

AD-A196 118

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY 1

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 88-125	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN IMPROVED DATA COLLECTION AND PROCESSING SYSTEM		5. TYPE OF REPORT & PERIOD COVERED MS THESIS
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) JOHN R. O'HAIR		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: TEXAS TECH UNIVERSITY		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 1988
		13. NUMBER OF PAGES 240
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) AFIT/NR Wright-Patterson AFB OH 45433-6583		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) DISTRIBUTED UNLIMITED: APPROVED FOR PUBLIC RELEASE		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) SAME AS REPORT		
18. SUPPLEMENTARY NOTES Approved for Public Release: IAW AFR 190-1 LYNN E. WOLAVER <i>Lynn Wolaver</i> 21 July 88 Dean for Research and Professional Development Air Force Institute of Technology Wright-Patterson AFB OH 45433-6583		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ATTACHED		

DTIC  
ELECTE  
AUG 03 1988  
S D

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AN IMPROVED DATA COLLECTION AND PROCESSING SYSTEM

by

JOHN R. O'HAIR, B.S. in E.E.

A THESIS

IN

ELECTRICAL ENGINEERING

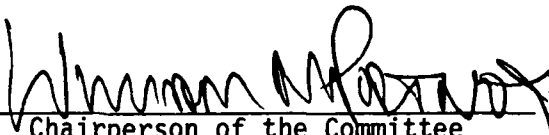
Submitted to the Graduate Faculty  
of Texas Tech University in  
Partial Fulfillment of  
the Requirements for  
the Degree of

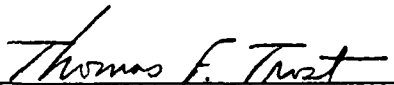
MASTER OF SCIENCE

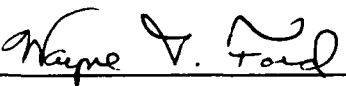
IN

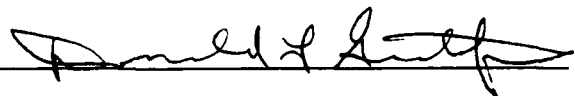
ELECTRICAL ENGINEERING

Approved

  
Chairperson of the Committee

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_

Accepted

\_\_\_\_\_  
Dean of the Graduate School

May, 1988

# ABSTRACT

Title: An Improved Data Acquisition and Processing System  
 Author: John R. O'Hair, 1Lt, USAF  
 Degree Awarded: Master of Science in Electrical Engineering  
 Texas Tech University, 240 pages, 1987.

This work details the development of a software control system to be used to collect data on instabilities occurring in transistors during turn-off (commonly, Reverse Bias Second Break-down). The system uses a Zenith Model 158 PC compatible microcomputer with a Hewlett-Packard Interface Bus (HPIB) 61062AA interface card linked to a Tektronix 7612D Programmable Digitizer (7612D) for data capture, processing and storage. The control system performs the functions necessary to read the breakdown data from the 7612D, process said data and provide output to the operator in one of several forms. Output of graphic data requires a Hewlett-Packard 7470A (or compatible) Graphics Plotter. Numeric output can be written to any standard line printer or to disk for later printing.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Availability Codes
A-1	



## ACKNOWLEDGEMENTS

I would like to thank Dr. William M. Portnoy for his assistance throughout my stay at Texas Tech. I extend my deepest appreciation to the other graduate students in the Pulse Power Conditioning Laboratory for their help both in academics and on the project reported in this thesis. I gratefully acknowledge the guidance and help of my parents who have supported me in all my undertakings, and lastly, and most importantly, I thank my Lord, Jesus Christ, for his nurturing and care.



## TABLE OF CONTENTS

ACKNOWLEDGEMENTS .....	ii
LIST OF FIGURES .....	v
CHAPTER	
I.    INTRODUCTION	
Focus.....	1
Background.....	1
II.   TEST APPARATUS .....	3
Incorporating the Zenith .....	4
III.  SOFTWARE DEVELOPMENT	
BASIC Programming .....	7
MS-PASCAL Programming .....	10
Turbo Pascal Programming .....	15
System Software .....	16
Data Acquisition Programs.....	16
Ancillary Programs .....	19
Files Created and Individual Structure.....	24
IV.   VERIFICATION OF SOFTWARE SYSTEM	
Accuracy of Measurement.....	27
Verification.....	29
V.    CONCLUSION .....	40
LIST OF REFERENCES .....	41
APPENDICES	
A.    OPERATING INSTRUCTIONS .....	42
B.    TAKEDATA.BAT PROGRAM LISTING.....	48
C.    HARDWARE.PAS PROGRAM LISTING.....	52
D.    GETDEVIC.PAS PROGRAM LISTING.....	65
E.    STORDATA.PAS PROGRAM LISTING .....	79
F.    PROCESS1.PAS PROGRAM LISTING .....	94
G.    PROCESS2.PAS PROGRAM LISTING .....	107
H.    PROCESS3.PAS PROGRAM LISTING .....	118
I.    REPEATER.PAS PROGRAM LISTING .....	127

J.	PLOTDATA.PAS PROGRAM LISTING.....	133
K.	TABULATE.PAS PROGRAM LISTING.....	187
L.	SORTABLE.PAS PROGRAM LISTING.....	202
M.	EDITLIST.PAS PROGRAM LISTING .....	221
N.	PRCSLTWO.PAS PROGRAM LISTING .....	231

## LIST OF FIGURES

1.	NPN Transistor showing voltage and current conventions.....	2
2.	Schematic of a simplified data acquisition system incorporating the Zenith Microcomputer. ....	5
3.	Drawing showing DOS memory allocation along with MS-PASCAL working area memory allocation. ....	11
4.	Normal Pascal program hierarchy compared to modified hierarchy.....	13
5.	Example of plot type 1.....	20
6.	Example of plot type 2.....	22
7.	Example output from SORTABLE.COM. Sorted on forward and reverse base drives, respectively .....	23
8.	Storage filename interpretation .....	26
9.	Sample plots along with photograph of Collector-Emitter Voltage and Collector Current. The figure shows the entire test cycle from TUT turn-on to turn-off. In data acquisition, the time of interest is during the peak in the Collector-Emitter Voltage. This is the region where breakdown will occur.....	30
10.	Sample plots showing safe turn-off. This is a blow-up of the peak region in Figure 9. Forward bias setting is 1 A. Reverse bias is 0.05 A .....	31
11.	Sample plots again showing safe turn-off. The rate of drop in the Collector-Emitter Voltage is now more precipitous indicating nearing the edge of safe operation. Forward bias setting is 3 A. Reverse bias setting is 0.05 A .....	32
12.	Sample plots showing breakdown. Forward bias setting is 3 A. Reverse bias setting is 0.1 A.....	33
13.	Sample plots showing breakdown. Although breakdown does occur, it takes considerably longer to happen. Forward bias setting is 3 A. Reverse bias setting is 0.06 A .....	34
14.	Verification of P and E by graphical means. Forward bias setting is 1 A. Reverse bias setting is 0.06 A .....	36
15.	Verification of P and E by graphical means. Forward bias setting is 2 A. Reverse bias setting is 0.1 A.....	37

16.	Verification of P and E by graphical means. Forward bias setting is 2 A. Reverse bias setting is 0.2 A.....	38
17.	Verification of P and E by graphical means. Forward bias setting is 2 A. Reverse bias setting is 0.4 A.....	39
18.	Integration Scheme .....	114
19.	Example of sorting function .....	212

## CHAPTER I INTRODUCTION

### Focus

The focus of this effort has been to continue with the work previously done at Texas Tech University in the area of Reverse Bias Second Breakdown of transistors. Past work has centered around the development of a complete data acquisition system to examine the above mentioned phenomenon. This endeavor has concentrated on improving one major aspect of the system, data collection and processing.

In the following, discussion shall be limited primarily to the work undertaken by the author, and only enough background material as is necessary to provide a complete picture of the project shall be included; therefore, the reader should first familiarize his or herself with the work done previously before continuing (1,2,3,4).

The data collection system consists primarily of a Tektronix 7612D Programmable Digitizer with two Tektronix 7A16P Programmable Amplifiers, Hewlett-Packard HP-85 Microcomputer, HP-9895A Flexible Disc Memory, HP-7470A Plotter and Zenith Model 158 Microcomputer with Hewlett-Packard Interface Bus (HPIB) 61062AA interface card. This setup, except for the Zenith Microcomputer, has been used in all the past work, although the digitizer was not integrated during the earliest works.

The specific objective of this effort was to utilize the new Zenith Microcomputer in the collection system as efficiently as possible. Thus gaining the maximum improvement to the overall system possible with the Zenith Microcomputer addition.

Work was broken into three phases. Phase one consisted of integrating the new Zenith Microcomputer into the data collection system. Physical placement, hardware checkout and software development are covered here. Phase two consisted of initial data measurements, and phase three consisted of verification of said data. Of the three phases, phase one consumed the most time and effort, and, for that reason, reporting of the work done during that phase represents the bulk of this thesis.

### Background

Reverse Bias Second Breakdown, as it is referred to here, is an instability that occurs in transistors during turn-off. The most common environment for breakdown is a transistor being used to switch an inductive load, but this instability does occur under many other operating conditions.

The breakdown instability is characterized by a sudden decrease in Collector-Emitter Voltage ( $V_{ce}$ ) and increase in Collector Current ( $I_c$ ). The change in  $V_{ce}$  is on the order of 500 Volts in 10 to 20 nanoseconds. This precipitous variation is used to trigger the protection circuit described in the afore mentioned references.

To force the Transistor Under Test (TUT) into breakdown, it is first turned on by a forward current pulse into the base of the transistor. After approximately 5ms, a reverse current pulse is applied to the base of the transistor. A forward pulse is defined as being in the direction of  $I_b$ , and a reverse pulse is in the opposite direction (see Figure 1).

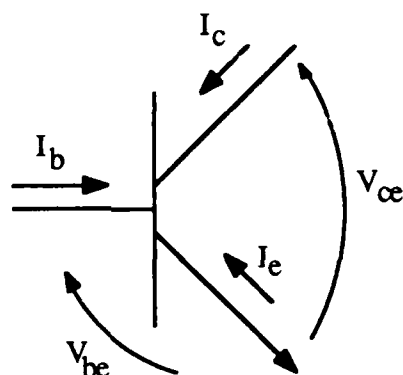


Figure 1. NPN Transistor showing voltage and current conventions.

If breakdown occurs, it appears shortly after the application of the reverse current pulse, when the transistor is attempting to turn off. The time interval between the application of the reverse current pulse and breakdown is of specific interest in this research. Observing the values of  $I_c$ ,  $V_{ce}$  and  $V_{be}$  during the time interval provides crucial information in the evaluation of breakdown. From  $I_c$  and  $V_{ce}$ , the instantaneous Power,  $P$ , and the Energy at Breakdown,  $E$ , can be determined. These last two values, combined with the preceding three, define, for the most part, the conditions of breakdown.

This work concentrated on developing a system which would collect  $I_c$ ,  $V_{ce}$  and  $V_{be}$  and calculate  $P$  and  $E$  in as efficient a manner as possible given the equipment at hand.

## CHAPTER II

### TEST APPARATUS

A more detailed examination of the equipment used in the data collection system is now in order. Previous efforts used a HP-85 Microcomputer as System Controller tied into the rest of the system via a HPIB plug-in module. Waveform data could be digitized by a Tektronix 7612D Programmable Digitizer (7612D) in which case the data could be directly read into the System Controller across the HPIB, or it could be digitized by, first, taking a photograph of the waveform trace captured on an oscilloscope and then digitized by placing the picture on a graphics tablet and manually transcribing the waveform.

The system also contained a HP 3467A System Voltmeter, a IIP 3497A Data Acquisition Control Unit and a HP 9111A Graphics Tablet; however, these devices are no longer used as the functions they perform and the data they provide can all be obtained by using the 7612D alone. As far as data collection is concerned, the system requires only the 7612D, once it has been properly interfaced into the system controller.

In the most recent work done prior to this effort, the 7612D interface had been completed, and it was the primary source of data acquisition for that work. The software that was developed to control the interface between the HP-85 System Controller and the 7612D is adequate for the task of taking data; however, it is an exceedingly slow system.

At the start of this effort, a Zenith Model 158 Microcomputer (referred to hereafter as the Zenith) was added to the system. The goal of this addition was to correct the one major deficiency of the past system, speed. It was hoped that the Zenith would be able to handle the task of data acquisition and processing in a near real time fashion, something that was entirely lacking in the past configuration of the system.

There are several factors, relevant to this discussion which have an impact on the speed of the data acquisition system. First, the HP-85 uses a microprocessor which is not as sophisticated or powerful as the microprocessor used in the Zenith. The HP-85 microprocessor operates at sub-megahertz clock speeds as compared to the Zenith microprocessor, the Intel 8088, which operates at between 4.77 and 8 MHz. Although raw clock speed is not a complete indicator of processing speed, the order of magnitude difference between the HP-85 and the Zenith is a rough indication of the improvement in computational speed that could be gained by upgrading the system to include the Zenith.

Another factor which affects the speed of the acquisition process is the programming language available for collection control and data manipulation. The HP-85 uses

BASIC as its upper level programming language. Non-compiled BASIC, like that used in the HP-85, tends to be much slower than the more powerful compiled languages available to the Zenith such as PASCAL or FORTRAN. Combining the programming language with some shortcomings in the way the HP-85 talks to the HP-9895A disk drive, the process of data transfer and storage becomes bogged down.

Another deficiency of the HP-85 that the Zenith was meant to correct was shortage of memory. The HP-85 contains approximately 32 kilobytes of direct access memory known as Random Access Memory (RAM). This limited amount of memory made it impossible to read in more than 1024 points for each curve on the 7612D. The 7612D is capable of taking up to 2048 samples on two separate waveforms simultaneously; therefore, for a given time interval, half the sampling bandwidth was lost. The Zenith is capable of taking in the full 4096 points available, having 640 kilobytes of RAM at its disposal.

#### Incorporating the Zenith

The Zenith was, at first, designed to be an addition to the acquisition system and not a replacement for the HP-85. Because of certain difficulties encountered, this proved to be impossible within the timeframe of this work.

Figure 2 shows the expected change from the old system to the new. The HP-85 was to remain the System Controller up to the point of actually transferring the data from the 7612D. At that point, the Zenith would takeover, transfer the data to itself, relinquish control to the HP-85 and begin processing the data in a real time fashion. In theory, according to the supposed capability of the HPIB, this plan should have been possible. It was, however, not.

The HPIB is Hewlett-Packard's emulation of the IEEE-488 standard set in 1978. This standard contains the proper protocols to transfer active control of the bus to another controller, called the Active Controller. This transfer facility is what would be necessary to use both the HP-85 and the Zenith on the same bus at the same time.

As the Zenith stands now, it contains the HPIB interface card which is responsible for carrying out all communications over the bus. Control of bus communications is done via a set of software commands inside the Zenith which were provided with the interface card and corporately are referred to as the Command Library. In order to use the Command Library, the HPIB interface card must be set as the System Controller; however, when set as the System Controller, the Zenith is unable to pass control of the bus to an Ac-



tive Controller (in this case, the HP-85). With the Zenith set as System Controller, it is impossible to employ both computers in the data acquisition system.

If the Zenith is not set as the System Controller, but the HP-85 is (as shown in Figure 2), then the Zenith cannot use the Command Library. This library is absolutely essential to do any type of communication through the bus. It would be impossible to transfer the data from the 7612D without the aid of the Command Library.

### Simplified Data Acquisition System

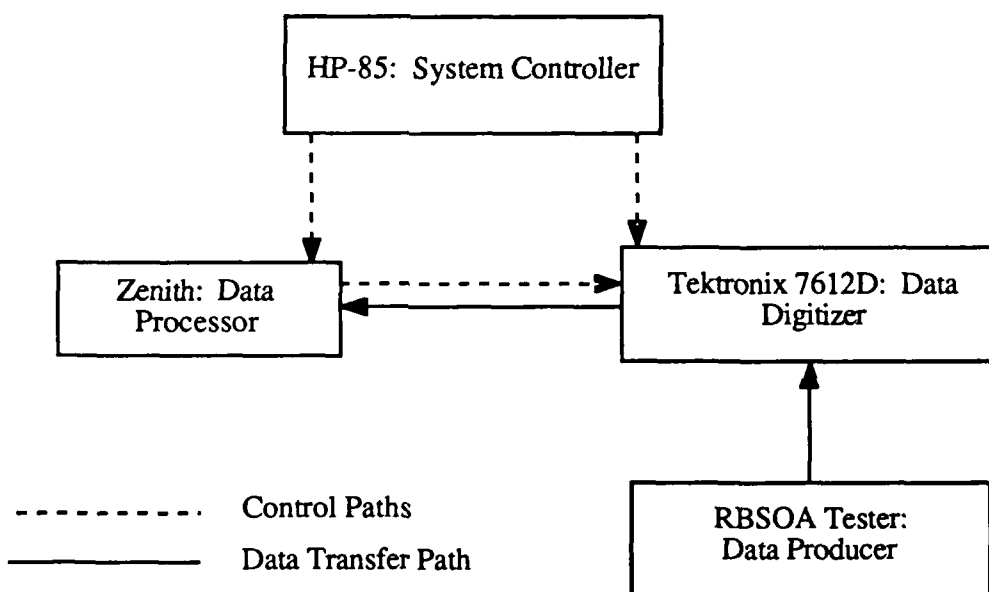


Figure 2. Schematic of a simplified data acquisition system incorporating the Zenith Microcomputer.

An effort was made to obtain the source code used to develop the Command Library from Hewlett-Packard. It was hoped that a modification of the code could be arranged so that communication via the bus would be possible without the interface card being set as System Controller. Hewlett-Packard was not forthcoming with the information, and the line of research was discontinued.

Without being able to pass control from the Zenith to the HP-85 while the Zenith was System Controller and being unable to get the information into the Zenith when it was

not the System Controller, it was decided to scrap the idea of using both computers in the improved system and concentrate on using the more capable Zenith, only.

## CHAPTER III SOFTWARE DEVELOPMENT

### BASIC Programming

With the decision to use the Zenith as the sole controller, it became necessary to develop a complete software system which could perform all the needed control and processing tasks. It would have to supply the test apparatus with the instruction it needed to acquire the specific data of interest, and the software would have to conduct all the applicable processing and information storage, retrieval and output functions. In short, the system would have to be able to do everything that the HP-85 could do. Re-invention of the wheel was, however, something to be avoided.

The first step toward developing the system software was to first determine in what language to program. The HPIB interface card's Command Library was capable of being used with a number of upper level computer languages including: Pascal, BASIC and C. All of these languages can be run on the Zenith given the proper compiler or environment. It should be noted that the control software for the HP-85 was written in BASIC.

From the beginning, time had been a crucial factor in all decisions made concerning this project. The quicker, the faster, more capable, Zenith could be brought on line, the better. As control software was already available in BASIC on the HP-85, initial effort were directed toward transferring the BASIC source code from the HP-85 to the Zenith in the hope of running the code, or a slightly modified version of the code to control the acquisition system.

The actual source code for the HP-85 is stored on the HP-9895A Flexible Disc Memory which is connected to the HP-85 via the HPIB, but the Zenith was also connected to the bus. It was thought that the Zenith might be able to transfer the files from the HP-9895A to the Zenith's own floppy disk drive by accessing them using the HPIB card. The code could then be run on the Zenith using the BASIC line interpreter, GWBASIC®.

Normally, the HP-85 talks to the disk drive using control codes stored on a special Read Only Memory (ROM) chip. If the Zenith were to properly address the HP-9895A, it would have to use these codes, but they are inaccessible from the Zenith. Additionally, none of the manuals for either the disk drive, the mass storage ROM or the HP-85 contained information on the necessary codes. Hewlett-Packard was again contacted and asked for information regarding the control codes for the HP-9895A, but they were unable to provide the necessary information. Instead they offered a disk drive controller software

package (part no. HP 88500B) which they claimed would talk to any of HP's HP-IB disk drives. When the software arrived, it proved to be incompatible with the HP-9895A. Again Hewlett-Packard was contacted, and asked why the package would not work. After a somewhat lengthy delay, a Hewlett-Packard representative responded by stating that the package would not work on the HP-9895A disk drives and that it had never been intended to do so. Further, HP did not and would not ever have a package available to control those disk drives from an IBM PC compatible. It was, therefore, not possible to directly access the source code on the HP-9895A.

While directly accessing the files on the HP-9895A was impossible, it was possible to perform a one time only transfer of the files from an HP-85 to an IBM PC compatible at an off-site location. The files were first transferred to a newer HP personal computer and then to another IBM PC compatible computer. Copies of the source code were then transported on five-and-one-quarter-inch floppy diskettes to the lab where this effort was being undertaken.

As mentioned earlier, time was a strong factor in the decision process. So, the quickest possible method of using the source code which had been transferred into the Zenith was attempted, running unmodified code in GWBASIC. This met with failure.

On examination of the code, it was discovered that the HP BASIC that was running on the HP-85 was significantly different than the GWBASIC that was used on the Zenith. The first problem with directly running the software was in the file format in which the source code was transferred. The HP-85's screen can only display 40 columns of data or text in any particular row. This is also how the data is stored on the HP-9895A disk drives. If a BASIC line was longer than 40 characters, the line was wrapped around to the next row down. When the code was read in from the HP-9895A to the HP-85, the line or word wrap was ignored as far as the function of the program was concerned.

