a semicolon in a print statement. 14 spaces are reserved for each Variable regardless.

No attempt is made to give an automatic carriage return, line feed, at the end of 72 characters (width of line on TTY). The carriage return line feed is only given when the print statement is complete.

It is possible to get a carriage return without a line feed. This is accomplished by placing a # separated from other output by commas, but not in quotes. This will cause the TTY to give a carriage return only.

Example:

```
    10 PRINT "IS    RIGHT?",#

    20 PRINT "  ",A
```

If A is 5, then the TTY will type:

```
    IS 5. RIGHT?
```

This is very useful in plotting or graphing.

A carriage return may be substituted for a " sign if the quote is the last character in a line.

### 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 question mark "?", 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 following would result: the computer prints a "?" and waits for input from the keyboard. If you type a 5 terminated with a comma, the computer assigns the value 5 to the variable A, types a "?" and waits again. If you type a 70.2 terminated by a comma, the machine sets B = 70.2 and types a carriage return. The result on paper looks like this:

```
    ?5,?70.2,
```

A value may also be terminated with a space, carriage return, or semicolon.

Variations from Other Basics - The control characters " and # of the print statement are also operative on the input statement.

Example:

```
    150 INPUT "TYPE THE AMOUNT OF INCOME", A
```

The TTY would type the following and then wait for you to assign a value to 'A' as shown below.

```
    TYPE THE AMOUNT OF INCOME?
```

### 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 |
|---|---|---|
| + | addition | A + Z |
| - | subtraction | 3 - 5.03 |
| * | multiplication | B * C |
| / | division | 12/3 |
| ↑ | exponentiation | A↑2 (means $A^2$)<br>only integer portion of an exponent will be used |
| ( ) | parentheses may be used in pairs to clarify any algebraic expression. | |

Order of priority of operations:

1. Values inside parentheses
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction

Examples:

```
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 $\frac{2(2A-B)}{3}$

Functions - In addition to the five arithmetic operations, the computer can evaluate several mathematical functions. These functions are given special 3-letter names.

| Functions: | Interpretation: |
|---|---|
| SIN(X) | Find the sine of X |
| COS(X) | Find the cosine of X |
| ATN(X) | Find the arctangent of X |
| EXP(X) | Find $e^X$ or $(2.718281)^X$ |
| LOG(X) | Find the natural log of X |
| ABS(X) | Find the absolute Value of X |
| SQR(X) | Find the square root of X |
| INT(X) | Find integer part of number in the range ±2046 |
| RND(X) | Find a nonstatistical pseudo-random number between ±1 |

Variations from Other Basics - The Tangent function is not available. It is possible in CINET-BASIC to allow one LET statement to define a series of variables by separating each equation with a comma or semicolon.

Example:

```
10 LET A = 2, B = 3, C = B*A

20 PRINT A, B, C
```

This program would give

```
2.          3.          6.
```

### GO TO

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

Example:

```
50 GO TO 25
```

### IF, THEN

This statement allows the computer to make a decision. Each statement must be of the form:

IF (variable) (relation) (expression) THEN (line number)

The relation may be made up of a <, >, =, or any combination of the three.

Examples:

```
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<= 3+5*A THEN 97
```

If X is less than or equal to 3+5*A, statement 97 will be executed. If not, the next statement after 34 will be executed.

Variations from other Basics - The left member may only be a variable.

### FOR

This instruction is used for setting up program loops and iterations. It must be used in conjunction with a NEXT statement. The general format is:

```
100 FOR A = X TO Y STEP Z
```

The variable A is initialized to the value of X, then those statements following the FOR are executed until a NEXT statement is encountered. When a NEXT statement is found, control is sent back to the preceding FOR. The value A is then incremented by Z and compared to the value Y. If A is less than or equal to Y, then those statements following the FOR will once more be executed. This process will be repeated until A is greater than Y. At that time the statements following the NEXT will be executed. X, Y, and Z may be algebraic expressions. Z must have a positive value. If STEP is omitted, then the increment will be one.

Example:

```
10 FOR A = 1 TO 2↑2  STEP 2.5

20 PRINT A

30 NEXT A
```

When RUN is typed, this program would give:

```
1.

3.5
```

### NEXT

Used only with FOR statements. The general form is:

```
20 NEXT A       (See FOR)
```

### GOSUB

When a particular part of a program is to be performed at several different places in the program, it is most efficiently programmed as a subroutine. The subroutine is entered with a GOSUB statement, where the number is the first line number in the subroutine. The subroutine must logically end with a RETURN instruction. This transfers control back to the statement following GOSUB.

Example:

```
 10 PRINT "A",

 20 GOSUB 100

 30 PRINT "B",

 40 GOSUB 100

 50 PRINT "C"

 60 STOP

100 PRINT "XXX"

110 RETURN
```

This program would give:

```
AXXX
BXXX
C

READY
```

### RETURN

The RETURN must be the last statement in a subroutine.

### DIMENSION (DIM)

DIM is used to reserve space for subscripted variables greater than 10. However, the DIM statement may be omitted, as it is not necessary to reserve space in CINET-BASIC.

### REMARKS (REM)

This statement is used to allow the programmer to insert explanatory remarks in a program. The computer will completely ignore the line when RUN is typed.

Example:

```
100 REM STATEMENT 100 MAY BE CHANGED TO GIVE
        NEGATIVE ROOTS.
```

### STOP

The stop instruction is used to designate a logical stop in the middle of a program.

### END

The END instruction is used to designate the last statement in a program.

Variations from Other Basics - In CINET-BASIC the END instruction is not necessary. However, the computer will not acknowledge the completion of the program with READY, unless END or STOP is used.

## DIRECT MODE

If you wish to run a short program only once, such as:

```
PRINT SQR(49)
```

you may use direct mode in CINET-BASIC. This is accomplished by simply typing the above instruction (without a statement number) followed by a carriage return. In this example, "7." will be printed immediately. The instruction is executed but not stored for future use.

## COMMANDS

There are certain commands that are intended for direct mode only. These are:

### RUN

RUN will direct the computer to execute the stored program.

Examples:

| | |
|---|---|
| RUN | This executes a stored program starting with the lowest-numbered instruction. |
| RUN 50 | This executes a stored program starting with statement 50 and ignoring all lower-numbered instructions. |

### LIST

The LIST command will direct the computer to list out the stored program. LIST followed by a statement number would direct the computer to list from that statement number on.

Examples:

| | |
|---|---|
| LIST | Lists the complete stored program |
| LIST 37 | All instructions with statement numbers equal to or greater than 37 will be listed. |

### EDIT

This command must be followed by a statement number. The statement designated will be set up for edit upon receiving a carriage return. The user then types a search character. The computer will type out the contents of that line until the search character is typed. The user may then use a number of operations within the Edit mode, such as typing new characters to be added to the line at the point of insertion; or he may type a RUBOUT. This deletes characters to the left. One character is deleted for each time that the user strikes the RUBOUT key.

### ERASE

Erase must be followed by some parameter.

Examples:

```
ERASE 30
```

The above command will erase from the stored program statement number 30.

```
ERASE ALL
```

The above command will erase the entire stored program.

INTERRUPT

You may interrupt CINET-BASIC at any time to return control to the user by typing a CTRL/C.

With only a few short minutes of instruction, students can use CINET-BASIC in direct mode. They would use the compiler as they might a desk calculator. If they wanted the square root of a number, they would simply type:

```
     PRINT SQR(1024)
```

and upon receiving the carriage return, the computer would type:

```
     32.
```

In this case no program is stored nor is any existing program affected.

It is possible to define variables in direct mode as you see in this example of defining the coefficients of a quadratic equation.

```
     LET A=1, B=5, C=6, D=SQR(B 2-4*A*C)
```

In this case the values for the variables are defined and stored in the computer. The LET statement is not saved but simply executed. The direct command:

```
     PRINT (-B+D)/(2*A), (-B-D)/(2*A)
```

would cause the computer to look up the stored values for B, D and A, compute and print out

```
     -2.          -3.
```

Should the student wish to use the program again, it is necessary to retype it with the new coefficients.

An hour or so of additional instruction will allow the student to write stored programs. Using this concept, he may write a permanent program that may be used again without retyping. However, this will force him to consider the general case for all possibilities such as imaginary roots. As a result, the permanent program will be somewhat longer as you see here:

```
 10 PRINT "THIS PROGRAM GIVES SOLUTION TO THE QUAD-
          RATIC"
 20 PRINT "EQUATION OF THE FORM A*X↑2+B*X+C=0"
 30 PRINT
 40 PRINT "PLEASE TYPE A, B, AND, C",
 50 INPUT A,B,C
 60 LET D=B↑2-4*A*C
 70 IF D 0. THEN 110
 80 PRINT "X1=",(-B+SQR(D))/(2*A)
 90 PRINT "X2=",(-B-SQR(D))/(2*A)
100 GO TO 30
110 PRINT "X1=",-B/(2*A),"+I",SQR(ABS(D))/(2*A)
120 PRINT "X2=",-B/(2*A),"-I",SQR(ABS(D))/(2*A)
130 GO TO 30
```

When RUN is typed, the program is executed; and, with the appropriate input, the following would result:

```
RUN
THIS PROGRAM GIVES SOLUTION TO THE QUADRATIC EQUA-
TION OF THE FORM A*X↑2+B*X+C=0

PLEASE TYPE A, B, AND, D?1 ?5 ?6
X1=-2.
X2=-3.
PLEASE TYPE A, B, AND,C?2 ?0 ?2
X1= 0.          +I 1.
X2= 0.          -I 1.
```

From a linear program with simple branching, we would proceed to an elementary loop using an IF THEN decision statement to test the limit. This would be followed by automatic looping, using the FOR statement. This program for printing out the value of factorials to the limit given is an example:

```
0010 INPUT "PLEASE TYPE A LIMIT ",L
0020 LET X=1
0030 FOR I=1 TO L
0040 LET X=I*X
0050 PRINT I,#,"    FACTORIAL IS ",X
0060 NEXT I
0070 END

RUN
PLEASE TYPE A LIMIT ?14

 1.  FACTORIAL IS 1.
 2.  FACTORIAL IS 2.
 3.  FACTORIAL IS 6.
 4.  FACTORIAL IS 24.
 5.  FACTORIAL IS 120.
 6.  FACTORIAL IS 720.
 7.  FACTORIAL IS 5040.
 8.  FACTORIAL IS 40320.
 9.  FACTORIAL IS 362880.
10.  FACTORIAL IS 3628800.
11.  FACTORIAL IS 39916800.
12.  FACTORIAL IS 479002000.
13.  FACTORIAL IS 0.62270  E 10
14.  FACTORIAL IS 0.87178  E 11