GWBASIC interprets lines slightly different than does the HP-85's BASIC. Where the HP-85 will ignore the word wrap, the Zenith, using GWBASIC, will not. GWBASIC requires every row to begin with a line number. If the line number is missing as in the case of the transferred files where line wrap has occurred, GWBASIC will declare an error and stop reading in the source code. To illustrate this, here is an example:

```
70 FOR I=CURSROW to 64 @ DISP @  
NEXT I
```

This is the format of the files that were transferred over from the HP-85, but the Zenith using GWBASIC would require the following format for an input source code file.

```
70 FOR I=CURSROW to 64 @ DISP @ NEXT I
```

However, just because GWBASIC reads the file in does not mean it will run it unmodified.

To correct this problem, a program which would remove the word wrap from the HP-85 source code files that were in the Zenith had to be written. If there had been only a few files, it would have been possible to correct the files using a screen text editor; however, there were 20 files for a total of over 2000 lines of code. After writing the program and running the source code files through it, again an attempt was made to run the HP-85 software using GWBASIC. At this point, it was discovered that GWBASIC was incompatible with the HP-85's BASIC.

When a copy of the HP-85's source code was first obtained for the Zenith, it was known that, at the very least, certain sections would have to be rewritten. It had been thought that the time involved in this rewrite would be minimal; however, with the discovery of the compatibility problems in the source code and the resulting requirement to rewrite the source code line by line, it became evident that this avenue of approach was no longer worthwhile. The expense in time and the resulting product could not be justified when the entire control software could be re-programmed using another language in less time and end up with a more versatile and faster system.

At this point, it was decided to begin developing software using one of the languages which was supported by the Command Library, Microsoft Pascal (MS-PASCAL). MS-PASCAL has several distinct advantages over BASIC. First, it is a compiled language which means much faster execution. It is a structured language which allows for more complex programming with easier debugging. Lastly, and probably most importantly, MS-PASCAL allows the program to utilize math coprocessors if they are installed in the computer. In the case of the Zenith, an 8087 Math Coprocessor is installed.

A little background on the 8087 Math Coprocessor before proceeding. The 8087 is designed to do nothing but floating point mathematical operations. It does these operations typically over 100 times faster than they can be done by the standard 8088 Microprocessor. In heavy math applications, the 8087 can speed the operation of the application by as much as 15 times. This increase in speed is critical if data acquisition system is to be able to process in real time.

It should also be noted that BASIC, any type of BASIC, cannot utilize a math coprocessor. It becomes obvious that any new code that needs to be written should be written in a language that can use an 8087. MS-PASCAL could, and so, it was selected for use in programming the new system software.

### MS-PASCAL Programming

At the start of the programming effort, a set of requirements was created. Any software developed was expected to meet all of the stated requirements. What follows is a list of these requirements:

1. The system must perform initial setup procedures which include: poll bus to insure the 7612D is on line and functioning, calibrate the 7612D, set the system time-out error checking level, check for any other devices on-line, identify those devices and prepare them for use.
2. Perform the necessary bookkeeping functions, such as: set up and maintain an index of devices, keep track of how many tests have been run on a particular device, maintain a record of which forward and reverse base drive current settings have been used and keep a count of the number of breakdowns compared to the number of tests on a given device at a particular setting.
3. Allow the 7612D to acquire  $V_{ce}$  and  $I_c$  and then enter these waveforms into the Zenith for processing and storage.
4. Process  $V_{ce}$  and  $I_c$  to obtain P and E.
5. Store all data on floppy disk.
6. Repeat measurements in as efficacious a manner as possible.
7. Plot specific waveforms from selected tests on the HP-7470A plotter.
8. Compile and output a table of breakdown values on a particular device. Breakdown values include:  $V_{ce}$  and  $I_c$  at the moment the protection circuit fires, the forward and reverse base drive values, the energy at breakdown and the time to breakdown.

Work was directed toward obtaining each requirement in the order they are presented above. Prior to achieving the last two goals in requirement 1, it was determined that the only device to be used in the data acquisition process would be the 7612D; therefore, it would not be necessary to check for any other devices on-line or identify those devices and prepare them for use. Those requirements were dropped.

The first snag in programming was encountered after approximately 550 lines of code had been written. A stack overflow error was encountered. Stack overflow occurs when the amount of memory used by the variables contained in the stack exceeds a specific limit. This problem is demonstrated in Figure 3 below.

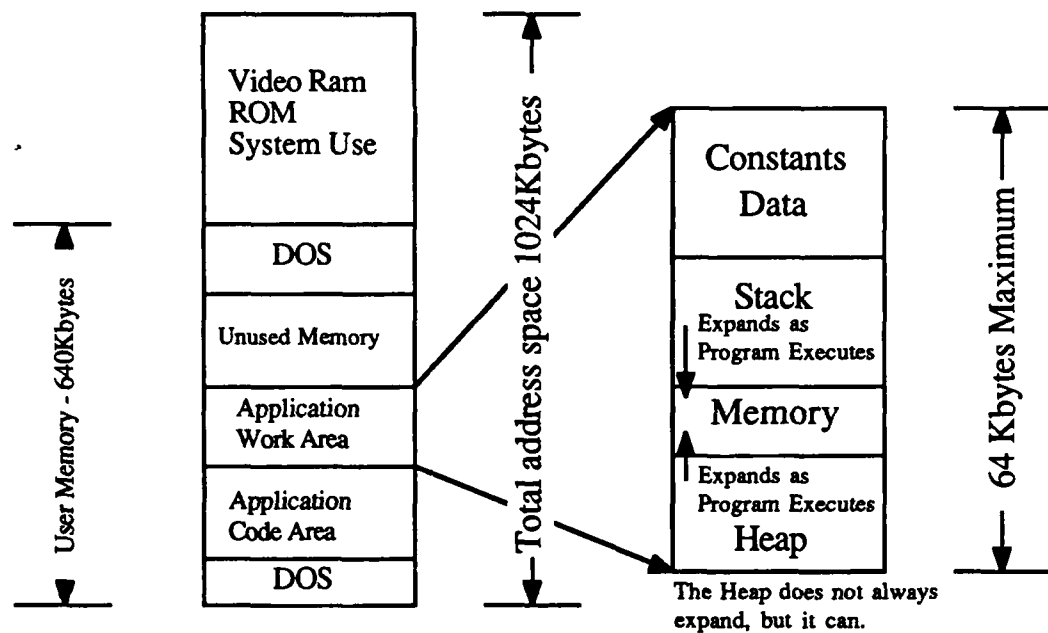


Figure 3. Drawing showing DOS memory allocation along with MS-PASCAL working area memory allocation.

At the time the error was encountered, it was unclear as to the cause. The individual procedures within the system software did not appear to contain enough variables or functions to fill the stack space. Without really understanding the problem, a solution of sorts was found, but it allowed only an additional 200 to 250 lines of code to be added.

Later, the probable cause of the problem was located. At the end of each program's execution, an error message, "Error Code: 2001, Null Pointer Assignment," would be displayed. Since the program was finished running, the error message was merely annoying; however, the source of the message was investigated. Microsoft, the maker of the Pascal compiler, was contacted. Their analysis of the problem was that a variable pointer had not been cleared before the end of the program.

No pointers were used in the program under development at that time or at any time during the course of this project. The only possible source of a pointer assignment was the Command Library for the HPIB.

From this information, a theory was developed. Apparently, the functions contained in the Command Library use pointers, and the developers forgot to clear their pointer assignments before closing the functions. Without the specific pointer designations, it is impossible to clear the assignments.

As the program executes, it calls the Command Library functions. Each time there is a function call, pointer assignments are made which use up stack space. The stack space cannot be reclaimed unless the assignments are released. They are not; therefore, as execution continues, more and more stack space is used, until the available stack space is filled and the program crashes.

Whether the theory is correct or not, it was not possible to extend the program length beyond the limit encountered. So, an alternative method was developed which proved to be a workable solution to the stack overflow problem. The Disk Operating System on the Zenith (ZDOS) provides a batch file utility which can be used to call separate programs in a sequential fashion. ZDOS, also, has an ability to create loops in the execution structure of the batch file. With this capability, it is possible to break the system software into several separate programs which execute sequentially and perform all the necessary data acquisition and processing tasks.

As each program executes, it fills up the useable stack space, but when execution is completed and the program ends that stack space becomes available for another program to use. Thus, if the original program is broken at the point just prior to the stack overflow, the memory is cleared by the ending of one program and the beginning of another before the problem is encountered.

Each program in the sequence does, however, require a certain amount of information from the previously executed programs. This information would be kept as global variables in a normal Pascal program. In order for the new programming scheme to be effective, a method for passing data between the programs, quickly, efficiently and without requiring user interference had to be developed.

Another very useful utility now comes into play. One of the common utilities on almost any IBM PC compatible is a RAM disk. This is a program which sets up a segment of user memory for use as a RAM disk. RAM disks behave exactly as normal disk drives do, meaning they can be written to and read from in the same fashion as floppy or hard



disks, only much faster. Using the utility, RAMDISK.SYS, a new drive was setup on the Zenith. A text file called PASSER.DAT was created to contain those global variables necessary to the execution of the software system.

With the RAM disk to contain the necessary global variables, an extended program architecture or hierarchy was created. In an application such as this, where there are distinct steps that do not require a high degree of interaction with other parts of the program and execution follows a set order, the batch file system performs just as well as any other program structure available. The difference between the new and old hierarchy is shown in Figure 4.

Normal Pascal Hierarchy	Modified Pascal Hierarchy
Program Level - Highest Normal Level in Pascal	DOS Level - Batch file calls Programs as if they were Procedures
Procedure Level - Called by Program Level	
Sub-Procedure Level - Called by Other Procedures. Structurally the same as Procedure Level, can even be called by Program Level	Program Level - Same as Normal Pascal Hierarchy from here on

Figure 4. Normal Pascal program hierarchy compared to modified hierarchy.

The RAM disk was given the drive designation G:, and all communications with that disk are preceded with that address designation. (On a Zenith Model 158, the drive designations A through F are already used whether a physical drive for the specific designation is present or not. For example, on the machine used in this work only drive designations A,B and C actually refer to a physical drive. D,E and F are completely unused. RAMDISK.SYS automatically uses the first unassigned drive designation available, but even though D,E and F were unused, they were assigned in the Zenith hardware/software reserved designations.)

PASSER.DAT consists of various parameters kept on set lines and positions in a text file. The first line contains a string of ten toggle switches. The toggles are simply

characters like "Y" or "N." Depending upon the state of a particular letter, an action may or may not be taken in a given program. This is one way of letting all the succeeding programs know what actions need to be taken depending upon what has previously occurred. In the system completed in this effort, only five of those toggles are used. The other five are for use by follow-on programming efforts and lends the system some expandability.

Lines two and three of PASSER.DAT contain the device number and the system time-out error value, respectively. The device number is an integer used as an index value which references the device record contained in the device index file, INDEX.DEV. With the device number it is possible to obtain all the information currently stored on a specific device in the index file. It also tells the preceding programs which device is currently installed on the tester. The system time-out error value is a real number value which tells each program that performs Input/Output (I/O) functions over the HPIB how long to wait for a response before declaring an error.

Line four contains the most recent base drive forward and reverse bias current settings. These values are stored as part of all the files created during the acquisition and processing phases. They represent the variables in the test, and it is important that all data taken during a particular test run be labeled with these values as a protection against mistakenly grouping and processing files from different tests with different settings.

Lines five and six contain file designations. The fifth line contains the name and location of the bookkeeping file which contains a listing of the various forward and reverse bias settings, a comment line for descriptive information and a count of the number of times a particular device was tested at given bias settings and the number of times the device entered into Reverse Bias Second Breakdown. The sixth and final line contains the root filename for storage of incoming data and the files created during processing.

The individual system control programs are stored on the hard disk until needed. Although storing the files on the hard disk results in slower overall program execution, since between each program the computer must stop and load the succeeding program into memory before it can be executed, the delay is so minimal that it is barely noticeable. Primarily, this is due to the relatively small size of the programs (less than 50 kilobytes each) and the inherent speed of the hard disk.

The two main advantages achieved by using a batch file execution scheme are, first, it is very versatile in the application at hand. If at any time it is decided to add additional processing or analysis steps, all that need be done is write an additional program in any compiled language (such as PASCAL, C, FORTRAN or Compiled BASIC), place the

executable version of the program into the appropriate directory on the Zenith's hard disk and include a statement in the controlling batch file to call the new program at the appropriate point in the overall acquisition cycle.

The second main advantage attained by the batch file approach is the alleviation of the problem that led to its adoption in the first place, stack overflow. With this execution scheme there are no limits to the size of the system software package save the size of the available secondary storage on the Zenith.

### Turbo Pascal Programming

As mentioned above, the new hierarchy allows the use of any compiled language as a programming tool. Each language typically offers features not found or difficult to use in other languages. This makes the different languages convenient for use in particular applications. Sometimes compilers for the same language produced by different publishers will also have different features which makes it more advantageous to use one compiler over another. For instance, MS-PASCAL has the ability to use libraries created by third parties. This is the feature that allows the Zenith to use the HPIB Command Library and control the 7612D.

Another Pascal compiler which has some other advantages for use in program development is Borland International's Turbo Pascal (or Turbo for short). Turbo is, generally, a much friendlier development environment. Where MS-PASCAL requires the programmer to create the source code in a separate text editor, Turbo provides an integrated text editor and compiler.

After developing the source code for MS-PASCAL, the programmer must exit the text editor and attempt to compile the code. MS-PASCAL can take up to three or four minutes for a 300 line program. If any errors are encountered, they are printed out all at once. The programmer must then either get a hard copy listing of the errors or attempt to remember them all, enter the text editor, make the necessary corrections, exit the text editor and attempt to compile the program again.

In Turbo, the editor and compiler are bundled together. After the source code is developed, two key strokes will bring the programmer into the compiler. To compile, only one key needs to be pressed, and the compilation begins. A typical 300 line program would take less than 15 seconds to compile. If any errors are encountered, the compiler stops, enters the editor and brings the programmer to the exact location in the source code where the error occurred. This makes debugging a much simpler process.

Although Turbo had several features that made it desirable, it could not use external libraries; therefore, it could not be used to communicate between the Zenith and the instruments on the HPIB. For those applications where control of the HPIB was not necessary, the greater ease-of-use of Turbo made it the compiler of choice. These applications included data storage, processing and output. Thus, those programs not concerned with HPIB control were all written and compiled using Turbo.

### System Software

The collection and processing system software consists of 12 individual executable programs and one batch file routine. The batch file routine controls data acquisition which involves only seven of the 12 programs. The remaining five are referred to as the ancillary programs and perform necessary maintenance tasks as well as providing data output in various forms. A complete commented listing of all of the programs and the batch file are contained in the appendices.

### Data Acquisition Programs

The batch file routine, TAKEDATA.BAT, performs the function of the main program block in the normal Pascal hierarchy. It keeps the flow of program execution orderly and well defined. It, also, checks the amount of storage available on the floppy disk drive where all data will eventually be stored, it will cause those files which have been created on the RAM disk during one data acquisition cycle to be copied over to the floppy disk drive and then delete those files from the RAM disk to conserve space. At the end of the acquisition cycle, it will check for the existence of a file on the RAM disk called CONTINUE.ANS (created by the program REPEATER.COM). If the file is found, the batch file will loop back to the beginning of the data acquisition cycle and start the process going again. If the file is not present, the cycle is ended and any remaining files contained on the RAM disk are copied over to the floppy disk drive and the batch file execution is terminated.

The first program called by TAKEDATA.BAT is HARDWARE.EXE. This program is compiled using MS-PASCAL and is used to set the system time-out error value, create the file PASSER.DAT and calibrate the 7612D. Calibration is done by obtaining a zero level line on both Channel A and Channel B of the 7612D by setting the input coupling to GND and manually triggering the 7612D (11). These zero level lines are then read in as a series of 2048 points for each channel.

Each point consists of one byte. The value of the byte indicates the vertical deflection where each division on the Tektronix 7A16P Programmable Amplifiers is divided into 32 levels, the difference between successive levels being equal to a binary increment of one. So, for the lowest point of the vertical deflection, the byte value would be  $00000000_2$  (decimal equivalent = 0). This is equivalent to the largest negative number displayable by the 7612D. The highest point would have a byte value of  $11111111_2$  (decimal equivalent = 255) which is the largest positive number displayable by the 7612D. The midpoint between the high and the low would be  $01111111_2$  (decimal equivalent = 127). This midpoint value is really the equivalent of zero with deviations from it being the same as excursions from zero.

Since the data contained in the 7612D is binary in nature, it would be nice to be able to read and manipulate the data in that state; however, neither MS-PASCAL or Turbo Pascal are very efficient at handling single byte binary numbers. Both compilers do have a facility to handle ASCII characters, and single byte binary numbers can be referenced by their ASCII equivalent.

An ASCII character is a symbol which is used to refer to a particular single byte binary value. For instance, the byte,  $01000001_2$ , is referenced by the symbol, "A." In the extended ASCII character set used by both MS-PASCAL and Turbo, all 256 possible single byte binary values have a corresponding symbol.

It is possible to read a single byte binary value, such as a data point, into a program as an ASCII character. The program will handle it exactly as an ASCII value which means it can be written to disk or read from disk as if it were text. By treating the data from the 7612D as ASCII values, it is possible to read the binary data and handle it within the program easily. When it comes time to evaluate or use the data, it can be read from where it was stored on disk and translated into decimal form by the built-in procedure "ORD." The data can then be handled as integer values.

HARDWARE.EXE first reads in the total 4096 points from Channels A and B. The program reads those points in as two sets of 16 128 character long ASCII strings. The program will read the data one 128 character string at a time and write each string out to the RAM disk one at a time until all the points have been entered. This method of reading the data from the 7612D has been found to be the fastest available given the limitations of the 7612D, MS-PASCAL and the HP-IB Command Library. The total operation requires less than a second and a half. All of the programs which read data from the 7612D use this procedure.

The first 2048 characters (points) of the file created on the RAM disk is then read into an array in the program, the ASCII characters are translated into their integer equivalents, all 2048 points are then averaged and the average compared to the equivalent zero value of 127. If the average is within  $\pm 1$ , Channel A is considered calibrated.

The second 2048 characters are then read and processed the same way. This is the data from Channel B. If neither channel needs to be calibrated, `HARDWARE.EXE` ends and the batch file takes over. If one or both of the channels need to be calibrated, a message is output to the operator to adjust the position knob on the 7612D in whatever direction is necessary. The operator then re-triggers the 7612D, and the program reads the data as before. This continues until both channels on the 7612D are calibrated.

The program `GETDEVIC.COM` is called next. This program is responsible for setting up and maintaining the index of devices contained on the file `INDEX.DEV`. The program provides a listing of all devices currently kept on `INDEX.DEV` and prompts the operator to input the device number corresponding to the device being tested. Using this value, the program retrieves the proper bookkeeping and storage filenames from `INDEX.DEV`. The filenames are then written out to `PASSER.DAT` in lines five and six.

Just prior to the program `GETDEVIC.COM` the batch file enters the top of its execution loop. This means that `HARDWARE.EXE` is executed only once during a given data acquisition session. Where a data acquisition session is defined as the period of time between the operator beginning `TAKEDATA.BAT` until the operator terminates the session when queried if he or she desires to continue or not. Because the loop marker is prior to `GETDEVIC.COM`, it is likely that when `GETDEVIC.COM` is called it might not be the first time through the acquisition process. If there has been no change in the test device (ie, the TUT has not been replaced), then there is no need for `GETDEVIC.COM` to run as `PASSER.DAT` contains the proper bookkeeping, storage filenames and device number. To save time and avoid having to go through `GETDEVIC.COM` when it is not necessary, the first toggle switch in the first line of `PASSER.DAT` has been assigned to toggle the execution of `GETDEVIC.COM` on or off. The program is still loaded into memory; however, the first step it takes is to read `PASSER.DAT`. If the toggle is set to "N," then the program will not execute, and control is returned to the batch file.

`STORDATA.COM` is the next program to be executed. It is responsible for setting up, maintaining and updating the bookkeeping file. The program executes once during each acquisition loop, and it is, also, responsible for setting the condition of toggles 2,3 and 6. These toggles control the execution of the programs which read the data in from the

7612D and process it. Toggle 2 is used to control the execution of PROCESS3.EXE which reads  $V_{be}$  from the 7612D. Toggle 3 informs the succeeding programs whether or not breakdown has occurred, and toggle 6 controls the execution of PROCESS1.EXE which reads  $V_{ce}$  and  $I_c$ . Toggle 6 also controls the execution of PROCESS2.COM which calculates P and E from  $V_{ce}$  and  $I_c$ . After STORDATA.COM, PROCESS1.EXE, PROCESS2.COM and PROCESS3.EXE execute in that order. The function of these programs are given above.

The final program to execute in the data acquisition phase is REPEATER.COM. This program performs two functions. First, it queries the operator as to whether or not he or she would like to continue. If the answer is affirmative, the program creates the file CONTINUE.ANS on the RAM disk which will tell the batch file to proceed to the beginning of the loop and start again. A negative answer will cause nothing to happen and the batch file will terminate after some clean up procedures such as copying the new book-keeping file to the floppy disk drive from where it was temporarily placed on the RAM disk. The second function is to update the INDEX.DEV file. During the data acquisition loop, the storage filename might have been altered. The alteration must be kept to maintain a proper record of storage filenames.