READY
```

Notice that there is no need to specific format of Input or Output statements. Once the limit has been reached, exponential notation is automatically used.

The study of programming would continue to encompass the important concept of subroutining using the GOSUB statement. From this point, students branch out and apply their programming knowledge to areas of interest ranging from games and computing basketball statistics to doing their physics homework. There seems to be no limit to the ways that their imaginative minds can find to use the computer.

With the 4K memory, there is understandably a limit to size of program possible, but most programs that are attempted at the highschool level can be handled by CINET-BASIC. Depending on the number of variables and types of instructions, it is possible to have up to about sixty stored instructions.

With our multi-level language approach to computer science, ending with CINET-BASIC, we of the C. I. NETWORK feel that students get a good well-rounded survey of computers and programming. Although we do not claim to necessarily train technologically-capable programmers, we do feel that our students have the general knowledge necessary for an informed public in this age of computers. We also feel that he is now in a position to make a knowledgeable decision about choosing a career in the computer field.

# PATTERN AND RATE OF CONVERGENCE OF THE PERTURBATION TECHNIQUE IN THE TREATMENT OF LINEAR AND NONLINEAR PLATE PROBLEMS

S.F. Ng
University of Ottawa
Ottawa, Canada

## ABSTRACT

An approximate method based on the perturbation technique is used to solve the small and large deflection problems of the bending of plates resting on an elastic support. The influence of the various parameters such as the plate aspect ratio, skew angle and foundation modulus on the pattern of convergence is investigated. Results of this investigation indicate that in general, the rate of convergence of the method decreases with increase in the number of variable parameters both in the linear and nonlinear theory of plates. The algorithm of the solutions and the manipulations of the matrix equations are programmed utilizing direct access devices.

## INTRODUCTION

Based on the small deflection theory of thin elastic plates, approximate solutions to the problems of bending of plates of various shapes are known[1]. These solutions are obtained either by solving approximately the biharmonic fourth order partial differential equation

$$W,_{xxxx} + 2W,_{xxyy} + W,_{yyyy} = q/D \qquad (1)$$

or by minimizing the total energy of the plate using the energy integral equation

$$I = \iint \left( \frac{D}{2} \left\{ (W,_{xx} + W,_{yy})^2 - 2(1-\nu) \left[ W,_{xx} W,_{yy} - (W,_{xy})^2 \right] \right\} - wq \right) dxdy \qquad (2)$$

Here, W is the lateral deflection of the plate
q is the uniform distributed load on the plate
D is the rigidity of the plate material
$\nu$ is the Poisson's ratios
I is the total energy (potential energy + strain energy)

In using Eqs. (1) or (2) to solve plate problems, it is understood that the maximum deflection of the plate is considered to be of such magnitude that the effect of stretching of the middle plane of the plate on its curvature can be neglected. When the lateral deflection of the plate is moderately large, that is, in the neighbourhood of one half the plate thickness or more the small deflection theory is no longer applicable and the effect of the forces acting in the middle surface must be taken into account. By considering the effect of the membrane forces on plate curvature, the analysis becomes nonlinear and much more complicated. The governing partial differential equations for such an analysis were first formulated by von Karman[2]. For plates uniformly loaded and resting on elastic foundations, the von Karman equations can be slightly modified and expressed in terms of the three displacement components U, V, and W of a point in the middle plane of the plate. In rectangular cartesian co-ordinates, these equations can be written as:

$$U,_{xx} + W,_x W,_{xx} + \nu(V,_{xy} + W,_y W,_{xy}) + \frac{1}{2}(1-\nu)(U,_{yy} + V,_{xy} + W,_x W,_{yy} + W,_y W,_{xy}) = 0 \qquad (3)$$

$$V,_{yy} + W,_y W,_{yy} + \nu(U,_{xy} + W,_x W,_{xy}) + \frac{1}{2}(1-\nu)(V,_{xx} + U,_{xy} + W,_y W,_{xx} + W,_x W,_{xy}) = 0 \qquad (4)$$

$$D\, \nabla^2\nabla^2 w = q - kw + h\left\{ \frac{E}{(1-\nu^2)} \left[ U,_x + \frac{1}{2}(W,_x)^2 + \nu(V,_y + \frac{1}{2}(W,_y)^2 \right] W,_{xx} + \frac{E}{(1-\nu^2)} \left[ V,_y \ \ \frac{1}{2}(W,_y)^2 + \nu(U,_x + \frac{1}{2}(W,_x)^2) \right] W,_{yy} + \frac{E}{(1-\nu^2)} (U,_y + V,_x + W,_x W,_y)\, W,_{xy} \right\} \qquad (5)$$

Where U, V, and W are the displacements of the plate in the x, y, and z directions respectively and k is the foundation modulus.

## METHOD OF SOLUTION

The perturbation technique[3] is now used to solve approximately the linear (Eq. 1) and nonlinear (Eqs. 3, 4, and 5) problems of the bending of plates. This technique requires the expansion of the displacement components U, V, and W as well as the uniformly distributed load q in a power series in ascending powers of the center plate deflection

Wo. Thus,

$$q = \alpha_1 Wo + \alpha_3 Wo^3 + \alpha_5 Wo^5 \quad \ldots \qquad (6)$$

$$W = w_1(x,y)Wo + w_3(x,y)Wo^3 \quad \ldots \qquad (7)$$

$$U = u_2(x,y)Wo^2 + u_4(x,y)Wo^4 \quad \ldots \qquad (8)$$

$$V = v_2(x,y)Wo^2 + v_4(x,y)Wo^4 \quad \ldots \qquad (9)$$

By substituting equation 6 through 9 into the three governing partial differential equations Eqs. 3, 4, and 5 and by equating coefficients of terms containing Wo, the following differential equation governing the linear deflection problem is obtained:

$$D\, \nabla^2 \nabla^2 w_1 = \alpha_1 - k w_1 \qquad (10)$$

By successively equating higher powers of Wo, differential equations governing the in-plane displacements and the non-linear load-deflection relationships are obtained.

Thus, for example, by equating $Wo^2$, the differential equations governing the first approximation for the in-plane displacements are:

$$2u_{2},_{xx} + (1-\nu)u_{2},_{yy} + (1+\nu)v_{2},_{xy} + 2w_{1},_{xx}w_{1},_{x} + (1+\nu)\,(w_{1},_{xy}w_{1},_{y}) + (1-\nu)w_{1},_{yy}w_{1},_{x} = 0 \qquad (11)$$

$$2v_{2},_{yy} + (1-\nu)v_{2},_{xx} + (1+\nu)u_{2},_{xy} + 2w_{1},_{yy}w_{1},_{y} + (1+\nu)w_{1},_{xy}w_{1},_{x} + (1-\nu)w_{1},_{xx}w_{1},_{y} = 0 \qquad (12)$$

### Plate Shapes and Displacement Functions

For plates of a given shape, displacement functions satisfying the kinematic boundary conditions have to be assumed. The accuracy of the solution depends not only on the number of undetermined parameters but also on their convergence. For the sake of brevity, the theoretical aspects of convergence of the perturbation technique and the choice of suitable displacement functions are not elaborated here but can be found elsewhere in the technical literature.[4,5]

## DISCUSSION OF RESULTS

Typical results for center deflections of circular, skewed, elliptical, and rectangular plates are shown graphically in Figs. 1, 2, 3, and 4, respectively. For the case of the small deflection of circular plates (Fig. 1), it can be observed that when the plate is free from foundation support (k = 0), the center deflection of the plate agrees exactly with that obtained by Timoshenko[1] and is invariant with the number of terms used in assumed displacement function. As the foundation modulus K is increased, the rate of convergence decreases slightly but nevertheless all the results reach a stationary value after a six-term solution. This is in contrast with the typical small deflection result for a skewed rhombic plate shown in Fig. 2. Here, due to the additional parameter of the skew angle, the rate of convergence deteriorates significantly and the results do not come to a stationary value. For the sake of comparison, the result obtained by Kennedy[6] using the Galerkin's method is also plotted.

Typical non-linear results for elliptical and rectangular plates are plotted in Figs. 3 and 4. From these figures it can be observed that while there is no absolute monotonic convergence the results seem to vary cyclically and within reasonably narrow limits.

## CONCLUSION

From the brief analysis presented herein, the following conclusions may be drawn:

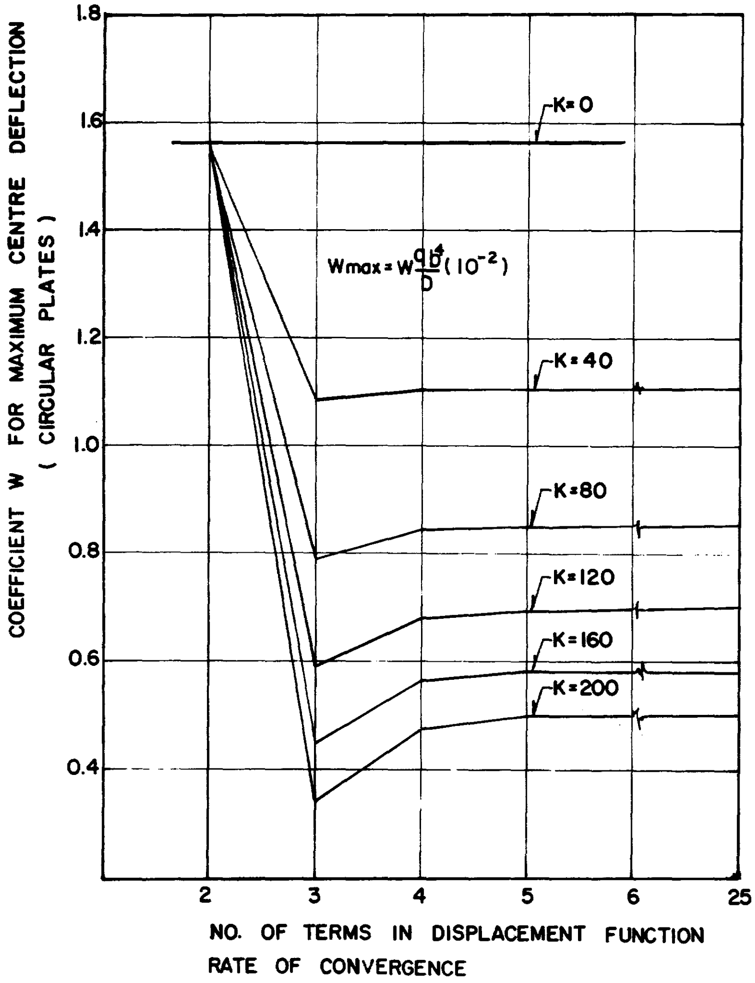
1. The small parameter perturbation technique is relatively simple to apply both to linear and nonlinear problems.
2. For linear plate problems with simple geometric shapes (circular, elliptical and rectangular), the rate of convergence is fast and invariably a stationary value in the results is reached for a 6th-term solution. The rate of convergence decreases with an increase in the number of variable parameters such as the plate aspect ratio, the foundation modulus and the angle of skew.
3. While there is no well-defined pattern of convergence of the perturbation technique in the nonlinear treatment of plate problems, the results seem to vary cyclically as the number of terms is increased. The numerical results vary only within narrow limits and are, in general, accurate enough for engineering purposes.

## ACKNOWLEDGMENT

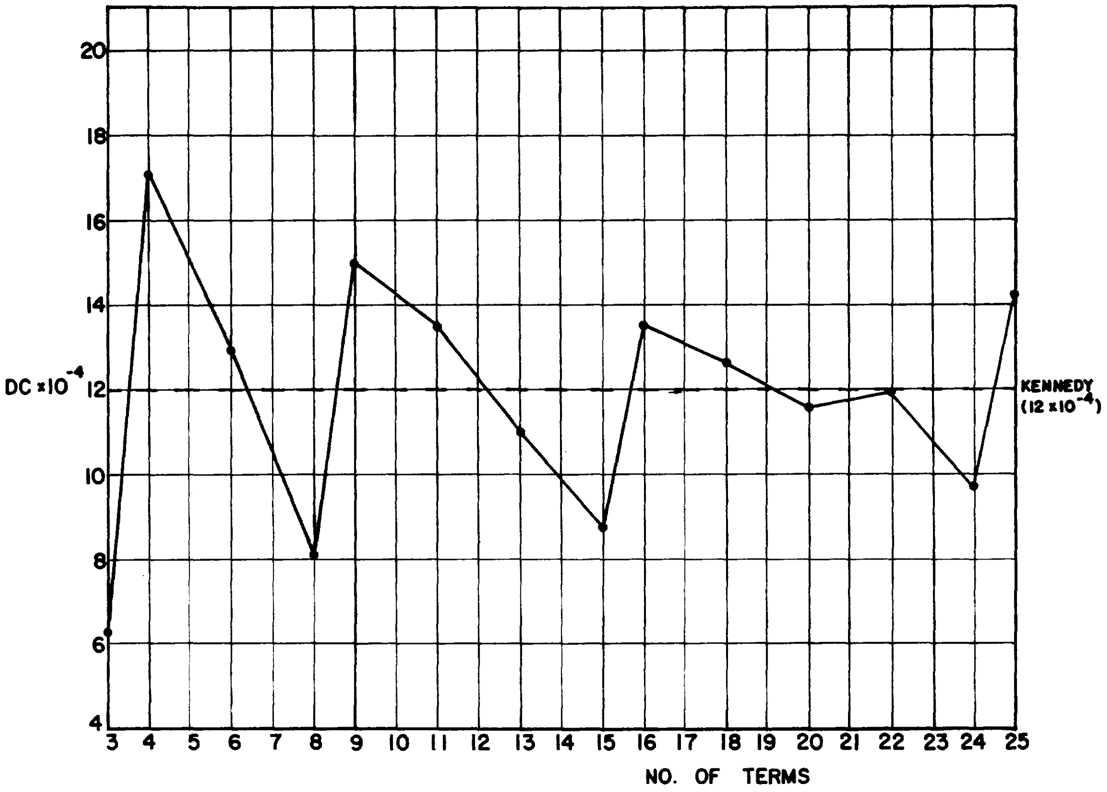
This work forms part of a research program supported by the Department of University Affairs of Ontario, Ontario, Canada.

## REFERENCES

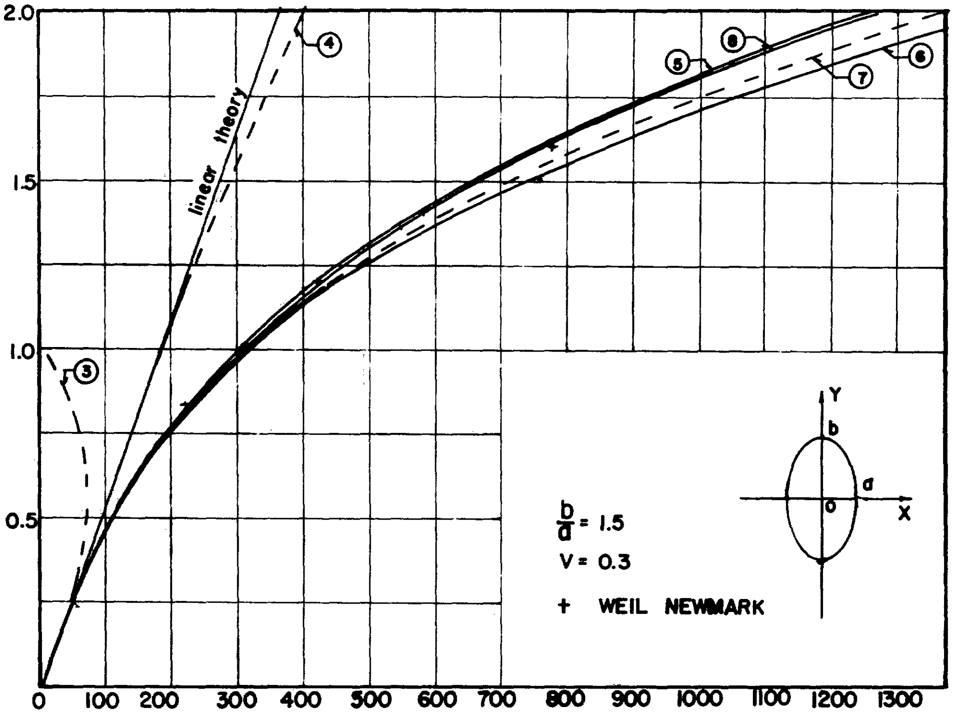
1. S.P. Timoshenko and S. Woinowsky-Krieger, "Theory of Plates and Shells", Second Edition, McGraw-Hill, 1959.
2. Theo. von Karman, "Encyklopadie der Mathematischen Wissenschaften", Vol. 4, p. 349, 1910.
3. J.B. Kennedy and S.F. Ng, "Linear and Nonlinear Analyses of Skewed Plates", Journal of Applied Mechanics, Vol. 34, Series E, No. 2, p. 271-277.
4. M. Van Dyke, "Perturbation Methods in Fluid Mechanics", Academic Press, New York, N.Y., 1964, p. 27.
5. S.F. Ng, and J.B. Kennedy, "On the Application of Variational and Perturbation Methods to the Solution of Linear and Nonlinear Elastic Problems in Engineering", Proceedings of the Fourth International Conference on Applied Mathematics to Engineering, Germany, June 1967.
6. J.B. Kennedy, "On the Bending of Clamped Skewed Plates Under Uniform Pressure", Journal of the Royal Aeronautical Society, May 1965.



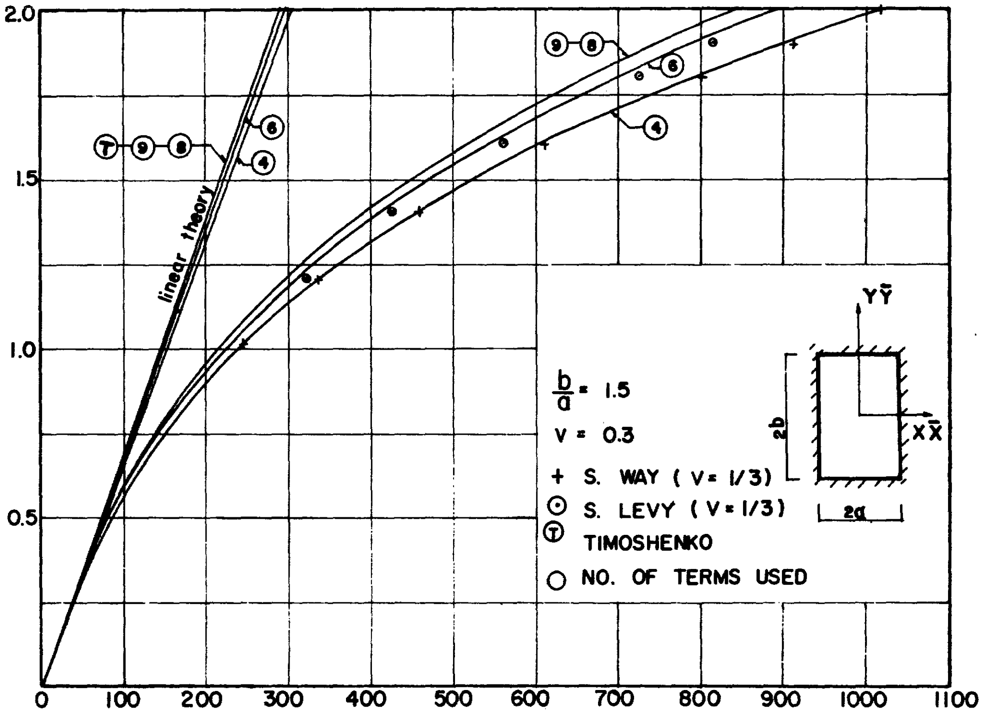
1. Pattern of Convergence of Clamped Circular Plates (Linear Theory)



2. Pattern of Convergence of a 60° Skewed Rhombic Plate (Linear Theory)



3. Pattern of Convergence of an Elliptical Plate (Nonlinear Theory)



4. Pattern of Convergence of a Clamped Rectangular Plate (Nonlinear Theory)

# A GENERALIZATION OF COMPUTER-ASSISTED INSTRUCTION

Dr. Ludwig Braun
Polytechnic Institute of Brooklyn
Brooklyn, New York

## ABSTRACT

A description is given of the Huntington Computer Project, its objectives, and its methods of operation. Its objectives are:

1. To explore the potential impact of the computer on learning in high school courses in biology, chemistry, physics, mathematics, and social studies. In this Project, the computer is used as a highly flexible laboratory rather than as a "programmed-instructional" device.

2. To explore the relative merits of time-shared and stand-alone computing.

3. To attempt to determine the differential effect, if any, of socio-economic conditions on the learning experience of participating students.

Some of the programs already written and under development are described; also, a compiler and operating system which implements the full capability of BASIC on a PDP-8/I is described.

## CAI -- A DIFFERENT APPROACH

### 1. Introduction

Conventionally (if one can use that term in such a new field), the term CAI (computer-assisted instruction) has been employed to describe the use of a computer in a programmed-instructional environment. In this application, the educator lays out a programmed lesson as a series of statements and questions. The computer is programmed to present the text and questions to the student in a step-by-step manner. The answer to the present question is used by the computer to determine the next piece of text and the next question. In some CAI systems, the computer also controls the presentation of visual cues with slides and auditory cues using a tape recorder.

Professor B. F. Skinner, one of the pioneers in programmed instruction, has been quoted in Computerworld to the effect that a "simple programmed workbook will do what the computer can do at one-tenth the cost". This author does not wish to add to the controversy -- the validity of claims and counterclaims will be resolved as we gain experience in this field; however, it does seem clear that conventional CAI takes advantage only of a small part of the power and flexibility of the modern digital computer.

Professor W. H. Huggins, of Johns Hopkins University, has suggested that the digital computer is a general-purpose laboratory device which may be used to create real or imaginary environments which the student may explore, preferably in an interactive mode. Utilizing the computer in this manner, appears to take full advantage of the capabilities of the digital computer.

The Huntington Computer Project, to be described in this paper, was conceived to explore the potential which the latter mode of computer utilization has to offer to the high-school teacher. The Project has been funded by the Office of Computing Activities of the National Science Foundation.

### 2. Objectives of Huntington Computer Project

There are three major objectives of the Huntington Computer Project. These are:

a. Verification of the hyposthesis that the digital computer may be used as a general-purpose modelling device, rather than merely as a device for solving mathematical equations. Used in this way, the computer permits the teacher and the student to explore situations which otherwise would be inaccessible because of time required, cost, danger, complexity, requirement for elaborate equipment, or needed expertise. This hypothesis will be tested in regular courses in biology, chemistry, mathematics, physics, and social studies.

b. Exploration of the relative merits of time-shared and stand-alone computing. Over a three-year period, purchase of a Digital Equipment Corporation PDP-8/I, with a 4096-word core and a 32000-word disk, costs about $21,000 including maintenance. The cost of time-sharing over the same period, from the most inexpensive service company (Call-A-Computer Company of Raleigh, North Carolina) of which the author is aware, also is about $21,000 including communication costs. The stand-alone machine has the clear economic advantage, because it represents purchase of a piece of capital equipment, rather than a service. The time-sharing has the advantage that the participating schools can access the same program library, that there is more computing power available, and more storage.

c. Exploration of the relative motivating capability and learning effectiveness which the computer offers to schools in communities with widely-different socio-economic backgrounds. The participating schools are in communities ranging from upper-middle income to poverty-level.

3. Participating Schools

All of the participating schools are in Suffolk County, New York. There are thirteen schools in all:

Cold Spring Harbor High School
Commack High School
John H. Glenn High School
Half Hollow Hills High School
Harborfields High School
Holy Family High School
Huntington High School
Longwood High School
Northport High School
Patchogue High School
Setauket High School
Walt Whitman High School
Wyandanch High School

There are 45 high-school teachers and about 1200 students participating in the Project.

Nine of these schools have been supplied with time-shared terminals into a GE 265 computer owned by Call-A-Computer Company, two have been supplied with Digital Equipment Corporation PDP-8/I computers, one has been supplied with both time-sharing terminals and a PDP-8/I, and one is using its own IBM 1130.

4. Teacher Preparation

Only two or three of the 45 participating teachers had any experience with computers prior to the initiation of the Project. A six-week summer institute was held during the summer of 1968 in order to prepare the teachers to work with the computer during the 1968-69 school year.

During this institute, each day was divided into three parts: the first was a two-hour lecture on the programming language, programming technique, or computer-related mathematics; the second was devoted to a two-hour discipline-oriented seminar led by a faculty member from the discipline (there were five such groups -- biology, chemistry, mathematics, physics, and social studies) who explored their curricula with the teachers to uncover places where the computer might be used fruitfully, and who reviewed with the group each program as it was developed; and, third, a laboratory period during which the teachers got hands-on experience on the time-shared computer through one of sixteen teletypewriter terminals, and during which they were able to de-bug and test the performance of their programs. This latter phase was of indefinite duration. The computer terminals were available 22 hours each day, seven days each week, and were used frequently on weekends, past midnight, and before six in the morning.

The teacher preparation is continuing during the school year through monthly general lectures on programming and other computer-related topics, twice-monthly seminar meetings in subject-matter groups, and through individual assistance by a circulating consultant.

5. Examples of Programs

By January 1, a total of 80 programs has been entered into the system library (including a Christmas message from Snoopy), and more than 50 others are under development. Among these are:

a. DROS**. This program determines the genetic characteristics of the offspring of a pair of drosophila flies with specified traits.

b. PAV-7*. Here the student learns about the doubling of a colony of paramecia using a gambling-casino environment for motivation.

c. DAMMA*. The boy student develops an intuition about metric units by expressing the critical measurements of his ideal girl in centimeters, meters, and kilograms, and is told what she looks like in English units.

d. PAV-4*. The student specifies the angle of incidence of a light ray at a boundary and the refractive indices of two materials, and the computer plots the indicent and refracted rays.

e. EPIDM*. Epidemic model which permits student to vary size of population, number initially infected, percent immunized, and infection and recovery rates. He then receives a plot of the course of the disease.

f. CAC11*. A simulation of Young's 2-slit experiment.

g. EDER3*. A model of the economy of the U.S.A. dealing with the flow of goods and services.

In addition, there are mathematics programs covering the curriculum from tenth grade through twelfth, ecology programs, programs on chemical equilibrium, and on a variety of physics problems.

6. Evaluation of Project

Since enthusiasm and personal biases tend to color the opinions of participants in an experiment such as the one described here, the Psychological Corporation has been engaged to prepare an independent evaluation of the extent to which the objectives of the Project have been fulfilled.

7. Development of BASIC Compiler

The time-sharing language chosen for use in this Project was BASIC, because of the ease with which it is learned, and because it appears to be the universally-accepted language in high schools. Easy interchange of programs between the time-sharing and stand-alone schools was desired, which required the availability of BASIC on the PDP-8/I computers. Only a subset of BASIC is available for these machines, and, so, a decision was made to develop a compiler which would be compatible with the BASIC available on the time-shared computer. In order to make this compiler as powerful as possible, it was necessary also to write a new operating system.

The configuration for which the compiler and operating system were written is a PDP-8/I with a 4096-word core, and a 32000-word disk. The compiler has the following features:

a. It has all BASIC system commands (NEW, OLD, RENAME, SAVE, LENGTH, etc.).

b. It has all BASIC operations (LET, FOR-NEXT,etc.)

c. It contains all built-in functions except TAN.

d. Its accuracy is 1 part in $2^{23}$ rather than 1 part in $2^{35}$, because of word-size difference.

e. Maximum program size is 6144 characters as in regular (Dartmouth) BASIC.

f. Maximum useable statement number is 4095 rather than 99999.

g. Maximum array space is 3600 characters, and maximum number of statements is 330; however, these can be traded off against one another at the rate of 2.5 array elements per statement.

h. There are no matrix operations.

i. There are no EDIT commands (EDIT MERGE, EDIT DELETE, etc.); however, EDIT RESEQUENCE is under development.

j. There is a set of error messages to signal compilation errors (e.g., unbalanced FOR-NEXT group, no END statement, uneven parentheses, etc.), and a set for execution errors (e.g., program out of data, log of negative number, division by zero, etc.).

As soon as the documentation is complete, the compiler and operating system will be released to DECUS.

## 8. Background

The program outlined here is patterned after a similar computer experiment initiated during the 1967-68 school year within the Engineering Concepts Curriculum Project (ECCP), a National Science Foundation-sponsored curriculum-development project. Within the ECCP computer project, the computer is used in the following ways:

a. As a general-purpose modelling device (as described earlier in this paper).

b. To help the students to understand the culture of computers (i.e., an awareness of what computers can -- and cannot -- do, in order to contribute to the students' technological literacy).

c. To assist in the development of an understanding of algorithmic problem formulation.

d. To enhance the students' precision of thought. (The development of programs for a computer is very valuable in developing analytic thinking.)

e. To provide a substantial increase in the students' motivation.

During the 1967-68 school year, eight high schools located in Connecticut, New York City, Long Island, Northern New Jersey, and Maryland, participated in this experiment. The results of the first year's effort was sufficiently encouraging that the ECCP Directors and the National Science Foundation approved a continuation into the 1968-69 school year, and an expansion to 20 schools.

A report describing this computer project, and the programs which were developed, is available from the Engineering Concepts Curriculum Project, at the Polytechnic Institute of Brooklyn.

## 9. Acknowledgements

The financial support of the National Science Foundation through Grants GJ-79, and Grant GE-5973, is gratefully acknowledged. The author received a great deal of advice in formulating the Huntington Computer Project from Doctor Balomenas and Mr. Melmed of the Office of Computing Activities of the National Science Foundation. The support provided by Dr. Marian Visich, Jr., the Assistant Project Director of the Huntington Project, has been invaluable.

# THE COMPLEAT COMPUTER ENGINEERING LABORATORY

Fred F. Coury
Department of Electrical Engineering
The University of Michigan
Ann Arbor, Michigan

## ABSTRACT

This paper describes a laboratory facility currently in use in the Department of Electrical Engineering at The University of Michigan. This facility provides for a wide range of laboratory experiments and research into many aspects of digital computer engineering.

Equipment available includes small scale digital computers, analog computers, logic labs, and data sets. The laboratory also contains two unique devices interfaced to a PDP-8. They are the "micro-8" (a device for external control of PDP-8 internal micro-operations) and a powerful patchboard-oriented logic breadboard device.

The paper also describes how this equipment is integrated with a sequence of computer engineering courses offered in the Department to provide extensive laboratory experience in such areas as small computer programming (PDP-8 and Linc-8), computer organization and operation (micro-8), logical design (logic labs and special patchboard device), and hybrid computer systems (Linc-8/AD-24).

## INTRODUCTION

In the development of a computer engineering program, the need arises for a "hands-on" computer engineering laboratory facility. The student specializing in computer engineering needs more intimate contact with a computer than a remote teletype terminal or the program input/ output window of a busy computing center. The study of switching theory benefits from practical experience with real switching circuits; and the actual solution of realistic design problems for a small-scale system - from system specification, through detailed design, to actual construction, debugging, and demonstration - provides a solid foundation for advanced studies in large scale system design. Such practical experience gives the student an intuitive feeling for the topics which are discussed in class in a more theoretical vein, and allows him to view particular areas of study with a better perspective.

## PURPOSE

The facilities of the Digital Computer Laboratory in the Department of Electrical Engineering at The University of Michigan have been developed to provide extensive "hands-on" experience in the following areas:

1. Small computer programming
2. Computer organization and operation, and
3. Logical design.

The laboratory facilities are used in conjunction with regularly scheduled courses, and are available for undergraduate and graduate research projects as well as to the faculty and staff for special applications.

The intent here is to provide opportunities for students which are not available elsewhere on the campus.

## DESIGN CRITERIA

The design of such a facility is subject to the following constraints:

1. It must afford many student "hands-on" contact with the equipment in parallel and for as long a time as possible. Scheduled laboratory sessions typically last three hours and involve as many as fifteen students. When doing a programming exercise, for example, it is desirable that as much as possible be done off-line (e. g. source tape preparation and listings) and that the actual time on the machine be used as efficiently as possible.

2. The student-to-student transitions within a given session or between sessions must be optimized. Equipment setup and takedown time must be kept to a minimum.

3. The overall system must be flexible. It must allow for very simple investigations and rather sophisticated projects to be carried on at the same time or alternated on a random basis.

4. The components of the system must be mobile, both to add flexibility and to allow certain subsystems to be connected and disconnected with minimum effort. This allows, for example, instructors to actually bring a small computer into a lecture room for demonstration purposes.

5. The individual component cost must be kept to a minimum. In most cases it is better for four students to carry out independent investigations on rather simple devices than to have three students watch the fourth doing an experiment on a rather sophisticated device.

6. The equipment should be as near state-of-the-art as possible.

7. The equipment must be maintainable. The failure of a crucial component can cause up to forty-five precious student-hours lost in a single afternoon. Backup systems, preventive maintenance, and immediate malfunction reporting are absolute necessities.

## DESCRIPTION

### Computers

A block diagram of the laboratory system is given in Figure 1. The system is built around two DEC computers, a PDP-8 donated by DEC and a Linc-8 purchased by the EE department with matching funds from NSF.

Each CPU and its corresponding teletype has been connected to a low speed dataset through a rather simple adapter[1], thence to the switched telephone network. This adds flexibility to the system and aids in maintenance. It also provides at least a superficial introduction to data communications equipment. Each student using either computer operates this simple, yet complete, full-duplex system. He sees that each CPU and teletype has its own telephone number and he can set up any desired interconnection by merely dialing one dataset from another.

The lab also contains two more ASR 33 teletypes. These are used primarily for off-line tape preparation and listings, but they are also used in hardware experiments and can be directly substituted for the dataset teletypes in case of malfunction.

### Switched I/O

In order to provide for programming experience at any reasonable level of efficiency, high speed I/O on both machines is an absolute necessity. However, due to budget constraints, only one high speed paper tape reader/punch could be purchased. Rather than being committed to one machine, it is connected to a special set of interface lines which can be connected to either processor's I/O bus by means of a single toggle switch (and two ECI 48-pole-double-throw relays). This approach has proven to be quite effective, since students generally use high speed I/O somewhat less than 50 percent of their allocated machine time.

This switched I/O interface is also used to share other peripherals between the two computers. An example of this is a high speed dataset and adapter[2] which can be connected to either computer via the switched interface. This provides computer-to-computer communications capability for graduate research projects. Either computer can also be used to test high speed data links at other installations. Simply load in the appropriate operating system into, say, the PDP-8 and switch the interface to the PDP-8. Then dial the PDP-8 from a teletype located at the remote installation. Using this scheme, a single operator at the remote station can control and monitor both ends of the link without the need for another operator in this laboratory.

Interfacing to the switched I/O lines is no different from interfacing directly to either computer. Pin connections, voltages, loading, driving, clamping, and daisy-chaining conventions are identical. Each CPU is connected to the switched I/O panel via a twenty foot cable. The individual interface lines are available for connections both at the computers and at the switched I/O panel. These points are shown as large arrowheads in Figure 1.

### Analog Facilities

Two Applied Dynamics AD-24 analog computers are also provided for analog experimentation. In addition, a simple level converter allows for the interconnection of the Linc-8 to one (or both) of the AD-24's to form a simple hybrid computing system.

### "micro-8"

Special control lines are brought out of the PDP-8 to allow for its use with the "micro-8."[3] The "micro-8" is a device which allows its user to cause basic micro-operations to be executed within the adjoining PDP-8. The "micro-8", with its illustrated front panel and functionally located pushbuttons is a useful device for graphic classroom demonstrations of the principles of computer organization and operation in the introductory computer course. For example, the concept of a "core memory cycle" is reinforced by actually going through a memory cycle step-by-step. The "What happens if... " questions can be answered by direct observation rather than abstract reasoning, and students come away with a deeper understanding of the topics demonstrated.

It is also very effective in the "hands on mode" in which students attempt to perform a certain complex operation (such as loading the RIM Loader) as a sequence of basic micro-instructions.

Such an exercise is especially valuable, since the student is actually simulating a computer's control unit at the level at which he may some day be designing a control unit. At this level "nobody does anything for you." That is to say, if the student does not push the button to rewrite the memory buffer into core after a core read half-cycle (or if the designer does not explicitly provide a signal to do so), the original contents of the core location are lost forever.

The "micro-8" teaches students to think in an orderly fashion and to check at each step for details which must be taken care of at that time. These are two qualities necessary to become a good design engineer. It also provides unique insight into control unit operation since the approach is basically one of synthesis rather than the more common analysis.

Logical Design

The facility includes four DEC R-series logic labs which have been and continue to be used extensively for relatively simple stand-alone experiments in logical design.

More complex design projects and experiments which involve designing computer peripherals require a more powerful breadboarding device, one which can be permanently interfaced to a small computer. Such a device must contain more modules than are normally available on a DEC logic lab. And, since it is permanently connected to the computer, it must be time-shared in a very efficient manner. Such a device, in the form of a patchboard-oriented logic lab[4] was developed and is currently being used in the lab. Although intended primarily

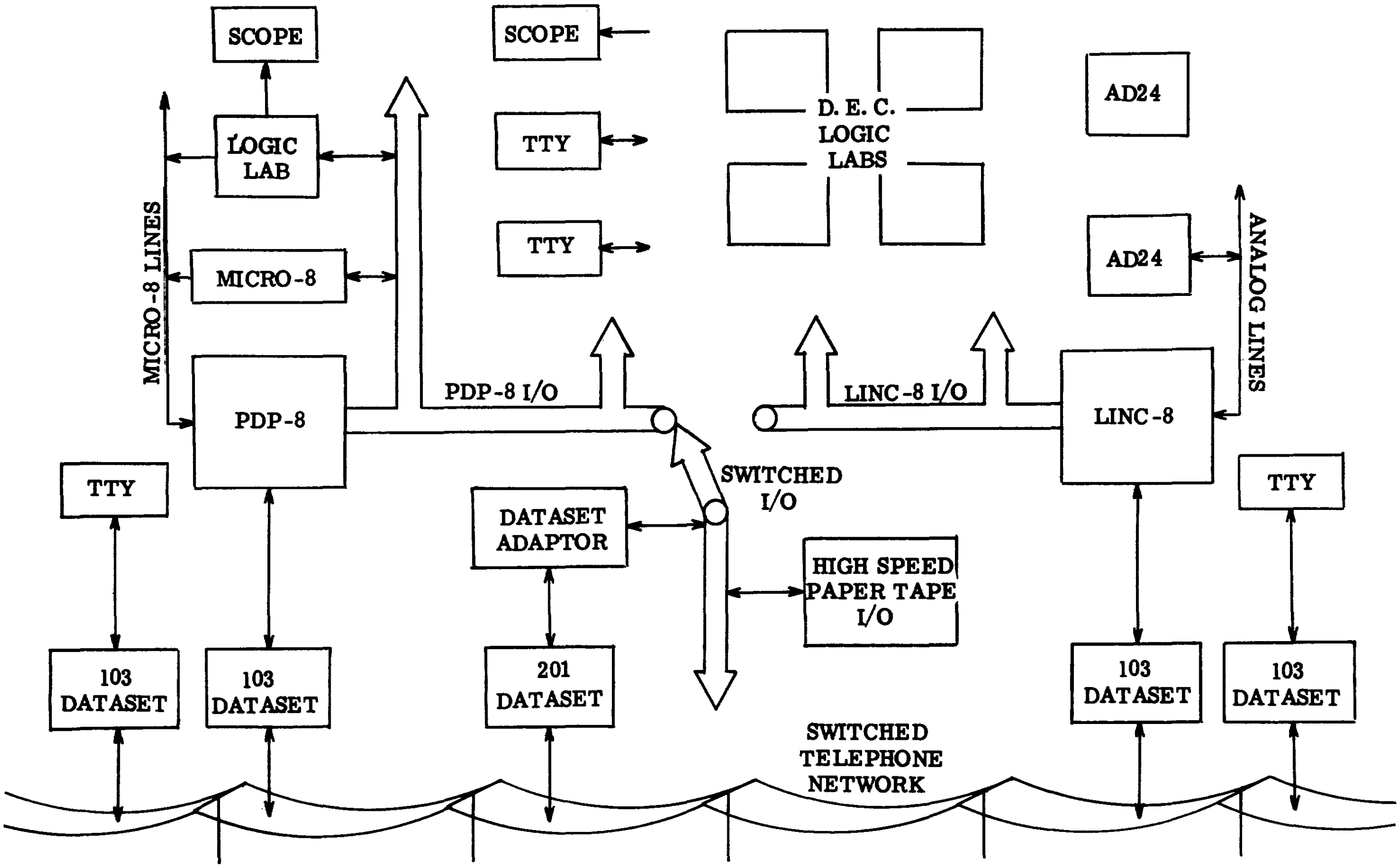


Figure 1. Block Diagram of Educational Laboratory System

as a prototype device to point the way to bigger and better things, it has become a very useful device in itself and should continue to be useful for some time to come.

## COURSE INTEGRATION

The Electrical Engineering Department offers an introductory course which surveys the field of computer engineering. A three-hour-per-week laboratory session is included in the course, and deals primarily with small computer assembly-language programming. (A comprehensive sequence of large-scale computer programming courses is offered in the Mathematics Department.) All program tape preparation, assembly, and debugging for this course is done using the two PDP-8's (one in the Linc-8) and other laboratory equipment described above. Some simple logical design work is also assigned to be done on the DEC logic labs.

Students interested in further work in digital design may then elect a laboratory-oriented course stressing practical design problems. In this course, students use the DEC logic labs to investigate some of the non-ideal characteristics of real switching devices, then go on to design some useful devices. The "micro-8" is used to gain insight into the actual operation of a real computer's control unit while, at the same time, students are building what amounts to active registers of a simple computer.

Finally, the students do some rather sophisticated experiments involving the design of computer peripheral devices. Here the emphasis is placed on the hardware-software interface. These experiments are carried out using the PDP-8 and the patchboard logic lab.

## CONTINUITY

Note how the PDP-8 provides continuity throughout. In the first course, the student becomes very familiar with the PDP-8's outward behavior and attains a reasonable level of proficiency in PDP-8 programming. In the second course, the student looks at the PDP-8 through the eyes of a control unit designer when experimenting with the "micro-8". Finally, the student becomes a peripheral unit designer, involved in PDP-8 I/O interface characteristics and hardware/software interaction.

## OTHER USES

Besides being used in these courses, the facilities are also available to graduate students and undergraduates for research projects. The DEC logic labs, Applied Dynamics AD-24's, and the PDP-8/"micro-8" are mounted on wheeled tables for use in classroom demonstrations. The equipment is also available to faculty and staff members for computer programming and logic circuit breadboarding applications.

Figure 2 shows the DEC logic labs, spare teletypes, and some of the workspace available in the lab. It also points up the fact that the room can also be used for lab lectures and demonstrations.

Figure 3 includes the two computers, their teletypes, and associated datasets. The high-speed paper tape unit and interface switch are mounted on the table between the computers, and their associated hardware, as well as the 201 line adapter, are mounted in the pedestals below.

Figure 4 is a detail of the switched interface panel. The two "shoes" bring in the computer I/O interface leads and the three (PDP-8, Linc-8, and switched) interface connectors are visible.

Figure 5 shows the same PDP-8 as above on its movable table with the "micro-8" and patchboard logic lab nearby.

## FUTURE PLANS

In spite of our attempts to optimize the facility for multi-student use, equipment access is still a major constraint. Current plans include making the lab facilities available on a signup basis from eight a. m. to midnight each weekday. Priority is given to regularly scheduled classes during the day, and student research projects at night.

Future expansion is planned in three directions:

1. The only way to further increase "hands-on" access to small computers is to get more small computers. Simulation of many small machines on a larger system defeats the purpose of the lab. We will probably add a number of PDP8/L's with inexpensive or shared high speed I/O to solve this problem.

2. We plan to take a new approach to laboratory experiments involving relatively small-scale logical design problems. Rather than have students come to the lab, we are developing small, inexpensive, self-contained integrated circuit logic labs which the students can take home with them. This approach offers many advantages to the student and greatly reduces the scheduling problems in the main laboratory facility.

3. For more sophisticated design projects, we are developing a large-scale logic lab based on experience gained on our current patchboard system. This new device will contain its own core memory, a large number of integrated circuits, and extensive I/O capabilities. It will be time-shared in much the same manner as our current patchboard device. Using this scheme, it will be possible for several groups of students to actually build and demonstrate several complex systems (a variety of small computers, display terminals, data communications devices, etc.) in parallel and without interference between groups.

## REFERENCES

1. Lundstrom, S. F., Dataphone Interface, CONCOMP Memorandum, University of Michigan, 1966.

2. Wood, D. E., A 201A Data Communication Adapter for the PDP-8: Preliminary Engineering Design Report, CONCOMP Memorandum 15, University of Michigan, 1968.

3. Coury, F. F., The "micro-8", DECUS Proceedings, Fall 1968.

4. Coury, F. F., A Patchboard-Oriented Digital Logic Breadboard, DECUS Proceedings, Fall 1968.

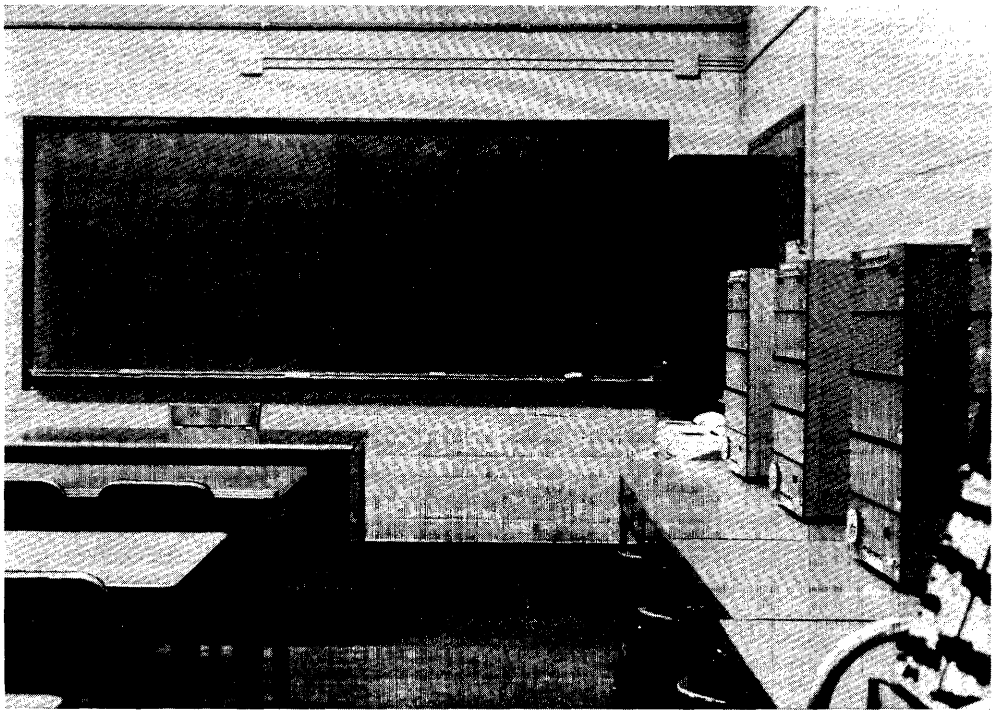


Figure 2.

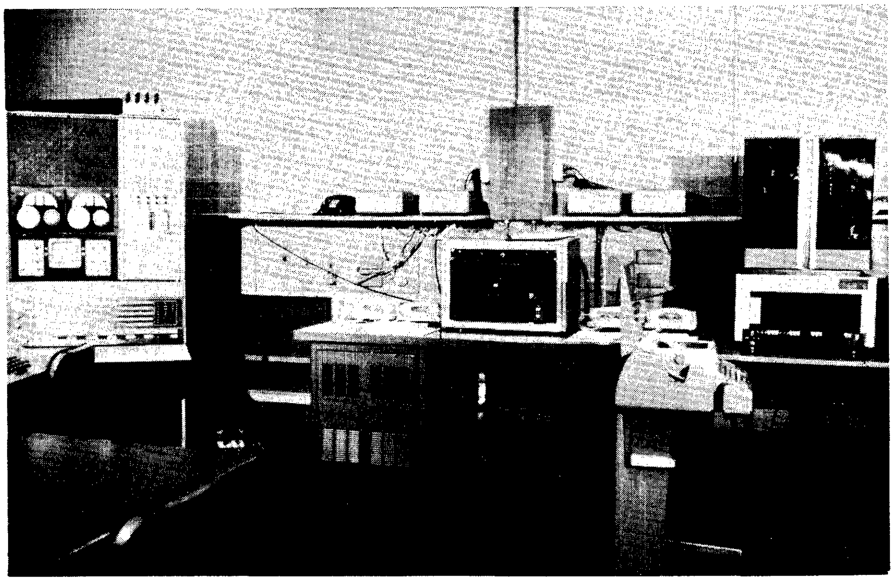


Figure 3.

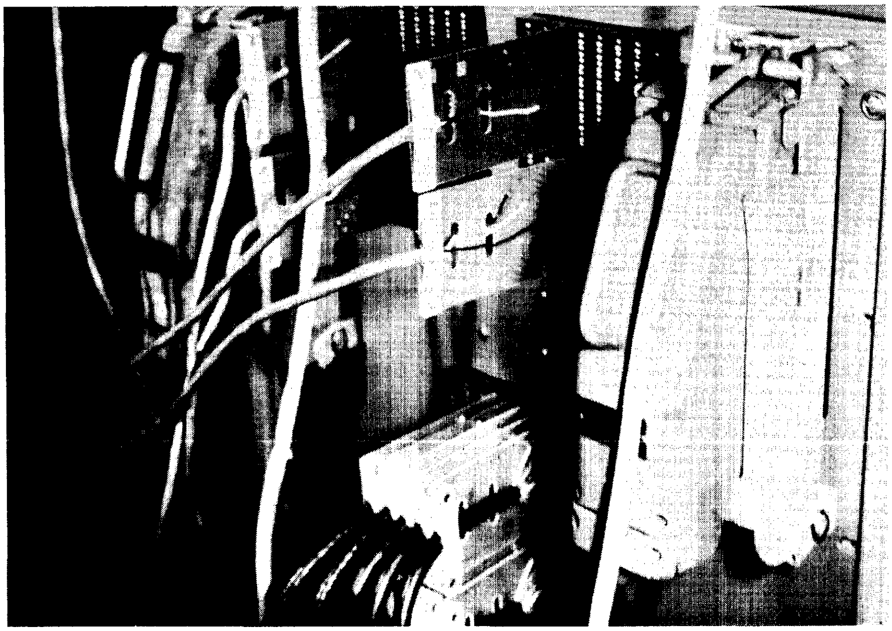


Figure 4.

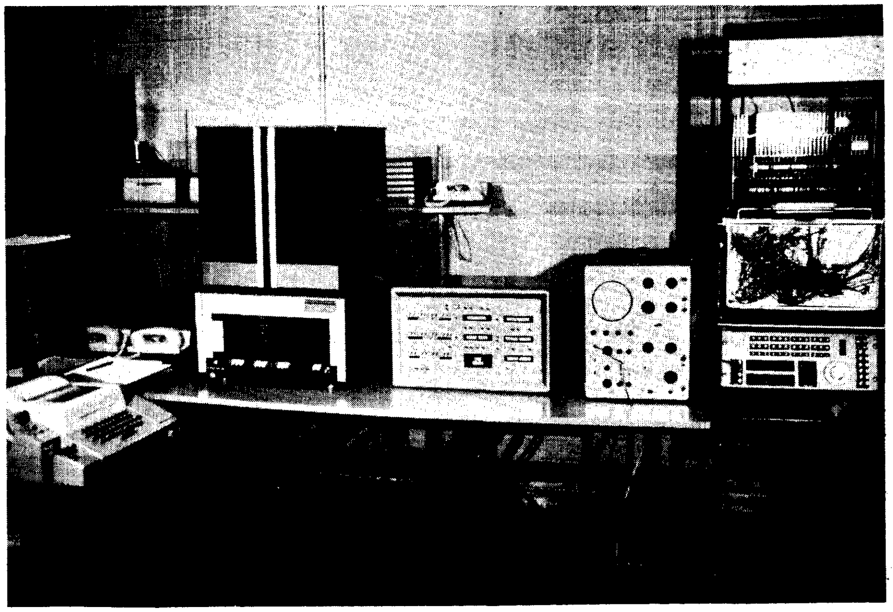


Figure 5.

# THE "micro -8"

Fred F. Coury
Department of Electrical Engineering
The University of Michigan
Ann Arbor, Michigan

## ABSTRACT

The "micro-8" is a device designed to demonstrate and provide insight into the detailed internal operation of a digital computer, specifically, a PDP-8. It consists of a pushbutton control panel, minimal internal circuitry, connecting cables, and wiring additions to a standard PDP-8.

The pushbuttons are so arranged on an illustrated front panel, outlining the major functional blocks of the PDP-8, that they show the micro-operations which can be performed on and between the various blocks. Pushing the appropriate button causes the desired operation to actually be performed within the PDP-8.

Toggle switch registers simulate data input buses, and the results of an operation are visible in the PDP-8 console indicators. External logic can be used in place of the pushbuttons, allowing student-designed control units to manipulate the PDP-8 registers.

When the "micro-8" is disabled, it has no effect on the standard operation of the PDP-8 to which it is connected.

## INTRODUCTION

The "micro-8" is a teaching aid designed and constructed in the Department of Electrical Engineering at The University of Michigan. It allows its user to selectively initiate basic hardware micro-operations in its adjoining PDP-8. In this way a student or instructor can effectively simulate the control unit of a small computer and generate more complex operations as sequences of these micro-operations. It is currently in use as a classroom demonstration device and as a laboratory tool in computer courses offered in the Department.

## HARDWARE

The "micro-8" consists of four major parts: the illustrated pushbutton control panel, some internal logic circuits, interconnecting cables, and additional wiring in the adjoining PDP-8. It is not the intent of this paper to go into the detailed circuit description of the "micro-8", rather to describe its operation and use. Two design features, however, should be mentioned here:

1. The "micro-8" was designed in such a way that when it is turned off it has no effect on the normal operation of the PDP-8 to which it is connected.

2. The control signals can also be generated by external hardware "daisy-chained" through the "micro-8" pushbutton panel. This allows students to construct their own hardware control units to manipulate the PDP-8 registers.

## OPERATION

The "micro-8" pushbutton control panel is shown in Figure 1. The three twelve-bit toggle switch registers on the left allow the user to simulate data on the accumulator, memory buffer, and memory address register input lines. The PDP-8 registers (the link, accumulator, memory buffer, memory address register, front panel switch register, and the program counter) as well as the core memory are shown as large blocks. Each operation which can be performed on or between blocks using the "micro-8" is represented by an arrow. The pushbutton which causes that operation is placed next to the arrow, and labeled as to the type of operation performed. The types of operations are as follows:

Clear
Complement
One's transfer (OR)
Zero's transfer (AND)
Rotate
Half Add (EXOR)
Generate carry

Increment
Jam Transfer
Shift

The data to be right-shifted into MBØ is determined by the setting of the toggle switch adjacent to the "shift right" pushbutton. For completeness, a list of the "micro-8" operations is given in the appendix.

The toggle switch marked "ON" in the lower left-hand corner functionally connects the "micro-8" to the PDP-8 when in the up position. When down, the "micro-8" is effectively disconnected, allowing the PDP-8 to be used as a standard machine. The pushbuttons marked "RUN" and "STOP" have the same effects as the PDP-8 console "CONTINUE" and "STOP" switches respectively.

The "DISABLE" switch is used to cause more than one operation to be performed at the same time. When it is depressed, it inhibits the effects of all other pushbuttons on the "micro-8". When released, it causes the operations specified by any and all pushbuttons currently held depressed to be performed simultaneously.

For example, in order to exchange the contents of the Memory Buffer with the contents of the Program Counter:

1. Depress and hold "DISABLE",

2. Depress and hold both "PC jam MB" and "MB jam PC" (This has no effect at this time.)

3. Release "DISABLE" (This causes the exchange.)

4. Release the other buttons.

More than two operations can be performed using this scheme.

## EXAMPLE

As an example of the "micro-8" in operation, let us examine the contents of location $7756_8$ in the PDP-8 core memory using only the "micro-8" pushbuttons. The following is a sequence of operations which will have that result:

1. Press "STOP" (if the PDP-8 is running)

2. Set the "Data Address Input" switches to $7756_8$

3. Jam the Data Address into the Memory Address Register (the MA indicators on the PDP-8 console should now show $7756_8$.)

4. Clear the Memory Buffer (all the MB console lights should now be off)

5. "OR" the core memory into the MB.

6. "OR" the MB back into core! (Destructive read, you know.)

7. Read the contents of location 7756 from the MB console indicators.

It is also possible to perform operations utilizing core memory half-cycle operations as well as complex arithmetic-logic instructions which are of particular educational value to students.

## APPENDIX

Operations on the Link (L)
1. Clear the Link
2. Complement the Link

Operations on the Accumulator (AC)
1. Clear the AC
2. Complement the AC
3. Rotate the AC (and L) left one
4. Rotate the AC (and L) right one
5. "OR" the AC Input into the AC
6. "OR" the PDP-8 SR into the AC
7. Half Add (EXOR) the MB into the AC
8. Generate Carry in the AC
9. "AND" MB into the AC

Operations on the Memory Buffer (MB)
1. Clear the MB
2. Increment the MB
3. "OR" Data Bit Input into the MB
4. Jam the PC into the MB
5. "OR" the AC into the MB
6. "OR" Core Memory into the MB
7. Shift the MB right one

Operations on the Program Counter (PC)
1. Clear the PC
2. Increment the PC
3. "OR" the PDP-8 SR into the PC
4. Jam the MB into the PC

Operations on the Memory Address Register (MA)
1. Clear the MA
2. Jam the Data Address Input into the MA
3. Jam the MB into the MA
4. Jam the PC into the MA

Operations on the Core Memory
1. Select a cell (passively determined by the contents of the MA)
2. "OR" the MB into that cell
3. Clear that cell (when executing a core memory read)

Operations on the PDP-8 Control Unit
1. Start the PDP-8 CPU
2. Stop the PDP-8 CPU

Miscellaneous Operations
1. Connect the "micro-8" to the PDP-8
2. Temporarily inhibit (disable) the "micro-8" pushbutton switches.

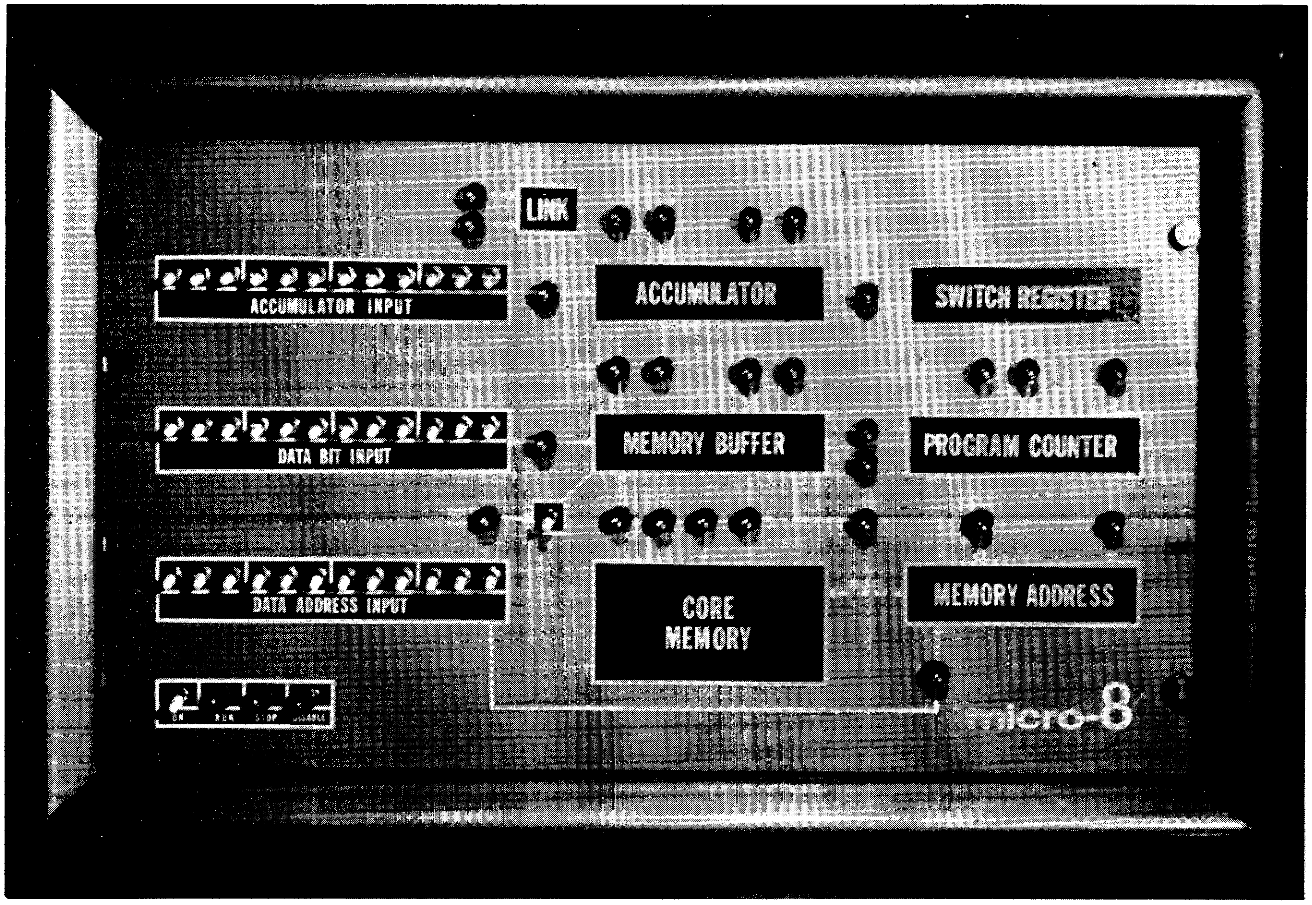


Figure 1 Photograph of "micro-8" Front Panel

# A PATCHBOARD-ORIENTED DIGITAL LOGIC BREADBOARD

Fred F. Coury
Department of Electrical Engineering
The University of Michigan
Ann Arbor, Michigan

## ABSTRACT

This paper describes a prototype patchboard breadboarding device currently in use in The University of Michigan Department of Electrical Engineering. The purpose of the device is to allow students to carry on advanced digital design projects in parallel and with minimal equipment expenditure.

The patchboard-oriented device can be compared to a standard D. E. C. logic lab in principle, but is much more powerful in many respects. It provides many more available module positions, a much greater range of support functions, a greatly expanded control panel, and access to all standard PDP-8 I/O lines.

The principal difference, however, is that all of these signals are mapped into a 34 by 66 pin patchboard receptable. This allows for off-line wiring of several devices on removable patchboards, and "time-sharing" of the main facility for on-line debugging and demonstration.

Devices built using this facility are described, and an extension of the patchboard concept is discussed.

## INTRODUCTION

In the process of setting up a facility for student experimentation in the field of logical design, the need was felt for a vehicle to allow for the construction and operation of rather sophisticated devices. A careful search of the literature for commercially available breadboarding devices failed to turn up one which met our requirements. Consequently we decided to attempt to build a device which would overcome the problems which we had encountered.

## DESIGN CRITERIA

The device in which we are interested must work in a student laboratory environment, which imposes some rather unique design constraints. Specifically, the design criteria include:

1. The device must be powerful enough to allow for the construction and operation of a variety of complete, rather complex devices. Students should be able to design and implement meaningful, useful devices such as they would encounter in industrial design projects.

2. In order to provide experience in the design of an increasingly important class of devices, computer peripherals, the new breadboard should have access to the I/O lines of a small computer whose electrical characteristics are compatible with the logic contained in the breadboard.

3. The breadboard should include provisions for a wide variety of I/O including analog-to-digital and digital-to-analog converters, and a variety of input jacks of different types. It should also provide for ease of operator communication in the form of an ample set of input switches and output indicators.

4. Such a device, with its connection to a computer and with the complexity and I/O capability described above is bound to be too expensive to provide on a one-to-a-student basis. This introduces the need to time-share the device and the requirement that this be done as efficiently as possible. In order to allow for efficient student time-sharing of a hardware device of this type, two criteria must be met:

(a) the time spent on the device must be used as effectively as possible. Any operations which can be performed off-line should be done so, and the more that can be done off-line, the better.

(b) the student-to-student transitions must be accomplished as quickly and efficiently as possible. Setup time must be kept to an absolute minimum.

5. The cost of any equipment used off-line on a one-to-a-student basis must be kept low in order to provide for as many students as possible.

6. Hopefully, the design techniques evolved can be extended to a much more powerful stand-alone device which can be used for the construction of much more complex devices including a variety of complete small-scale computers.

## THE PROTOTYPE

The patchboard technique, which is used extensively in analog computer systems and low speed digital programming applications, seemed to be an ideal solution to the time-sharing problem. A 34 by 66 pin patchboard receptacle and eleven patchboards were salvaged from a discarded computer and used as the basis for a prototype version of the new breadboarding device. Since this approach had not, to our knowledge, been used before in this particular application, the decision was made to build a prototype as quickly and inexpensively as possible. Experience gained by using the prototype in a student laboratory environment should prove or disprove the feasibility of the approach and point the way to bigger and better devices.

DEC R-series modules were used throughout because of their availability and compatibility with the rest of the lab equipment, including a PDP-8.

## CONSTRUCTION

The prototype device is shown in Figure 1. The components from top to bottom, are as follows:

1. A DEC power supply,

2. A DEC mounting panel containing support modules and PDP-8 I/O connectors,

3. A DEC mounting panel whose signal pins are mapped into the patchboard receptacle (shown with a full complement of modules),

4. The 34 x 66 pin patchboard receptacle (shown with a patchboard in place), and

5. The control panel for user interaction.

A complete list of the signals available at the patchboard is given in the appendix.

## OPERATION

Since eleven patchboards and a number of patchboards are available, up to eleven different experimental devices can conceivably be wired by different students at the same time. Since the wiring operation requires only a patchboard and a set of patchcords, the patchboards are usually taken home by students and wired within the time normally allocated for lab preparation. The scheduled laboratory periods are then used exclusively for dynamic testing and debugging.

The whole device is normally used in a "time-slice" mode. That is, a student has access to the whole device until his time slice is used up or until he finds an error which can be corrected off-line.

The transitions are very fast. All that is necessary to go on-line is to insert a patchboard (and load a program into the PDP-8, if necessary). Program support can be developed off-line using a Linc-8 with high speed I/O which is also available in the laboratory.

## RESULTS

The prototype device has been used extensively for three semesters and has met with enthusiastic student reaction. The final experiment in one laboratory course requires that students (usually in groups of two or three) design, build, test, and demonstrate a display controller. This device operates as a peripheral unit to the PDP-8 and drives a standard Tektronix oscilloscope. It uses the PDP-8 data break facility to read a series of coordinates from a display file in core, sequentially displays each point, then ceases operation and causes a program interrupt when it detects an end-of-file (denoted by an all-zero word at the end of the vector).

Hardware/software interaction is stressed in this experiment, and each student is required to write diagnostic and demonstration programs for his display. Figure 2(a-d) shows the end result of one group's efforts - an animated display produced by a device designed, built, and programmed entirely by students using the prototype breadboard device. Three weeks are allocated to the experiment and up to four different displays have been built and tested at the same time.

## FUTURE PLANS

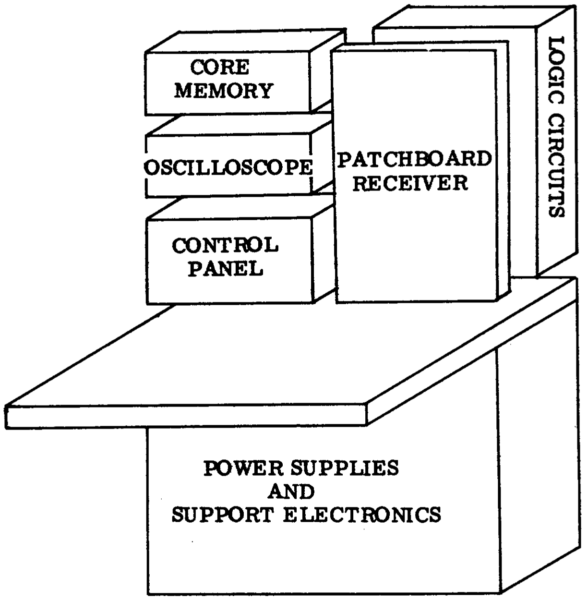
As a result of the experience gained in the construction and use of the prototype device, we have received funds from NSF and the Electrical Engineering Department of The University of Michigan for the construction of a successor to the prototype. The proposed system is shown in Figure 1. It will contain a very large number of integrated circuits and an extended control panel. Also included will be analog-to-digital and digital-to-analog converters, and a rack-mounted oscilloscope for displays and signal tracing. It will also contain its own core memory, a feature which we believe will be unique in educational devices of this type. The system is intended to accomodate a large number of sophisticated designs including small computers, data channels, display controllers, and remote computer terminals.

It is currently in the specification stage. Actual construction is expected to begin in early 1969.

## APPENDIX

### Signal Lines Mapped Into the Patchboard

1) From the Support Bay (Input/Output Pins Only)
   a) Four Three-Bit Digital to Analog Converters
   b) An Analog Comparator
   c) Three Schmitt Triggers
   d) Teletype Connector



LAYOUT OF PROPOSED PATCHBOARD LOGIC SYSTEM

Figure 1.

Layout of Proposed Patchboard Logic System

2) From Module Bay
   a) Ninety-Six PDP-8 I/O Lines (Condensed)
   b) Fifty-Eight Modules (Pins D-V)

3) From Control Panel
   a) x, y, and z Scope Inputs
   b) Four 20 K Log Potentiometers
   c) Four Switch-Selectable Capacitors (5 Values Each)
   d) Twenty-Four Indicator Lamp Inputs
   e) Twelve Toggle Switch Contacts (N.O.)
   f) Ten Pushbutton Outputs (Filtered)
   g) Telephone Dialer Contacts (2 Sets)
   h) Four Shielded Two-Circuit Phone Jacks
   i) Twelve Binding Posts

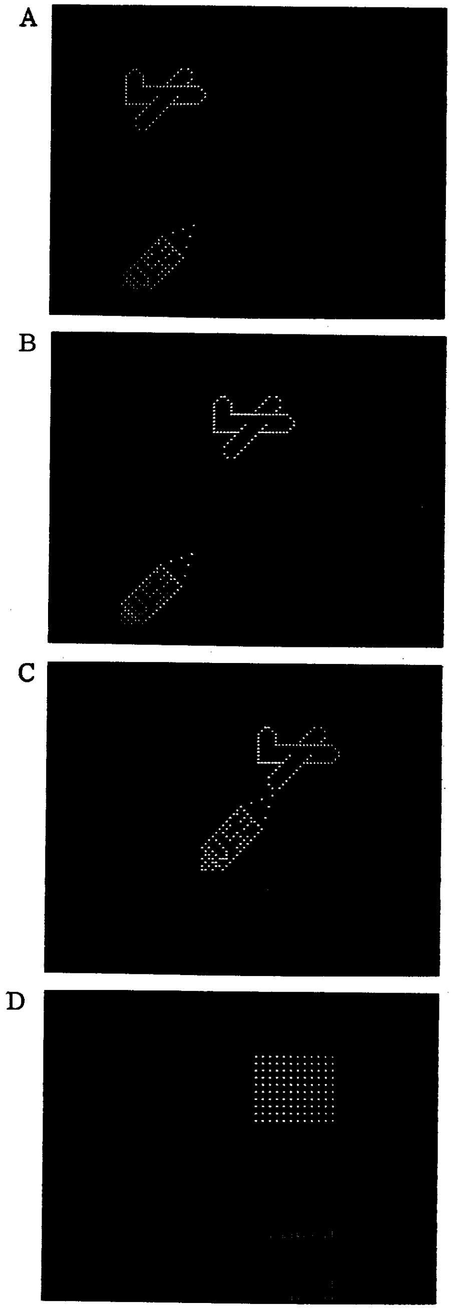


Figure 2.

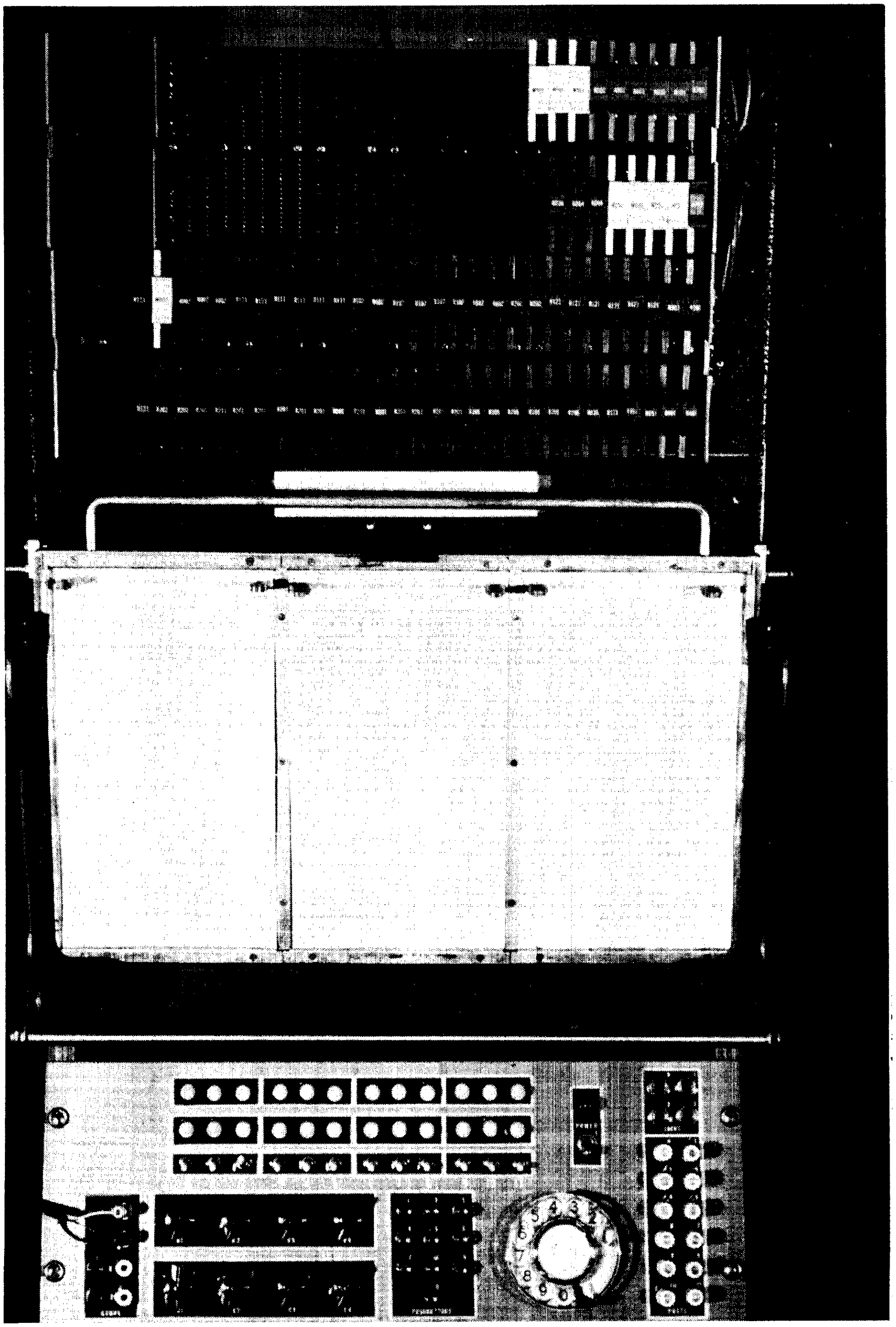


Figure 3. Patchboard Logic Lab

# BIOMEDICINE

# ARBUS - ADVANCE BED RESERVATION AND UTILIZATION SYSTEM*

Robert P. Abbott and Judith Ford
Institute of Medical Sciences
San Francisco, California

## ABSTRACT

ARBUS was originally designed to meet the two specific hospital needs as implied in the name. Subsequently, the goals were modified to include other scheduling, inventory, and communication needs within the hospital environment. The system employs the concept of a small computer at the hospital site with a communication link to a larger computer located elsewhere. Terminals located throughout the hospital are connected to a small computer—the PDP-8.

Ten years ago, the United States began to phase out the aircraft industry in favor of the missile industry. Some of the resultant surplus talent was directed towards the problems of the Medical Sciences. This shifting of technological might was, of course, motivated by that prime American motivator, money. Indeed, the gross annual expenditure in the health field exceeds $45 billion. Approximately 90% of this amount is spent on patient care. Hospitals and medical office complexes constitute the primary resource for patient care. It is then no surprise that the subject of Hospital Automation has enjoyed the attention of hardware manufacturer and software entrepreneur alike.

Let us pause for a moment and enumerate the areas within the hospital in which automation is expected to make a contribution:

Patient billing
Payroll
General ledger preparation
Inventory
Information interchange
Management and operational information systems
Diagnosis of patients (e.g. EKG, EEG analysis)
Treatment of patients (e.g. calculation of radiation dosages)
Automated clinical laboratories
X-ray analysis
Physiological monitoring of patients
Automated multiphasic examinations (i.e. screening)
Closed loop control of certain equipment (e.g. respirators, IV feeding)
Optimum scheduling of doctors, patients, and procedures
Medical records.

Let us also reflect on those techniques of the Computing Sciences which will be required to implement an integrated whole consisting of the above parts:

Time sharing
Fail safe modularity
Highly interactive remote terminals
On-line real-time signal processing
Information storage and retrieval
Mass storage (on the order of $10^{12}$ bits)
Linear programming algorithms.

To date, some abortive attempts have been made to attack the total problem. They have not been successful. The most success has been attained by attacks on individual areas. Some of these activities will be reported in the papers presented here today.

The Research Data Facility (RDF) of the Institute of Medical Sciences, Pacific Medical Center, is engaged in certain research and development activities which pertain to hospital automation. The two major activity areas are physiological patient monitoring and management and operations information systems.

ARBUS is a management and operations information system, describing the initial functtions implemented. ARBUS is an acronym, namely: Advance Reservation and Bed Utilization System. In addition to performing certain tasks, the system has the goal of reducing hospital automation costs by sharing hardware costs between more than one hospital.

The Advance Reservation portion of the system is very similar to the airline reservation systems. There is one notable difference. As far as the airlines are concerned, a seat is sold for the length of time between reservation placement and when the plane comes down, regardless of how the plane comes down. A hospital bed has a variable time period. In maternity cases, even the onset of occupancy has a variable time period.

We must make one thing clear: the system did work - on Thursdays and Fridays between the hours of 6:00 AM and 10:00 AM. It did not work very well after 10:00 AM because that's when the programmers from other time-sharing customers would get to work and swamp the system. The system would show some signs of life around 12:30 PM when some of the other customers would go to lunch. The response time was still sluggish (up to 10 minutes as compared to nearly instantaneous at 6:00 AM). The system would again disappear as the programmers for the other users would reappear from their extended lunch hours.

By the time the other users went home, the hospital would be so far behind in its transactions that it took the rest of the evening to catch up. The time sharing vendor went off the air at 11:00 PM Mondays thru Thursdays. This wasn't too bad in spite of the 24 hour a day nature of a hospital. There are relatively few transactions over night. It is not unusual for an admission office to be closed between midnight and 6:00 AM.

On Fridays, the vendor turned off at 6:00 PM. Saturday, the turn-off time was 5:00 PM, and on Sunday there was no time-sharing at all. It also turns out that Sunday is the biggest admission day of the week, so by Monday morning there was really a lot of catching up to be done. Catching up lasted Monday and Tuesday. Wednesday was spent reconciling the errors. It would be fine again on Thursday and Friday between the hours of 6:00 AM and 10:00 AM.

We thus see that time-sharing fails to qualify as a utility for two reasons: 1) it is not available 24 hours per day, seven days per week; 2) time-sharing service is overloaded (i.e. oversold) during the peak user hours. Electric utilities, by comparison, must design their circuits to function on the 21st of December. It is the shortest day of the year and the height of the Christmas season. I don't think any of you have gone without electricity on that day, nor have your Christmas three lights burned any dimmer.

When the conventional utilities make an improvement in their service to you, it is seldom noticed. Your appliances do not suddenly go kaput because of a change in service. Indeed, utility changes are invisible to the subscriber. Not so with time-sharing systems. A change may or may not be invisible to a programmer sitting at his terminal. In general, system changes are NOT invisible to a computer whose responsibility it is to count characters, search for pre-ordained sentinels, insert certain required sentinels, and activate various programs.

A so-called "invisible" change would frequently result in the complete recompilation of our entire set of programs. The programs which resided in the time-sharing computer were in excess of 6000 Fortran statements. It was no small task to recompile the lot. As you well know, an action as drastic as complete recompilation normally results from a major change in the vendor's system. It is conceivable that one such requirement per six month period might be tolerated. Anything more frequent than that is clearly intolerable. It is unfortunate that we experienced a partial to complete recompilation once every 30 days. And so we have a third reason as to why time-sharing fails as a utility: 3) it is not sophisticated enough to mask the user from system improvement procedures.

We do not intend to take any vendor to task. The vendor which was used in this experiment has, in our considered opinion, one of the best time-sharing systems currently available. Let the experience recorded here be a warning to potential users and to vendors alike that the time-sharing utility is not only not here, it is not even on the horizon. If and when it does arrive, it must stand the three tests developed herein. Since this is not the case, then ...

In the interim, the projects at hand require solutions. The economic reality of hospital automation is that the patients must bear the automation costs. Sharing a computer dedicated to hospital use seems to be the most convenient course. ARBUS,using a dedicated PDP-9 and Data Cell,will return to operation with two hospitals on or about 2 April, 1968.

*Supported by USPHS Contract No. PH 108-66-288

The Bed Utilization portion of the system provides information as to bed occupancy, forecasted bed occupancy, discharge schedules, surgical schedules, nursing station census reports, hospital census reports, utilization effectiveness reports, etc.

The system consists of terminals located at the admission office, administration office, and at each nursing station within a hospital. For economic reasons the first terminals are teletypes and Selectrics. Any bit serial terminal could be used. Typical functions which may be exercised from the terminals are: reserve, cancel, admit, transfer, expiration, discharge, census, etc. In addition, various listings are also available.

The connection between the terminals and a PDP-8 are full duplex with the exception of the Invac modified Selectrics. There are a variety of validity checks to insure that unwarranted transactions do not jeopardize the data. A nurse at station X cannot perform a transaction affecting a patient belonging to station Y. Admission clerks and administrative personnel cannot perform any transactions which would affect a patient belonging to nursing station X. Prior to updating the computer files for a transfer or discharge, the file for the bed in question is checked to see if it is indeed occupied. If it is not, appropriate messages are printed on the terminal which requested the transaction.

Transactions are initiated at a nursing station by simultaneously pressing the color coded CNTRL key and the appropriate color coded transaction key. As an illustrative example, let's say that DSCH is in red in the upper case position on the letter W on the teletype keyboard. When the nurse presses the control W, the PDP-8 responds with the date and time of day. The PDP-8 will also print: "DISCHARGE", and, finally the PDP-8 will print: "BED NO.".

As the nurse might suspect, the terminal wants the bed number of the patient being discharged. When she types, say, 123-A, the bed files associated with that terminal will then print the patient's name on the terminal for the nurse to visually verify that she is indeed discharging the correct patient. Again, assuming a valid transaction, the nurse types a period to signify verification. At this point the nurse is finished with the transaction. The computer files, however, are only now updated to reflect the transaction. If this were not a valid transaction, appropriate error messages would have appeared on the terminal.

The terminals, thus, interact with the nurse to guide her towards the appropriate actions. As a further aid to the nurse, if she cannot recall how to use the terminal, she may type: "HELP" and she will receive a short paragraph of instructions.

The PDP-8 is located on the premises of the hospital. It serves as a message collector and distributor. A special electronic interface was designed and constructed to permit the PDP-8 to service the hospital terminals. That interface is described in a separate paper being presented at this DECUS meeting. The PDP-8 is connected via telephone lines to a centrally located larger computer with appropriate storage facilities to contain 90 days worth of bed reservations, and various files describing the hospital, nursing stations, and beds. The second version of ARBUS will go on the air in April of 1969 and will use a PDP-9 attached to an IBM Data Cell as the main processing and storage units.

Two hospitals will be served initially by this configuration. This system will be described in a later paper.

The first version of ARBUS was tested using the services of a time-sharing vendor as the main processor and storage facility. The remainder of this paper will review that experience.

Our use of a PDP-8 as a data concentrator is not unique. Many other applications have used this technique. There was, however, some uniqueness in requiring that the PDP-8 function both as a data collector and as an interpreter between the lay personnel (i.e. nurses, etc.) and the highly specialized language of a programmer oriented time sharing system. As a matter of fact, in the environment of a time-sharing utility, the idea still sounds pretty good. The user, in this case a hospital, need only pay for the amount of gas, water, electricity, and computing actually used. We were, unfortunately, born about ten years too soon. The time-sharing utility does not exist today.

# DESIGN PHILOSOPHY OF AN INTEGRATED LABORATORY-HOSPITAL INFORMATION SYSTEM

Garth Thomas, Systems Research Department
The Ohio State University Hospitals
Columbus, Ohio

## ABSTRACT

The integration of a Laboratory Information System being developed within the larger framework of a Hospital Information System will be presented. Using a small LINC-8 computer to perform the required functions within the clinical laboratories and divorcing its operation from any required hospital functions, provides the maximum flexibility in its utilization with the laboratory operation. Whereas, those functions which can be performed more conveniently and economically by a central computer facility can be used to maximum advantage without any major effect upon efficient operation of the laboratory. The significant consequences, advantages, and disadvantages will be discussed within the framework of the general system design philosophy employed.

## Introduction

In as much as this paper deals with a system which is still under development it may be regarded as an exercise in wishful thinking, however, it may also serve to motivate those individuals who have not committed themselves to a specific point of view. The widespread interest in hospital and laboratory systems and extensive efforts to independently develop both systems require that they eventually be integrated in a fashion consistant with effective hospital and laboratory operation. The following discussion delineates those factors of primary importance to both systems and presents a schematic of how the two systems may be effectively combined. It is presented in terms of functions and procedures as opposed to the actual hardware and software elements required to implement the system since these two items are normally peculiar to the individual hospital or laboratory.

## Hospital Information System

Initially, the primary application of computer technology within the hospital was confined to accounting and administrative applications. Within the past few years however, there has been an increasing proliferation in hospital computer applications in such diverse areas as patient monitoring, EKG analysis, patient scheduling, and clinical laboratories. It was soon realized that the additional personnel and equipment required to support each of these independent applications would offset both in terms of cost and efficiency, any individual gains which could be realized. This observation has resulted in the construction of what has come to be known as the Hospital Information System, which basically provides a framework in which the varied applications of hospital systems can be supported in a single environment. This environment consists of a computer-based network of remote terminals located in strategic areas throughout the hospital, each serviced by its own set of application programs which in turn share a common data base, namely the patient master record, containing all data relative to the particular patient. Since each terminal is associated with a given hospital area or department, it is possible to tailor the programs which service the terminal to the requirements of the individual users, while providing the necessary mechanics of an exchange of information between application areas by applying the contraint of the common data base. Consequently, when a request for a glucose tolerance study is ordered, the clinical chemistry application programs may issue a diet hold by setting the appropriate flag in the patient master record. Later, this flag is detected by the dietary application programs and a message indicating the diet hold may then be given to the floor dietitian. Similarly, when the results of the laboratory procedures are reported, the fact that the patient has been receiving drugs which may influence the value of the laboratory determination, may be noted for evaluation by the physician. Within these two examples lies the primary motivation for the Hospital Information System in that it provides the necessary mechanics to develop a balanced and consistant program of total patient care, the mere logistics of which, when approached in a less formal manner, are clearly impossible. In addition, by automating the flow of information within the hospital complex, it is impossible to significantly reduce the volume of transactions and consequently the time, required to initiate, execute and report a given request for patient services, resulting in an increased level or intensity of patient care. The level of patient care refers to the ratio of administrative or clerical tasks to the total task of providing the necessary service to the patient, or equivalently the time available for rendering professional medical service to the patient. It is then, this combination of a program of total patient care operating in conjunction with an increased level of patient care which makes the Hospital Information System a desirable quantity and the increased efficiency of hospital operations which makes it possible. It must be realized, on the other hand, that the Hospital Information System remains in the embryonic stages of development and, as such, is something less than a reality. However, the limited successes of its early versions give every reason to accept it as a workable concept. Table 1 gives a summary of the Hospital Information System as currently defined at the Ohio State University Medical Center.

## Laboratory Information System

Analogously, there exists a variety of clinical laboratory applications such as data acquisition, quality control, test requisition procedures and patient summary reports that are susceptible to computer techniques but offer the same disadvantages in terms of cost and efficiency as before, when approached in an individual fashion. The utilization of a computer within the laboratory has been necessitated in order to process the output of the automatic instrumentation and to assist in the clerical functions associated with the ever increasing volume of laboratory tests. This increased reliance upon laboratory findings has been precipitated by the increasing body of knowledge concerning disease mechanisms and treatments to that the test volume will tend to increase as new advances are made. The Laboratory Information System concept evolved in order to accommodate these applications on an integrated basis by creating a patient file from the test requisitions, monitoring the automated instruments and providing a summary report from the information contained in the patient file as well as facilitating the various clerical and management functions of the laboratory. In order to implement the previously defined program of total patient care, the demands of the Hospital Information System and those of the Laboratory Information System must be integrated in a consistent manner so that the operation of the hospital compliments the operation of the laboratory, resulting in an increased level of available patient care services. This has been accomplished by defining an environmental control system and a mutually dependent laboratory control system, since the hospital information systems currently in development either employ stand-alone data acquisition devices or rely entirely upon manual entry to incorporate laboratory results into the system, both of which have serious deficiencies with respect to solving the problems of the laboratory operation, especially when large amounts of instrumentation are used. On the other hand, a stand-alone laboratory information system is also unsatisfactory in that the benefits of the Hospital Information System cannot be obtained and it requires the laboratory to assume responsibility for supplying computer support to the hospital.

## Environmental Control System

The environmental control system is supported by the IBM 360 Model 40 computer and uses the IBM 1050 terminal for data communications as well as the IBM 2260 data display stations. From the remote terminals, laboratory test results may be ordered, inquiry into a patient file concerning laboratory test results may be made and in those areas not serviced by the laboratory control system, laboratory tests results may be entered. For example, in bacteriology where the results are primarily textual in nature as opposed to numerical, the result may be reported in a conversational mode using the data display station rather than requiring fixed format entries using a typewriter.

Stat results can be immediately displayed upon a terminal after receipt of such results from the laboratory and in the event the patient has moved since the test was ordered, the result will be routed to the appropriate terminal. The phlebotomist on the floor may inquire for a list of specimens to be collected for the nursing unit at which she is currently located, eliminating the need for elaborate specimen collection schedules. Patient summary reporting and ward reports may be compiled using the central computer's high speed line printer eliminating the need for duplication of such equipment in the laboratory while providing equivalent service. Additional information processed by the Hospital Information System may be reported in conjunction with the patient and ward reports, as requested by the hospital staff. Processing of cumulative files of laboratory data may be performed using the extensive facilities of the hospital's main computer for long term analysis of quality control and normal values.

Management reports for use in laboratory administration may be expanded to include year to date, and other historical summaries in order to provide the laboratory director with the required information. Research projects are greatly enhanced by the availability of historical, diagnostic and clinical findings in one central location in a machine processable form. Correlation of such data was formerly impossible because of the sheer mechanics of collecting and organizing the necessary data for processing. Patient billing and laboratory income reports may be generated as a by-product of the aforementioned activities allowing the laboratory to make a more direct correlation between cost and price resulting in a more equitable charging policy. These items are summarized in Table II.

## Laboratory Control System

The LINC-8 computer is used to support the activities under the Laboratory Control System. Its primary responsibilities include data acquisition and instrumentation control, serving as a reactive element in the laboratory by monitoring the automatic instrumentation and alerting the medical technologist to any possible sources of error or malfunction. Using the resources of the computer, the laboratory has available a wider range of instrumentation techniques which would not be feasible using manual procedures. For example, techniques have been developed for driving the auto analyzer systems at steady-state increasing the effective throughput of the device and a procedure for determining amylase involving the subtraction corresponding data values for the same sample processed on separate channels of the auto analyzer can be implemented. The most obvious use of the computer is to assist in the calculation of laboratory results obtained in manual and automatic proceduces. Laboratory data files giving the laboratory personnel a single source of information concerning tests performed and tests ordered can be accessed through any terminal and results determined manually can also be entered at any convenient terminal. Short range quality control giving the controls which are out of tolerance and daily statistics for immediate review can be generated on a routine or a request basis. Research studies can be facilitated by providing a shorter turn around time than can be obtained from the central computer facilities when the data to be analyzed is available to the system and the programming already exists. The laboratory computer can also be used on an educational basis for the medical technology students who, if the current trend in laboratory computer systems continues, will be required to work with them throughout their careers. Table III summarizes these items.

## Summary

In summary, the concept of an integrated hospital-laboratory information system provides the necessary operational facilities to the laboratory and the hospital by particularizing the computer system to the requirements of the application, using the facilities of each computer to maximum advantage as opposed to forcing either or the other to satisfy all requirements of the system. The additional complexity introduced by interfacing the two computer systems is offset by the increased flexibility in application design and overall efficiency of the resulting system. The applicability of the system described above is obviously confined to those institutions in which patient care is a critical factor and the economics of scale are such, that the required computer hardware can be supported, however, the increasing availability of shared hospital computer services will make this a realistic consideration.

### TABLE I

HOSPITAL INFORMATION SYSTEM

MOTIVATION:

a) A balanced and consistant program of total patient care.

b) An increased level of patient care.

JUSTIFICATION:

a) Increased efficiency and effectiveness in hospital operation.

ENVIRONMENT:

a) Network of remote terminals each serviced by its own set of application programs.

b) A common data base termed the Patient Master Record.

STATUS:

a) Pre-admission and Admissions System

b) Medical Records Identification and Retrieval System

c) Clinical Laboratories (chemistry only)

### TABLE II

ENVIRONMENTAL CONTROL SYSTEM

1) Total Patient Care Program

2) Test Requisition

3) Result Reporting including STATS

4) Inquiries to Patient Files

5) Specimen Collection

6) Long Term Quality Control

7) Laboratory Management Reports

8) Laboratory Research Projects

9) Laboratory Billing and Accounting System

### TABLE III

LABORATORY CONTROL SYSTEM

1) Data Acquisition

2) Instrumentation Control

3) Improved Instrumentation or Laboratory Techniques

4) Laboratory Calculations

5) Establish and Maintain Laboratory Files

6) Short Range Quality Control

7) Laboratory Research Studies

# THE PDP-8/I AS A SATELLITE COMPUTER FOR BIO-MEDICAL APPLICATIONS (SYSTEMS SOFTWARE)

R. Bush, D. Domizi, R. Lee,* M. McKeown
University of Chicago, Lying-in Hospital
*IBM Corporation
Chicago, Illinois

## ABSTRACT

This paper describes the configuration and software system used to support a patient filè and an on-line fetal-monitor. The PDP-8/I (satellite) to 360/50 communication is described.

The Chicago Lying-in Hospital is evaluating the effectiveness of automated data processing in improving obstetric patient care. Although a complete hospital system is still in the future, segments of such a system can be implemented now which will interface with the full system when it arrives. Given a hypothetic future system (Fig. 1), two particular projects illustrate implementation of one section of this plan; the satellite computer and its private section of the central facility. Some of the systems software developed to support these two projects may be of general interest.

The satellite is presently configured as:

1. a PDP-8/I with 4K of core, extended arithmetic, and powerfail/restart;
2. a PT08 connection to a 360/50 at 110 baud and a remote KSR-33;
3. two TU-55 DECtapes with TC01 control;
4. a Tektronix 601 storage CRT with a VD8/I control;
5. an analog subsystem interface using an AD08B A/D converter, a KW8I/F crystal clock, twelve relays, and twelve sense-lines.

A disk will be added in the future to allow more sophisticated use of the CRT(1). The central computer is an IBM 360/50 running under release 15/16 of MVT. For the purpose of this discussion, an abbreviated view of the two projects will suffice, as more detail is available elsewhere (2,3).

The OBFILE is a patient record IS&R system which runs once a day, in batch, on the 360. It will have provision for on-line retrieval of a patient record via a remote teletype. Since the main file on the 360 is updated daily, the economics of the situation make it desirable to support the remote facility from a satellite computer.* The PDP-8 will be updated by the 360 batch program, and the teletype will access the PDP-8 subset of the file. The patient records in the PDP-8 will be stored on DECtape, one record to one 128-word block. Later phases of this project will include data acquisition utilizing the satellite.

The second project is an on-line fetal monitor which is designed to run mainly in the PDP-8. The satellite is connected via A/D, sense lines, and relays to an analog subsection which is used to monitor fetal parameters. CRT output is relayed by closed circuit television to the patient's bedside as a display for the physician. As new parameters are investigated and new algolrithms are tried, it is desirable to do the exploratory work in a high-level language on the central computer; with the satellite doing only data acquisition and display. A later phase of the project may require on-line statistic evaluation of a large data base. This would be done by the central computer.

In this discussion, the two projects are considered as users of the system.

## PDP-8 I/O

All I/O in the satellite is done by the system in response to user requests and hardware interrupts. The system is device oriented, with specialized routines to handle each I/O unit. Once a requested I/O operation has been initiated, control is returned to the user. Each device has an IN USE/COMPLETION flag which may be tested by the user. If the user issues an I/O request specifying a device whose flag is in the IN USE state, the system retains control until processing of the new request has been initiated.

*360 core is expensive, as is a large resident disk file.

## CONCLUSION

The system presented in this paper is an example of the small satellite system in a large hospital environment. With the advent of the central hospital computer system the satellite system will be completely interfaceable. Such systems may be the most economic and least performance degrading approach to special use situations in the larger hospital systems.

## BIBLIOGRAPHY

1. Programming the VD8/I Display Control. Digital Equipment Corporation.

2. McKeown, M.J., Bush, R. and Domizi, D., A Computer System For The Monitoring Of Intensive Care Obstetric Patients. Lying-in: The J. Reprod. Med. Vol. 1, No. 3. May, June 1968.

3. Burks, J.L., Bush, R. and McKeown, M.J., A Computer Based Obstetric Information System. In press.

4. ABBOTT, R., ARBUS - Automated Reservation and Bed Utilization System. Proceedings DECUS Symposium, Dec. 1968.

## ACKNOWLEDGMENTS

This work was supported by NIH Research Grant # HD 03432-01.

Computing assistance was obtained from the Health Sciences Computing Facility, UCLA, sponsored by NIH Grant F-3; and Biological Sciences Computation Center, U. of C., sponsored by NIH FR-00013 and the generous assistance supplied by the University of Chicago Computation Center Staff.

In cases in which the user is dealing with a character or a character string (ie. teletype, 360 communication, etc.), a packed 6-bit code is supported as well as ASCII.

## INTERPROCESSOR COMMUNICATION

The PDP-8 user considers all interprocessor communication as the transmission of a record of not more than 96 characters in length. He may read or write, request a record be sent from the 360 consisting of the time and date, or send a message to the 360 system console. The 360 operator may reply. All communication, as seen in core by the user, is in the character set of the appropriate machine; in the 360, EBCDIC: in the PDP-8, packed 6-bit code.

The content of all transmissions between the two processors (excluding control characters and the checksum) is in ASCII with the parity bit in the hold state. Control characters have the parity bit in the mark state (Fig. 2. Tab. 1). Translation is done by the 360 from ASCII to EBCDIC and vice versa. The types of allowable transmissions are shown in Tables 2 and 3.

Due to hardware peculiarities in the 360, a byte transmitted by the PDP-8 is seen as the mirror image in 360 core. The 360 program deals with this in two ways: 1-translation tables set up as mirror ASCII-EBCDIC; and 2- addition used in checksumming is done with the carry propagated to the right instead of to the left.

To ensure proper data transfer, all transmissions are vertically checksummed in the following manner. The last four characters of a transmission are: "CKSM", "1 XXX XXX", "1 YYY YYY", "XOF"; where XXXXXXYYYYYY* is the least significant twelve bits of the sum of all eight bits of every byte preceding the "CKSM" character. When either machine detects a transmission error, it requests the other to repeat the invalid transmission. Provision has been made to prevent looping upon multiple errors.

There are two versions of the communications package for the 360, differing only in their handling of "data" transmissions. Both versions appear identical to the PDP-8 system, and no adjustment is necessary by the system or by the PDP-8 user program.

Version "one" resides permanently in a 6K partition of the 360. At very small cost it provides "off-line"** communication facilities to the satellite. It gets all "data" to be sent from a small file, and saves received data on another. A user batch job can be run on the 360 to update the two files and to deal with the received data.

Version "two" consists of a 4K PL/1 callable subroutine which may be used to communicate directly with the satellite. This version provides to the user's 360 programs the analogous facility that the PDP-8 system provides his PDP-8 programs; ie. he may read or write. Thus the user can maintain direct communication between two processes; one in the satellite, and the other in the central computer. When using version "two", the 360 user may access the "off-line" files used by version "one"; making alternate use of the two versions quite profitable. When ROLLIN/ROLLOUT is implemented on the 360, a facility will be developed whereby the PDP-8 user may choose not only which version is active in the 360, but also which of a library of 360 user programs he is communicating with.

## PDP-8 RESOURCE ALLOCATION

The 4K of PDP-8 core is allocated in the following manner (Fig. 3.):

The system occupies locations 6000-7577, 0000-0007, and uses 0140-0177 for linkage between it and the user(s).

The fetal monitor (which may be considered the foreground) may occupy locations 0200-3777 and 0010-0137. It is given control by the system at one of a number of different entry points depending on what type of interrupt has required action on its part (ie.: clock or sense lines).

The OBFILE (considered the background) may occupy locations 4000-5777. One set of three pages is used to handle each remote teletype currently active. The three pages are used in the following manner: Page one contains the appropriate patient record as it is read from the subset file; page two is occupied by consecutive segments of an overlay structure which handle the record for the teletype (uncoding, formatting, etc.); page three is used for buffers and for communication between the overlays and with the system.

The background has control of the processor until an interrupt occurs which requires activation of the foreground. Control then passes to the foreground at the appropriate entry point. It handles the interrupt, updates its data and/or display(s), and returns control (via the system) to the interrupted background.

*as seen by the PDP-8

**That is, off-line to a user's 360 program.

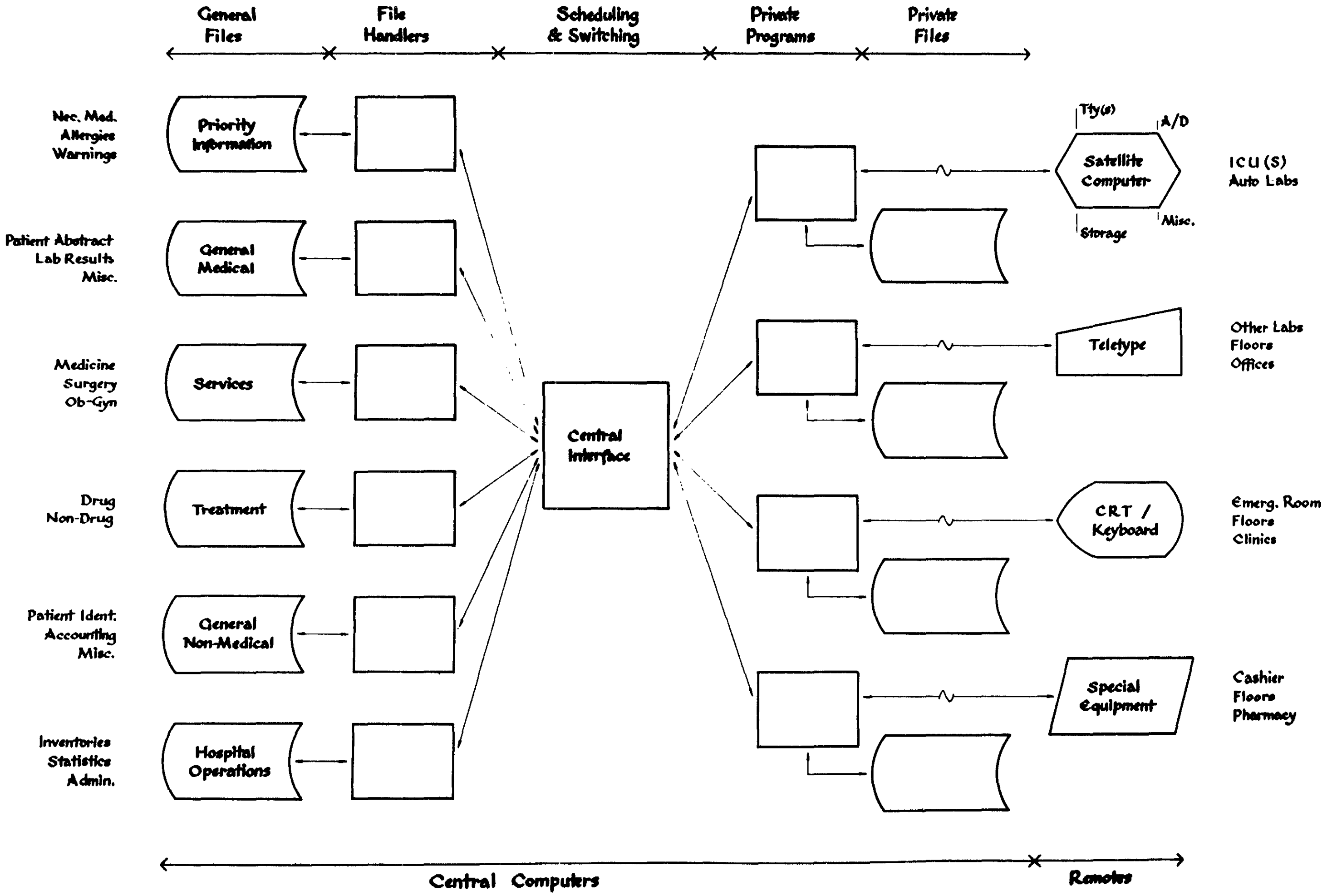
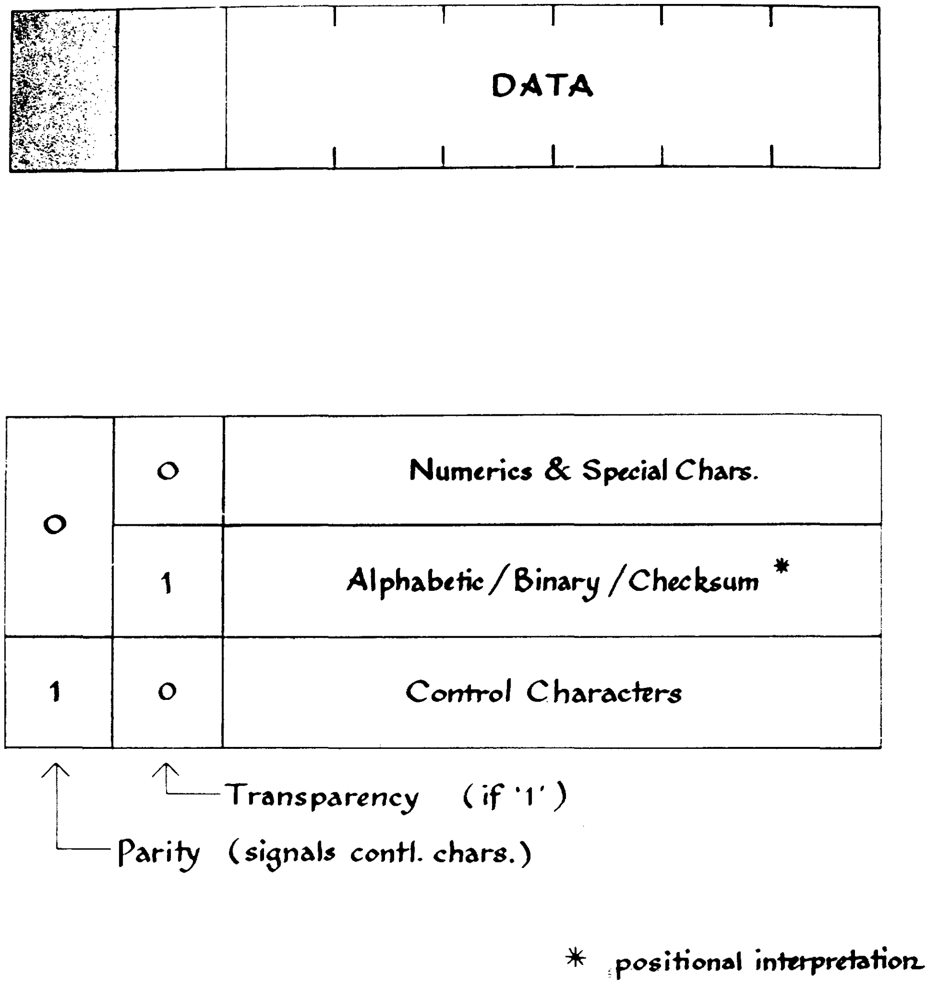


Figure 1. Hypothetic Hospital System: general flow diagram.



| | | |
|---|---|---|
| 0 | 0 | Numerics & Special Chars. |
| | 1 | Alphabetic / Binary / Checksum * |
| 1 | 0 | Control Characters |

Transparency ( if '1' )

Parity (signals contl. chars.)

* positional interpretation

Figure 2. Transmitted byte PTO-8F 2702.

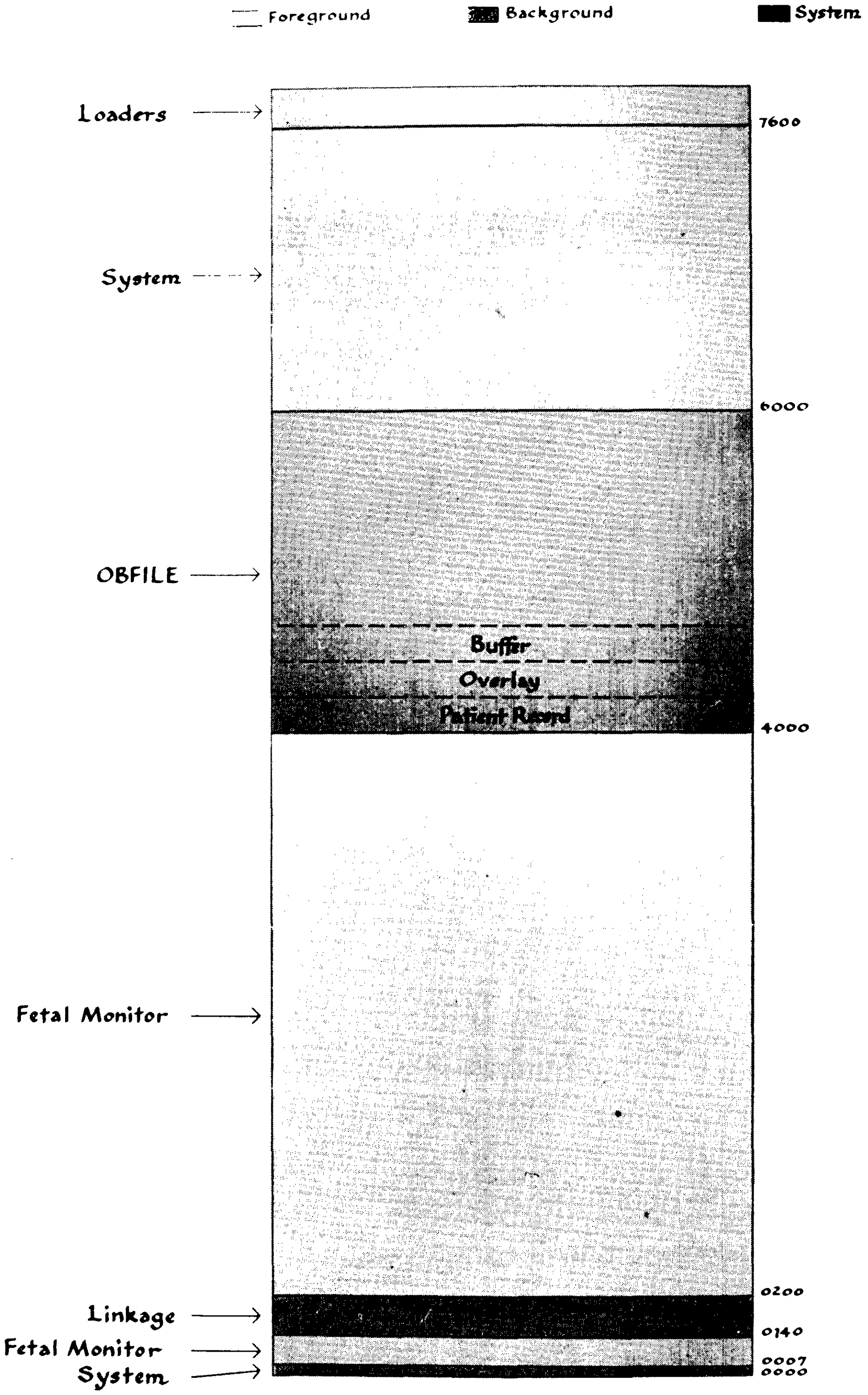


Figure 3. PDP-8 core allocation

CONTROL CHARACTERS

| PDP-8 | 360 | NAME | EXAMPLE | USAGE |
|---|---|---|---|---|
| 200 | 01 | CKSM | ALL | PRECEDES CHECKSUM AS THIRD FROM LAST CHARACTER IN ALL TRANSMISSIONS. |
| 201 | 81 | GETR | E | SENT BY PDP-8 TO REQUEST A 360 "DATA" TRANSMISSION (A'). |
| 202 | 41 | HELO | C,C' | SENT BY EITHER ON SYSTEM RESTART, ECHO'D IF RECEIVED BY 360 WHILE RUNNING. |
| 203 | C1 | RDER | G,G' | SENT BY EITHER ON RECEIPT OF AN IMPROPER TRANSMISSION, (IE: CHECKSUM, ERROR, UNKNOWN CONTROL CHAR, ETC). |
| 206 | 61 | GDBY | D,D' | SENT BY PDP-8 (D) TO CAUSE TERMINATION OF 360 PROGRAM. SENT BY 370 (D') UPON OPERATOR REQUEST FOR EXIT (NOTE: 360 CONTINUES NORMAL OPERATION UNTIL RECEIPT OF (D) ). |
| 207 | E1 | RDOK | H' | SENT BY 360 IN ACKNOWLEDGEMENT OF PROPER RECEIPT OF "DATA" (A) FROM PDP-8. |
| 210 | 11 | SEND | B,B' | SENT BY EITHER AS FIRST CHARACTER IN TRANSMISSION TO THE OTHER'S CONSOLE. TYPEWRITER. |
| 211 | 91 | GETT | F | SENT BY PDP-8 TO REQUEST TIME AND DATE TO BE TRANSMITTED BY THE 360 (I'). |

Table 1. Control characters

ALLOWABLE TRANSMISSIONS

360 to PDP-8

| A' | B' | C' | D' | H' | G' | I' |
|---|---|---|---|---|---|---|
| . | SEND | HELO | GDBY | RDOK | RDER | H |
| .<br>. | .<br>. | CKSM | CKSM | CKSM | CKSM | H |
| (DATA) | .<br>(TEXT) | 1XX | 1XX | 1XX | 1XX | M |
| .<br>. | .<br>. | 1YY | 1YY | 1YY | 1YY | M |
| .<br>CKSM | .<br>CKSM | XOF | XOF | XOF | XOF | S |
| 1XX | 1XX | | | | | S |
| 1YY | 1YY | | | | | Y |
| XOF | XOF | | | | | Y |
| | | | | | | D |
| | | | | | | D |
| | | | | | | D |
| | | | | | | CKSM |
| | | | | | | 1XX |
| | | | | | | 1YY |
| | | | | | | XOF |

Table 2. Allowable transmission records; 360 to PDP-8

ALLOWABLE TRANSMISSIONS

PDP-8 to 360

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|
| | SEND | HELO | GDBY | GETR | GETT | RDER |
| . | . | CKSM | CKSM | CKSM | CKSM | CKSM |
| .<br>. | .<br>. | 1XX | 1XX | 1XX | 1XX | 1XX |
| (DATA)<br>. | (TEXT)<br>. | 1YY | 1YY | 1YY | 1YY | 1YY |
| .<br>. | .<br>. | XOF | XOF | XOF | XOF | XOF |
| CKSM | CKSM | | | | | |
| 1XX | 1XX | | | | | |
| 1YY | 1YY | | | | | |
| XOF | XOF | | | | | |

Table 3. Allowable transmission records; PDP-8 to 360

# LIFE WITH A LABORATORY COMPUTER SYSTEM*

Irwin R. Etter
The Mason Clinic
Seattle, Washington

## ABSTRACT

The Laboratory of the Mason Clinic and Virginia Mason Hospital has used a totally dedicated computer system for the past two and a half years. During that time the laboratory staff has become highly dependent on the functioning of the computer. Despite great increases in work load in the laboratory, the size of the staff has been held constant with a decrease in direct line personnel. The use of the computer allows the staff to pay greater attention to the technical aspects of the laboratory while the computer handles an ever growing portion of the clerical chores. The role of the computer is continually being modified as our experience increases. These changes are due to both the technical changes in the laboratory and to revision of our concepts of the role of the computer. The success of our program is due to the high reliability of the computer system as well as the widespread interest in data processing among the staff.

The laboratory without a computer system is completely reliant upon the technologists for accurate and reliable information. At current work loads, the technologists are under significant pressure to turn out not only high quality work, but high quantity as well. Under such conditions, a technologists' life is not a happy one. In a typical laboratory, a technologist must care for the instrumentation, performing preventive equipment maintenance and making at least minor repairs to keep the laboratory operating. Assuming the equipment is operational she must also insure that the results produced are reliable and accurate. In addition the technologist must document the laboratory work performed both for the physician and for medical records besides maintaining a laboratory file of work performed.

On a typical day the technologist must assemble and in some cases prepare the reagent required, start the instruments, sort samples to be processed, perform analyses and associated calculations, compile work sheets, maintain the instruments, and file reports. Nearly all these procedures can be made less time consuming without loss of accuracy or reliability by computerizing them. Approximately four years ago the laboratory staff began learning the basic facts about computers. At that time the small computers at reasonably low prices were just beginning to be made commercially available. At the same time, interest in laboratory automation was growing among computer manufacturers as a marke area for their new equipment. This required learning most of the procedures followed by the laboratory. As each side learned about the other, mutual interest grew. The laboratory had a requirement that could well be handled by the small digital computer. Laboratories represented a significant market area for small, highspeed, low priced computers.

Out of this mutual interest several laboratories and computers were "married". The laboratory of the Mason Clinic was "married" with the computer in April 1966. The system was composed of a PDP-8 with 4-K of core. A special purpose interface composed mainly of DEC logic modules was built to connect the computer to the laboratory instruments. A complete set of software was also provided which would handle the basic processing of the raw data. The tasks of the initial system were limited to data acquisition and reduction in order to obtain a system that would provide useful data while at the same time allowing for continual reassessment laboratory requirements and allowing the laboratory personnel to adjust to the computer.

The system functioned in the initial configuration for one and one half years. During that time, extensive evaluations of both hardware and software were conducted. Based on the findings it was decided that the hardware configuration was sound and should be expanded to handle a larger work load. The software was adequate but major revisions were made to improve system operation. Personnel training began during this period and continues as additional people become involved with the system. Initial response to the computer was very good and ranged from immediate acceptance to minor reluctance to accept the computer results. As improvement in the software were made, resulting in greater reliability all resistance to the computer disappeared.

One of the major difficulties encountered in using the system was that of monitoring the operation without interrupting the on-line real time procedures. This was especially important during the debugging phases of new programs. Some of this monitoring could be done with the software supplied but it became necessary to be able to check any core location and if necessary change the contents of the location. A routine was written to allow this to be done on-line which has resulted in a greater of the functioning of the system. Another type of difficulty with the system was the determination of sources of error or malfunction that occurred with very low frequency. In most cases, the computer very quickly hides all traces of its work making it very difficult to determine a pattern for the malfunction. Tracking down such problems is very much like playing detective, hunting for a clever criminal. One such problem was an occasional sugar determination that the computer would read significantly higher than the actual result. Several software traps were devised to force duplicate calculation or other checks but none solved the problem. Part of the difficulty in finding the error lay in retrieving the scratch areas of core used for calculation as these would be overwritten as soon as another calculation was performed. After several weeks of closely monitoring the system, the error was caught while all evidence was still available and the problem was solved. Part of the difficulty in solving these problems is that once a correction is made there is no sure way to know the problem is solved except for its lack of occurrence over a long period of time.

The interaction of the laboratory staff with the computer has been interesting. The dislike for manual calculation methods becomes obvious as the dependency upon the computer grows. It has become difficult to work on the computer during normal laboratory hours. The technologists raise very strong objections to any interference with the normal computer operation. Significant time savings have resulted from use of the computer. As a result, we have been able to reduce the size of the automated laboratory staff by 20%. This decrease is possible despite an increase in work load each year of 10%-20%. Further increases in work load will be offset by increasing the work load of the computer.

The laboratory staff has been a great help in the success of the computer program. Periodic training courses have been run to acquaint them with the workings of the computer system and in the basics of programming. Several staff members have begun writing programs to increase their understanding of computers. They have also been very helpful, suggesting modifications to the system to make their jobs easier. Many of these suggestions have been successfully implemented.

The state of the system is continually undergoing change. An additional 4-K of core and the extended arithmetic option have been added to the system. These additions have enabled us to improve the methods of calculation and typeout separating to give more reliable data in a more usable form. We have recently added both disc and tape bulk storage to allow for the development of patient laboratory records. This will greatly decrease the clerical work load of the laboratory with the elimination of a major source of error, the hand transferring of data. The bulk storage will also make it possible to maintain a large library of routines for the processing of data from the manual test procedures.

The design philosophy of the system is open ended, allowing for additions and modifications as laboratory procedures change. It is anticipated that in the next few years new laboratory instrumentation will be developed that will be directly computer compatible. Much of the present instrumentation requires major engineering to make the output signals available to the computer. The widespread acceptance of computers in hospital laboratories will force instrument designers to develop compatible equipment. Similarly, a new breed of computers will be forthcoming that will be designed primarily of laboratory usage will include flexible interfaces to enable instruments to be rapidly "married" to it.

## Acknowledgments

This work is being supported by Public Health Service Grant No. 5 R01 GM 15581.

# DIAGNOSTIC USES OF THE AVERAGED EVOKED POTENTIAL IN CLINICAL NEUROPSYCHIATRY

Enoch Callaway, M.D.
Langley Porter Neuropsychiatric Institute
University of California School of Medicine
San Francisco, California

## ABSTRACT

Over the past four or five years, small high-speed digital computers have been used to process human brain waves in a variety of ways that have potential clinical value. The usual procedure has been to digitize a set of potentials from the head and treat these by a variety of techniques, the most popular of which is averaging sequences of potentials with each sequence having the same time relation to some recurrent event. These averages, frequently referred to as Averaged Evoked Potentials or AEPs, have been put to a variety of uses. The purpose of this paper is to review some of these uses.

AEPs can be used to test the various input channels to the central nervous system such as hearing, vision, touch, smell, and acceleration. They undergo changes with growth and development and hence can be used as measures of central nervous system maturation. They appear much more sensitive to destructive lesions in the central nervous system than the conventional EEG, although they may not be as sensitive or useful as a conventional EEG in detecting irritative lesions such as appear in epilepsy.

In the borderland of application, we find these computer-derived brain wave measures being related to such things as intelligence, motivation, perceptual style, and psychiatric diagnosis. In spite of the tremendous promise of this area, only one of these techniques has reached the status of a routine clinical procedure. That is the use of the averaged evoked potential in performing audiometry in children and infants. There is much to be done in closing the gap between laboratory demonstrations and routine clinical use.

When, for any reason, normal communication is poor, then the unique qualities of evoked responses can be used to advantage in evaluating psychological function. To get an evoked response, various stimuli are given, various brain wave samples are taken, and various computations are made using these brain wave samples. By properly designing the test situation and the data analysis, things such as attention deployment, distractibility, and certain aspects of intelligence can be probed--all without demanding much in the way of voluntary responses from the subject. No matter whether the subject is uncooperative because of neurological defects, cultural differences, or willfulness, if he will sit relatively still, his mind can be probed.

A variety of more or less complex computational schemes have been devised for studying evoked responses, but simple averaging has yielded so many interesting findings that we can profitably restrict this review to the so-called averaged evoked potential or AEP.

Of all the AEP techniques that hold promise for clinical application, only one at present has the status of a routine procedure. That one is AEP audiometry in neonates. In neonates, vocal ano other motor functions have not developed to a very useful point so that electrical activity from the head has much to recommend it as an output device for the computer inside the infant's skull. In most cases, however, language is the output device of choice for humans.

To be applied in the clinic, a technique must be useful in addition to being feasible. Here I will discuss technical feasibility, but I want to emphasize that these technical possibilities are not useful in practice unless, for some reason, there is something the matter with simply asking the subject what we want to know.

There are roughly three classes of situations when just asking doesn't work, so that evoked response procedures hold real promise. These are: (1) when neurological factors such as lesions or lack of maturity block verbal exchange; (2) when psychological factors such as cultural difference or mendacity interfere; (3) when the state we're testing for is not accessible to introspective report and its behavioral consequences are most inexpensively tapped by our evoked potential measure.

## I. SENSORY TESTING

### A. Hearing

The auditory evoked potential test for hearing loss[14,36,15] is a straightforward procedure. A tone, a tone-pip, or click, is presented and this presentation is repeateda number of times. Using vertex to ear leads, the averaged evoked potential is computed, and if an AEP is distinguishable from the background activity, the subject is inferred to have heard the sound. This is close enough to the truth to be quite useful. Using this method, estimates of auditory threshold approximate those obtained by conventional techniques in cooperative subjects.

Averaged evoked potentials can be o tained with subthreshold stimuli[28], deaf subjects may have AEPs because of a myogenic reflex[11] to the vestibular effects of loud sound, and the absence of a sound may even evoke a response[3]. Nonetheless, good audiometry can be done using AEPs. Some of the above points do, of course, have practical consequences. For example, an AEP evoked by a very loud sound does not rule out total hearing loss if vestibular function remains intact, although the myographic responses usually are earlier than the actual auditory evoked potentials. Perhaps of more clinical importance are those cases where gross brain damage prevents the appearance of the AEP even though hearing may be relatively intact.

Marked moment-to-moment variability in AEP amplitude occurs without apparent cause. In addition, background EEG and state of arousal are confounded and both probably also affect the AEPs. Although these must be considered in evaluating a record, good AEPs can be obtained during sleep. Although slow wave sleep raises the auditory threshold about 10 db in adults, it has no measurable effect on the thresholds of infants. This is lucky because it is hard to find a time when a small infant is quiet except when he is asleep.

### B. Vision

The visual evoked response is potentially useful for testing the presence or absence of vision. In addition, both color and form resolution can also be assessed. Finally, since the averaged electroretinogram (ERG) and the averaged occipital EEG (AEP) can both be computed, lesions along the optic pathways can be localized.

Different colors apparently induce different AEPs in individuals with color vision, and this differential responsiveness is absent in the color-blind. Several methods of obtaining color AEPs have been suggested[42,45,38] but the one described by Clynes[10] is probably the most feasible for clinical use. Use is made of a patch that alternates between two test colors at about 24 cycles per second. If the colors are matched for brightness, these alternations will evoke a response in normals, but will not do so in subjects blind to the particular color change. This procedure has certain obvious advantages for, at 24 cycles per second, epochs of two cycles in length can be averaged a number of times in a very short period. By having two or more cycles in an average, an estimate of the reliability of the AEP can be obtained by simply looking at the record. However, it should be noted that Shipley et al.[46] found the most dramatic effects of color in 4 to 8 cps fourier components, and so slower stimulus repetition rates may have some advantages that still need evaluation.

Clynes et al., also records from four bipolar pairs derived from a rosette of electrodes at the occiput. This arrangement demonstrates color evoked potentials in some leads and not in others. It would appear from his data that the use of multiple leads increases the precision with which a response can be detected and hence, increases the confidence in identifying cases where the response is absent. The more recent work of Regan[39] also indicates some peculiar interactions between color and stimulus frequency for maximum response amplitude.

Pattern also influences the AEP and here it seems clear that lateral bipolar leads, as for example $O_z$ - $O_1$ may be more sensitive than an AP arrangement at right angles as, for example, $O_z$ - $C_z$[9,40]. In general, the amplitude of the AEP and particularly the 200 msc component increases with the number of contrasting borders resolved at the fovea. If the number of contrasting borders is increased stepwise and the AEP is recorded at each step, then at the point when resolution is lost, the AEP is found to drop abruptly.

The fovea is almost entirely responsible for the visual AEP[16]. Thus in infants with severe retinal disease with macular sparing, the retinogram may be absent, while the visual AEP may be almost normal[53].

There are, naturally, pitfalls. For example, the effect of pattern can be obliterated by using very bright flashes. The results depend on the subject fixing and focusing on the target, so the procedure is not applicable to very uncooperative or sleepy subjects. However, Carol White, in a personal communication, has suggested that the use of brief flashes can allow clinical refraction without cycloplegia because the flash can be brief enough to supply the stimulus before pupilary constriction can compensate for refractive errors.

Lesions in the optic pathway distal to the optic nerve will reduce both the ERG and the AEP. More central lesions, however, will leave the ERG intact and abolish or diminish the AEP. Asymmetry of the left and right occipital AEP with stimulation of one eye occur as would be predicted from the anatomy of the optic nerve[27,5,20].

### C. Somatosensory

In somatosensory evoked responses to electrical stimulation of the nerve, the peripheral nerve action potential can substitute for the ERG in the visual analogue[41]. There are, however, some additional interesting complexities. The somatosensory AEP to electrical stimulation of the nerve seems to reflect primarily the activity of the dorsal white columns, since total lesions of the anterio-lateral columns can leave the AEP quite normal. When lesions involve a peripheral nerve, latencies of almost all AEP peaks are lengthened. Wave durations are increased and the whole AEP may be prolonged. This is because an afferent peripheral lesion reduces excitability and results in a temporal dispersion of the afferent volley.

Bergamini and Bergamasco[5], in their discussion of the above points, give an interesting clinical application of the somatosensory evoked potential, although one that is not likely to present itself every day in clinical practice. They encountered a pair of 6 year old siamese twins who were joined through a partial dorsal fusion of the lumbar sacral spinal column. Before surgical separation it was important to determine whether or not any nerves or roots were fused. Clinical evaluation was unsatisfactory, but somatosensory evoked potentials saved the day. Evoked potentials could only be recorded from the scalp of the twin receiving the electrical stimulus. With this evidence that the twins had no spinal nerves in common, surgical separation was undertaken and successfully completed.

D. Other Sensory Pathways

There is no reason that other sensory modalities could not be assessed using AEPs. Allison and Goff[2] for example, have described an olfactory AEP which might be applied to testing smell. Greiner et al.[23], have shown that vestibular evoked responses can be obtained by rocking the individual. This is an interesting AEP for it is of long duration and almost completely unilateral (tempero-occipital), being on the right side in right-handers and on the left in left-handers.

Evoked potentials can be produced by using touch or a puff of air as a stimulus. Although there is still some doubt on the matter, the consensus is that hysteria does not abolish the evoked potential. Therefore, the differentiation between organic and hysterical sensory loss could be made by using the AEP. However, touch evoked potentials can, unlike shock evoked potentials, be almost totally abolished by distraction[19]. This is not surprising in view of the rapid accommodation of touch and the contrasting stability of vibration and position sense over long periods of stimulation.

## II. NEUROLOGICAL DISEASE

As the search for pathology leads us rostral along the neuraxis, we come to consider neurological diseases of the cerebrum. For our purposes, these may be divided into disorders of maturation, destructive lesions, and irritative lesions.

A. Disorders of Maturation

Evoked potentials showed dramatic changes with maturation. The auditory evoked potential is easily obtained in infants as noted above, but it does not show the marked changes with maturation that are found in the visual evoked potential[18]. Recording from inion and presenting flashes to a sleeping infant, a primary visual evoked response can be recorded. The latency of the onset of this component is linearly related to conceptual age and ranges from about 200 msec at 35 weeks gestation to 400 msec at 45 weeks. Engel used the onset rather than the more conventional wave peak as the point for measuring latency since he found that blink artifacts tend to obscure the peak. This recommends itself as a pediatric technique for distinguishing full term "runts" from premature infants.

Maturational changes continue in a fairly dramatic fashion up to and perhaps past puberty, to be followed by evidences of aging, such as increased amplitude and length of latency commencing at about age 40[17,51].

The interval between infancy and puberty holds great promise but is still being explored. Eventually, the AEP should provide a handy measure of central nervous system developmental age that could be used for comparison with chronological age, skeletal age, and so on. This should be of particular interest to those doing cross-cultural studies. Arakawa et al.[1], found children about age 9 with ariboflavinosis showed prolonged AEP latencies to light. This needs to be confirmed and compared with developmental changes. Does a child with ariboflavinosis simply have a less mature AEP, or does he have a distinctly different AEP? Actually, the reported effects of ariboflavinosis on the AEP are much less obvious than effects on power spectra of background EEG. However, these preliminary reports are enough to merit further study.

B. Destructive Lesions

Destructive intercranial lesions make themselves known by marked asymmetry in the AEP. Later, we will indicate how AEP asymmetry may be a correlate of intelligence. This, however, relates to the 150 msec parietal AEP to flashes. The asymmetry produced by cortical lesion is non-specific, gross, and total. For example[21], in a patient with a rather small, superficial parietal lesion on the dominant hemisphere, nerve shock contralateral to the lesion produced little or no response on the affected side and an abnormal response on the unaffected side. Nerve shock ipsilateral to the lesion evoked a normal response on the unaffected side and a slightly reduced but otherwise normal late response on the diseased side. This is of some theoretical interest since it indicates that the late somatosensory AEP apparently requires a functioning lemniscal pathway for its normal development. The AEP may also be grossly asymmetrical when the raw EEG is symmetrical, thus, the AEP gives some gain over conventional electroencephalography[5].

A prognostic test for brain damage due to cerebral circulatory insufficiency has been proposed by Crighel et al.[13]. They observed that normals show an increase in visual AEP amplitude and recovery after breathing $O_2$. Hyperoxia also increases AEP responsiveness in patients with circulatory insufficiency when the prognosis for return of function is good but may actually have a reverse effect and further diminish AEPs in cases that subsequently show no recovery.

C. Irritative Lesions

In epilepsy, the AEP may not even offer as much as does the conventional EEG. Amplitudes may be increased during spike and wave discharge, and the late "ringing" of the visual AEP may be more marked during interseizure periods[5,31]. In general, however, the raw EEG serves as well as or better than the AEP. Morrell[32], however, has reported the interesting case of a patient who had an epileptogenic lesion in the auditory cortex. Recording from over the area of the lesion ordinarily revealed little or no response to flash. However, by a prior repetitive pairing of flash and click, the patient could be "conditioned" so his auditory cortex would give an abnormally large visual evoked response. The generality of such a phenomenon, however, remains to be explored.

## III. INTELLIGENCE

The first study relating intelligence to averaged evoked potentials was reported by Chalke and Ertl[8].

They used a population with widely dispersed I.Q.s and found shorter latencies in the brightest subjects. This has been confirmed in essence by Plum[35], although the correlations between latency and I.Q. in the study of a more homogeneous group of subjects was not impressive.

The most recent work on intelligence is that of Beck et al.[4]. They recorded flash evoked responses from parietal leads and found that the negative going wave at about 150 msec is larger on the right than on the left in the bright subjects, but almost identical on the two sides of the dull subjects. The larger right-sided potentials in the bright children were also more stable than those of the dull children.

Unpublished studies from a number of laboratories (American Psychological Association meeting, 1968 Symposium: Brain Function, Cognitive Performance, and the Developing Child, San Francisco) suggest that various AEP measures may be related to both age and adequacy of performance in children under the age of 9 years. These same measures also seem related to performance in children when age is held constant.

In summary, good (older) performance goes with high amplitude, assymetry, stability (i.e. low single sample variability) and long latency. This last is notable since it seems opposite to the finding in adults noted above.

## IV. PSYCHIATRIC DIAGNOSIS

The sensitivity of the AEP to subtle psychological and physiological influences makes this potential application very intriguing. Certainly for the psychiatrist, electroencephalography has been more impressive in promise than in pay off. Although work such as that by Stevens et al.[50] and L. Goldstein, et al.[22], would suggest that all the potentials of the standard EEG have not been exhausted, the AEP has a great appeal.

### A. Recovery Cycles

One of the first applications of averaged evoked potentials in clinical psychiatry was made by Shagass and his group[43]. They examined the recovery cycle of the early components of the somatosensory AEP using carefully place bipolar leads over the appropriate sensory field. They gave pairs of shocks to the ulna nerve. The first, or conditioning shock, was presumed to produce a phasic inhibition and recovery. This recovery cycle was then probed by the second, or test, shock. By varying the time interval between the conditioning and test shock, the time course of the recovery cycle could be plotted.

Psychiatric patients were found to have slower and less adequate recovery than normals. As studies progressed, however, it appeared that the situation was more complex. Some of the patients studied had abnormally large initial responses so that a delayed recovery might be relative (i.e. a small per cent of a large conditioning response) rather than absolute. Older patients were found to have larger responses, and females were found to have both larger responses and shorter latencies. This problem was tackled and solved by means of statistical procedures which removed the effect of the conditioning response amplitude. In spite of this, the reduced recovery cycle was still observed.

This appears to be a rather general phenomenon. For example, they counted 10 peaks in the first 80 msec. of the somatosensory averaged evoked potential, and found the recovery cycle effect to a greater or lesser degree in all of these peaks. Furthermore, the phenomenon was found in almost all psychiatric conditions. In fact, the only psychiatric patients who did not differ from normals were a group composed of psychoneurotics with anxiety and depression and a group suffering from psychophysiological reactions.

These studies were done with fairly intense stimuli More recent studies using milder stimuli have shown that some subjects who show a delayed recovery with high intensity stimuli may show a supernormal recovery with low intensity stimuli.

The recovery cycle phenomenon seems to be a real one, for other laboratories have confirmed it, using visual evoked potentials[48,24]. However, since depressed recovery to intense stimuli can be observed in such a variety of conditions, it would seem premature to attempt any clinical use. However, as the complexities of the recovery cycle are unraveled and the underlying neurophysiology is clarified, this technique may offer promise for clinical use.

Inter-modal recovery cycle effects have been described by Goff[21]. These effects are highly subject-dependent and can be noted with interstimulus intervals as great as one second. This intriguing effect has yet to be investigated for possible clinical significance.

### B. Contingent Negative Variation

If the active electrode is place somewhere over the front of the head, a slow negative-going wave can be observed during the period that the subject anticipates a stimulus. This slow wave has been called he contingent negative variation (CNV). Because of ts slow time course, it must be recorded with direct coupled amplifiers and relatively nonpolarizeable electrodes. In the usual experimental situation, a warning stimulus is presented, and then, after a predetermined interval of, say, two seconds, a second stimulus is presented to which the subject must make some response. The CNV goes to a maximum just before the final or "imperative" stimulus, then resets to normal.

This phenomenon was discovered by Grey Walter's group[54,12]. They observed that in reaction time experiments, the amplitude of the CNV is related to the speed of the response. It seemed to them as though this reflected a kind of expectancy--the greater the expectancy, the higher the negative wave, and the faster the reaction time.

The CNV can be ovserved when the second stimulus does not require any gross motor response as for example in viewing the presentation of a picture. However, the amplitude of the CNV has been shown to be related to the amount of energy required by the motor response if motor response is, indeed, required[37]. If the second stimulus is occasionally omitted, then, presumably, the subject's expectancy is reduced and the negative contingent wave is also reduced.

Grey Walter has suggested that some level of intelligence is necessary for the development of the contingent negative wave so that this can be used as an estimate of intelligence. It is reduced in anxiety and in schizophrenia, is increased in obsessive neuroses, and is absent entirely in psychopaths. Thus, it would appear that the contingent negative wave might be the basis for an entire psychodiagnostic inventory[55].

There are, however, some problems with this. DC recordings in an uncooperative subject can be difficult because lead sway may cause blocking of the amplifier. Eye movements also are difficult to exclude as a factor in slow wave changes, particularly with frontally placed electrodes. Naatanen[33] has presented evidence that phasic arousal of any sort causes a negative DC shift and that this is the cause of enhanced AEP amplitudes with attention.

An important series of objections is raised by Vaughan[52]. He finds that the contingent negative wave actually occurs over motor cortex and parallels an inhibition of motor activity that sets the stage for a sudden release of a motor response. Vaughan suggested that most of Grey Walter's findings could have been predicted by looking at the distributions of the reaction times. For example, the relationship between reaction time and the CNV reflects the fact that when reaction times are slower, their distribution is more spread out. In such a case, if peak CNV coincided with response, peak CNVs would not be well time-locked to the stimulus when reaction times were slow, and the average CNV would be smaller than in a more closely time-locked average associated with fast reaction times. To demonstrate this, he averaged backwards from the motor response in situations of fast and slow reaction times and showed that the CNV was not different in the two conditions, when one thus time-locked the averaging to the response rather than to the stimulus.

The CNV is also accused of being a motor inhibitory potential because the amplitude of the H-reflex (the monosynaptic reflex observed by stimulating the nerve in the popliteal fossa and recording from the gastrocnemius), is reduced in proportion to the amplitude of the CNV. Obviously, the CNV supports considerable controversy. Nevertheless, the potential of the CNV for psychodiagnostic procedures should not be discounted at this time.

C. Stimulus Intensity Control

Buchsbaum and Silverman[6] introduced the averaged evoked potential amplitude in the investigation of a phenomenon that they call stimulus intensity control. Based on the work of Petrie[34] and of Silverman[47] they assume that some people are able to reduce their responsiveness to strong stimulation. Petrie had used a kinesthetic figural after effects measure to assess this tendency. She found that after interposed tactile stimulation, some subjects would judge the width of a bar to be narrower than previously. Such people who did this were called "reducers" and she found such people to be less sensitive to pain. Silverman had also noticed that nonparanoid schizophrenics tended to be reducers. Buchsbaum and Silverman then reasoned that the characteristic of these people was an ability to reduce their responsiveness to strong stimulation, and if this was the case, they should show a reduced AEP to intense light stimulation.

They used a mastoid to vertex derivation and varied the flash of a light provided by a Grass Photostimulator. They found that the subjects classified as reducers on the kinesthetic figural after effects measure showed less increase in the amplitude of evoked potentials as the intensity of the stimulus was raised (and some showed a paradoxical decrease!).

This correlation between "reducing" as determined by visual AEP and as determined by kinesthetic judgment has been confirmed using 10 Hz sine wave light as a stimulus[49]. Thus, the phenomenon seems to be a genuine cross-modality perceptual style. The relevance of this for clinical work (e.g., pain sensitivity, schizophrenia, etc.) remains to be determined. However, preliminary data from our laboratory indicates that chronic LSD users tend to be "reducers" while feeble minded children tended to be "augmenters."

D. Auditory AEP Variability and Schizophrenia

The thought disorder of schizophrenia is characterized by variability of behavioral responses. It appears that an increase in auditory AEP variability parallels this behavioral variability.

Some years ago we designed an AEP procedure to study schizophrenic thought disorder. In this procedure, tones of 600 and 1,000 Hz are repeated in a haphazard order, and the subject is told to ignore the tones. The subject, in a quiet, semidark room, watches his brain wave ($C_z$ - $A_1$) on an oscilloscope monitor. After demonstrating the effects of tensing muscles, rolling eyes, and so on, he is asked to maintain a stead EEG. Under such conditions the tones seem trivial and, in nonschizophrenics, the high (1,000 cycle) tone averaged evoked potential and the low (600 cycle) tone averaged evoked potential will be almost identical. In other words, the two physically different tones evoke almost identical responses when no particular psychological distinction is being made between them. However, when a normal individual has reason to distinguish between the tones, there is then a difference between the two AEPs.

A variety of models of schizophrenia ranging from Shakow's[44] notion of segmental set to McGhie's[30] notion of over-inclusiveness predicted that schizophrenics would behave as though they had some reason to make a distinction between these tones. In assigning psychological significance to difference between tones, the schizophrenic should also show a differentiation in his AEPs. That is to say, the averaged evoked potentials to the two different tones should be more dissimilar in the schizophrenic.

In the situation described above, schizophrenics tend to have more dissimilar AEPs than do nonschizophrenic psychiatric patients. And among schizophrenics, disturbed nonparanoid patients have the most dissimilar AEPs[7,25,26].

There are, of course, a number of possible explanations for such an observation. One possibility is the effect of drugs. In our hospital, we have no good facility for keeping schizophrenics drug-free for the several months that is probably required. However, we are able to follow patients as their clinical course fluctuates and have been able to show that two tone averaged evoked response test results vary in parallel with the thought disorder even while drug dose remains relatively constant. Furthermore, normal values are obtained from neurotic patients and patients with affective psychoses even though such patients may also be on fairly large doses of phenothiazines.

Our findings could also result from the use of correlation coefficients as a measure of similarity between the two AEPs. Such a measure cannot be interpreted like ordinary correlation coefficients since the points along an AEP are not serially independent. Nevertheless, a high correlation between two averaged evoked potentials indicates that the two curves are very similar. Two dissimilar curves (yielding low correlations), could result from consistent distinction being made between individual evoked potentials. On the other hand, low correlations could also be produced by a low signal-noise ratio (e.g., a large amount of background activity, considerable irregularity in evoked potentials or low voltage evoked potentials). This is a serious problem, for schizophrenics tend to have low voltage evoked potentials. Furthermore, we have found both a correlation between amplitude and clinical state and also a correlation between the two-tone evoked potential measure and evoked potential amplitude.

In our most recent study we use the old two-tone procedure and in addition a mock two-tone procedure where both "tones" were 1,000 Hz. The correlation measure distinguished schizophrenics from normals in both the old and mock procedures. When only a single tone was sounding the correlation measure can only be reflecting sample variability.

This matter of AEP variability is made more explicit by computing the standard deviation at each AEP time point. If, for each AEP, the maximum peak to trough amplitude is plotted against the largest one of the standard deviations, then it can be seen that the two are highly correlated. A regression line will however separate schizophrenics (low amplitude, high standard variations) from normals.

An elegant discriminant function measure developed by Donchin was the only one of several methods tried that distinguished the real two-tone from the mock procedure. Yet schizophrenics did not differ from normals significantly on this measure. Although some problems of background EEG remain to be solved it seems that auditory AEP variability accounts for the two-tone correlations sensitivity to schizophrenic thought disorder.

This AEP test is rarely of value clinically. Occasionally it has helped confirm the diagnosis of hysterical pseudo-psychosis. Although patients with depression and with character disorders usually have normal two-tone AEP scores, an abnormal score is not specific to schizophrenia. For example, Korsakoff patients have low scores (low correlations) and two-tone AEP scores correlate with clinical rating of confabulation and confusion in such patients[29].

## CONCLUSION

The AEP has been feasible as a routine procedure for almost 10 years. Except for the diagnosis of deafness in infants, it still remains a research technique. The promise of clinical utility seems brighter each year, however, and the challenge for the researcher to close the gap between laboratory and clinic is becoming harder to ignore.

## BIBLIOGRAPHY

1. Arakawa, T.; Mizuno, T.; Chiba, F.; Saki, K.; Watanabe, S.; Tamura, T.; Tatsumi, S.; and Coursin, D. B.: Frequency Analysis of Electroencephalograms and Latency of Photically Induced Averaged Evoked Responses in Children with Ariboflavinosis. Tohoku J. Exp. Med., vol. 94, 1968, pp. 327-335.

2. Allison, T.; and Goff, W. R.: Human Cerebral Responses to Odorous Stimuli. Electroenceph. Clin. Neurophysiol., vol. 23, 1967, pp. 558-560.

3. Barlow, J. S.; Morrell, L.; and Morrell, F.: On Evoked Responses in Relation to Temporal Conditioning to Paired Stimuli in Man. In Quarterly Progress Report #78, Research Laboratory of Electronics MIT, 1965, pp. 263-272.

4. Beck, E. C. and Dustman, R. E.: Personal communication, 1968.

5. Bergamini, L.; and Bergamasco, B.: Cortical Evoked Potentials in Man. C. C. Thomas (Springfield, Ill.), 1967.

6. Buchsbaum, M.; and Silverman, J.: Stimulus Intensity Control and the Cortical Evoked Response. Psychosom. Med., vol. 30, 1968, pp. 12-22.

7. Callaway, E.; Jones, R. T.; and Layne, R. S.: Evoked Responses and Segmental Set of Schizophrenia. Arch. Gen. Psychiat., vol. 12, 1965, pp. 83-89.

8. Chalke, F. C. R.; and Ertle, J.: Evoked Potentials and Intelligence. Life Sci., vol. 4, 1965, pp. 1319-1322.

9. Clynes, M.; Kohn, M.; and Gradijan, J.: Computer Recognition of the Brain's Visual Perception through Learning the Brain's Physiologic Language. Part 9, I.E.E.E. International Convention Record, 1968, pp. 125-142.

10. Clynes, M.; and Kohn, M.: Spatial Visual Evoked Potentials as Physiological Language Elements for Color and Field Structure. Electroenceph. Clin. Neurophysiol., supp. 26, 1967, pp. 82-86.

11. Cody, D. T.; Jacobson, J. L.; Walker, J. C.; and Bickford, R. G.: Averaged Evoked Myogenic and Cortical Potentials to Sound in Man. Ann. Otol., vol. 73, 1964, pp. 763-777.

12. Cohen, J.; and Walter, W. Grey: The Interaction of Responses in the Brain to Semantic Stimuli. Psychophysiol., vol. 2, 1966, pp. 187-196.

13. Crighel, R.; Poilici, I.; and Marinchescu, C.: The Influence of Hyperoxia on Flash Evoked Potentials in Normals and in Patients with Cerebral Circulatory Insufficiency. Confina Neurologica, vol. 28, 1966, pp. 348-354.

14. Davis, H., ed.: The Young Deaf Child: Identification and Management. Acta Otolaryng. (Stockholm) Suppl. 206, 1965.

15. Davis, H.; et al.: Further Validation of Evoked Response Audiometry (ERA). J. Speech and Hearing Research, vol. 10, 1967, pp. 717-732.

16. DeVoe, R. G.; Ripps, H.; and Vaughan, H. G.,Jr.: Cortical Responses to Stimulation of the Human Fovea. Vision Research, vol. 8, 1968, pp. 135-147.

17. Dustman, R. E.; and Beck, E. C.: Visually Evoked Potentials: Amplitude Changes with Age. Science, vol. 151, 1966, pp. 1013-1015.

18. Engle, R.: EEG Responses to Sound and Light in Premature and Full Term Neonates, J. Lancet, vol. 87, 1967, pp. 181-186.

19. Ervin, F. R.; and Mark, V. H.: Studies of the Human Thalamus IV: Evoked Responses. In Whipple, E.; and Katzman, R.; eds., Sensory Evoked Responses in Man. Ann. N. Y. Acad. Sci., vol. 112, 1964, pp. 81-92.

20. Gevin, R.; Ravaut, P.; David, C.; Munier, F.; and Parmeland, D.: Potentiels Evoques Moyens Occipitaux et Lesions du Nerf Optique. Le Journel de Medicen de Lyon, vol. 47, 1966, pp. 1725-1748.

21. Goff, W. R.: Evoked Potential Correlates of Perceptual Organization in Man. Proceedings of Teddington Conference on Attention in Neurophysiology, Teddington, Middlesex, England, Oct. 3-5, 1967.

22. Goldstein, L.; Sugerman, A. A.; Stolberg, H.; Murphree, H. B.; and Pfeiffer, C. C.: Electro-Cerebral Activity in Schizophrenics and Non-Psychotic Subjects: Quantitative EEG Amplitude Analysis. Electroenceph. Clin. Neurophysiol., vol. 19, 1965, pp. 350-361.

23. Greiner, G. F.; Collard, M.; Conraux, C.; Picart, P.; and Rohmer, F.: Recherche de Potentiels Evoques d'Origine Vestibulare Ches l'Homme. Acta Otolaryng., vol. 63, 1967, pp. 320-329.

24. Heninger, G.; and Speck, L. B.: Visual Evoked Responses and the Mental Status of Schizophrenics. Arch. Gen. Psychiat., vol. 15, 1966, pp. 419-426.

25. Jones, R. T.; Blacker, K. H.; Callaway, E.; and Layne, R. S.: The Auditory Evoked Response as a Diagnostic and Prognostic Measure in Schizophrenia. Amer. J. Psychiat., vol. 122, 1965, pp. 33-41.

26. Jones, R. T.; Blacker, K. H.; and Callaway, E.: Perceptual Dysfunction in Schizophrenia: Clinical and Auditory Evoked Response Findings. Amer. J. Psychiat., vol. 123, 1966, 639-645.

27. Kooi, K. A.; Govener, A. M.; and Bagchi, B. K.: Visual Evoked Responses in Lesions of the Higher Optic Pathways. Neurology, vol. 15, Sept. 1965, pp. 841-854.

28. Libet, B., et al.: Responses of Human Somatosensory Cortex to Stimuli Below Threshold for Conscious Stimulation. Science, vol. 158, Dec. 22, 1967, pp. 1597-1599.

29. Malerstein, A. J.; and Callaway, E.: Two-Tone Average Evoked Response in Korsakoff Patients. J. Psychiat. Res., in press, 1968.

30. McGhie, A.: Psychological Studies of Schizophrenia. Brit. J. Med. Psychol., vol. 39, 1966, pp. 281-288.

31. Mirsky, A. F.; and Tecce, J. J.: The Analysis of Visual Evoked Potentials During Spike and Wave EEG Activity. Epilepsia, in press.

32. Morrell, F.: Clinical Neurology: Some Applications of Scanning by Computer. Calif. Med., vol. 103, 1965, pp. 406-416.

33. Naatanen, R.: Selective Attention and Evoked Potentials. Annales Academiae Scientiarum Fennicae (Helsinki), 1967.

34. Petrie, A.: Individuality in Pain and Suffering. Univ. of Chicago Press, Chicago, 1967.

35. Plum, A.: Ph.D. Thesis. Univ. of Florida, in preparation.

36. Rapin, I.: Evoked Responses to Clicks and Tones of Varying Intensity in Waking Adults. Electroenceph. Clin. Neurophysiol., vol. 21, 1966, pp. 335-344.

37. Rebert, C. S.; McAdam, D. W.; Knott, J. R.; and Irwin, D. A.: Slow Potential Change in Human Brain Related to Level of Motivation. J. Com. Psychol., vol. 63, 1967, pp. 20-23.

38. Regan, P.: An Effect of Stimulus Color on Average Steady-State Potentials Evoked in Man. Nature, vol. 210, 1966, pp. 1056-1057.

39. Regan, D.: Chromatic Adaptation and Steady-State Evoked Potentials. Vision Research, vol. 8, 1968, pp. 149-158.

40. Rictveld, W. J.; et al.: Visual Evoked Responses to Blank and to Checkerboard Patterned Flashes. Acta Physiol. Pharmacol. Meerl., 1967, pp. 259-285.

41. Rosner, B. S.: and Goff, W. R.: Electrical Responses of the Nervous System and Subjective Scales of Intensity. Vol. 2 of Contributions to Sensory Physiology, D. Neff, ed., Academic Press (New York, 1967).

42. Seigfried, J. B.; et al.: Evoked Brain Potentials Correlates of Psychophysical Responses: Hetrochromatic Flicker Photometry. Science, vol. 149, 1965, pp. 321-323.

43. Shagass, C.: Averaged Somatosensory Evoked Responses in Various Psychiatric Disorders. Vol. X of Recent Advances in Biological Psychiatry, J. Wortis, ed., Plenum Press (New York), 1968.

44. Shakow, D.: Psychological Deficit in Schizophrenia. Behav. Sci., vol. 8, 1963, pp. 275-305.

45. Shipley, T.; et al.: Evoked Visual Potentials and Human Color Vision. Science, vol. 150, 1965, pp. 1162-1164.

46. Shipley, T.; Jones, R. W.; and Fry, A.: Spectral Analysis of the Visual Evoked Occipitogram in Man. Vision Research, vol. 8, 1968, pp. 409-431.

47. Silverman, J.: Variations in Cognitive Control and Psychophysiological Defense in the Schizophrenias. Psychosom. Med., vol. 29, 1967, pp. 225-251.

48. Speck, L. B.; Dim, B.; and Mercer, M.: Visual Evoked Responses of Psychiatric Patients. Arch. Gen. Psychiat., vol. 15, 1966, pp. 59-63.

49. Spilker, B.; and Callaway, E.: Augmenting and Reducing Phenomena--A Cross Correlation Between Visual Evoked Responses and Kinesthetic Figural After Effects. Comm. in Behav. Biol., vol. 1, May 1968, abst. No. 05681153.

50. Stevens, J. R.; Sachdev, K.; and Milstein, V.: Behavior Disorders of Childhood and the Electroencephalogram. Arch. Neurol., vol. 18, 1968, pp. 160-177.

51. Straumanis, J. J.; Shagass, C.; and Schwartz, M.: Visually Evoked Cerebral Response Changes Associated with Chronic Brain Syndromes and Aging. J. Gerontology, vol. 20, 1965, pp. 498-506.

52. Vaughan, H. G.: Verbal communication at workshop of Objective Indicators of Psychopathology, sponsored by Biometrics Research New York Department of Mental Hygiene, Feb. 1968, Tuxedo, New York.

53. Walsh, T. J.; Smith, J. Z.; and Shipley, T.: Blindness in Infants. Amer. J. Opthalmol., vol. 62, 1966, pp. 546-556.

54. Walter, W. G.; Cooper, R.; Aldridge, V. J.; McCallum, W. C.; and Winter, A. L.: Contingent Negative Variation: An Electric sign of Sensorimotor Association and Expectancy in the Human Brain. Nature, vol. 203, 1964, pp. 380-384.

55. Walter, W. Grey: The Contingent Negative Variation as an Aid to Psychiatric Diagnosis. Paper presented at Biometrics Workshop on Objective Indicators of Psychopathology, discussion by W. Vaughn, Sterling Forest Conference Center, Tuxedo, (New York), Feb. 1968.

# A DESIGN CRITERIA FOR DDC SYSTEM FOR ULTRASONIC IRRADIATION OF BRAIN

Dr. Hideo Seo
Biophysical Research Laboratory
University of Illinois
Urbana, Illinois 61801

## ABSTRACT

Ultrasound irradiation for quantitative neuroanatomy at the Biophysical Research Laboratory requires 0.001 inch accuracy positioning of the transducer anywhere in the brain so that carefully controlled lesion can be introduced without destroying the intervening brain tissue except at its focal point.[1,2]

The above operation is done manually until the forthcoming use of automatic DDC system to prevent human errors and reduce operation period substantially.

This paper describes the basic procedures and steps desired for optimum design for automatic irradiation. The associated hardware for the PDP8 interface and software programs for the system diagnostics and the routine operating procedures are discussed.

## INTRODUCTION

The ultrasonic irradiation system developed at the Biophysical Research Laboratory requires the precise positioning with equally controlled irradiation in power levels in recent years. The latest transducer with a xtal oscillator plate and plastic lens can focus the ultrasound field to a conical shape and introduces a carefully selected lesion and the center of the focus can be positioned to 0.001 inch without destroying the intervening tissue. Histological study of the above animal brains (primates and cats) provided the finest of quantitative network structure of the nuclei in detail.

Manual positioning of the transducer to place its focal point to exactly preselected spots and subsequent irradiation of ultrasound are slow and tedious procedures; the tension of the operator in order to prevent errors increases as the number of irradiation increases. Since the above procedures are repetitive in nature and similar, this type of fatigue can be removed and the technician could run with less supervision the irradiation process if the system is driven by a small and reliable process control EDP on-line system. Also, it is felt that the selected number of lesions could be increased by more than a factor of 100 with other advantages such as simple data storage and retrieval capabilities.

The basic criteria in the automatic system for irradiation involves the following requirements:

(a) Precision beam plotting of transducers,

(b) System diagnostic and tests before irradiation,

(c) Data logging of the operating step for irradiation,

(d) Infalliable and simple programming methods to avoid human errors in certain sequential procedures.

The first dry run operation of the entire system was conducted successfully in November,1968 and proved the feasibility of the automatic irradiation system.

## IRRADIATION SYSTEM

A block diagram of the overall system is shown in Fig. 1. The heart of this system is a 4K PDP8 with additional 4K core storage. Inputs as well as outputs are TTY ASR 33 and high speed paper R/P (Model PC01 and Model PC03). A combination of IOT pulse circuits and interrupt facility are employed for appropriate selection of the peripheral system components and also for proper sequencial steps within the specific system components.

The positioning of a transducer is accomplished by three stepping motors (Superior Elec. Co.) which drive the lead screws of the Bridgeport milling machine carriage. Each stepping motor is pulse driven both ways by its indexer from the manual setting, or via PDP8 interface. The hardware feature of the indexer prevents backlash error.

The sensing of the position of the transducer is done with the linear measurement system by Philbrick Inc., within 0.0001 inch, utilizing the optical encoder, metallic grating lines on a glass plate, and the reversible electronic counter.

The output power level is regulated by the GR Precision Capacitor (Cs). The angular positioning of Cs is done by the same stepping motor indexer combination as in the x,y, or z positioning. The angular encoder is supplied from Theta, Inc.

The irradiation period is set by the program.

The last block indicated by the RADIATION is the ultrasound generator and shown in Fig. 2. This is a X-tal controlled R.F. amplifier in mHz range which drives the ultrasonic transducer. The power level of the output in Fig. 2 is up to 2 KW. However, we have a facility to produce 200 KW max. if needed.

## OPERATIONAL PROCEDURES

### (a) Precision Beam Plotting of Transducers:

This program is to determine the focal point of the ultrasonic transducer. The thermocouple is initially placed near the focal point. Ultrasonic field pattern is then properly scanned in order to determine the exact position of the focal point. It is convenient to select the path of scanning in the direction parallel to x,y or z axis. A small program which makes a decision to select the minimum number of travels of the transducer is incorporated.

(b) System Diagnostics and Preliminary Test Before Irradiation:

It consists of all I/O tests by operator intervention. For example, if the key T on TTY keyboard is depressed with the proper X-coordinates, the machine carriage is moved to that position and the X-sensing mechanism transmits the X-coordinate to PDP8 with the TTY printout. This is adequate and simple for X-positioning and sensing. The operator will repeat the similar test procedure by means of TTY keyboard intervention. Also, small logic program is added to check if all necessary tests are completed. If not, the program does not reply with printout a message which states "the test is completed". The operator should look carefully at the TTY printout again so that what tests were missing. Therefore, the operator must depress all required keys on TTY before entering the irradiation procedure.

(c) Irradiation Program

A new version of the operating system for auto-irradiation is being compiled and tested. PDP8, (1) computes many irradiation sites from the given X-ray data, (2) positions the transducer, (3) records the resultant coordinates sensed by the position detectors, and finally (4) irradiate the given amount of ultrasonic energy with the power level specified by Cs and also with the period given.

In this system, infalliable and simple programs are added to avoid human errors in certain procedures.

Some of the system components are not automated because of simplicity and less chance of error, however, we do not wish to overlook these errors. The bath and brain temperatures must be kept within ± .1 degree C. This temperature must be observed and verified at least for every five irradiations. The program simply count the number of irradiations and ask operators to look at the temperature and to press key T on TTY keyboard for verification if the operator forgets to press key T on TTY after five shots and proceeds. Also, similar programs are included to confirm the operator to see if high and low limits of A.C.R.F. power levels. The high power limit test will prevent the damage of expensive transducers and also the undesirable destruction of tissue. The low power limit test will avoid missing proper lesion introduction to the tissue.

## SUMMARY

The preliminary test shows that the human error due to fatigue may be almost eliminated in this automatic system and that the number of irradiations can be definitely increased (10 to 100 times) by this procedure.

The separate program on beam plotting reduces the usual manual procedure to a factor of at least 10.

Also, considerable time saving is expected in the orderly daily logging of the system test and the operational procedure.

This is a precision N/C machine. If cutters and bits are attached instead of the transducers this system can cut the transducer parts from the program and data tape.

This shows the great advantage of employing N/C system combined with DDC system in the Biomedical area and could obtain a high degree of accuracy, automatic logging of test and operation, reduction in processing time and the number of operators where procedures are long and tedious.

## ACKNOWLEDGEMENTS

The author wishes to express sincere appreciation to Professor F. J. Fry for technical discussions. Thanks also go to Mr. Albert Weidner for programming.

## References

1. Fry, W. J., "Intense Ultrasound in Investigations of the Central Nervous System", Advances in Biological and Medical Physics, V. VI, Academic Press, Inc., 1958, pp 281-384.
2. Fry, W. J., and Dunn, F. "Ultrasound: Analysis and Experimental Methods in Biological Research, Physical Techniques in Biological Research, Academic Press, Inc., 1962, pp 261-394.

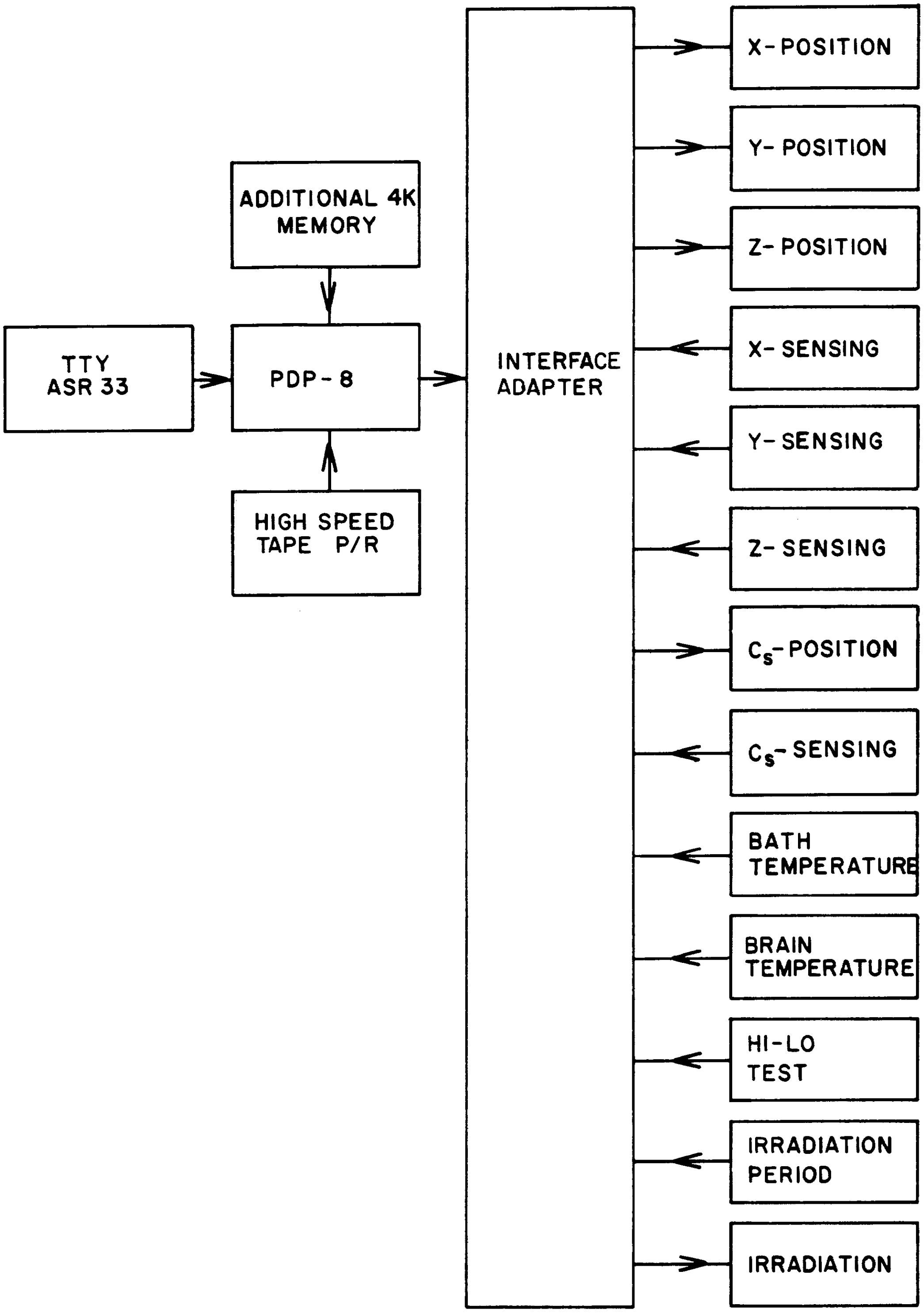


FIG. I

BLOCK DIAGRAM OF AUTOMATIC IRRADIATION SYSTEM

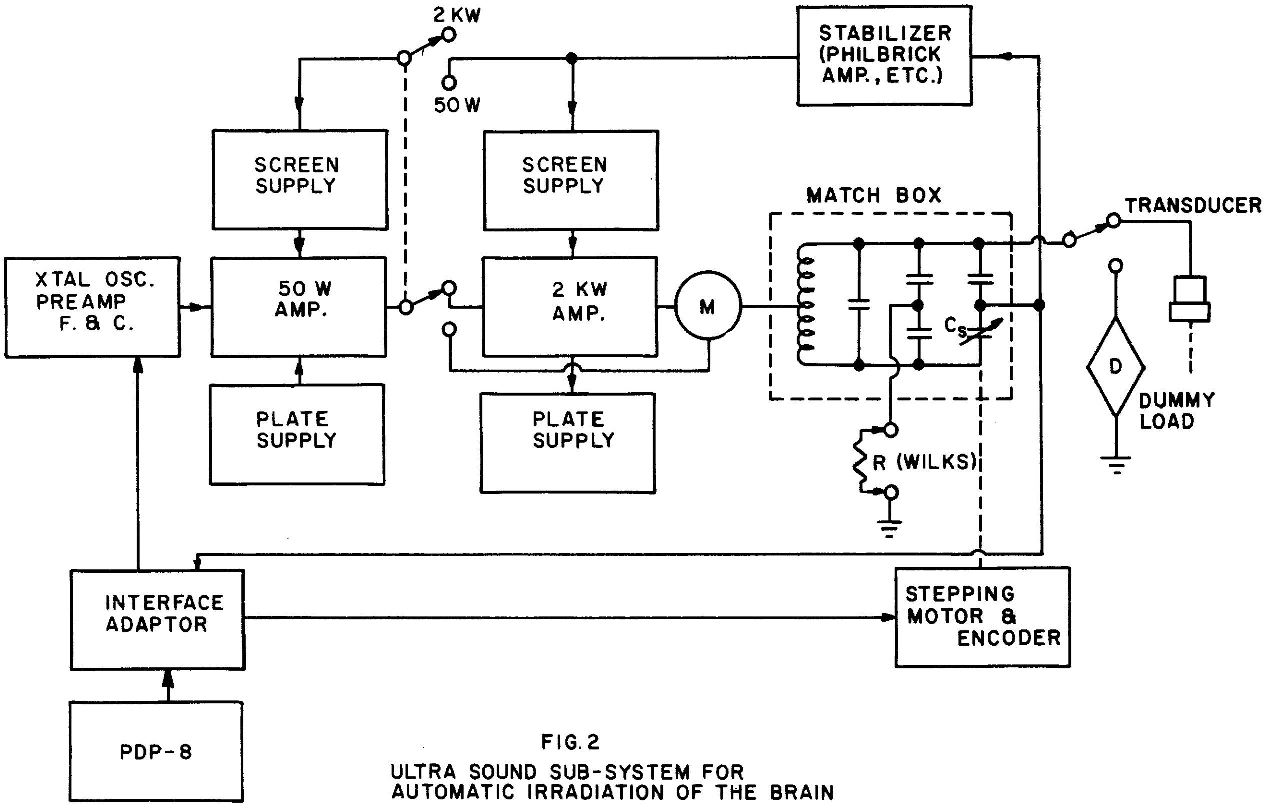


FIG. 2

ULTRA SOUND SUB-SYSTEM FOR AUTOMATIC IRRADIATION OF THE BRAIN

# AN APPROACH TO MICRO-IMAGE ENCODING AND AREA SCANNING OF CELLS OR NUCLEOLI OF BRAIN

Dr. Hideo Seo
Biophysical Research Laboratory
University of Illinois
Urbana, Illinois

## ABSTRACT

A new system consists of the three sub-systems, namely (1) PDP338-PDP8, (2) optical sybsystem which overlays visual display and photomicrographic images, and (3) photomicrograph projector.

Significant contributions in this system are in the use of a light pen with visual display as a data inputting device instead of mere functional control of the computer via interrupt mode and in the optical image superposition technique.

The boundary of a cell or nucleous is traced with a light pen, and the area is computed immediately within three percent accuracy. Real microscopic image is also successfully processed with CCTV (closed Circuit Television System).

Several ophthalmoscopic attachments were designed for the light pen so that the tracing of the image boundary can be made easily.

## INTRODUCTION

The physical neuroanatomy research at the Biophysical Research Laboratory, the University of Illinois called for the precision area measurement of cells and nucleoli in the animal brain.[1,2] The initial attempt was made by using a planimeter for the area measurement of the cells and nucleoli in the photomicrograph. The specimen was first magnified optically with a Leitz microscope and the photomicrograph was taken in 35 mm black and white film. Each negative was then enlarged with the ordinary photographic enlarger ten times and the final positive print was obtained. The average size of the nucleoli in the above positive print was in the order of one square cm. The routine area measurement taken by a planimeter introduces an error greater than 10%. The accuracy of 3% or better is desired for our cell scanning for the cell population distribution measurement in the latest quantitative physical neuroanatomy research.

In view of the above requirements, the architecture of the first realistic and economical system was formulated as shown in the Fig. 1. The above new system consists of the following three subsystems: (1) Direct Digital Computer Controlled (DDC) display sub-system (PDP338-PDP8), (2) optical sub-system which produces overlay of visual display and photomicrographic images, (3) photomicrographic projector.

## PDP8-PDP338 SUBSYSTEM

The computer organization in this paper is shown in Fig. 2. PDP8 with 4K main memory is augmented by 12K extra core memory. Two DEC tape units are used for high speed data transfer with the data break mode. Also, TTY ASR 33 is attached with a standard tape reader/punch and typewriter keyboard. PDP338 has a light pen and program display buttons. A 1024x1024 discrete point raster is used for the visual display. PDP 338 has hardware vector and character generators. Data transfer to and from PDP is via data break mode.

## OPTICAL SUBSYSTEM

The heart of the optical subsystem which superimposes the photomicrographic images onto the visual display scope of PDP338 was the dichotomic mirror. The mirror was constructed at EERL (Electrical Engineering Research Laboratory), University of Illinois. A glass plate (5"x5"x1/32") is placed inside the large vacuum deposition system and the silver coating of 150  200 A produced a satisfactory mirror.

In order to facilitate the proper and firm mounting of the optical system components, namely, projector, opaque screen on which the photomicrographic image is projected, and the half mirror, an enclosed oblong box of 1'x1'x4' was constructed with plywood. The box was mounted on two wedge-like supports so that the tilted surface of the display oscilloscope may be co-planar with the mirror image plane of the photomicrograph image as shown in the photographic print I.

## PHOTOMICROGRAPH PROJECTOR

A small movie camera Bolex-H16 attached to the ocular end of a microscope with a sylindrical coupler and a 16 mm motion picture of the photomicrograph is taken frame by frame. The microscope was the Olympus Model EHCrTr. The above 16 mm movie film is then developed and projected on the opaque screen by the movie projector Bolex-G816. The advancement of the movie film was made by manually turning the gear mechanism.

## SYSTEM PERFORMANCE

The accuracy of the area of the cell surrounded by its boundary depends on the quality of the photomicrograph and the number of dots (bright spots) which represent the boundary on the display screen.

There was no problem in setting the accuracy of the area on the display screen by the polygoral approximation to 3% in our experiment. This accuracy was tested by placing several stencils which were calibrated to 1% before the experiment.

A slight difficulty was encountered in tracing the cell boundary due to the uncertainty of photographic reproduction.

## CONCLUSION

(1) This is essentially a machine-aided image enhancement and area measurement system. Depending on the quality of photographic images, the accuracy of measurement can be made to 3% or better whereas 10% is the limit of measurement by the ordinary planimeter.

(2) The processing time of this system increases the output production more than ten times easily.

(3) This system provides two additional advantages:

(a) Economical storage of data: The coordinates of the boundary and its associated area can be recorded in the digital tape in the most economical and compact manner.

(b) Verification of data: the stored coordinates and area associated with the particular cell can be displayed on the visual display screen from the magnetic tape so that the old record of tracing can be verified by another operator or yourself at a later date.

(4) The same system can be used to count the total number of cells in the specifically bounded area. The minor change in the program leaves a flag (visually observable) near the cell which should be counted. After all the desired cells are tagged, the stored program can be initiated to count the total number of flags (which is equal to the number of cells) in less than a fraction of a second.

(5) The steep and tìlted surface of the oscilloscope in this system made it difficult to trace the cells physically. However, the scope can be mounted so that the face is horizontal and then there will be no difficulty in the tracing.

(6) A similar system with CCTV (COFU System) and microscope (Olympus Model EH Cr Tr) was tested satisfactorily as shown in Fig. 3. In the latest laboratory test of the similar system with the photomicrograph image from the video tape was found very satisfactory. The photomicrograph was taken by Sony's CCTV system with the Leitz microscope.

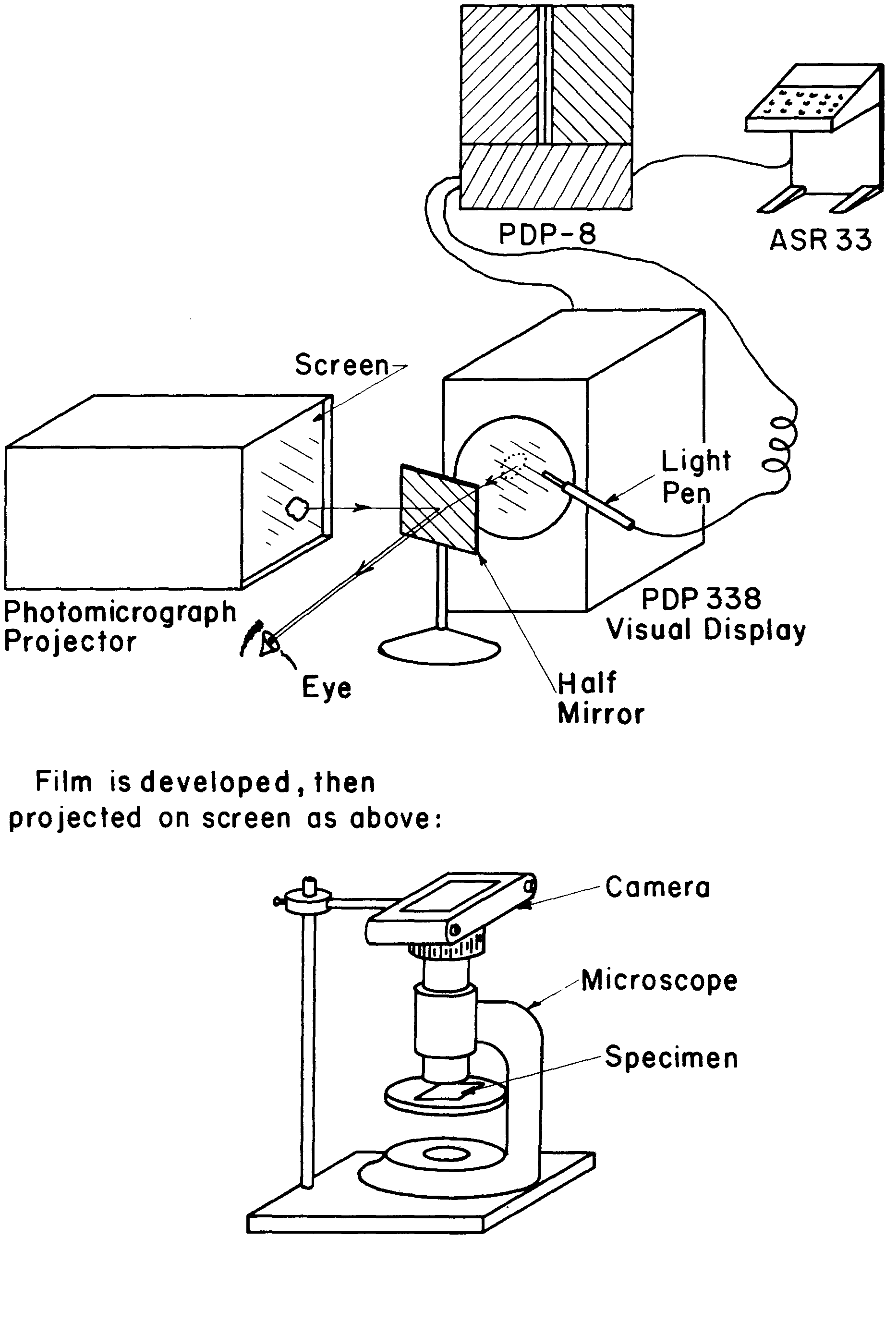
## ACKNOWLEDGEMENT

The author wishes to thank Mr. Carter and Dr. Richardson of DEL, U. of I. for providing him with the system organization and the available advanced software programs. Thanks also to to Mr. Eggleton of BRL, U. of I. for the design and construction of the optical subsystem. He also wishes to thank Albert Weidner for the programming done at BRL, U. of I.

## REFERENCES

1. Fry, W. J., Mammillary Complex of Cat Brain - Aspects of Quantitative Organization, Anat. Rec., v. 154, No. 1, January, pp 175-184.

2. Fry, W. J., Fry, F. J., Malek, R., Pankau, J., Quantitative Neuroanatomic Studies Implemented by Ultrasonic Lesions - Mammillary Nuclei and Associated Complex of Cat Brain, The Journal of the Acoustical Society of America, v. 36, No. 10, pp 1795-1835, October 1964.



Film is developed, then projected on screen as above:

FIG. 1

THE PRECISION AUTOMATIC AREA SCANNING SYSTEM FOR B.R.L.

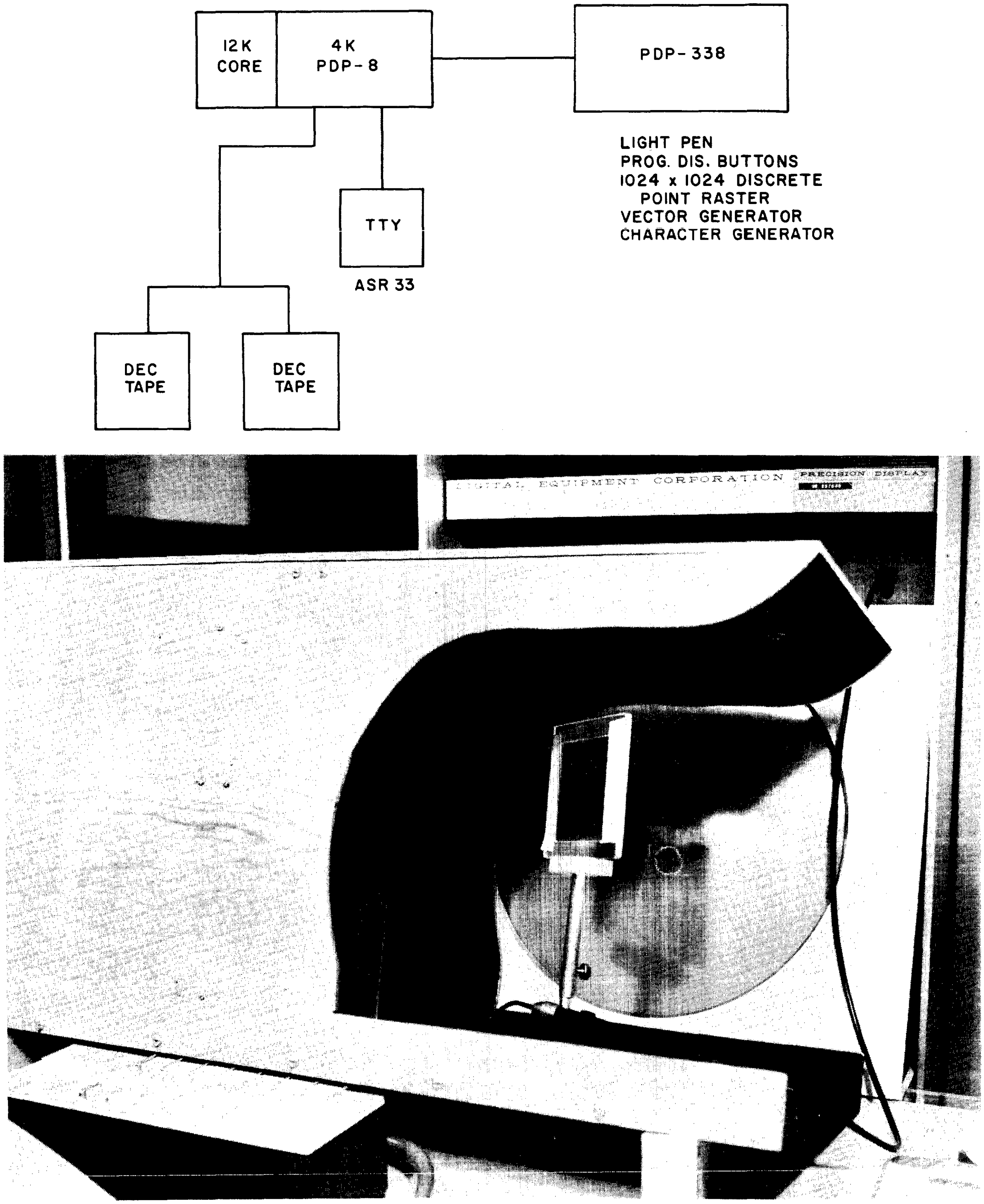


FIG. 2
PDP- 8 - PDP-338 CONFIGURATION
FOR MICRO-IMAGE ENCODING AND
AREA SCANNING

AT THE UNIVERSITY OF ILLINOIS

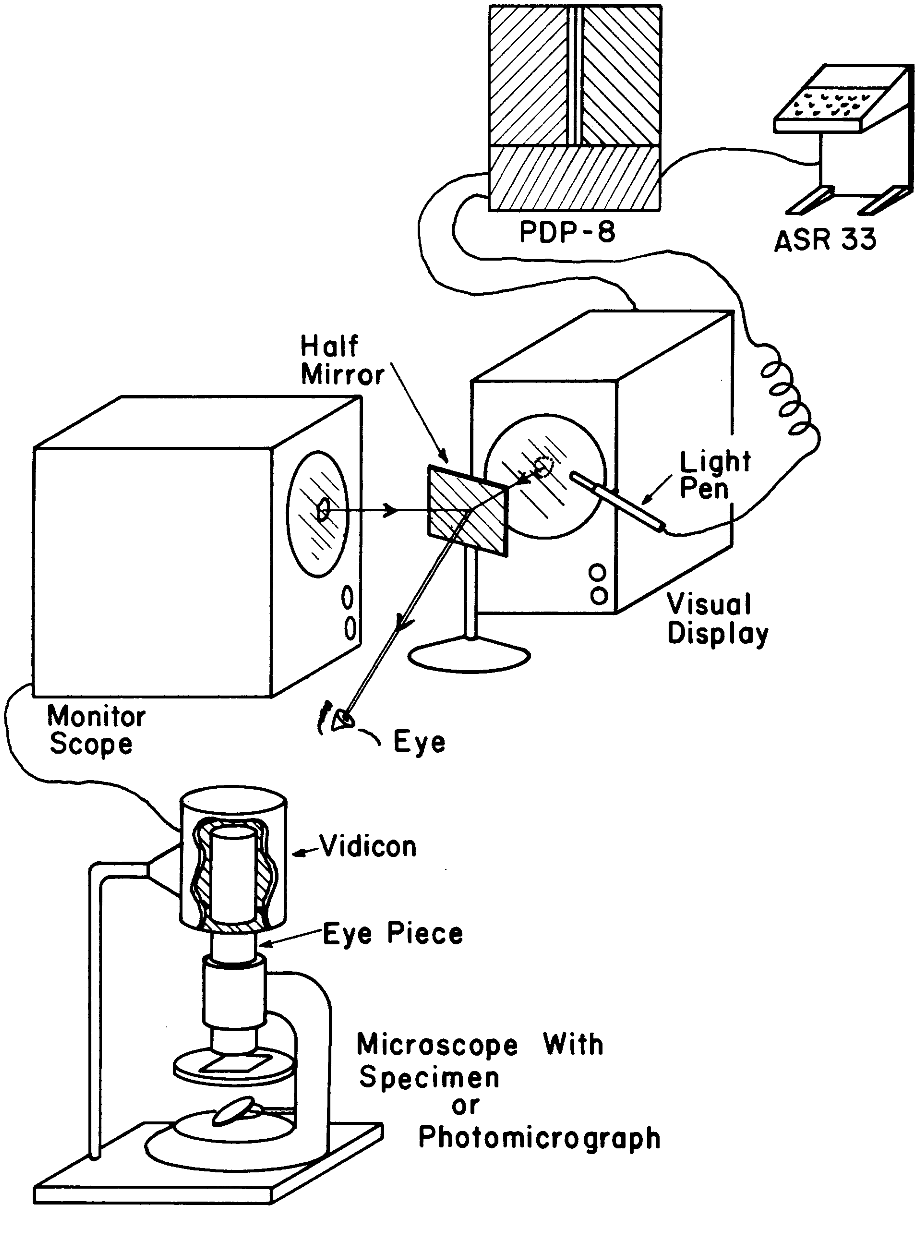


FIG. 3

THE CCTV SYSTEM OF IMAGE RECOGNITION AND AREA SCANING

# USE OF A PDP 8/S COMPUTER FOR ON-LINE MONITORING AND CONTROL OF BLOOD GLUCOSE IN HUMAN SUBJECTS

Arnold Henry Kadish, M.D., Cedars Sinai Medical Research Institute, Los Angeles, California, and Robert L. Litle, Beckman Instruments, Inc., Fullerton, California

## ABSTRACT

During the past several years, a continuous monitor for blood glucose has been used to study the response of human subjects to various inputs, thus providing data for modeling the human homeostatic system. Recently, a PDP-8/S computer has been incorporated into the system making possible on-line reduction of the data. In addition, a control system has been developed whereby computer derived control signals regulate glucose infusion rates to the subject. This has made possible more detailed studies of human natural control mechanisms.

## INTRODUCTION

The study of glucose homeostasis and its aberrations has received considerable impetus from the development, during this decade, of methods for the continuous monitoring of blood glucose in human subjects.[1, 2, 3]
In our laboratory, continuous glucose monitoring is carried out on a routine basis in support of two areas of research:

1. Theoretical studies of the homeostatic control system, i.e. modeling studies [4, 5] and

2. Studies in insulin-carbohydrate utilization relationships as they affect diabetic therapy. [6]

Experiments typically include intravenous tolerance tests (glucose, insulin, glucagon, growth hormone), and control experiments in which various glucose input policies are employed in an attempt to maintain a constant glucose level following the administration of insulin in various doses.

Since most of the data processing, simulation studies, file maintenance, etc. for the research are done in batch or time-sharing modes on large computers, it was decided to dedicate a real-time system to acquisition and control functions, with a minimum of data processing. Since the required data rates are very slow ($\leq$ =4/min) and data storage requirements are modest, a small, slow computer is adequate.

## LABORATORY SYSTEM

The experimental arrangement is shown in figure 1. Signals from the glucose monitor (0-10 MV.) are amplified and presented to a standard 138E A/D Converter interfaced with an 8/S. Numerical input and output are by way of an ASR33 teletype. Not shown are a 60 HZ 12-bit clock, used for sample-timing, and an analog recorder at the glucose monitor. The latter is a research prototype developed from a version previously described. [7]

The flow system required to transport blood continuously through the monitoring system of necessity introduces transport delay into the glucose signal. Under typical operating conditions delay times of about 5 min. are observed.

## CONTROL PREDICTORS

This time delay introduces a complication into the control algorithm. We have used a conventional approach to control--proportional, derivative and integral modes, but the error signal, its derivative and integral are predicted values:

$$C = K1*EP + K2*D(EP) /DT + K3* \int (EP) DT$$

Where EP is the predicted error signal.

The prediction, or extrapolation, from current data, is done by least-squares fit of a function of M parameters and time, to the current data point and its N-1 predecessors (N>=M). To avoid long initial delays and to simplify the fitting process we have used N=3, i.e. a three-point predictor "window", and have used only polynomial predictors (1st or 2nd degree). Our initial work has been with a quadratic. While this gives a good fit to rapidly changing glucose levels, the use of three points to fit a three parameter curve leaves no degree of freedom for noise-smoothing, hence the quadratic predictor is quite noise-sensitive. Simulation studies have shown that in the presence of our typical noise (1 mg% glucose), a 3-point linear predictor will generally perform better (in the sense of minimum RMS deviation from setpoint) than a quadratic. Figure 2 shows the predictor equations for the linear quadratic cases.

## CONTROL SYSTEM

In our work to date the control loop has been closed through a human operator manning a Sigmamotor variable infusion pump. The control signals are scaled to correspond directly to meter readings on the pump. As a control value is typed out at the TTY, the operator sets this value on the pump.

A completely automated control system is currently under construction. In this system each of twelve individual inputs may be enabled from a single (12-bit) control word. Where multilevel control of a variable is required, e.g. for glucose, several bits may be grouped in parallel. For example, as shown in figure 3, bits 8-11 of the control word activate four parallel input pumps operating at relative pumping rates of 1:2:4:8. Any of 16 discrete control levels can thus be selected. Other variables than glucose, viz. pH, electrolytes, etc. will be entered for control in subsequent studies.

## PROGRAM OPERATION

The requests and typical responses generated by the program in the initialization phase are shown in figure 4.

A. Name:, Date:, Miscl:
Provide for entering identification information.
B. Accepting system baseline:
When this message is printed the program enters an "idle" phase. When a carriage return is entered at the keyboard the current analog signal is converted and the value stored as the zero glucose level.

Following assignment of the base line the subject is catheterized. The time delay and time constant are determined from the analog record and entered at the keyboard. Meanwhile a sample of blood obtained at catheterization has been analyzed for glucose.[8] This value is then entered as the value of the calibration standard.

C. Accepting calib.:
Program idles until carriage return is received. Then current analog signal is assigned to the glucose level of the calibration standard. All subsequent calculations of glucose values are relative to this standard value.

D. Setpoint: (etc)
The control setpoint may be entered as a numerical value or assigned the current glucose level.

E. Kl = (etc)
The control parameters are entered. The example is of proportional control only (K2=K3=0).

F. Headings
The headings represent respectively, time, glucose, predicted glucose, predicted error, predicted derivative, predicted integral, and control signal.

After the headings are printed the program idles until a carriage return is entered. This synchronizes time zero, and initiates data acquisition and control.

Figure 5 shows print-out during execution of the program.

## RESULTS

Figures 6-9 show some typical results obtained with the monitor/control system.

Figure 6 shows an intravenous insulin tolerance test in a normal subject. Insulin was infused at 1 unit/min during the first three minutes. At the nadir of the glucose curve one mg . glucagon was administered i.v., and produced the rapid rise in blood glucose shown. Results of this type of test are used for determination of control parameters for models of the glucose regulatory system.

Figure 7 shows the results of a control experiment following infusion of three units of insulin. An initial open-loop policy was followed. Figure 8 shows a more successful control run for the same insulin dose. In this case full closed-loop control was used.

Figure 9 shows results obtained with a severe diabetic. The initial high fasting level of glucose was reduced by i.v. insulin injection before the control experiment. The control phase followed the injection of 5 units of insulin.

## REFERENCES

1. Weller, C. et al., Ann. N.Y. Acad. Sci., 87: 658 1960.

2. Ferrari, A. and Kessler, G. IBID., 729

3. Kadish, A.H., Am. J. Med. Elect., 3: 82 1964

4. Charette, W.P., Kadish, A.H. and Sridhar, R. Math. Bio. Sci. Spec. Ed. 1969

5. Charette, W.P., Thesis, Cal. Inst. Technology, 1968

6. Kadish, A.H., Proc. 5th Ann. Symp. Biomath. & Comp. Sci. 1967

7. Kadish, A.H. and Hall, D., Clin. Chem. 11: 869 1965

8. Kadish, A.H., Litle, R.L. and Sternberg, J.C. Clin. Chem. 14: 116, 1968

## ACKNOWLEDGEMENT

These studies were supported by a John A. Hartford Foundation Grant.

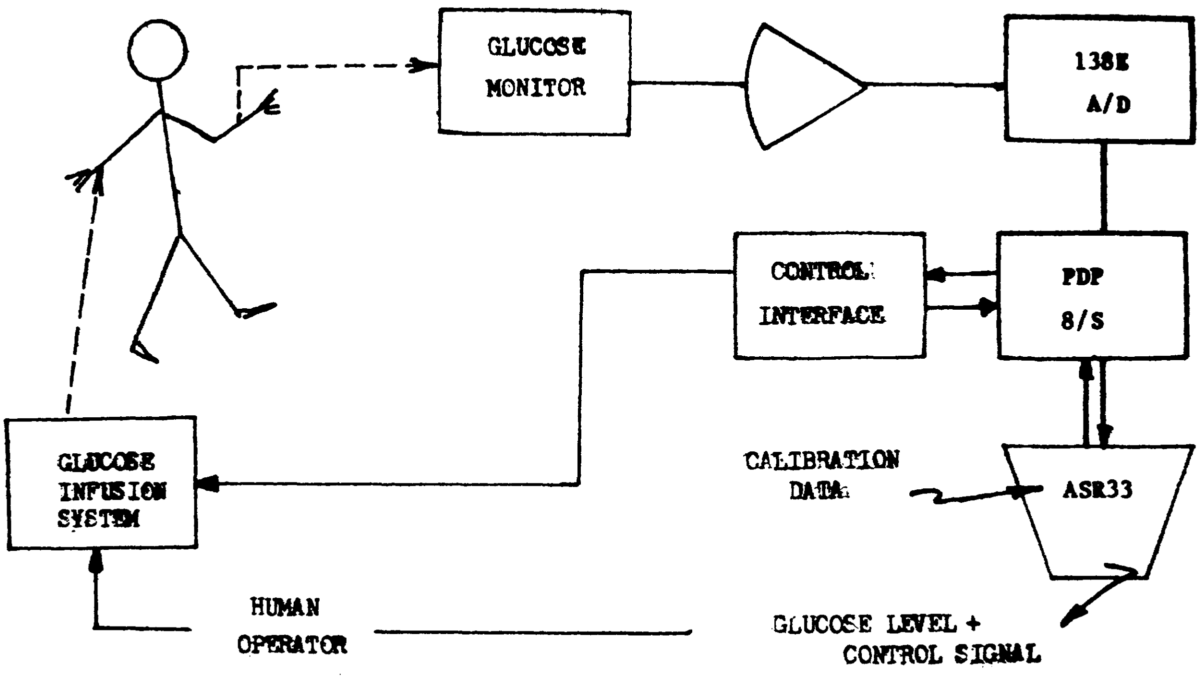


Figure 1 Glucose Monitor/Control System

## 3-POINT PREDICTOR

FIT: $a_o + a_1t + a_2t^2$

FROM: THREE EQUALLY SPACED POINTS, $b_{-1}$, $b_o$, $b_{+1}$

| LINEAR | QUADRATIC |
|---|---|
| $a_o = (b_{-1} + b_o + b_{+1}) /3$ | $a_o = b_o$ |
| $a_1 = (b_{+1} - b_{-1}) /2$ | $a_1 = (b_{+1} - b_{-1}) /2$ |
| $a_2 = 0$ | $a_2 = (b_{+1} + b_{-1} -2b_o) /2$ |

Figure 2 Predictor Equations

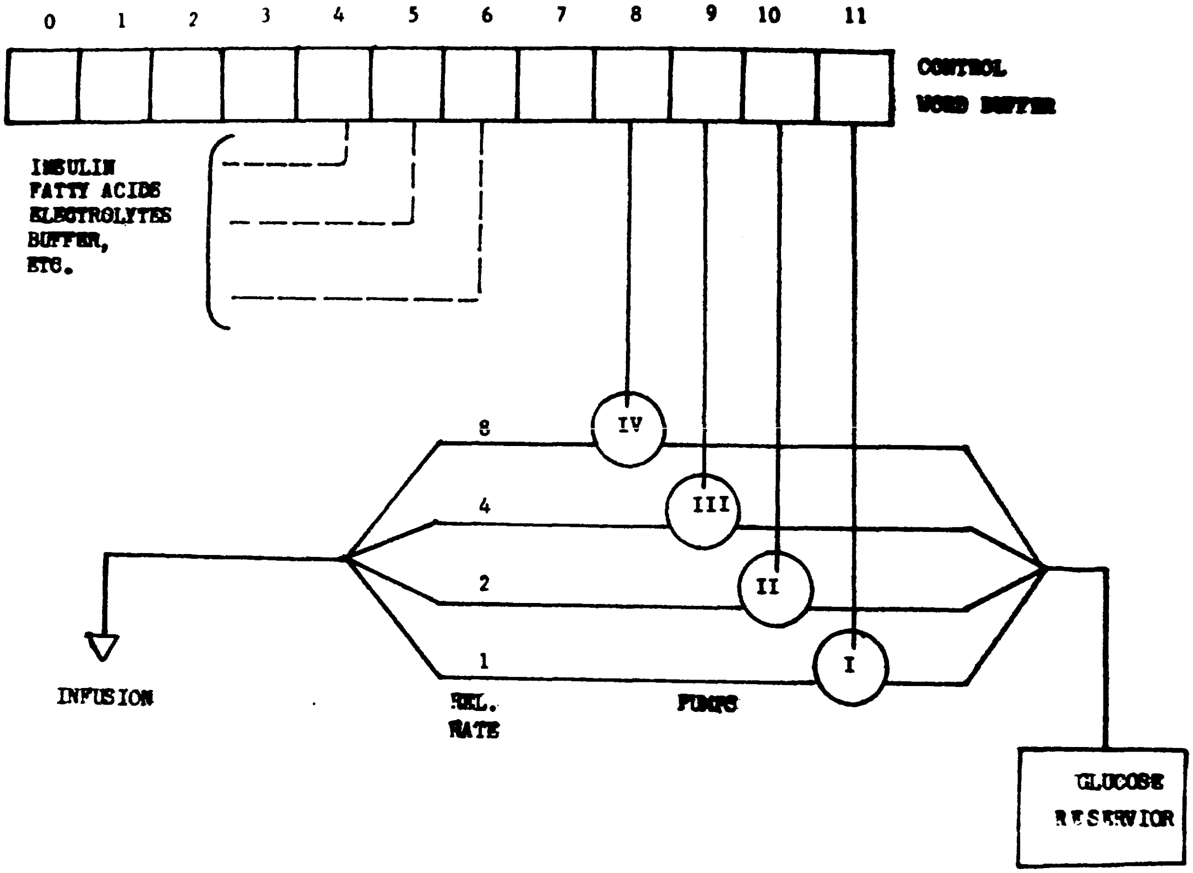
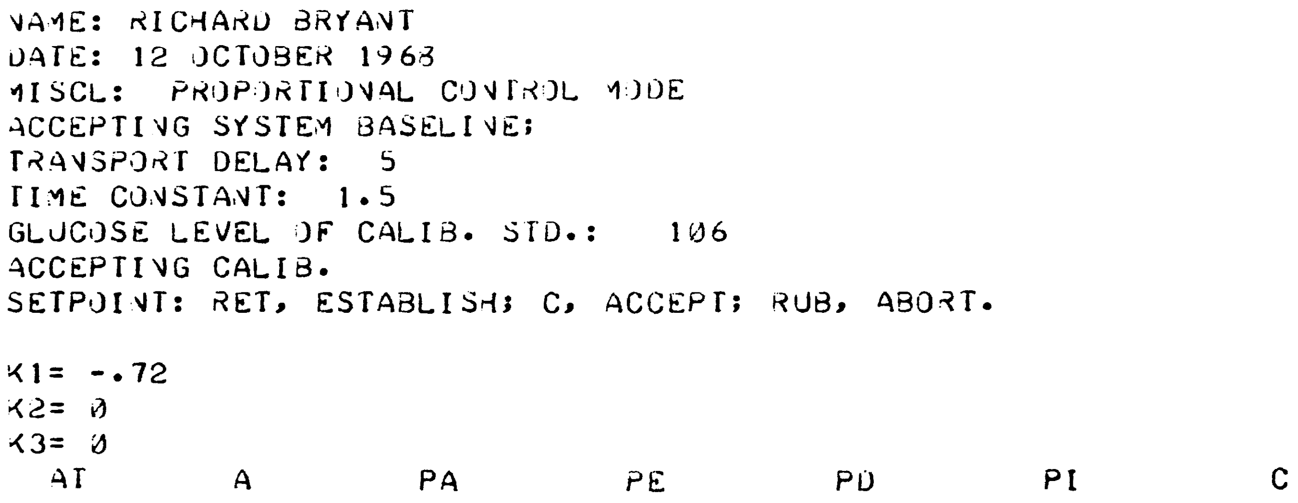


Figure 3 Control Logic