With the execution of REPEATER.COM, the batch file program has reached the end of one data acquisition cycle. Repeated measurement would follow the same steps save the call to HARDWARE.EXE.

### Ancillary Programs

There are five programs included with the system software which are not used during the data acquisition process. Three of the five are concerned with providing data output. One is used in maintenance of the file INDEX.DEV, and the last is to either calculate or recalculate P and E from  $V_{ce}$  and  $I_c$ .

The three output programs are PLOTDATA.COM, TABULATE.COM and SORTABLE.COM. The largest of these programs is PLOTDATA.COM, and the source code used to generate the program represent roughly a third of the entire programming effort. As its name implies, it is used to generate a plot of the data. The program is capable of generating one of two different outputs. The first plots the waveforms for  $V_{ce}$ ,  $I_c$ , P and E as well as printing out the breakdown values for  $V_{ce}$ ,  $I_c$  and E, time to breakdown, forward and reverse base current bias settings, device number and test series number. Figure 5 is an example of one of these plots.

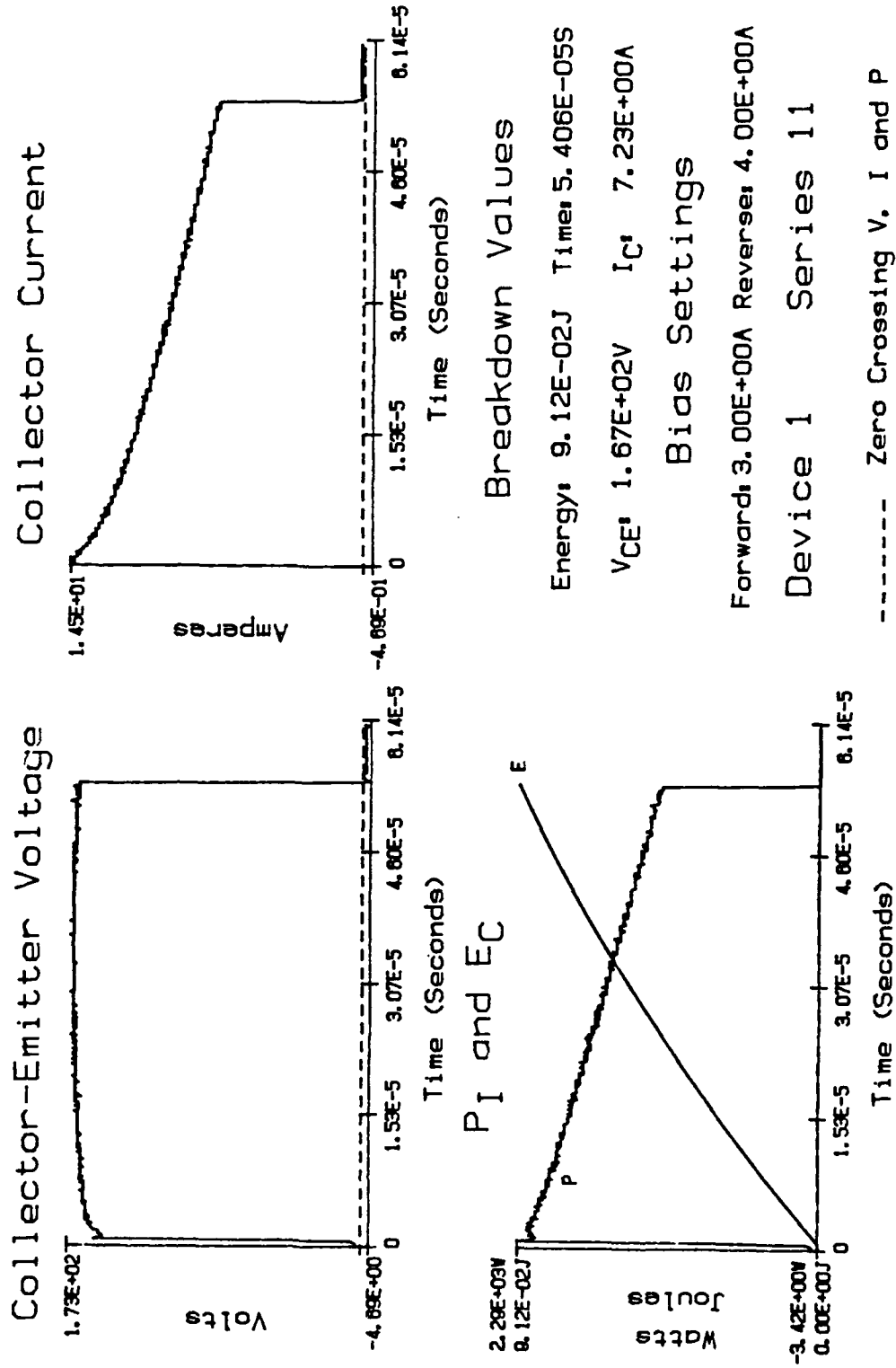


Figure 5. Example of plot type 1.



The time to breakdown is defined as the time interval over which the instantaneous power is being integrated. The test series number is the order of occurrence of the test run where the data being plotted was collected. For example, if the test series number was six, it would mean the data being plotted was from the sixth test made on that particular device.

The second type of plot available is one which plots out all of the previous information, and in addition, plots  $V_{be}$ . The waveforms for  $V_{ce}$  and  $I_c$  are superimposed upon on another, and  $V_{be}$  takes the place of  $I_c$ . An example of this plot format is shown in Figure 6.

The two other programs involved in producing outputs are TABULATE.COM and SORTABLE.COM. TABULATE.COM does not directly produce output but merely tabulates the breakdown values for all the tests done on a particular device and write these values to disk. On disk, these values are not stored in a fashion which can be directly read. This is where SORTABLE.COM comes in to play.

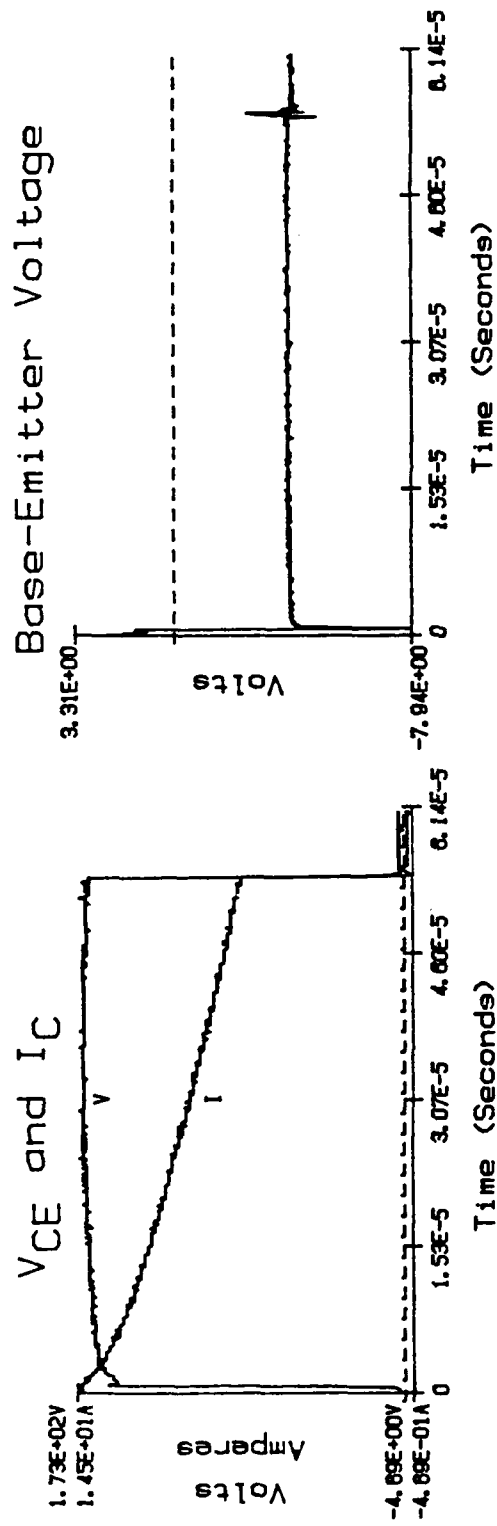
SORTABLE.COM is a routine to sort and output the assembled breakdown values created during TABULATE.COM. It provides the ability to sort the file created in TABULATE.COM on one or two key values. The sorting can be done using any of the breakdown values as keys. It is, therefore, possible to produce several different output based upon the same tabulated file. SORTABLE.COM can provide output either to a text file on disk for use in latter printing or directly to a line printer. An example of SORTABLE.COM's output can be seen in Figure 7.

The last two programs contained in the software system are EDITLIST.COM and PRCSLTWO.COM. EDITLIST.COM is used to create, repair, maintain and add to the file, INDEX.DEV. This capability was found to be necessary during software development as there are numerous ways for INDEX.DEV to be damaged, and without it, none of the programs will execute properly. It was, therefore, imperative to create a program with the capability of restoring any data lost with the damage of INDEX.DEV.

The final program, PRCSLTWO.COM, is merely a repeat of PROCESS2.COM. Except, it is capable of working outside the batch file environment. It is used to either calculate or re-calculate the values for P and E.

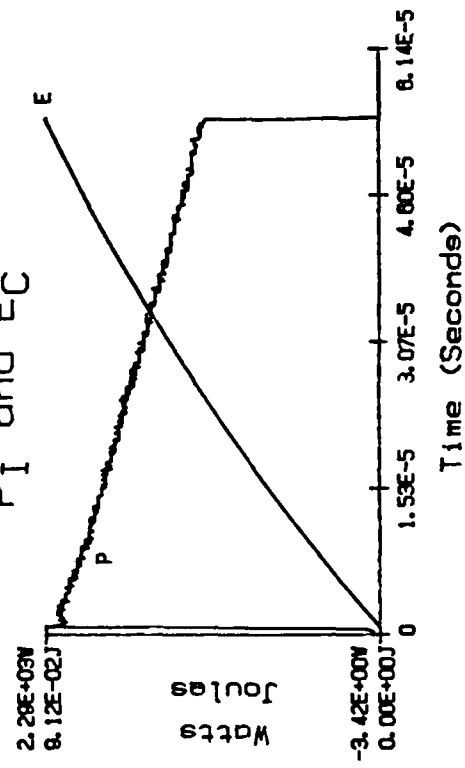
#### Files Created and Individual Structure

An important element of the system software is the files it creates. The first file created is PASSER.DAT which was discussed in detail earlier. The next file created or used is the bookkeeping file.



Time (Seconds)

P<sub>I</sub> and E<sub>C</sub>



Time (Seconds)

Breakdown Values

Energy: 9.12E-02J Time: 5.406E-05S

V<sub>CE</sub>: 1.67E+02V I<sub>C</sub>: 7.23E+00A

Bias Settings

Forward: 3.00E+00A Reverse: 4.00E+00A

Device 1 Series 11

----- Zero Crossing V and P

..... Zero Crossing Current

Figure 6. Example of plot type 2.

For primary key of Forward Base Current = 1.00E+00 A

For secondary key of Reverse Base Current = 6.00E-01 A

Collector-Emitter Voltage ..... 1.34E+02 V

Collector Current ..... 2.47E+00 A

Energy at Breakdown ..... 1.19E-01 J

Time at Breakdown ..... 1.21E-04 Sec

For primary key of Forward Base Current = 2.00E+00 A

For secondary key of Reverse Base Current = 1.00E-01 A

Collector-Emitter Voltage ..... 1.59E+02 V

Collector Current ..... 5.80E+00 A

Energy at Breakdown ..... 1.14E-01 J

Time at Breakdown ..... 7.58E-05 Sec

For secondary key of Reverse Base Current = 2.00E-01 A

Collector-Emitter Voltage ..... 1.62E+02 V

Collector Current ..... 6.36E+00 A

Energy at Breakdown ..... 1.10E-01 J

Time at Breakdown ..... 7.01E-05 Sec

For secondary key of Reverse Base Current = 4.00E-01 A

Collector-Emitter Voltage ..... 1.61E+02 V

Collector Current ..... 6.36E+00 A

Energy at Breakdown ..... 1.05E-01 J

Time at Breakdown ..... 6.79E-05 Sec

For primary key of Forward Base Current = 3.00E+00 A

For secondary key of Reverse Base Current = 6.00E-02 A

Collector-Emitter Voltage ..... 1.52E+02 V

Collector Current ..... 2.98E+00 A

Energy at Breakdown ..... 1.47E-01 J

Time at Breakdown ..... 1.25E-04 Sec

For secondary key of Reverse Base Current = 1.00E-01 A

Collector-Emitter Voltage ..... 1.63E+02 V

Collector Current ..... 5.33E+00 A

Energy at Breakdown ..... 1.13E-01 J

Time at Breakdown ..... 7.80E-05 Sec

Figure 7. Example output from SORTABLE.COM. Sorted on forward and reverse base drives, respectively.

For secondary key of Reverse Base Current =  $8.00\text{E-}01$  A

Collector-Emitter Voltage .....  $1.64\text{E+}02$  V  
 Collector Current .....  $6.47\text{E+}00$  A  
 Energy at Breakdown .....  $1.03\text{E-}01$  J  
 Time at Breakdown .....  $6.49\text{E-}05$  Sec

For secondary key of Reverse Base Current =  $4.00\text{E+}00$  A

Collector-Emitter Voltage .....  $1.67\text{E+}02$  V  
 Collector Current .....  $7.23\text{E+}00$  A  
 Energy at Breakdown .....  $9.12\text{E-}02$  J  
 Time at Breakdown .....  $5.46\text{E-}05$  Sec

Figure 7. Continued.

The bookkeeping file is a text file and contains, first, a listing of forward and reverse bias current settings. The settings are written in ten lines of two numbers. The first number in the line is the forward bias setting, and the second is the reverse bias setting. Initially, all the values are zero, but as tests are performed using different settings, one by one the zeros are changed to actual values. A maximum of 10 forward and 10 reverse settings are allowed for a total of 100 different combinations. Following the settings segment of the bookkeeping text file is a comment line. This is an 80 character string which is used to record operator comments about the TUT. Its use is completely optional.

The last segment of the bookkeeping file is 10 lines of 20 numbers each. The first line corresponds to the first reverse bias setting (bias settings segment, line one, second number). All of the numbers on this line correspond to tests made on the TUT using that particular setting. The first two numbers of each line correspond to the first forward bias setting (bias settings segment, line one, first number), the second two numbers to the second forward bias setting (bias settings segment, line two, first number), the third two to the third forward bias setting and so on. In this way, each two numbers are connected to one forward and one reverse bias setting. The first number in each pair of numbers represents the number of times the TUT broke down at the corresponding bias settings. The second number represents the total number of time the device was tested at those settings.

The last segment is used to help determine device threshold tolerances. It is an added feature above and beyond what was really necessary in the system, but it is thought that in future efforts it might be useful. Whether or not the feature is used, the program

which controls it, STORDATA.COM, must be run as part of the data acquisition process as that program also performs other functions vital to the overall system.

The filename for the bookkeeping file has the following general structure: DEVxxxLL.BK. The "xxx" represents the device number (for device 1, "xxx" = 001), and the "LL" specifies the inductor attached to the collector.

The next set of files created are all related to each other by a common root name, BDxxxLLq.yyy. This root name is explained in Figure 8. The filenames for all the data taken on one particular test run are all the same except for the character represented by "q." The changes made to "q" are, also, explained in Figure 8. All of the test runs on a particular device will have the same "xxx" value but different "yyy." This allows anyone to determine which group of files belongs to a particular device and/or test run, as well as what is contained in the file.

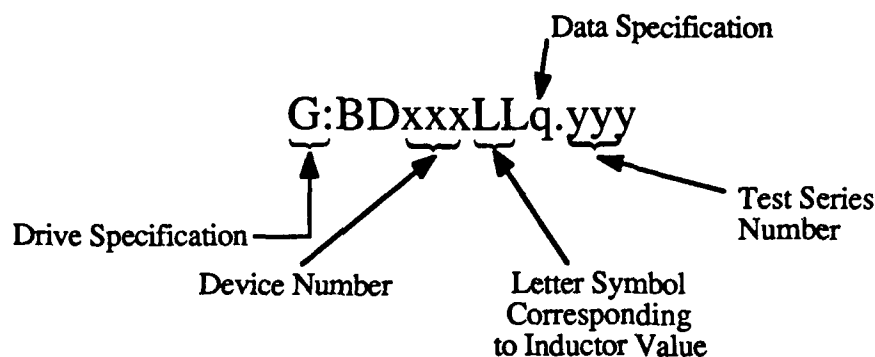
Files A, B and C (where A, B and C refer to what replaces "q") all have the same basic format. The only difference is in file A which contains an extra line placed at the very beginning of the file. The line contains a single character a "Y" or a "N." Examining the character is an easy way of determining if breakdown occurred during the test run or not. "Y" indicates breakdown. "N" indicates no breakdown and, probably, safe turn off.

Line two of file A is exactly the same as line one of B and C. It contains the forward bias current setting and the reverse bias current setting. The next line contains, first, the Volts per Division setting of the appropriate plug-in on the 7612D, second, the horizontal scale factor (time between each sample) and, last, the probe correction factor. The probe correction factor is use to make sure the scale is properly set on the vertical deflection. It is set to one on files A and C since correction is automatically done by the Channel A plug-in. The file B value should be set to 10 which is the correction factor necessary to convert the value of the vertical deflection in volts to amperes. The final "line" is a string of 2048 ASCII characters which are the actual data points.

File E contains a column of 2047 numbers. The numbers represent the cumulative energy up to and including the particular time step corresponding to the numbers position in the column (ie, The fifth number is the total energy up to and including the fifth time step.).

File P is a mixture of the formats from the preceding files. The first line contains the forward and reverse bias settings. The second line contains the total cumulative energy to breakdown, horizontal scale factor and vertical scale factor, respectively. The third line is a string of 2048 integers which are the result of multiplying the ordinal (integer) values

of the ASCII characters in files A and B. These integers represent the unscaled values of Instantaneous Power.



### Data Specification Key

- A: File contains  $V_{ce}$  waveform data
- B: File contains  $I_c$  waveform data
- C: File contains  $V_{be}$  waveform data
- E: File contains Cumulative Energy values
- P: File contains Instantaneous Power waveform data

Figure 8. Storage filename interpretation.

The final file created by the software system is the file created by TABULATE.COM. The file is a direct access file with a maximum of 999 records, one for each possible test series number. Each record consists of an integer value which contains the particular test series number for the data contained in the record. Additionally, each record contains six real number values corresponding to the forward and reverse bias current settings, the Collector-Emitter Voltage at breakdown, the Collector Current at breakdown, the Energy at breakdown and the time to breakdown.

## CHAPTER IV

### VERIFICATION OF SOFTWARE SYSTEM

#### Accuracy of Measurement

Prior to detailing the steps used to verify the correctness of the data and the processing thereof, the issue of the accuracy of the data and how it effects the calculated values needs to be discussed. The principal source of inaccuracy is the 7612D and the 7A16P combination. Once the data has been digitized, the transfer to the Zenith and the ensuing calculations performed on the data are basically error free (ie, The round off errors encountered during the computations are negligible compared to the error introduced by the 7612D/7A16P combination.). For this reason, discussion of the accuracy of the measurement and data processing results will focus on the accuracy of the 7612D/7A16P.

There are two main sources of error introduced in the digitizer. First, there is the Digital-to-Analog (D/A) conversion error. Second, there is the error due to the 7612D's electrical characteristics. Basically, the second source of error is a catch-all for the error not accounted for in the D/A conversion error.

Looking first at the D/A conversion error, according to reference 11, the error is a function of the frequency of operation of the D/A circuitry. This means that depending upon the sampling period selected by the operator, the error will change. Typical operation of the 7612D in the data acquisition process involves using sampling periods of between 20 and 40 nanoseconds (ns). Using values provided in reference 11 and interpolating, yields an estimated Signal-to-Noise Ratio (S/N) of 32dB, or an error of approximately 0.1%. This will be the typical worst case D/A conversion error and will be used to estimate the total measurement error.

The error due to the 7612D's electrical characteristics is provided by reference 11. The stated value is 2%.

Combining the two sources of error gives the total measurement error,  $e_m$ , as 2.1% which can be approximated simply as 2%. The data taken from each channel will be estimated to have this error. Thus, all of the plots created by directly using the data from Channel A and B will contain up to 2% error, as will the breakdown values,  $V_{ce}$  and  $I_C$ , which are calculated directly from the Channel A and B data.

The error found in P and E must be calculated. Let  $a_A$  and  $a_B$  be the actual value of the data being measured (ie, no error) and  $e_A$  and  $e_B$  be the amount of error in the mea-

surement. Then,  $(a_A + e_A)$  would be the value stored on Channel A of the 7612D and read by the Zenith. Likewise,  $(a_B + e_B)$  would be stored on Channel B.