```
NAME: RICHARD BRYANT
DATE: 12 OCTOBER 1968
MISCL:  PROPORTIONAL CONTROL MODE
ACCEPTING SYSTEM BASELINE;
TRANSPORT DELAY:  5
TIME CONSTANT:  1.5
GLUCOSE LEVEL OF CALIB. STD.:    1Ø6
ACCEPTING CALIB.
SETPOINT: RET, ESTABLISH; C, ACCEPT; RUB, ABORT.

K1= -.72
K2= Ø
K3= Ø
  AT         A          PA          PE          PD          PI          C
```

Figure 4 Program Print-Out: Initialization

```
+ 38   +  99.32  +  97.65  -   9.18  -    .41  + 155.66  +   6.61
+ 40   +  98.48  + 100.99  -   5.84  +   1.04  + 147.45  +   4.20
+ 40   +  96.81  +  95.14  -  11.68  -    .41  + 127.70  +   8.41
+ 42   +  96.81  +  96.81  -  10.01  +    .00  + 119.35  +   7.21
+ 43   +  95.98  +  98.48  -   8.34  +   1.04  + 106.97  +   6.00
+ 43   +  95.14  +  97.65  -   9.18  +   1.04  +  91.94  +   6.61
+ 44   +  94.31  +  90.14  -  16.69  -   1.46  +  71.22  +  12.01
+ 46   +  95.14  +  99.32  -   7.51  +   1.46  +  69.41  +   5.40
+ 46   +  95.14  +  95.14  -  11.68  +    .00  +  54.25  +   8.41
+ 48   +  95.98  + 100.15  -   6.67  +   1.46  +  49.79  +   4.80
+ 48   +  96.81  + 100.99  -   5.84  +   1.46  +  42.70  +   4.20
+ 50   +  97.65  +  95.14  -  11.68  -   1.04  +  31.57  +   8.41
+ 50   + 100.15  + 105.99  -    .83  +   1.87  +  41.03  +    .60
+ 51   + 101.82  + 103.49  -   3.33  +    .41  +  38.81  +   2.40
+ 52   + 101.32  + 101.82  -   5.00  +    .00  +  32.13  +   3.60
+ 53   + 100.99  +  96.81  -  10.01  -   1.46  +  19.89  +   7.21
+ 54   + 101.32  + 105.99  -    .83  +   1.46  +  24.76  +    .60
+ 56   + 100.99  + 103.49  -   3.33  +   1.04  +  13.91  +   2.40
+ 56   + 100.99  + 100.99  -   5.84  +    .00  +   6.25  +   4.20
+ 58   + 100.99  + 100.99  -   5.84  +    .00  +    .41  +   4.20
+ 58   + 100.99  + 100.99  -   5.84  +    .00  -   5.42  +   4.20
+ 60   + 101.82  +  99.32  -   7.51  -   1.04  -   8.90  +   5.40
+ 61   + 102.66  + 106.83  +    .00  +   1.46  -   4.86  +    .00
+ 62   + 103.49  + 107.66  +    .83  +   1.46  -   5.28  -    .60
+ 62   + 104.33  + 108.50  +   1.66  +   1.46  -   4.86  -   1.20
+ 64   + 104.33  + 104.33  -   2.50  +    .00  -  10.85  +   1.80
+ 64   + 104.33  + 104.33  -   2.50  +    .00  -  13.35  +   1.80
+ 66   + 104.33  + 104.33  -   2.50  +    .00  -  15.85  +   1.80
+ 66   + 103.49  +  99.32  -   7.51  -   1.46  -  25.59  +   5.40
+ 68   + 103.49  + 103.49  -   3.33  +    .00  -  25.45  +   2.40
```

Figure 5 Program Print-Out: Execution

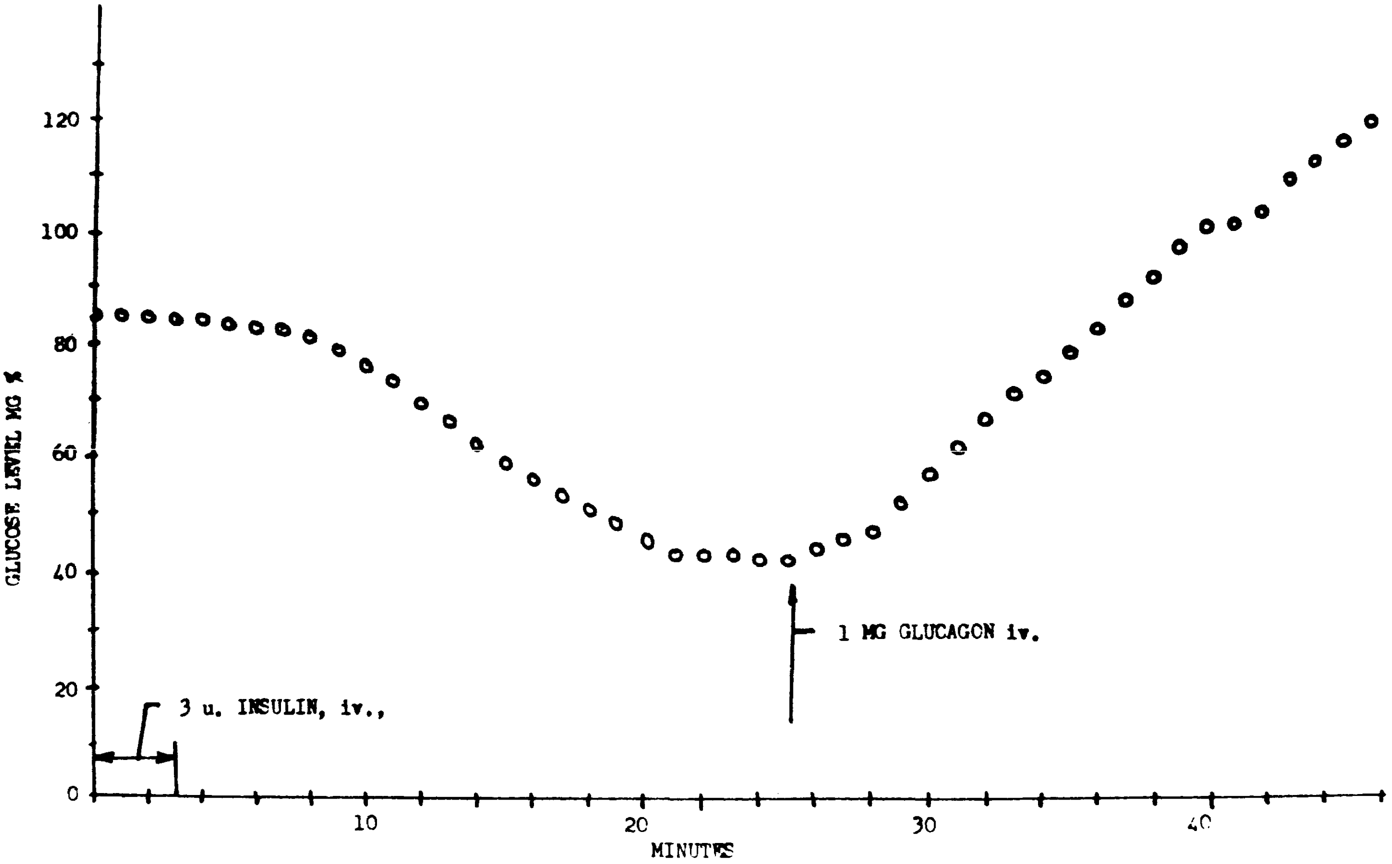


Figure 6 I.V. Insulin Tolerance Test and Glucagon Response. Normal Subject

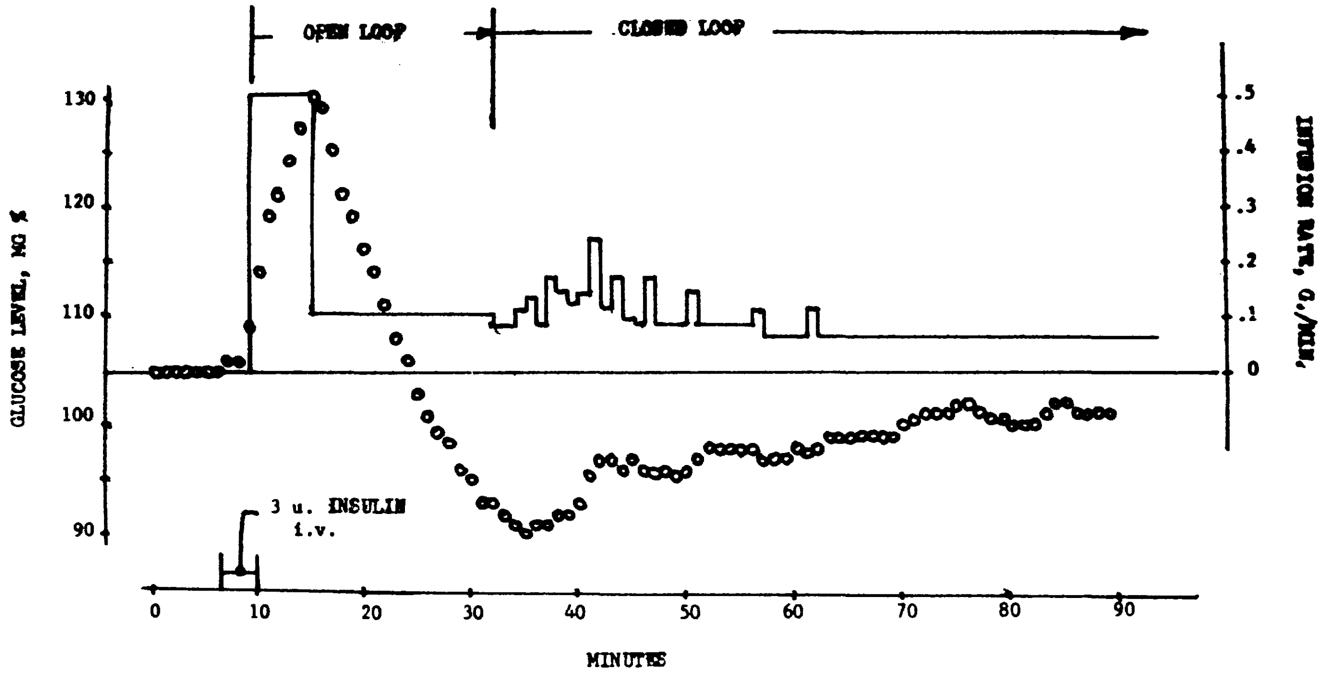


Figure 7 Glucose Control. Normal Subject. Circles: Glucose Level. Solid Line: Glucose Input

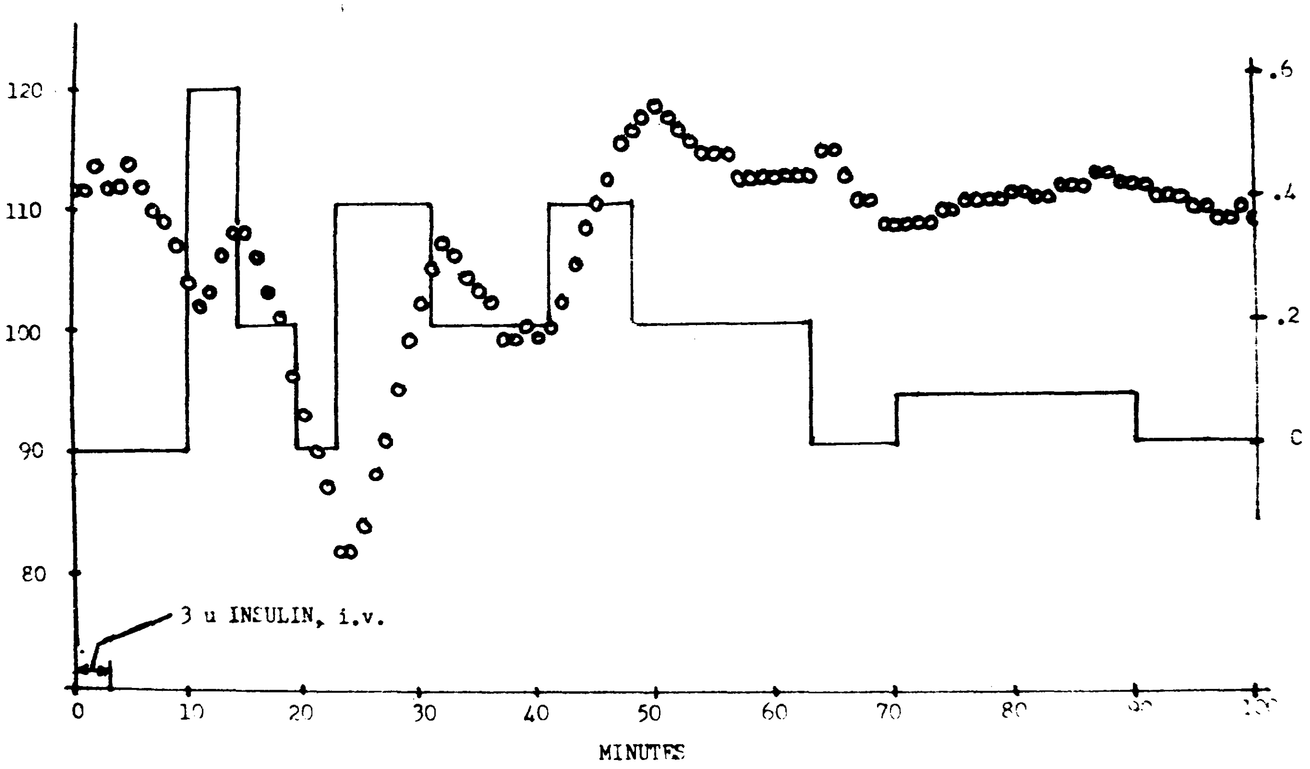


Figure 8 Glucose Control. Normal Subject

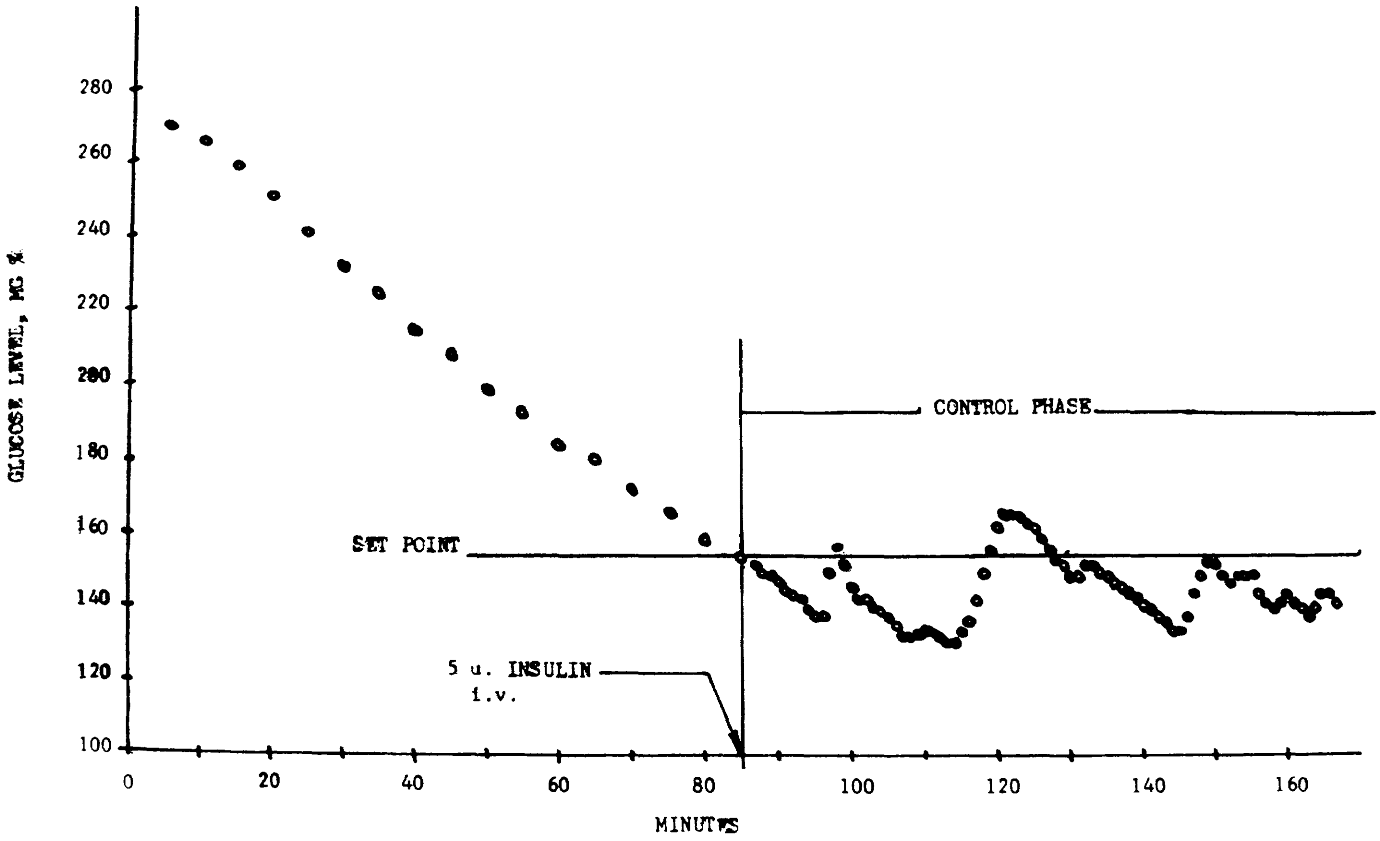


Figure 9 Glucose Control. Diabetic Subject

# A COMPUTER CONTROLLED CONTRACTILE COMPONENT LENGTH CLAMPING TECHNIQUE FOR SKELETAL MUSCLE

Paul J. Paolini, Jr.
Division of Biological Sciences
The University of Georgia
Athens, Georgia 30601

## ABSTRACT

An experimental method has been devised to allow the computer-controlled determination of stress-strain characteristics of the parallel and series elastic components (PEC,SEC) of excised amphibian skeletal muscle, according to the well-known three component mechanical analog of Hill. The information is used to calculate an extension vs time waveform of the muscle's SEC during an isometric twitch; the calculated curve is mechanically applied to the muscle during a subsequent twitch so that, on the average, no contractile component (CC) shortening is allowed to occur. The muscle's tension and volume change waveforms are recorded with this condition imposed. The tension-time response, corrected for PEC extension during the applied stretch, represents the muscle's active state intensity vs time curve, i.e. active tension developed at constant CC length. The system employed to control and monitor contraction parameters consists of an EAE-equipped PDP-8 computer and Teletype, a multiplexed A-to-D converter, a set of programmable relays, a display interface to an oscilloscope, and the required length, tension and volume transducers. One of the interface's two D-to-A converters provides the analog input to a high speed servo motor which sets muscle length. Program output consists of many keyboard-selectable oscilloscope display formats, x-y pen recorder graphs, and data table listings on the Teletype.

## Introduction

An accurate analysis of the variations in several physical properties during the contraction of living, excised muscle (and the dependence of these variations upon the many physicochemical factors which affect contractility) continues to be a major goal of the muscle physiologist. This task has been facilitated considerably in recent years by the advent of small laboratory computers which make possible on-line acquisition and reduction of data otherwise inaccessible during the course of the physiology experiment.

The techniques outlined in this paper rely not only upon the computer's ability as a high speed data processor, but also upon its potential as a feedback control device, where independent parameters (the input) are regulated in real time during the experiment to produce a desired behavior of dependent variables (the output). Specifically, the techniques allow (1) a single twitch contraction of muscle to be switched almost instantaneously from isometric (fixed length, tension varying) to isotonic (fixed tension, length varying) mode by using a program controlled servo motor to continuously adjust muscle length, and (2) a calculated length change to be imposed on the muscle during a twitch in such a way as to maintain length constancy of its contractile element (the active element of the widely accepted mechanical model of muscle, described below).

## Muscle Contraction Model

The configuration of the three elements constituting the Hill mechanical analog of muscle is shown in figure 1.[1] This analog provides a simplified operational model which can successfully explain many of the fundamental properties characterizing skeletal muscle behavior, i.e. the active and passive isometric length-tension curves, the force-velocity relationship, the series elastic tension-extension curve, and the active state intensity vs time curve.[2]

In the illustration, "CC" designates the muscle's contractile component, that portion capable of active shortening and tension development. "SEC" represents the series elastic component, an inert, undamped elastic structure which transmits the force developed by the CC to the muscle's external connections. An additional elastic property of the muscle, one which acts in parallel with the CC and the SEC, is the parallel elastic component, or "PEC." While some correspondence exists between the three lumped parameters and identifiable histological structures, these properties are in general distributed throughout the muscle.

The SEC obscures the actual time course and magnitude of the CC's capacity to shorten and develop tension: these are the mechanical parameters which are of greatest interest to the physiologist. Although the so-called force-velocity relation characterizes the CC's rate of shortening as a function of external load, obtained in a sequence of isotonic contractions, no simple method is available for establishing the time course and magnitude of CC tension development during a single twitch. If the muscle is stretched passively, tension is borne almost exclusively by the PEC, since the resting CC is highly extensible (figure 1b). If the active muscle is allowed to shorten against a load (1c), the SEC must first be stretched to an extent indicated by its tension-extension curve so that the SEC may bear the load. Or, if the muscle is activated but is held at a fixed length (1d), the CC still shortens as the SEC is being stretched: during the relaxation phase of the twitch, the extended SEC shortens to return the CC to its initial length. Externally monitored force development in this case corresponds to a complex situation of CC tension development at a continuously changing length. In these three simplest configurations of recording muscle length and tension interdependence (figure 1b-1d), the CC's ability to develop tension at a fixed length is not measurable. However, two indirect methods can be used to allow the curve representing CC tension vs time, at constant length, to be constructed.

The first method involves the release of an isometrically contracting muscle at various times following stimulation, so as to allow the muscle to take up slack in its connection to a force transducer and thereby redevelop to some extent its isometric twitch tension.[3] The peaks of resultant tension vs time curves represent coordinates on the falling phase of a tension-time curve at constant CC length, since these peaks represent the only instant during each curve when tension is maximum and the CC is therefore neither shortening nor lengthening. There are several means of estimating the rising phase of the twitch tension-time curve at constant CC length, but all are subject to some error. In one method, the muscle's developed peak tension is recorded as the response to quick stretches applied at various moments after stimulus.[1] A quick stretch extends the SEC so that the CC need not shorten against the SEC before external tension can manifest itself.

The curve illustrating tension development at constant CC length is also designated as the muscle's active state curve; the intensity of this curve at any time during the twitch can be defined in an equivalent way as "...the isometric tension which the contractile component can develop (or just bear) at that instant."[2] The active state curve is perhaps the best single parameter available as a measure of muscle contractility: it provides a measurement, in terms of a convenient mechanical parameter, of the basic molecular processes occurring within the muscle's contractile machinery, processes which are directly responsible for the muscle's ability to develop force and to shorten.

There are several additional interesting thermodynamic parameters which change during contraction besides the fundamental mechanical variables of length and tension. In particular, a considerable amount of work has been done during the past five or six years, in the muscle biophysics laboratory at the University of California, Davis, concerning the slight change in total volume undergone by contracting muscle.[5]

During a single twitch, for example, a 100 mg frog sartorius muscle will rapidly undergo an increase in volume by about $10^{-6}$ cc, reaching a peak in about 5 msec after stimulus (at room temperature), then abruptly decrease by approximately 2 or 3 x $10^{-6}$ cc and return to initial value, all within 50 or 60 msec. One major goal in this laboratory has been to establish a correlation between length-tension relationships of the three Hill model components, and the volume change.

Three PDP-8 programs have been prepared specifically to allow length-tension-volume interdependences to be distinguished among the PEC, SEC and CC. These programs will be outlined in turn.

## Instrumentation

A block diagram of the data acquisition and experiment control system is shown in figure 2. During an experiment, transducer output waveforms corresponding to muscle length, volume and tension are multiplexed, digitized and stored in the core memory of a PDP-8 computer. Stimulus application to the muscle is initiated by IOPs through device selector circuitry, as is the connection of one of the digital-to-analog converters to the input of a servo amplifier. To control muscle length the computer loads one D-to-A converter buffer with a number corresponding to the required length. A fast response servo motor rotates a shaft to which the muscle is tied; the muscle is clamped at its other end to a force transducer. The muscle is enclosed in a sealed chamber entirely filled with a physiological saline solution. Pressure fluctuations inside the chamber correspond exactly to volume changes within the muscle.[5]

## PEC Program

Table 1 lists the experiment control input variables and data output parameters used in the first of the three programs, a program designed to allow evaluation of the tension and volume change dependence upon imposed PEC length changes. The last six listings designate the relevant data output from the experiment. The first three correspond to single tension and volume coordinate responses to rapidly applied step changes in muscle length. Data output in these forms consist of x-y oscilloscope displays of sets of these coordinates (for a wide range of applied steps in length). The last three listings represent digitized waveform responses of tension and volume to a slowly applied length change (a ramp function). In this case, data output consists of all $400_8$ coordinates of one waveform plotted against the time-correspondent $400_8$ coordinates of another waveform. Distinction is made between rapidly and slowly applied length changes because the differences which occur in the muscle's response to each are physiologically significant.

Table 1

Parameters for PEC Program

| | |
|---|---|
| $\Delta L_{in}$ | programmed step change in length input signal to servo motor |
| $\Delta L$ | muscle length transducer output response to applied input $\Delta L_{in}$ |

| | |
|---|---|
| $L^0_{in}$ | length input signal producing no change in muscle length |
| $L^S_{in}(t)$ | step change length input vs time (from $L^0_{in}$ to $\Delta L_{in}$ to $L^0_{in}$ ) |
| $L^R_{in}(t)$ | programmed ramp length input signal vs time |
| $L(t)$ | actual time course of applied length change: servo output response to $L_{in}(t)$, step or ramp |
| $L^0$ | length transducer output at $L^0_{in}$ input signal |
| $P^0$ | tension transducer output at $L^0_{in}$ input signal |
| $V^0$ | volume transducer output at $L^0_{in}$ input signal |
| $P_{iso}(t)$ | isometric twitch tension vs time |
| $V_{iso}(t)$ | isometric twitch volume change vs time |
| $P(t)$ | tension vs time response to $L_{in}(t)$ input (step or ramp) |
| $V(t)$ | volume change vs time response to $L_{in}(t)$ input (step or ramp) |
| $\Delta P(\Delta L)$ | PEC tension vs (rapidly applied) extension |
| $\Delta V(\Delta L)$ | passive stretch volume change (rapid step) |
| $\Delta V(\Delta P)$ | PEC volume change dependence upon passive tension |
| $P(t)$ vs $L^R(t)$ | tension vs (slowly applied) length ramp coordinates |
| $V(t)$ vs $L^R(t)$ | volume change vs length ramp coordinates |
| $V(t)$ vs $P(t)$ | volume change vs passive tension coordinates |

Figure 3 diagrams the flow chart of the PEC evaluation program. All operations of display format selection ("A" through "J"), data generation ("K,L") and experiment control ("M-Z") are initiated via single letter keyboard commands. The program runs in display mode until a command is sensed, returning to the last chosen display format upon completion of any requested operation. Rapidly applied step functions ("M") or slow ramp functions ("N") of length change can be chosen as the length input signal, and the magnitude of the step function can be varied ("O,P"). Tension and volume coordinate step responses can be printed out singly ("X") or all stored data can be tabulated at the termination of the experiment. X-y recorder plots of stored waveforms can also be generated ("T").

The flow chart of figure 4 illustrates the manner in which keyboard selected subroutines set a branch point for two independent coordinate display (x,y) or for single parameter waveform display (y,t). Data output produced by the program in the form of x-y pen plots, Teletyped length-tension-volume tables, and Polaroid photographs of oscilloscope displays, provide information not only about the tension-extension characteristic of the muscle's PEC, but also about the relative contribution to the net observed volume change in active muscle from the PEC.

SEC Program

The conventional method of SEC evaluation is known as the "quick release" technique. A muscle is attached to an isotonic lever which is held fixed in position by an electromagnetic stop. The muscle is stimulated and develops tension; some time during this isometric twitch the stop is removed and the muscle shortens against a load attached to the lever arm. Upon release, the muscle shortens quite suddenly, because the SEC shortens from one length characteristic of isometric tension at the instant of release, to a length corresponding to the new isotonic (load) tension. After this sudden length change, the muscle continues to shorten in a slower fashion as the CC is allowed to contract. This experiment is rather elaborate, requiring as it does a specialized apparatus, including the electromagnetic release, isotonic lever and intrinsic force transducer; and, during the experiment the afterload on the lever must be changed frequently. Such a technique could not easily be employed simultaneously with volume measurements, which require the muscle to be enclosed in a sealed chamber.

The new approach described here avoids many such experimental inconveniences by relying on the controlled tension via length feedback capability of the computer system represented by figure 1. Figure 5 helps to explain the basic features of this approach. The magnitude of an isometric twitch tension-time record is calculated and a range of convenient isotonic loads is chosen (from a heavy load $P_i = P_0$ to a light load $P_i = \Delta P'$). The lower segment of figure 5 illustrates an idealized controlled tension response of the SEC program. Basically, this program mimics the quick release technique by allowing the muscle to develop tension isometrically from $t_0$ (stimulus applied) to $t_1$ (release time). At $t_1$, muscle length is driven by the computer so that tension reaches the desired isotonic load $P_i$ as quickly as possible. If $t_2$ is the time at which release is accomplished, then the controlled length change from $t_1$ to $t_2$ corresponds to SEC shortening $\Delta X_{SEC}$ under tension change $\Delta P_{SEC}$.

Table 2 summarizes those terms (in addition to table 1) used for input control and data output parameters of an SEC evaluation experiment. Here $L_{in}(t)$ is now the stored servo input which has been generated by the program during the release, to accomplish the release. The last three terms constitute the required data output coordinates.

The basic flow chart for the SEC evaluation program is presented in figure 6. As before, there are many keyboard selectable waveform displays. The quick release controlled length twitch is elicited by the command "H": during this operation the length control waveform $L_{in}(t)$

is synthesized and stored. "I" causes $L_{in}(t)$ to be reapplied passively to the muscle, and the accompanying pressure transducer output $V_{art}(t)$ is stored (see Artifact Subtraction below). "J" triggers a simple isometric twitch, while "K" and "L" initiate the calculation of isotonic load, and "M" causes the SEC length-tension-volume coordinates to be listed on the Teletype.

Table 2

Additional Parameters for SEC Program

| | |
|---|---|
| $L_{in}(t)$ | program-synthesized length control signal vs time |
| $P_0$ | maximum amplitude of $P_{iso}(t)$ waveform |
| $\Delta P'$ | programmed tension interval (e.g. $P_0/16$) |
| $P_i$ | current "isotonic load": a programmed value of tension |
| t | current time (relative address in a stored waveform) |
| $t_1$ | time of isotonic release (apply condition of $P(t) = P_i$ ) |
| $t_2$ | time after release instruction at which $P \leqq P_i$ (mechanical response time) |
| $t_3$ | time at which L returns to $L^0$ (isotonic-isometric transition of afterloading) |
| $t_4$ | time of forced return to isometric mode (set $L_{in} = L^0_{in}$ ) |
| $V_{art}(t)$ | volume vs time waveform with $L_{in}(t)$ applied passively |
| $V(t)-V_{art}(t)$ | volume change vs time in muscle during SEC evaluation (quick release) twitch, less reproducible artifact caused by rapid mechanical movement during application of $L_{in}(t)$ |
| $\Delta X_{SEC}$ | $= L^0 - L(t_2)$ SEC extension |
| $\Delta P_{SEC}$ | $= P(t_1) - P_i$ SEC tension |
| $\Delta V_{SEC}$ | $= (V(t_1)-V_{art}(t_1)) - (V(t_2)-V_{art}(t_2))$ volume change accompanying extension of SEC |

Figure 7 presents the sequence of operations performed during the quick release controlled length twitch. In each cycle, V, P and L coordinates (sampled transducer outputs) are stored and then tested: first, time limits (the relative channel numbers of a waveform being stored) are checked to see if it is too early (preceding the release time) or too late (almost at the conclusion of the twitch) to exert experimental control. Next, length is tested to insure that it has not been set to exceed a maximum allowable (equal to initial, resting) length. Then the length input is incremented or decremented according to whether tension is greater than or less than what the released tension should be. The cycle stores the time coordinates representing the instant at which the release is achieved ($t_2$) and the time at which the afterloaded stop is applied: that is, the moment when the muscle is about to be stretched beyond reference length by the load ($t_3$). The time required for each cycle following data coordinate storage and during the test sequence is independent of the path; each branch point out of the sequence cycles a delay subroutine the required number of times before reinitiating A-to-D conversion for the next set of coordinates.

Significant data outputted from this program consist of the Teletype compilation of SEC length and volume changes corresponding to a range of isotonic release loads between $\Delta P'$ and $P_0$. A curve representing SEC extension vs tension is punched out as a binary tape through the DEC software package, ODT (which is used in conjunction with all programs involved in these experiments); the curve constitutes essential input for the CCLC program described below.

Artifact Subtraction

The rapidity with which changes in length must be imposed upon the muscle during the release experiment introduces a serious problem in making simultaneous volume measurements. The rapid (complete in 60 or 70 msec) clockwise (stretch) and counterclockwise (release) movement of the shaft and pulley to which the muscle is attached cause a relatively enormous shock artifact in the volume waveform. It can be demonstrated that these artifacts are caused by the rapidity of the movement rather than by an actual change in the muscle chamber volume.[6]

However, shock artifacts in the volume change records are consistent and repeatable during the course of an experiment. Because of this consistency it is possible to "extract" the effect of artifact by subtracting such a waveform from the original record of active muscle response with superimposed artifact.

Figure 8 demonstrates how this is done. "A" represents the volume change accompanying the application of a rapid stretch pulse to a resting muscle: the volume response is totally obscured by artifact. (Some overloading of the display interface 10 bit range is occurring here.) If this record is regenerated and the newly evoked record subtracted digitally, time coordinate by coordinate, from the original waveform, a difference waveform such as "B" results. This baseline waveform shows some scatter, especially where onset and return of the rapid stretch occur, but most of the artifact is absent. In the same way, if a $V_{art}(t)$ signal accompanying the controlled stretch response of unstimulated muscle is subtracted from a V(t) signal, most of the artifacts effect can be removed and the resultant $V(t)-V_{art}(t)$ waveform presumably corresponds to an actual response of the muscle during contraction. ("C") This method of artifact subtraction is used by display subroutines in both the SEC evaluation and CCLC programs to provide usable data.

CC Length Clamp Program

The third program utilizes data provided by the previous program in order to control muscle length in such a way as to prevent the CC from shortening at the expense of the SEC as it does during an isometric twitch (figure 1d). A controlled rate stretch, during the rising phase of the twitch, followed by a controlled release back to initial length during the relaxation phase, is applied so that the SEC is "pulled out" by the stretch (figure 1e). Since the stretch is applied slightly after the CC has been activated, the CC is no longer compliant, as it is in resting muscle, but exhibits a high resistance to stretch,[1] so that the effect of the stretch is to extend the undamped SEC.

Table 3 lists those parameters required, in addition to tables 1 and 2, to specify input and output variables in the CCLC experiment. $\Delta X_{SEC}(\Delta P_{SEC})$ represents the source data, a sequence of x-y coordinates which are most easily fitted by an exponential function.[4] Since $t_1$ is constant and the course of developed tension is identical throughout the sequence of isometric twitches released to various isotonic loads in the SEC evaluation program, the tension $P(t_1)$ is the same for each release. Therefore, the calculated sequence of SEC tension changes, $\Delta P_{SEC}$, vary by intervals of $\Delta P'$. A tension-extension curve for the SEC can then be stored as a sequence of extension values, starting in some memory address identified with $\Delta P'$ tension, with successive locations representing $2\Delta P'$,... through $P_0$. The number of memory locations required for curve storage (called "W" here) for the illustrative example in figure 5, would be 8. $\Delta X_{SEC}(t)$ designates the controlled length change signal to be applied to the muscle for the purpose of holding CC length fixed. $P_{art}(t)$ is analogous to $V_{art}(t)$: it is a transducer output waveform accompanying the length change applied to the unstimulated muscle. Since the muscle is being extended beyond reference length, its PEC bears a changing tension component. As described under Muscle Contraction Model, the active tension response during CC length clamping, when corrected by $P_{art}(t)$, characterizes the muscle's active state intensity-time curve.

Table 3

Additional Parameters for CCLC Program

| | |
|---|---|
| $\Delta X_{SEC}(\Delta P_{SEC})$ | SEC tension-extension relation evaluated from previous program |
| $\Delta X_{SEC}(t)$ | waveform synthesized from $P_{iso}(t)$ and $\Delta X_{SEC}(\Delta P_{SEC})$ data:<br>used as $L_{in}(t)$ to clamp the CC |
| W | width of $\Delta X_{SEC}(\Delta P_{SEC})$ waveform:<br>number of addresses corresponding to SEC tension in range of 0 to $P_0$ |
| $P_{art}(t)$ | tension vs time response of muscle stretched passively by $L_{in}(t)$ |
| $P(t)-P_{art}(t)$ | active tension vs time during CC length clamping, less passive tension vs time: this waveform represents the muscle's active state curve |

The flow chart in figure 9 illustrates the major features of the CCLC program. Once again, the program cycles continuously in display mode until a letter is struck on the keyboard. "A" through "K" call for various waveforms to be displayed, while "L" through "R" elicit fresh data. Two DEC software packages are used with the program to provide floating point calculation capability: calibrated markers are shown with each possible waveform display.

Figure 10 indicates roughly how the controlled length change waveform is calculated. (The steps involved with interpolation of extensions corresponding to tensions between the evenly spaced tension increments are not shown.) Basically, each coordinate of a stored isometric tension-time waveform is "looked up" in the SEC tension-extension curve, and stored to make a new waveform with the same time scale.

Sample Data

Figure 11 summarizes representative data from a typical CCLC experiment. The first curve is a smoothed plot of SEC tension-extension (i.e. somewhat idealized compared to the actual appearance of experimental data). Each waveform display includes an identification tag, x and y markers, and a listing of marker magnitudes and units of measure. Three words of storage space are provided for fixed point marker magnitudes of each display.

Waveform "A" of figure 11 corresponds to isometric twitch tension developed by a 125 mg frog sartorius muscle. Each tension coordinate of "A" has been transformed into a SEC extension coordinate from "H" and assembled into the waveform "I". This last waveform constitutes $L_{in}(t)$, the servo motor input applied as a length change to the muscle. The data waveform beneath "I", designated "G", is the length transducer output signal, i.e. the resultant change in muscle length with time. "G" lags "I" by a few msec; since the waveform is roughly sinusoidal (or at least exhibits no sharp breaks) it is possible to partially compensate for the mechanical response time of the servo motor by shifting the input waveform slightly to the left, that is, to apply it slightly earlier in phase than actual twitch occurrence.

The last curve of figure 11 illustrates the muscle's tension development during application of the controlled stretch. Assuming the validity of the three component mechanical model, it is this waveform which represents the CC's active state intensity vs time curve.

This last waveform, "F", was adjusted to maximum usable display height by the subroutine entered through keyboard command "U" of the CCLC program (figure 8), a normalization subroutine. Since "F" is the largest anticipated tension response generated in the experiment, it is convenient to then multiply all other tension displays by the same normalizing factor, so that all display amplitudes can be compared directly. The normalizing factor is used in conjunction with calibration constants for the various transducers, preamplifier gain settings, etc., to provide automatic calibration of the x-y scale markers for each waveform, in the subroutine mentioned earlier.

The active state curve differs from an isometric twitch tension waveform in several conspicuous ways. A comparison of "A" and "F" in figure 11 indicates that, while tension onset occurs at about the same instant, its rate of rise is considerably greater in "F". Active state peak intensity is 2 1/2 to 3 times as large as $P_0$, and occurs several msec earlier than peak twitch tension; and, the active state curve exhibits a more rapid rate of tension decay, returning to resting tension considerably earlier than in the isometric twitch. A more complete and quantitative description of experimental results from the CCLC technique will be presented elsewhere.

## Summary

The incorporation of a small, high speed on-line digital computer as a data reduction and control device for a muscle physiology laboratory has made possible several innovations in experimental techniques. In particular, the following have been discussed in the present paper: (1) the calculation of controlled stretches exactly tailored to the elastic properties of a muscle preparation, designed to maintain a constant contractile component length during contraction; (2) the availability of programmable contraction modes, isometric or isotonic, with possible instantaneous transitions between the two modes, or to phrase it differently, the control of instantaneous tension during twitch contraction; and, (3) the subtraction of repeatable artifacts much larger than the data signal itself.

## Acknowledgements

Much of the work described here was done while the author was a Public Health Service Postdoctoral Trainee in the Department of Zoology, University of California, Davis campus, in the laboratory of Prof. Ronald J. Baskin, under PHS grant no. AM 10726-03. Dr. Baskin's generosity in continuing to make available the equipment necessary for this work is gratefully acknowledged.

## References

1. Hill, A.V. The abrupt transition from rest to activity in muscle. Proc. Roy. Soc.(London) Ser.B 136 pp. 399-420 (1949)

2. Wilkie, D.R. The mechanical properties of muscle. British Med. Bull. 12 (3) pp. 177-182 (1956)

3. Ritchie, J.M. The effect of nitrate on the active state of muscle. J. Physiol. 126 pp. 155-168 (1954)

4. Davson, H. A Textbook of General Physiology 3rd ed. Little, Brown & Co., Boston (1964) pp.954-5.

5. Baskin, R.J. and P.J. Paolini. Muscle Volume Changes. J. Gen. Physiol. 49 (3) pp. 387-404 (1966)

6. Paolini, P.J. Volume Changes upon Stretch of Resting and Active Striated Muscle. Ph.D. Dissertation, University of California, Davis. June, 1968.

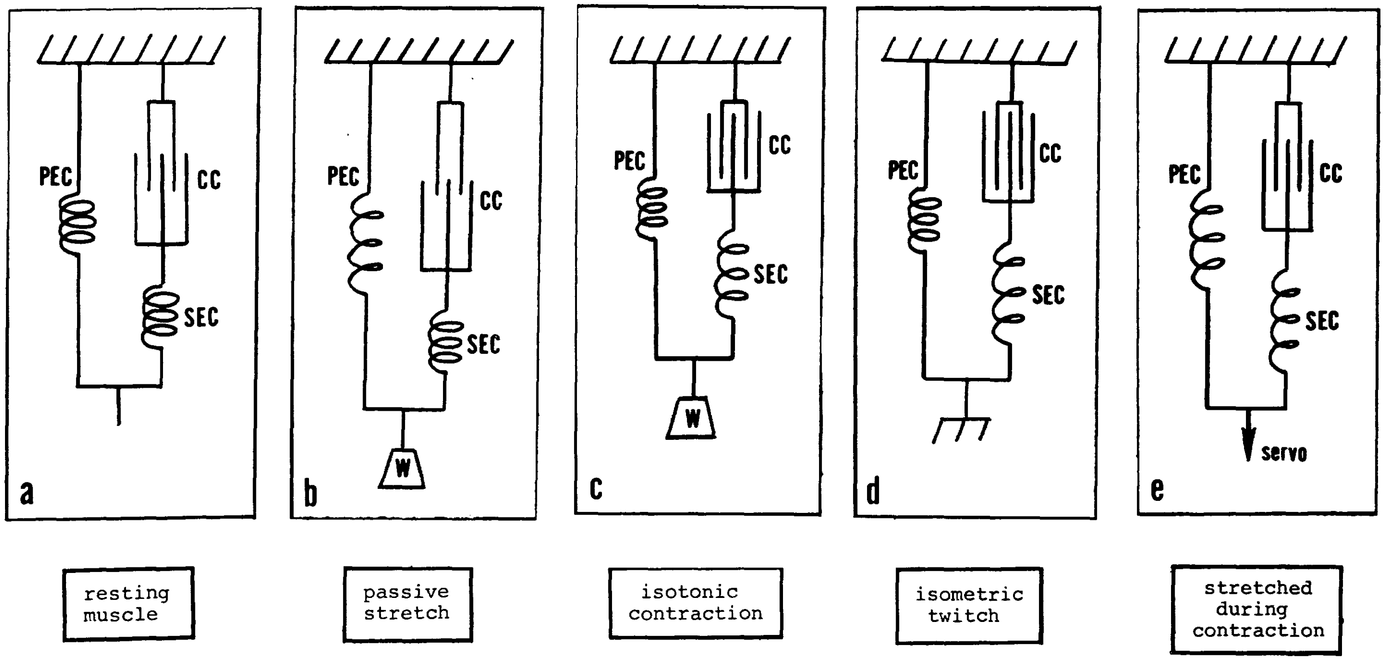


Figure 1 Three component mechanical analog of muscle

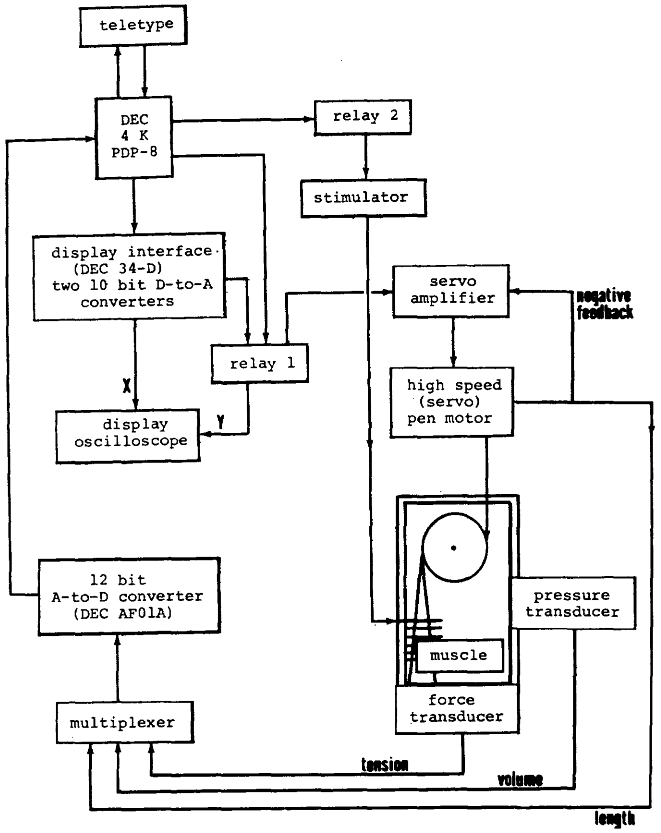


Figure 2 Experiment control and data acquisition system

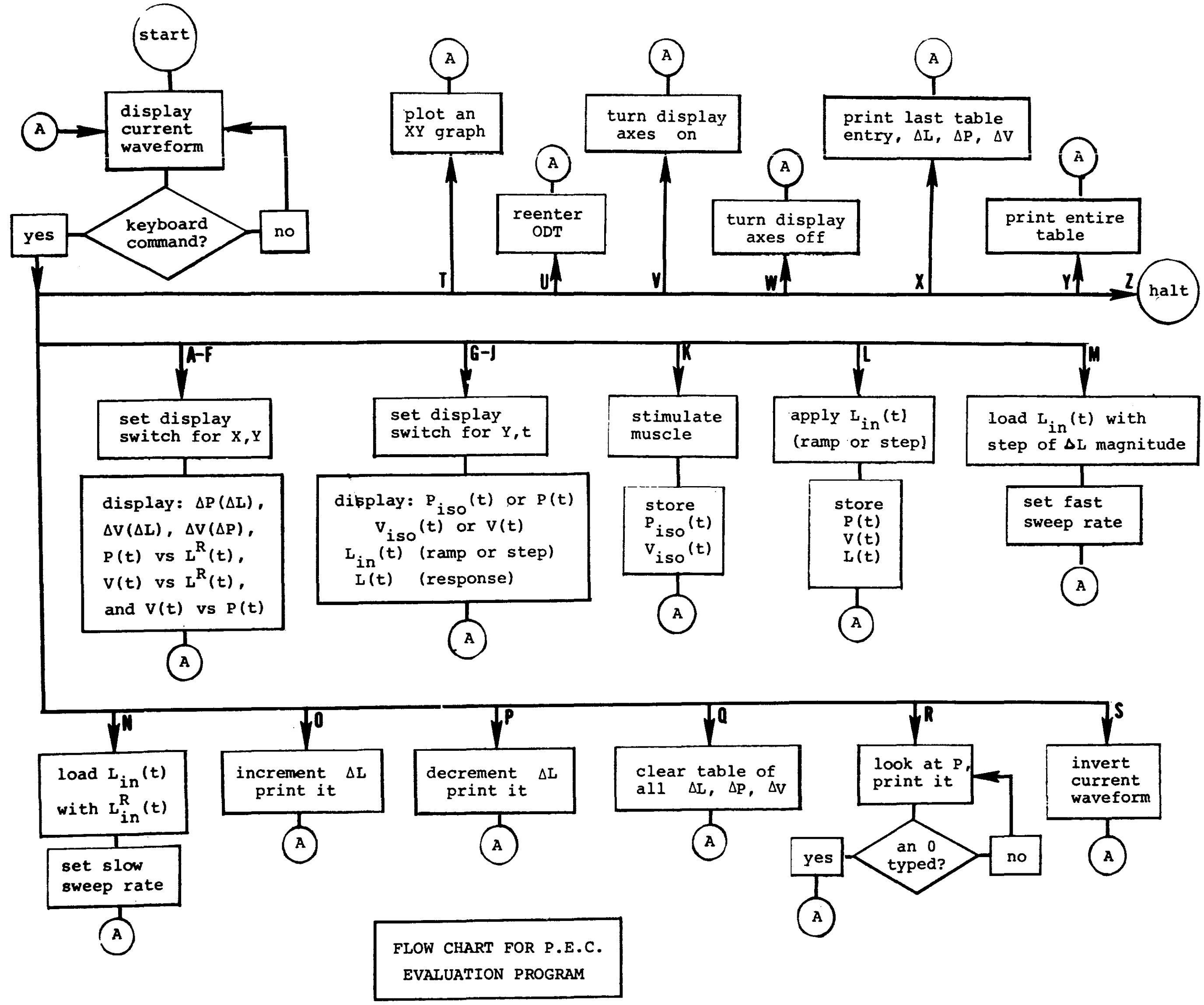
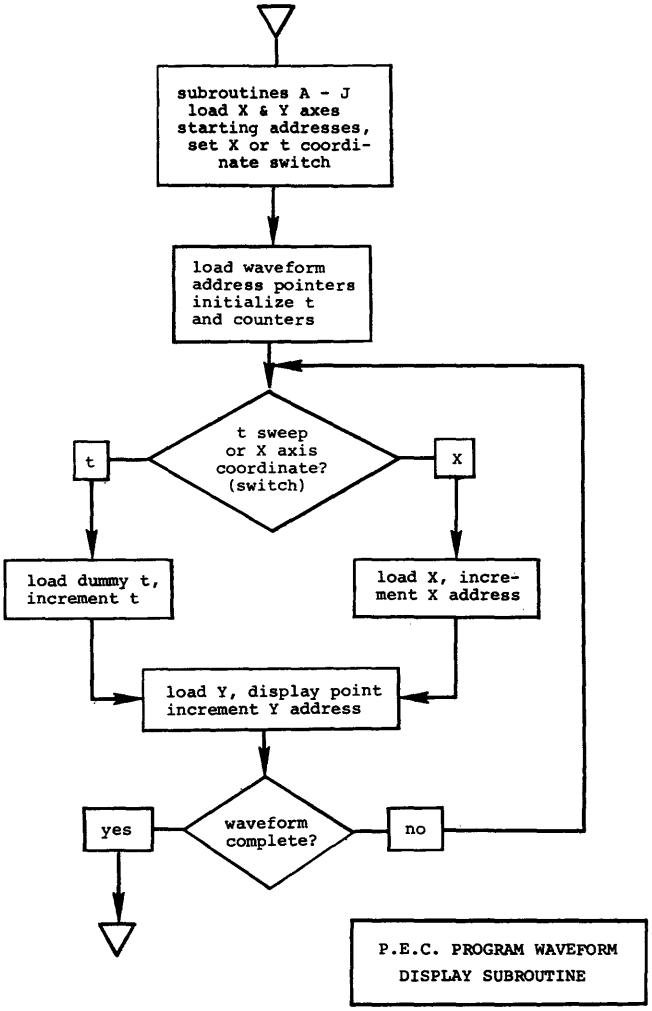

start
A
display current waveform
keyboard command?
yes
no
A
plot an XY graph
A
reenter ODT
A
turn display axes on
A
turn display axes off
A
print last table entry, ΔL, ΔP, ΔV
A
print entire table
T
U
V
W
X
Y
Z
halt
A-F
G-J
K
L
M
set display switch for X,Y
display: ΔP(ΔL), ΔV(ΔL), ΔV(ΔP), P(t) vs $L^R(t)$, V(t) vs $L^R(t)$, and V(t) vs P(t)
A
set display switch for Y,t
dispplay: $P_{iso}(t)$ or P(t) $V_{iso}(t)$ or V(t) $L_{in}(t)$ (ramp or step) L(t) (response)
A
stimulate muscle
store $P_{iso}(t)$ $V_{iso}(t)$
A
apply $L_{in}(t)$ (ramp or step)
store P(t) V(t) L(t)
A
load $L_{in}(t)$ with step of ΔL magnitude
set fast sweep rate
A
N
O
P
Q
R
S
load $L_{in}(t)$ with $L^R_{in}(t)$
set slow sweep rate
A
increment ΔL print it
A
decrement ΔL print it
A
clear table of all ΔL, ΔP, ΔV
A
look at P, print it
an 0 typed?
yes
no
A
invert current waveform
A
FLOW CHART FOR P.E.C. EVALUATION PROGRAM

Figure 3 Flow chart for PEC evaluation program


subroutines A - J load X & Y axes starting addresses, set X or t coordinate switch
load waveform address pointers initialize t and counters
t sweep or X axis coordinate? (switch)
t
X
load dummy t, increment t
load X, increment X address
load Y, display point increment Y address
waveform complete?
yes
no
P.E.C. PROGRAM WAVEFORM DISPLAY SUBROUTINE

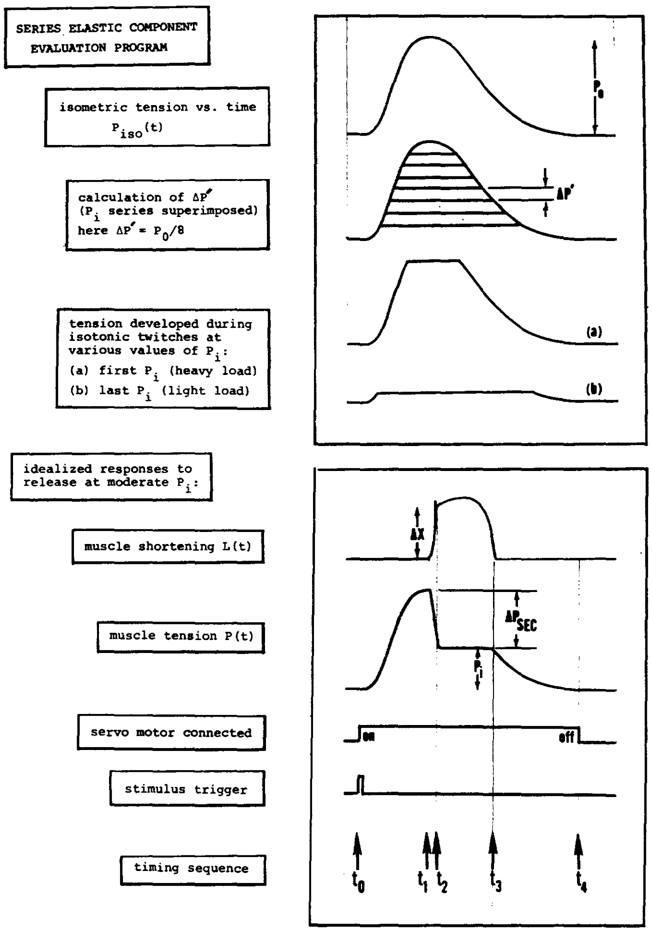

SERIES ELASTIC COMPONENT EVALUATION PROGRAM
isometric tension vs. time $P_{iso}(t)$
$P_0$
calculation of $\Delta P'$ ($P_i$ series superimposed) here $\Delta P' = P_0/8$
$\Delta P'$
tension developed during isotonic twitches at various values of $P_i$:
(a) first $P_i$ (heavy load)
(b) last $P_i$ (light load)
(a)
(b)
idealized responses to release at moderate $P_i$:
muscle shortening L(t)
$\Delta X$
muscle tension P(t)
$\Delta P_{SEC}$
$P_i$
servo motor connected
on
off
stimulus trigger
timing sequence
$t_0$
$t_1$
$t_2$
$t_3$
$t_4$

Figure 5 Illustrating steps of the SEC evaluation technique

Figure 4 PEC program waveform display subroutine

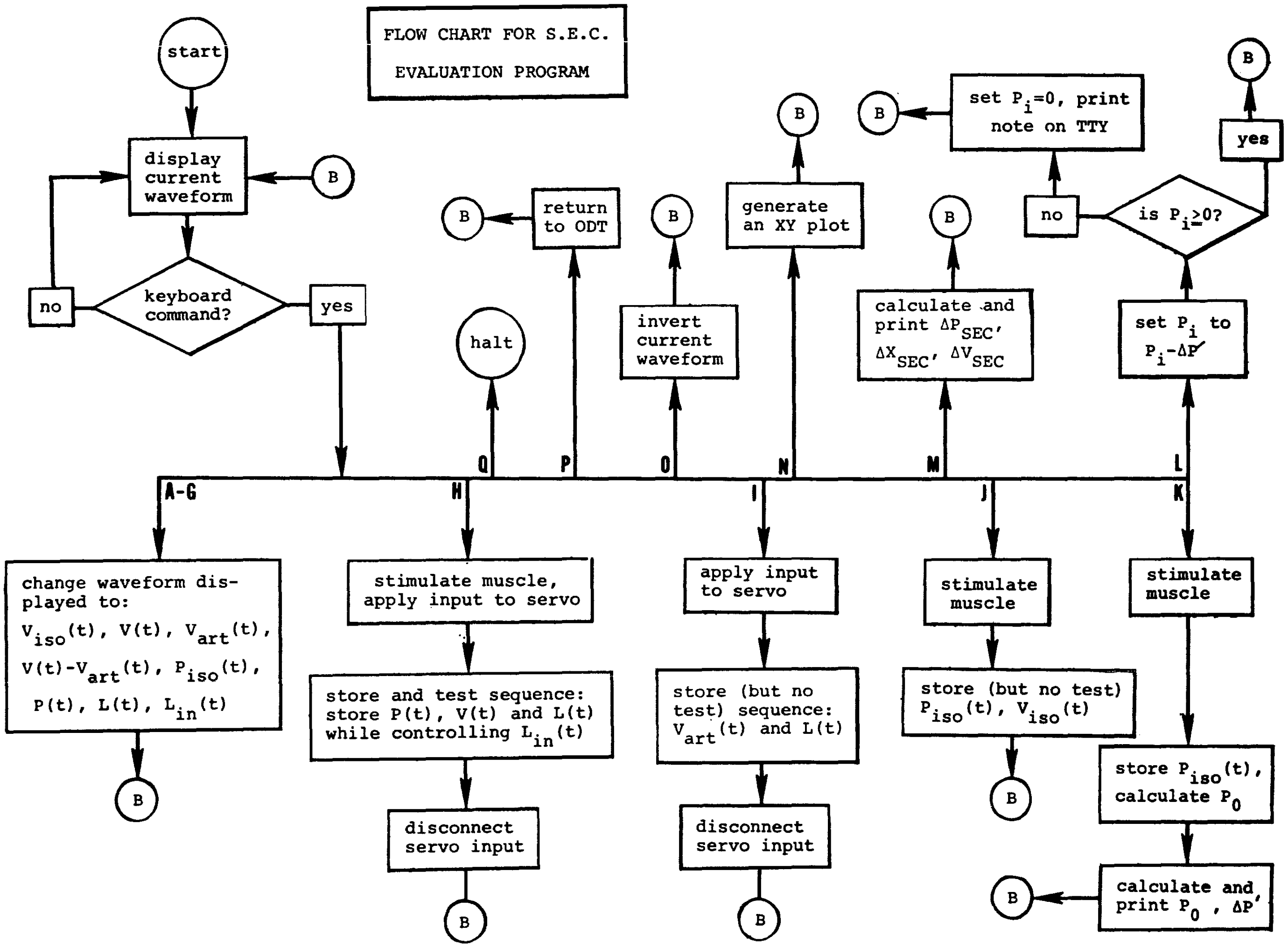

FLOW CHART FOR S.E.C. EVALUATION PROGRAM
start
display current waveform
B
keyboard command?
no
yes
A-G
H
I
J
K
L
M
N
O
P
Q
halt
return to ODT
invert current waveform
generate an XY plot
calculate and print $\Delta P_{SEC}$, $\Delta X_{SEC}$, $\Delta V_{SEC}$
set $P_i$ to $P_i-\Delta P'$
is $P_i \geq 0$?
yes
no
set $P_i=0$, print note on TTY
change waveform displayed to: $V_{iso}(t)$, $V(t)$, $V_{art}(t)$, $V(t)-V_{art}(t)$, $P_{iso}(t)$, $P(t)$, $L(t)$, $L_{in}(t)$
stimulate muscle, apply input to servo
store and test sequence: store $P(t)$, $V(t)$ and $L(t)$ while controlling $L_{in}(t)$
disconnect servo input
apply input to servo
store (but no test) sequence: $V_{art}(t)$ and $L(t)$
disconnect servo input
stimulate muscle
store (but no test) $P_{iso}(t)$, $V_{iso}(t)$
stimulate muscle
store $P_{iso}(t)$, calculate $P_0$
calculate and print $P_0$ , $\Delta P'$

Figure 6 Flow chart for SEC evaluation program

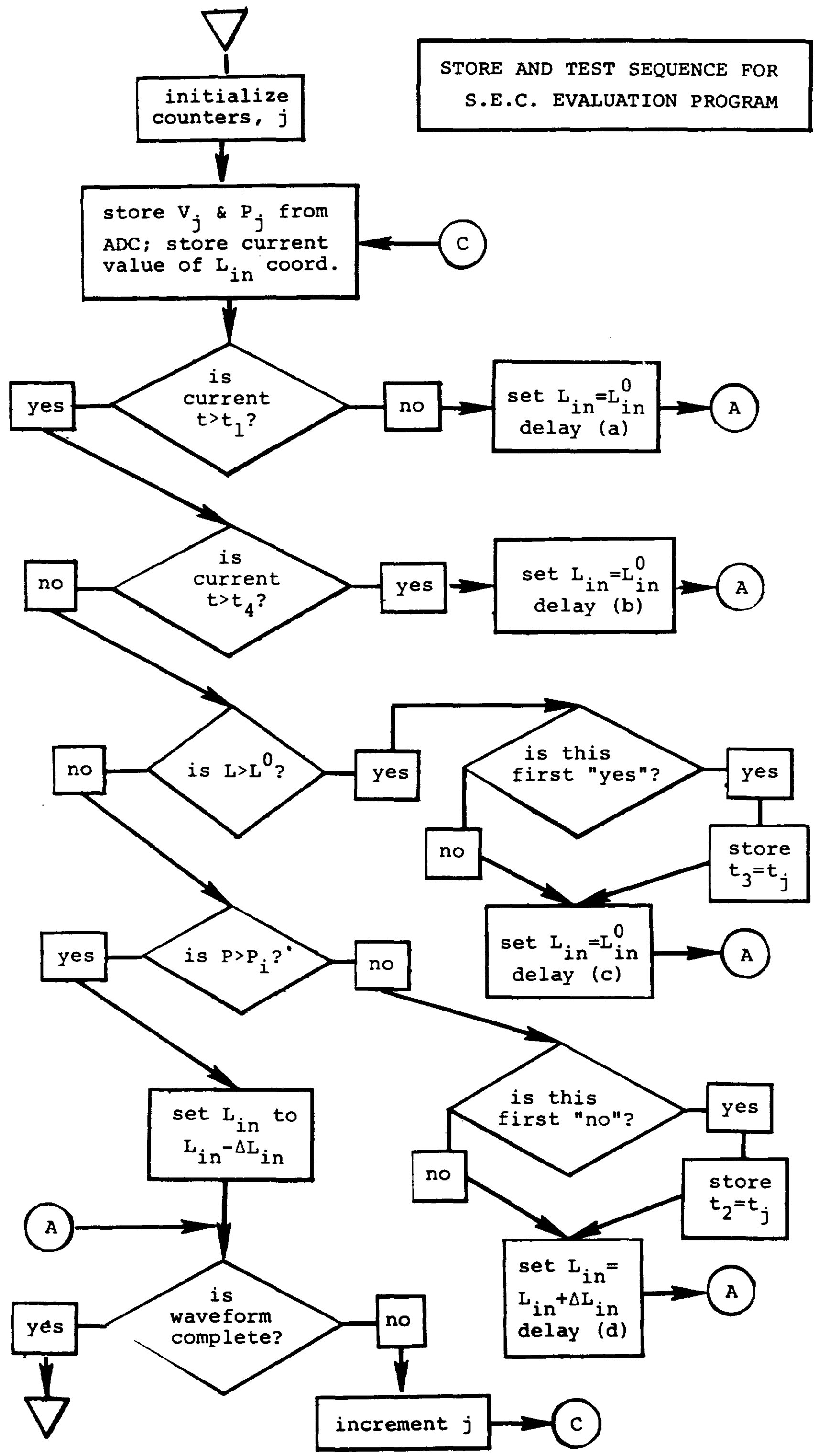

STORE AND TEST SEQUENCE FOR S.E.C. EVALUATION PROGRAM
initialize counters, j
store $V_j$ & $P_j$ from ADC; store current value of $L_{in}$ coord.
C
is current $t>t_1$?
yes
no
set $L_{in}=L^0_{in}$ delay (a)
A
is current $t>t_4$?
no
yes
set $L_{in}=L^0_{in}$ delay (b)
A
is $L>L^0$?
no
yes
is this first "yes"?
yes
no
store $t_3=t_j$
set $L_{in}=L^0_{in}$ delay (c)
A
is $P>P_i$?
yes
no
is this first "no"?
yes
no
store $t_2=t_j$
set $L_{in}$ to $L_{in}-\Delta L_{in}$
set $L_{in}=$ $L_{in}+\Delta L_{in}$ delay (d)
A
A
is waveform complete?
yes
no
increment j
C

Figure 7   Store and test sequence for SEC evaluation program

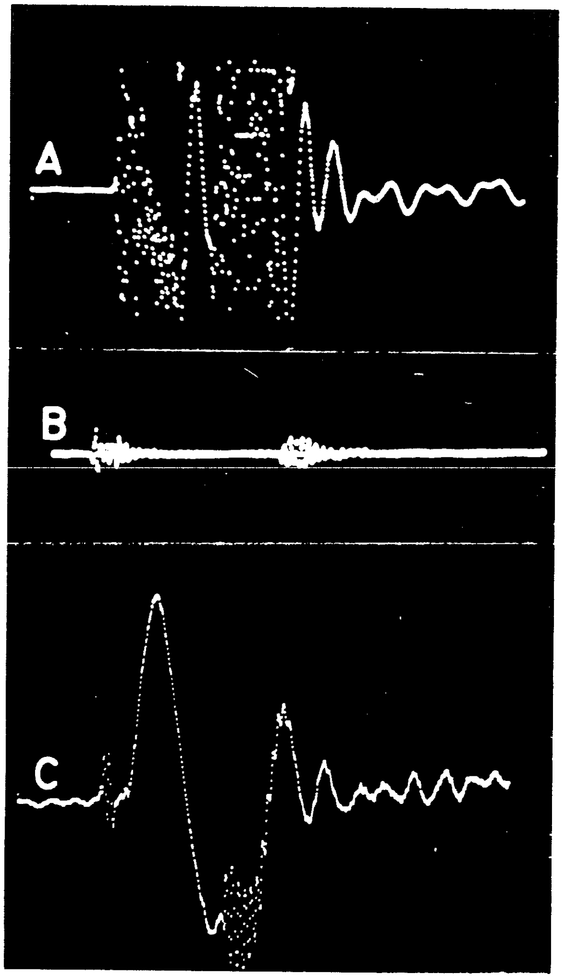


Figure 8

Volume artifact subtraction.
(A) artifact (B) artifact minus artifact
(c) signal minus artifact

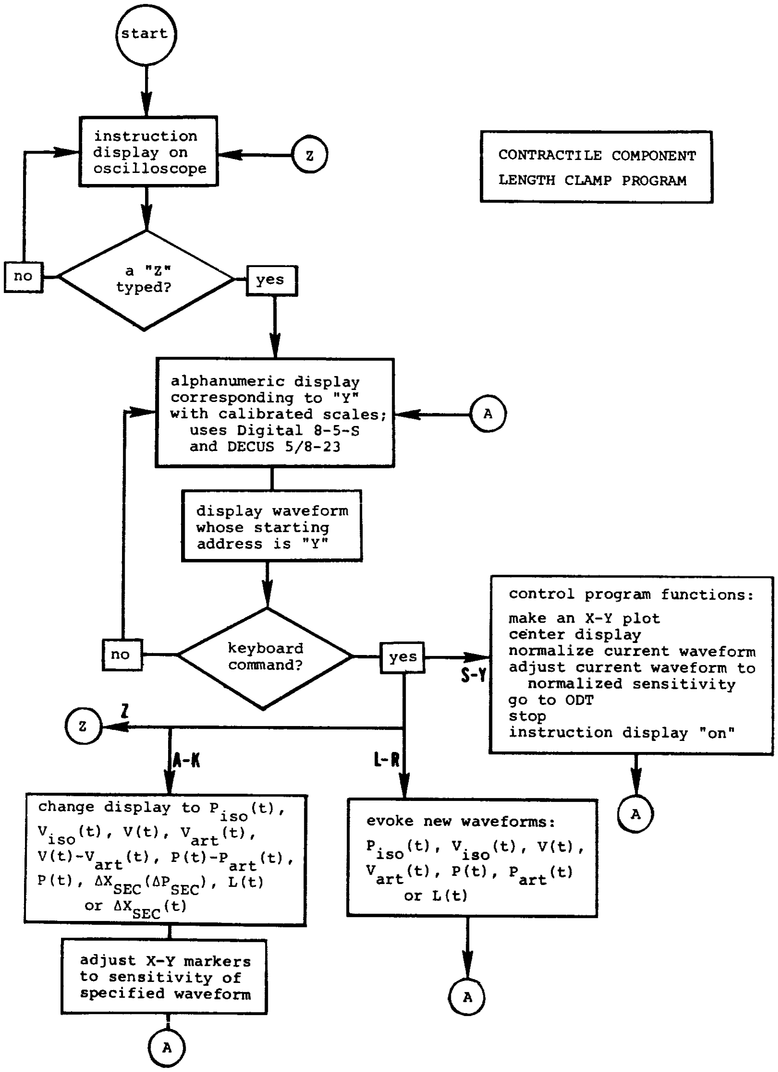

start
instruction display on oscilloscope
Z
CONTRACTILE COMPONENT LENGTH CLAMP PROGRAM
no
a "Z" typed?
yes
alphanumeric display corresponding to "Y" with calibrated scales; uses Digital 8-5-S and DECUS 5/8-23
A
display waveform whose starting address is "Y"
no
keyboard command?
yes
S-Y
control program functions:
make an X-Y plot
center display
normalize current waveform
adjust current waveform to normalized sensitivity
go to ODT
stop
instruction display "on"
A
Z
Z
A-K
L-R
change display to $P_{iso}(t)$, $V_{iso}(t)$, $V(t)$, $V_{art}(t)$, $V(t)-V_{art}(t)$, $P(t)-P_{art}(t)$, $P(t)$, $\Delta X_{SEC}(\Delta P_{SEC})$, $L(t)$ or $\Delta X_{SEC}(t)$
evoke new waveforms:
$P_{iso}(t)$, $V_{iso}(t)$, $V(t)$, $V_{art}(t)$, $P(t)$, $P_{art}(t)$ or $L(t)$
adjust X-Y markers to sensitivity of specified waveform
A
A

Figure 9 CC Length Clamp program

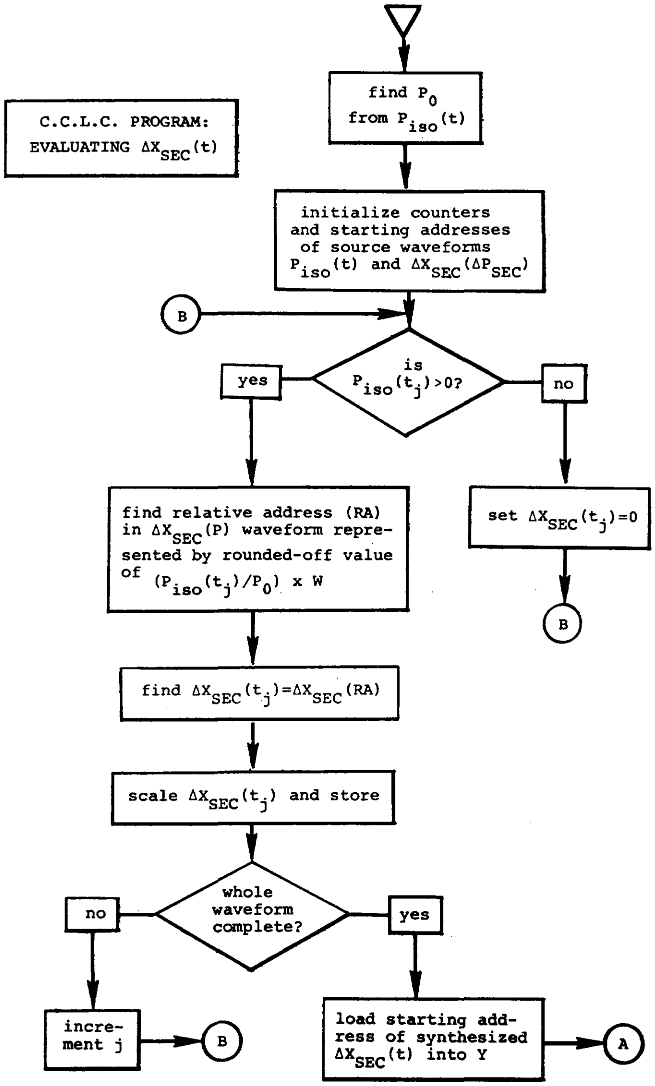

C.C.L.C. PROGRAM:
EVALUATING $\Delta X_{SEC}(t)$
find $P_0$ from $P_{iso}(t)$
initialize counters and starting addresses of source waveforms $P_{iso}(t)$ and $\Delta X_{SEC}(\Delta P_{SEC})$
B
is $P_{iso}(t_j)>0$?
yes
no
find relative address (RA) in $\Delta X_{SEC}(P)$ waveform represented by rounded-off value of $(P_{iso}(t_j)/P_0) \times W$
set $\Delta X_{SEC}(t_j)=0$
B
find $\Delta X_{SEC}(t_j)=\Delta X_{SEC}(RA)$
scale $\Delta X_{SEC}(t_j)$ and store
whole waveform complete?
no
yes
increment j
B
load starting address of synthesized $\Delta X_{SEC}(t)$ into Y
A

Figure 10 CCLC program: evaluating $\Delta X_{SEC}(t)$

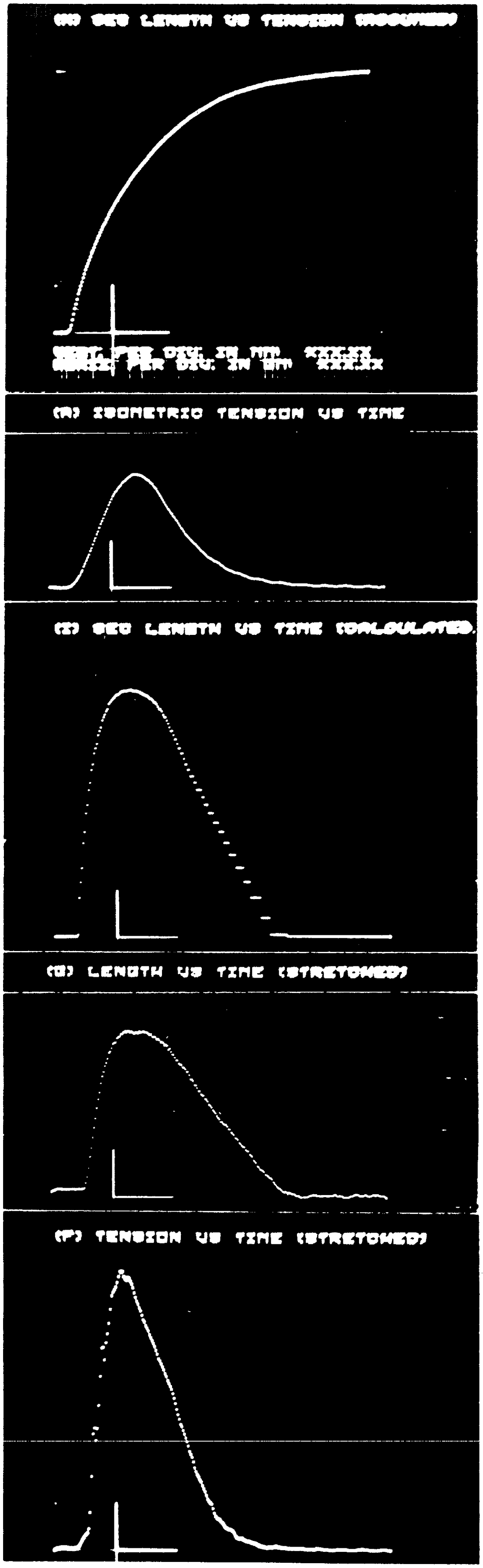


Figure 11  Sample data output display from CCLC program

# FLIRT: FILE LANGUAGE IN REAL TIME

Kenneth R. Morin
Department of Pathology, University of British Columbia
and St. Paul's Hospital
Vancouver, British Columbia

## ABSTRACT

FLIRT, an intermediate-level language which directs file transactions between teletype, core and DECtape is being developed for a clinical laboratory application using a PDP-9 computer. A FLIRT file may contain any number of records; each record contains a number of items. An item can be 12-bit binary, variable-length alphanumeric, or it can be another record (nesting limited to 4-deep). Each file is stored in as many blocks on DECtape as necessary.

FLIRT contains about 25 verbs, e.g., ASK (ask question on teletype and store response in core), MOVE (move item(s) from one record to another in core), WRNFL (write a new file on DECtape), and LOCREC (locate a record on DECtape which meets the specified conditions). Four verbs direct movements to/from a queue-buffer area. Record formats and most mnemonics are user-defined. The file structure, FLIRT language, and some aspects of the internal software are described.

## 1. BACKGROUND

A file-handling scheme is a major requirement in a data-processing system for a clinical laboratory. Many such systems use a dedicated computer, usually having a 12- to 18-bit word, 8K to 24K word core memory, 250K to 1000K word disc memory, analog multiplexer and A/D converter, several teletypes or other input terminals, and perhaps a low-speed line printer and one or more magnetic tape units (DECtape and/or IBM-compatible tape). Few if any computers of this size offer suitable file-handling software such as COBOL. For the laboratory computer system at St. Paul's Hospital an 8K PDP-9 is being used and FLIRT, an intermediate-level file-handling language, is being developed. The present version of FLIRT stores files on DECtape, and we plan a second version which will store files on disc as well.

## 2. FILE STRUCTURE

A general file structure was chosen which can be tailored easily to the user's specific requirements. The largest entity in FLIRT is a file, which consists of one or more records; each record consists of one or more items; each item consists of one or more characters. The 6-bit character is the smallest unit of data.

The FLIRT file structure and language are sufficiently general to be useful for many applications besides hospital laboratory systems.

### 2.1 Files

Any number of files can be created, each being of a specific type. For example, laboratory data will be stored in a separate file for each patient, producing between 500 and 1000 Patient Files (PATF), and test data will be stored in a separate Test File (TESTF) for each test batch. Each file of a given type may contain any number of records. The mnemonic names, chosen by the user, can be up to 6 characters long.

### 2.2 Records

Each record type must be defined by giving the record name, and a list of the items which the record will contain. For example, Patient Identification records (PATIDR) will include such items as Patient Name (PATNAM), Patient Number (PATNO), and Birth year (BRTHYR).

### 2.3 Items

Each item in a record can be either:

- a variable-length alphanumeric item;
- a 12-bit binary item; or
- another record.

The latter produces a nested record structure which allows many types of data to be stored. This nesting has been arbitrarily limited to four levels.

### 2.4 Characters

All three types of items are composed of 6-bit bytes or characters, the smallest unit of data manipulated in FLIRT.

### 2.5 File Numbers

Each file is uniquely identified by its type and number. When a file type is defined, the name of the item to be taken as the file number must be specified. A code for the file type is automatically stored in a special location in the file. The file number must appear in every record in the file. For example the patient number will be taken as the file number in our patient files, so all records in a patient file must contain the patient's number (at nesting level 1).

## 3. THE FLIRT LANGUAGE

There are about 25 verbs in the FLIRT language at present. Some verbs require no arguments, and others require as many as eight. Some arguments may assume different forms; e.g., the file number for the search verb can be specified by either a literal, the name of a pointer which points to the file number, or the mnemonic "ALL".

### 3.1 The Queue

We expect file transactions to be required at an average rate of one every 5 to 10 seconds, and the maximum rate may greatly exceed this. DECtape has a maximum access time of about 30 seconds. To overcome this disparity, requests for DECtape transactions are entered into a queue; a search routine continously cycles the DECtape from end to end unless the queue is empty, checking every file it encounters against the list of requests, and executing all matching requests. FLIRT verbs are used to queue and to delete the transaction data.

### 3.2 An Example

The example shown in Fig. 1 will introduce the reader to the FLIRT language. This is the admitting section of a demonstration program; several questions are asked on the teletype, and the responses are stored in a new file on DECtape.

```
/***DEMON*** DEMONSTRATION PROGRAM USING FLIRT.
/DEMON ALLOWS THE SELECTION AND EXECUTION OF
/SIMPLE DEMONSTRATION ADMITTING, RETRIEVAL,
/AND DELETION PROGRAMS.
/NUMERICAL ARGUMENTS ARE IN OCTAL.

        RECORD 76,REC

PRSEL   ASKX <"PROGS: 1-ADM 2-RETR 3-DEL SELECT?">,3
        JMP ADM       /ADMITTING
        JMP RETR      /RETRIEVAL
        JMP DEL       /DELETION

/**ADMITTING**
ADM     OPNREC REC,PATIDR
ASK1    ASK <"1 NAME    ">,PATNAM,REC,25
        ASK <"2 HOSP NO.">,PATNO,REC,6
        ASK <"3 ROOM-BED">,ROOM,REC,6
        ASK <"4 COMPLAINT">,ADMDIA,REC,60
        EDIT ASK1

        QREC REC
        QINS
        WRNFL PATF,REC
        DELREC REC
        DELINS
        ASKYN <"MORE? ">,ADM,PRSEL
```

FIG. 1 EXAMPLE OF A FLIRT PROGRAM.

The RECORD assignment reserves $76_8$ locations for a record, with the tag REC. The ASKX verb asks the question in quotation marks, accepts a response X from 1 to 3, and jumps to the Xth location following; thus if the response is 1, the JMP ADM instruction is executed and the admitting program is entered.

The OPNREC verb opens a blank PATIDR record at REC, to provide a place to store the admitting information. The first ASK verb asks the question in quotation marks on the teletype, accepts a response up to $25_8$ characters long, and stores the response as the PATNAM item in the record at REC. After the last ASK, the EDIT verb is executed; it types EDIT> on the teletype, and any of the preceding questions can be repeated by typing in the line number (1 to 4). Typing only a carriage return causes the program to proceed.

The QREC verb transfers the record at location REC into the queue, thus freeing the area at REC for furthur input. The QINS verb queues an instruction group containing the verbs which follow QINS, up to and including the first DELINS. At the same time a request (section 3.1) is automatically queued in accordance with the first verb in the group. In the example the request is for a blank block of tape, since the WRNFL verb will write a new PATF file containing the record which was queued from location REC. After the QINS has been executed, the ASKYN verb is executed, while the DECtape search program (section 3.1) seeks to satisfy the request. If the operator's response to the question MORE? is Y for yes, the program branches to ADM to admit another patient; if N for no, the program branches to PRSEL; meanwhile the DECtape search program runs concurrently (time-interlaced).

When the search program has located a blank block of DECtape, the WRNFL verb in the queued instruction group is executed: the record which was queued from REC is transferred to an output buffer, the necessary file information is added, and the output transfer is initiated. Next, the DELREC verb deletes the record from the queue, and finally the DELINS verb deletes the instruction group (and request) from the queue, which will now be empty unless furthur data has been added.

### 3.3 Reference to Queued Records

After a record is queued, the instruction group which defines the required file transaction is queued. Since these instructions specify the record to be processed by giving the location from which the record was queued, ambiguity might arise if the queue contained several records which had been queued from the same location. This ambiguity is avoided by the QINS verb which, when executed, replaces each reference the instruction group contains to a queued record with an internally-assigned label which refers to the record most recently queued from the specified location.

### 3.4 Pointers

Several coordinates are required to identify an item which is nested several levels deep within a record. Pointers are used for this purpose. Space for a pointer (3 words) is allocated by the POINTR TAG verb. The pointer at TAG can be set to identify an item P which is nested in the record at RECSA by using the POSPNT TAG, RECSA, ELT1, ELT2, ELT3, ELT4 verb. The item R (another record) at nesting level 1 which contains P is specified by ELT1. The item within the record R and which contains P is specified by ELT2, and so on.

### 3.5 Summary of FLIRT verbs

Figure 2 is a list of FLIRT verbs. APTFL appends a record to a file; the file number is implicit in the record contents (section 2.5). DELFL deletes the specified file. MOVE moves the SELT item in the record at SRECSA to the DELT item in the record at

DRECSA (the SELT is unchanged); these arguments can take several forms, such as simple mnemonics, pointers, and "CORESP".

```
RECORD SIZE,TAG
POINTR TAG
QREC RECSA
DELREC RECSA
QINS
DELINS
CLOSE
EXIT
WRNFL FLTYP,RECSA
APTFL FLTYP,RECSA
DELFL FLTYP,FLN
POSPNT TAG,RECSA,ELT1,ELT2,ELT3,ELT4
LOCREC FLTYP,FLNO,RECTYP,NLEVEL,ITEM,FN,SIZE1,SIZE2
MOVE SELT,SRECSA,DELT,DRECSA
COMPAR ITEMA,FN,ITEMB,ITEMC
INSREC RECSA,WHERE
OPNREC DRECSA,RECTYP
OPMATR DRECSA,N1,N2
PUTCR DRECSA
OVRLAY SRECSA
ASK <"QUESTION">,DELT,DRECSA,NCHARS
ASKX <"QUESTION">,N
ASKYN <"QUESTION">,YADR,NADR
EDIT TAG
PRINT RECSA
```

FIG. 2 SUMMARY OF FLIRT VERBS.

LOCREC is the tape search verb, and requests that a file of type FLTYP and number FLNO be found, containing a record of type RECTYP at nesting level NLEVEL, in which the contents of the ITEM item obeys the FN relationship (greater than, equal, etc.) to SIZE1 (and SIZE2 if FN = between or outside). The return is with the accumulator positive if successful, and negative if unsuccessful. The record so found becomes the current record (CUREC). If it is modified by a MOVE, it must be re-written with the CLOSE verb. The whole current record may be copied into core with the PUTCR verb, or overlaid by another record with the OVRLAY verb. A record may be inserted into the file in front of or following the CUREC with the INSREC verb; this allows records to be stored in sequence in a file.

A one or two-level "matrix" record may be opened with the OPMATR verb. In a matrix record, items are referenced by their numerical coordinates rather than by user-assigned mnemonics. The PRINT verb, as tentatively defined, will print the contents of a 2-level record in a row-column array; this could be used for printing patient summary reports.

The EXIT verb is used instead of DELINS to exit from a queued instruction group without deleting it; e.g., after a LOCREC with the FLNO = "ALL", an EXIT would leave the instruction group in the queue and thus the next file which meets the specified conditions would be found. The COMPAR verb is similar to the comparison function in the LOCREC verb, with the result indicated by the contents of the accumulator.

Several verbs to do floating-point arithmetic have been defined tentatively. As FLIRT evolves, some of the verbs defined above may be deleted and others may be added.

## 4. INTERNAL STRUCTURE

Several aspects of the internal structure of FLIRT files and the subroutines which execute FLIRT verbs are discussed below.

### 4.1 Program Size

A FLIRT program is assembled with the PDP-9 macro assembler, which expands the FLIRT verbs and arguments into subroutine calls. The subroutines which execute the FLIRT program will probably occupy about 8K words when all the verbs have been implemented. We hope to use program-swapping between core and disc to reduce the resident core requirement to about 3K.

### 4.2 File Storage on DECtape

There is no limit to the length of a FLIRT file: each file is stored in as many chained blocks of DECtape as are necessary. We plan to increase the present 64-word block size and perhaps allow the block size to be defined by the user. A record cannot be greater than one block in length.

### 4.3 Index and Keys

In the present DECtape version of FLIRT, no index is used, and the number of the block in which a given file starts bears no relationship to the file type or file number. An index or a "most likely starting address" computed from the file number may be used in the disc version of FLIRT.

### 4.4 Internal File Delimiters

Each record begins with a three-character header "(HH" and ends with a ")". HH is the code for the record type. Items in a record are separated either by a colon, which precedes a binary item, or by a semi-colon, which precedes an alphanumeric or a record item. The characters ( ) ; and : cannot be used as data. The following example shows what a file containing a patient identification record and a test result record might look like.

```
(HH;DOE,J;35681;211-1;UREMIA)(HH;35681;3JAN;327
 ;(HH;ELECT;(HH;123;3.5;82;20));(HH;BUN;69)))
```

In a record of given type, the items are always stored in the order they were listed in the record definition. Specific items are located according to their sequence within the record.

### 4.5 Queue Manipulations

When records or instruction groups are queued, they are added to the end of the queue; when they are deleted, the remainder of the queue is moved up to fill the gap. If all records and instruction groups were the same size a list of free space (more efficient) could be used instead.

When a MOVE verb will alter a queued record, that record is moved to the end of the queue, so a change in the size of the record will not affect the remainder of the queue.

## 5. ACKNOWLEDGEMENT

The author is grateful to Dr. R. H. Pearce who played a large role in the design of FLIRT, and who is providing a continuing role in its evaluation; and to Dr. J. S. MacDonald, Dr. C. K. Harris, Miss M. L. Gillies, Mr. H. J. Bridges, and the staff of the laboratory at St. Paul's Hospital, who provided many of the initial guidelines. We are grateful to the Mr. & Mrs. P. A. Woodward Foundation, the British Columbia Hospital Insurance Service, St. Paul's Hospital, Vancouver, and the Departments of Pathology and Electrical Engineering of the University of British Columbia, who are jointly sponsoring the development of this project.

# MODULES/HARDWARE

# RECENT EXPERIENCES IN DESIGNING MODULAR INTERFACES USING M-SERIES LOGIC[†]

Gary B. Morgan
National Reactor Testing Station
Idaho Nuclear Corporation
Idaho Falls, Idaho

## ABSTRACT

The use of M-series ($T^2L$ integrated circuit logic) modules for interfacing to computers which are constructed using B and R series modules involves some new considerations. This paper will deal with three major areas of concern: (1) the economics of using M-series vs standard R-series, (2) system design of I/O using $T^2L$ logic for interfacing on-line control and data acquisition experiments, and (3) specific problems and solutions when using the M-series line for an interface (including special cards which have been designed). A specific data acquisition and control interface for a PDP-9 will be used as an example of the three general areas above.

## INTERFACING SYSTEM CONCEPTS

The task of interfacing external instruments, experiments, controllers, etc. to small general purpose computers is one which should be approached from an over-all systems standpoint. The idea of adding one interface after another without considering the overall system development will undoubtedly lead to poor results when the system grows beyond the first few small interfaces. Three major areas of consideration are discussed below.

### System Modularity

The users of these general purpose digital computers for data acquisition and control of experiments must have the ability to change the system configuration and alter its capabilities easily and without a great amount of manpower effort or wasted component costs. To satisfy this demand the system must be designed in a modular fashion. It must grow or change within this modular environment. Modules must be designed such that the system does not depend on any one of them to be installed or working for the rest of the system to function correctly.

### Programability vs Versatility

The design of all parts of the interface which interact with a program (the program controlled input/output) must be given careful attention from a software (program) point of view. Ideally the person designing such a system should be a competent programmer as well as a logic designer. There should also be a constant interchange of ideas and/or concepts between the three types of people involved in the project (logic designer, programmer, experimental user). The modular components of the system should be designed such that the program can determine unattended the status of the module (is the module on or off line, etc.). Whenever possible the hardware design should be such that the interface is as versatile as possible, allowing the program and programmer or user to exercise their options on how the device will be run. This means that the same piece of hardware may be used for several different experiments without major future modifications. The hardware may be used by different programs in completely different operating environments (interruptable real-time vs non-interruptable program controlled).

### Maintainability

By designing the interfaces with an over-all system modular concept in mind, it is possible to greatly reduce the down time of the total system due to minor hardware failures or experimental changes. An obvious, but often overlooked, area is making sure that when a module is unplugged from the system (cable is disconnected for off-line maintenance or because of a failure) the open circuited logic inputs don't tie vital signal lines in the processor to the wrong state (the interrupt bus or skip bus enabled).

A second point is the cable configuration of the system. If every cable in the system is made in some standard configuration, then when a module is disconnected for changes or maintenance it can easily be connected to some general kind of simulation jig or mock-up device to enable bench check-out. The simulator jig could be as simple as a set of switches to simulate a register output or a set of indicator lights to simulate a register input.

The systems concepts discussed in the preceeding section are illustrated by the following discussion of a time-of-flight interface to a PDP-9. Figure 1 shows a block diagram of the complete PDP-9 I/O system as designed. The devices on the left side are shown daisy-chained to the PDP-9 I/O bus in the normal manner. The high-speed paper tape equipment could be a device 1 and 2 for example. Device 3, as shown in Figure 1, is a large interface which will control several remote (100-500 feet) experimental interfaces. Device 3 (IC common logic) changes the PDP-9 bus appearance from the display chain to a fanout where each remote interface is completely modular. All lines to and from the IC common logic (Device 3A,

[†] Work performed under the auspices of the U. S. Atomic Energy Commission.

3B, . . . 3N) are driven and received by active gates in such a way as to allow any separate remote device to fail or to be disconnected without affecting the rest of the PDP-9 system. All the logic in device 3 (IC common and remote interfaces 3A, 3B, . . .) with the exception of one set of I/O bus level converters is implemented using TTL integrated circuits. Figure 2 shows the time-of-flight interface in more detail. The interface is capable of measuring the time between nuclear events from up to 32 different detectors. At each event, the time (from some relative $t_o$) is stored in a buffer register 1 (13 bits). A 5-bit tag word indicating in which detector the event occurred is stored along with the event time. This makes up the full 18-bit word which is dropped down through the 5 buffer registers and into the PDP-9 memory through the DMA 09 multiplexer. The dead time of the front end is less than 200 nsec. Since there are 5 buffer registers in front of the DMA 09, as many as 5 random events may be accumulated in one microsecond before a hardware overflow will occur. If any buffer register has data in it a DMA request is given to the PDP-9 processor.

All initial set-up parameters (channel width, timed initial delay, list location and size, etc.) are under program control. The remote time-of-flight chassis also has a set of switches and indicators on its front panel to allow the experimenter to communicate with the PDP-9 program (these are very useful in debugging and maintenance as well as during routine operation). Figure 3 shows the remote time-of-flight interface front panel and chassis unit. The word count and current address logic for the DMA channel is located in the PDP-9 cabinet since its logic is not necessarily associated with a time-of-flight experiment. An indicator panel is located at the top of the cabinet in the PDP-9 logic and is shown in Figure 4. Note also in Figure 4, there are flag indicators for every interrupt coming into the common logic and power indicators for every experiment hooked to the common logic.

## INTERFACES BUILT USING M-SERIES LOGIC MODULES

### PDP-8 Interfaces

1. A hardware bootstrap loader. This bootstrap will load any twenty-word program into the core of the PDP-8 under pushbutton control.

2. A general data acquisition and control interface for an 8/S using integrated circuits (scalers, counters, clock, DAC and ADC).[1]

3. Data acquisition and control interface for an 8/S to control a beta-ray spectrometer using integrated circuits.[2]

### PDP-9 Interfaces

1. Remote modular interface for 3-axis spectrometer on the MTR reactor.

2. Remote modular interface for measuring the time-of-flight information on the MTR reactor experiments.

3. General purpose control switch panel for inputing information to the processor.

4. 10-key function box.

### Logic Circuit Considerations

The interfaces listed above were constructed using M-series logic modules. This logic was chosen for two reasons: (1) The $T^2L$ logic is superior to DTL logic. It has a wider system bandwidth, its noise susceptibility is better and its drive capability is higher. This reduced problems and cost in driving long wire runs and cables. (2) At the time we started design (August 1967) the M-series line was the least expensive $T^2L$ module line on the market.

A number of cards were designed and built "in house" to supplement the DEC M-series line. Since then DEC has added these functions to their standard M-series inventory.

We find that the reliability of the $T^2L$ integrated circuits is very high (better than R-series) and the logic is easy to use. The power distribution system for $T^2L$ logic must be very low inductance and well bypassed to keep system noise to a minimum. Reasonable care at the time of construction is sufficient.

All interfaces designed for DEC machines (-3 volts machines included) should be designed using $T^2L$ integrated circuits. The time-of-flight interface described earlier was priced out at $3200 using M-series logic. The same functional unit would cost $5300 using B and R series modules.

[1] This interface was designed and built by Donald Staples.

[2] This interface was designed by Paul Hofhine (Phillips Petroleum Company employee).

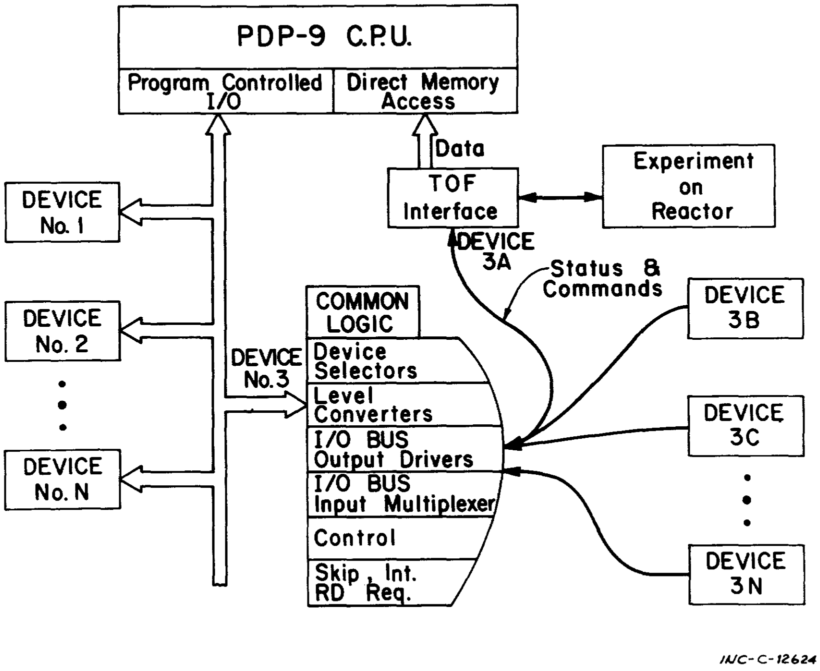


Figure 1. PDP-9 Input/Output System.

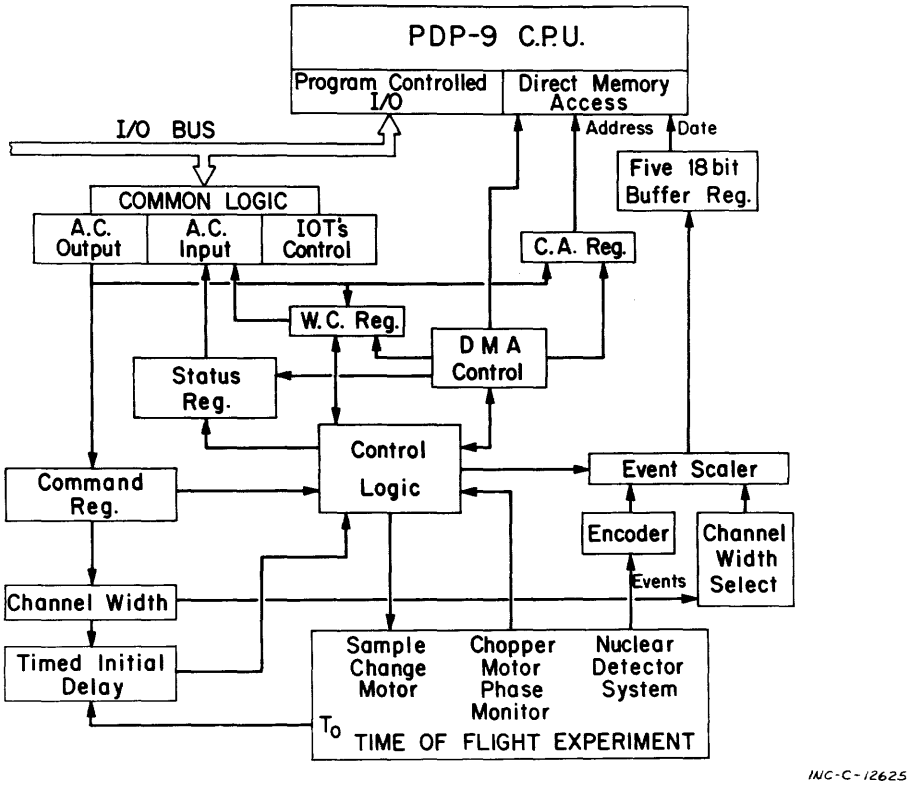


Figure 2. Time-of-Flight Interface Diagram.

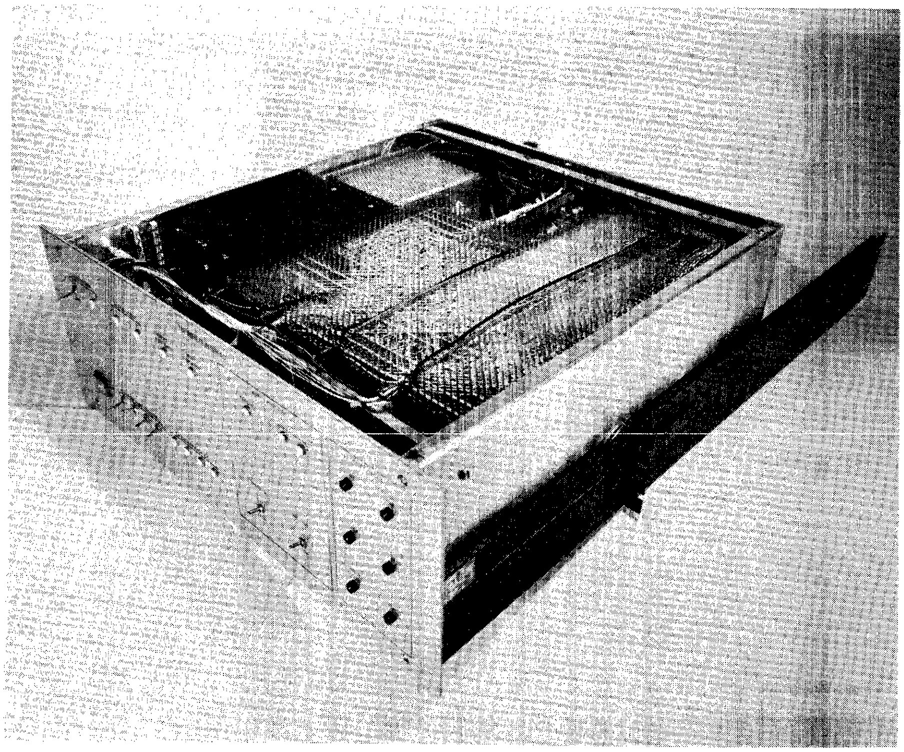


Figure 3. Time-of-Flight Chassis.

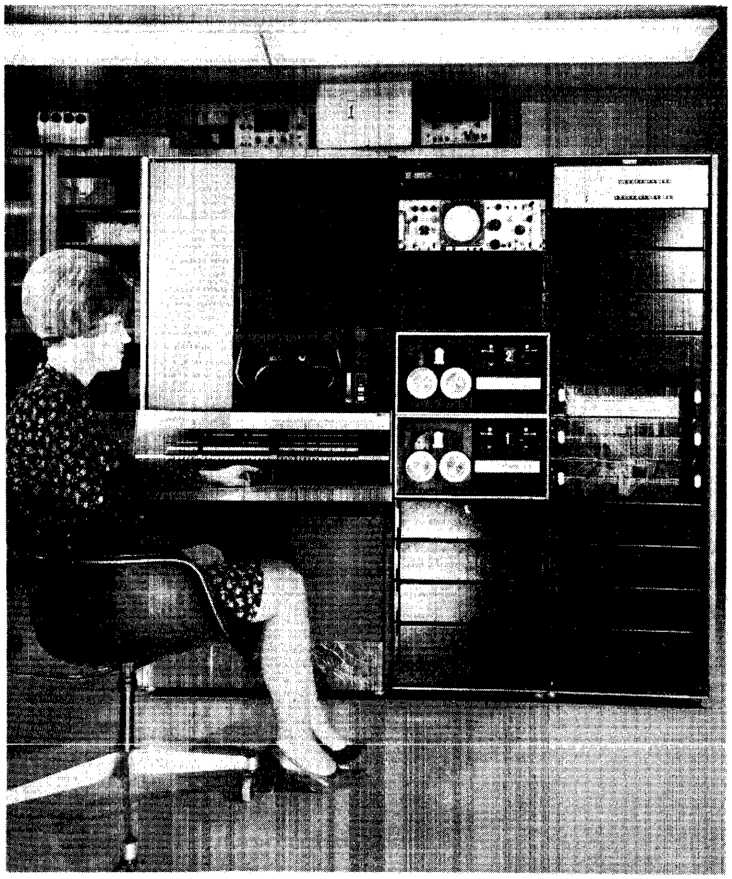


Figure 4. PDP-9 and Common Logic.

# A DATA COMMUNICATION SYSTEM FOR THE PDP-8 IN ARBUS*

Thomas G. Taussig
Lawrence Radiation Laboratory
University of California
Berkeley, California

## ABSTRACT

A data communication interface developed for ARBUS (Advanced Reservation and Bed Utilization System--paper to be presented by R. Abbott, Biomedical Session) will be described. The low cost, high speed data break interface allows the connection of multiple duplex Teletype stations and data phones to the PDP-8. The design of this system results in low program overhead requiring less than 5 percent of computation time to assemble characters from 32 lines, sampling at 4 times the 110 baud rate. The system requires 2 words of memory per line.

## Introduction

The Advanced Reservation and Bed Utilization System (ARBUS), consists of a PDP-8 System operating in conjunction with a commercial time-sharing facility. The PDP-8 system is located within the hospital and continuously executes the ARBUS system program to provide local control functions. Connection is established as required to the Tymshare facility via the Data Set and commercial telephone lines on a "dial-up" basis under control of the PDP-8 program.

The PDP-8 system consists of various remote input/output devices, a special interface to allow the devices to communicate with the computer, and the PDP-8 computer. The devices that were to be required by the system are; Model 33 Teletype units, Invac modified Selectric typewriters, Type 103A Data Set, Type 801A Data Auxiliary Set (Automatic Calling Unit), and a real-time clock derived from 60 cycle ac power.

## Interface Description

A block diagram of the ARBUS system is shown in Figure 1. The Computer Interface Control is basically a software controlled multiplexer with serial to parallel assembly and parallel to serial disassembly capability. Extensive use is made of PDP-8 control logic and core memory by use of the Data Break facility in a modified form. All devices are connected to the ARBUS Computer Interface Control, and send or receive data, bit serially, at Teletype rate of 110 baud. As an exception the Real Time Clock sends a 60 Hz square wave that is used to synchronize the programs to real time. The ARBUS Interface may be expanded to control a maximum of 128 channels. Expansion beyond this number will result in real time problems unless average activity is very low. We have implemented control for 32 channels in the ARBUS prototype, with TTY line circuits for 18 channels. Expansion is a straight-forward procedure.

## Input/Output Devices

Although the channels appear identical to the programs and to the remote devices, it was convenient to make certain channel assignments. Channel 0 input is used for the 60 Hz real time clock. Channel 0 output is unused. Channels 1 and 2 are reserved for the Data Set and the associated Automatic Calling Unit. Channel 1 input and output are used for the data receive and transmit functions of the Data Set. Channel 2 input provides status information for both the Data Set and the Automatic Calling Unit, while Channel 2 output is used for Automatic Calling Unit Control. The remaining channels are used for Teletype or Invac Selectric typewriters.

## Memory Allocations

When operating, the ARBUS Interface associates with each input or output channel two memory locations. One of these locations is called the "channel control word", the other is the "channel data word". The channel control word is loaded by the program and specifies; the number of bits to be shifted into or out of the channel data word, the relative phase of actual input or output of the bit to the initiation of the process, and the status of the channel. For the 32 channel system, the output control words are located in core memory at locations $200_8$ through $237_8$, while the input control words are located at locations $240_8$ through $277_8$ (see Figure 2). The least significant five bits of the memory address of these locations are therefore the channel number. The associated output data words are located at locations $300_8$ through $337_8$ and the input data words at locations $340_8$ through $377_8$. The data word address for a particular channel is therefore always $100_8$ greater than the control word. For a 128 channel system the least significant 7 bits of address would correspond to channel number and the data word address would be $400_8$ greater than the control word address. The four arrays, containing control and address words, would each be 128 words in length or one full page.

* This work was done under the auspices of Contract #PH 108-66-288.

Operation

In explaining the operation of the interface, one might note that the processing of the arrays of control words and data words always begins in synchronism with an internal 440 Hz clock. This clock, when "on", initiates a set of PDP-8 Data Break cycles every 2.273 milliseconds. The clock may be turned on by an IOT Instruction or by the presence of a logical zero or "Space" (the beginning of any character) on any input channel. The first Data Break cycle of the set is associated with the output control word for channel 0 (location $200_8$). If the control word indicates that this channel is active, it is incremented and every fourth time (every 9.09 msec) a bit is shifted out of the output data word (location $300_8$) by a subsequent Data Break cycle. The remaining output and then the input control words are sequentially incremented, if active, and bits shifted out of the associated output data words or into the associated input data words as specified by the control word timing. The data Break cycles taken by the interface alternate with memory cycles taken by the PDP-8 central processor until the last control word is reached. No additional Data Break cycles take place until the next set begins at the output of the 440 Hz clock. During a set of Data Break cycles, any active control word that is incremented to equal zero indicates that the corresponding data word has been fully transmitted or received.

Program Control

The completion of transmission of a character is indicated to the program by the setting of the Input Interrupt Flag or the Output Interrupt Flag in the interface. These flags cause an interrupt if the Interrupt Enable is set. The Interrupt Enable is controlled by the following IOT instructions:

6404 and $AC_0$ = one -- Set the Interrupt Enable.

6404 and $AC_I$ = one -- Reset the Interrupt Enable.

The 6404 instruction also turns on the 440 Hz clock. Two other instructions allow interrogation of the Output and Input Flags:

6401 -- Reset Output Flag and skip if
        flag is ON.
6402 -- Reset Input Flag and skip if flag
        is ON.

These are the only IOT Instructions used by the interface.

Control Word (See Figure 3)

The Control Word is composed of four logically distinct segments: A the Active bit, E the Enable bit, NB the Number of Bits to be shifted, and T the Timing Bits.

Bit 0 of the Control Word is the A or Active bit. If this bit is on, the channel and the associated line is actively transmitting or receiving data. It is set by the program to initiate data output but is set by the interface logic for Input Control Words after the line associated with the control word becomes active. The control word is incremented during Data Break cycles if this bit is set. Bit 1 of the Control Word is the E or Enable bit. For input control words only, the Enable bit enables the setting of the Active bit during the associated Data Break cycle when a Start Space (a logical zero) is detected on the line. If the Enable bit is a zero the associated line is ignored and thus disabled by the program. Bit 1 is always set for output control words. Bits 2 through 9 provide control for the Number of Bits (NB) to be input or output. The number of bits to be input or output is essentially equal to the two's complement of NB. Bits 10 and 11 provide the Timing (T) for shifting bits into or out of the data words.

Data Word

Data bits are shifted into or out of data words when T = 3 in the associated control word. For input, for example, the time of sampling and shifting a data bit after the transition of a Start baud is;

$= (3-T)\ 2.273 + 1.136 \pm 1.136$ msec. Data words are shifted one bit right, by the Data Break cycle that occurs after the corresponding Control Word is incremented such that T = 3. This shift takes place every 4 x 2.273 msec = 9.09 msec (corresponding to 110 baud) on active channels. Input data is shifted into bit 0 and information in bit 11 is lost. Output data is shifted to the right, and bit 11 is shifted into the corresponding output line flag. The output line flag is a one bit register that will hold the bit that drives the line until the next bit is to be transmitted.

Summary

The total time overhead of the interface is quite low. For an output character, 11 bits are shifted onto an output line using 5 Data Break cycles per bit. The full character is output in 100 msec using a total of 55 Data Break cycles. The overhead for outputting is approximately 83 microseconds out of every 100 milliseconds or .083% per line. For input only 10 bits are shifted in at 5 Data Break cycles per bit. The input overhead is then 75 microseconds out of every 100 milliseconds or .075% per line. For a 128 channel system, the total interface overhead would be about 20.2% of computer time available.

This system represents an effort to design a low cost, synchronous communications interface utilizing a minimum of wired logic and a maximum of the power available in the logic of the PDP-8 computer.

Acknowledgements

I would like to give credit to Joe Strople of Data Systems Design for contributions to the design and to Dave Pehrson of Lawrence Radiation Laboratory, Livermore for system checkout and documentation, and members of the Research Data Facility of the Institute of Medical Sciences in San Francisco for their support.

Bit No.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | E | NB | | | | | | | | T | |

A = Active bit. If this bit is on, the line is actively transmitting data.

E = Enable bit. The active bit will automatically be set on, for an input control word, if this bit is on and the corresponding input line is zero.

NB = Number of bits. This field determines the number of bits to be input or output.

For input:
Number of bits input = two's complement of NB

For Output:
Number of bits output = two's complement of NB if $T \neq 3$
two's complement of NB-1 if $T = 3$

T = Timing. These bits determine the time at which bits are shifted into or out of the data words.

For Input:
Time to sampling after Start baud
$= (3-T)2.273 + 1.136 \pm 1.136$, msec

For Output:
Time to output after control word initialization
$= (2-T)2.273 + 1.136 \pm 1.136$, msec, if $T \neq 3$
$7.955 \pm 1.136$, msec if $T = 3$

CONTROL WORD FORMAT

FIGURE 3

XBL 691-104

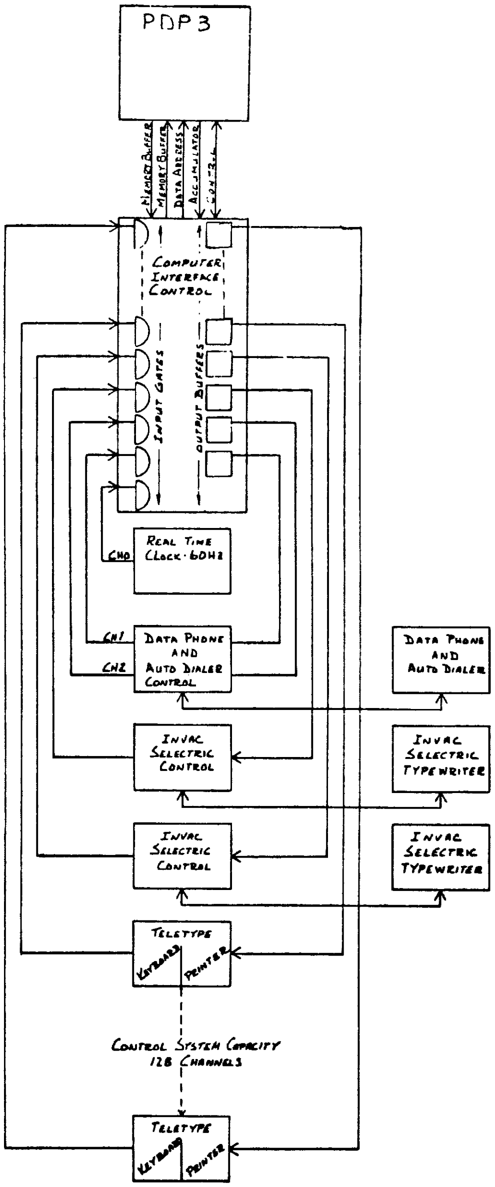


FIGURE 1

SYSTEM DIAGRAM

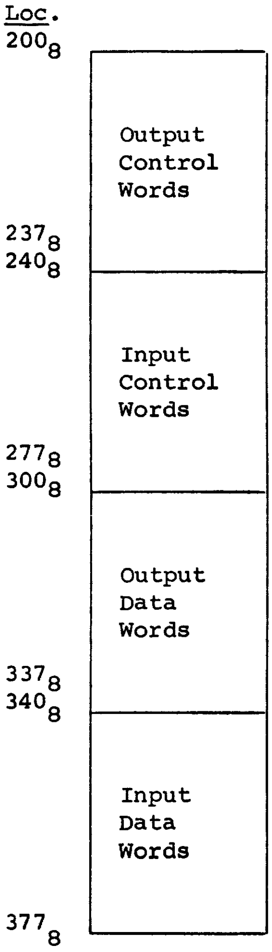


FIGURE 2

MEMORY ALLOCATION

# PDP-8
# WORKSHOP

# A DISC ORIENTED REAL-TIME EXECUTIVE FOR THE PDP-8/S COMPUTER

W. T. Lyon
Development Engineer
Aluminum Company of America
Pittsburgh, Pennsylvania

## ABSTRACT

An executive was written for the PDP-8/S Computer to allow it to be used in a real-time environment for process control.

In its present form, it allows 12 levels of program priority, with a maximum of 115 programs running at the various levels within the priority structure. It decodes 24 process interrupts and provides 24 software timers.

Although the system was tailored to a particular process, it can be restructured by reassembly.

## INTRODUCTION

Real-time control of a process with a computer presents a variety of problems to the programmer.

Many movements or events can occur concurrently in the process, demanding the computer's attention at unpredictable times. In addition, the computer's own peripherals must be scheduled in an efficient manner to gain the greatest throughput from them.

Because of the complexity of this scheduling problem, a real-time operating system was written to satisfy the demands of all the interactive programs that would go into the process control environment.

The system was tailored to a particular process application, but was written to enable it to be easily modified for other applications.

The requirements to be placed on the computer were as follows:

(1) Control material handling equipment associated with a machining process.

(2) Provide scheduling information to the truckers who supply the process.

(3) Keep a complete inventory of all material within the process.

(4) Classify and identify the pieces prior to their leaving the process.

(5) Provide communications with the operator to alert him of alarm conditions or to provide him with information about the process upon demand.

## HARDWARE

To satisfy the demands of the process, the following hardware was selected:

(1) PDP-8/S Processor with 4K of core memory.
(2) DF32 Disc File
(3) CR02 High Speed Tape Reader
(4) CR03 Card Reader
(5) 132 Digital Outputs for Control
(6) 180 " " " Displays
(7) 180 " Inputs
(8) 24 process interrupts including 1 - 10 Hz timer
(9) 1 RO33 Teletype Machine
(10) 1 ASR33 Teletype Machine

## SOFTWARE

The executive was written in modular form and consists of five parts:

(1) Scheduler
(2) Interrupt Procesoor
(3) Disc Driver
(4) Time Delay Program
(5) Keyboard Monitor

### Scheduler

The executive centers around the Scheduler, which determines the order in which programs will be executed. All programs communicate either directly or indirectly with the Scheduler. As can be seen in Figure 1, the only exception to this rule is the fast response interrupt programs and the portion of the Disc Driver that processes a disc interrupt.

The Scheduler was intended to be as flexible as possible. It provides up to 12 separate levels of priority. Any number of programs can be run at sublevels of a priority level. Programs can be either core resident or disc resident, but must all be the same at any one priority level. All of these options are fixed at the time the system is assembled by the way the system tables are structured.

Figure 1 illustrates the flow of control. Every program that runs is called for by the Scheduler. When the program terminates, it exits to the Scheduler. If the program is momentarily suspended by a disc call or a time delay, control is returned to the Scheduler via the Disc Driver or the Time Delay Program.

Programs are requested to run by setting a bit in a bidding matrix. The Scheduler scans this matrix periodically to see if any programs have been bid for. If so, the highest priority program is run.

### Interrupt Processor

The Interrupt Processor runs whenever a program is interrupted. It saves the contents of the accumulator, linc and program counter, and then checks to see if the interrupt was caused by the disc. If so, it goes to the Disc Driver. The Disc Driver returns control to the Interrupt Processor which then proceeds to decode other interrupts that may have occurred and to take appropriate action. The "appropriate action" falls into three categories:

(1) Run a short program immediately (a "fast response program").

(2) Bid for the Time Delay Program (when the timer interrupts.)

(3) Bid for a process program.

After processing the interrupts and taking action, control is returned to the Scheduler.

### Disc Driver

The Disc Driver is necessary to provide an orderly flow of information between core and disc memory. All disc transfers, whether they be requested by the Scheduler, Keyboard Monitor, or a process program, are implemented by the Disc Driver.

When a process program requests a disc transfer, its priority level is suspended until the transfer is complete. If the disc is busy when the request is made, the request is queued up until the disc becomes free. The queue is processed according to priority rather than on a first-come basis.

Completion of a transfer is sensed when a disc interrupt is received. When this occurs, the Disc Driver bids to continue the suspended program. At this time, it also initiates the next transfer if one is being requested.

### Time Delay Program

The Time Delay Program runs as a process program under the Operating System. It is bid for by the Interrupt Processor when a timer interrupt occurs and runs at the highest priority level, level 1. This program allows other process programs to momentarily suspend their own operation or to request that another program run after some time has elapsed. It also keeps a real-time clock.

### Keyboard Monitor

The Keyboard Monitor program provides the ability for the programmer to communicate with the computer from the Teletype keyboard. It runs as a process program under the Operating System at the lowest level of priority, level 12. Any time there are no process programs to run, the Keyboard Monitor is scanning the keyboard waiting for input. This allows one to interrogate or modify portions of computer memory without interrupting the control of the process.

The functions available with the Keyboard Monitor are as follows:

(1) DC xxxx yyyy (Dump core in octal from location xxxx to location yyyy).

(2) LC xxxx (Load core starting at location xxxx with the octal numbers to be input on the keyboard).

(3) BP xxxx yyyy (Binary Punch from core from location xxxx to location yyyy).

(4) BL (Load a Binary tape into core).

(5) DD xxxx (Dump 2 blocks from disc in octal starting with block xxxx).

(6) LD xxxx yyyy (Modify block xxxx on disc starting with relative address yyyy).

(7) CD (Call DEC Disc System).

## TIME AND SPACE REQUIREMENTS

The executive was written for a mini-disc based system, but can be used for an all core resident system. It requires a minimum of $2100_8$ words of core memory including half of page zero.

Included in the $2100_8$ words are several subroutines, which can be used by any program as long as it turns interrupt off before going to the subroutine and turns it back on after returning. These subroutines include the following:

(1) Exclusive Or

(2) Inclusive Or

(3) Single Precision Multiply

(4) Single Precision Divide

(5) Single Precision Arithmetic Right Shift

(6) Single Precision Arithmetic Left Shift

Also included in the 2100 words are the tables which contain all the parameters needed to define the way the Operating System is structured. These tables vary in length with the maximum number of programs, interrupts and time delays, but occupy a minimum of $110_8$ locations.

In addition to the $2100_8$ locations required for the executive, $400_8$ words are required for the Keyboard Monitor.

The original version of this system occupies a total of $3200_8$ locations. It allows for up to 115 separate programs, 24 process interrupts and 24 software timers.

The Scheduler is the largest time user. It requires between 2.1 and 12.7 milliseconds to execute. The time of execution is measured from the time the Scheduler is entered until it exits to a program. It varies with the priority of the first bidding program that is found. To determine that a program is bidding to run would require the Scheduler 2.1 milliseconds if the program is assigned to level 1 and 12.7 milliseconds if the program is assigned to level 12.

The Interrupt Processor requires about 5 milliseconds to execute. This means that the fastest recognition rate for interrupts approaches 200 interrupts per second.

## INTEGRATION WITH THE DEC DISC SYSTEM

To be able to use PALD, Disc Editor and other options of the DEC Disc System, it was necessary to devise a scheme to integrate the DEC system and the Alcoa system on the same Disc.

The approach taken was to save the Alcoa system as a "user" program under the DEC system. Then, dummy entries were stored into the DATA NAME FILE and SAM table to reserve some space on disc for the process programs to reside.

One of the options of the Keyboard Monitor calls the DEC System and gives control to the DEC Monitor. (This function will operate only if the computer is in the off-line mode). The Alcoa System, in turn, can be called by name with the DEC Monitor.

## CONCLUSION

A small computer like the PDP-8/S, can be used for process control. Even though it is slow, there is no reason why it should be confined to overseeing one task at a time. The process that this computer is to direct is not yet in operation, but all preliminary tests indicate that when it does go on line it will be another example of the process industry moving away from the "overkill" approach to computer control.

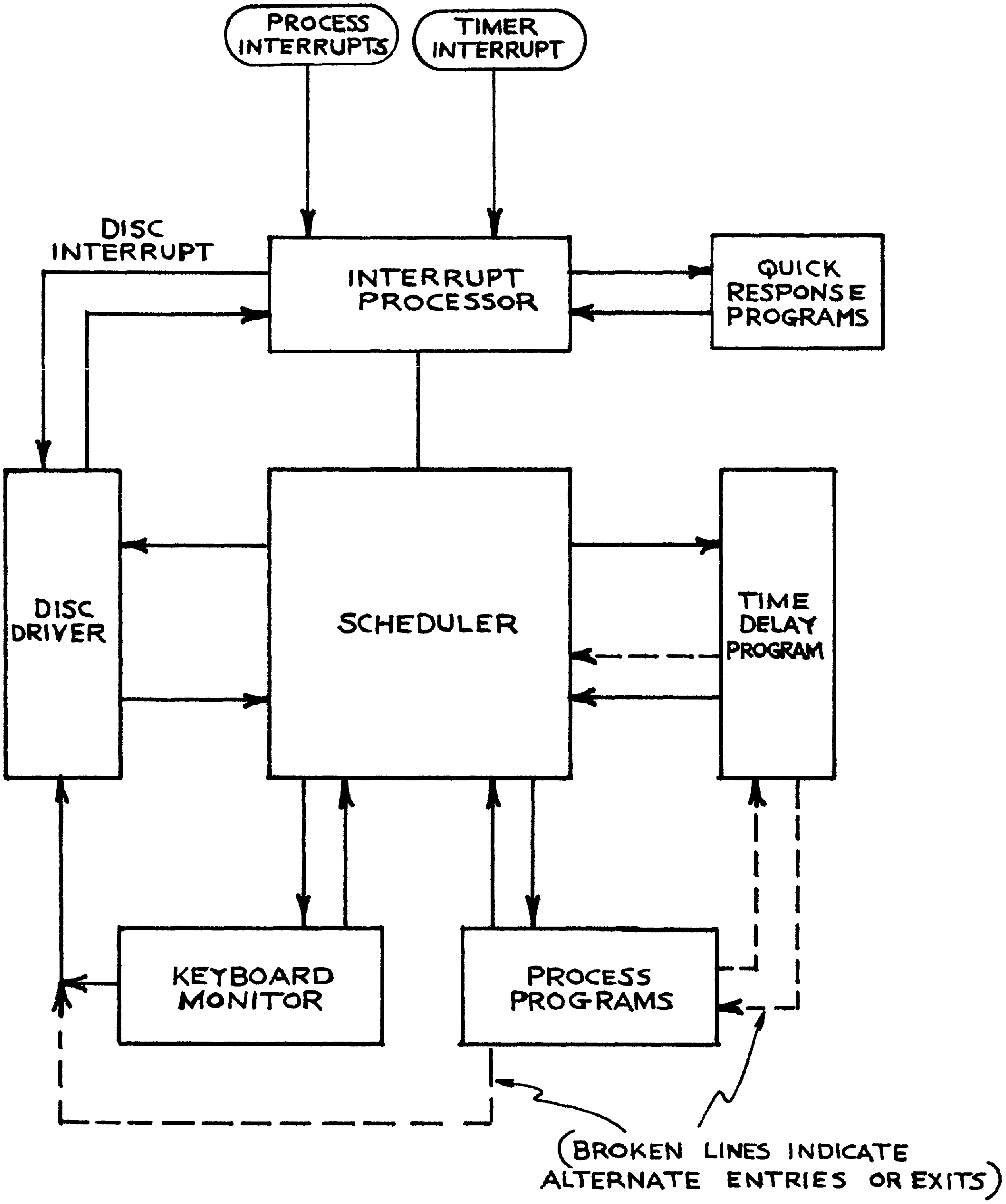


Figure 1 System Architecture

# TMF - A PROCEDURAL SOFTWARE PACKAGE FOR BIOMEDICAL AND SCIENTIFIC APPLICATION ON THE PDP-8

Fred R. Sias, Jr. and Allen B. Wilson
Division of Neurology, Department of Medicine
University of Mississippi Medical Center
Jackson, Mississippi 39216

## ABSTRACT

TMF is a procedural software package including an Input-Output Control System (IOCS-8) for use with the PAL III or MACRO-8 assembly systems on the PDP-8 series of machines. A number of definitions are added to the permanent symbol table of each assembler to permit execution of procedural subroutines analogous to Fortran statements such as IF, GOTO, DO, READ, WRITE, CALL, and RETURN. A keyboard monitor is included in the basic system.

## INTRODUCTION

Programmers using higher-level compilers such as Fortran find the procedural capabilities of the language at least as useful as the arithmetical features. While floating point arithmetic software packages are usually provided by computer manufacturers to ease the programming effort when using assembly languages, analogous procedural packages are usually not available. TMF is a first effort at developing such a procedural software system.

## WHAT'S IN TMF?

The basic requirements for a procedural software package were suggested by the procedural statements we take so much for granted in FORTRAN. We first implemented IF, Computed GOTO, DO loop and two-dimensional array handling routines. Then it became apparent that a useful software package should include an input-output control system and some means of formatting input-output records. After adding some device handlers we had a package of programs that look much like the operating system used with an interpretive FORTRAN such as PDP-8 Fortran. However, programs are prepared using either the PAL III or Macro-8 assembler. A number of definitions are added to the symbol table of either assembler to permit execution of procedural subroutines analogous to Fortran statements. The nature of the assemblers are such that several statements may be strung together in a manner reminescent of a Fortran statement.

In addition to analogs of the basic procedural Fortran statements, we have incorporated a keyboard monitor system, CALL and RETURN statements for handling subroutines, as well as necessary overlays so that TMF will function with the PDP-8 Floating Point Interpreter and Input-Output controller. Using some offerings of DECUS we have provided for automatic round-off and proper handling of integers when output is by the DEC Floating Point output controller. Finally, TMF includes a linking or chaining feature for segmenting programs.

The Table in the appendix shows the core allocation for the basic TMF package. Note that only four extra pages of coding are required beyond the space required by the floating point interpreter. If we compare this operating system with the analogous Basic 4K PDP-8 Fortran operating system we find that TMF leaves about 3200 octal locations for user programs while PDP-8 Fortran leaves only 2400 octal locations. In addition, TMF user programs tend to be shorter and optional subroutines such as the extended functions can be deleted from TMF when not needed. PDP-8 Fortran does not offer this option.

## STRUCTURE OF TMF

The structure of TMF is determined by five factors:

1. The structure of PAL III Macro-8 Assembler.
2. The structure of the Floating Point Interpreter.
3. The requirements of device independent of I-O handling.
4. Operation on data in the real Accumulator.
5. Operation on data in the Floating accumulator.

Of course, shifting data back and forth between the real and floating accumulators is handled by pseudo instructions FIX and FLOAT. We chose to define all subroutine jumps on page zero rather than entering the procedural package through an interpreter. This is faster than using an interpreter and is not too restricting since the keyboard monitor and CALL table is located on page one rather than on page zero.

All input-output functions are handled by IOCS-8, a one page device independent control system. It may be used independently of the rest of TMF by entering the device number in location DEVICE on page zero and entering the subroutine with data to be outputted in the real accumulator. A (decimal) 12 slot device assignment table contains the addresses of appropriate device handlers. Also included in IOCS-8, is a text handler and a subroutine for chaining program segments. Input-output formatting is handled by another segment of the TMF package.

Fortran-like input-output statements are interpreted by a one page set of subroutines entered by the pseudo instructions READ or WRITE. To be compatible with PAL III, the following form may be used:

```
WRITE           /WRITE JMS I IRITE
4               /DEVICE NUMBER
306             /ASCII CODE FOR F FORMAT
6               /LIKE F6.2
2               /     IN FORTRAN
```

Using Macro-8 the following format produces the same object code and is easier to read:

```
WRITE; 4; "F; 6; 2 (carriage return)
```

This is the first example of what might be called a TMF language except that its really assembly language.

WRITE is defined as a JMS to a write handling subroutine. 4 is the device number and is the first parameter provided to the subroutine. "F is assembled as 306 and is the second parameter delivered to the subroutine. "F; 6; 2 then is the Format description. This certainly isn't Fortran but a knowledge of Fortran provides the understanding needed to prepare programs using TMF. The fact that PAL III and Macro-8 interpret a semicolon like a carriage return enables one to string out assembly language coding so that it looks like a higher language and provides the capabilities of a higher language.

READ statements have the same structure. A,B,E, F, and I formats are the same as the analogous Fortran formats. C format outputs a carriage return and line feed. A and B formats are via the real accumulator while E, F, and I formats utilize the floating point accumulator. Decoding is handled by these routines while actual input-output is handled by IOCS-8. The subroutines that decode the E, F, and I formats set control words on page zero that then utilize the floating point I-O controller for actual format control.

Another page of coding is used to interpret several Fortran-like procedural pseudo-instructions. These include IF, computed GOTO, FIX, FLOAT, DO loops with unlimited nesting, and an ARRAY pseudo code to handle two-dimensional arrays.