In the case of calculating P, the data from Channel A is multiplied by the data from Channel B.

$$(a_A + e_A)(a_B + e_B) = a_A a_B + a_A e_B + a_B e_A + e_A e_B . \quad (1)$$

But, the worst case for  $e_A$  is,  $e_A = \epsilon_A a_A$  where  $\epsilon_A$  is the total measurement error,  $e_m$ , for Channel A, and the worst case for  $e_B$  is,  $e_B = \epsilon_B a_B$  where  $\epsilon_B$  is  $e_m$  for Channel B. Thus, for the worst case,

$$a_A a_B + a_A e_B + a_B e_A + e_A e_B = a_A a_B + a_A \epsilon_B a_B + a_B \epsilon_A a_A + \epsilon_A a_A \epsilon_B a_B . \quad (2)$$

The total error,  $e_{mp}$ , in the calculation of P can be found by using the formula,

$$\frac{(\text{Actual} - \text{Estimated})}{\text{Actual}} = e_{mp} . \quad (3)$$

The "Actual" value is the  $a_A a_B$  term while the "Estimated" value is  $a_A a_B + a_A \epsilon_B a_B + a_B \epsilon_A a_A + \epsilon_A a_A \epsilon_B a_B$ . The total error in the power calculation is then,

$$e_{mp} = \epsilon_A + \epsilon_B + \epsilon_A \epsilon_B . \quad (4)$$

However, the value,  $\epsilon_A \epsilon_B$ , is negligible and can be neglected. The end result for the power calculation total error is,

$$e_{mp} = \epsilon_A + \epsilon_B = 0.02 + 0.02 = 0.04 = 4\% . \quad (5)$$

The total error of the energy calculation,  $e_{me}$ , can be determined in a fashion similar to that used above for the power calculation. Instead of the subscripts A and B referring to the channel, let them now refer to value of the Instantaneous Power and the time, respectively. Thus,  $e_{me}$  becomes,



$$\epsilon_{me} = \epsilon_A + \epsilon_B = 0.04 + 0.000035 \approx 0.04 = 4\% . \quad (6)$$

Where the value,  $\epsilon_B$ , is provided by reference 11.

### Verification

An important step in the overall development of the software system is proving that it works. Verification is done as a two step process. The first step being to prove that the data is accurately read from the 7612D and that it is properly scaled. The second step is to demonstrate that the processing of that data to find P and E is done correctly.

The first step is the easiest to prove as it can be done pictorially. Figures 9 through 13 show a side by side comparison of data taken in from the 7612D and a photograph of the same waveforms displayed on a Tektronix 7834 Storage Oscilloscope using two 7A19 Amplifiers, a 7B85 Delaying Time Base and a 7B80 Time Base. The photographs are shown in the lower right-hand corner of the figures.  $V_{ce}$  is the bottom trace in the photographs, and  $I_C$  is the upper trace in the photographs (The bottom trace is defined as the trace beginning at the lowest point on the left edge of the photograph.). All other data is labelled.

A comparison of the photographs to their respective plots demonstrates that  $V_{ce}$  and  $I_C$  are accurately digitized and fed to the Zenith. As near as can be told, given the blurred lines of the photographs, the vertical scale factor of the plots are also correct. It is indeed quite likely that the data transported from the 7612D into the Zenith is more accurate than the data shown in the photographs. This is due to the width of the lines when compared to the size of the photograph. The natural blurring that occurs adds a degree of inaccuracy to the data seen in the photograph when compared to the digitized 7612D data.

Figures 9 through 13 directly verify only two of the five possible output wave forms.  $V_{be}$  is indirectly verified as it is obtained in exactly the same fashion that  $V_{ce}$  is, and  $V_{be}$  even comes from the same channel on the 7612D. Another photograph of that waveform is not really necessary.

Verifying the calculated plots, P and E, is not quite as simple. Starting with the safe assumption that  $V_{ce}$  and  $I_C$  are accurate, one can prove by induction that the Instantaneous Power plot is indeed accurate. The inductive argument that if one point on the plot is

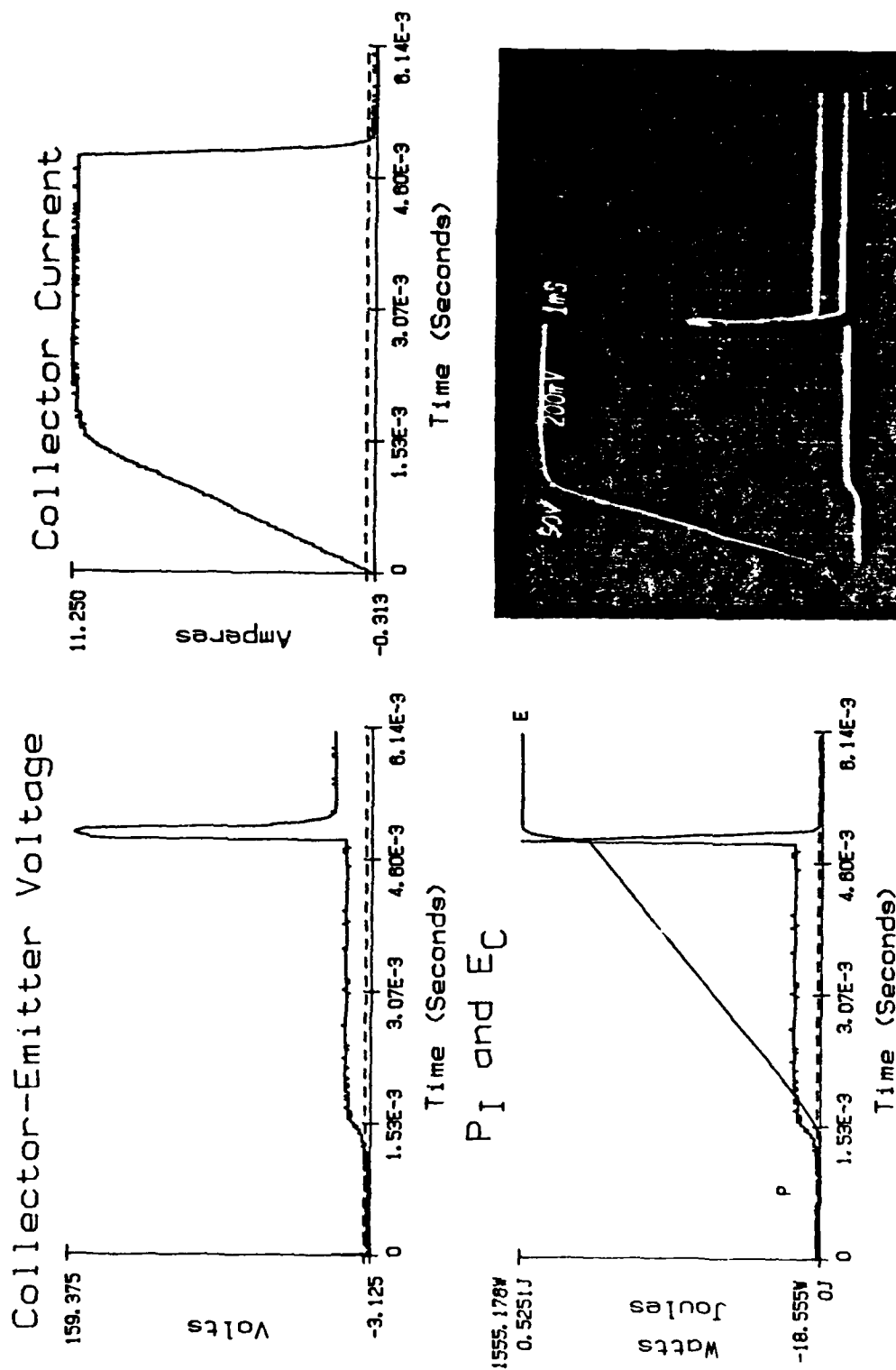
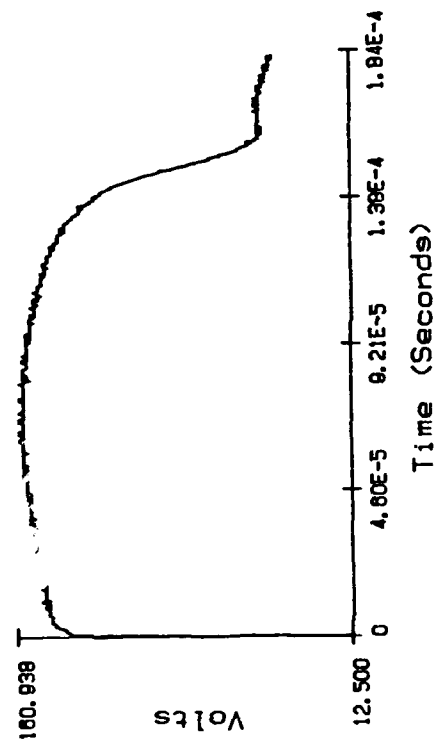
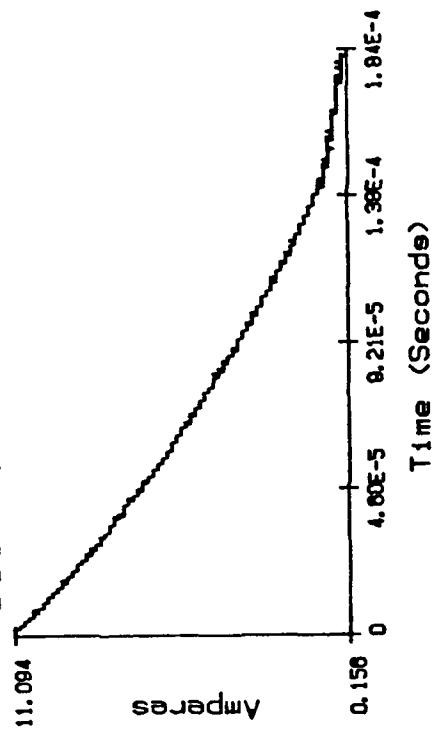


Figure 9. Sample plots along with photograph of Collector-Emitter Voltage and Collector Current. The figure shows the entire test cycle from TWT turn-on to turn-off. In data acquisition, the time of interest is during the peak in the Collector-Emitter Voltage. This is the region where breakdown will occur.

Collector-Emitter Voltage



Collector Current



$P_I$  and  $E_C$

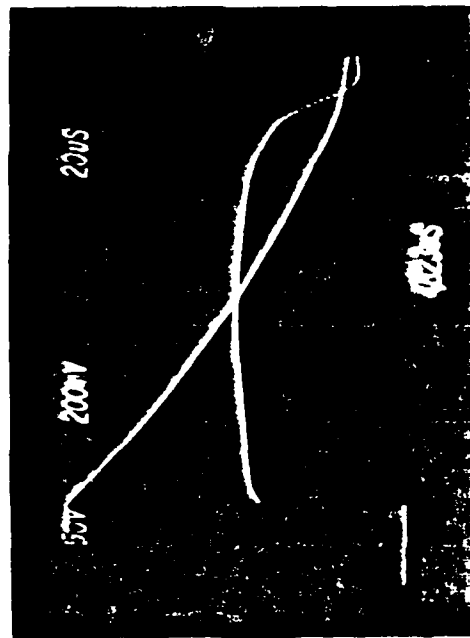
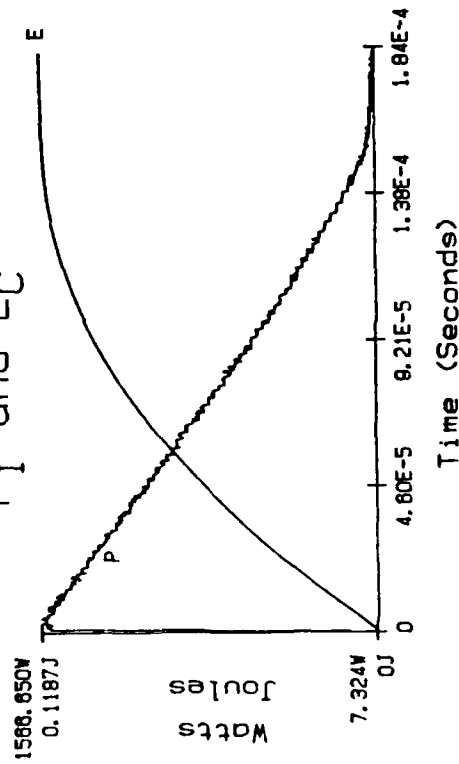


Figure 10. Sample plots showing safe turn-off. This is a blow-up of the peak region in Figure 9. Forward bias setting is 1 A. Reverse bias setting is 0.05 A.

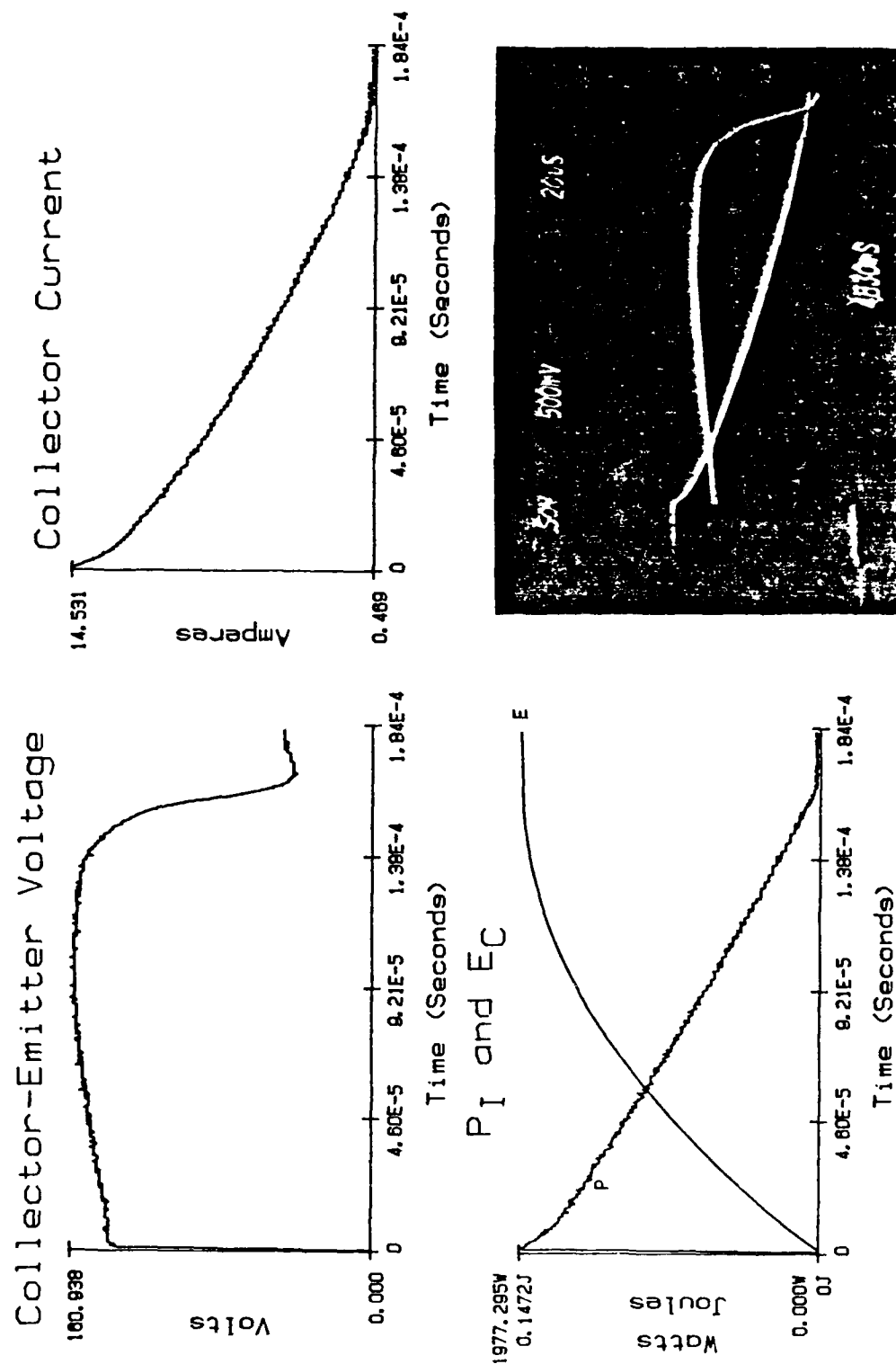


Figure 11. Sample plots again showing safe turn-off. The rate of drop in the Collector-Emitter Voltage is now more precipitous indicating nearing the edge of safe operation. Forward bias setting is 3 A. Reverse bias setting is 0.05 A.

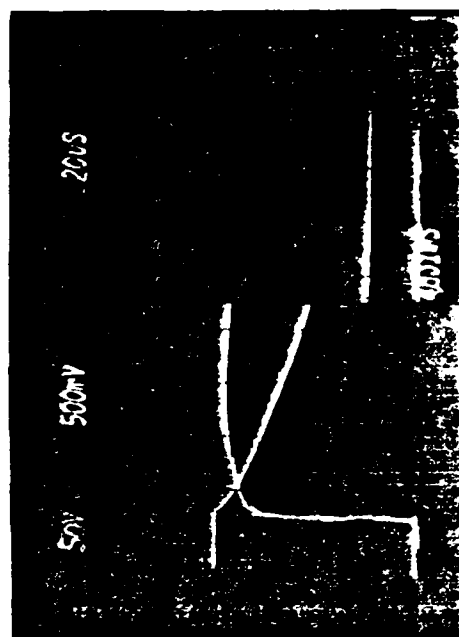
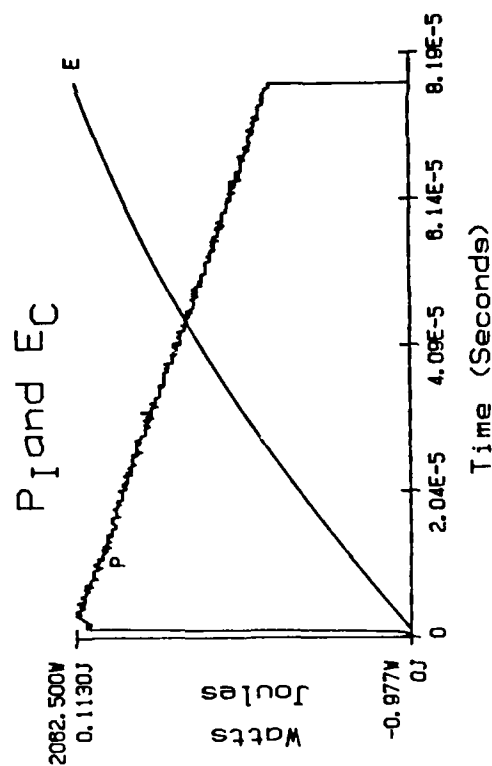
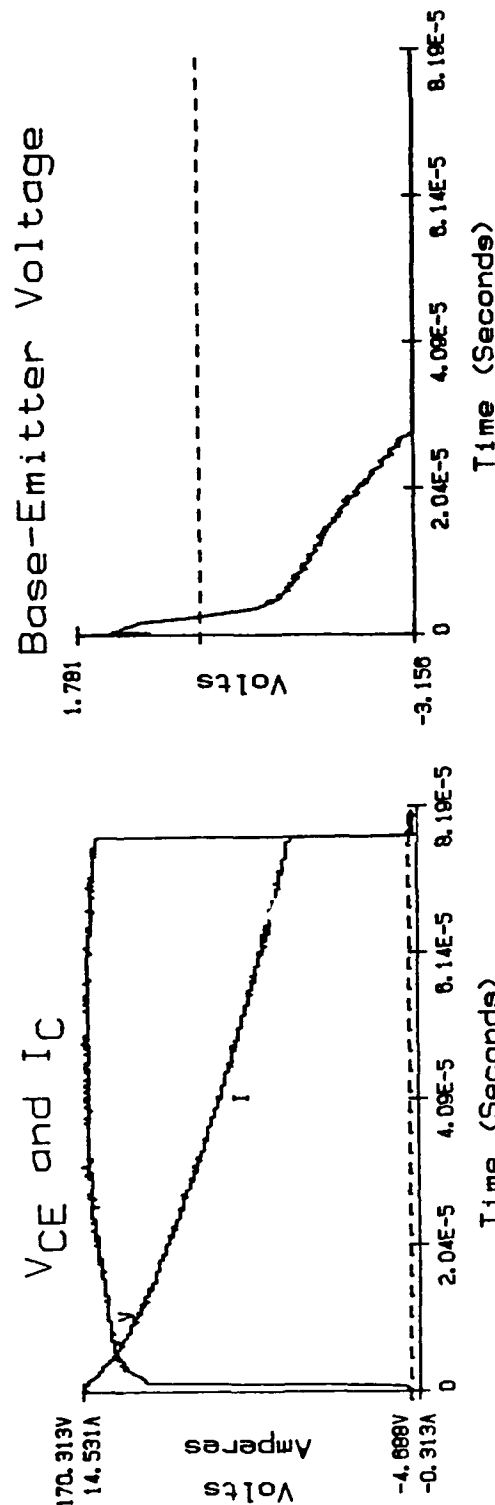


Figure 12. Sample plots showing breakdown. Forward bias setting is 3 A. Reverse bias setting is 0.1 A.