```
IF; ADDRI; ADDR2; ADDR3
```

causes a JMS to a subroutine that examines the contents of the real accumulator and causes a jump to one of the three addresses based on whether the accumulator is negative, zero or positive.

A statement with the form

```
GOTO; ADDR1; ADDR2; etc.
```

is interpreted as the standard computed GOTO based on the integer contained in the real accumulator. The unconditional GOTO was not implemented since the standard JMP instruction is adequate.

The following format is used to set up a DO loop:

```
DO; N; J, Ø
(Instructions in do loop)

CONT; J
```

As implemented now, J must be set to the starting value before entering the DO loop and CONT increments J N-times following each pass through the loop. Since independently defined indexes are used any nesting depth is possible.

Arrays are frequently handled in DO loops. To facilitate this, the following statement is used:

```
ARRAY; NAME; INDEX1; INDEX2
```

Where the indices are defined in a DO loop or elsewhere and INDEX1 and INDEX2 are addresses of the index values. The array must be defined in memory as follows:

```
NAME, A          /NO. OF WORDS PER ELEMENT
      B          /NO. OF ROWS
      C          /NO. OF COLUMNS
      (REST OF ARRAY)
```

The subroutine is designed to handle two-dimensional arrays. If a one-dimensional array is specified, INDEX2 is not required, however, the number of columns in array definition must be set to zero. Execution of the ARRAY subroutine leaves the address of the selected array element in POINT on page zero. Of course, data is deposited or retrieved from the array by indirect instructions.

Finally, the basic TMF package includes coding for a keyboard monitor and routines for locating subroutines named by single ASCII characters. For example:

```
CALL; "D    or   CALL; 304
```

causes a JMP to the address stored at location 304. The return to the mainline program is executed by the pseudo code RETURN via an indirect address stored on page zero. Locations corresponding to the ASCII code for letters of the alphabet are available to name subroutines. In addition, the same table of addresses is used by the keyboard monitor, so subroutines must not have the same "names" as programs to be accessed by the keyboard monitor. Separate address tables could be used but the letters of the alphabet should provide plenty of names for both subroutines and keyboard called programs. The statement CALL; "K is used to return to the keyboard monitor.

The appendix defines the complete structure of TMF.

## CONCLUSION

A procedural Software Package, TMF, has been developed to simplify assembly language programming. A set of pseudo-instructions similar in appearance to analogous FORTRAN procedural statements are interpreted by the basic TMF package. Combined with the floating point interpreter the system is both faster and more compact than the PDP-8 Fortran Operating System. Since we only appear to be using a higher-level programming language we have the efficiency of assembly language programming along with the speed and documentation advantages of a higher-level language. TMF has been used both to prepare standard statistics programs, and to program a real-time data-acquistion system for neurophysiological research.

## ACKNOWLEDGEMENT

This work was supported by Research Grant NB05090 from the National Institute of Neurological Diseases and Blindness, Public Health Service.

## APPENDIX

```
/        TMF:  THINKING MAN'S FORTRAN - A PROCEDURAL
/              SOFTWARE PACKAGE
/
/FRED R. SIAS, JR. AND ALLAN B. WILSON
/DIVISION OF NEUROLOGY
/UNIVERSITY OF MISSISSIPPI MEDICAL CENTER
/JACKSON, MISSISSIPPI 39216
/26 JUNE 1968
/
/
/THE FOLLOWING SET OF SUBROUTINES CAN BE USED TO SIMULATE
/FORTRANLIKE STATEMENTS WHILE ACTUALLY PROGRAMMING IN PAL III
/OR MACRO-8.  THE PACKAGE IS ORGANIZED AS A GROUP OF MODULES
/LOADED DOWNWARD IN MEMORY BELOW THE STANDARD FLOATING POINT
/PACKAGE (DIGITAL 8-5B-S) THAT STARTS AT LOCATION 5400.
/
/LOADED IN THE FOLLOWING ORDER, EACH IS INDEPENDENT OF ANY
/MODULE IN LOWER ORDER MEMORY.
/
/
/   7600-7777   /LOADERS
/
/   5400-7577   /BASIC FLOATING PT PACKAGE
/               /  WITH I/O CONTROLLER
/   5200-5377   /IOCS8, TEXT, AND LINKAGE ROUTINES
/
/   5000-5177   /FORTRANLIKE SUBROUTINES
/
/   4600-4777   /FORMATTING SUBROUTINES
/
/   4200-4577   /SCOPE OUTPUT PACKAGE
/
/   3600-4177   /FLOATING PT EXTENDED FUNCTIONS
/
/   0200-0332   /KEYBOARD MONITOR AND CALL HANDLER
/
/   0155-0177   /INDIRECT ADDRESSES
/
/   0040-0062   /FLOATING PT PACKAGE
/   0005-0007
/
/        0004   /ODT
/
/   0000-0003   /INTERRUPT
/
/LIBRARY TAPE IS A SERIES OF MODULES STARTING WITH IOCS8.
/(FLOATING POINT PACKAGE MAY BE PLACED FIRST ON THE SAME TAPE.)
/PRESS CONTINUE REPEATEDLY FOR AS MANY MODULES AS REQUIRED.
/LOWER ORDER MEMORY MODULES INSERT REQUIRED OVERLAYS. FOR INSTANCE
/SCOPE PACKAGE OVERLAYS ITS DAT SLOT IN IOCS8.
/
/
/         IOCS8:  INPUT-OUTPUT CONTROL MODULE
/
/IOCS8 USED WHENEVER THERE IS I/O TO BE DONE
/TEXT SUBROUTINE REQUIRES THAT STARTING ADDRESS
/  OF STRING BE IN ACC ON ENTRY
/FLOATING PT PACKAGE, IF USED, SHOULD BE LOADED
/  BEFORE THESE ROUTINES
/
/
/IOCS8 IS AN INPUT - OUTPUT CONTROL SUBROUTINE
/  ENTERED WITH NUMBER TO BE OUTPUTTED IN ACC
/  AND THE DEVICE CODE IN "DEVICE" AT LOCATION
/  Z 177
/  DEVICE REMAINS SELECTED UNTIL 177 IS CHANGED
/  EXIT FROM SUBR WITH INPUT CHAR IN ACC
/
```

```
/A DEVICE ASSIGNMENT TABLE STARTS AT LOCATION 52Ø5 WITH TWELVE
/POSITIONS AVAILABLE.  THE FOLLOWING DAT SLOTS HAVE BEEN
/ASSIGNED:
/
/      1  /TTY PUNCH
/      2  /TTY READER
/      3  /HIGH SPEED READER
/      4  /STORAGE SCOPE
/      5  /HIGH SPEED PUNCH
/
/   6-12  /SPARES
/
/
/THE MESAGE SUBROUTINE CAN ALSO BE USED ALONE BY ENTERING WITH
/THE ADDRESS OF THE PACKED, TRIMMED ASCII CODES IN THE ACC.
/THE CODES ARE INDENTICAL WITH THE DEC ALPHATYPEOUT CODES AND
/MUST TERMINATE WITH A ØØ CODE.  RETURN IS TO THE STATEMENT
/FOLLOWING THE ØØ WORD.
/
/
/  LINKAGE FACILITY
/
/GIVES THE PROGRAMMER THE ABILITY TO SEGMENT LONG PROGRAMS
/AND CALL IN THE SEGMENTS AS NEEDED.
/
/   LINK
/   3       /DEVICE NO.
/   "M      /PROGRAM NAME (ONE ASCII CHAR)
/
/OR  LINK; 3; "M
/
/BINARY PROGRAM SOUGHT MUST HAVE THE ASCII CODE PUNCHED
/PRECEDING IT.  THE CORRECT PROGRAM IS FOUND AND LOADED
/BY THE MISSISSIPPI LOADER WHICH WILL START IT AUTOMATICALLY.
/
/
/FORTRANLIKE STATEMENTS MODULE
/
/DESIGNED TO APPROXIMATE FORTRAN STATEMENTS
/
/STATEMENT SYNTAX:
/
/
/  IF STATEMENT
/
/ENTERED WITH TEST QUANTITY IN ACCUMULATOR
/
/   IF; ST1; ST2; ST3
/WHERE ST1, ST2, ST3 ARE ADDRESSES OF STATEMENTS FOR HANDLING
/MINUS, ZERO, AND PLUS CONDITIONS, RESPECTIVELY.
/
/
/COMPUTED GOTO
/
/ENTERED WITH INDEX IN ACCUMULATOR
/
/   GOTO; ST1; ST2; ST3; ST4...
/WHERE ST1, ST2, ...STN ARE ADDRESSES OF STATEMENTS
/FOR INDEX EQUAL 1, 2, ...N
/
/
/  DOLOOP
/
/USED TO ITERATE A SEQUENCE OF STATEMENTS
/
/   DO; N; INDEX, Ø
/   ...
/   CONT
/   INDEX
/
/WHERE N IS THE NUMBER OF ITERATIONS TO BE MADE, AND INDEX
/FOLLOWING CONT(INUE) IS THE ADDRESS OF THE LOOP INDEX.
/THE INCREMENT FOR INDEX IS ALWAYS ONE.  UNLIMITED NESTING
/IS PERMITTED.
/
```

```
/
/  FLOAT
/
/USED TO FLOAT A NUMBER IN THE REAL ACCUMULATOR.  IT IS
/LEFT IN THE FLOATING ACCUMULATOR.
/
/  FIX
/
/USED TO FIX A NUMBER IN THE FLOATING ACCUMULATOR.
/NUMBER MUST BE LESS IN MAGNITUDE THAN 2Ø47.  RESULT
/LEFT IN REAL ACCUMULATOR.
/
/
/       INPUT-OUTPUT FORMATTING MODULE
/
/  READ
/
/USED TO READ AN ASCII CHARACTER, A DECIMAL NUMBER OR A
/BINARY WORD.
/   READ
/   2       /DEVICE NO.
/   "I      /FORMAT TYPE - A, B, E, F, I
/
/OR READ; 2; "I
/
/READING BY A FORMAT INPUTS A CHAR AND LEAVES IT IN
/THE ACCUMULATOR.  READING BY B FORMAT INPUTS TWO CHARACTERS
/AND ASSEMBLES A 12 BIT BINARY WORD.  THIS CONTINUES UNTIL
/LEVEL 8 CODES ARE FOUND (RUBOUT OR LEADER/TRAILER CODES).
/THE DATA STRING IS LOADED INTO SEQUENTIAL WORDS STARTING AT
/THE ADDRESS STORED IN "PLACE" (LOC 16Ø) ON PAGE ZERO.
/A CHECKSUM IS CALCULATED ON PAGE ZERO AND COMPARED WITH THE
/CHECKSUM ON THE DATA TAPE; IF THE TWO DO NOT AGREE PROGRAM
/HALTS.  REPOSITION THE BAD SEGMENT OF TAPE AND REREAD BY
/PUSHING CONTINUE.
/
/READING BY E, F, OR I FORMAT USES THE FL PT INPUT
/ROUTINES BUT OF COURSE USES THE DEVICE SPECIFIED AFTER
/"READ".
/
/
/  WRITE
/
/   WRITE
/   4       /DEVICE NO.
/   "F      /FORMAT TYPE - A, B, C, E, F, H, I
/   6       /LIKE F6.2 IN FORTRAN
/   2
/
/OR  WRITE; 4; "F; 6; 2
/
/A FORMAT OUTPUTS THE CHAR IN THE ACC.  B FORMAT OUTPUTS
/TWO 6 BIT CHARACTERS AND INCREMENTS CHECKSUM.  CHECKSUM
/MUST BE EXPLICITELY SET TO ZERO AND PUNCHED BY PROGRAM.
/ C FORMAT OUTPUTS
/A CARRIAGE RETURN - LINEFEED.  E FORMAT OUTPUTS LIKE THE
/FORTRAN E FORMAT.  F FORMAT OUTPUTS LIKE THE FORTRAN F
/FORMAT.  I IS ALSO LIKE THE FORTRAN OUTPUT.  E, F, AND I
/FORMATS ALL OUTPUT THE QUANTITY IN THE FLOATING ACC.  H OR
/HOLLERITH FORMAT OUTPUTS THE TEXT IN THE CODED STRING FOL-
/LOWING THE CALL:
/
/   WRITE; 4; "H
/   TEXT *THIS IS HOLLERITH TEXT%#*
/
/
/  STORAGE SCOPE OUTPUT
/
/CALLED BY USING DEVICE NO. 4 FOLLOWING "WRITE".  WILL
/HANDLE ANY OF THE OUTPUT FORMAT SPECS.
/
/
```

```
/
/  ARRAY SUBROUTINE
/
/GIVES THE ADDRESS OF AN ELEMENT IN A ONE OR TWO
/DIMENSIONAL ARRAY.
/
/ARRAY MUST APPEAR IN MEMORY AS:
/
/      *STARTING ADDRESS OF ARRAY
/      NAME,    A    /NO. OF WORDS PER ELEMENT
/               B    /NO. OF ROWS
/               C    /NO. OF COLUMNS
/
/"C" SHOULD BE Ø FOR A ONE DIMENSIONAL ARRAY
/
/USE:
/
/      ARRAY
/      NAME        /NAME (STARTING ADDRESS) OF ARRAY
/      I           /NO. OF ROW OF ELEMENT
/      J           /NO. OF COLUMN OF ELEMENT
/
/OR ARRAY; NAME; I; J
/
/THE SECOND SUBSCRIPT SHOULD NOT BE PRESENT FOR A
/ONE DIMENSIONAL ARRAY (THOUGH "C" OF NAME, A; B; C
/MUST BE Ø).
/
/CALLING THE SUBROUTINE PUTS THE ADDRESS OF THE
/DESIRED ELEMENT IN "POINT" ON PAGE ZERO, LOCATION
/164.  IT CAN THEN BE USED TO REFER TO THE ELEMENT
/INDIRECTLY.

/
/  KEYBOARD MONITOR AND CALL ROUTINES
/
/
/KEYBOARD MONITOR  WHEN OPERATING READS A ONE
/CHAR ASCII CODE FROM THE TELETYPE.  THIS CHAR
/IS USED IN A TABLE LOOKUP TO JMP TO THE
/APPROPRIATE ROUTINE.  THE FINAL STATEMENT OF
/A KEYBOARD MONITOR CALLED PROGRAM SHOULD
/BE CALL; "K, WHICH WILL RETURN CONTROL TO THE
/KEYBOARD MONITOR. THE MONITOR CAN THEN
/BE USED TO CALL OTHER PROGRAMS.
/TELETYPE BELL SIGNALS RETURN TO MONITOR.
/
/  CALL ROUTINE
/
/THE CALL STATEMENT, USED MUCH LIKE THE SAME FORTRAN
/STATEMENT, SERVES TO CALL SUBROUTINES LOCATED
/ANYWHERE IN MEMORY.  THE SYNTAX IS CALL; "N
/WHERE N IS THE NAME OF THE DESIRED SUBROUTINE.
/THE FIRST STATEMENT OF THE SUBROUTINE SHOULD NOT
/BE Ø, BUT AN EXECUTABLE STATEMENT.  THE TERMINATING
/STATEMENT OF THE SUBROUTINE SHOULD BE RETURN.
/THE CALL STATEMENT MAY NOT REFER TO ANY SUBROUTINE
/NAME USED BY THE KEYBOARD MONITOR AS THE SAME TABLE
/IS USED FOR BOTH SUBROUTINES AND KEYBOARD CALLED
/MAIN ROUTINES.
/
/SUBROUTINES SHOULD BE COMPILED WITH THE FOLLOWING
/FORM WHICH AUTOMATICALLY LOADS THE CALL TABLE:
/
/        *"A                      /ADDRESS IN CALL TABLE
/        A                        /NAME OF PROGRAM
/        *ADDR                    /ACTUAL DESIRED ADDRESS OF START
/        A,      INSTRUCTION      /EXECUTABLE STATEMENT
/
/THE SUBROUTINES MAY BE MADE RELOCATABLE BY USING
/THE MISSISSIPPI LOADER.
/
```

# ACCESSING DATA ARRAYS AND TELETYPE TEXT INPUT/OUTPUT

David G. Frutchey
Beckman Instruments, Inc.
Fullerton, California

## ABSTRACT

Two subroutine packages are presented which, when used with any of the standard Floating Point Packages, provide the user with a powerful, yet concise programming methodology. Data storage or retrieval with floating to fixed, or fixed to floating point conversion can be performed on terms analogous to single variable, subscripted FORTRAN array terms such as "ARRAY (a*J+b)." A concise facility for text output (63 characters), character input, line spacing and tabulating is also demonstrated. These subroutine packages were written for the PDP-8 family of computers and were intended for use with the PAL-III language.

## EFFICIENT STORAGE UTILIZATION

The programming of a digital computer requires a knowledge of both procedural and arithmetic techniques. Higher level languages such as FORTRAN and FOCAL are recognized as being a significant aid to the programmer particularly for procedural operations and through the relative ease in which the language can be learned. Also, the completeness of these languages can be illustrated by the simple, direct way in which the procedural tasks of input/output, branching logic, data conversion (integer storage/floating point storage), looping to repeat procedures, etc. can be done. Among these procedural tasks is the ability to process single-subscripted-variable data array terms of the form ARRAY (a*J+b). The terminology used here is as defined in the American Standard FORTRAN in that the symbol, ARRAY, is a name given the beginning of successively stored data values and the subscript enclosed in parentheses with integer values of 1, 2, 3, ... refers, respectively, to the first, second, third, ... data array number.

Analogous procedural programming requires more effort in the assembly language of the computer than of the higher level languages available. However, quite often, reverting to the use of assembly language can permit significant gains in the critical availability of computer core storage. In a small machine, such as the PDP-8 family of computers, core storage is extremely limited and the desire to achieve improved core storage utilization is often an application program necessity.

What are some of the programming concepts which adversely affect core utilization? The following are a few:

1. Direct storage of higher level language in a form much more lengthy than the assembly language equivalent code.

2. Use of an interpretive system for all coding which in equivalent assembly form would not always require the use of an interpreter program.

3. Lengthy preparations for calling subroutines or lengthy calling sequence requirements to subroutines.

4. Duplication of coding in that an internally available routine is not used.

5. Failure to share work region requirements; that is, core storage which is temporarily used and otherwise available.

6. Overemphasis of the coding consideration of the seldom used (if ever) requirements, rather than permitting omissions with an attendant savings in core; or by providing alternately coded packages so that the user may have a choice.

The appearance of programming systems which attempt to preserve the procedural niceties of the higher level languages are prevalent today; however, many of these systems suffer from some of these same adverse conditions and produce little improvement in core utilization. The adverse concepts listed above may serve as a guide when evaluating both the system presented herein and other such systems.

## TWO NEW PACKAGES

Two subroutine packages were developed during a biomedical project in which minimizing the core storage requirements was essential. To this end, the packages entitled "Teletype Text Input/Output Package" and the "Array Accessing Subroutine Package" were written. The usage and features of these routines enable a user to code several procedural tasks such as teletype page formatting, output and input, and processing data arrays. Also illustrated in the examples given are means by which other procedural tasks such as looping and decision branching could be coded. These two packages require the use of the Floating Point Package[1] and are partially integrated in their use and performance. The overall purpose in developing these packages was to ENABLE MINIMUM CORE STORAGE REQUIREMENTS. At the same time, the peripheral benefits of being able to think or plan a program in FORTRAN are not altogether lost. In fact, the examples show FORTRAN equivalents which are a suggestion that FORTRAN coding could readily precede the use of these routines. A second peripheral benefit is in the increase obtained in computational computer speed by these routines when compared to the present FORTRAN-produced code.

## THE TELETYPE TEXT I/O PACKAGE

This package consists of three routines; namely, subroutines TAB, TEXT and ECHO. The subroutines are "called" in the PAL-III assembly language through the use of the Jump to Subroutine (JMS) mnemonic. As in the Floating Point Package, a TLS must be used at least once prior to using routines TAB and TEXT.

The Teletype Text I/O Package is available in two forms which are a Stand-Alone version and a version to be used when the Floating Point Package is resident which will be referred to as the FPP version. The Stand-Alone version occupies 63 decimal storage locations. The FPP version requires 55 decimal storage locations. The FPP version has the distinct advantage of using the Floating Point Package for all teletype commands. This feature is useful in that if interrupt I/O is implemented, modification to the Floating Point Package only need be made.

### Subroutine TAB

It performs tabulation across a page by typing blanks, or performs multiple line spacing down a page by typing carriage return (CR), line feed (LF) combinations. The routine reacts to the accumulator c(Ac) contents as follows:

| If c(Ac) | Output |
|---|---|
| =0 | One CR and one LF |
| =n positive, | One CR and n+1 LF |
| =n negative, | \|n\|-1 blanks |

No calling sequence parameters are expected by subroutine TAB other than the c(Ac), and the routine always returns with c(Ac)=0.

### Subroutine TEXT

This subroutine performs the output of a message stored as six-bit (stripped) code with two characters per computer word. The message is formed by subtracting octal 240 from any USASC-II eight-bit code in the octal range 240 to 336, inclusively. The two-digit, octal results representing a character are placed into the first half, followed by a character in the last half of each computer word and continued into successive core storage locations. The two-digit, octal code 77 is used as the message termination signal and must appear in either the first or last half of the final computer word used for the messages. Messages may be as long as desired and may overlap page boundaries. The code includes the use of the alphabet, numerals, punctuation, etc. The routine is called without regard to the c(Ac) and requires the address of the first word of the message to follow the call. This feature was provided rather than following the call with the message itself to enable portions of a message to be reusable and to permit the placement of the message into core locations which would not otherwise be used. The routine differs from previously written routines in that the link register (a one-bit, two-state register) is used to alternately point to the character to be printed. Also, the switching of the link pointer is performed without command through the arithmetic in testing for the terminating character.

Example 1 - Using Subroutine TEXT to type "NO GO"

```
⋮
JMS TEXT          /JUMP TO SUBROUTINE TEXT
MSGADR
          ←  PROCESSING CONTINUES HERE
⋮
MSGADR, 5657      /NO
0047              /-G
5777              /O TERMINATION
```

Subroutine TEXT always returns with the c(Ac)=0.

### Subroutine ECHO

This routine performs the input and typing of a single USASC-II, eight-bit code character. When called, the routine will "wait" until a keyboard entry is made and will return to the location immediately following with c(Ac)=0 if the RUB-OUT key is entered. If any other key is entered, the routine will return to resume past the RUB-OUT return point and will contain the USASC-II character in the c(Ac). Additionally, the character will be stored at location CHAR in the listing of the subroutine ECHO used. When the Floating Point Package is to be used, the character is returned at octal location CHAR=0057. The Stand-Alone version of ECHO deposits the character at location CHAR=TEXT and it will be destroyed upon any subsequent call to either TAB, TEXT or ECHO.

Example 2 - Using subroutine ECHO to input a character.

```
       ⋮
AGAIN, JMS ECHO
JMP AGAIN           /WHEN RUB-OUT IS ENTERED
              ←  PROCESSING CONTINUES HERE
                 WHEN THE CHARACTER IS NOT
                 THE RUB-OUT KEY.
```

## THE ARRAY ACCESSING SUBROUTINE PACKAGE (AASP)

This package provides subroutines which can be used to access single subscripted variable arrays of the form ARRAY (a*J+b). The use of the Floating Point Package is required in that several of the Floating Point Package internal routines and services are used. Also, work regions of the Floating Point Package are shared.

The Array Accessing Subroutine Package requires 110 (decimal) storage locations and is designed to reside beginning at core location octal 4400. When a Floating Point Package version is used which does not include the subroutine FIX at location 4557, the Array Accessing Subroutine Package will supply subroutine FIX. In this event, all of the locations on the page between octal 4400-4577 are required. Ten of the locations required reside at the end of two pages within the Floating Point Package (octal 6173-6177, and 6573-6577).

The routines to be used are named as follows and will perform the attendant procedure:

| Name | Procedure |
|---|---|
| LOAD | Clear the floating accumulator c(FAC) and load from a floating point array. |
| ILOAD | Clear c(FAC) and load from an integer data array. |
| STORE | Store c(FAC) into a floating point data array. |
| ISTORE | Store c(FAC) into an integer data array. |
| FIND | Without disrupting c(FAC), load a temporary accumulator defined as ANS at location 52, 53 and 54 from a floating point data array. |
| IFIND | Without disrupting c(FAC) load c(ANS) from an integer data array. |

The routines are entered while operating within the Floating Interpreter via the table provided at octal locations 6556 through 6563. The proper contents of these locations will be furnished in the Array Accessing Subroutine Package. A user must define the table entry naming convention by providing the following octal equates to the Assembler:

```
LOAD   = 0012
ILOAD  = 0013
STORE  = 0014
ISTORE = 0015
FIND   = 0016
IFIND  = 0017
```

The FIXTAB pseudo-op may be used for this purpose if desired. It may be well to note that these routines may not be called while not within the floating interpreter. Notice that the use of the ILOAD, ISTORE initiates both data array accessing and c(FAC) conversion. The routines can certainly be used for the simpler usage where conversion only is desired. The resulting value will be found in the c(FAC) after either the ILOAD (load & fix/float) or the ISTORE (float/fixed & store).

Each routine requires either a two or three-word calling sequence as follows:

| | |
|---|---|
| ENTRY NAME | The routine to be entered. |
| ARGUMENT 1 | The address of the array. |
| ARGUMENT 2 (OPTIONAL) | The values of a and b. |

The ENTRY NAME is coded by selecting the desired array operation, namely, LOAD, ILOAD, etc. Argument one is the array starting address which can be supplied by writing the same name used to identify the beginning of the data array in the program. If the optional argument two will follow argument one, bit zero of argument one must be turned on. This can be accomplished in a variety of ways, but perhaps the most direct way is to add octal 4000 to the array name coded as argument one.

The second argument is optional and should be coded only when the first argument has bit zero set to a one. This argument two contains the subscript values for a and b in the subscript (a*J+b). If the second argument is not supplied, these values are taken as a=1 and b=0. Otherwise both of these values are supplied in argument two as shown in the following 12-bit computer word:

| Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | b | a | a | a | a | a | b | b | b | b | b | b |

The sign of b. The value of a. The value of b.

If b is negative, b must be supplied as a twos complement number.

Lastly, the variable parameter, J, of the array subscript is provided as an integer value by the user on page zero at core location 0050. This location is internal to the Floating Point Package requirements and, hence, does not constitute an additional core storage requirement. The value of J should be set to 1, 2, 3, ... etc. to access the first, second, third, ... etc. data array number regardless of whether the data is stored as floating point or fixed point data. Additionally, J may be given the value of zero as a special case in which the first number of either a floating or fixed data array will be accessed.

Arrays may consist of either integer values stored successively in core storage or as three-word groupings of floating point numbers. In either case, the first computer word of an array should be given a name in a PAL-III program to be used as the array starting address. Arrays may be of any desired length and may overlap page boundaries in any way desired. Also, arrays may consist of only one value which is not really an array; however, the ability to access any symbolic named integer in a program is useful in that the value can be loaded with conversion to floating point for computational use. The load with conversion characteristic is also useful in that arrays which otherwise might have been maintained as floating point arrays can, in some cases, be stored as integer arrays with a significant savings in core space.

The following examples will illustrate the usage of the Array Accessing Subroutine Package:

Example 3 - Move array named HERE to occupy the array named THERE. Both arrays are 10 words long.

```
        :
CLA
TAD NEGTEN      /PREPARE FOR LOOP
DCA CTR
DCA J           /ZERO J
LOOP, ISZ J     /INCREMENT J
FINT            /ENTER INTERPRETER
LOAD            /CLEAR AND LOAD C(FAC)
HERE            /FROM HERE (J)
STORE           /STORE C(FAC)
THERE           /INTO THERE (J)
FEXT            /EXIT INTERPRETER
ISZ CTR
JMP LOOP
        :
/DATA SECTION
J=50
ANS=52
FINT=JMS I 7  /TO ENTER THE INTERPRETER
NEGTEN,-12
CTR,Ø
```

```
HERE=.          /DIMENSION HERE(10),THERE(10)
*.+36           /THAT'S OCTAL LENGTH OF 10 FLOATING
                /POINT NUMBERS
THERE=.
*.+36
```

In the preceding example, the coding of a loop is seen to be a direct feature of the PAL-III language Also, the equivalent of a FORTRAN DIMENSION statement is shown. The coding of the equivalents to a FORTRAN 'DO' loop and a FORTRAN 'IF' statement is shown in the following example, in addition to further use of the Array Accessing Routines, LOAD and FIND.

Example 4 - Write the following FORTRAN coding in PAL-III with use of the Array Accessing Subroutine Package (AASP). Notice that the use of mixed mode computation is indicated.

FORTRAN

```
DO 180 J=1,4
IF(DATA(2*J+3)-IVAR(J)) 155,160,160
      :
155   :
160   :
180 CONTINUE
      :
      :
```

Using AASP

```
CLA
TAD NEG4        /THAT'S -4
DCA CTR
DCA J           /ZERO J
DO180,ISZ J     /BEGINS DO LOOP, DO 180
FINT            /FLOATING INTERPRETER
LOAD
DATA+4000
0203            /LOADS DATA (2*J+3)
IFIND           /NOTICE MIXED MODE IS SUPPORTED
IVAR            /C(ANS) HAS IVAR(J) NOW CONVERTED
FSUB ANS        /SUBTRACT
FEXT            /EXIT INTERPRETER
TAD 45          /GET WORD WITH SIGN FOR 'IF'
SMA CLA         /SKIP 'IF' MINUS
JMP S160
S155,  :
       :
S160,  :
       :
S180, ISZ CTR   /180 CONTINUE
JMP DO180
       :
       :
/DATA SECTION
DATA=.          /DIMENSION DATA(4),IVAR(4)
*.+14           /FOUR FLOATING POINT NUMBERS
IVAR=.
*.+4            /FOUR INTEGER NUMBERS
```

Limitations of AASP

The Array Accessing Subroutine Package has several deliberate limitations; however, the major objective in permitting savings in core storage is the underlying justification for these limitations. The limits are as follows:

Consider the term ARRAY (a*J+b)

1. Range of a is given by $0 \le a \le 31$

2. Range of b is given by $-64 \le b \le 63$

3. Range of J is given by $-2048 \le J \le 2047$ (Range of 4K machine)

4. The starting address of any array to be accessed must be in the first 2K of storage (0000-3777 octal).

5. The temporary accumulator, ANS, located at c(0052, 0053, 0054) will be destroyed during the use of the square root, square, or an I/O controller INPUT or OUTPUT. Note that in the case of an OUTPUT operation, the c(FAC) is destroyed also.

Numeric Input and Output

The Array Accessing Subroutine Package provides an additional service to augment the use of the standard Floating Point Package numeric input and output routines. The subroutine entry table is supplied with input and output transfer addresses at octal locations 6554 and 6555. Hence, I/O can be performed while within the interpreter if the octal equates INPUT=0010 and OUTPUT=0011 are supplied to the Assembler. Numeric input in integer, fixed point or floating point notation may be encoded by simply writing INPUT while within the interpreter. Similarly, output of the c(FAC) may be initiated by encoding OUTPUT. Use of the output controller is expected when using this OUTPUT feature; however, if desired, the omission of the output controller must be indicated by placing octal 7200 into location 6555 by the user. When the output controller is used, the octal storage locations at c(62) and c(63) control the output format where c(62) should be given the field width, c(63) the decimal field width. These locations must be maintained by the user and are not initialized. If the output controller is not used, the locations are not required. A routine is provided to place the decimal field width into the accumulator as required when using the Floating Point Package output controller.

## OTHER TECHNIQUES

Relational Operators

It might be valuable to point out that the Relational Operators usable in 'IF' statements of recent FORTRAN versions can be equivalently coded in PAL-III through the use of the following convenient mnemonics:

| MNEMONIC | | SKIP NEXT INSTRUCTION IF C(AC) ... |
|---|---|---|
| SGT=7550 | /SPA SNA | SKIP IF .GT. Ø |
| SGE=7510 | /SPA | SKIP IF .GE. Ø |
| SEQ=7440 | /SZA | SKIP IF .EQ. Ø |
| SNE=7450 | /SNA | SKIP IF .NE. Ø |
| SLE=7540 | /SMA SZA | SKIP IF .LE. Ø |
| SLT=7500 | /SMA | SKIP IF .LT. Ø |

These mnemonics have been found to be extremely beneficial to the FORTRAN-oriented mind and have been made a permanent part of our PAL-III Assembler.

Coding Across Page Boundaries

Another procedural benefit in using FORTRAN or FOCAL is the absence of 'page' considerations in the PDP-8 family of computers. A technique can be employed in the PAL-III language which can greatly reduce the problems associated with pages.

Coding may begin at a convenient location (sav octal 0200) and continue straight through crossing page boundaries.

Adopt the following coding conventions to obtain this end:

1. Use page zero for integer constants, a small work area (loop control, etc.) and for indirectly addressing absolute references to floating point values or subroutines.

2. Use the Array Accessing Subroutine Package for array processing.

3. Use the Teletype Text I/O package for referencing messages.

4. Code all jump (JMP) instructions without using indirect addressing.

Upon assembling a program written by these conventions, the Assembler will report which jump instructions must be changed to indirect addresses. Changing the jump instructions and adding the indirect addresses to page zero will produce a program which will execute across page boundaries.

## SUMMARY

When combined with the Floating Point Package, the Teletype Text I/O Package and the Array Accessing Subroutine Package provide a PAL-III programming methodology. The discussion and several examples presented show the coding of the following techniques:

1. Multiple tabulating and line spacing.

2. Multiple-word, message output stored as two characters per word.

3. USASC-II, eight-bit code character input with RUB-OUT key recognition.

4. Numeric input and output in any of the formats: integer, fixed or floating point.

5. Accessing single variable, subscripted array terms of the general form ARRAY (a*J+b) to perform loading, storing, locating and, inherently, data conversion and in supporting mixed mode data calculation.

6. The directly available features of PAL-III in coding equivalents to FORTRAN 'DO' loops, 'IF' statements with either arithmetic or relational arguments, DIMENSION statements, and coding across page boundaries.

The emphasis in developing and presenting these techniques has been to utilize a minimum of core storage for each procedural task presented. These routines have been intensively used and appear to be completely reliable.

A brief testimonial is contained in my initial usage of these routines. A biomedical computer application,[2] which was only partly written in FORTRAN, occupied 96 per cent of available core storage. Text information was stored as two characters per word and routines not required within the FORTRAN Controller were overlaid when the program was in working order. The need for additional computational application features prompted the rewrite of this program to eliminate the use of the FORTRAN controller and the remaining FORTRAN code. The resulting program (which also worked) not only occupied less core space, now requiring only 63 per cent of core storage, but also executed much faster. The former FORTRAN program version required 12 seconds per compute cycle to initiate plotter movements while the program version employing the techniques of this paper required less than three seconds per compute cycle to initiate plotter movements.

## REFERENCES

1. Floating-Point System Programming Manual, 8-5-S, Digital Equipment Corporation, Maynard, Massachusetts, 1965.

2. Savaglio, Fred J., Pacela, Allan F., "Monitoring and Display of Continuous Real-Time Blood Chemistry Data", Proceedings of the 21st Annual Conference on Engineering in Medicine and Biology, November 1968.

```
/
/  TTY TEXT I/O SUBROUTINE PACKAGE, STAND-ALONE VERSION
*5400    / PLACE WHERE WANTED
/     THIS PACKAGE PERFORMS INPUT AND OUTPUT (I/O)
/
/     PREPARED BY D G FRUTCHEY, BECKMAN INST.S, JUNE, 1968
/
/  SUBROUTINE TAB
/  THIS ROUTINE OUTPUTS BLANKS, CARRIAGE RETURNS (CR),
/  AND LINE FEEDS (LF) ACCORDING TO THE ACCUMULATOR CONTENTS.
/     IF C(AC) = POSITIVE INTEGER, N, GET CR & N+1 LF'S
/     IF C(AC) = ZERO, GET CRLF
/     IF C(AC) = -1, GET A CR
/     IF C(AC) = NEGATIVE INTEGER, N, GET ABS(N)-1 BLANKS
TAB,NOP
STL IAC
SMA
CMA CLL    / C(L) = 0 MARKS CRLFS
DCA TEMP
SNL
TAD POS3
TAD BLANK
SNL
TAD DIFF
JMS TYPE
ISZ TEMP
JMP .-5
JMP I TAB
POS3,3
DIFF,3752
BLANK,0240
TEMP=TEXT    / TEMPORARY STORAGE
/
/        SUBROUTINE TEXT
/     OUTPUT STRIPPED CODE BEGINNING AT ADDRESS PASSED.
/     EXAMPLE OF USAGE;
/     JMS TEXT          /TO CALL THIS ROUTINE
/     MSG1              /THE SYMBOL USED TO BEGIN THE MESSAGE
TEXT,NOP
CLA CLL    / INSURE THAT LINK IS CLEAR
TAD I TEXT
ISZ TEXT
DCA TEMP2
GET1,TAD I TEMP2
SZL
JMP .+5
RTR      /FIRST TIME THROUGH
RTR
RTR
CLL
AND MSK
TAD M77
SNA
JMP I TEXT    / DONE, RETURN
TAD MSK /RESTORE AC AND COMP LINK
TAD BLANK
JMS TYPE
SNL
ISZ TEMP2    / SECOND TIME THROUGH
JMP GET1
MSK,0077
M77,7701
TEMP2=TAB    / TEMPORARY STORAGE
/
/        SUBROUTINE ECHO
/        USAGE :
/        JMS ECHO        /TO GET ASCII CHAR INTO C(AC) & TYPED
/        JMP AGAIN       /IF RUBOUT WAS INPUT
ECHO,NOP
KSF     / SKIP ON INPUT
JMP .-1 / WAIT
KRB     / CLEAR C(AC), READ BUFFER, CLEAR FLAG
DCA CHAR
TAD CHAR
SNA
JMP ECHO+1    / ALSO WAIT WHEN NULL OR BLANK TAPE .
JMS TYPE    / ECHO THE CHARACTER
TAD CHAR
TAD MRBOUT    / HAVE RUB-OUT?
SNA CLA
JMP I ECHO    / RUB-OUT WAS ENTERED
ISZ ECHO      / INCREMENT TO BYPASS
TAD CHAR
JMP I ECHO
MRBOUT,-377
CHAR=TEXT
/
/  SUBROUTINE TYPE
TYPE,NOP
TSF
JMP .-1
TLS
CLA
JMP I TYPE
$
```

```
/  TTY TEXT I/O SUBROUTINE PACKAGE, FPP VERSION
*5400    / PLACE WHERE WANTED
/     THIS PACKAGE PERFORMS INPUT AND OUTPUT (I/O)
/     BY USING THE ROUTINES FURNISHED IN THE
/     FLOATING POINT PACKAGE (INPUT & OUT) TO
/     PERFORM SINGLE CHARACTER TRANSFER.
/
/     PREPARED BY D G FRUTCHEY, BECKMAN INST.S, JUNE, 1968
/
/  SUBROUTINE TAB
/  THIS ROUTINE OUTPUTS BLANKS, CARRIAGE RETURNS (CR),
/  AND LINE FEEDS (LF) ACCORDING TO THE ACCUMULATOR CONTENTS.
/     IF C(AC) = POSITIVE INTEGER, N, GET CR & N+1 LF'S
/     IF C(AC) = ZERO, GET CRLF
/     IF C(AC) = -1, GET A CR
/     IF C(AC) = NEGATIVE INTEGER, N, GET ABS(N)-1 BLANKS
TAB,NOP
STL IAC
SMA
CMA CLL    / C(L) = 0 MARKS CRLFS
DCA TEMP
SNL
TAD POS3
TAD BLANK
SNL
TAD DIFF
JMS I TYPE
ISZ TEMP
JMP .-5
JMP I TAB
POS3,3
DIFF,3752
BLANK,0240
TEMP=TEXT    / TEMPORARY STORAGE
/
/        SUBROUTINE TEXT
/     OUTPUT STRIPPED CODE BEGINNING AT ADDRESS PASSED.
/     EXAMPLE OF USAGE;
/     JMS TEXT          /TO CALL THIS ROUTINE
/     MSG1              /THE SYMBOL USED TO BEGIN THE MESSAGE
TEXT,NOP
CLA CLL    / INSURE THAT LINK IS CLEAR
TAD I TEXT
ISZ TEXT
DCA TEMP2
GET1,TAD I TEMP2
SZL
JMP .+5
RTR      /FIRST TIME THROUGH
RTR
RTR
CLL
AND MSK
TAD M77
SNA
JMP I TEXT    / DONE,RETURN
TAD MSK /RESTORE AC AND COMP LINK
TAD BLANK
JMS I TYPE
SNL
ISZ TEMP2    / SECOND TIME THROUGH
JMP GET1
MSK,0077
M77,7701
TEMP2=TAB    / TEMPORARY STORAGE
/
/        SUBROUTINE ECHO
/        USAGE :
/        JMS ECHO        /TO GET ASCII CHAR INTO C(AC) & TYPED
/        JMP AGAIN       /IF RUBOUT WAS INPUT
ECHO,NOP
CLA CLL
TAD ECHO
DCA I RESTRT
JMS I INPUT
CLA CLL
TAD INPKG
DCA I RESTRT
ISZ ECHO         /TO RETURN PAST THE RUBOUT JMP
TAD CHAR         /RETURNS WITH CHAR IN C(AC) & AT LOC 0057
JMP I ECHO
RESTRT,7167
INPUT,7142       /FPP ROUTINE INPUT
INPKG,7401
TYPE,7344     /FPP ROUTINE OUT ADDRESS
CHAR=0057
$
```

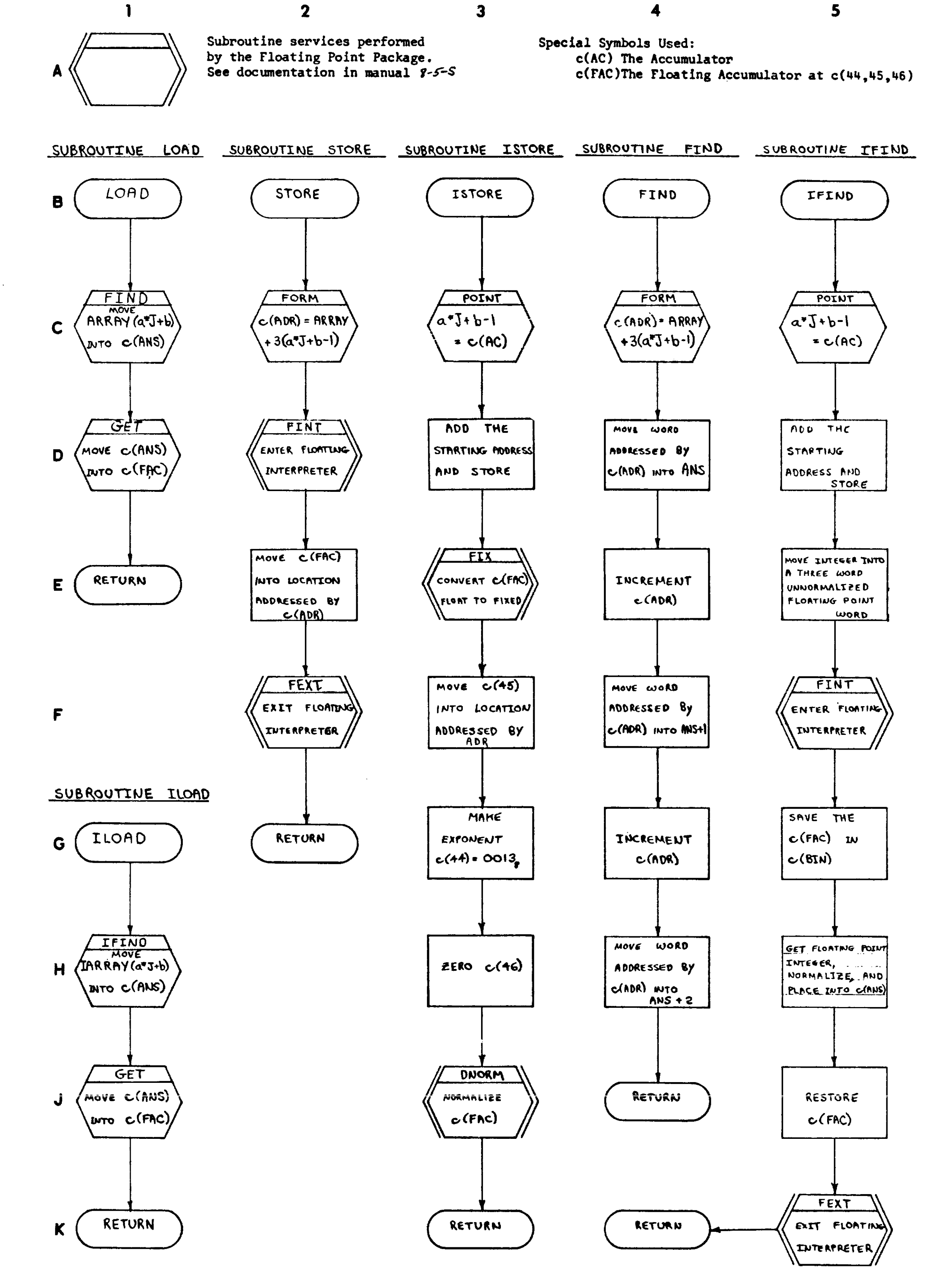
```
/
/
/       ARRAY ACCESSING SUBROUTINE PACKAGE
/       LOAD AFTER LOADING THE FLOATING POINT PACKAGE.
/       THESE ROUTINES ARE USED TO ACCESS ARRAYS OF THE
/       FORM, ARRAY(A*J+B) AND TO PROVIDE FIXED/FLOAT
/       AND FLOAT/FIXED DATA CONVERSION.
/  THIS PACKAGE WAS PREPARED BY D G FRUTCHEY,
/    BECKMAN INSTRUMENTS, FULLERTON, CALIF, OCTOBER, 1968
*6554
FINT=4407       /JMS I 7
7400    / INPUT = 10
DECOUT     / OUTPUT = 11
LOAD       / LOAD = 12
ILOAD      / ILOAD = 13
STORE      / STORE = 14
ISTORE     / ISTORE = 15
FIND       / FIND = 16
IFIND      / IFIND = 17
*6573
DECOUT,NOP
DEC=63
TAD DEC /LOAD NR OF DEC PLACES
JMS I FPPOUT
JMP I DECOUT
FPPOUT,7200
*6173
/
/  SUBROUTINE GET
GET,NOP    / RESTORE THE C(FAC) FROM ANS
FINT
FGET ANS
FEXT
JMP I GET
/
/  EQUATE SYMBOLS
A=40
B=41
SEE=42
J=50    /USER SETS J HERE
ADR=51
ANS=52    / TEMPORARY ACCUMULATOR, ANS
/
/  SUBROUTINE LOAD
*4400
LOAD,NOP    / LOAD C(FAC) WITH FLOATING POINT NR
JMS FIND
JMS I AGET
JMP I LOAD
/
/  SUBROUTINE ILOAD
ILOAD,NOP    / LOAD C(FAC) WITH FIXED POINT NR
LFIND=IFIND
JMS LFIND
JMS I AGET
JMP I ILOAD
/
/  SUBROUTINE STORE
STORE,NOP
JMS FORM
FINT
FPUT I ADR
FEXT
JMP I STORE
/
/  SUBROUTINE ISTORE
ISTORE,NOP    / CONVERT C(FAC) TO FIXED & STORE
JMS POINT
TAD ADR
DCA ADR    / POINTS TO PLACE TO STORE
JMS FIX
TAD 45
DCA I ADR
TAD P13
DCA 44
DCA 46
JMS I NORM
JMP I ISTORE
/
/  SUBROUTINE FIND
FIND,NOP    / LOAD ARRAY(A*J+B) INTO ANS
JMS FORM
TAD I ADR    / MOVE INTO ANS
DCA ANS
ISZ ADR
TAD I ADR
DCA ANS+1
ISZ ADR
TAD I ADR
DCA ANS+2
JMP I FIND
```

```
/
/  SUBROUTINE IFIND
IFIND,NOP    / LOAD FIXED INTO ANS & FLOAT
JMS POINT
TAD ADR
DCA ADR    / POINTS TO FIXED POINT ARRAY MEMBER
TAD I ADR
DCA P13+1    / SET UP INTEGER
FINT
FPUT BIN    / HOLD C(FAC)
FGET P13
FNOR        / NORMALIZE
FPUT ANS
FGET BIN    / RESTORE C(FAC)
FEXT
JMP I IFIND
/
/  SUBROUTINE FORM
FORM,NOP    / FORM ADDRESS FOR FLOATING POINT
JMS POINT    / RETURN WITH A*J+B-1
TAD B
TAD B     / TRIPLE
TAD ADR   / FORM POINTER
DCA ADR
JMP I FORM
/
/  SUBROUTINE POINT
/     CALCULATE A*J+B-1 AND RETURN IN C(AC)
/     ALSO, RETURN ARRAY ADDRESS IN ADR
POINT,NOP
TAD I G02    / GET POINTER TO CALLING SEQUENCE
DCA SEE
TAD I SEE    / GET ARRAY ADDRESS
AND MSK1    / 3777  DELETE ZEROTH BIT
DCA ADR
TAD I SEE   / ARRAY ADDRESS AGAIN
ISZ I G02    / TO BYPASS ARGUMENT 1
SPA CLA
JMP A5
TAD J    / NO 2ND ARGUMENT
SZA     / J MAY BE 0 OR 1 TO GET ARRAY(1)
TAD NEG1
JMP A10
A5,ISZ SEE
TAD I SEE    / GET   BAA AAA BBB BBB   WORD
AND MSK2    / 4077  GET SIGN AND B
SPA
TAD MSK3    / 3700  EXTEND MINUS BITS
DCA B    / -64<B<+64  DECIMAL
TAD I SEE
ISZ I G02    / TO BYPASS ARGUMENT 2
AND MSK3    / 3700  GET A
CLL RTR    / SHIFT
RTR        .
RTR
DCA A     / 0<=A<+32  DECIMAL
/  EVALUATE A*J+B-1
TAD J
DCA I MP2P   / MULTIPLICAND
TAD A    / MULTIPLIER
SZA      / NO NEED TO MULT IF A=0
JMS I MP4P   / 12 BIT MULTIPLY
TAD B
TAD NEG1
A10,DCA B
TAD B
JMP I POINT    / RETURN
/  CONSTANTS
G02,5655    / IN THE INTERPRETOR
MSK1,3777
MSK2,4077
MSK3,3700
MP2P,6471    / LOCATION OF MULTIPLICAND
MP4P,6437    / FPP ROUTINE FOR FIXED POINT MULTIPLY
AGET,GET
NEG1,-1
NORM,6600    / FLOATING POINT NORMALIZE
P13,13
0
0
BIN,0
0
0
/
/  SUBROUTINE FIX
*4557
FIX,NOP    / FIX C(FAC)
TAD 44
SMA SZA
JMP .+3
CLA
JMP FIXEND
TAD M13
DCA 44
LOOP,TAD 44
SMA CLA
JMP I FIX
JMS I .+2
JMP LOOP
6200      / DIV1 IN INTERPRETER
FIXEND,DCA 45
JMP I FIX
M13,-13
$
```

Beckman® **FLOWCHARTING WORKSHEET**

Program Name ARRAY ACCESSING SUBROUTINE PACKAGE Deck Name AASP

Programmer DAVID G. FRUTCHEY Date NOVEMBER 18, 1968 Chart Page 1 Of 2


1
2
3
4
5
A
B
C
D
E
F
G
H
J
K
SUBROUTINE LOAD
LOAD
FIND
MOVE ARRAY (a*J+b) INTO c(ANS)
GET
MOVE c(ANS) INTO c(FAC)
RETURN
SUBROUTINE ILOAD
ILOAD
IFIND
MOVE IARRAY(a*J+b) INTO c(ANS)
GET
MOVE c(ANS) INTO c(FAC)
RETURN
SUBROUTINE STORE
STORE
FORM
c(ADR) = ARRAY +3(a*J+b-1)
FINT
ENTER FLOATING INTERPRETER
MOVE c(FAC) INTO LOCATION ADDRESSED BY c(ADR)
FEXT
EXIT FLOATING INTERPRETER
RETURN
SUBROUTINE ISTORE
ISTORE
POINT
a*J+b-1 = c(AC)
ADD THE STARTING ADDRESS AND STORE
FIX
CONVERT c(FAC) FLOAT TO FIXED
MOVE c(45) INTO LOCATION ADDRESSED BY ADR
MAKE EXPONENT c(44) = 0013₈
ZERO c(46)
DNORM
NORMALIZE c(FAC)
RETURN
SUBROUTINE FIND
FIND
FORM
c(ADR) = ARRAY +3(a*J+b-1)
MOVE WORD ADDRESSED BY c(ADR) INTO ANS
INCREMENT c(ADR)
MOVE WORD ADDRESSED BY c(ADR) INTO ANS+1
INCREMENT c(ADR)
MOVE WORD ADDRESSED BY c(ADR) INTO ANS + 2
RETURN
RETURN
SUBROUTINE IFIND
IFIND
POINT
a*J+b-1 = c(AC)
ADD THE STARTING ADDRESS AND STORE
MOVE INTEGER INTO A THREE WORD UNNORMALIZED FLOATING POINT WORD
FINT
ENTER FLOATING INTERPRETER
SAVE THE c(FAC) IN c(BIN)
GET FLOATING POINT INTEGER, NORMALIZE, AND PLACE INTO c(ANS)
RESTORE c(FAC)
FEXT
EXIT FLOATING INTERPRETER

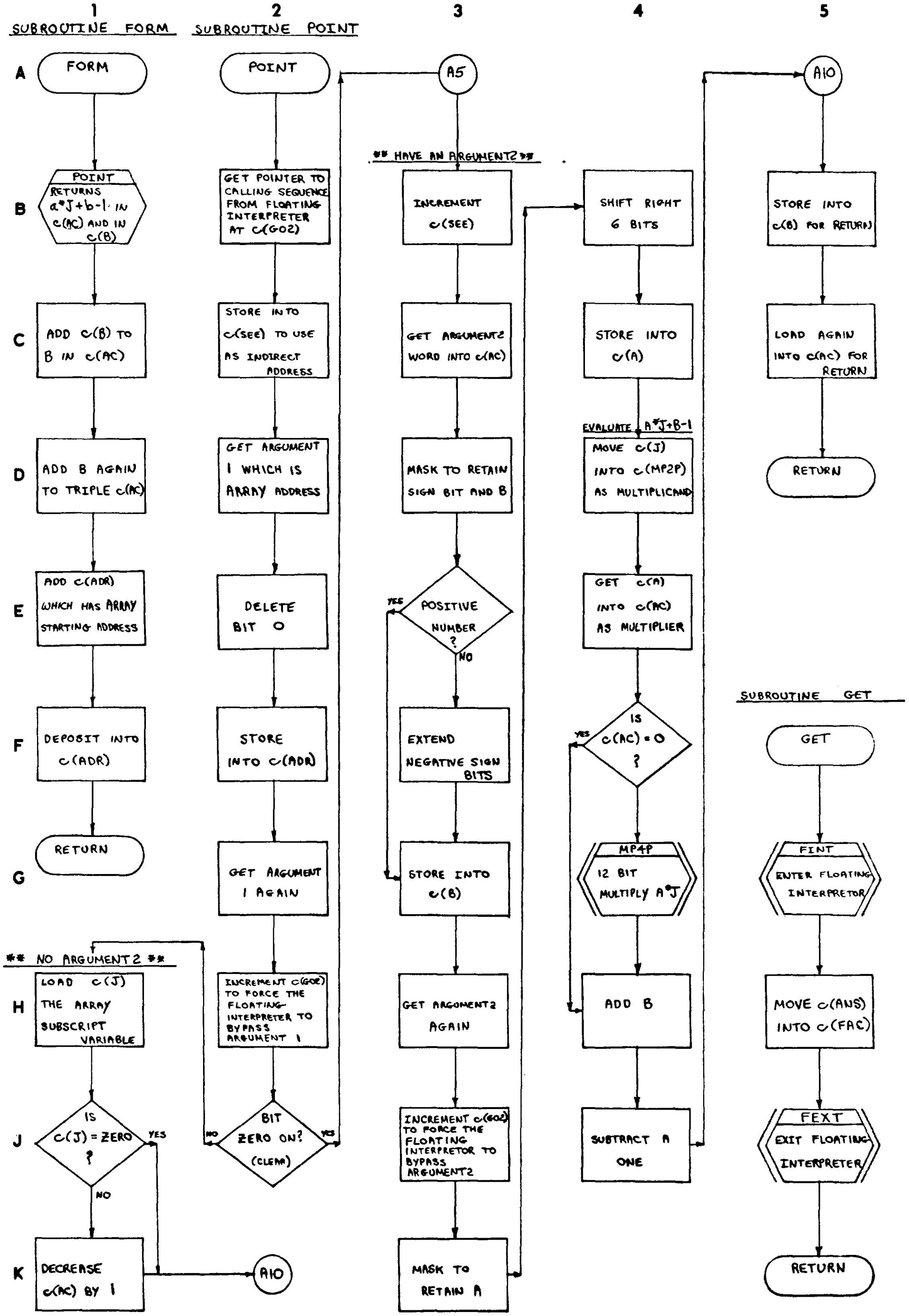
Subroutine services performed by the Floating Point Package. See documentation in manual 8-5-S

Special Symbols Used:
c(AC) The Accumulator
c(FAC)The Floating Accumulator at c(44,45,46)

**Beckman® FLOWCHARTING WORKSHEET**

Program Name ARRAY ACCESSING SUBROUTINE PACKAGE Deck Name AASP

Programmer DAVID G. FRUTCHEY Date NOVEMBER 18, 1968 Chart Page 2 Of 2


1
2
3
4
5
SUBROUTINE FORM
SUBROUTINE POINT
A
B
C
D
E
F
G
H
J
K
FORM
POINT RETURNS a*J+b-1 IN c(AC) AND IN c(B)
ADD c(B) TO B IN c(AC)
ADD B AGAIN TO TRIPLE c(AC)
ADD c(ADR) WHICH HAS ARRAY STARTING ADDRESS
DEPOSIT INTO c(ADR)
RETURN
** NO ARGUMENT2 **
LOAD c(J) THE ARRAY SUBSCRIPT VARIABLE
IS c(J) = ZERO ?
YES
NO
DECREASE c(AC) BY 1
A10
POINT
GET POINTER TO CALLING SEQUENCE FROM FLOATING INTERPRETER AT c(G02)
STORE INTO c(SEE) TO USE AS INDIRECT ADDRESS
GET ARGUMENT 1 WHICH IS ARRAY ADDRESS
DELETE BIT 0
STORE INTO c(ADR)
GET ARGUMENT 1 AGAIN
INCREMENT c(G02) TO FORCE THE FLOATING INTERPRETER TO BYPASS ARGUMENT 1
BIT ZERO ON? (CLEAR)
NO
YES
A5
** HAVE AN ARGUMENT2 **
INCREMENT c(SEE)
GET ARGUMENT2 WORD INTO c(AC)
MASK TO RETAIN SIGN BIT AND B
POSITIVE NUMBER ?
YES
NO
EXTEND NEGATIVE SIGN BITS
STORE INTO c(B)
GET ARGUMENT2 AGAIN
INCREMENT c(G02) TO FORCE THE FLOATING INTERPRETOR TO BYPASS ARGUMENT2
MASK TO RETAIN A
SHIFT RIGHT 6 BITS
STORE INTO c(A)
EVALUATE A*J+B-1
MOVE c(J) INTO c(MP2P) AS MULTIPLICAND
GET c(A) INTO c(AC) AS MULTIPLIER
IS c(AC) = 0 ?
YES
MP4P 12 BIT MULTIPLY A*J
ADD B
SUBTRACT A ONE
A10
STORE INTO c(B) FOR RETURN
LOAD AGAIN INTO c(AC) FOR RETURN
RETURN
SUBROUTINE GET
GET
FINT ENTER FLOATING INTERPRETOR
MOVE c(ANS) INTO c(FAC)
FEXT EXIT FLOATING INTERPRETER
RETURN

# PDP-9
# WORKSHOP

# FASBAC PDP-9
# TIME SHARING OPERATING SYSTEM

V. J. Zapotocky
University Computing Company
Dallas, Texas

## ABSTRACT

The U.C.C. FASBAC System provides for remote access to a general purpose file editing capability and a string handling programming language. Input files may be UNIVAC 1108 program files or data files which are to be submitted through direct access to run on the 1108. The time sharing operating system has been implemented on a 32K PDP-9 with a 524K drum and a specially built controller to allow sharing of a FASTRAND mass storage device with the 1108 and direct core-to-core transfers between the PDP-9 and 1108. This paper consists of a functional description of the PDP-9 operating system and some implementation problems which should be of common interest to PDP-9 users.

## INTRODUCTION

The time sharing operating system described in this paper comprises the heart of the FASBAC Remote Access System.

FASBAC provides the facility for large volume input and output processing on a UNIVAC 1108 data processing system initiated from remote terminals (Fig. 1).

The user's programs and data files may be accessed selectively for debugging purposes (to obtain, for example, a compiler diagnostic typeout) or directed to a high-speed printer connected to a COPE remote terminal subsystem for larger volume output.

A general purpose direct access file system, along with line and context editing capabilities, is provided to allow storing and editing of customer programs and data files. In addition, a string manipulative language called FASTRAC and an interpretive arithmetic expression analyzer called CALC are part of the FASBAC Remote Access System.

The PDP-9 Time Sharing Operating System has been implemented on the following hardware configuration:

32K PDP-9 with all processor options

524K Drum, 8.65 ms average latency

Special multi-access controller to allow sharing of UNIVAC FASTRAND with UNIVAC 1108 and provide direct core-to-core transfers between PDP-9 and 1108.

1000 card-per-minute reader

DEC Tapes (3)

Interprocessor Buffer for core-to-core transfers between PDP-9 and PDP-8.

## MAJOR SYSTEM COMPONENTS

### The Message Handler

All communication over the Interprocessor Buffer between the PDP-8 and PDP-9 is controlled by this program.

### Processor Scheduler

The queuing of all requests by various system facilities for use of the PDP-9 processor is handled by this program.

### Processor Dispatcher

This component is responsible for the orderly dispatching of control to the highest priority system facility currently requesting the PDP-9 processor.

### I/O Handlers

All input/output processing is controlled through standard device handlers providing queuing, where applicable, error detection and automatic retry.

### PDP-9/1108 Communications

The handling of all I/O between PDP-9 and 1108 is through a device handler which makes the 1108 look like a peripheral to the PDP-9.

### Service Functions

Certain service routines and status tables are essential in the operating system. They include an overlay loader, memory dump routine, trace facility, time keeping routines, a "CAL" handler, a system communications region, system macros, and a Logical Channel Table which provides status information about each active user.

File System

The File System provides a common interface for creating, retrieving, updating and deleting files on the system's permanent storage device which is the FASTRAND Drum. Both random and sequential files are allowed, and in fact either mode may be used to access any file created by the file system. A file security system is provided to maintain file protection against unwanted access.

Language Processor Scheduler

This component is responsible'for maintaining control over the language processors. (The term language processors refers to the FASTRAC and EDIT processors.)

Language Processor Interface

This component contains many subroutines common to the language processors and also provides a common interface with the file system and with various operating system procedures.

## BASIC DESIGN PHILOSOPHY

### Queuers and Doers

In order to provide efficient and equitable access to system resources, it is necessary to utilize a method for queuing requests for a given facility and a method for processing these requests.

The operating system contains routines called "Queuers" to provide a means for scheduling the use of system facilities. The operating system also contains routines called "Doers" which obtain their input from the queues, perform the required actions to initiate the use of the system facility, and make the required alterations to the queue itself. Queuers are generally activated by software calls; (except for certain processor dispatching) Doers are generally activated by hardware events. Examples of System Facilities requiring Queues:

PDP-9 Central Processor

PDP-9 Drum

FASTRAND Drum

PDP-9 to PDP-8 Output Processing

PDP-8 to PDP-9 Input Processing

Queuers and Doers possess the following characteristics:

1. Multi-priority level queues are provided.
2. Processing is first-in first-out within a given priority level.
3. The capability exists for deleting items from a queue.

Lockout:

Only one queuer or one doer may be actively manipulating a given queue item at any time. This is obtained by establishment of appropriate hardware and software priorities.

### Tells

Many operating system processes are required to enter a "waiting" state until some external event occurs before they can continue processing. Examples are 1) waiting for an I/O command to terminate before using requested input information, or 2) waiting for an overlay area to become free so a subsequent overlay may be read into the same area.

A "tell" is a means by which information is passed to inform a given process (actually via processor queuing) that an awaited event has occurred. The request for a tell is normally attached to a queued request and is activated by the satisfaction (completion) of its associated requested event. After telling, the doer (the processor dispatcher) initiates the next entry in its multi-priority queue.

A tell is not a process in itself, but rather is information associated with a queue entry which is to be passed on by the Doer of the queue to a destination queue upon completion of the requested action. The destination queuer is usually the Processor Scheduler.

Summary:

Three operating system scheduling facilities have been discussed to this point. They are:

Queuers

Doers

Tells

Queuers and Doers work together to provide optimum use of critical system hardware. Queuers make entries in queues according to priorities established by the programs which call them. Dequeuing must be also provided, as in removing outstanding I/O requests when a process has been aborted. Doers process items in the queues and remove items when appropriate. A tell is a software event similar to a hardware interrupt event.

Storage

Queue entry storage must generally be provided by the program calling the queuer. The program activated by the tell must arrange for release of the storage if it comes from a pool, as well as examine I/O status for error exception conditions (if appropriate).

A queue entry contains for example:

1. Forward link (used by queuer and doer).

2. Backward link (used by queuer and doer).

3. Status parameters, such as error count.

4. Tell parameters such as software priority level and program entry point (more fully explained later).

5. Job ID for accounting or aborting.

6. Other parameters as needed by told program.

Time-Out Lists

The operating system provides means for keeping track of time so that processes may be timed and "awakened" if required. An alarm time may be either a time interval or an absolute time of day.

Time of day and date may be read on request. Hardware interrupts caused by real time clock overflow causes control to be passed to a routine which scans the appropriate clock lists to determine if any clocks have exceeded their alarm time. If so, a tell function is then activated to notify a prearranged process that time has run out. By definition, the activated program must be core resident. The identification of the affected activity is made available to the processing program, which in turn schedules any necessary successors.

The time-out capability is especially useful in the following situations:

Notifying the language processor scheduler when a swap-out is in order for a particular user program.

Providing a means for notifying the system when a new process is to be initiated. (e.g., an accounting run is to be initiated at midnight.)

Memory Layout

The memory resources have been divided into three major portions:

1. Operating system, related functions and buffer areas.

2. FASTRAC processor and its buffer areas.

3. EDIT (or other processors) overlay area.

The operating system functions are basically core resident with overlay areas for bringing in routines which are infrequently used.

The FASTRAC processor remains core resident. It is used by certain operating system processes as well as by user programs. CALC, the calculation language runs essentially as a FASTRAC program, in a time sharing mode and is swapped in and out of memory. The Language Processor Scheduler initiates action to load waiting programs into assigned memory and to swap long-running programs. Swapping is performed by using standard routines provided through the operating system for physical I/O processing, queuing, etc. The Language Processor Scheduler is responsible for scheduling the FASTRAC processor when a FASTRAC job has been swapped in and is ready to execute. In addition, a timer is set, allotting a certain time quantum to the new FASTRAC user. If the time quantum is exceeded, a time-out occurs and a tell causes an entry to be made in the processor queue. The FASTRAC processor is allowed sufficient time to perform clean-up functions after which the user work area is swapped to the drum (swapping occurs only if other users are waiting in the Language Processor's queue for the FASTRAC processor).

The third area of memory is occupied by the EDIT processor, although this particular area is designed such that the processor occupying this space can be swapped to allow other language processors to use the area. The language processor is responsible for scheduling the use of the EDIT user area and assigning a time quantum.

Both FASTRAC and EDIT are designed such that the processors need not be swapped. Only their data areas (variable for each user) need by swapped. The FASTRAC processor is effectively re-entrant since it is allowed the opportunity to clean up prior to having its user area swapped. The EDIT processor has been implemented as an actual re-entrant program.

VRC Drum Layout

An area is set aside on the VRC fast-access drum for each possible simultaneous remote user. Two drum sectors are allocated for input file building. Teletype input is appended to this area. Overflow goes to FASTRAND mass storage. One sector is reserved for output information. Sixteen sectors (4096 words) are reserved for a swap area. The FASBAC System object programs (Operating System and Language Processors) are also maintained on the drum.

Maintaining Control over Remote Users

Each user who is logged on to the FASBAC system is assigned a logical channel number. This number is used as an index to a table of status information maintained in core memory. Included in this logical channel table (LCT) is the physical line number allocated to this user, user identification information, the language processor or system facility currently being used, and other information necessary to maintain control over a given remote user at all times. In order to allow referencing of information in this table, the address of the first word of a given logical channel table is frequently passed from one process to another and is a vital part of the tell process. The address of the first word of a given logical channel table is referred to as the LCTA for a given user. System macros are provided to allow:

Accessing information from the LCT

Storing information in the LCT

Setting indicators on or off in the LCT

Testing indicators in the LCT

## The Executive Functions

An underlying and somewhat unique philosophy has been used in the design of the "hard core" executive portions of the operating system. This executive consists of the Message Handler, Processor Scheduler, and Dispatcher.

These components have been designed to allow other portions of the operating system and the language processors to perform their work as though they are never interrupted except for the voluntary relinquishes as described above. In fact, it is possible for these service functions to alter the executive's procedures with respect to message handling in order to provide for the unique requirements of a given remote user. Message handling then can become flexible instead of a very rigid set of rules enforced from the executive end. This philosophy encourages additional capabilities being added to the language processors without major overhauling of these "hard core" executive functions.

The system component or language processor can in effect program (by means of software switch settings) the action to be taken by the Message Handler as well as the PDP-8 remote processor due to input from a user terminal.

The idea is for these executive functions to act as a good traffic cop (to perform a service without getting in everyone's way or making a nuisance of himself).

The following portion of this paper provides a more detailed look at those components which make up the executive portion of the operating system.

## THE EXECUTIVE FUNCTIONS IN DETAIL

### Processor Scheduler

The processor Scheduler is the Queuer of requests for the PDP-9 normal level processor (non-interrupt).

Level 9 and its Priority Scheduling - The term "level 9" is used to describe those activities which run at the normal processing level on the PDP-9. For example, the FASTRAC processor, the EDIT processor, the language processor interface functions, the file system, and parts of the operating system run at this level.

While the API and Program Interrupt features of the PDP-9 hardware provide the mechanism for the operation of levels 0 through 8, the scheduling and dispatching of control to level 9 processes is a service provided by the Executive. Provision has been made for 5 priority sub-levels within level 9. This number can be easily increased or decreased by a minor program change. The following level 9 priority assignments are used:

| Level | Use |
|---|---|
| 9-0 | Operating System functions associated with file building. |
| 9-1 | Other Operating System functions. |
| 9-2 | File System. |
| 9-3 | Language Processor Scheduler |
| 9-4 | FASTRAC and EDIT. Operating System clean-up tasks, and other low priority processing. |

### Current API (Hardware Priority) Assignments

| Level | |
|---|---|
| 0 | Power failure imminent, dead man |
| 1 | VRC Drum transfer completed |
| 2 | FASTRAND, DECtape, card reader, line printer |
| 3 | Interprocessor Buffer, clock |
| 4 | CAL handler |
| 5 | VRC Drum read/write initiation for message handling |
| 6 | Not used |
| 7 | Processor dispatcher |

### Current Program Interrupt Assignments

Paper Tape Reader/Punch

Console Teletype

Scheduling an Activity

There are 4 elements associated with a request for processor scheduling:

1. Priority of the requested activity.

2. Entry address (core memory location to which control is to be transferred.

3. Logical Channel Table Address (LCTA), described under "Maintaining Control over Remote Users".

4. Parameter pointer (to list elsewhere in memory).

Relinquishing the Processor

When an activity has run to its logical completion, or when some form of I/O is required before processing can continue, the activity executes a Relinquish macro.

Queuing the Processor Request

The Processor Scheduler accesses the four parameter words and the queue entry is made at the end of the queue such that a "first-in, first-out" relationship prevails within a given priority level. Priority 0 (zero) is considered highest and (for example) no priority 1 activity will be permitted until all priority 0 activities have been completed.

Format of the Processor Queue

The processor queue is a multi-level queue composed of entries built from a common pre-linked space pool. For each priority level there is a pointer to the first entry in the queue, and a pointer to the last entry in the queue. Also available at any time is a pointer to the next free entry in the space pool. At the time the queue entry is made the tell parameter words are inserted in the queue.

Processor Dispatcher

The Processor Dispatcher acts as the Doer on the queues built by the processor scheduler. Its function is to determine the highest priority activity currently requesting the processor and to dispatch control to that activity. The Dispatcher is called in the following two ways:

1. Each time a new activity is scheduled, a request is made by the Processor Scheduler for an interrupt at software level 7. When all higher priority processing has been completed, the Dispatcher program gains control at its entry point due to this earlier interrupt request. It determines if the newly scheduled activity is equal to or lower in priority to the one interrupted. If so, control is returned to the interrupted activity. If the new request is of higher priority, the registers of the interrupted activity are saved and control is given to the new higher priority activity.

2. When an activity has completed, it relinquishes the level 9 processor by executing the Relinquish macro. At this point control goes to the Dispatcher and the queue is searched for the highest priority waiting activity. If no activity was suspended, a new queue entry is selected. If an activity had been suspended and is now of highest priority, control is returned to the point of original interrupt.

Saving Control Information - The following registers are saved when level 9 control is switched.

1) AC
2) MQ
3) Auto index registers (10-17)
4) Program counter, link, extend mode, memory protect mode.

The four parameters associated with each entry in the system communication region at all times during execution of a level 9 program. When an activity is suspended, this information is also saved and restored at the appropriate time.

Dequeuing the Entry - Immediately prior to dispatching control to the entry address of an activity, the entry is dequeued as described in the section entitled "Format of the Processor Queue."

The Idle Loop - When there are no requests for the level 9 processor and no API or PI interrupt processing is active, the operating system enters an idle loop. This loop runs essentially as a level 9 lowest priority activity.

Message Handler

The Message Handler is the operating system component responsible for the processing of all messages going to or coming from the remote processor subsystem (PDP-8) via the interprocessor buffer.

Significance of User Mode - The PDP-8 and PDP-9 components of FASBAC must be aware of the processing mode in order for all required message handling logic to be handled correctly. The following four processing modes have been defined.

0) Line
1) Text
2) FASTRAC N (normal)
3) FASTRAC S (special)

Line Mode is used by the EDIT Processor. In this mode an input line which begins with a number (0-9) is automatically appended to the input file. Each line is transmitted from the PDP-8 to the PDP-9 with a "meta received" message type (the "meta character" or "end of message character" is a carriage return in line mode). A line beginning with a non-numeric character is assumed to be command information (see heading "Handling of Command Information").

FASTRAC N is the normal FASTRAC mode. In this situation input information is automatically appended to the input file. When a "meta received" message type is received, the message is appended to the input file and after completion of the append to drum or FASTRAND, the meta received entry (see "Language Processor Entry Points") is scheduled for the FASTRAC processor.

FASTRAC S is used when a one character input message is anticipated from the user. This character is transmitted to the FASTRAC processor through memory and no appending to file on VRC drum is required.

Text is used to build a file where all characters typed on the terminal become part of the text of the file. No line deletes or character deletes are allowed and no meta character is recognized. This mode is terminated by use of the break key from the terminal.

Normal file building to VRC Drum or FASTRAND occurs up to the time the break is depressed. The last message typed up to the break character is also appended to the file and the "break received" entry for the appropriate EDIT process is scheduled. Thus, the language processor may exercise certain types of control over the PDP-8 remote processor and the responses of the PDP-9 message handler by setting the proper mode for a given logical channel.

Language Processor Entry Points - The term language processor or "system" is used to describe the

FASTRAC Processor

EDIT Processor

Each logical channel connected to FASBAC has associated with it a particular system. Five different events cause the active system to be scheduled by the operating system. These events are:

1) Input message received
2) Break received
3) Meta received
4) Command Analysis required
5) "Send" has been completed (remote message has been typed on user's console)

Each system may specify its unique entry points for receiving control upon detection of an appropriate event by the operating system. For example, FASTRAC may wish to receive control at a location called %BROKE (a global symbol) upon detection of a "break received" message while EDIT may wish to receive control at a location called %BREAK for the same event. Each system has its own jump vector associated in numerical correspondence to the five common events mentioned earlier. The operating system computes the correct entry point by accessing the specified system for a given logical channel and scheduling the appropriate language processor activity.

Handling of a Large Input File - Storage for approximately 1000 characters is available on the VRC drum for building an input file from a teletype. Two sectors containing space for approximately 500 characters each are permanently assigned for each logical channel.

When sufficient input has been received to fill the first sector, the operating system switches over and begins appending to the second sector. It then schedules a request to open a temporary file on FASTRAND and writes the first drum sector to FASTRAND. This procedure is essentially repeated as many times as is required to complete the input file building procedure. (The only file size limitation is that imposed by the file system). If the input file does not exceed two drum sectors, there is no need for a FASTRAND file. In the situation where more than one sector, but not more than two sectors, are required to hold the input file, the FASTRAND file is "killed" and the language processor is informed that the file is contained on the VRC drum.

Handling of Command Information - When the user is processing in line mode the operating system performs a check on the first character of each line received from the PDP-8. If the first character is numeric (0-9) the line is appended to the input file. If the first character is non-numeric the command analysis entry is scheduled for the current active language processor. In addition, the input line is written to the VRC drum sector normally used for output message buffering (no output may be sent concurrent with input file building). This strategy allows the operating system to dispose of the input message, schedule a lower priority task to make a detailed analysis of the "command" line, and immediately be in a position to begin processing another input message. The language processor's command analyzer determines if the input line is a legitimate command. If so, the operating system must determine if the input file is totally contained on the two VRC drum sectors or if a FASTRAND file has been created. This information is passed to the language processor and processing of the requested command begins.

If the command is not recognized, a "WHAT?" message is sent to the user terminal.

Message Acknowledment - Each message received by the PDP-9 from the PDP-8 contains a transmission number. The message is examined for format and validity and if no errors are detected, an acknowledgment is sent to the PDP-8. This acknowledgment contains the transmission number of the PDP-8 message. The PDP-8 in like manner acknowledges receipt of all PDP-9 initiated messages.

If any errors are detected, a negative acknowledge message is sent. This results in a re-transmission by the originating processor.

Interface with the IPB Handler - All transmission over the Interprocessor Buffer is managed by a special handler which operates essentially like any other I/O device handler. The IPB handler allows for a "receive" and a "transmit" to be active simultaneously. Two buffers are available for handling IPB input and two also for sending output information to the PDP-8. Thus, it is normally possible to be processing an input message and simultaneously receiving into the other buffer; the same strategy applies to output processing.

All IPB message buffers are 128 words in length. In fact, all messages transmitted over the IPB are 128 words long. Due to header and trailer information, a maximum of 119 data characters can be sent in one message.

Output Message Handling - Two basic types of messages are required in the FASBAC System. One is used for sending data to remote terminals; these are called data messages. The second is used to control information to the PDP-8. This control information may pertain to a given remote channel or it may be of a general nature. These are called operational messages.

Output messages are initiated through system macros provided for this purpose. The execution of these macros results in the queuing of a message for transmission to the PDP-8. All operational messages receive priority over data messages. The Output Message Processor Routine acts as the doer on the remote output message queue.

The strategy of the Output Message Processor Routine (OMPR) is to keep the two output buffers ready to transmit. When a transmission terminates (detected by the IPB handler), OMPR gets an opportunity to initiate another transmission if a buffer is ready to send. Any time an output buffer is free, this routine attempts to fill it so that another transmission may begin as soon as the one in progress terminates.

Data Messages on Drum - A 256 word drum sector is reserved for output data for each logical channel. The language processors make use of this facility to store up to 510 characters of remote output (approximately 51 seconds of teletype output). Normally a given user will be swapped during the processing of this lengthy output request. At the time an output buffer is filled by a language processor, the information is written to the drum and the appropriate macro is executed. The Message Handler provides the facility for the building of interprocessor buffer messages, and transmitting these messages until the drum sector has been "emptied" without additional action by the language processor. At the completion of the processing of the output buffer, the language processor will be scheduled at its "send complete" entry with the logical channel number as an entry parameter.

The facility to send a message from core memory is also available. This type of message is limited to a maximum of 119 characters (for one IPB buffer) and the message area must be guaranteed to be present in memory when processed by the Output Message Processor Routine. This facility is primarily for standard messages such as the log-in, "WHAT", "READY", etc.

## OBSERVATIONS

### Debugging a Time Sharing System

Time sharing debugging presents serious problems in that it is often impossible to repeat the exact time-event sequence which causes a bug to appear. Several debugging aids have been implemented to help solve this problem. The best and most frequently used is a system trace facility. Each component of the operating system has been assigned a unique two character alpha-numeric. This code along with certain interesting variables (up to 6 locations) and the time of day is traced (total of 8 words) at strategic places within the system. This information is placed in memory in a circular fashion such that the last 64 system traces appear in core memory. In addition, a switch option allows tracing to DECtape or to drum. A post mortem trace listing provides a very helpful picture of system activity. Each 8 word trace appears as a line in the listing.

In addition, the old reliable mnemonic memory dump is used with both absolute core locations and a relativized reference for each subprogram available. The ASCII representation is also printed in a separate column.

It is also possible to display specified areas of memory on the console teletype and patch specified memory locations from the teletype while the system is in operation.

### Use of the Advanced Software

Heavy use has been made of MACRO 9, PIP and EDIT provided by D.E.C. during system development. Although DDT was useful in early debugging, it is not possible to use it in the time sharing environment. The Input/Output Monitor system is not used in the time sharing software, but many good ideas were used from its design. The Linking Loader has been modified and is used in our system generation procedure.

The Keyboard Monitor has been modified and can be run as either a VRC drum or FASTRAND resident system.

In summary we have made extensive use of the D.E.C. Advanced Software package, we have been very pleased with it and feel that we would not be nearly as far along as we are without it. The device independent features have been especially useful in helping us make use of the VRC drum and FASTRAND during development.

Implementation Problems

Our largest single problem has been one of memory size. We attribute this to two primary reasons.

1) A very limited instruction set means many memory locations must be used to get a given job done.
2) Lack of index registers is a serious handicap in any program development, but especially serious in time sharing work. Much space is used up in address calculations. Even one index register would be nice.

## ACKNOWLEDGMENTS

The system described in tnis paper represents the work of a group of individuals working under the leadership of Dr. D. W. Scott, Vice President of Research and Development, Computer Utility Network, University Computing Company. Those who made significant contributions include Paul J. Bell, Dennis Burke, Dwight W. Doss, George E. Friend, Elton K. Helwig, Cora M. Jones, Richard L. McDaniel, David Z. Polack, Frank D. Raunikar, Robert D. Ward, and Alex Zakson.

I am grateful for the typing and editorial assistance of Mrs. Velma Fruge'.

## REFERENCES

1. Scott, D. W. The FASBAC Remote Access System. DECUS Proceedings, Spring, 1968

2. Friend, G.E., Bell, P. J. The FASBAC System - Time Division Multiplexing and the PDP-8. DECUS Proceedings, Spring, 1968

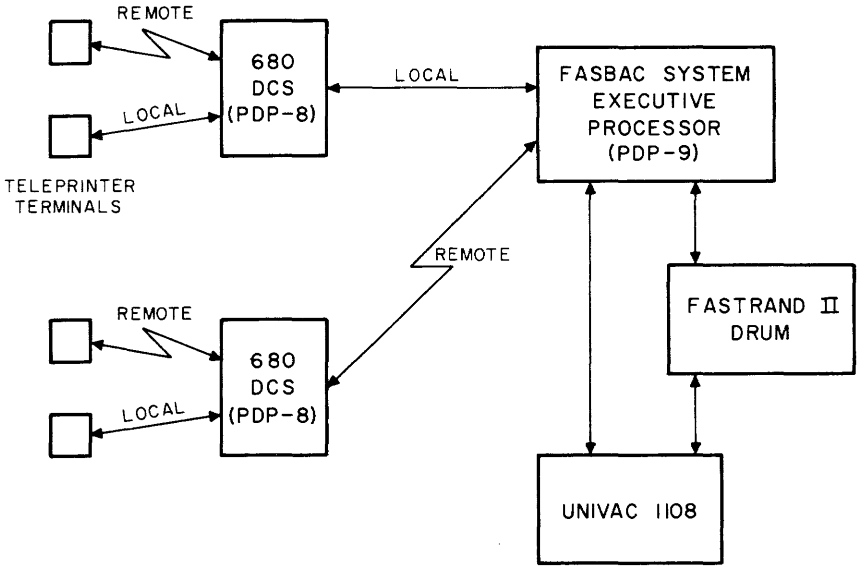


Figure 1. FASBAC System Block Diagram

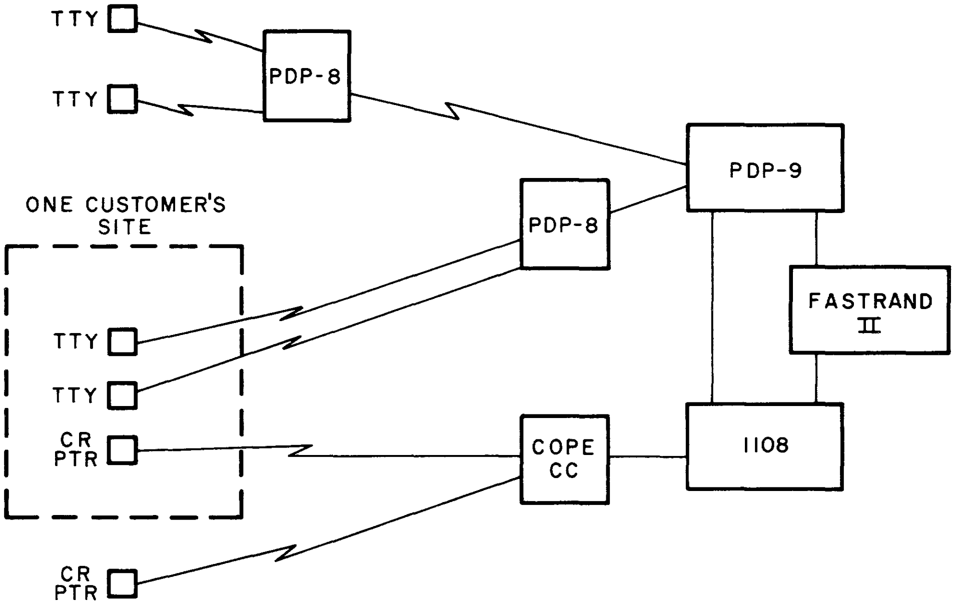


Figure 2. The UCC Direct-Access Computer Utility

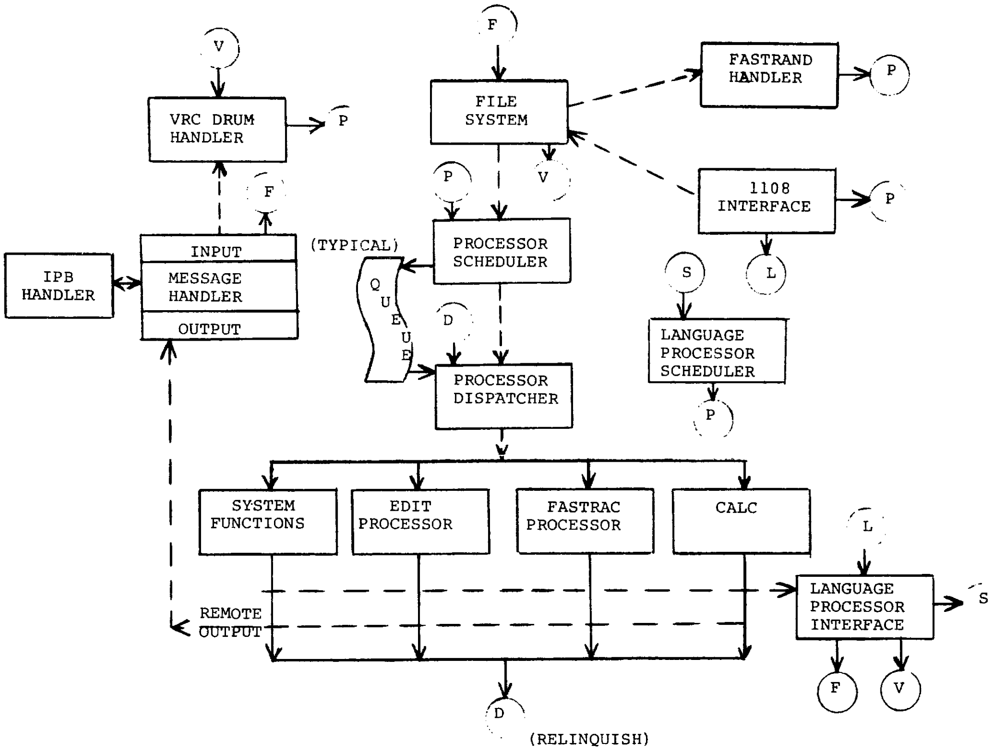


Figure 3. PDP-9 Operating System Flow Diagram

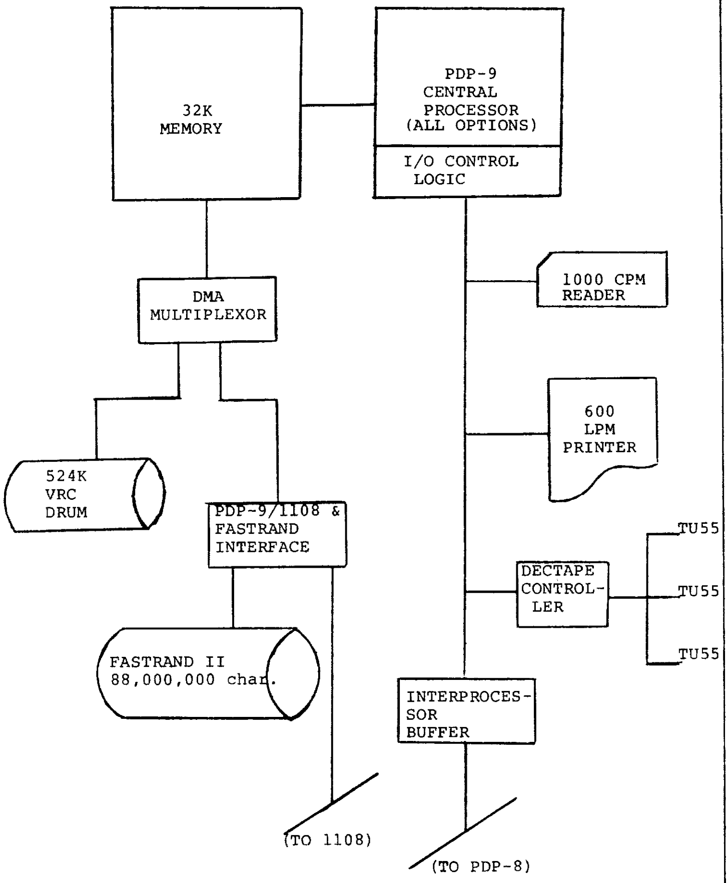


Figure 4. PDP-9 Hardware Configuration

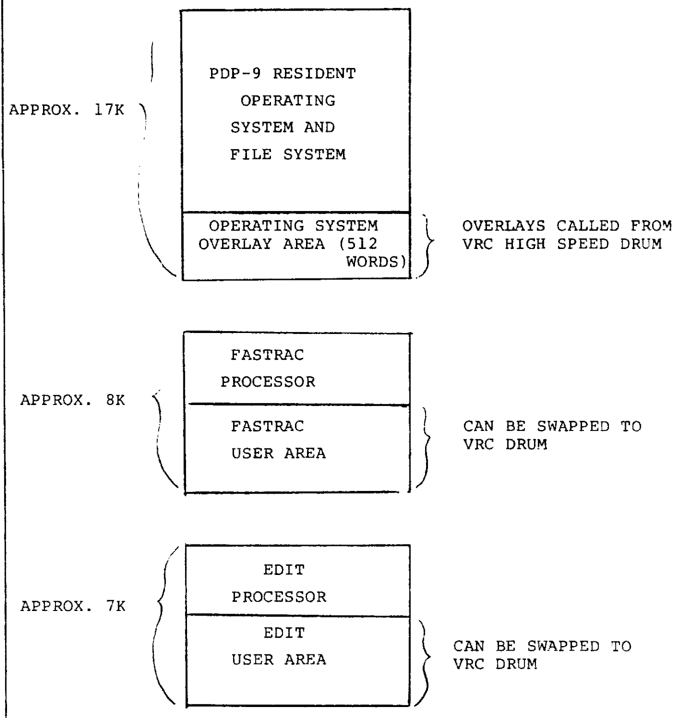

APPROX. 17K
PDP-9 RESIDENT OPERATING SYSTEM AND FILE SYSTEM
OPERATING SYSTEM OVERLAY AREA (512 WORDS)
OVERLAYS CALLED FROM VRC HIGH SPEED DRUM
APPROX. 8K
FASTRAC PROCESSOR
FASTRAC USER AREA
CAN BE SWAPPED TO VRC DRUM
APPROX. 7K
EDIT PROCESSOR
EDIT USER AREA
CAN BE SWAPPED TO VRC DRUM

Figure 5. PDP-9 Memory Layout

# PDP-9T TIME SHARING:

## Progress Report 12/68

M. M. Taylor
Defence Research Establishment Toronto
Downsview, Canada

D. M. Forsyth
Department of Psychology
University of Vermont

## ABSTRACT

At the end of 1968, the time-sharing hardware of the PDP-9T was not fully operational. Design changes in the operation of the XMR have been implemented, to permit ORing of the 6 bits of the extension of the XMR with the target address of any memory reference instruction found in the vector table other than JMP, JMS, or CAL. This provides the user machine with an additional form of memory, supplementary memory, which is available in 64-word blocks outside the 32K working memory. Another modification permits the monitor machine access to the current user's supplementary memory, and to supplementary memory tables of its own.

The minimonitor design and coding is essentially complete. It awaits completion of the hardware before it can be properly debugged and used. In the initial version, three 8K user machines reside in the 32K of core available in each of the delivered PDP-9Ts. The fourth 8K contains the minimonitor itself. With core to disc swapping of one of the user areas, the machine will be capable of supporting half-a-dozen conversational users using independent languages. The other two user machines each may support one or more real-time processes, with response latencies of the order of a few milliseconds. The DEC Advanced Software System will be available to all users, but with some of the restrictions on the use of large device handlers in 8K machines lifted. Users are not forced to use the ADSS, since they are provided with a simulated 8K PDP-9 with program interrupt. Essentially, any operation that does not require the full speed of the PDP-9 may be run within any of the user machines.

This paper reports the status of the PDP-9T hardware and software at the end of 1968. We assume that the reader is familiar with the previous reports on the PDP-9T, published in the proceedings of the last two DECUS Symposia (Fall, 1967 and Spring, 1968). The earlier of these papers presented the major elements of the hardware design and the overall plan of the time-sharing monitor system eventually to be implemented, while the second paper presented the outlines of the interim minimonitor software. The present paper is intended to be read as an updating of these two reports, rather than as an independent entity.

## HARDWARE

### Supplementary Memory

The action of the XMR (Execute Monitor) command has been altered to permit the user access to 64-word blocks of memory totally outside his 32K work memory. All of the memory reference commands except JMP, JMS, and CAL will have their analogues in supplementary memory reference instructions, though indirect addressing in supplementary memory will be possible only by the use of Vector Service Routines (VSRs) which will be implemented only for certain commands.

The revised action of the XMR command is shown in Fig. 1. The XMR has an operation code field of 4 bits, an 8-bit function or address field, and a 6-bit extension or microcode field. If the machine is in user mode, or if the command MVEC or UVEC (see below) has just been executed in monitor mode, then the operation code 70, normally associated in the PDP-9 with IOT, becomes an XMR. The function field then selects a word from the appropriate "Vector Table"; the selected word is executed as an instruction, except that the contents of the extension field of the XMR is ORed with the last 6 bits of the effective address of the instruction found in the vector table. This ORing does not happen if the

[*] This is DRET Tech. Paper #733.

command is JMP, JMS, or CAL, which operations remain as described in the earlier report. The action is like that previously described (Taylor, Forsyth and Seligman, 1967) for the case when IOT is foundinthe vector table.

As an example of the use of supplementary memory, suppose that the command XMR 12345 is issued by the user, and that location 523 (400+123) of the monitor machine contains LAC 4000; the instruction actually executed is LAC 4045, where the target address is in the monitor machine. Supplementary memory thus supplies a small region of readily accessible memory which can be used without altering the contents of the virtual core of the user machine. By changing the target addresses of the instructions in the vector table, one supplementary memory table after another can be made available to the user. Such changes must be made at the activation of each new user, and whenever a user demands to open a new supplementary memory table.

One use of supplementary memory is to facilitate the use of pure procedure (1) bodies in read-only memory. In the original design of the PDP-9T, it was envisaged that the impure parts of shared procedures would be held in read-write pages of each user's work memory. There are several reasons why this approach is unsatisfactory in practice. It is inefficient in terms of core usage, in that the ratio of the sizes of the pure and impure parts may vary greatly. Routines with small pure parts are not worth writing since little if any oore is saved and system overhead is added in keeping track of the sharing operation. Routines which have small impure parts are worth writing, but lead to difficulties when more than one shared routine is used at a time, because the loader needs to know about addressing conflicts in all the sharing machines. If the PDP-9 had an index register, this difficulty would be much less severe.

The use of supplementary memory improves the situation, in that although the addressing is still absolute within the supplementary memory table, the table itself is outside the work memory; it is automatically changed for each new user and on a request from any user. Loading conflicts may be alleviated, if not eliminated, by the judicious use of supplementary memory tables.

A related second benefit of the use of supplementary memory is that the system can provide the user with small data areas which do not use any read-write page of core. This means that there may sometimes be no need for any read-write page of user memory to be in core during the execution of a shared routine. Most important, there is no need for one particular read-write page to be always in core when a particular pure procedure is active.

The modified XMR operation permits another variation in the use of the vector table. If the instruction found in the table is XCT, the 6 bits of the extension field are ORed into the effective address field of the XCT. This permits the XCT to address any of 64 different commands, some of which may be JMS, and some not. If the command found by the XCT is JMP, JMS, or CAL, the operation is exactly as if the command had been found in the vector table directly by the XMR; priority is raised to API level 4, and the jump performed. Hence, the use of XCT in the vector table provides entry to 64 Vector Service Routines (VSRs) which do not use the extension field for numerical parameters. If the extension field is needed for a numerical parameter of a routine, then that routine will still be accessed by a JMS placed directly in the vector table. Potentially, the user machine now has quick access to a great many more VSRs than was hitherto the case.

### Monitor Vectoring

A second change in the hardware, used in conjunction with supplementary memory, is the provision of monitor vectoring. This provides the monitor machine with the equivalent of the XMR command. If the monitor issues IOT 3121 (MVEC) or IOT 3161 (UVEC) the next instruction, which must be an IOT, is interpreted as an XMR. If UVEC was issued, the XMR operates exactly as in the user machine, but if MVEC was issued, the vector table is taken to start at location 0 instead of 400.

The provision of the MVEC and UVEC commands permits the monitor to access directly the user's supplementary memory, and to enter VSRs in the same way that a user does. Previously, to enter a VSR from the monitor machine required the use of an ISA command to activate API level 4, with the concomitant necessity of saving and restoring the accumulator. MVEC permits the monitor a rudimentary indexing capability which assists in handling the various control tables used in the system.

## SOFTWARE

The construction of the software is proceeding in two stages. The full system will be that designed largely by Strom, and described by Taylor, Forsyth and Seligman (1967). An interim system, intended to permit useful real-time work under a time-sharing regime, is a modified version of that described by Forsyth, Forshaw and Taylor (1968). It permits several simulated 8K PDP-9s to run "simultaneously" and independently. A whole simulated machine must be entirely in core in order to run; this "minimonitor" does not permit page-turning.

A basic decision valid for both the minimonitor and the full system is that the DEC Advanced Software System (ADSS) will at all times be available to the users. This decision requires that the user machines include a simulated programme interrupt. The implementation of the minimonitor is thus somewhat different from that described in the earlier report, in that virtual machines, rather than tasks, are the natural unit of user programming. The current design, which has been coded and is now being debugged along with the hardware, is due largely to R. Walton of Princeton University and to R. Strom of Harvard.

The initial minimonitor is intended to run on the PDP-9Ts at Harvard and DRET. Both these machines have 32K of physical core at present. The minimonitor uses 8K, and runs 3 in-core simulated 8K machines. Early revisions to the minimonitor should permit core to disc swapping of one of the user areas, thus providing adequate service at conversational teletypes for 4-8 users, while still maintaining service latencies of a few milliseconds for users in the remaining two areas. Another anticipated early revision should relax the 8K boundary size, thus permitting more smaller machines to be locked into core without disturbing either the rapid response time of the locked-in machines or the use of the conversational terminals on the machines being swapped to and from disc.

A standard 8K PDP-9 cannot handle Fortran or Macro-9 when DECtape is used for both input and output, because of the size of the device handlers. The 8K user machines under the minimonitor do not have this problem, since the handlers are simply XMR calls to routines held in the monitor area, and thus take very little space in the user's 8K work memory. These XMRs are activated directly by the .READ and similar calls to the IOPS system of the ADSS. This type of operation applies to all those devices which users think they alone have, but which are actually shared, such as the DECtapes, Paper Punch and Reader, and the Line Printer. Devices which do belong uniquely to a user, such as his teletype or a special interface, normally have their handlers in his space and the actual IOTs become XMRs which are interpreted by the monitor. On the DRET machine, one user at a time may have the 340 display, which he may use just as if he had an 8K machine with a 340.

The scheduling algorithm is not uniquely determined within the minimonitor, and many scheduling variations are possible. Initially, users are scheduled on a round-robin basis, every user who has work to do being given 2 msec of computing time before being deactivated. If a programme interrupt has come up since the start of his last computing period, he starts at location 1, with the return information stored at 0, just as if a natural PI had happened in his simulated PDP-9 at the moment when he was last deactivated. He can test simulated device flags by using the same IOTs (which become XMRs) that he would have used in the standard PDP-9. Programmes written for a PDP-9 without API and which do not use cycle counts for timing purposes should run unmodified on the simulated machines under the minimonitor. The simulated real-time clock should tick every real 1/60 sec in location 7 of each user machine in core, so that real-time programmes should be no less precise in the simulated machines than in the standard PDP-9. In addition, the initial version of the minimonitor permits real-time operations to occur at times pre-specified to the nearest 10 msec.

In the ADSS system, the user must be able **to communicate with** his own user programme and with the monitor, using the command teletype. When he initiates the system, he communicates enough with the monitor to bring in a system programme or his own programme, which then takes over control. Typing CTRL/C puts him back in communication with the monitor. In the time-sharing system, another level of communication is necessary. The user must communicate with the time-sharing monitor as well as with the ADSS monitor. The TS monitor allocates resources, in particular processor power, to the user, and at the very least the user must make his presence known to the system before he can begin work. He may also want to communicate with the TS monitor at other times. To accommodate this, typing CTRL/A returns control to the TS monitor, as CTRL/C does for the ADSS monitor. In addition, CTRL/B bootstraps a copy of the ADSS monitor into the user machine, thus eliminating the need for a paper tape bootstrap.

## FOOTNOTE

[1] Any routine may be divided into those locations which change during the execution of the routine and those which remain fixed throughout. If the unchanging locations are grouped together, they are called the "pure" part of the procedure. The rest, by changing locations, are collectively called the "impure" part of the procedure. In a time-sharing system, considerable saving of real core may sometimes be gained by writing commonly used routines as "pure procedures", that is to say, by deliberately separating the pure and impure parts of the procedures, and sharing the pure parts in a read-only region of memory common to the users who "simultaneously" require the procedure. Each user then has his own copy of the impure parts of the procedure.

## REFERENCES

Taylor, M.M., D.M. Forsyth and L. Seligman. PDP-9T: compatible time-sharing for the real-time laboratory. Proceedings of DECUS Symposium, Fall, 1967.

Forsyth, D.M., S.E. Forshaw and M.M. Taylor. PDP-9T Time-sharing: Phase 1. Multiprogramming. Proceedings DECUS Symposium, Spring 1968.

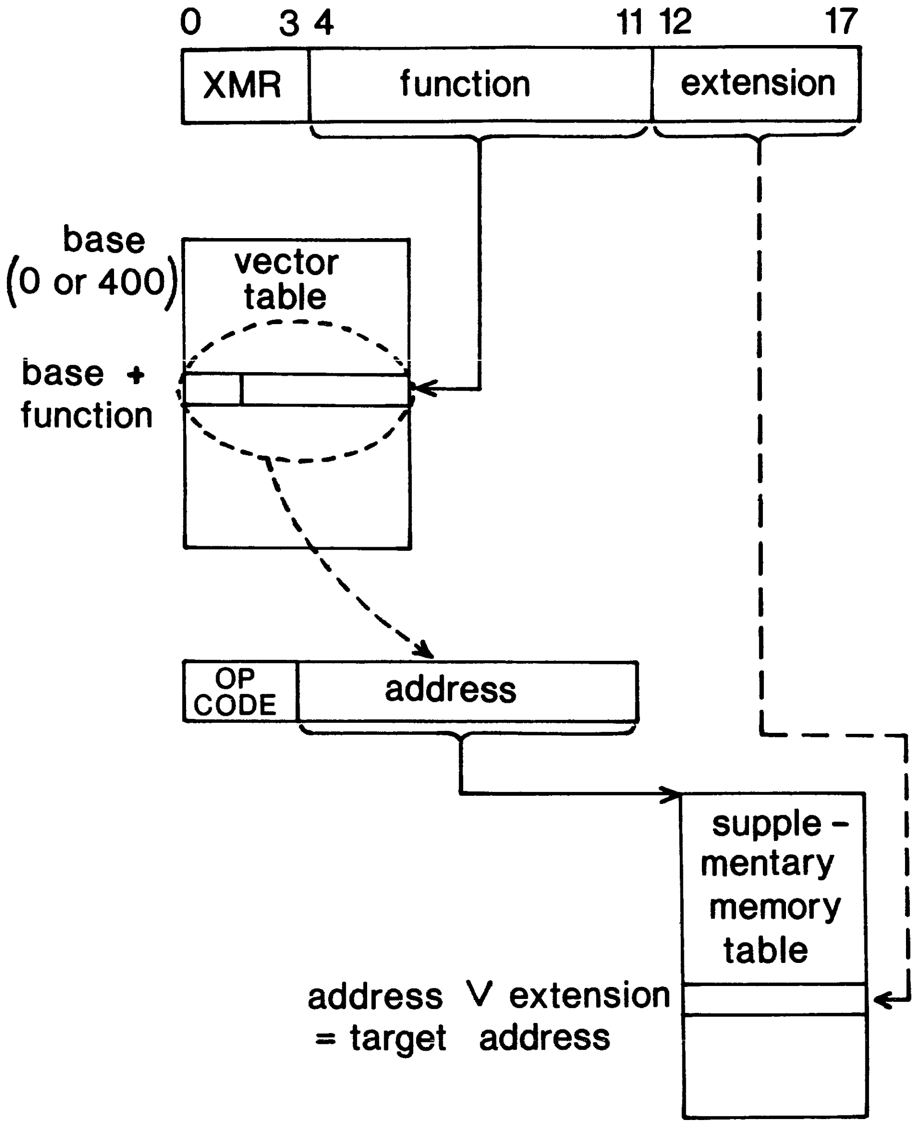


Fig. 1. Modified action of the XMR command when the instruction found in the vector table is not JMP, JMS, or CAL. The 8 bit function field finds an instruction in the Vector Table. This instruction is executed, but the 6 bits of the extension field of the XMR are ORed with the last 6 bits of the effective target address of the vector instruction to determine the final target address. In practice, the vector instruction defines the base of a supplementary memory table, while the extension field of the XMR defines the target address within that table.

# OPERATING THE KEYBOARD MONITOR SYSTEM FROM A DISK

C. Wendell Richardson
Phillips Petroleum Company
Atomic Energy Division
Idaho Falls, Idaho

## ABSTRACT

The Disk Monitor Program is designed to allow efficient use of a disk or drum by the Keyboard Monitor System. A basic PDP-9 with DECtape and any size drum or disk is sufficient to operate the monitor. DECtape is used for permanent storage. There is no need for protected or reserved areas for system programs since only those programs being used need be on the disk. System and user programs and user data sets are transferred from the DECtape to the disk for fast access by the computer. The time reauired for such tasks as program editing and compiling can be reduced by a factor of 10.

## THE SYSTEM

The Digital Equipment Corporation has developed a significant amount of software (The PDP-9 Advanced Software System) for the PDP-9 computer. The system includes a FORTRAN compiler, a MACRO Language assembler, a peripheral interchange program, a system generator, a linking relocatable loader, two monitor systems, and others. One monitor is used on the basic PDP-9 without DECtape or other bulk storage. On configurations with at least DECtape a more versatile monitor (The Keyboard Monitor) is used. It will execute commands from the keyboard to load system programs into core with designated device handlers and will provide communication between the programs and the handlers. The monitor allows complete device independent programming.

The system generator can be used to create new systems to fit any configuration. Any bulk storage device may be used as the system device provided suitable handlers are available. On configurations with both DECtape and high speed auxiliary storage (disk) there is a question as to which one should be used for the system device. The space which must be dedicated to the system and the time required to access the system programs are the main factors to be considered.

At least 400 blocks (256 words per block) are required for the system since there is no provision for deleting unused parts. The use of 400 dedicated blocks for the system is acceptable on DECtape because the amount of storage medium is unlimited. However, that amount is serious on smaller disks and prohibitive on a 512 block drum.

Even though the system uses DECtape very efficiently the time required for access is undesirable especially during program development. It was determined that 70% of the time required to update, assemble, and load a program is DECtape access time. A high speed device can reduce the access time by an average of 90%. Thus, operating the system from a disk is considerably faster than from DECtape.

In February 1968 we received a PDP-9, two DECtape units, 131-K word drum, and a 339 display scope. Soon after beginning our program development we realized that operating the system on DECtape was much too time consuming but the drum was too small to be the system device. We solved the problem by developing a program which we named the Drum Monitor. The drum monitor together with five drum device handlers, a drum bootstrao, and the advanced software programs make up the system.

The drum monitor was designed to use DECtape for permanent storage of programs and the drum for temporary storage and high speed transfers between the drum and the computer memory. The user first initializes the drum with <u>only</u> those programs he intends to use. When he has completed his operation the complete drum image can be saved and later rèstored from DECtape. Any system or user program can individually be added to or deleted from the drum. Hence, the entire drum is available for each user and no portion must be dedicated or protected.

The time required to save or restore the drum image is less than the time needed to load one program from DECtape. Thus, both the time and space problems have been solved.

Some of the capabilities of the drum monitor are as follows: 1) One block of the drum is reserved for an index. When a data set is written on the drum its name, size, and location are entered in the index. Upon request the monitor will print on the teletype the information contained in the index. 2) The system and user programs can be loaded into core from the drum for execution. 3) Up to 8 different core images can be dumped on the drum and restored from the drum. 4) The entire drum image can be dumped on DECtape and restored from DECtape. 5) System programs, user programs, and user data sets can be individually transferred to and from DECtape. 6) The keyboard monitor can still be used to execute commands from the keyboard. However, the keyboard monitor is now loaded into core only by the drum monitor.

## THE DRUM HANDLERS

Five drum handlers with varying capabilities have been written for the system. They are designated as DRA, DRB, DRC, DRD, and DRE (see Figure 1).

The DRA handler is the most versatile and combines the capabilities of all the other handlers. It controls 3 files (input or output) simultaneously with all modes and functions applicable to the drum. The three files are channeled through the same 256-word buffer to conserve core storage.

The DRB handler also has a simultaneous 3-file capacity but does not allow any but the system IOPS (Input/Output Programming System) ASCII and binary data modes and the standard input-output functions. Its purpose is to be used in place of the DRA handler to conserve core storage when the special functions and modes are not needed.

The DRC handler is the smallest being designed for input only applications. It has a one-file capacity with all modes but has only the standard functions available.

The DRD handler allows all of the functions and data modes but only one file (input or output) at any given time. The one-file limitation gives it a much higher rate of transfer than the DRA handler has. Any number of sequential file references are allowed.

The DRE handler allows all of the functions and data modes and has a simultaneous 2-file (input-output) capacity. Included are two 256-word buffers giving it the high rate of transfer of the D handler but making it approximately 256 words larger than the A handler.

## THE DRUM

The drum monitor was written specifically for a 131-K word drum. However, the system will work from larger drums or disks. The drum is divided into 256-word blocks. The blocks on a 131-K word drum are numbered from 0 to $777_8$. Block 0 is used for the drum index maintained by the drum monitor. The drum monitor bootstrap makes use of block $777_8$ for temporary storage purposes. Figure 3 shows a drum monitor printout of a sample index. Figure 4 shows the format of the drum index.

## DECTAPE

Three different types of DECtape are used with the drum monitor system. They are the drum monitor tape, the copy tape and the data tape. The copy tape is the only tape without a keyboard monitor system index. No DECtape backup is required once the drum is loaded with the necessary data. Only the drum monitor tape is necessary in initializing the drum. The copy tape and the data tape are used only for storage.

The following changes to the standard keyboard monitor tape are made to create a drum monitor tape: 1) The drum monitor program is dumped into the ↑Q dump area (see Section 3.2.3.5 of the PDP-9 Monitors Manual). 2) The system loader is altered (at system generation time) to load the linking loader from the drum rather than from DECtape. 3) The drum device handlers are assigned to most device assignment table slots (see Figure 2).

The copy tape is used to quickly store the entire drum image. The copy tape has no other use than to store the drum contents for future retrieval. The transfer is accomplished by a special double-buffered routine. The 512 blocks of the drum are copies directly into the first 512 blocks of DECtape. On larger drums or disks only 512 blocks would be stored on the copy tape.

The data tape may be used instead of/or in addition to the copy tape for permanent storage. Since all system programs exist on the drum monitor tape the storage may consist of only user programs and user data sets. Transfers between the data tape and the drum are identical to those between the drum monitor tape and the drum. The index of the drum monitor or data tape may be obtained with the peripheral interchange program.

## THE BOOTSTRAP

The keyboard monitor DECtape bootstrap handles many transfers between the computer memory and DECtape. The various kinds of transfers can be listed as follows: 1) The keyboard monitor from tape-to-core upon initial loading or upon unrecoverable errors. This transfer includes both resident and non-resident portions of the keyboard monitor. 2) All system programs except DDT, Chain, and the linking loader from the tape-to-core. 3) ↑Q dumps from core-to-tape. 4) ↑Q dumps from tape-to-core.

The drum monitor bootstrap handles only transfers between the drum and core. The various transfers can be listed as follows: 1) The drum monitor program from drum-to-core. 2) All system programs except DDT, Chain, and the linking loader from drum-to-core. 3) ↑Q dumps from core-to-drum. 4) ↑Q dumps from drum-to-core.

The drum monitor bootstrap differs considerably from the keyboard monitor bootstrap. The drum monitor bootstrap searches the drum index for the element to be transferred, whereas the keyboard monitor bootstrap loads all elements from absolute physical location on the bulk storage device. To make room for the additional instructions in the drum monitor bootstrap, 256 words starting at core location 10,000 are temporarily stored in block $777_8$ on the drum. The additional instructions for the bootstrap are then read from the bottom portion of block 0 at the end of the index into the locations starting at 10,000 and are executed. The bootstrap then restores the original instructions at location 10,000 from block $777_8$.

## THE INDEX

The format of the index entries is shown in Figures 3 and 4. The general entries use three words to specify the entry name. The first two words of the entry specify a six-character name which appears as six-bit ASCII in the index. The third word is used for a three-character extension to the name also in six-bit ASCII. The fourth word of an entry indicates the number of blocks used by the data and the starting block of the data. System programs can be identified by SYS as their extension. General entries will use SRC for alphanumeric information and BIN for data or binary information in most cases. Entries not using an extension of SYS, SRC or BIN must use the peripheral interchange program for any transfers. Additions to the drum index use the first available blocks. All entries use contiguous blocks. Deletions cause the drum to be repacked.

## CORE DUMPS

↑Q is a keyboard monitor command. If the keyboard monitor is being operated from DECtape a number is typed with the ↑Q to specify which unit the dump is to be written on. The dump is a complete core image into prespecified blocks ($101_8$-$141_8$) on the DECtape.

The keyboard monitor command GET is used to retrieve core dumps. If DECtape is being used a number is typed with the command to specify the unit. The keyboard monitor bootstrap handles the transfers between the bulk storage device and core.

The drum monitor bootstrap allows core dumps to be handled differently. The ↑Q command initializes the dump and the number specifies which of 8 possible dump areas is to be used. The numbers can be any between 0 and 7. Any number of dump areas up to 8 can be reserved on the drum at drum initialization time. Areas which have not previously been reserved should not be used.

## THE DRUM MONITOR COMMANDS

When the drum monitor is ready for the user to type a command it will print an asterisk at the left-hand side of the page. The return of an asterisk after receiving a command indicates to the user that the command has been successfully executed (see Figure 5.)

Index

Form: INDEX

Index prints the drum index on the teletype. Figure 3 is an example of an index printout. This command can in no way be used to print the index of DECtapes. The peripheral interchange program can be used to print the index of DECtape.

Prime

Form: PRIME

Prime causes the following items to be added to the drum: 1) the keyboard monitor, 2) the drum monitor, 3) system skip block, 4) the system loader, 5) the indicated number of core dump areas. The user is interrogated as to the number of core dump areas to be reserved on the drum. Each area takes forty 256-word drum blocks. Any number from 0 to 8 is acceptable.

This command also transfers the resident drum bootstrap instructions to the core.

Clear Index

Form: CLEAR

Clear will create a new drum index on the drum. Everything previously in the index is deleted.

Set Transfer Direction: Tape-to-Drum

Form: DRUM

The direction tape-to-drum is initially set when the drum monitor is loaded and remains set unless changed by the command TAPE. To reset the transfer direction to tape-to-drum the command DRUM is given. Transfers in either direction will replace any file with the same name. If it is desired to change a file name during a transfer, the peripheral interchange program should be used for the transfer.

Set Transfer Direction: Drum-to-Tape

Form: TAPE

The command TAPE is used to set the transfer direction to drum-to-tape. This direction will remain set until the DRUM command is given or until the drum monitor is reloaded. In order for this command to work the DECtape unit WRITE-ENABLE switch must be on. Transfers in either direction will replace any file with the same name. If it is desired to change a file name during a transfer the peripheral interchange program should be used for the transfer.

Copy Drum Image

Form: COPY

The COPY command transfers the entire drum image from drum-to-tape or from tape-to-drum depending upon the current transfer direction. The transfer will be to and from DECtape unit 1.

Transfer File

Form: xxxxxx yyy

Typing any file name and extension will cause the corresponding file to be transferred from drum-to-tape or from tape-to-drum depending on the current status of the transfer direction. The transfer will always be to DECtape unit 8. The file name and extension will be inserted in the index of the destination device but will also remain on the source device. Only the three extensions, BIN, SRC and SYS, are recognized by the drum monitor. Files with an extension of SYS may be transferred only from the DECtape to the drum. Files with extension of other than these three must be transferred with the peripheral interchange program.

Delete

Form: DELETE

When the DELETE command is given the user will be asked to type a name and extension of the file to be deleted from the drum. Any files existing below the deleted files will be repacked.

Load Keyboard Monitor

Form: K

The K command will cause the drum monitor to load the core with the keyboard monitor. The keyboard monitor can now be used for all functions except printing or creating a DECtape index. The DECtape index can be obtained only from the peripheral interchange program. All other aspects of the keyboard monitor remain the same.

Replace DECtape Bootstrap with Drum Bootstrap

Form: BOOT

When the drum monitor is not on the drum the DECtape bootstrap must be used for loading (see section on loading procedures). The BOOT command can be used to replace the DECtape bootstrap with the drum bootstrap after the drum monitor is placed on the drum from a COPY tape.

## THE LOADING PROCEDURE

It should be noted that when the user first sits down to use the computer he has no way to determine what is on the drum. Even if the drum index indicates the presence of certain programs, there is no assurance that the programs are there in their correct form. However, if the last user was using the drum monitor system it is fairly certain that the current programs are safe to use.

The suggested way to initialize the drum assuming nothing about the present contents is as follows:

1. Place the paper tape labeled "Drum Monitor Bootstrap" in the paper tape reader and set the ADDRESS switches to 000000. Push and release the hard-wire READIN switch.

2. If the words DRUM MONITOR are printed on the teletype the monitor should be ready to use and steps 3, 4 and 5 can be skipped. Under any other condition go on to step 3.

3. Mount the DECtape labeled Drum Monitor and dial to unit number 8. Place the unit switches on RIGHT-LOCK and REMOTE. Without changing the paper tape bootstrap position in the paper tape reader or the ADDRESS switches as they were left from step number 1, again push and release the hard-wire READIN switch. The second half of the drum monitor bootstrap will read in.

4. The teleprinter will type MONITOR. The regular keyboard monitor is now in core with the regular keyboard monitor bootstrap. Since the drum monitor is stored in the DECtape dump file, it can now be loaded with the keyboard monitor command GET.

5. The teleprinter will type DRUM MONITOR. At this point the drum monitor has been loaded and is ready to use. However, the keyboard monitor bootstrap rather than the drum monitor bootstrap is in core. This will cause all bootstrap transfers to be to and from DECtape rather than the drum. The drum monitor bootstrap may be placed in core in three ways. The commands PRIME or BOOT will place the drum monitor bootstrap in core or after the drum monitor is placed on the drum from a copy tape the drum monitor bootstrap may be reloaded as in step 1.

6. The drum monitor is now ready to use and any of the commands may be given.

## OPERATION SUGGESTIONS

Figure 6 is an example of the teletype record for an initial setup. Only those words that are underlined were typed by the user. The user followed steps 1, 2 and 3 of loading procedures to load the keyboard monitor from DECtape which then printed out the first line. The user typed "GET 0" to load the drum monitor from the dump file on the DECtape. The user next issued the PRIME command. At this point there are 4 system programs in the index. The next 4 commands were given to transfer the system library, the linking loader, the editor and the FORTRAN compiler to the drum. An INDEX was requested with the expected results. At this point the drum was initialized to meet his particular needs. The user next desired to reserve his drum image for future use. The TAPE command was given to set the transfer direction to drum-to-tape. The COPY command was given to write the complete drum image on DECtape unit 1.

Figure 7 is an example of how the user initializes the drum from a COPY tape. The single command COPY is used to transfer a complete drum image from DECtape to the drum. The system and user programs can now be operated from the drum or additional system and user programs can be readily added.

It is intended that most usually the drum monitor will be on the drum allowing the user to just load his COPY tape and type COPY to get started.

## THE SYSTEM PROGRAMS

The following is a list of required system programs for drum operation.

| Name | Index Entry | Command to Transfer to Drum |
|---|---|---|
| Keyboard Monitor | KM9 SYS | PRIME |
| Skip Block | SKPBLK SYS | PRIME |
| System Loader | SYSLD SYS | PRIME |
| Drum Monitor | DRUM M SYS | PRIME |
| Linking Loader | .LOAD BIN | .LOAD BIN |
| System Library | .LIBR BIN | .LIBR BIN |

The following is a list of optional system programs operable from the drum.

| Name | Index Entry | Command to Transfer to Drum |
|---|---|---|
| Dump Files 0-7 | DMPBLK SYS | PRIME |
| Editor | EDIT SYS | EDIT SYS |
| MACRO Assembler | MACRO SYS | MACRO SYS |
| MACROA Assembler | MACROA SYS | MACROA SYS |
| FORTRAN Compiler | F4 SYS | F4 SYS |
| Update | UPDATE SYS | UPDATE SYS |
| Execute | EXECUTE SYS | EXECUTE SYS |
| Patch | PATCH SYS | PATCF SYS |
| Peripheral Interchange | PIP SYS | PIP SYS |
| Dump | DUMP SYS | DUMP SYS |

DRUM HANDLERS

| NAME | MODES | SIZE | FILES |
|---|---|---|---|
| DRA | ALL | 1001 | 3 |
| DRB | IOPS | 872 | 3 |
| DRC | IOPS | 576 | 1 (input) |
| DRD | ALL | 937 | 1 |
| DRE | ALL | 1267 | 2 |

DECTAPE HANDLERS

| | | | |
|---|---|---|---|
| DTA | ALL | 2316 | 3 |
| DTB | IOPS | 1547 | 2 |
| DTC | IOPS | 674 | 1 (input) |
| DTD | ALL | 1564 | 1 |

Figure 1. DECtape and drum handler comparison.

```
C
DRUM MONITOR
*K
MONITOR V3C
$REQUEST

 DAT           DEVICE           USE

-15             DRE             OUTPUT
-14             DRE             INPUT
-13             DRB             OUTPUT
-12             TTA             LISTING
-11             DRB             INPUT
-10             PRA             INPUT
-7              DRC             SYSTEM DEVICE FOR .SYSLD
-6              NONE            OUTPUT
-5              NONE            EXTERNAL LIBRARY FOR .LOAD
-4              DRC             SYSTEM INPUT
-3              TTA             TELEPRINTER OUTPUT
-2              TTA             KEYBOARD INPUT
-1              DRC             SYSTEM DEVICE FOR .LOAD
 1              TTA             USER
 2              PRA             USER
 3              NONE            USER
 4              DTD1            USER
 5              DRD             USER
 6              NONE            USER
 7              NONE            USER
 10             NONE            USER
```

Figure 2. Device assignment table.

```
*INDEX

DRUM DIRECTORY
NAME                NUMBER              STARTING
                    OF BLOCKS            BLOCKS
.LIBR BIN            200                  001
.LOAD BIN            014                  201
DMPBLK SYS           040                  215
DMPBLK SYS           040                  255
DRUM M SYS           040                  315
KM9    SYS           036                  355
SKPBLK SYS           001                  413
SYSLD  SYS           011                  414
MACRO  SYS           027                  425
EDIT   SYS           012                  454
F4     SYS           031                  466
DUMP   SYS           004                  517
USERPG SRC           003                  523
251      FREE BLOCKS

*
```

Figure 3. Drum index.

| | | .SIXBT CODE | | |
|---|---|---|---|---|
| | | OR SPECIAL | | |
| PROGRAM | | SYSTEM PROGRAM | STARTING | |
| NAME | INDEX WORD | CODE | BLOCK # | # OF BLOCKS |
| SYSTEM | 561411 | .LI | | |
| LIBRARY | 022200 | BR | | |
| | 021116 | BIN | | |
| | 200001 | | 1 | 200 |
| LINKING | 561417 | .LO | | |
| LOADER | 010400 | AD | | |
| | 021116 | BIN | | |
| | 014201 | | 201 | 14 |
| DUMP | 021101 | BLK 101 | | |
| AREA O | 041520 | DMP | | |
| | 233123 | SYS | | |
| | 040215 | | 215 | 40 |
| DUMP | 121101 | BLK 101 | | |
| AREA 1 | 041520 | DMP | | |
| | 233123 | SYS | | |
| | 040255 | | 255 | 40 |
| DRUM | 560414 | .DL | | |
| MONITOR | 170104 | OAD | | |
| | 233123 | SYS | | |
| | 040315 | | 315 | 40 |
| KEYBOARD | 000000 | BLK 0 | | |
| MONITOR | 041520 | DMP | | |
| | 233123 | SYS | | |
| | 036335 | | 355 | 36 |
| SKIP | 000044 | BLK 44 | | |
| BLOCK | 041520 | DMP | | |
| | 233123 | SYS | | |
| | 001413 | | 413 | 1 |
| SYSTEM | 000056 | BLK 56 | | |
| LOADER | 041520 | DMP | | |
| | 233123 | SYS | | |
| | 011414 | | 414 | 11 |
| MACRO | 000742 | BLK 742 | | |
| | 041520 | DMP | | |
| | 233123 | SYS | | |
| | 027425 | | 425 | 27 |

Figure 4. Format of drum index.

| Command | Action |
|---|---|
| INDEX | Drum index printed on teletype. |
| PRIME | Create following items on drum:<br>1. Keyboard Monitor<br>2. System Skip-block<br>3. System Loader<br>4. Drum Monitor<br>5. The indicated number of core dump block. The user is interrogated as to the number of core dump blocks to be reserved on drum. Each block takes 40 256-word drum blocks. Any number from zero to seven is acceptable. |
| CLEAR | Create new drum index. |
| DRUM | Set transfer direction: tape to drum. |
| TAPE | Set transfer direction: drum to tape. |
| COPY | Copy entire drum on tape or retrieve drum image from tape depending on current transfer direction. Transfer is on DECTAPE unit 1. |
| xxxxxx yyy | Transfer file name xxxxxx extension yyy from drum to tape or from tape to drum depending on current status of the transfer direction. The DECTAPE unit must be dialed to 8. The file name xxxxxx with extension yyy will be inserted in the destination device index. If it is already in the index the old file will be deleted. |
| DELETE | User will be asked to type the name and extension of file to be deleted from drum. |
| K | Load core with keyboard monitor |
| BOOT | Replace DECtape bootstrap with drum bootstrap. |

Figure 5. Summary of drum monitor commands.

```
MONITOR V3C

$GET 0

DRUM MONITOR

*CLEAR

*PRIME

TYPE NUMBER OF CORE DUMP BLOCKS
0

*.LIBR BIN

*.LOAD BIN

*EDIT SYS

*F4 SYS

*INDEX

DRUM DIRECTORY
NAME                NUMBER               STARTING
                    OF BLOCKS              BLOCKS
DRUM M SYS            040                  001
KM9    SYS            036                  041
SKPBLK SYS            001                  077
SYSLD  SYS            011                  100
.LIBR  BIN            200                  111
.LOAD  BIN            014                  311
EDIT   SYS            012                  325
F4     SYS            031                  337
407       FREE BLOCKS

*TAPE

*COPY

*
```

Figure 6. Initial drum setup.