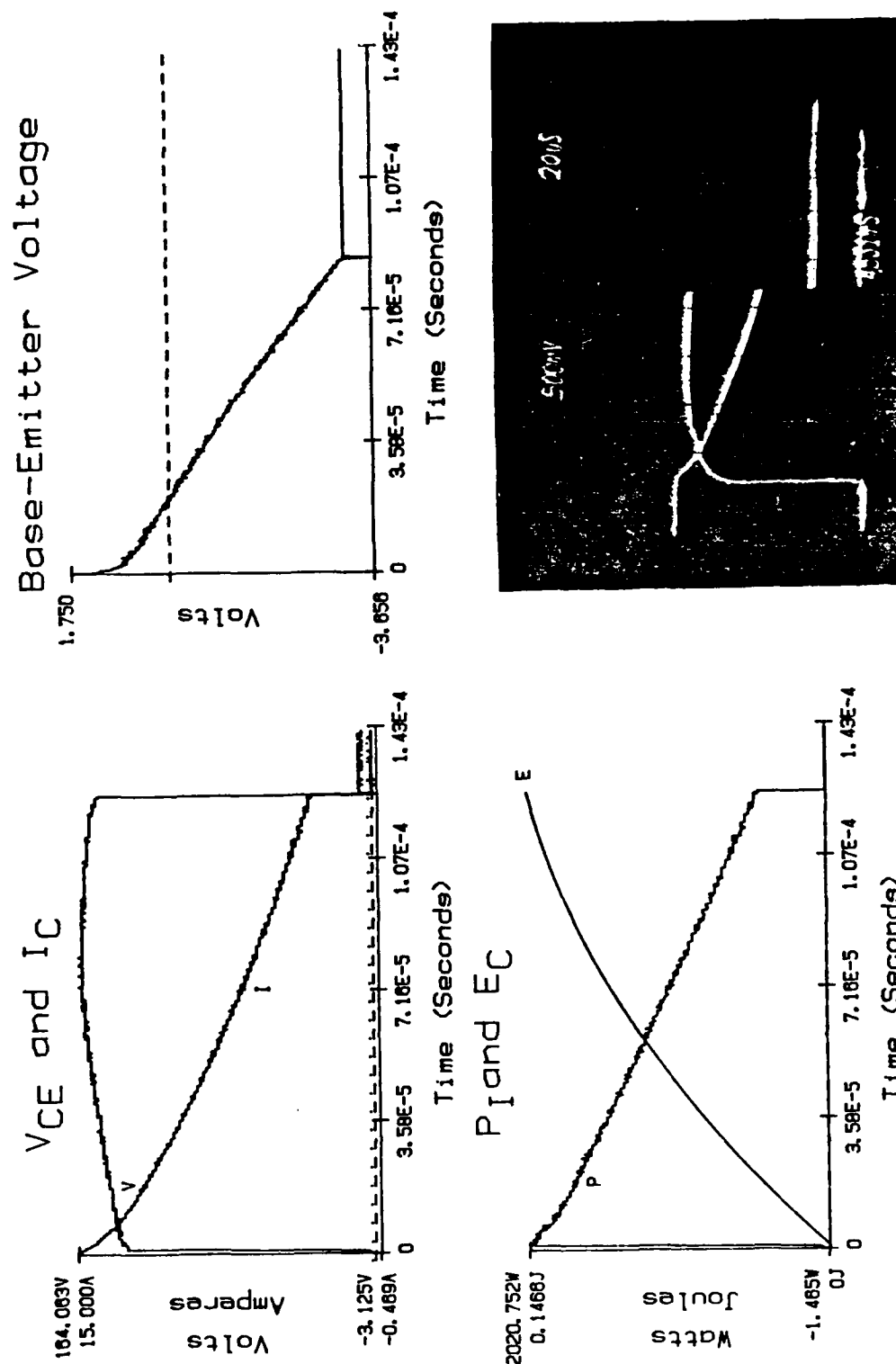


Figure 13. Sample plots showing breakdown. Although breakdown does occur, it takes considerably longer to happen. Forward bias setting is 3 A. Reverse bias setting is 0.06 A.

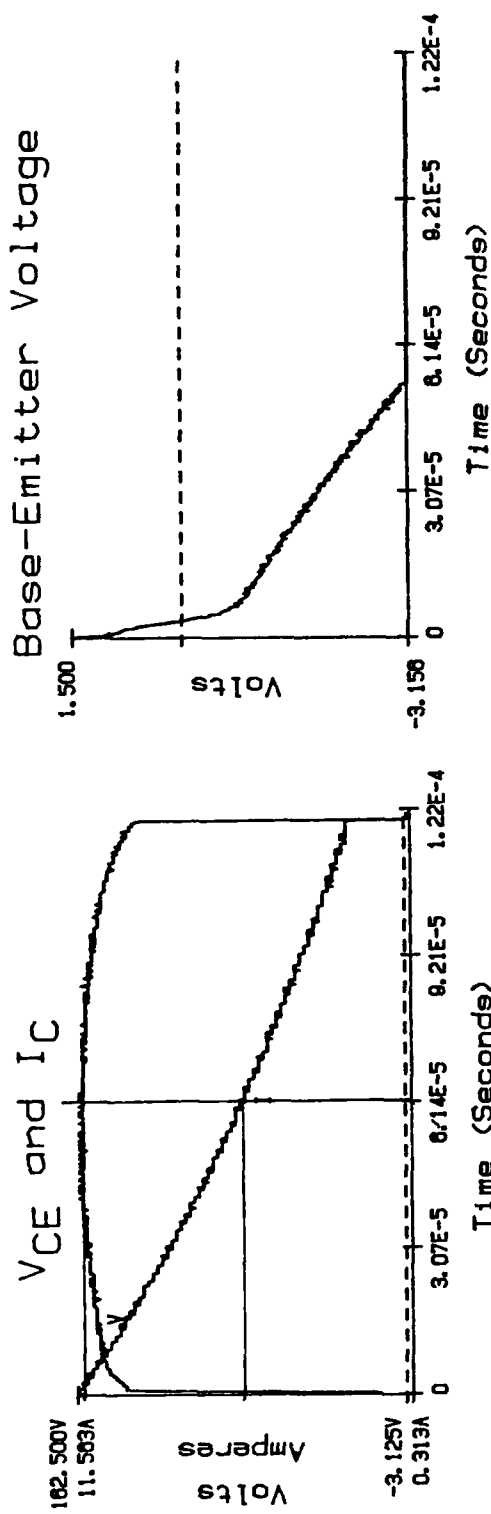
correct then all the points are correct is valid in this instance since all of the points were generated in exactly the same manner; therefore, any error introduced in the calculation would be found in all of the data points used to create the plot.

To verify P, Figures 14 through 17 are used. A perpendicular line is drawn upward from the x-axis at a set time value for  $V_{ce}$ ,  $I_c$  and P on each plot. Next, a horizontal line is drawn to vertical axes, and the magnitude of the described point is calculated. The values for  $V_{ce}$  and  $I_c$  are multiplied together to get the expected value of the power,  $P_e$ .  $P_e$  is then compared to the value determine from the plot for Instantaneous Power,  $P_m$ . The resulting error in all cases was less than 2%. This value is well within an error range which can be attributed to human error. Thus, the plot for Instantaneous Power is verified.

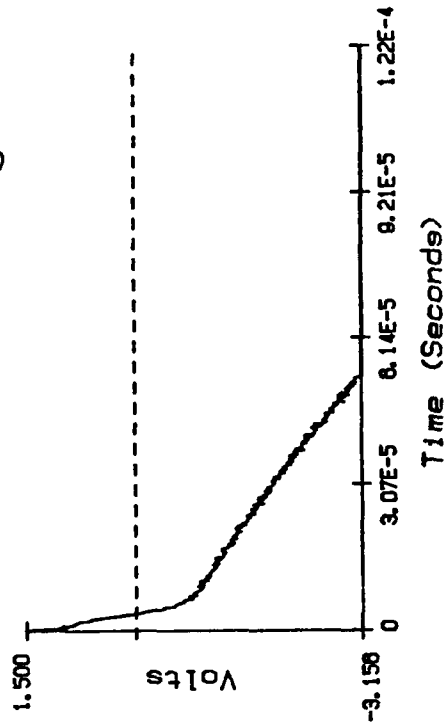
The breakdown values for  $V_{ce}$  and  $I_c$  were verified in a like manner. A horizontal line was drawn from the point where breakdown occurred for  $V_{ce}$  and  $I_c$ . The values so obtained were compared to the values printed out under "Breakdown Values." The average error was less than 2%, and the values are considered verified. The lines described in this procedure are not shown in the figures.

The last item to be verified is the Energy plot. To prove the accuracy of the plot a deductive argument is used. The Energy plot is actually the summation of the individual areas under P for the 2047 time intervals. If the total energy is proven to be accurate then the individual calculation of the area must be accurate, too.

Using the safe assumption that P is accurate, an approximation of P was made by forming a right triangle mounted on a rectangular base. The hypotenuse of the triangle was a line approximating the falling value of P. The height of the triangle was determined by drawing a vertical line approximately parallel to the rising edge of P but slightly offset to take into account the missing notch of the triangle at the top. The length of the vertical line represents the sum of the height of the triangle and the rectangle. The height of the rectangle was determine by measuring the distance from the x-axis to the point of the sudden drop in P. The base is the distance between the long vertical line and the point where the sudden drop in P intersects the x-axis. With all the dimension known, a calculation of the area which represents the total energy was performed. The resulting error between the energy measured as stated above and the total energy value printed under "Breakdown Values" averaged less than 2%. Although the extreme closeness of the results is mainly luck, it does provide a basis from which it can be stated that the Energy plot is accurate.



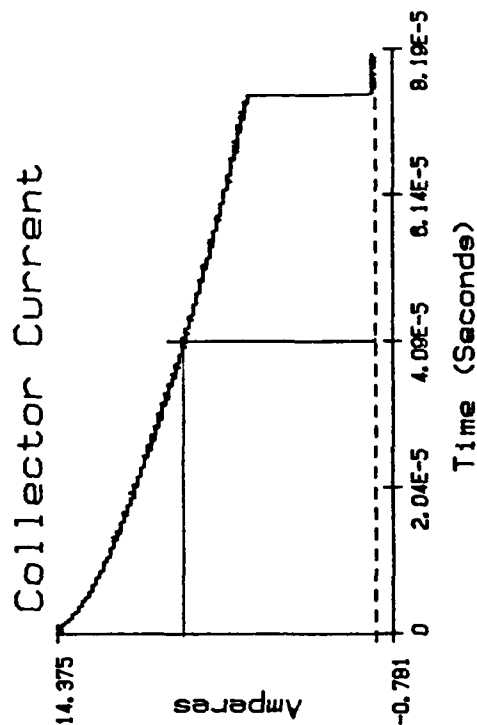
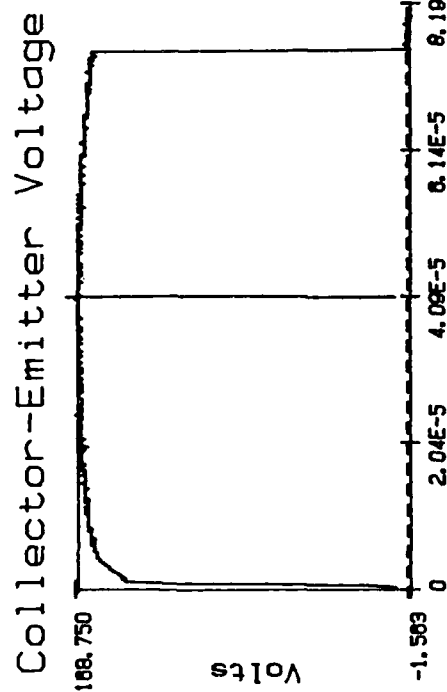
# Base-Emitter Voltage



$$\begin{aligned}
 V_m &= 159.829 \text{ V} & I_m &= 6.027 \text{ A} \\
 P_m &= 950.370 \text{ W} \\
 P_e &= (V_m)(I_m) = (159.829)(6.027) = 963.289 \text{ W} \\
 e_p &= \frac{|P_e - P_m|}{P_m} = 0.014 = 1.4\% \\
 E_m &= b_t (h_R - 0.5 \cdot h_T) \cdot b_t = \text{base length (time)} = 1.18E-4 \text{ sec} \\
 h_R &= \text{height of Rectangle} = 368.842 \text{ W} \\
 h_T &= \text{height of Triangle} = 1294.000 \text{ W} \\
 &= 0.1199 \text{ J} \\
 E_a &= 0.1191 \text{ J} \\
 e_E &= \frac{|E_a - E_m|}{E_m} = 0.0065 = 0.65\%
 \end{aligned}$$

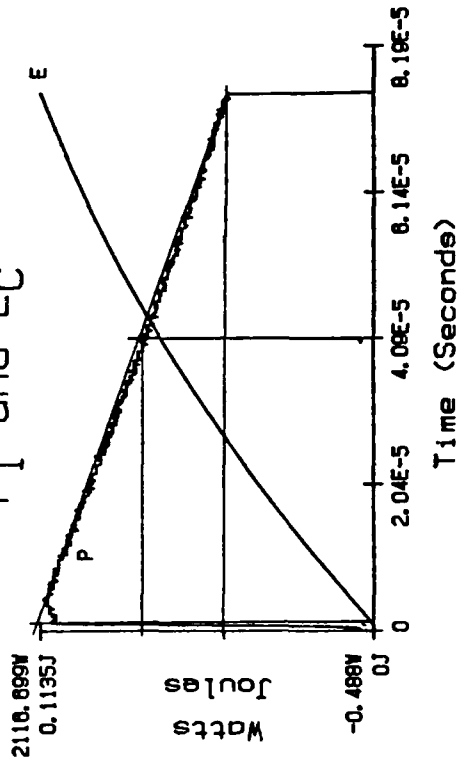
Figure 14. Verification of P and E by graphical means. Forward bias setting is 1 A. Reverse bias setting is 0.06 A.





Time (Seconds)

P<sub>I</sub> and E<sub>C</sub>



$$V_m = 167.378 \text{ V} \quad I_m = 8.722 \text{ A}$$

$$P_m = 1461.867 \text{ W}$$

$$P_e = (V_m)(I_m) = (167.378)(8.722) = 1459.871 \text{ W}$$

$$e_p = \frac{|P_e - P_m|}{P_m} \approx 0.014 = 1.4\%$$

$$E_m = b_t (h_R - 0.5 \cdot h_T) \quad b_t = \text{base length (time)} = 7.415E-5 \text{ sec}$$

$$h_R = \text{height of Rectangle} = 957.287 \text{ W}$$

$$h_T = \text{height of Triangle} = 1175.727 \text{ W}$$

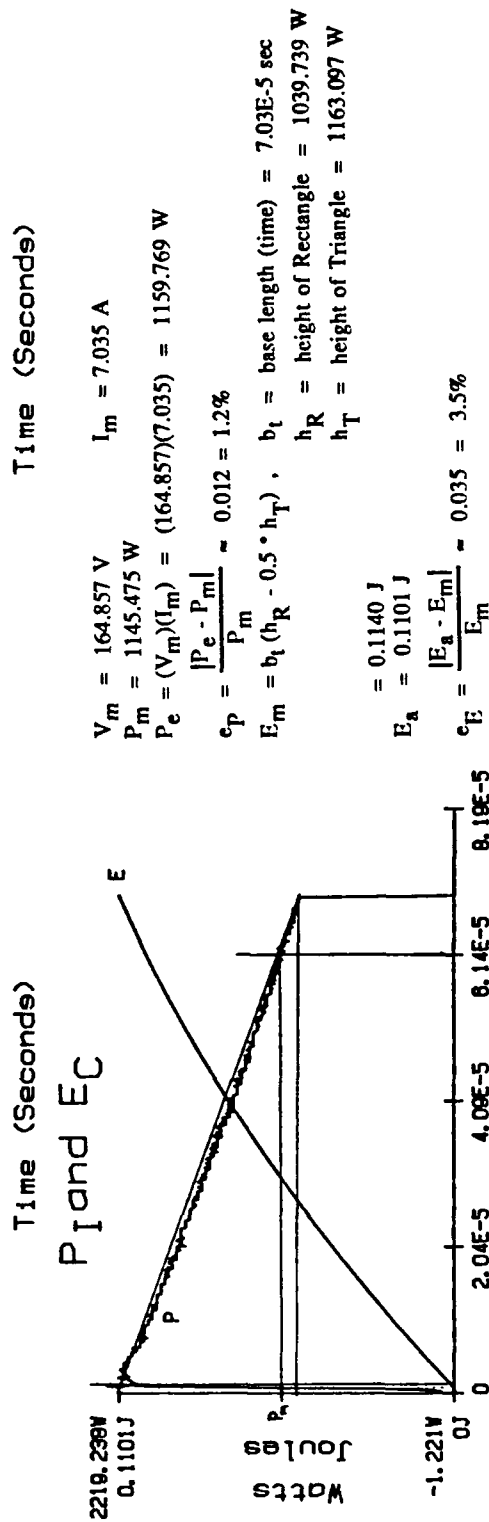
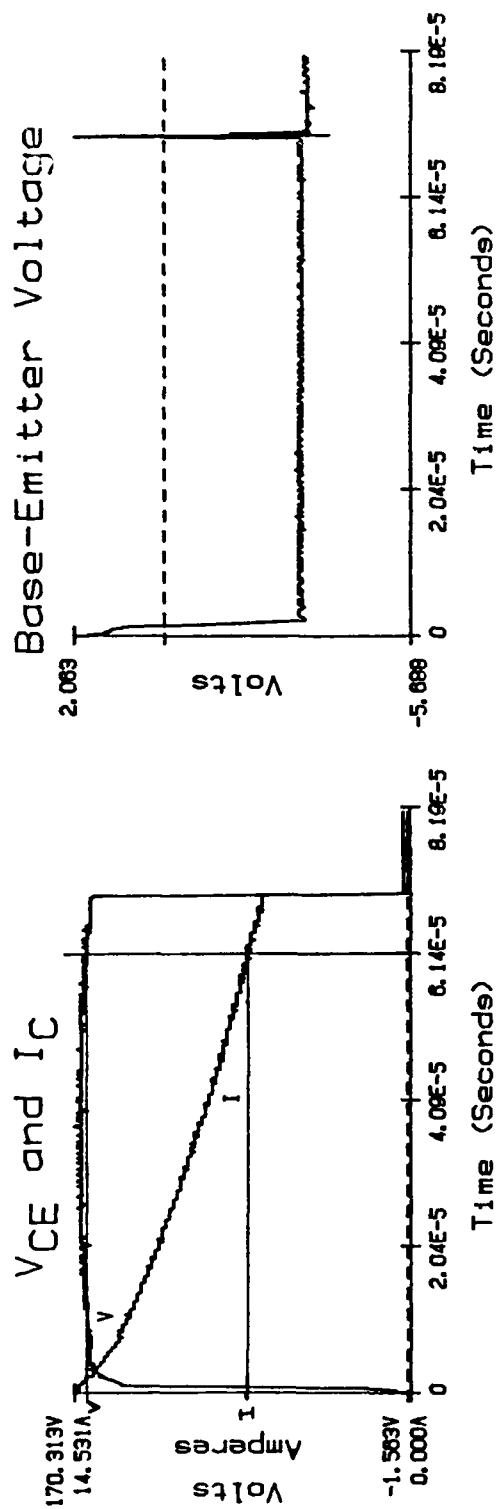
$$= 0.1146 \text{ J}$$

$$E_a = 0.1135 \text{ J}$$

$$e_E = \frac{|E_a - E_m|}{E_m} = 0.0094 = 0.94\%$$

Time (Seconds)

Figure 15. Verification of P and E by graphical means. Forward bias setting is 2 A. Reverse bias setting is 0.1 A.



$$\begin{aligned}
 V_m &= 164.857 \text{ V} & I_m &= 7.035 \text{ A} \\
 P_m &= 1145.475 \text{ W} \\
 P_e &= (V_m)(I_m) = (164.857)(7.035) = 1159.769 \text{ W} \\
 e_p &= \frac{|P_e - P_m|}{P_m} = 0.012 = 1.2\% \\
 E_m &= b_t (h_R - 0.5 \cdot h_T) & b_t &= \text{base length (time)} = 7.035 \text{ E-5 sec} \\
 h_R &= \text{height of Rectangle} = 1039.739 \text{ W} \\
 h_T &= \text{height of Triangle} = 1163.097 \text{ W} \\
 &= 0.1140 \text{ J} \\
 E_a &= 0.1101 \text{ J} \\
 e_E &= \frac{|E_a - E_m|}{E_m} = 0.035 = 3.5\%
 \end{aligned}$$

Figure 16. Verification of P and E by graphical means. Forward bias setting is 2 A. Reverse bias setting is 0.2 A.