```
DRUM MONITOR

*COPY

*INDEX

DRUM DIRECTORY
NAME                NUMBER                STARTING
                    OF BLOCKS               BLOCK
DRUM M SYS            040                   001
KM9    SYS            036                   041
SKPBLK SYS            001                   077
SYSLD  SYS            011                   100
.LIBR  BIN            200                   111
.LOAD  BIN            014                   311
EDIT   SYS            012                   325
F4     SYS            031                   337
407        FREE BLOCKS

*
```

Figure 7. Day-to-day usage of the drum.

# PDP-9 MONITOR SYSTEM WORKSHOP

David Leney and James Murphy
Digital Equipment Corporation
Maynard, Massachusetts

## ABSTRACT

This lecture, discussion session, and demonstration is directed towards the presentation of major new developments in the PDP-9 ADVANCED Software System and towards the solution of existing trouble areas of general concern.

The new developments include:

1. Background/Foreground (two user time-sharing) Monitor System.

2. Multi-user FOCAL system which may operate as the Foreground or Background job under control of the B/F Monitor System.

3. The 339 Software Package.

On Friday and Saturday, December 13 and 14, 1968, the PDP-9 equipped with 32K of core memory, API, EAE, Memory Protect, and DEÇtapes along with knowledgeable DEC personnel was available for the purposes of problem solving and specific demonstrations.

Three major developments in the PDP-9 Advanced Software area are:

1. Background/Foreground Monitor System which is a two user time sharing system. The Foreground job is quite applicable to real-time work and is completely protected from the Background job, which is anything currently available under the Keyboard Monitor System.

2. FOCAL language within Advanced Software

3. 339 Graphics package

## BACKGROUND/FOREGROUND MONITOR SYSTEM

The prime goal is to give the user more of the available CPU time to accomplish his work. With the Keyboard Monitor System a statisical analysis would show that anywhere from 60 to 80% of the time the system is I/O bound (waiting for some I/O to complete so processing can continue on). With the Background/Foreground System, control goes to the Background job to accomplish something useful if the Foreground is I/O bound. The way we define the system is that the Foreground job is a checked out user program, which is quite applicable to real-time processing, and this program will have complete priority over core memory, peripherals that are available, and CPU time. The only time it gives control to the Background job is when it is I/O bound. This program is completely protected from the Background job which could be a non-checked out program, which the user might be debugging at this time. It is protected by hardware with a hardware memory protect option which won't let the Background job reference the core of the Foreground area. It is also protected software-wise by the Background/Foreground Monitor System. All I/O that the Background job does is accomplished via the CAL instruction identical to the current method in the Keyboard Monitor environment. The Monitor will verify all addresses and word counts that the Background job is passing on before it allows the actual operation to take place. The Time-Sharing Algorithm is that control does not go to the Background job until the Foreground job becomes I/O bound. When control goes to the Background job, it keeps control until the Foreground job's I/O, which caused the Foreground job to become I/O bound, is completed.

There are two sections to the Background/Foreground Monitor System which control this operation:

1. The CAL handler takes control from the Foreground job in the I/O busy situation;
2. Each of the I/O device handlers, upon completion of a Foreground I/O operation which caused an I/O busy situation, will return control to the Foreground job.

One of the nicest features of the system, especially to people familiar with the size of DECtape handlers, is that both users can utilize the same DECtape handler. Currently, we have four versions of the DECtape handler and four versions of the DISK handler. They vary according to the number of data modes that each handler will process, and the number of functions each will perform.

In the Background/Foreground Monitor System there will be one handler. Its called the N-file handler. The buffer space required is built dynamically at run-time and both users can use the same basic handler. The buffers that the handler requires for each job are in the appropriate job area. The way it works is that the Foreground job (even if there is Background I/O underway, say for DECtape) will push its way in, stop the Background I/O so it won't delay the Foreground real-time environment, and process the Foreground I/O request. Upon completion of the Foreground job's I/O, the Background job's I/O will be restarted again.

## HARDWARE REQUIREMENTS FOR BACKGROUND/FOREGROUND SYSTEM

To run the Background/Foreground Monitor requires at least 16K of memory, the memory protect option, bulk storage device (DECtape or DISK), and LT19 teletype multiplexer with at least one additonal teletype. One teletype is the Background control teletype and this is the normal console teletype. The other teletype on the LT19 is the Foreground job control teletype.

One of the options which brings more power to the system is the automatic priority interrupt. Without the API, the Foreground job runs at the mainstream level with its I/O at the program interrupt level, and the Background job also runs at mainstream, with its I/O at the program interrupt level. With the API, the Foreground job gets 3 additional levels of priority (software levels 5, 6, & 7). The Foreground job can utilize all these levels. Also, control does not go to the Background job until the Foreground job is I/O busy at all levels it happens to be using at this time.

## MONITOR CONTROL FUNCTIONS

There have been two major additions to the Monitor control functions. The .TIMER Macro has been expanded to allow many intervals to be going on at the same time, so that the clock can be used by both Background and Foreground and can be used many times by each job. The address that is specified for the completion of the interval can also have a priority level attached to it. So this gives you all kinds of program priority capabilities. Now we also have real-time .READs (.REALR) and .WRITEs (.REALW) so that, rather than waiting for completion of I/O with a .WAIT, an address can be specified with a priority level for I/O completion (like .TIMER). The API brings these three additional levels, so there could be four Foreground programs running at different priority levels as all the internal workings are there to allow this to happen quite easily.

Question: With these multiple calls to the clock, does that mean that each time you get an overflow it will reset the clock and then transfer control to the user address?

Answer: As a .TIMER request comes in, the question is asked "Is there an interval going on at the moment?". If there isn't, this request gets stuck in the actual clock register (register7 ). If there is one going on, the decision is made as to if this one is shorter than the current one. If it is, the shorter one gets stuck in the clock register, and all the other entries in the queue are updated, based on the current interval. The clock doesn't interrupt every 1/60th of a second; it depends on the smallest interval that has been requested at this time.

Question: How big is the clock Queue?

Answer: This will be a system generation variable. The one running at the Fall Joint had 20 slots in it for the clock intervals. You can specify as large a queue as you like.

Question: Can you use the system without adding the teletype multiplexer and 2nd teletype?

Answer: It's possible that you won't have to add it. The reason we requested it is because we feel the API is a strong option in this system and that you would want an external device that was connected to API. The console teletype is only on the program interrupt, so that there is no way you could break into your program if you saw an error condition while you were running at API software levels. The console teletype would not get service. The LT19 is an API device.

Question: How do you determine if the program is I/O bound?

Answer: The way a program becomes I/O bound is that .READ or .WRITE operation actually runs into a .WAIT or another .READ or .WRITE to the same .DAT slot. It then becomes I/O bound. If the program user saw fit to use .WAITR, then he would keep control and go to the busy address of the .WAITR. So a Foreground program could keep control completely. But this is a function of your installation. If you want to utilize the Background, you shouldn't keep control forever by using .WAITR in the Foreground.

Question: Is the Foreground area actually fixed?

Answer: No. The hardware memory protect option on the PDP-9 allow you to move the bound in 1K decimal increments. What happens is that the Foreground job gets loaded first, and the rest of core is for the Background. In addition, there is a FCORE command which allows you to specify the amount of additional space (free core) you want to give to the Foreground job.

Question: Can the Background/Foreground jobs communicate with one another?

Answer: Yes. We have a core to core handler, which looks like any other I/O device. It accepts Monitor .READs and .WRITEs. One time you could be using this handler and another time DECtape. When a job requests a core to core transfer, the request won't get honored unless there is a complimenting request from the other job. The Foreground program could be passing live data (output) to the Background job as test data (input).

Question: Please discuss operation procedures; that is, how do you get started?

Answer: The Foreground job gets loaded first in Linking Loader fashion. You can specify the program you want to load. It gets loaded. The protection registers get set. The Background Monitor comes in, which looks like the keyboard monitor as it is now. Now both jobs are running. If you hit Control C on the Background control teletype, the Background job ends and Background monitor comes in, just like under the keyboard monitor. If you hit Control C on the Foreground teletype, it does the same thing, also killing the Background job. If the Foreground job completed naturally, then the Foreground user has the option of hitting Control C to kill the Background job, or let the Background job continue to completion.

Question: What kind of restrictions are there for the real-time loader? Will it load a main program and all its associated subroutines, or must the Foreground job be one program?

Answer: It is essentially the Linking Loader as it stands now. It will load whatever you specify in the command string, plus any library routines required.

Question: Can you run Fortran as the Foreground job?

Answer: You cannot compile Fortran programs, but you can run the Fortran object time sytem.

Question: Would a program be considered I/O bound if it requested a .WAIT for a software level?

Answer: There is no .WAIT for the software levels. Only the real-time reads and writes or the .TIMER requests have software level specification. These will not give control to Background. Once the real-time .READs, .WRITEs, or .TIMERs have been initiated, a .IDLE (equivalent of JMP.) can be given to give control to the Background.

Question: Will the N-file handler allow multiple output files to one device?

Answer: You can't have two output files open at the same time or. the same DECtape. The Background and Foreground jobs can both use the DECtape handler, but they can't share the same physical unit; this is to protect the Foreground job.

Question: Would you comment on the kind of environment this Monitor System was written for:

Answer: It is mainly applicable to the real-time environment, where someone has a particular job which takes up most of the computer's 24 hour day (e.g. laboratory experiments, process control, etc.). We are buying a customer two computers with Background/Foreground Monitor System. In addition to running his production type work, he can develop new things, which can then be put into the production system as soon as they are checked out.

Question: Can the system be run in Background only?

Answer: There will be monitor commands which specify Background only or Foreground only.

Question: Can there be simultaneous output files to the same unit on a DISK system?

Answer: Not on the interm DISK system, which is currently available. But on the final system, many output files can be opened at the same time. You will also be able to have files of the same name with different user ID codes.

Question: Can Foreground programs be developed in the Background?

Answer: Yes. One of the main features of the system is that you can develop and test the Foreground programs in the Background (even using DDT).

Question: What happens if a Foreground error occurs (e.g. mark track error on DECtape)?

Answer: Foreground error gives the Foreground control teletype the option of letting the Background go to completion or hitting Control C, which aborts the Background job.

Question: How is Foreground system loaded in core?

Answer: The Foreground Linking Loader will load up rather than down. It will not use any core in the second bank until there is no area large enough left in the first bank, etc.

Question: How are the Background I/O requests prevented from destroying the Foreground job by random writes into core?

Answer: Any I/O handler that the Background job requires that isn't already in core will get brought in below the memory protect boundary. All handlers run with the memory protect off. The monitor checks to see that the word counts and addresses on I/O requests are legal. It also prevents the Background job from modifying the handlers. We assume the handlers to be checked out, like the Foreground user program.

Question: Is memory protect a complete protection?

Answer: Our memory protect scheme is write only protect. You can read anywhere in core. The hardware is the complete protection, but the software will allow you to use the index registers for both jobs and to reference for read only all areas of core. One application in this area would be a Background job, which was actually monitoring the Foreground system. (e.g. was checking registers and producing on-the-fly data for immediate analysis).

Question: Are any auto index registers reserved for the system use ?

Answer: The Monitor System does not utilize the autoindex registers and neither do the I/O handlers. The only system program that requires an index register is DDT, and that allows you to specify before you set break points which index register it should use. The standard is 17. You can change it by opening the register AX$ and change the index register number.

Question: Has the restriction which prohibits CAL's from being executed in the .TIMER completion subroutine been removed?

Answer: The problems of re-entrancy, due to issuing of CAL's from real-time subroutines, have been solved in Background/Foreground System. A CAL can be issued from any real-time routine.

Question: What is the minimum size of the Background job?

Answer: There is a minimum of 4K on the Background job. This will allow you to run the non-resident Monitor and to do things such as editing, loading, and running of very small programs. The recommended size for the Background is 6 to 8K. If you want to use the 4K for the Foreground job, at load time you specify no Background and then Foreground has the entire machine.

Question: How can I tell if a device is bulk storage or not?

Answer: That the buffer size is 377$_8$ is the clue to the fact that a handler is bulk storage. Any other buffer size assumes non-bulk storage.

Question: I understand that the new version of the Monitor will allow assemblies of the .ABS codes on DECtape. Is there any facility for loading from the DECtape?

Answer: That feature was put in primarily to allow you to assemble system programs and load them with PATCH. We have not written a loader for these DECtape files.

Question: Can I prevent the system from returning to the Monitor on an error?

Answer: The new system never returns directly to the Monitor except on a .EXIT. It will print the error and sit there. If you want to go back to the Monitor use Control C, or if you want to restart the program use Control P.

Question: What is the maximum block length with the Editor in Block Mode?

Answer: It is a function of line length and core size, so with an 8K machine and average size lines, stick to the standard (55$_{10}$).

Question: How can I speed up load time of system and user programs?

Answer: If you are not using handlers for devices, such as the drum, card reader, etc., you can speed up loading time by deleting unwanted handlers from the library.

Question: How can I add symbols to DDT permanently?

Answer: Get sources from Program Library and edit new symbols into DDT. Then reassemble DDT (requires 16K for reassembly).

Question: How can I assemble large .ABS programs?

Answer: If the program is too large to assemble with MACRO with PRB. and PPB., then you can try the PDP-9L COMPACT assembler.

Question: What work is being done on the DISK system?

Answer: The DISK System is currently available as modified DECtape system, but work is being done to treat the entire DISK as a real disk. We are also looking into the possibility of using an area of the DISK for symbol table storage.

Question: How can I find the starting address of a user program.

Answer: Location .SCOM + 6 (1Ø6$_8$) contains the starting address of the main program. If you are using DDT it is contained in the register SA$.

Question: How about optional deletion of the memory map by the linking loader?

Answer: In the Background/Foreground System there is an option to delete the loader memory map and an option to list common allocation and/or global definitions.

Question: Why do undefined globals cause a fatal error?

Answer: If this is a problem in development of new programs, there is a way to get around it. Besides writing the initial calling program, you can write another program which defines all the globals as dummy entries. Then in the loader command dummy string, specify that the dummy program be loaded after any of the real programs you are developing.

Question: Can a program in the library request a .DAT slot by using a .IODEV?

Answer: You have to make sure the program with the .IODEV is in the library before all the I/O handlers. In the Background/Foreground System we have gone to a three library structure: an I/O Library which contains nothing but I/O handlers; a Fortran Library which contains the Fortran Object

Time System; and the standard user library. You can have all three on the same system tape. Right now, in order to have a user library routine on a system tape, you have to insert it into the system library which increases the length of time required to load system programs.

Question: Can I load parameter tapes for MACRO using an ASR33?

Answer: The ASR33 does not have a select the reader IOT, so you can end up losing characters because of the speed. The minute you read one character, the next character is selected and when you are running in the interrupt system you could be interrupted from the Teletype handler long enough to get the next interrupt from the ASR. A solution would be to write on ASR handler which runs with the interrupt off.

## FOCAL FOR THE PDP-9

Focal on the 9 takes the FOCAL language as it appears on the PDP-8 * with a few expanded features. The current version which we are hoping to release in March will be a single user FOCAL with library facilities. The FOCAL language itself is a language similar to BASIC or JOSS. It is an on-line interactive language which allow most of the features which are available in FORTRAN. However, it is an interpeter rather than a compiler, so it tends to be slower than the object code generated by FORTRAN.

Question: What are the limits on array dimensioning?

Answer: There are no dimensions for an array. You do not have to dimension a variable in order to subscript. There are no array declarations. It is a completely flexible subscripting. Storage for an array is allocated on an individual element basis.

Question: How can I modify FOCAL Programs?

Answer: There is a dynamic editing program built right in to FOCAL. If you type in a line and make a mistake, you can use the MODIFY command which allows you to change, add or delete individual characters within the line.

Question: Is there a subroutine facility in FOCAL?

Answer: Yes. The DO statement allow you to execute a single line or group of lines, which has been typed elsewhere in you program, as a subroutine.

Question: Can I define subroutines as part of FOCAL?

Answer: External subroutines can be added as functions to FOCAL for the PDP-9. The names of the new user functions are built into a user defined function table, which is loaded with FOCAL. A dummy function table will be supplied for those users not desiring external functions.

Question: Can the FOCAL functions be defined in FOCAL?

Answer: That is something else we are looking into, but there are no definite plans.

Question: Does FOCAL take advantage of EAE?

Answer: The version which will be released from the Program Library will use the standard PDP-9 arithmetic library. So if you have EAE, in the library as a result of System Generation, FOCAL will automatically use it.

Question: How can I save FOCAL Programs?

Answer: There are library commands which allows the FOCAL programs to be created, stored, and loaded from bulk storage or paper tape as named files. FOCAL programs can also be prepared off line on a teletype or prepared on line using PIP or the EDITOR. These files can have calls to other files within themselves so that you can actually chain FOCAL programs and let one call another or just call part of another.

Question: How is FOCAL loaded by the system?

Answer: FOCAL is a user program which is loaded using the Linking Loader. It will make use of all available core which is left after loading for text, variable, and push-down list storage.

Question: How about multi-user FOCAL?

Answer: This is what we had in the Foreground system at the FJCC. With the Background/Foreground System, we will be offering a multi-user FOCAL package, which will allow from two to eight users.

Question: How can I get a single paper tape of FOCAL and thereby not have to use the Linking Loader each time I want to use FOCAL?

Answer: There is a utility program in the new paper tape system which will allow you to link load any program with its I/O handlers and punch it out as a standard .ABS tape.

Question: Can you read in the FOCAL source program from the reader and still have teletype input?

Answer: The way the FOCAL has been set up on the PDP-9 is that it assumes -2 and -3, which are the standard teletype

* See FOCAL PROGRAMMING MANUAL (DEC-Ø8-AJAC-D) or INTRODUCTION TO PROGRAMMING (Small Computer Handbook Series) Chapter 9 for description of the FOCAL language.

.DAT slots, as your normal FOCAL control teletype. It then uses .DAT slots 3 as input and 5 for output. So you can assign any device to input and output.

Question: Can I have FOCAL output on the line printer rather than teletype?

Answer: One possible extension of the language would be to allow you to change the ASK device or the TYPE device and thereby make FOCAL really device independent. This feature may be added at a later date.

Question: How does FOCAL accomplish mix mode arithmetic?

Answer: All numbers are converted into floating point format. That is why you can use mixed modes. It just assumes a decimal point.

# PDP-6/10 WORKSHOP

# SYNCHRONOUS COMMUNICATION INTERFACE FOR THE PDP-10

Norman Housley, (In collaboration with J. C. Pooley)
University of Western Ontario
London, Ontario, Canada

## ABSTRACT

Most remote card reader/printer terminals (whether computer or wired logic controlled) utilize the synchronous communication method of data transmission. The University has developed a controller for use with the PDP-10 in order that it may provide remote computing facilities. This controller places minimum requirements on PDP-10 software/hardware while communicating at 2400 bits per second. Use is made of teletype receiver/transmitter modules (DEC W706, W707) as a front end to a PDP-10 data line scanner communication line. Worst case I/O service requirement is approximately one character time as opposed to the data line scanner requirement of 1.5 bit times.

## OVERVIEW OF THE FACILITY

### Introduction

Figure 1 illustrates the UWO Computing Facility (as of December, 1968). The centre is involved in providing computing facilities for research and teaching in most disiplines. Typical job loads are from 3000-6000 per month. A GE-115 is presently acting as a remote card reader/printer terminal (RPT) prototype on a local telephone loop. It has been in operation for about one month while developing the necessary software.

The PDP-10 section of the facility will be expanded in early 1969 by the addition of :-

1. A further 8 data line scanner ports.
2. An additional 32K of 1.6 microsec core.
3. Another RD-10 disk and synchronizer.

Plans are in hand to have three RPT'S operational by early February 1969. These will be 2 - GE-115's, 1 - UNIVAC 9200.

## COMMUNICATION PRINCIPLES

Two methods of communication are generally used when it is necessary to transfer information between digital computers over telephone lines.

1. Asynchronous
2. Synchronous

### Asynchronous

Essentially each character transmitted contains information that will allow the receiver to synchronize on that character without reference to any other. Figure 2 (a) shows a typical asynchronous transmission sequence. Most terminals used on the PDP-10 are teletypes, they utilize asynchronous transmission at 110 bits per second.

### Synchronous

In synchronous communication a continuous bit stream is used to transmit a message. The initial three or four Characters transmitted allow the receiver to lock onto the bit stream. Figure 2(b) illustrates 2 characters transmitted in synchronous system. Additionally at the end of a message a special character is appended to indicate to the receiver that it should go to an examine mode. In this mode the receiver is again looking for the beginning of a message.

### Comparison of the Systems

Synchronous is inherently faster since stop/start bits are not required, it tends not to be used at low bit rates because more complicated and hence expensive hardware is required. A loss of synchronism may cause only one character to be in error for the asynchronous system. In a synchronous system however, this may destroy the remainder of the message.

## SIMPLIFIED VIEW OF THE CONTROLLER

Figure 3 shows the basic data flow paths in the controller. It is worthwhile noting at this point that although the system was developed with a view to providing a synchronous communication facility it may be used wherever conversion from 8 bit synchronous to 11 bit asynchronous (and vice versa) is required.

The controller uses one communication port on the PDP-10 Data Line Scanner (DC-10). This port is a bidirectional, full duplex asynchronous communication device with a maximum rate of 100K bits per second. Communication with the data set occurs at 2400 bits per second.

The 11 bit serial asynchronous output of the DC-10 is received by an unmodified W706 receiver. The 8 data bits contained in that stream are parallel transferred to a modified W707 for synchronous transmission. Synchronous data is received in a modified W706. After 8 bits have been received they are parallel shifted to an unmodified W707 for asynchronous transmission to the DC-10. The W707 performs the functions of bit rate change and addition of start and stop bits to the 8 bit data character.

The controller is a full duplex device. Due to the extra registers outside the DC-10 more time is allowed for the monitor to service peak I/O demands. Since the device is interfaced to the I/O bus by the DC-10 minimal changes are necessary to the monitor I/O service routines.

## DETAILED DESCRIPTION OF THE CONTROLLER

Figure 4 shows more details of the controller. W706A, W707A are the asynchronous unmodified modules. W707S, W706S are the synchronous modified modules. The DC-10 is an open loop device, that is it does not rely on demand response control. If it's output register (W707) is empty, that fact is indicated to the PDP-10 and another character will be sent to the register. This will be serially output as soon as synchronism (with the scanner clock) is achieved. The controller must only allow the DC-10 to transmit if it requires another character. The "WAIT" signal achieves this requirement. It only allows the DC-10 to transmit if the W706A is empty. The asynchronous section of the controller i.e. W708, W706A, W707A is identical in components and logical operation to one port of the DC-10. The W708 module organizes the 800KHZ scanner clock for both transmission and reception of asynchronous data.

### Output Sequence

Output is initiated by a sequence of synchronizing characters, these are under program control. Typically four 'SYNC's' are sent. On initiation of transmission, no data is contained in the controller. Thus the WAIT signal will allow transmission from the DC-10. When the first 'SYNC' character is received by the W706A, (contents of this register are continuously decoded), the command 'REQUEST TO SEND' is issued to the data set. This instructs the data set to go into transmit mode (in a half duplex system the line will turn around, typical turn around time 200m sec. Minimum 8 msecs). The data set responds with 'CLEAR TO SEND' when it is ready.

At this time the SYNC character contained in the W706A is transferred in parallel to the W707S for synchronous transmission to the data set. While data was in the W706A the WAIT signal inhibited transmission from the DC-10. Once the SYNC character is loaded to the W707S another character may be sent from the DC-10. It in turn will request a further character from the PDP-10 as appropriate. When the first SYNC character has been transmitted, the next SYNC will be loaded to the W707S. After the first SYNC character all other

SYNC's are treated in an identical manner to the data characters that follow SYNC. Data characters will be appended to the synchronous bit stream after all the SYNC characters have been transmitted. Some of these characters may have program control significance.

The last character of the message must be an END OF MESSAGE CHARACTER (EOM). This is used by the remote receiver to change it to a search for SYNC character mode. The REQUEST TO SEND signal is held for the full duration of transmission. The character END OF TRANSMISSION (EOT) is used by the controller to turn of the REQUEST TO SEND command. This is accomplished by a decoder on the W706A output. EOT is only issued by the program if it is desired to turn off the data set carrier. In a half duplex system it is advantageous not to issue EOT unless necessary since this will save line turnaround delays.

## Input Sequence

The normal state for input is that of searching for a SYNC character. The output of the W706S is decoded continuously. Shifting of the received signal taking place as soon as a carrier is detected. As soon as one SYNC is detected the receiver is considered locked. The SYNC character is transferred in parallel to the W707A for serial transmission at 100K bits per second to the DC-10. It is assumed that the DC-10 is always ready to receive a character.

All following characters are transferred as appropriate. The EOM character is detected in order to take the receiver out of lock. When this occurrs no further transfers take place between the controller and the DC-10 until another SYNC character is detected.

Figures 5 and 6 show the actual logic for the controller. Figure 5 is the output logic, figure 7 the input.

## Conclusion

In conclusion I would like to summarize the reasons for this particular implementation.

1. the converter had to be developed quickly. The time from conception of the idea to a working system was about four weeks. Hardware was chosen that was available quickly.

2. changes to the PDP-10 hardware and software were to be kept to a minimum.

3. physical size and complexity to be minimized.

I consider that this goal was achieved. Other implementations are possible as are systems with additional features. Future designs may encompass some of these features.

## Acknowledgement

I would like to acknowledge the work of J. C. Pooley on the staff of the Computing Centre. He performed all the detailed design of the synchronous controller.

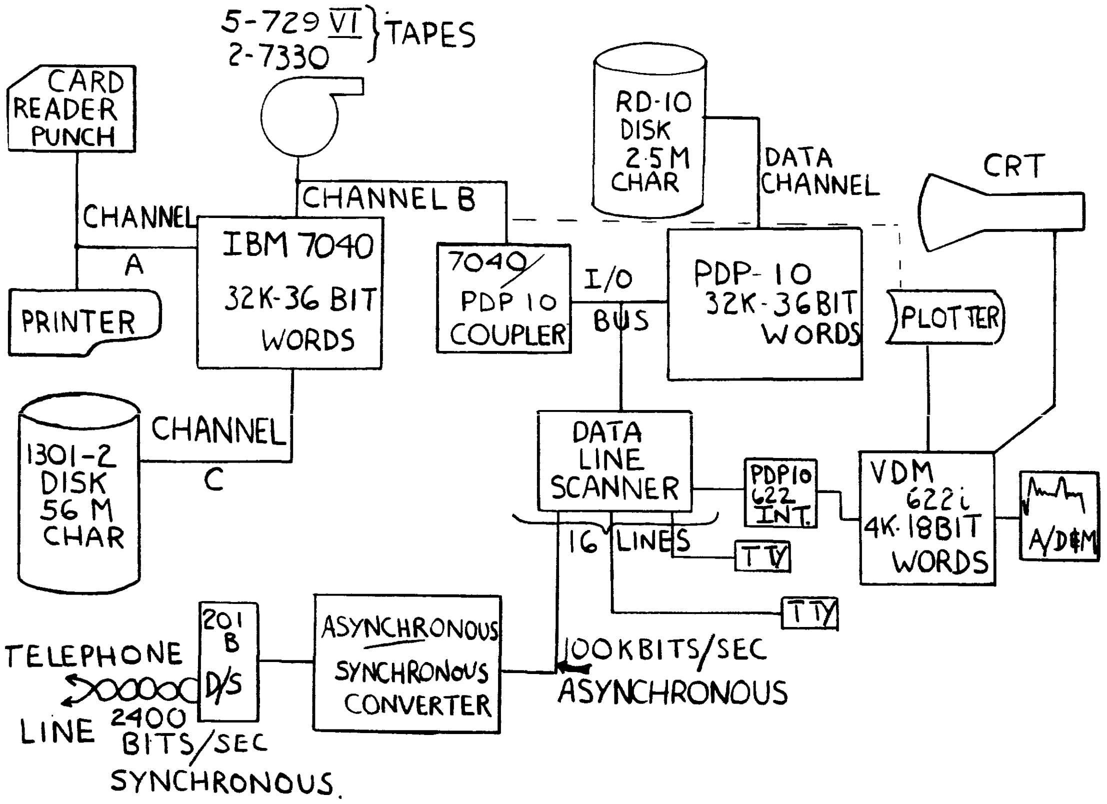
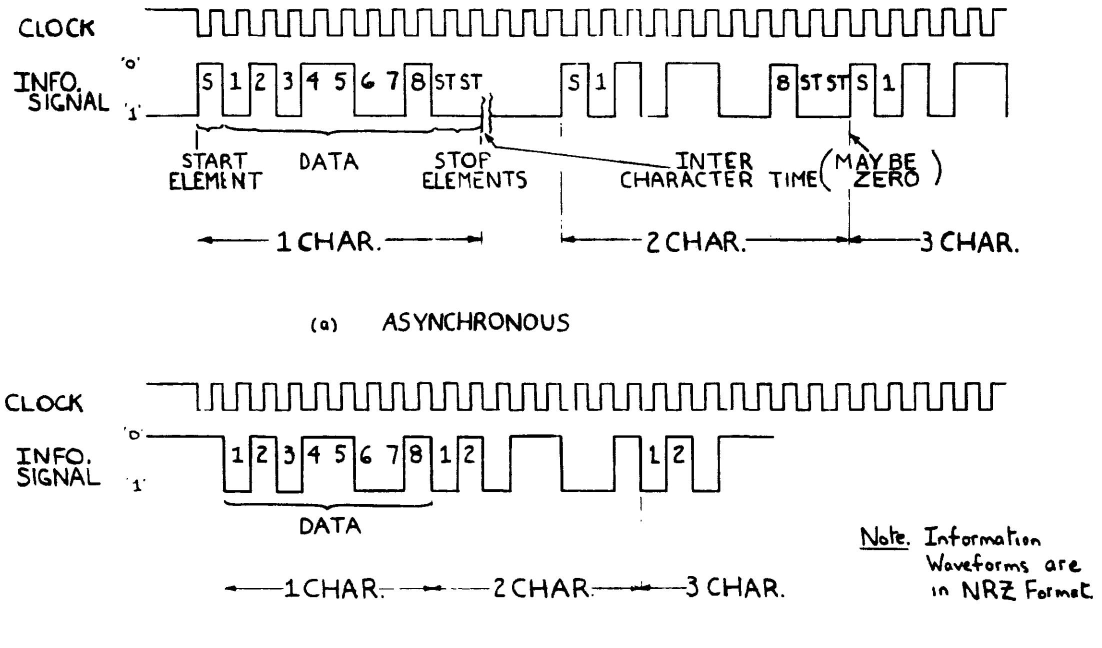


Figure 1 UWO Computing Centre.



(a) ASYNCHRONOUS

Note. Information Waveforms are in NRZ Format.

(b) SYNCHRONOUS.

Figure 2 Transmission Modes.

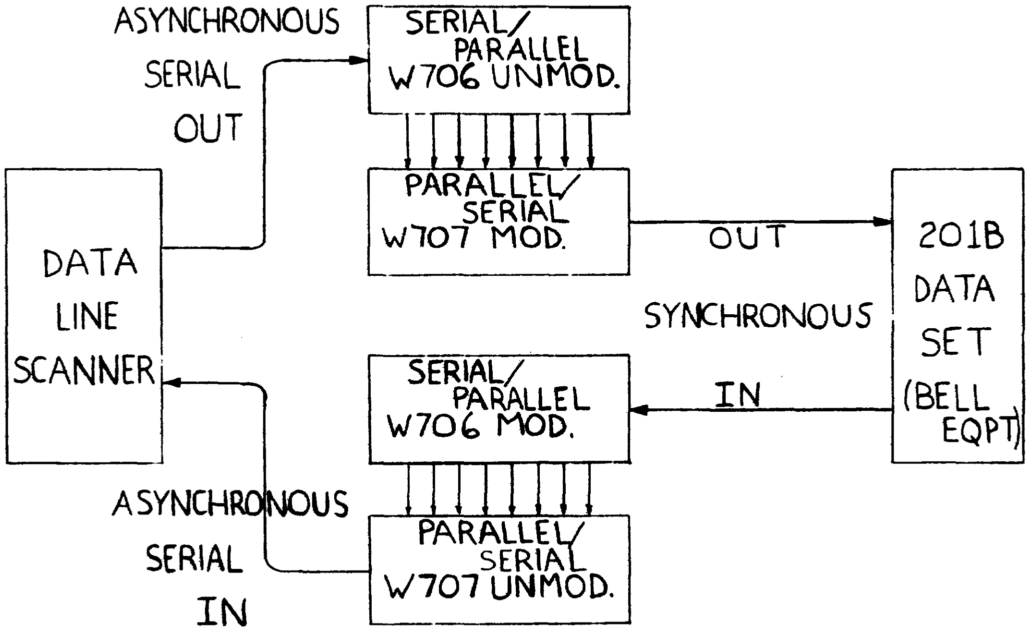

ASYNCHRONOUS SERIAL OUT
SERIAL/ PARALLEL W706 UNMOD.
PARALLEL/ SERIAL W707 MOD.
OUT
201B DATA SET (BELL EQPT)
SYNCHRONOUS
DATA LINE SCANNER
SERIAL/ PARALLEL W706 MOD.
IN
ASYNCHRONOUS SERIAL IN
PARALLEL/ SERIAL W707 UNMOD.

Figure 3 Simplified View of Controller.

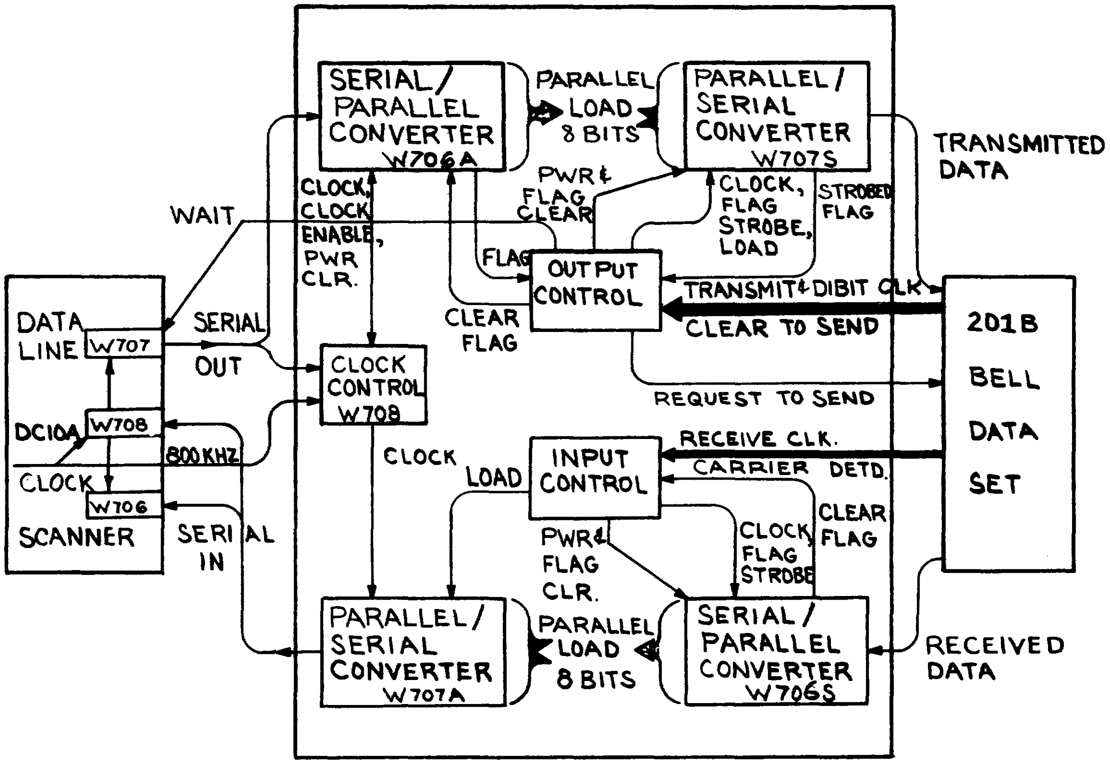


Figure 4 Asynchronous/Synchronous Converter.

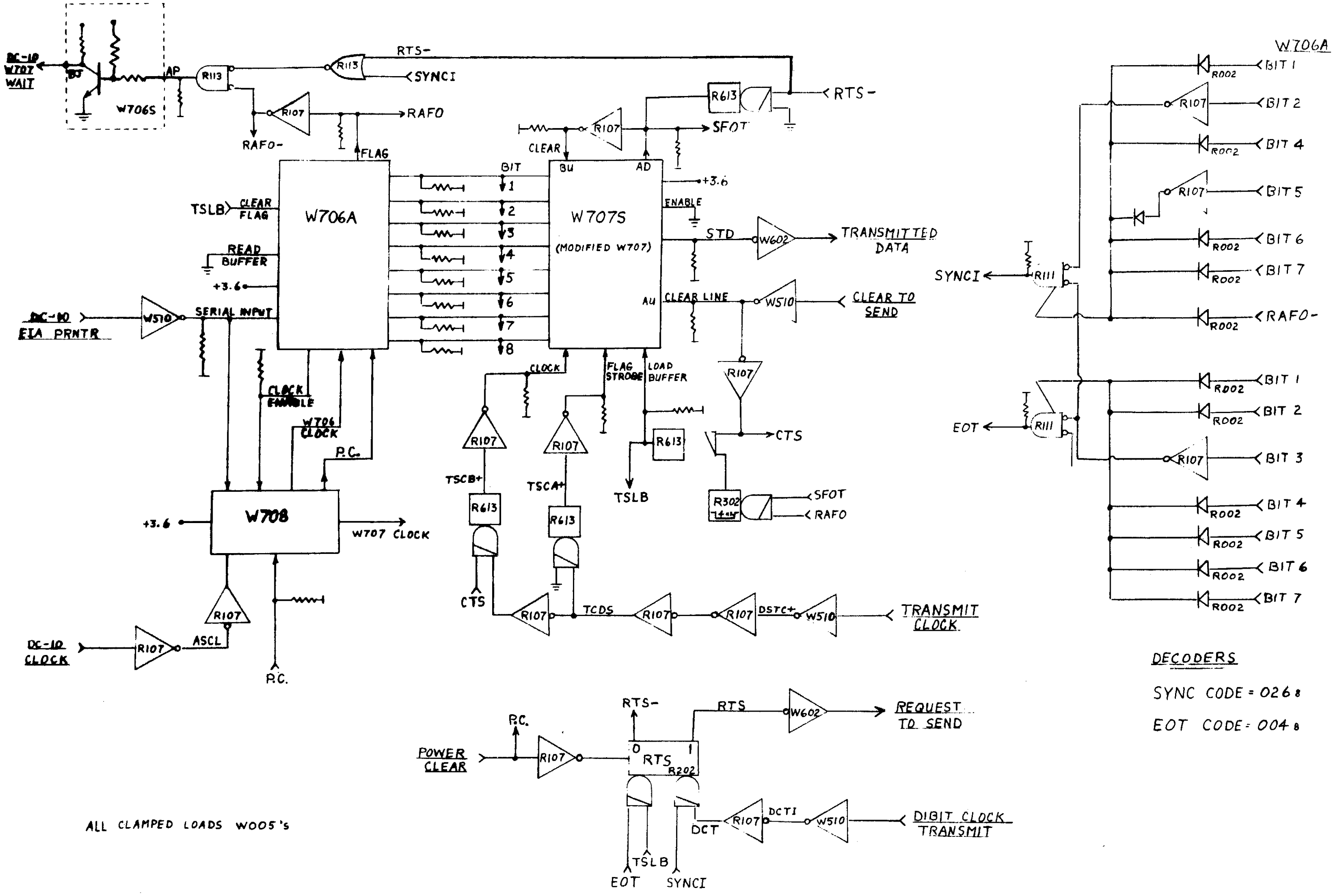
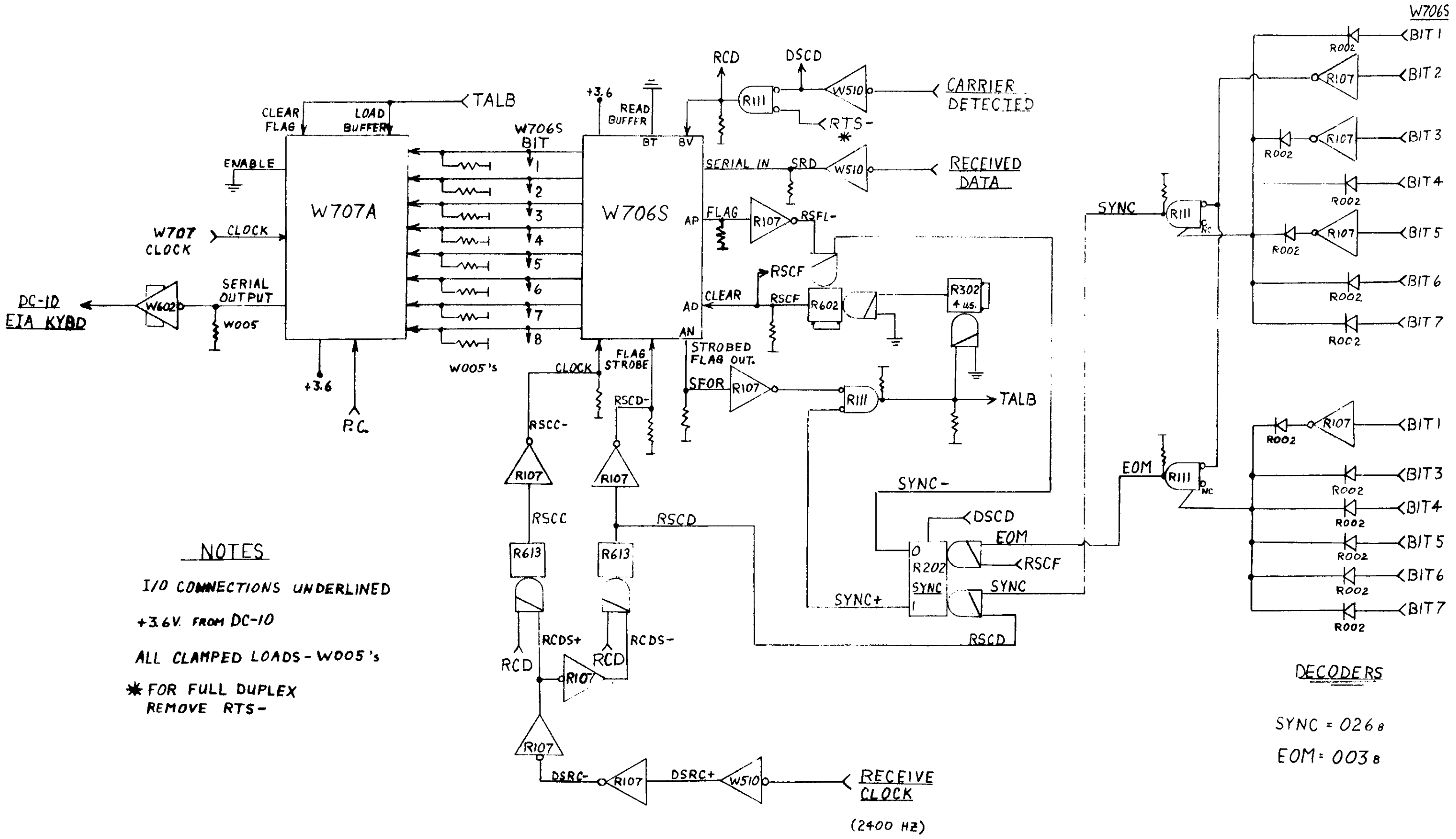


Figure 5 Sync Out



NOTES

I/O CONNECTIONS UNDERLINED

+3.6V. FROM DC-10

ALL CLAMPED LOADS - W005's

* FOR FULL DUPLEX REMOVE RTS-

DECODERS

SYNC = $026_8$

EOM = $003_8$

Figure 6 Sync In

# RAPID PROGRAM GENERATION PROGRAM

William F. Weiher
Stanford University Artificial Intelligence Laboratory

and

Richard P. Gruen
Digital Equipment Corporation
Palo Alto, California

## ABSTRACT

Rapid Program Generation on the PDP-6/10 has been made possible by the addition of five commands to the DEC Time-Sharing Monitor. These commands (EXECUTE, DEBUG, COMPILE, LOAD, and CREF) allow the time-sharing user to specify the names of the programs which he wishes to use and then delegate to the Rapid Program Generation System the task of compiling, assembling, and loading these programs, as needed, without requiring the user to type CUSP commands. The implementation makes use of an RPG cusp and some small files on the disk; only nine additional instructions have been added to the Time-Sharing Monitor.

## INTRODUCTION AND EXAMPLE

Several new commands have been added to the DEC PDP-6/10 Time-Sharing System to facilitate Rapid Program Generation. In the ordinary case a time-sharing user need type only the EXECUTE command, followed by a list of the files with which he would like to work. The Rapid Program Generation System (RPG) translates to machine language those of the user's programs which need translation, loads all of the programs into core, and starts execution of the object program. Thus the ordinary time-sharing user need not be concerned with the various and sundry command lists required by each of the individual translators which he uses.

An example will help to illustrate the use of the new commands. Suppose a time-sharing user wanted to work with a main program and a subroutine, both in the FORTRAN language. He might proceed as follows:

```
.create main.f4                                          ;create a disk file
*i1ØØ                                                    ;begin inserting on line 1ØØ
ØØ1ØØ                 type 69                            ;statements in FORTRAN
ØØ2ØØ   69            format (' this is the main program')
ØØ3ØØ                 call sub1
ØØ4ØØ   $                                                ;ALT-MODE ends the insert
*e                                                       ;e ends the edit and exits
EXIT
↑C                                                       ;control has returned to monitor
.create sub1.f4                                          ;use another disk file
*i1ØØ                                                    ;insert starting at line 1ØØ
ØØ1ØØ                 subroutine subr                    ;FORTRAN subroutine
ØØ2ØØ                 type 1Ø5
ØØ3ØØ   1Ø5           format (' this is sub1 ')
ØØ4ØØ   $                                                ;ALT-MODE ends the insert
*e                                                       ;end and exit to monitor
EXIT
↑C
.execute main,sub1                                       ;request RPG execution
```

```
MAIN.    ERRORS DETECTED:  Ø              ;FORTRAN reports its progress
SUBR     ERRORS DETECTED:  Ø
LOADING
?ØØØØØ1  UNDEFINED GLOBALS
?  SUB1  ØØØ152                           ;there is no subroutine named sub1
LOADER 5K  CORE                           ;this includes the space for loader
?EXECUTION DELETED                        ;no execution
EXIT
↑C
.edit                                     ;user now asks to edit sub1.f4
                                          ;he need not mention the name, since
                                          ;this is the file he edited last
*R1ØØ                                     ;replace line 1ØØ
ØØ1ØØ              subroutine sub1
*g                                        ;user asks to go to his program
                                          ;this ends the edit (like e command)
                                          ;and then does the last command typed
                                          ;to the RPG system
SUB1     ERRORS DETECTED:  Ø              ;only the subroutine is recompiled,
                                          ;since only it has been edited
LOADING                                   ;both main and sub1 are loaded
LOADER 5K  CORE
THIS IS THE MAIN PROGRAM                  ;execution begins
THIS IS SUB1
EXIT                                      ;execution ends
↑C                                        ;and so does this example
.
```

## COMMAND DESCRIPTIONS

### COMPILE command

COMPILE takes a list of file names as its argument. It is assumed that all of the files are on the disk. Each file is processed by the appropriate processor if necessary. It is considered necessary to process a file if no .RELfile of the same name exists or if the .RELfile was created before the last time the text file was edited. COMPILE determines which processor is appropriate by examining the extension of the file. The following is a table showing which processor will be used for various extensions:

| | |
|---|---|
| .MAC | MACRO |
| .FAI | FAIL (an assembler developed at Stanford) |
| .F4 | FORTRAN |
| .REL | No processing will be done |

If the file has the blank extension, COMPILE will use whichever processor is considered "standard." The "standard" processor starts out as FORTRAN and may be changed by use of the various switches (see below). It is not necessary to indicate the extension of a file to COMPILE. If only the file name is given it will first look to see if there is a file of that name with no extension. If there is not, it will look for one with the extension .FAI then .MAC then .F4 then .REL. It is only if none of these are found that it will use the "standard" processor.

As an example, suppose you wished to compile the files FOO.FAI with FAIL, GARP.FAI with FAIL and BAZ.F4 with FORTRAN. Any of the following command strings would be appropriate:

```
COMPILE FOO.FAI,GARP.FAI,BAZ.F4
COMPILE FOO,GARP,BAZ
COMPILE FOO,GARP.FAI,BAZ
```

### LOAD command

LOAD operates as COMPILE, and then loads the specified files.

### EXECUTE command

EXECUTE operates as load, and then starts execution of the object program.

### DEBUG command

DEBUG operates as LOAD, but also loads DDT (the DEC Debugging Tape). It then starts DDT rather than the object program.

### CREF command

Each of the above commands generates information as to which cross-reference listings have been generated since the last time CREF was used. The command CREF prints (on device LPT:) all cross-reference listings generated since the last time this command was used. In some systems, this command is CROSS.

### CROSS command

Same as CREF command.

### EDIT command

Takes a single file name. Brings EDIT1Ø into core, and initializes it to edit that file.

### CREATE command

Operates like EDIT, except that it first initializes a disk file with the given name. CREATE should be used to create a file which has never before been used.

### TECO command

Operates as EDIT, but brings in TECO rather than EDIT1Ø.

### MAKE command

Operates as CREATE, but uses TECO rather than EDIT1Ø.

### TYPE command

Types the file (whose name is the argument of the command on device TTY:.

### LIST command

Lists the file whose name is the argument of the command on device LPT:

### DELETE command

Accepts a list of file names, separated by commas, and deletes those files from the disk. Wild names or extension may be specified by asterisks (*); for example DELETE *.MAC would delete all files with the extension .MAC from the disk.

### RENAME command

Rename accepts a list of file name pairs, with the elements of a pair separated by equals (=) and the pairs separated by commas. The file having the second name of each pair is renamed to have the first name of each pair. For example, RENAME NEW1= OLD1, NEW2=OLD2 would rename OLD1 to be NEW1 and rename OLD2 to be NEW2.

### DIRECTORY command

Lists on device TTY: the names and creation dates of all files on the user's disk area. If an argument is given, that argument is taken as the name of the device whose directory is desired (rather than DSK:).

## NOTES ON GENERAL SYNTAX

Spaces, tabs, and form feeds are completely ignored in all commands. Carriage returns and line feeds are ignored if they are followed by a comma, and are treated as commas otherwise. Blank lines are ignored completely. A semi-colon may be used to insert a comment; all characters between the semi-colon and the following linefeed (inclusive) are ignored.

## NOTES ON FILE DESCRIPTORS

Files have been shown in most examples as just a name, perhaps with an extension. In general files are described as:

```
          DEV:NAME.EXT[PRJ,PRG]
```

where DEV: is a device name (followed by a colon; assumed as DSK: if absent)
Name is the name of the file
EXT is the filename extension, preceeded by a period
[PRJ, PRG] is the project and programmer pair describing some disk area other than that of the current user. For example, execute DTA4:PROG1, PROG2 would compile and execute two files from DTA4: (since device names are sticky). Similarly, LOADFOO. F4[1,WFW] would compile and load the file FOO.F4 from the disk area of 1, WFW. Whether or not re-compilation was needed would have been determined from the date and time (if extent) of the file FOO.REL in the current disk area.

## SWITCHES TO COMPILE

Switches to COMPILE are preceeded by a "/" and are multiple character rather than single character. They may appear either before or after a file name (i.e.

```
COMPILE FOO/MACRO,BAZ
COMPILE /MACRO FOO,BAZ
```

If a switch appears after a file name (first example) it applies only to that file, if it appears before a file name it changes the "standard mode" of the compile command for that file and all following files (until changed by another switch). In the above examples if neither FOO nor BAZ had an extension then in the first example FOO would be processed by MACRO and BAZ by FORTRAN. In the second example both FOO and BAZ would be processed by MACRO.

All switches may be abbreviated, requiring only enough characters to distinguish them from all other switches. The switches currently available are:

| | |
|---|---|
| FORTRAN | set standard mode to FORTRAN (originally set to FORTRAN) |
| MACRO | set standard mode to MACRO |
| M | same as above |
| FAIL | set standard mode to FAIL. |
| F | same as FORTRAN (the M and F switches are to allow frequently used commands to be shortened without confusion) |
| LIST | create listing files (.LST) for all files processed |
| L | same as above |
| NOLIST | turn off listing (original setting is NOLIST) |
| N | same as above |
| REL | do not process file at all |
| CREF | create cross-reference files (.LST) for all files processed |
| C | same as above |
| LIBRARY | do not process and if loading (see LOAD) do a library search |
| NOLIBRARY | turn off library mode |
| COMPILE | process even if dates indicate it is not needed |
| NOCOMPILE | turn off COMPILE mode |
| MAP | if loading create a loader map at this point, output file is MAP.MAP There is no difference in placing this switch before rather than after a file name. |

EXAMPLE: Suppose FOO1,FOO2, and FOO3 were to be processed by MACRO and GARP1, GARP2, and GARP3 by FORTRAN. Also suppose that listings of FOO2,GARP1 and GARP2 and a cref of FOO3 were desired. In addition suppose a library search of the hand-eye library were necessary. The following command string would accomplish this:

```
COMPILE /MACRO FOO1,/LIST FOO2,FOO3/C,/FORT
GARP1,GARP2,GARP3/N,HELIB[1,3]/L
```

## MULTIPLE FILES

It is sometimes necessary to indicate to the processor that the program to be processed is contained in the concatenation of several files. For example consider the command string

```
    APRSER←S,APRSER
```

given to macrox. Such constructions are allowed by COMPILE using the "+" feature, to get the same effect as above the proper command to COMPILE is

```
    COMPILE S+APRSER/MACRO
```

A maximum of 7 files may be combined in this way. The name given to the binary (.REL) and listing (.LST) files will be that of the last program in the string (thus APRSER in the above example). It is possible to give the binary and listing files some other name by the use of the "=" feature. Suppose it were desired to give the name FOO to the binary produced above. Either of the following would work:

```
    COMPILE FOO=S+APRSER/MACRO
    COMPILE S+FOO=APRSER/MACRO
```

## SWITCHES TO LOADER

It is possible to pass switches to the loader (assuming one of LOAD, EXECUTE, or DEBUG is being used). For example to pass the "S" switch to the loader, one would use %S. This construction may appear either before or after a file name and the place of appearance is preserved in the commands given to the loader. Thus the command LOAD %SFOO would give the command string /SFOO to the loader while LOAD FOO%S would give FOO/S to the loader. The "%" take the next letter following it to pass to the loader. There is one exception to this. If the character following the "%" is a digit, it and all the digits following it and the letter following them will be taken. Thus to pass the string /6ØØØ0 to the loader use %6ØØØ0.

## SWITCHES TO PROCESSORS

Switches may also be passed to processors by enclosing them in "( )". Let #1,#2, and #3 each denote a string of letters and digits. The construction (#1) will pass the string #1. to the source portion of the processor. (#1,#2) will pass #1 the binary portion and #2 to the source and (#1,#2,#3) will pass #1 to binary, #2 to list and #3 to source.

```
EXAMPLE  :  COMPILE FOO=GARP.MAC/LIST(,A,Q)
+GARP2(B, C)+GARP3(G)
```

This will produce the macro command string:

```
FOO(B),FOO(A)←GARP(Q),GARP2(C),GARP3(G)
```

The "( )" construction must follow the file name. An error will result if an attempt is made to specify more than one set of binary or listing switches for any given output file or if the "( )" construction is used more than once after any single file name.

## PARAMETER FILES

It is sometimes useful to assemble several programs with the same set of parameters such as accumulator assignments, etc. This results in commands of the form

```
    LOAD S+APRSER,S+SCNSER,S+DISSER
```

Such commands may be simplified by using the following format

```
    LOAD S+<APRSER,SCNSER,DISSER>
```

Note that the constructions

```
    LOAD <FOO,BAƵ>+GARP
    LOAD GARP+<FOO,BAƵ>+FOOBAR
```

are not permitted.

## COMMAND FILES

At any point in the command string, the construction @filename may occur. The @filename is replaced by the entire contents of the file named. This construction may be nested to a depth of 9 (this number may change). An error message will be given if deeper nesting occurs.

## REMEMBERED COMMANDS

The last command given is remembered by the program so that the sequence of commands

```
    COMPILE FOO
    DEBUG FOO
```

may be replaced by

```
    COMPILE FOO
    DEBUG
```

## HOW COMPILE WORKS

Compile works by generating files containing command for the various processors. These have the following names

| | |
|---|---|
| QQMACR.RPG | MACRO |
| QQFAIL.RPG | FAIL |
| QQFORT.RPG | FORTRAN |
| QQCREF.RPG | CREF |
| QQLOAD.RPG | LOADER |
| QQPIP.RPG | PIP |
| QQSVCM.RPG | To remember last command given |
| QQSVED.RPG | To remember last file edited. |

## ERRORS

TOO MANY SWITCHES too many switches for the internal storage have been given for either the loader or a processor

TOO MANY NAMES too many names given in the + construction

DISK NOT ABAILABLE something is probably wrong with the system

OUTPUT ERROR an output error on the disk which writinga command file

INPUT ERROR an input error from disk while reading a file indicated by an @

PROCESSOR CONFLICT COMPILE does not know what processor to use, can result from a command string line
COMPILE FOO.MAC+GARP.FAI

NOT ENOUGH CORE there is not enough core for buffers available

SYNTAX ERROR something is wrong with your syntax

NESTING TOO DEEP the depth of permissible use of @has been exceeded

UNRECOGNIZABLE SWITCH a switch (/ type) has been given which COMPILE does not recognize, try typing more of the full name of the switch

NO SUCH FILE - filename this file could not be found

FILE IN USE OR PROTECTED - filename COMPILE could not write one of the QQ files

Any error will cause COMPILE to call exit

# A FLYING SPOT SCANNER ON-LINE TO A PDP-6

I. G. Nicholls and C. L. Jarvis
Computing Centre
The University of Western Australia

## ABSTRACT

The interest in image processing shown by several research groups within the University of Western Australia has resulted in a standard Fernseh flying spot scanner being linked on-line to the PDP-6 within the time sharing system.

The paper reports the interface design and control in view of the restraints imposed by the existing PDP-6 configuration and the needs of the research groups involved.

## INTRODUCTION

The acquisition and interpretation of data in the form of photographic images is becoming increasingly important in many fields of research. In the biological sciences scanning microscopes on-line to small computers are opening up the way to computer interrogation and quantitative measurement of images of many kinds. Such integrated special purpose systems are commonplace among the bigger research institutions. Yet the cost of these systems, while not great, is often beyond the resources of smaller research teams where the decision of what system to implement or build is always governed by the availability of funds and the nature of existing equipment. This has been the case at the University of Western Australia for a number of workers interested in image processing.

The flying spot scanner belonging to the Electrical Engineering School has been coupled to the University PDP-6 to meet the combined needs of these groups for a picture digitizer.

This paper briefly describes the on-line linkage and discusses the resulting design in view of the user's needs and the system restraints.

## DESCRIPTION OF EQUIPMENT

### The Computing System

The University's computing system is an integrated one based on a time-shared 32K PDP-6 computer which has had extensive additions made to it by the real time group. These additions include:

1. A disc sub-system based on CDC 854 diskpacs upgraded to 12 million bytes and driven by a PDP-8.

2. A unit to communicate with up to 16 remote computers.

3. Six real time on-line experiments which unless hardware is common may run simultaneously in the system.

### Flying Spot Scanner

The FSS is a standard Fernseh which accepts slides with a 21 x 28 mm picture size and densities between 0.2 and 2.2. Vertically these are made up of two interlaced fields of 312 and 313 lines. Time to scan one pitcure is 1/25 second and at a sample rate of 14 Mc/s approximately 700 nyquist elements are sampled in a horizontal direction. For the Australian standard of 10 Mc/s 500 elements are possible. The current quantizer built by the Electrical Engineering School has 16 adjustable quantization level settings.

## DESIGN PHILOSOPHY

As the link will be used by a number of research groups with varying goals, a flexible system is required. To achieve this it has been decided to have the computer send to the FSS an (x,y) coordinate and have the FSS return the quantized optical density at that point, together with those of the next eight points (36 bits of information).

This allows the user to either:

1. Generate a matrix of points to cover the whole picture.

2. Select a portion of the picture for digitization.

3. Or generate (x,y) requests conditional on the data acquired to date.

The immediate requirements of other devices and the unpredictable nature of other users within the system make it desirable to design an asynchronous link. While it is true the computer is far from saturated, the number of real time devices already in the system and the demands of the swapping system argue further for making the FSS an on-line asynchronous linkage.

The synchronous nature of the device has been bypassed by having the scanner's response to a point request wait until it next passes that point in its synchronous scan. It then collects the generated levels of the required 9 points and transmits them to the computer.

Two other features are included in the design. A manual interrupt button that allows the user to cease data collection halfway through a picture and regain program control together with a "resolution switch" that allows halving of the sample rate in each direction.

## SCAN TIME CONSIDERATIONS

Assuming that the user requests consecutive points he would collect data for 9 points once per scan taking 32.4 minutes to collect a full picture. It is obvious that if estimates can be made of the response time of the PDP-6 to service interrupts within any one configuration, then the time to scan one picture can be appreciably reduced by ordering the coordinate requests in a systematic manner to optimize on the movement of the sampling spot. For example, if the user anticipated that the computer would take on an average 500 µsec. per sampling period (and was correct in this assumption) then allowed for this in generating the coordinates, it follows the computer would collect 8 amounts of data or 72 points per scan, taking 4.05 minutes to collect a full picture or just over 1 minute at half resolution.

## HARDWARE

The control register of the interface with the PDP-6 contains 3 flip flops for the P.I. channel number, a done flag, a busy flag, a manual interrupt flag and an error flag which is set by the FSS if the (x,y) coordinates requested are out of range. A maintenance flag is also available so that the x and y coordinate flip flops may be read back instead of data.

The data register is used to buffer 10 bits for the x coordinate and 10 bits for the y coordinate. They are wired to come from different halves of the I/O bus so as to allow the use of the half word instructions in setting them up.

## SOFTWARE

The FSS service routine added to the time shared monitor accepts a buffer of (x,y) coordinates and once started uses a BLKI followed by a BLKO to accept data and send out the next request coordinate pair. This cuts monitor overheads on the normal INPUT, OUTPUT UUO's to a minimum.

The buffer length is set at 100 words and a 1K user program can be used for straight data collection. To avoid setting in the middle of core for long periods while the device is active (shuffling being inhibited), the job can be made a real time job by using the REALT UUO.[1] This prevents the job from being swapped out of and into core, resulting in it always being resident in lower core with all other real time jobs.

## COMMENTS

The total cost of the linkage including additions to the FSS but excluding labour is under $2,500. It should be noted, however, that the quantizer unit at the FSS had been built by the Electrical Engineering School as part of another project and is not included in this figure.

This is the second on-line device at our installation where the I/O service routine handles inputs to, mixed with outputs from the same buffer and it is clear that a REPLACE UUO would be useful. As well as being more meaningful, it could save the user fiddling with the IOUSE bit to keep buffers in order.

## ACKNOWLEDGEMENTS

The authors thank Mr. A. Scolaro, Mr. R. Perry and Mr. G. Myers of the Electrical Engineering School for the design and building of additions to the FSS.

## REFERENCE

1. Nicholls, Ian G. "Real Time Jobs in the Swapping System" The University of Western Australia Computing Centre Information Sheet No. 17. February 1968.

# APPENDIX

# PAPERS PRESENTED BUT NOT SUBMITTED FOR PUBLICATION

ON-LINE NAVIGATION AND DATA LOGGING FOR THE MPL DEEP TOW
Carl D. Lowenstein

SOFTWARE FOR A PDP-8 PULSE HEIGHT ANALYZER SYSTEM
W. J. Edwards

AUTOMATED ALPHA PULSE ANALYSIS — PART II
J. E. Evans and G. G. McMillan

A SYSTEM FOR ON-LINE COLLECTION OF REPETITIVE DATA USING THE PDP-8/S
C. Douglas Creelman

LOGO — A PROGRAMMING LANGUAGE FOR CHILDREN, TEACHERS, AND MATHEMATICIANS
Wallace Feurzeig

REVISED SUBROUTINE LIBRARY FOR EAE PDP-9
E. C. Itean, Paul Manos, and L. R. Turner

INTERACTIVE SYSTEMS APPLIED TO THE REDUCTION AND INTERPRETATION OF SENSOR DATA
Charlton M. Walter

A SIMPLE NEW DISK MONITOR SYSTEM FOR THE PDP-8
Theodore R. Sarbin and Richard A. Roth

OBTAINING A CASE HISTORY BY COMPUTER
Theodore R. Sarbin

A DISC ORIENTED REAL-TIME EXECUTIVE FOR THE PDP-8/S COMPUTER
W. T. Lyon

TECO-8: TEXT EDITOR AND CORRECTOR PROGRAM FOR THE PDP-8
Russell B. Ham

OS/8: OPERATING SYSTEM FOR PDP-8
Russell B. Ham

AUTOMATIC SUBROUTINE LINKAGE ACROSS CORE BANKS IN A PDP-8
Joseph Rodnite

TSS/8 — GENERAL PURPOSE TIME-SHARED PDP-8
Don Witcraft

PDP-8/S IBM 360/65 DATA LINK
Larry Green

PROGRAMMED MIXED MODE DATA COMMUNICATION
Hans J. Breme

# AUTHOR/SPEAKER INDEX

## ATTENDEES

Mr. John Alderman
Georgia Institute of Technology

Mr. Derik V. Armstrong
University of California
Lawrence Radiation Lab

Mr. Robert P. Abbott
Institute of Medical Sciences

Mr. Roger E. Anderson
University of California
Lawrence Radiation Lab

Mr. Sypko Andreae
University of California
Lawrence Radiation Lab

Mr. Andrew A. Allison
Digital Equipment Corporation

Mr. John B. Brown
Betatech

Mr. Stephen P. Bartok
U.S. Public Health Service

Mr. Franklin M. Battat
Liberty Gold Fruit Co.

Mr. Virgil Bedwell
EAI

Mr. Al Beal
Digital Equipment Corporation

Mr. R. Blacker
Lockheed Missiles & Space Co.

Kathleen Benson
University of California

Mr. Dow Brian
Stanford University

Mr. Roman I. Bystroff
University of California
Lawrence Radiation Lab

Mr. Aleksander Bilinski
Perkin-Elmer

Mr. Arnold Branco
Institute of Medical Sciences

Mr. Don Barker
Digital Equipment Corporation

Mr. Michael Behar
University of California
Psychology Department

Mr. John Baker
University of California
Lawrence Radiation Lab

Mr. Joe Bourland
Baylor University College of Medicine

Mr. Dale A. Baker
Cutler-Hammer, Inc.

Mr. Barry R. Borgerson
University of California

Mr. Philip R. Bevington
Case Western Reserve University

Mr. Hans J. Breme
Western Electric

Mr. Randy Bush
University of Chicago

Mr. W. Wayne Black
Idaho Nuclear Corporation

Mr. Ralph Belluardo
United Aircraft Corporation

Mr. Michael Bertin
Stanford University

Carolyn Bailey
University of California
Lawrence Radiation Lab

Dr. Ludwig Braun
Polytechnic Institute of Brooklyn

Mr. Lawrence W. Brown
QEI

Mr. Walter R. Burrus
Tennecomp, Inc.

Mr. Howard Briscoe
M.I.T. Lincoln Lab

Mr. Stephen Boies
University of Oregon

Mr. Louis J. Bartscher
Minneapolis - St. Paul Sanitary District

Mr. John A. Bellantoni
Digital Equipment Corporation

Mr. James H. Boardman
South Dakota School of Mines & Technology

Mr. James M. Chandler
VIDAR Corporation

Miss M. Child
Langley Porter

Mr. Duward Cagle
University of California
Lawrence Radiation Lab

Mr. Riley D. Carver
University of California
Lawrence Radiation Lab

Mr. Douglas P. Chinnock
University of Arizona

Dr. E.R.F.W. Crossman
University of California

Mr. S. L. Cooke, Jr.
University of Louisville

Mr. Eugene S. Ceschini
Gulf Research & Development Co.

Mr. George S. Cooper
LOGIC, Inc.

Mr. Richard A. Cabot
S. Sterling Company

Mr. Ronald Compton
Beckman Instrument, Inc.

Mr. Fred W. Cowap
Badger Meter Manufacturing Co.

Mr. Donald V. Conroy
University of Alabama

Mr. Douglas Creelman
University of Toronto

Mr. James Crapuchettes
Stanford Electronics Labs

Mr. Fred F. Coury
University of Michigan

Claudia Gain Conn
Computer Sciences Corporation

Mr. Stephen F. Carlson
Minneapolis - St. Paul Sanitary District

Mr. R. deSaussure
University of California
Lawrence Radiation Lab

Mr. David Donaldson
Digital Equipment Corporation

Mr. Ric Davies
Phillips Petroleum Company AED

Mr. David A. Dalby
Bedford Institute

Mr. James L. Downs
Badger Meter Manufacturing Co.

Mrs. Evelyn S. Dow
Digital Equipment Corporation

Mr. Norman Doelling
Digital Equipment Corporation

Mr. James Delorey
San Francisco Bay Naval Shipyard

Mr. C. G. Donahoe
San Francisco Bay Naval Shipyard

Mr. Ronald J. Dupzyk
University of California
Lawrence Radiation Lab

Mr. Walter A. Danley
CSC

Mr. Jimmie E. Doyle
DOT, FAA Aeronautical Center

Mr. Ronald M. Davis
International Computer Science

Mr. Craig Denison
University of California
Lawrence Radiation Lab

Mr. R. D. Dobson
Arizona State University

Mr. Doug Dyment
Digital Equipment Corporation

Mr. Victor L. DaGragnano
U.S. N.R.D.L.

Mr. Richard DeJohn
Digital Equipment Corporation

Mr. Allan T. Devault
Digital Equipment Corporation

Mr. James E. Evans
University of California
Lawrence Radiation Lab

Mr. Marvin D. Erickson
Battelle Northwest

Mr. David A. Ensor
O.I.S.E.

Mr. James R. Emanuel
Gulf Research & Development Co.

Mrs. Judith B. Edwards
Computer Instruction NETWORK

Mr. Anthony D. Eppstein
Memorex Corporation

Mr. Glenn Elliott
Sandia Corporation

Mr. Dan L. Eisner
Conrac Corporation

Mr. W. J. Edwards
Atomic Energy of Canada Ltd.

Mr. Irwin R. Etter
The Mason Clinic

Mr. Lester Ernest
Stanford University

Mr. David A. Entrekin
C.I.C. Computer Systems, Inc.

Mr. Donald L. Frazer
University of California
Lawrence Radiation Lab

Mr. Alfred D. Ford
Department of Defense

Mr. J. A. Field
University of Waterloo

Mr. Wallace Feurzeig
Bolt Beranek and Newman, Inc.

Mr. J. E. Francis
Union Carbide

Mr. J. Ford
Institute of Medical Sciences

Mr. Donald R. Fanshier
University of California
Lawrence Radiation Lab

Mrs. Barbara J. Fanshier
Haward Unified School District

Mr. Eugene Fisher
University of California
Lawrence Radiation Lab

Mr. James W. Fryklund
Oregon State University

Mr. Dennis R. Friedrichs
Battelle Northwest

Mr. William G. Feeney
Chevron Research Company

Mr. David G. Frutchey
Beckman Instruments, Inc.

Mr. Richard J. Feldmann
National Institute of Health

Mr. A. E. Fredrickson
Digital Equipment Corporation

Mr. P. Fleck
M.I.T. Lincoln Lab

Mr. Alexander Firestone
University of California
Lawrence Radiation Lab

Mr. Daniel M. Forsyth
University of Vermont

Mr. Dave Friesen
Digital Equipment Corporation

Mr. John R. Griffin
Tektronix, Inc.

Mr. Peter P. Goldstern
Digital Equipment Corporation

Mr. John C. Gwinn
Stanford University

Mr. Joel Grossman
Interactive Computing Corp.

Mr. Roy Gould
Digital Equipment Corporation

Mr. Richard P. Gruen
Digital Equipment Corporation

Mr. W. Scott Gerstenberger
University of Michigan

Mr. Michael D. Greenblatt
University of Pennsylvania

Mr. C. W. Gear
University of Illinois

Mr. James R. Guggemos
University of California
Lawrence Radiation Lab

Mr. Robert A. Glenn
Fairchild Instrumentation

Mr. Larry Green
University of California

Mr. John J. Gould
Brookhaven National Labs

Mr. John S. Gentelia
S. Sterling Company

Mr. John R. Goltz
University of Arizona

Mr. Per Hjerppe
Digital Equipment Corporation

Mr. Tom Haratani
University of California
Lawrence Radiation Lab

Mr. Stan Hubler
RCA EASD

Mr. Larry M. Hunt
Martin Marietta Corporation

Mrs. Virginia Hunt
Martin Marietta Corporation

Mr. Donald Hodges
Stanford Medical Center

Mr. George L. Helgeson
Helgeson Nuclear Services, Inc.

Mr. Peter Helgeson
Helgeson Nuclear Services, Inc.

Mr. Peter Harris
Langley Porter

Mr. Larry Hawley
Beckman Instruments, Inc.

Mr. F. V. Harshbarger
University of California
Lawrence Radiation Lab

Mr. Stanley Hochman
Digital Equipment Corporation

Mr. Leonard M. Hantman
Keydata and Adams Associates, Inc.

Mr. Dick Hull
Hull Associates

Mr. Peter M. Hurley
Digital Equipment Corporation

Mr. R. Douglas Hunter
Interactive Computing Corp.

Mr. Robert Henry
University of California

Mr. Richard C. Hewitt
Bell Telephone Labs.

Mr. John Hunt
Berkeley, California

Mr. R. L. Heath
Idaho Nuclear Corp.

Mr. Peter D. Headly
University of Michigan

Mr. James R. Harvill
University of California
Lawrence Radiation Lab

Mr. James B. Hunter
University of California
Lawrence Radiation Lab

Mr. Frank W. Hafner, III
Jet Propulsion Laboratory

Mr. Russell B. Ham
U.S. Public Health Service

Mr. Gene Holt
Jet Propulsion Lab

Mr. Norman Housley
University of Western Ontario

Mr. Martin S. Itzkowitz
University of California
Lawrence Radiation Lab

Mr. Richard Imossi
Brookhaven National Laboratory

Mr. Clifton A. Johnson
Digital Equipment Corporation

Mr. J. A. Jones
Digital Equipment Corporation

Mr. Gordon D. Jones
University of California
Lawrence Radiation Lab

Mr. Glen C. Johnson
Phillips Petroleum Company AED

Mr. Ted Johnson
Digital Equipment Corporation

Nancy A. Jones
University of California
Lawrence Radiation Lab

Mr. David W. Jenson
University of California
Lawrence Radiation Lab

Mr. Robert Jahncke
Beckman Instruments, Inc.

Mr. Oliver W. Jones
University of California
Lawrence Radiation Lab

Mr. Edward Kramer
Digital Equipment Corporation

Mr. Arthur M. Kray
University of California
Lawrence Radiation Lab

Mr. Neale Koenig
Information Control Systems, Inc.

Mr. James E. Kuffert
Beckman Instruments, Inc.

Mr. David Kleinecke
General Electric

Mr. John R. Kosorok
Battelle Northwest

Mr. A. H. Kadish
Mt. Sinai Hospital

Mr. Robert Krones
University of California
Phonology Laboratory

Mr. Joe Katz
University of California
Lawrence Radiation Lab

Mr. Claude Kagan
Western Electric Company

Mr. Alan Kotok
Digital Equipment Corporation

Mr. David Leney
Digital Equipment Corporation

Mr. Alan Loceff
Information Control Systems

Mr. Harry R. Lewis
National Institutes of Health

Maj. Robert A. Leach
United States Military Academy

Mr. Hans P. Lie
Bell Telephone Labs.

Mr. Robert K. Lindsay
University of Michigan

Mr. Bruce D. Link
Oregon Research Institute

Mr. Don Larson
Digital Equipment Corporation

Mr. Kevin Langdon
University of California
Phonology Laboratory

Mr. Richard L. LaPierre
University of California
Lawrence Radiation Lab

Mr. Robert Lafore
University of California
Lawrence Radiation Lab

Mr. Rich Leres
University of California
Lawrence Radiation Lab

Mr. Almon E. Larsh, Jr.
University of California
Lawrence Radiation Lab

Mr. G. T. Lake
University of Western Ontario

Mr. William T. Lyon
Aluminum Company of America

Mr. Robert W. Luckey
Naval Research Lab

Mr. Donald C. Loughry
Hewlett-Packard Comapny

Mr. James J. Lacey
Gulf Research & Development Co.

Mr. John Leng
Digital Equipment Corporation

Mr. Robert L. Litle
Beckman Instruments, Inc.

Mr. Ken Larson
Digital Equipment Corporation

Mr. T. E. Lindsay
Bell Telephone Labs.

Mr. C. D. Lowenstein
Scripps Institution of Oceanography

Mr. Bobby J. Little
Sandia Laboratories

Mr. Mark F. Lewis
Civil Aeromedical Institute

Mr. Charles W. McMullen
University of California

Miss Marie V. Morello
Keydata and Adams Associates, Inc.

Mr. Thomas W. MacLean, Jr.
Tektronix, Inc.

Mr. Malcolm McAfee
University of California

Mr. Richard L. Miller
Center for Human Growth & Development

Mr. Richard Merrill
Digital Equipment Corporation

Mr. L. G. Mann
University of California
Lawrence Radiation Lab

Mr. C. D. Martin, Jr.
Union Carbide Nuclear Division

Mr. Craig MacKenna
Badger Meter Manufacturing Co.

Mr. Walter A. Miller
Chase Brass & Copper Co., Inc.

Mr. Philip D. Morse, II
University of California
Zoology Department

Mr. Arthur W. McLaughlin
AGA Corporation

Mr. Joseph L. McGregor
Mason & Hanger-Silas Mason Co.

Mr. Gary B. Morgan
Idaho Nuclear Corporation

Mr. Myron H. Myers
University of California
Lawrence Radiation Lab

Mr. Tom McGovern
Atomic Energy of Canada Ltd.

Mr. Adrian D. McGrede
VIDAR Corporation

Mr. Gerald N. Miller
Digital Equipment Corporation

Mr. James Murphy
Digital Equipment Corporation

Mr. David G. McLane
System Development Corporation

Mr. H. R. Morse
EDUCOM

Mr. Avram-Chaiyim Miller
Langley Porter

Mr. Wayne A. Muth
Southern Illinois University

Mr. J. S. MacDonald
University of B.C.

Mr. James M. Mathiesen
San Francisco Bay Naval Shipyard

Mr. Fred Macondray
University of California
Lawrence Radiation Lab

Mr. Robert D. McInnis
Digital Equipment Corporation

Mr. C. R. Martell
Sandia Corporation

Mr. Howard D. Marshall
Western Electric Company

Mr. R. L. McInturff
Digital Equipment Corporation

Mr. K. R. Morin
University of British Columbia

Dr. Walter H. Moran, Jr.
West Virginia University

Jule Meyn
Bio-Science Laboratories

Dr. Robert H. McKay
University of Hawaii

Dr. Michael J. McKeown
Chicago Lying-in Hospital

Mr. William A. Minnick
ITEK Corporation

Mr. Roger A. Morris
Los Alamos Scientific Laboratory

Mr. Richard J. McQuillin
Inforonics, Inc.

Mr. Joseph D. Naughton
National Institutes of Health

Mr. James B. Niday
University of California
Lawrence Radiation Lab

Mr. Pirkko Niemela
University of California
Department of Psychology

Mrs. Judy E. Nichols
Digital Equipment Corporation

Dr. Rolf Nordhagen
University of Oslo

Mr. Michiyaki Nakamura
University of California
Lawrence Radiation Lab

Dr. Robert F. Nickerson
University of California
Lawrence Radiation Lab

Miss Peggy Noble
Digital Equipment Corporation

Mr. Edward Nemeth
Digital Equipment Corporation

Mr. Allan Newman
Information Control Systems

Mr. Ray S. Newbury
University of California
Lawrence Radiation Lab

Dr. Simon F. Ng
University of Ottawa

Mr. Downey Overton
Digital Equipment Corporation

Mr. Kazuo Ochiai
University of California
Phonology Laboratory

Mr. Leif Ohlsen
AGA AB

Mr. Alan Olson
Interactive Computing Corporation

Mrs. Carole O'Hare
University of California
Department of Psychology

Mr. John F. O'Donnell, Jr.
Time, Inc.

Mr. Vernon C. Poulter
Digital Equipment Corporation

Mr. Roger L. Peterson
University of California
Lawrence Radiation Lab

Mr. Ronald S. Piatasik
Digital Equipment Corporation

Mr. Ken Peal
Canada Centre for Inland Waters

Mr. Harold L. Pearson
West Virginia University

Mr. Don Phelps
University of California
Lawrence Radiation Lab

Mr. Paul J. Paolini, Jr.
The University of Georgia

Mr. James E. Powell
Stanford University

Mr. Roger C. Pyle
Digital Equipment Corporation

Mr. James D. Pitts
Digital Equipment Corporation

Mr. James H. Puls
University of Arizona

Mr. Nicholas Pappas
Digital Equipment Corporation

Elizabeth Quigg
University of California
Lawrence Radiation Lab

Mr. Chris Raymond
Institute of Medical Sciences

Mr. Ronald G. Ragsdale
Ontario Institute for Studies In Education

Frederica Russ
Stanford Medical Center

Mr. Stephen Russell
Computer Center Corporation

Mr. Harry Rvoloe
Harvard University

Mr. R. W. Ranshaw
University of Pittsburgh

Mr. C. Wendell Richardson
Phillips Petroleum Company

Mr. Ross C. Richards
Bedford Institute

Mr. David A. H. Robinson
Bell Telephone Labs.

Dr. A. M. Revzin
Federal Aviation Administration

Mr. Joseph A. Rodnite
Information Control Systems, Inc.

Mr. Al Roddan
Beckman Instruments, Inc.

Mr. Tom H. Rau
The Dikewood Corporation

Mr. Lloyd Robinson
University of California
Lawrence Radiation Lab

Mr. Harry T. Ryan
Nuclear Data, Inc.

Mr. Hugo Ripp
CBS Inc.

Mr. Robert J. Reed
Digital Equipment Corporation

Mr. Triston J. Rosenberger
University of California
Human Factors Lab

Mr. D. Lloyd Rice
U.C.L.A.

Mr. Bruce A. Raby
University of California
Lawrence Radiation Lab

Phyllis Strawbridge
NASA Ames Research Center

Mr. William Segal
Digital Equipment Corporation

Mr. Gordon E. Stokes
Idaho Nuclear Corporation

Mr. David Slotnick
Digital Equipment Corporation

Mr. Charles P. Spector
Digital Equipment Corporation

Mr. Charles M. Stasey
ITEK

Mr. Paul G. Sandberg
Foxboro Company

Mr. Robert A. Saunders
Stanford University

Mr. Fredric M. Strange
University of California
Lawrence Radiation Lab

Mr. John Sauter
Stanford University

Mr. James M. Stephan
U.S.G.

Mr. E. H. Stewart
Digital Equipment Corporation

Mr. Eric L. Sigurdson
University of B.C.

Mr. William H. Sisson
Bell Telephone Labs.

Dr. V. Slamecka
Georgia Institute of Technology

Mr. Milton Schwartz
Conrac Corporation

Dr. Hideo Seo
University of Illinois

Mr. Jack Shields
Digital Equipment Corporation

Mr. David Schurr
AO Instrument Company

Mr. Patric Savage
Shell Development Company

Mr. Jon D. Stedman
Berkeley, California

Dr. Dan W. Scott
University Computing Co.

Mr. Fred R. Sias, Jr.
University of Mississippi Medical Center

Mr. Robert L. Siegel
Western Electric Co.

Mr. Bruce A. Sherwood
California Institute of Technology

Mr. Thomas E. Sullivan
Trans World Airlines

Mr. C. K. Stone
Digital Equipment Corporation

Mr. Robert W. Shannon
VIDAR Corporation

Mr. Robert L. Swenson
University of California
Lawrence Radiation Lab

Mr. Phillip D. Siemens
University of California
Lawrence Radiation Lab

Mr. Theodore R. Sarbin, Jr.
University of California
Human Factors Laboratory

Mr. Robert W. Skyles
VIDAR Corporation

Mr. Harold W. Shipton
University of Iowa

Mr. Joseph E. Strople
Data Systems Design

Mr. George R. Sugar
ESSA Research Laboratories

Mr. Ernest Singleton, Jr.
U.S. Naval Weapon Laboratory

Mr. Patrick Suppes
Stanford University

Dr. George Stone
University of California
Computer Center

Mr. Thomas G. Taussig
University of California
Lawrence Radiation Lab

Mr. D. R. Thorne
Arizona State University

Mr. Homer Tsuda
Fairchild Instrumentation

Marsh Tekawa
University of California
Lawrence Radiation Lab

Mr. Harry M. Taxin
Hughes Aircraft Company

Mr. Roy S. Taylor
National Security Agency

Mr. John Tubbs
Stanford Medical Center

Mr. H. Sandoe Thomsen
Beckman Instruments, Inc.

Mr. Garth Thomas
Ohio State University Hospitals

Mr. Burr M. Tupper
Digital Equipment Corporation

Dr. M. M. Taylor
DRET

Mr. Lyndon A. Thomas
New Brunswick Research & Productivity Center

Mrs. James R. Teays
Digital Equipment Corporation

Mr. L. R. Turner
NASA Research Center

Mr. Donald C. Uber
University of California
Lawrence Radiation Lab

Mr. John Van Drasek
Whirlpool Corporation

Mr. Charles A. M. Vandenbrekel
University of California
School of Optometry

Mr. Robert Willis
Digital Equipment Corporation

Mr. B. Michael Wilber
Stanford Research Institute

Miss Barbara Williams
Mobil Research and Development Corp.

Mr. Melvin D. Woolsey
Digital Equipment Corporation

Mr. Norris R. Weimer
University of California
Lawrence Radiation Lab

Mr. Howard Whiting
University of California
Lawrence Radiation Lab

Mr. Earl L. Wiener
University of Miami

Mr. Ronald W. Weigand
Digital Equipment Corporation

Mr. C. S. Wilson
Shell Development Co.

Mr. H. J. Ward
Noller Control Systems

Mr. Bruce A. Watson
Stanford University

Mr. David J. Waks
Applied Data Research, Inc.

Mr. Ronald I. Wallace
University of California
Lawrence Radiation Lab

Mr. Bruce Wood
Engis Equipment Co.

Mr. William F. Weiher
Stanford University

Mr. William A. Worden
San Francisco Bay Naval Shipyard

Barbara B. Wolfe
Wayne State University

Mr. Barry L. Wolfe
Information Control Systems, Inc.

Mr. Albert L. Wilson
Mason & Hanger-Silas Mason Co.

Mr. Charles Wagner
Fairchild Semiconductor

Wyoma M. Webber
FAA Aeronautical Center

Mr. Arnold Wasserman
First National City Bank

Mr. Ken Wiley
University of California
Lawrence Radiation Lab

Mr. Charlton M. Walter
AFCRL

Mr. Don A. Witcraft
Digital Equipment Corporation

Mr. Howard M. Yanof
Medical College of Ohio

Mr. Mark Zeligman
University of California
Lawrence Radiation Lab

Mr. Verlan J. Zapotocky
University Computing Co.

Mr. Ronald Zane
University of Hawaii

DECUS

PRINTED IN U.S.A.