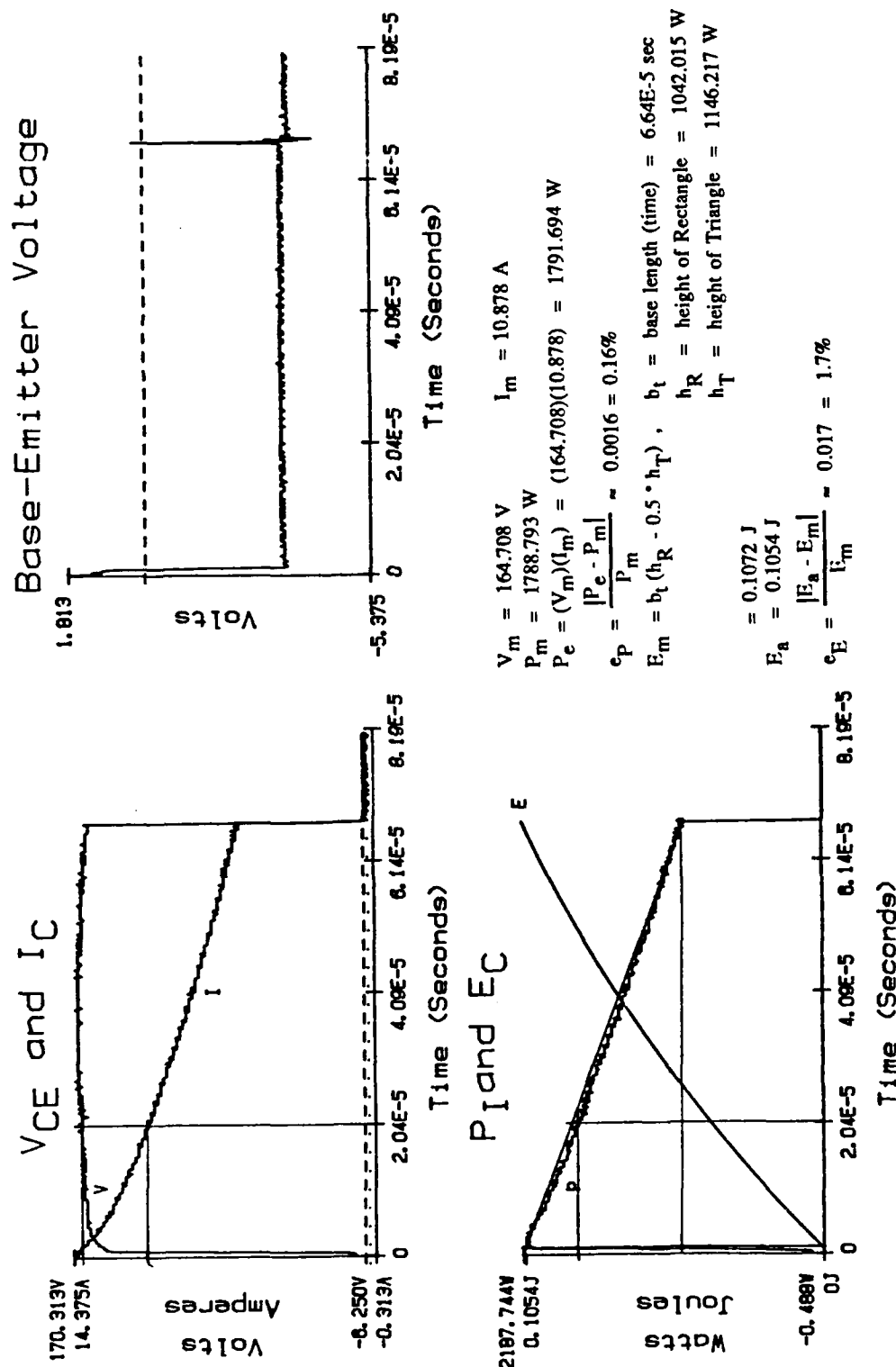


Figure 17. Verification of P and E by graphical means. Forward bias setting is 2 A. Reverse bias setting is 0.4 A.

## CHAPTER V

### CONCLUSION

From the beginning, the focus of this work has been to improve upon the old data collection and processing system used in previous efforts. The major shortcoming of the old data system was the speed with which data could be collected and processed. The addition of the Zenith Microcomputer to the acquisition system has served to correct that deficiency to a fairly great degree. In the old system, which used the HP-85 as the System Controller, the bottleneck in the acquisition process was the speed of the HP-85. The Zenith's superior performance capabilities were used to alleviate the bottleneck, and increase the number of data points that could be read from any particular test to the maximum provided by the 7612D.

Initial efforts were directed at using both the Zenith and the HP-85 in the acquisition system, but that became impossible given the hardware/software limitations of the HPIB interface card in the Zenith. The Zenith then became the sole controller of the system and new software was created to allow the Zenith to perform the necessary tasks.

The new software was originally written for the MS-PASCAL compiler, but complications with the memory allocation of the stack made it impossible to write a single integrated program which could perform all the required tasks. A new approach was undertaken which corrected the difficulties encountered, but it, also, allowed the use of any compiled language for those tasks not requiring the use of the HPIB. As a result of the extended capability of the new approach, large portions of the software system were written for and using a different, easier-to-use compiler, Turbo Pascal.

Once the system was completed, a series of tests were made upon a Texas Instruments 2N5886 transistor, and the results obtained were used as a basis for verifying the software systems accuracy.

On whole, the new data collection and processing system does what it should. It meets all of the requirements it was specifically tasked to meet, it allows for the acquisition of individual and multiple test runs as quickly as the data can be set up on the 7612D and it accurately collects and processes the data given it.

## LIST OF REFERENCES

1. Dale James Skelton, "Reverse Bias Second Breakdown in Power Switching Transistors," M.S. Thesis, Texas Tech University, 1982.
2. Scott Allen McMullen, "Energy Considerations in Second Breakdown," M. S. Thesis, Texas Tech University, 1984.
3. Michael E. Katsaras, "Turnoff Transients in Power Switching Transistors," M. S. Thesis, Texas Tech University, 1986.
4. C. M. Graves, "A Data Acquisition and Conditioning System for Analysis of Transients in Semiconductor," proposed title of M. S. Thesis, Texas Tech University, Unfinished.
5. Tom Swan, Mastering Turbo Pascal. Hasbrouck Heights, New Jersey: Hayden Book Company, 1986.
6. Elliot B. Koffmann, Pascal. Reading Massachusetts: Addison-Wesley Publishing Company, 1982.
7. HP-IB Command Library for MS-DOS. By Hewlett-Packard Company: U.S.A., 1986.
8. Turbo Pascal Reference Manual. By Borland International, Inc.: Scotts Valley, CA, 1986.
9. Microsoft Pascal Compiler User's Guide. By Microsoft Corporation, 1986.
10. Microsoft Pascal Reference Manual. By Microsoft Corporation, 1986.
11. 7612D Programmable Digitizer Operators Instruction Manual. By Tektronix, Inc.: Beaverton, Oregon, 1983.

APPENDIX A  
OPERATING INSTRUCTIONS

Read all of the following instruction before attempting to use the test system.

In the following, instructions regarding Tester Power-Up and Power-Down Sequences for the Reverse Bias Safe Operating Area Tester were developed by Michael Katsaras as part of his work toward his Master's Thesis but was not included in the final version of the thesis. Before proceeding, the reader should familiarize his or herself with references 1 and 3. Also, a thorough understanding of the operation of Tektronix 7612D and 624 Monitor is necessary. Finally, it would be beneficial if the operator understood the basic workings of an IBM PC compatible using DOS.

#### Tester Power-Up Sequence

1. Make sure there is a transistor in the TUT socket. Do not attempt to turn power on without a transistor loaded. (Note: The transistor can be changed while the power is on.)
2. Make sure base current controls are all set to minimum.
3. Make sure collector supply control cable is connected. Make sure the collector supply voltmeter is connected to the power supply (current limited outputs). Make sure the load inductor is connected.
4. Turn power to tester on. Turn collector current monitor biasing supply on (if collector current levels are expected to be higher than 10 A). The "power" indicator should be on at this time. Wait 30 seconds, until the H.V. switch tube (front panel) warms up.
5. Turn on the collector supply. No voltage should be measured by the voltmeter.
6. Turn on the external trigger power supply for the 7612D, located on the back right rear of the 7612D.

The tester is now ready for operation.

#### Setting Up to Take a Measurement

1. Connect the  $V_{ce}$  probe to the A input on Channel A of the 7612D.
2. Connect  $V_{be}$  to input B, Channel A.
3. Connect the  $I_c$  monitor to input A, Channel B.
4. The  $I_b$  monitor can be connected to input B, Channel B, but it is not really necessary.

5. Connect the Scope Trigger to the external trigger input on the 7612D. (The little blue box attached to the front of the 7612D.)
6. Set the 7612D to external trigger, negative going slope (to see turn-off/breakdown region) for both channel A and B.
7. Select the desired time step for both channel A and B. It is imperative that the settings for both channels be the same. If they are not, the data acquisition programs will not run.
8. Select the appropriate forward and reverse bias settings.
9. Press the manual start. According to the instruction written by Michael Katsaras, "Three things should happen: a) The 'Clamp Supply Indicator' should light, b) the clamp supply voltmeter should indicate the H.V. clamp setting, and c) the collector supply voltmeter should indicate the collector supply voltage. To set the collector supply, choose controls [forward and reverse bias settings] which do not cause breakdown and adjust the voltage during the five second delay after the 'manual start' button is pressed."

Assuming all of the above has been accomplished and no problems were encountered, the operator is now ready to begin taking data.

#### Taking a Measurement

1. Turn the Zenith computer on. The mainframe power switch is located on the back of the chassis on the right side. The monitor power switch is located on the front of the monitor.
2. Although the data acquisition software will prompt the operator to check to be sure there is a formatted disk in drive A:, it is easier to do so now, before entering the acquisition process. Further, it is recommended that the operator have on hand at least one additional blank formatted disk. To format a disk, place an unformatted (new) disk in drive A:, type `FORMAT A:` and press `<RETURN>`.
3. The Zenith must be configured with at least a 32 kilobyte RAM disk. The drive designation must be drive G:. To check the availability of drive G:, type `G:` and press `<RETURN>`. If the prompt symbol changes from `C:\>` to `G:\>`, then drive G: is active. Type `C:` and press `<RETURN>`. If the message "Invalid Drive Specification" appears when "G:" is typed, then the following line must be added to the `CONFIG.SYS` file in the root directory of drive C:, `"DEVICE = RAMDISK.SYS 32"`. This line will set up a RAM disk at the first available drive address. On a Zenith, this is drive G:. The file `RAMDISK.SYS` must also be present in the root directory on drive C:.



4. At the C:\> prompt or any C:\xxxxxx> prompt, type TAKEDATA and press <RETURN>. The data acquisition process now begins.

5. The first thing displayed is a reminder to place a formatted disk in drive A:. If this has not already been done, do so now.

6. Next, the operator is queried for the system time-out error value or the system time-out. This is the length of time in seconds that the Zenith will wait for a response from any of the devices on the bus before declaring an error. Typical values used in the development of this software have been three to four seconds.

7. The calibration cycle is then entered. Follow the instructions on the screen exactly. Remember, this is a loop. It may be necessary to go through the cycle several times before calibration is complete. When instructed to adjust the position knob on the Channel A or B plug-ins, a value of less than five is a very small adjustment. It is, also, easier to calibrate the 7612D if the Volts/Div is set on 5 Volts/Div on both plug-ins.

8. When calibration is completed, the program will then display a listing of all files stored on drive A:. It will also display the amount of storage space available. If there is less than 50 kilobytes, replace the disk with a blank formatted disk.

9. From this point on, follow the instructions that are printed on the screen. When an affirmative or negative answer is requested, enter a single "Y" or "N". The letters need not be capitalized. If a mistake is made, no damage will result as all query responses are "idiot" proofed. Numeric entries, on the other hand, are not "idiot" proofed. So, be sure the number is correct before hitting the <RETURN> key. The values can be edited prior to hitting the <RETURN> key by backspacing to the point where the error was made and re-typing. Textual entries, sentences and words, are also not "idiot" proofed, but they are typically not critical to program execution, and if an error is made, it can be corrected latter.

It is recommended that the operator become familiar with the operation of the entire system by taking data on one of the older, no longer needed, test devices.

#### Creating Output

The operator should be familiar with the use of the HP-7470A Graphics Plotter before proceeding. The GPIB peripheral drive should be loaded with the GPIB set to be LPT2. This function is performed in the CONFIG.SYS file with the line, "DEVICE = GPIB.SYS LPT2=705". If that line is not present and alternative method is to type GPIBMODE LPT2=705 at the C:\> prompt.

### Obtaining a Plot

1. Place the floppy diskette containing the device data to be plotted in drive A:. All of the BDxxxLLq.yyy files for the particular device and series to be printed must be present on the same disk for the data to be plotted. If the wrong disk has been inserted and the data is not present, the software will tell the operator, and allow him or her to replace the disk with the proper one.
2. Place a plotter pen in the right hand stall of the HP-7470A and turn the plotter on.
3. Load a sheet of plotter paper into the plotter.
4. Type PLOTDATA while in the C:\RBSOA directory, press <RETURN> and answer the questions that are asked. A plot will then be generated, but be patient. The process takes five to seven minutes.

### Creating a Table of Breakdown Values

1. Place the disk containing the data from the first test run in drive A:.
2. Type TABULATE while in the C:\RBSOA directory and press <RETURN>.
3. Answer the questions asked and a file will be created on the C:\RBSOA directory with the name BDxxxLL.TAB. This is the file containing the tabulated breakdown data. If the operator desires, the file can be copied to floppy diskette and then deleted from the C:\RBSOA directory to save room on the hard disk, but this is not necessary.

The operator has created a TAB file, but this cannot be directly printed out.

### Obtaining a Sorted Output of Breakdown Values

1. Type SORTABLE while in the C:\RBSOA directory and press <RETURN>.
2. Answer the questions asked. If output to the line printer is desired, tell the program to output to "LPT1", and be sure the line printer is on and ready to receive data. Otherwise, tell the program the name and destination for an output text file.
3. There is no limit to the number of time SORTABLE.COM can be run on a given TAB file. So, different sort keys can be used without trouble; however, be sure to use different output names, except in the case of the line printer, or the previous file will be overwritten by the new output file.

### Using EDITLIST.COM

This file is used to edit and maintain the device listing contained in the file INDEX.DEV.

1. Type EDITLIST while in the C:\RBSOA directory and press <RETURN>.
2. Answer the questions asked.
3. Make the changes or additions desired.

### Using PRCSLTWO.COM

This file is used to calculate or re-calculate the values for P and E. Its use can speed the execution of TAKEDATA as it can allow the operator to remove the call to PROCESS2. This is not recommended. The principle use for this program is to re-calculate P and E if it is feared the files have been corrupted. PRCSLTWO.COM can also be used in the case of a program crash after  $V_{ce}$  and  $I_c$  have been read but before PROCESS2 could run properly.

1. Be sure the disk containing the files BDxxxLLA.yyy and BDxxxLLB.yyy for the desired device and test series is in drive A: and that there is sufficient disk space 40 kilobytes, at least, for the storage of BDxxxLLE.yyy and BDxxxLLP.yyy.
2. Type PRCSLTWO while in the C:\RBSOA directory and press <RETURN>.
3. Answer the questions asked.

### Powering Down the Tester

1. Make sure all bias settings are set to minimum.
2. Turn off the collector supply first, then wait at least five seconds and turn off the tester.
3. Turn off the collector current monitoring supply and the external trigger power supply.
4. Turn everything else off.

APPENDIX B  
TAKEDATA.BAT PROGRAM LISTING

# Batch file program: TAKEDATA.BAT

ECHO off

- Stops DOS from echoing the following lines to the screen, except for those commands with "ECHO" in front of them. These are remarks to be printed to the screen.

CD\RBSOA

- Changes active directory to C:\RBSOA.

ECHO Before proceeding be sure you have a formatted disk in drive A:.

ECHO If you are not sure if the disk is formatted or not, type ^C to

ECHO terminate this program; otherwise, press any other key to continue.

PAUSE

- Execution halts until operator presses a key.

HARDWARE

- Calls the program HARDWARE.EXE, and executes it.

:loop

- Top of the acquisition loop.

ECHO Here is the directory on Drive A:.

DIR A:/W

- Prints the directory on drive A: to the screen.

ECHO If there is less than 50 kBytes free on this disk, replace it with a blank,

ECHO formatted disk before continuing.

PAUSE

IF EXIST G:CONTINUE.ANS DEL G:CONTINUE.ANS

- This line deletes the file CONTINUE.ANS if it exists. The file is used to control loop execution.

CLS

- Clears the screen.

IF NOT EXIST G:HALTER.PGM GETDEVIC

---

\* The "\*" mark denotes the beginning of a comment not found in the software used in running the system. These comments are placed purely as an aid to understanding how the programs work and why they were written the way they were.

- If the file HALTER.PGM does not exist, then execute the program GET-DEVIC.COM. HALTER.PGM is created when a program in the acquisition loop requires information and the operator refuses to provide it. The operator has at least two opportunities to do this and is warned before the file is created. In the acquisition loop, prior to program execution, a check is made for the existence of this file.

CLS

ECHO Acquire the Collector-Emitter Voltage on Channel A and the Collector ECHO Current on Channel B. These waveforms will be read in so be sure they ECHO are what you want before proceeding.

PAUSE

- Pauses execution while the operator obtains the desired waveforms on the 7612D.

CLS

IF NOT EXIST G:HALTER.PGM STORDATA

- Calls and executes STORDATA.COM.

CLS

IF NOT EXIST G:HALTER.PGM PROCESS1

- Calls and executes PROCESS1.EXE.

CLS

ECHO Calculating Instantaneous Power and Energy, please wait . . .

IF NOT EXIST G:HALTER.PGM PROCESS2

- Calls and executes PROCESS2.COM. If  $V_{be}$  is to be taken, there is a pause built into the end of PROCESS2.COM.

CLS

IF NOT EXIST G:HALTER.PGM PROCESS3

- Calls and executes PROCESS3.EXE.

CLS

IF EXIST G:BD???L??.\* COPY G:BD???L??.\* A:

- If the file BDxxxLLq.yyy exist, it is copied to drive A:, and then the next line deletes those files just copied. This save storage space on the RAM disk.

IF EXIST G:BD???L??.\* DEL G:BD???L??.\*

IF NOT EXIST G:HALTER.PGM REPEATER

- Calls and executes REPEATER.COM.

CLS

IF EXIST G:CONTINUE.ANS GOTO loop

- If the file CONTINUE.ANS was created in REPEATER.COM, then the batch file jumps back to the top of the loop and execution begins again.

CLS

COPY G:DEV?????.BK A:

- Copies the bookkeeping file to the floppy diskette in drive A:

ECHO                      Session Finished.

IF EXIST G:HALTER.PGM TYPE G:HALTER.PGM

- Prints the contents of HALTER.PGM if it exists. This tells the operator the reason for the halting of the program.

IF EXIST G:HALTER.PGM DEL G:HALTER.PGM

APPENDIX C  
HARDWARE.PAS PROGRAM LISTING



PROGRAM Data\_Acquisition (INPUT,OUTPUT);

- This program is used to first, pole the bus for the 7612D, and then, calibrate it. Written for the MS-PASCAL Compiler. The reader should be thoroughly familiar with the HPIB Command Library Reference Manual, as well as the manuals for MS-PASCAL (Reference and User's) and the 7612D Programmable Digitizer Operator's Instruction Manual. This statement goes for all of the programs written for the MS-PASCAL compiler (HARDWARE, PROCESS1 and PROCESS3).

```
{ ***** }
{$INCLUDE: 'IODECL.EX'}
```

- Declares the Command Library global variables.

```
{ ***** }
```

CONST

bus = 7;

TYPE

STR3 = STRING(3);

STR9 = STRING(9);

VAR

I : INTEGER;

endline : STRING(2);

command : STR3;

passinfo : STRING(10);

pass : STRING(12);

passer : TEXT;

timeout : REAL;

- These are the global variables used in this program.

```
{ ***** }
```

```
{ $INCLUDE: 'IOPROC.EX' }
```

- Declares the function names used in the Command Library. This way no declaration of external functions is necessary to use the HPIB interface card.

```
{ **** }
```

```
PROCEDURE error_handler (error : INTEGER; routine : STR9; A : CHAR);
```

- This procedure is used to check the error condition of the interface bus. It returns a statement as to which function the error occurred in and what the error was.

```
VAR
```

```
  estring : STRING(40);
```

```
  cmd : INTEGER;
```

```
BEGIN
```

```
  IF error <> noerr THEN
```

- The value "noerr" is a predeclared constant from IODECL.EX. Its value is zero.

```
    BEGIN
```

```
      Errstr(cmd,estring);
```

- The procedure Errstr() returns the statement describing the error.

```
      WRITELN('Error in call to ',routine);
```

```
      WRITELN(error:6,' ',estring);
```

```
      IF A = 'N' THEN
```

- The character "A" is used to cause the procedure to halt and allow the operator to correct the problem before going on. It tells the operator whether the error is one that needs to be corrected or not.

```
        BEGIN
```

```
          WRITE('Press <RETURN> to continue . . . ');
```

```
          READLN
```

```
        END;
```

```
      IF A = 'Y' THEN
```

```
        BEGIN
```

```

        WRITELN('CORRECT ERROR - Press <RETURN> to continue.');
```

    READLN

    END

END

END;

```
{*****}
```

```
FUNCTION DOSXQQ(command,parameter : WORD) : BYTE; EXTERN;
```

- This is an external function declaration, meaning the function is contained in a library. DOSXQQ happens to be in one of the standard libraries for MS-PASCAL which is automatically searched during LINK, the step used to create the executable file. Otherwise, the library would need to be specified during LINK. The function allows the program to place calls directly to DOS.

```
{*****}
```

```
PROCEDURE BLOCK;
```

- This procedure does nothing but perform ten WRITELN statements. The procedure is used to make the screen display more readable by breaking up blocks of text.

```

VAR
  I : INTEGER;

BEGIN
  FOR I := 1 TO 10 DO
    WRITELN
  END; {END PROCEDURE BLOCK}
```

```
{*****}
```

```
PROCEDURE hardwarecheck(VAR tout : REAL);
```

{Checks bus looking for active devices.}  
 {MOD 1 - checks 7612D}

VAR

R : REAL;

sent13 : STRING(13);

sent1 : STRING(12);

sent2 : STRING(2);

sent3 : STR3;

sent5 : STRING(5);

sent7 : STRING(7);

rec : STRING(20);

- These string values are used to hold commands to be sent the 7612D and receive a response back.

I,cmd : INTEGER;

- The integer value "cmd" is always used to receive the error code number from the IO functions in the Command Library. This value is passed to the procedure error\_handler for error checking.

test : BOOLEAN;

ans : char;

BEGIN {HC 1}

cmd := IOEOI(7,1); {\*\*\*\*\* These four lines set \*\*\*\*\*}

endline[1] := CHR(13); {\*\*\*\*\* EOI enable (last byte of \*\*\*\*\*}

endline[2] := CHR(10); {transmission sent with EOI active) and EOL char-}

cmd := IOEOL(7,endline,1); {as <CR> <LF> (carriage return, line feed). }

cmd := IORESET(bus);

- Resets bus to known state.

error\_handler(cmd,'IORESET ',N');

- Performs an error check on the function IORESET.

cmd := IOCLEAR(7); {clears bus to known levels}

WRITE('ENTER SYSTEM TIMEOUT (in seconds) . . . ');

- First operator response requested.

READLN(tout); {tout = time-out in seconds}

- The value "tout" is the time in seconds the system will wait for a response from any device before declaring an error.

```
cmd := IOTIMEOUT(7,tout); {defines time-out as tout}
```

```
error_handler(cmd,'IOTIMEOUT','N');
```

```
cmd := IOREMOTE(bus);
```

- This function is used to tell the 7612D to listen to the bus for command input and to disable the front panel.

```
error_handler(cmd,'IOREMOTE','N');
```

```
sent3 := 'ID?';
```

- The letters "ID?" are placed in the 3 character string "sent3." "ID?" is a query used to tell the 7612D to respond with its name.

```
test := FALSE;
```

```
REPEAT
```

- Top of a loop used to allow the operator to correct any problems and re-test the bus/7612D interface.

```
cmd := IOOUTPUTS(70200,sent3,3);
```

- The string "sent3" is actually sent to the 7612D here. This is done to check the 7612D to be sure the power is on and the bus is functioning properly.

```
IF cmd = 0 THEN
```

```
  BEGIN
```

```
    WRITELN('The 7612D is ready for use. ');
```

- If there is no IO error, this message is printed.

```
    test := TRUE;
```

```
  END;
```

```
  error_handler(cmd,'IOOUTPUTS','N')
```

```
UNTIL test;
```

- End of loop. If there has been no error or the error has been corrected, then the loop will stop execution.

```
cmd := IOLOCAL(7);
```

- Returns control of the 7612D to the front panel of that device rather than the bus.

```
END; {HC 1}
```

```
{*****}
```

# PROCEDURE Calibrate;

- This procedure is used to check the calibration of the 7612D. Theoretically, the zero point of the curves can be set by direct software control. This was not possible when following the information given in the 7612D operator's manual. So, this procedure for manual calibration was developed.

## VAR

```
totA,totB : REAL;
test : BOOLEAN;
line : STRING(128);
state,I,R,C,cmd : INTEGER;
response : STRING(40);
incoming : CHAR;
readit : STRING(6);
cal : TEXT;
```

## BEGIN

```
ASSIGN(cal,'G:CALIBRAT.DAT');
```

- Makes "cal" the file designation for the file named "G:CALIBRAT.DAT".

```
test := TRUE;
```

```
WHILE test DO BEGIN
```

- This is the beginning of the calibration loop. It will continue to execute until the condition "test" is false.

```
REWRITE(cal);
```

```
WRITELN('Set input coupling to GND on both Channel A & B. Then trigger');
```

```
WRITELN('the scope so there is a double flat line trace.');
```

```
WRITE('Press <RETURN> when ready to continue . . . ');
```

```
READLN;
```

```
BLOCK;
```

```
readit := 'READ A';
```

- This is the command to be sent to the 7612D. It tells the digitizer to read out the data stored on Channel A.

```
cmd := IOOUTPUTS(70200,readit,6);
```

- The command is sent.

```
FOR R := 1 TO 3 DO BEGIN
```

- This loop reads in the first three characters put out by the 7612D. They are control characters unnecessary for the function at hand and are discarded.

```
I := 1;
```

```
cmd := IOENTERS(70200,incoming,I)
```

```
END;
```

```
state := 0;
```

```
totA := 0;
```

```
cmd := IOMATCH(7,CHR(10),state);
```

- This function turns off the end of line check. The Zenith will ignore the normal characters indicating end of line. Otherwise, the lines about to be read in could terminate prior to their real end when data is read that has the same binary value as the end of line characters. In order for the data to be correctly read in all of the lines must contain a total of 128 characters each.

```
FOR R := 0 TO 15 DO BEGIN
```

- The three steps are taken 16 times.

```
I := 128;
```

```
cmd := IOENTERS(70200,line,I);
```

- A line of length 128 is entered into the string variable "line" from the bus. This is the data points from the 7612D.

```
WRITE(cal,line)
```

- The line value is then written to the file referred to as "cal."

```
END;
```

- The result of the above 16 lines of 128 characters being transferred is a text file of 2048 ASCII characters on the RAM disk. The characters represent the binary values of the data read from the 7612D.

```
state := 1;
```

```
cmd := IOMATCH(7,CHR(10),state);
```

- IO matching is turned back on.

I := 40;

cmd := IOENTERS(70200,response,I);

- The trailing values for Channel A, control and error checking values, are read.

readit := 'READ B';

cmd := IOOUTPUTS(70200,readit,6);

- The entire cycle just conducted with Channel A is repeated for Channel B.

FOR R := 1 TO 3 DO BEGIN

  I := 1;

  cmd := IOENTERS(70200,incoming,I)

END;

state := 0;

totB := 0;

cmd := IOMATCH(7,CHR(10),state);

FOR R := 0 TO 15 DO BEGIN

  I := 128;

  cmd := IOENTERS(70200,line,I);

  WRITE(cal,line)

END;

CLOSE(cal);

- The file referred to as "cal" now contains 4096 characters representing the data from Channel A and B.

WRITELN('Please wait. I am thinking.');

- This line tells the operator the program is working, the data has been read and processing is beginning.

RESET(cal);

- The file referred to as "cal" is re-opened, the cursor position is moved to the top of the file and the file is ready to be read.

state := 1;

cmd := IOMATCH(7,CHR(10),state);

I := 40;

cmd := IOENTERS(70200,response,I);



- Some final control characters are read from the 7612D returning it to normal functioning.

```
FOR R := 1 TO 2048 DO BEGIN
```

```
  READ(cal,incoming);
```

- The first 2048 characters in "cal" are read one at a time. (Channel A data)

```
  C := ORD(incoming);
```

- The characters are converted to there equivalent integer value.

```
  totA := totA + C
```

- The integer values are summed.

```
END;
```

```
  WRITELN('I'm thinking . . . . Believe me I'm thinking.');
```

- This line is written to tell the operator that the processing of the data from Channel A is finished and processing of data from Channel B is beginning.

```
FOR R := 1 TO 2048 DO BEGIN
```

- The steps done to sum the data from Channel A are repeated again, but for the data from Channel B.

```
  READ(cal,incoming);
```

```
  C := ORD(incoming);
```

```
  totB := totB + C
```

```
END;
```

```
CLOSE(cal);
```

```
  WRITELN('It just didn't look like it.');
```

- This line tells the operator that the processing is complete.

```
  WRITELN;
```

```
cmd := IOLOCAL(7);
```

```
totA := totA/2048;
```

```
totB := totB/2048;
```

- The average value of the data read from the 7612D is calculated.

```
IF ((totA < 126) OR (totA > 128)) THEN BEGIN
```

- If the value is  $\pm 1$  of 127, which is the equivalent of the zero point, then calibration is complete on Channel A. Otherwise, the following is printed and the calibration loop is executed.

```
  WRITELN('Channel A needs to be calibrated. ');
```

```

totA := -(totA - 127);
WRITELN('Adjustment amount . . . ',totA:4:2);
WRITELN('Positive, turn knob clockwise; Negative, turn knob counter-clockwise')
END;
IF ((totB < 126) OR (totB > 128)) THEN BEGIN
    • If the value is  $\pm 1$  of 127, which is the equivalent of the zero point, then
      calibration is complete on Channel B. Otherwise, the following is
      printed and the calibration loop is executed.
    WRITELN('Channel B needs to be calibrated. ');
    totB := -(totB - 127);
    WRITELN('Adjustment amount . . . ',totB:4:2);
    WRITELN('Positive, turn knob clockwise; Negative, turn knob counter-clockwise')
    END;
    IF ((totA > 126) AND (totA < 128) AND (totB > 126) AND (totB < 128)) THEN
        • If neither Channel A or B needs to be calibrated, then the file "cal" is
          erased from the RAM disk and the loop is terminated.
    BEGIN
        DISCARD(cal);
        WRITELN('Calibration completed. ');
        test := FALSE
    END
    ELSE BEGIN
        WRITE('Press <RETURN> after adjustment is made. ');
        READLN;
        BLOCK
    END
    END
END;

{ ***** }

BEGIN {Main Program}
    • The procedure described above are called and executed here in the Main
      Program.

```

```
hardwarecheck(timeout);
```

```
Calibrate;
```

- The following line create the file, "PASSER.DAT," which is used to pass global variables between the programs in the acquisition cycle.

```
pass := 'G:PASSER.DAT';
```

```
ASSIGN(passer,pass);
```

```
passinfo[1] := 'Y';
```

- The first character in the first line of PASSER.DAT is set to "Y." This character is used to control the execution of GETDEVIC. HARDWARE executes only once at the beginning of the acquisition cycle, GETDEVIC must execute after HARDWARE because the file PASSER.DAT does not contain complete information, and GETDEVIC must also execute whenever a change to the last two lines of PASSER.DAT is needed. So, the first character of line one is set at the creation of PASSER.DAT to "Y."

```
passinfo[2] := 'N';
```

```
passinfo[3] := 'N';
```

```
passinfo[4] := 'N';
```

```
passinfo[5] := 'Y';
```

- The next four characters in the first line of PASSER.DAT are set. The actual value is not all that important. These are set as seen mainly from a requirement of later programs that no longer exist.

```
FOR I := 6 TO 10 DO
```

```
  passinfo[I] := 'Q';
```

- The last five characters in the first line are set. There is no requirement that they be set to any particular value.

```
REWRITE(passer);
```

```
WRITELN(passer,passinfo);
```

- PASSER.DAT is opened and the first line, referred to as "passinfo," is written on the RAM disk.

```
WRITELN(passer,' 0');
```

- The second line is written in PASSER.DAT. This contains the device number for the TUT. It is set to zero here since GETDEVIC needs to run to place the proper value in this position.

WRITELN(passer,timeout:6:3);

- The time-out error value is written to PASSER.DAT.

WRITE(passer,' 0',' 0');

- The beginning values for the forward and reverse bias settings are written. They are both zero at this point. STORDATA will replace these values with the proper settings.

CLOSE(passer)

END. {End Program}

APPENDIX D  
GETDEVIC.PAS PROGRAM LISTING

PROGRAM getdevice;

- This program is compiled using Turbo Pascal (8087 Compiler). It is used to create and maintain a file of records. The records are used to keep track of the devices being tested. The file is called INDEX.DEV, and it is kept on the C:\RBSOA directory. The reader should have a thorough understanding of Turbo Pascal. For information on Turbo Pascal, see the Reference Manual.

TYPE

STR3 = STRING[3];

STR13 = STRING[13];

STR80 = STRING[80];

PASSARRAY = ARRAY[1..10] OF CHAR;

- This data type could be declared as a STRING; however, it was found that referencing individual elements, characters, directly was difficult, unreliable and basically impossible using the standard STRING type. Declaring the TYPE PASSARRAY allowed for direct access to specific array elements.

VAR

passinfo : PASSARRAY;

- "Passinfo" is the ten character string found at the beginning of PASSER.DAT. It was important to be able to address specific characters directly so this is of TYPE PASSARRAY. Later programs use a different method to do this.

pass : STRING[12];

passer : TEXT;

bookkeeper,storage : STRING[13];

device : INTEGER;

ForBias,RevBias,timeout : REAL;

answer : CHAR;

{\*\*\*\*\*}

```
PROCEDURE halter(errmsg : STR80);
```

- This is the same procedure used in `HARDWARE.PAS` modified for Turbo Pascal. It performs the same function, orderly discontinuance of the entire acquisition cycle when program continuance would result in a crash.

```
VAR
```

```
  name : STRING[12];
```

```
  going : TEXT[15];
```

```
BEGIN
```

```
  name := 'G:HALTER.PGM';
```

```
  ASSIGN(going,name);
```

```
  REWRITE(going);
```

```
  WRITELN(going,'There was some type of error in Program Getdevic.');
```

```
  WRITELN(going,errmsg);
```

```
  WRITELN(going,'PROGRAM HALTED');
```

```
  CLOSE(going);
```

```
  HALT
```

```
END;
```

```
{ ***** }
```

```
PROCEDURE Devices (VAR devic : INTEGER; VAR bookout,storout : STR13);
```

- "Devices" is used to create and maintain the device file. It allows the operator to see a complete listing of the file and add records, if desired. From the list provided, the operator selects the device record corresponding to the device under test. It passes out to the main program the names of the storage and bookkeeping files applicable for the devices selected.

```
CONST
```

```
  first : BOOLEAN = FALSE;
```

## TYPE

STR40 = STRING[40];

STR13 = STRING[13];

DEVICES = RECORD

- Declaration of the record type to be used in the file INDEX.DEV.

DEV : INTEGER;

descrip : STR40;

book : STR13;

stor : STR13

END;

## VAR

index : FILE OF DEVICES;

indexrec : DEVICES;

error,Count,MaxLineCount,LineCount,addeddevices : INTEGER;

test : BOOLEAN;

induct,ans : CHAR;

Four : STRING[4];

Three : STRING[3];

Two : STRING[2];

One1,One2 : STRING[1];

## BEGIN

test := TRUE;

addeddevices := 0;

ASSIGN(index,'C:\RBSOA\INDEX.DEV');

- The text variable "index" becomes the file designation for the file INDEX.DEV.

## { \$I- }

- This compiler option turns off the IO error checking normally done. It permits the program to keep running in the event the next command causes an IO error; however, the error condition must then be checked within the software.

RESET(index);



{ \$I+ }

IF IOresult  $\neq$  0 THEN BEGIN

- The error condition is checked. An value of "IOresult" other than zero indicates an error. In this case, the error is assumed to be file not found. So, the file is created in the lines below.

WRITELN('C:INDEX.DEV not found. Assumed first time used. ');

WRITELN('Creating C:\RBSOA\INDEX.DEV.');

first := TRUE;

REWRITE(index)

- The file INDEX.DEV is created.

END

ELSE BEGIN

WRITELN('You will be asked for the device number for the device under test.');

ans := 'Q';

WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO BEGIN

- This "WHILE NOT" loop is used throughout all of the system software. It can only be exited if the operator responds with either an upper or lower case "Y" or "N." It is the typical "idiot" proofing used on all yes or no responses. It also ensure that the value of "ans" is known when the loop is left.

WRITELN('Do you want to see a listing of all currently cataloged devices in');

WRITE('the index file C:\RBSOA\INDEX.DEV? (Y or N) . . . ');

READLN(ans)

- Reads the operator's response from the keyboard.

END;

IF ((ans = 'Y') OR (ans = 'y')) THEN BEGIN

- If the answer is yes, then a listing of all the devices kept in INDEX.DEV is printed out to the screen.

LineCount := 1;

Count := 0;

SEEK(index, LineCount);

WHILE NOT EOF(index) DO BEGIN

- Until the end of the file INDEX.DEV, the steps between here and the "Ω" symbol are executed.

READ(index,indexrec);

- Reads one record from INDEX.DEV.

WITH indexrec DO BEGIN

WRITE(DEV,' ');

WRITE(descrip:40,' ');

WRITE(book,' ');

WRITELN(stor)

- The individual fields in the record read from INDEX.DEV are printed to the screen on one line.

END;

LineCount := LineCount + 1;

- The value "LineCount" is incremented by one. This is the record location pointer used in retrieving the next record.

Count := Count + 1;

- "Count" is used to cause the reading of device records to pause after 21. This lets the operator read them at his or her own pace.

IF Count = 21 THEN BEGIN

- If this is the 21<sup>st</sup> line since the last pause or the beginning of the list the reading will stop until the operator hits <RETURN>.

Count := 0;

WRITELN('Press <RETURN> for more devices.');

READLN

END

- Ω.

END;

WRITELN('End of indexed devices.')

END;

IF ((ans = 'N') OR (ans = 'n')) THEN

- If the operator did not want to see a listing, he or she is now asked for the number corresponding to the device under test. This is the same number that is printed in the field "DEV."

WHILE test DO BEGIN

- This numeric response is "idiot" proofed somewhat (It will not detect a wrong number). The operator response is read in as a string then

converted to numeric format with error checking done to be sure the entry is indeed a number.

```
WRITE('Enter the device number. ');
```

```
READLN(Four);
```

```
VAL(Four,LineCount,error);
```

- Conversion to numeric format is done here.

```
IF error = 0 THEN test := FALSE
```

- If there is no error (ie, error = 0), then the loop is exited by setting "test" equal to "FALSE".

```
END
```

```
END;
```

```
ans := 'Q';
```

```
IF (first OR NOT test) THEN ans := 'N'
```

- This is a complex test done to see what has occurred above. The value "first" is BOOLEAN. If it is false, then this is not the first time the file, INDEX.DEV, has been used. In that case, the value "ans" will be set to "N." The other condition of the test is checking the value of "test." If it is false, then the device number has already been input and "ans" can be set to "N" so the next part of the procedure will not execute.

```
ELSE BEGIN
```

- If "first" is FALSE and "test" is TRUE, then the following will be executed. Otherwise, the value of "ans" has already been set.

```
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO BEGIN
```

```
WRITE('Was the device you are going to be testing in the list? (Y or N) . . . ');
```

```
READLN(ans)
```

```
END
```

```
END;
```

```
IF (((ans = 'n') OR (ans = 'N')) AND test) THEN BEGIN
```

- If "ans" is set to a negative answer and "test" is TRUE (indicating that the device number has not as yet been specified), then the following, until the "ΩΩ", is executed.

```
LineCount := FileSize(index);
```

- "LineCount" holds the value representing the total number of records stored in INDEX.DEV.

IF LineCount = 0 THEN LineCount := 1;

- The first record in INDEX.DEV is number 0. This is not used in the system software, and this statement forces that record to be skipped.

test := TRUE;

WHILE test DO BEGIN

- This is the top of a loop designed to let the operator double check the device entry he or she is about to make. Until "test" is FALSE, the loop will execute. End of the loop is denoted by "ΩΩΩ".

WITH indexrec DO BEGIN

WRITELN('Enter the Device Description for Device Number ',LineCount);

WRITELN('XX - X's indicate 40 Characters.');

READLN(descr);

- Reads in a 40 character long description of the device under test.

DEV := LineCount;

- Gives the device number,"DEV", the value of LineCount. This way the device record can be referred to and accessed using only the device number.

WRITELN('The following is a list of inductor value codes. Please enter the ');

WRITELN('the SECOND letter corresponding to the value of the inductor being used.');

WRITELN('LG . . . 44  $\mu$ H');

WRITELN('LH . . . 146  $\mu$ H');

WRITELN('LI . . . 267  $\mu$ H');

WRITELN('LJ . . . 426  $\mu$ H');

WRITELN('LK . . . 1 mH');

WRITELN('LL . . . 2.16mH');

READLN(induct);

- Since different inductors are used, a way to reference them needed to be incorporated into the system software. It had to be known from the start what the inductance was. So, a two letter code was developed to be used as part of the storage file and bookkeeping file names. The

first letter "L" indicates the load is an inductor. The second letter lets it be known what the value of the inductance was. The reason the second letter does not start with "A" or end with "Z" is to allow other inductances to be added at either end. It was deemed unlikely that values in between the present ones would be used.

```
induct := UPCASE(induct);
```

- Translates the character "induct" to uppercase, if needed.

```
INSERT(induct,One1,1);
```

- Places "induct" into the one character string "One1".

```
IF DEV < 10 THEN BEGIN
```

- Creates the storage and bookkeeping filenames for the case of a device number less than 10.

```
STR(LineCount,One2);
```

```
book := CONCAT('G:DEV00',One2,'L',One1,'.BK');
```

```
stor := CONCAT('G:BD00',One2,'L',One1,'.000')
```

```
END;
```

```
IF ((DEV > 9) AND (DEV < 100)) THEN BEGIN
```

- Creates the storage and bookkeeping filenames for the case of a device number greater than 9 and less than 100.

```
STR(LineCount,Two);
```

```
book := CONCAT('G:DEV0',Two,'L',One1,'.BK');
```

```
stor := CONCAT('G:BD0',Two,'L',One1,'.000')
```

```
END;
```

```
IF ((DEV > 99) AND (DEV < 1000)) THEN BEGIN
```

- Creates the storage and bookkeeping filenames for the case of a device number greater than 99 and less than 1000.

```
STR(LineCount,Three);
```

```
book := CONCAT('G:DEV',Three,'L',One1,'.BK');
```

```
stor := CONCAT('G:BD0',Three,'L',One1,'.000')
```

```
END;
```

```
WRITELN('Here is the entry for Device Number ',DEV);
```

```
WRITELN(DEV:3,' ',descrip:40,' ',book,' ',stor);
```

- This line lets the operator see the entire device record just created before it is added to INDEX.DEV.

```

END;
ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO BEGIN
    WRITE('Is the entry alright? (Y or N) . . . ');
    READLN(ans)
    

- Asks the operator to verify the accuracy of the record just created.


END;
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
    

- The device record is added to the INDEX.DEV.


    addeddevices := addeddevices + 1;
    SEEK(index, LineCount);
    WRITE(index, indexrec);
    ans := 'Q';
    WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO
        BEGIN
            WRITE('Do you want to add another Device to the file? (Y or N) . . . ');
            READLN(ans)
            

- The operator can add another record, if so desired.


        END;
    IF ((ans = 'n') OR (ans = 'N')) THEN test := FALSE
    ELSE LineCount := LineCount + 1;
END
ELSE BEGIN
    ans := 'Q';
    WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO
        BEGIN
            WRITE('Do you want to try again? (Y or N) . . . ');
            READLN(ans)
            

- If the operator was not satisfied with the last record created, the operator
                is asked if he or she wants to try again. A negative answer halts the
                program.


        END;
    IF ((ans = 'n') OR (ans = 'N')) THEN
        HALTER('Operator unwilling to try to define device.')

```

```

END
END
      • ΩΩΩ
END;
      • ΩΩ
IF (((ans = 'Y') OR (ans = 'y')) AND test) THEN BEGIN
      • This test uses the value placed in "ans" when the operator was asked if
        the device under test was in the list provided when the contents of
        INDEX.DEV was printed to the screen. A positive response means
        the device was in the list, and if "test" is TRUE, then the device num-
        ber has not yet been specified.
test := TRUE;
WHILE test DO BEGIN
      • The device number is now entered using the same error trapping routine
        used before and a check is made to see if the device number is within
        the range of values in INDEX.DEV.
WRITE('Enter the device number. ');
READLN(Four);
VAL(Four,LineCount,error);
IF error = 0 THEN test := FALSE;
MaxLineCount := FileSize(index);
IF MaxLineCount < LineCount THEN BEGIN
  WRITELN('Device Number exceeds current index limit (ie, no such device).');
  WRITELN('Try again.');
```

```

  test := TRUE
END
END
END;
test := TRUE;
WHILE test DO BEGIN
      • This loop displays the record for the device selected and asks the operator
        if it is correct. If it is, the storage and bookkeeping filenames are
        placed into the STRING variables "storout" and "bookout" in order to
        be passed out of the procedure to the main program, and the loop
```

ends. If the device is incorrect, the operator is asked for a new device number, the record for the new device is displayed and the operator is again asked to verify the record.

```

    FIVEK(index,LineCount);
    WRITELN('This is the device you selected. ');
    READ(index,indexrec);
    WITH indexrec DO BEGIN
        WRITE(DEV,' ');
        WRITE(descrip:40,' ');
        WRITE(book,' ');
        WRITELN(stor)
    END;
    ans := 'Q';
    WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO BEGIN
        WRITE('Is this the correct device? (Y or N) . . . ');
        READLN(ans)
    END;
    IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
        test := FALSE;
        WITH indexrec DO BEGIN
            devic := DEV;
            bookout := book;
            storout := stor
        END
    END
    ELSE BEGIN
        test := TRUE;
        WHILE test DO BEGIN
            WRITE('Enter the correct device number. ');
            READLN(Four);
            VAL(Four,LineCount,error);
            IF error = 0 THEN test := FALSE
        END;
        test := TRUE
    END

```



```

END
END;
CLOSE(index)
END;

```

```

{ **** }

```

```

BEGIN {MAIN PROGRAM}

```

```

pass := 'G:PASSER.DAT';
ASSIGN(passer,pass);
RESET(passer);

```

- The file PASSER.DAT is opened, ready to be read.

```

READLN(passer,passinfo);
READLN(passer,device);
READLN(passer,timeout);
READ(passer,ForBias,RevBias);

```

- The first four lines of PASSER.DAT are read into GETDEVIC. The last two lines are not needed.

```

CLOSE(passer);

```

```

IF ((passinfo[1] = 'Y') OR (passinfo[1] = 'y')) THEN BEGIN

```

- If the first character in the 10 character string in PASSER.DAT is a "Y" or "y," then the device under test is a new device, different from the last device tested, or this is the first time through the acquisition cycle. In either case, new bookkeeper and storage filenames need to be placed in PASSER.DAT lines five and six.

```

Devices(device,bookkeeper,storage);

```

- The new names are found.

```

WHILE NOT ((answer = 'y') OR (answer = 'Y') OR (answer = 'n') OR (answer = 'N')) DO BEGIN

```

- The bookkeeping file needs to be transferred from drive A: or created, if this is the first test for the device. This answer to this question tells the program STORDATA where to look for the bookkeeping file.

```

WRITELN('Has the book keeping file for this device been transferred to the ');

```

```
WRITE('RAM disk (drive G:)? {If you are not sure, answer "N"} (Y or N) . . . ');
READLN(answer)
END;
IF ((answer = 'Y') OR (answer = 'y')) THEN passinfo[7] := 'N';
IF ((answer = 'N') OR (answer = 'n')) THEN passinfo[7] := 'Y';
    • The seventh character in the first line of PASSER.DAT is used to convey
      the necessary information to STORDATA.
REWRITE(passer);
WRITELN(passer,passinfo);
WRITELN(passer,device);
WRITELN(passer,timeout);
WRITELN(passer,ForBias,RevBias);
WRITELN(passer,bookkeeper);
WRITELN(passer,storage);
CLOSE(passer)
    • The new PASSER.DAT file overwrites the old one.
END
END. {MAIN PROGRAM}
```

APPENDIX E  
STORDATA.PAS PROGRAM LISTING

PROGRAM Bookkeeping;

- This program is responsible for creating and maintaining the bookkeeping file. Written for the Turbo Pascal Compiler with 8087 support.

TYPE

ST10 = STRING[10];  
ST13 = STRING[13];  
STR80 = STRING[80];

VAR

keeper,stor : STRING[13];  
stopper : TEXT;  
passinfo : STRING[10];  
break,recordit : CHAR;  
device : INTEGER;  
ForBias,RevBias,timeout : REAL;

{\*\*\*\*\*}

PROCEDURE halter(errmsg : STR80);

- This is the same procedure found in all of the previous programs in the acquisition cycle. It is exactly the same as the procedure in GETDEVIC.

VAR

name : STRING[12];  
going : TEXT[15];

BEGIN

name := 'G:HALTER.PGM';  
ASSIGN(going,name);  
REWRITE(going);

```

WRITELN(going,'There was some type of error in Program Stordata. ');
WRITELN(going,errmsg);
WRITELN(going,'PROGRAM HALTED');
CLOSE(going);
HALT
END;

```

```
{ ***** }
```

```

PROCEDURE pass_in_out (VAR info : ST10; VAR DEV : INTEGER;
                      VAR timeo,fBias,rBias : REAL;
                      VAR bookkeeper,storage : ST13; IN_OUT : CHAR);

```

- This procedure is used in all of the succeeding acquisition cycle programs. It is used to read or write the file PASSER.DAT. Depending upon the character "IN\_OUT," the procedure reads the values in PASSER.DAT or writes new values to PASSER.DAT. The values read are passed to the main program as global variables. The same variables, possibly altered, are passed back to this procedure prior to the end of the program to be written to PASSER.DAT.

```
VAR
```

```

pass : STRING[13];
passer : TEXT;

```

```
BEGIN
```

```

pass := 'G:PASSER.DAT';
ASSIGN(passer,pass);
IF IN_OUT = 'I' THEN BEGIN

```

- If this test is met, then the values are read from PASSER.DAT.

```

RESET(passer);
READLN(passer,info);
READLN(passer,DEV);
READLN(passer,timeo);
READLN(passer,fBias,rBias);

```

```

READLN(passer,bookkeeper);
READLN(passer,storage);
CLOSE(passer)
END;
IF IN_OUT = 'O' THEN BEGIN
    • If this test is met, then the values are written to PASSER.DAT.
    REWRITE(passer);
    WRITELN(passer,info);
    WRITELN(passer,DEV);
    WRITELN(passer,timeo);
    WRITELN(passer,fBias,rBias);
    WRITELN(passer,bookkeeper);
    WRITELN(passer,storage);
    CLOSE(passer)
END
END; {End Pass_In_Out}

{ **** }

PROCEDURE findbookkeeper(bk : ST13; newdevice : CHAR);
    • This procedure is concerned with first, finding out if the bookkeeping file
      exists. If it doesn't, then it creates a new bookkeeping file on the RAM
      disk. If it does exist, the procedure will find it and transfer it to the
      RAM disk if it hasn't been already.

VAR
    transfer,ans : CHAR;
    book,temp : TEXT;
    abk : STRING[13];
    R,I : INTEGER;

BEGIN
    ASSIGN(book,bk);
    IF ((newdevice = 'y') OR (newdevice = 'Y')) THEN BEGIN

```

- The variable "newdevice" is actual the seventh character in the first line of PASSER.DAT. If that value meets the test, then the following, until the "Ω" symbol, is executed.

```
ans := 'Q';
```

```
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO BEGIN
```

```
  WRITE('Is this the first time this device has been tested? (Y or N) . . . ');
```

```
  READLN(ans)
```

```
END;
```

```
IF ((ans = 'Y') OR (ans = 'y')) THEN BEGIN
```

```
  WRITELN('Creating new book keeping file . . . ');
```

```
  REWRITE(book);
```

```
  FOR I := 1 TO 10 DO
```

```
    WRITELN(book, '  0  0');
```

- If this is the first time the device has been tested, then a new bookkeeping file is created. The file's first 10 lines are "0 0." These numbers will be changed as forward and reverse bias settings replace them. This allows for 10 different forward and 10 different reverse bias setting.

```
  WRITELN(book, 'None.');
```

- The next line will contain the comment line, but for now, holds only the word "None."

```
  FOR I := 1 TO 10 DO BEGIN
```

```
    FOR R := 1 TO 9 DO
```

```
      WRITE(book, '  0  0');
```

```
    WRITELN(book, '  0  0')
```

- The final section of the bookkeeping file is 10 rows of 20 columns of zeros. See Chapter 3, Files Created and Individual Structure for more on this section.

```
  END;
```

```
  CLOSE(book)
```

```
END; {IF ((ans = 'Y...}
```

- Ω

```
IF ((ans = 'N') OR (ans = 'n')) THEN BEGIN
```

- If this was not the first time the device had been tested, then the following, until the "ΩΩ" symbol, will be executed.

```
abk := bk;
```

```
DELETE(abk,1,1);
```

```
INSERT('A',abk,1);
```

- These two lines change the bookkeeping filename from G:DEVxxxLL.BK to A:DEVxxxLL.BK and places that new name in the STRING, "abk."

```
ans := 'Q';
```

```
WHILE NOT (ans = 'Y') DO BEGIN
```

- This is the top of a loop which looks for the file on drive A: and copies the file to the RAM disk.

```
  WRITELN('Insert the disk containing ',abk,' in drive A:');
```

- This line tells the operator to place the disk containing the bookkeeping file in drive A:.

```
  WRITE('Press <RETURN> when ready.');
```

```
  READLN;
```

```
  ASSIGN(temp,abk);
```

```
{ $I- }
```

```
  RESET(temp);
```

```
{ $I+ }
```

```
  IF Ioresult = 0 THEN BEGIN
```

- The program now looks on drive A: for the bookkeeping program. If it finds the file then it will be copied to the RAM disk by the next four lines. The value "ans" will be set to "Y" and this WHILE loop will end. Otherwise, the loop is re-executed and the operator can place a different disk containing the correct file in drive A:.

```
    REWRITE(book);
```

```
    WHILE NOT EOF(temp) DO BEGIN
```

```
      READ(temp,transfer);
```

```
      WRITE(book,transfer)
```

- The file is transferred from A: to the RAM disk, G:, one character at a time until the End Of File is reached.

```
    END;
```

```
  CLOSE(book);
```

```
  CLOSE(temp);
```



```

    ans := 'Y'
END
ELSE BEGIN
    • If the "Ioresult" was not zero (ie, there was an error), then the screen is
      cleared and the line below is written to the screen.
    CLRSCR;
    WRITELN('ERROR ACCESSING FILE ON SPECIFIED DRIVE.');
```

ans := 'Q';

```

    WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO
    BEGIN
        • This gives the operator a chance to decide to try again or not. If the an-
          swer is no the program is halted using the procedure HALTER.
        WRITE('Try again? (Y or N) . . . ');
        READLN(ans)
    END;
    IF ((ans = 'Y') OR (ans = 'y')) THEN ans := 'Q';
    IF ((ans = 'N') OR (ans = 'n')) THEN
        HALTER('User initiated Halt.')
    END {ELSE}
END {WHILE NOT ((ans = 'Y'...)}
    • End of the loop which transfers the bookkeeping file to drive G:.
```

```

END {IF ((ans = 'n...)}
    • ΩΩ
END {IF newdevice...}
    • If "newdevice" had been "N," then the entire procedure would have been
      skipped, and it would have been assumed that the file had already been
      transferred to the RAM disk. This is done so that on repetitive tests the
      procedure does not execute when it is not needed.
END; {END PROCEDURE}

{*****}

PROCEDURE updatebk(bdstring : ST13; VAR fBias,rBias : REAL; VAR bd : CHAR);
    • This procedure is used to maintain and update the bookkeeping file.
```

TYPE

```
setting = ARRAY[1..2,1..10] OF REAL;
intarr = ARRAY[1..2,1..10,1..10] OF INTEGER;
```

VAR

```
s : setting;
testcount : intarr;
long : STRING[80];
Y,X,I,R,C : INTEGER;
temp,book : TEXT;
t : STRING[13];
transfer,ans : CHAR;
```

BEGIN

```
ASSIGN(book,bdstring);
RESET(book);
ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO BEGIN
  WRITE('Did break down occur? (Y or N) . . . ');
  READLN(ans)
```

- The response here serves two purposes. First, it tells the rest of the procedure what type of update needs to be done. Second, the response will be stored as the first letter in the first line of the storage file, BDxxxLLA.yyy. The response is very important to the programs, PLOTDATA and TABULATE.

```
END;
CLRSCR;
bd := ans;
WRITELN('Here are the current forward and reverse bias settings.');
```

```
WRITELN('Forward Bias . . . ',fBias:0:6,' Reverse Bias . . . ',rBias:0:6);
```

- These lines print to the screen the current values for the forward and reverse bias setting stored in PASSER.DAT. If they are not correct, they must be changed. As it was not envisioned that the settings would

remain constant for a large number of tests in a row, the update process is executed every acquisition cycle whether needed or not.

```
ans := 'Q';
FOR I := 1 TO 10 DO
  READLN(book,s[1,I],s[2,I]);
  • The first 10 lines of the bookkeeping file are read in to the 2 by 10 array
    "s."
```

```
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO
  BEGIN
    WRITELN('Is the Forward Bias Setting correct for the measurement just taken?');
    WRITE(' (Y or N) ... ');
    READLN(ans)
    • The question refers to the bias setting currently in PASSER.DAT.
  END;
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
  • If the setting is correct, the breakdown counts in section three of the
    bookkeeping file are updated as outlined in Chapter 3, Files Created
    and Individual Structure.
  X := 1;
  WHILE fBias <> s[1,X] DO X := X + 1
  • The column position of the appropriate breakdown count pair is located
    by matching the current bias setting to the one contained in the array
    "s."
```

```
END;
IF ((ans = 'n') OR (ans = 'N')) THEN BEGIN
  • If the setting is not correct, then a listing of the settings stored in the
    bookkeeping file is printed to the screen.
  WRITELN('Here are the Forward Bias Settings currently on disk (zero = no setting).');
  FOR I := 1 TO 5 DO
    WRITE(s[1,I]:0:6, ' ');
  WRITELN;
  FOR I := 6 TO 10 DO
    WRITE(s[1,I]:0:6, ' ');
  WRITELN;
```

```

ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO
  BEGIN
    Writeln('Is the setting used on this test in the above list?');
    Write(' (Y or N) . . . ');
    Readln(ans)
  END;
IF ((ans = 'n') OR (ans = 'N')) THEN BEGIN
  • If the current setting is not contained in the bookkeeping file, then the
    operator is asked to input the new setting.
  Write('What is the current setting? . . . ');
  Readln(fBias);
  I := 1;
  WHILE s[1,I] <> 0 DO
    I := I + 1;
  s[1,I] := fBias;
  X := I
  • The new setting is placed in the position of the first zero value in the ar-
    ray "s." The column location of the breakdown count pair is also ob-
    tained.
END;
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
  Write('Please enter the value of the current setting. . . ');
  Readln(fBias);
  X := 1;
  WHILE fBias <> s[1,X] DO X := X + 1
  • If the current setting is in the bookkeeping file, the operator is again
    asked to input the current setting. The setting is compared to those kept
    in the array "s." When a match is found, the location is used to provide
    the column of the breakdown count pair.
  • This method of file update is not elegant, logical or simple, but it does
    work. More trouble was involved in changing this procedure than it
    was worth. So, it wasn't.
END;

```

```

END; {If ((ans = 'n'...forward bias)
ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO
  BEGIN
    WRITELN('Is the Reverse Bias Setting correct for the measurement just taken?');
    WRITE(' (Y or N) . . . ');
    READLN(ans)
  END;
  • The exact same method used to update the forward bias setting was used
    to update the reverse. The only difference is that instead of using the
    s[1,X] values, the s[2,Y] values were using. Where Y is the row
    position of the breakdown count pair.
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
  Y := 1;
  WHILE rBias <> s[2,Y] DO Y := Y + 1
END;
IF ((ans = 'n') OR (ans = 'N')) THEN BEGIN
  WRITELN('Here are the Reverse Bias Settings currently on disk (zero = no setting).');
  FOR I := 1 TO 5 DO
    WRITE(s[2,I]:0:6, ' ');
  WRITELN;
  FOR I := 6 TO 10 DO
    WRITE(s[2,I]:0:6, ' ');
  WRITELN;
  ans := 'Q';
  WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO
    BEGIN
      WRITELN('Is the setting used on this test in the above list?');
      WRITE(' (Y or N) . . . ');
      READLN(ans)
    END;
  IF ((ans = 'n') OR (ans = 'N')) THEN BEGIN
    WRITE('What is the current setting? . . . ');
    READLN(rBias);

```

```

I := 1;
WHILE s[2,I] <> 0 DO
  I := I + 1;
s[2,I] := rBias;
Y := I
END;
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
  WRITE('Please enter the value of the current setting. . . ');
  READLN(rBias);
  Y := 1;
  WHILE rBias <> s[2,Y] DO Y := Y + 1
END;
END; {If ((ans = 'n'...reverse bias)
  • The column and row position for the breakdown count pair have now
    been found.
  • The following is involved with updating or not updating the comment
    line. If desired, the line can be changed.
WRITELN('Here is the current comment line for ',bdstring,'');
READLN(book,long);
WRITELN(long);
ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO
  BEGIN
    WRITE('Do you want to change the comment line? (Y or N) . . . ');
    READLN(ans)
  END;
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
  WRITELN('Enter new comment line.(Maximum of 80 characters)');
  READLN(long)
END;
  • This ends the section concerned with updating the comment line.
FOR R := 1 TO 10 DO
  FOR C := 1 TO 10 DO
    FOR I := 1 TO 2 DO

```

READ(book,testcount[I,C,R]);

- The third section of the bookkeeping file is read and place in a 3 D array.

IF ((bd = 'y') OR (bd = 'Y')) THEN BEGIN

testcount[1,X,Y] := testcount[1,X,Y] + 1;

testcount[2,X,Y] := testcount[2,X,Y] + 1

END;

- If breakdown occurred both the number of breakdowns, "testcount[1,X,Y]," and the total number of tests, "testcount[2,X,Y]," are increased by one.

IF ((bd = 'n') OR (bd = 'N')) THEN

testcount[2,X,Y] := testcount[2,X,Y] + 1;

- If breakdown didn't occur, then only the total number of tests is increased.

CLOSE(book);

REWRITE(book);

- The bookkeeping file is closed and then re-opened in preparation for it being overwritten with a new, updated, version of the bookkeeping file.

FOR I := 1 TO 10 DO

WRITELN(book,s[1,I], ' ',s[2,I]);

- The bias settings are written to the bookkeeping file on the RAM disk.

WRITELN(book,long);

- The comment line is written.

FOR R := 1 TO 10 DO BEGIN

FOR C := 1 TO 9 DO

FOR I := 1 TO 2 DO

WRITE(book,testcount[I,C,R], ' ');

WRITELN(book,testcount[1,10,R], ' ',testcount[2,10,R])

END;

- Section 3 is then written.

bd := UPCASE(bd);

CLOSE(book)

END;

```
{ ***** }
```

```
BEGIN {MAIN PROGRAM}
```

```
pass_in_out(passinfo,device,timeout,ForBias,RevBias,keeper,stor,'T');
```

- The global variables from PASSER.DAT are read.

```
findbookkeeper(keeper,passinfo[7]);
```

- The bookkeeping file is created or copied over to the RAM disk.

```
updatebk(keeper,ForBias,RevBias,break);
```

- The bookkeeping file is updated.

```
DELETE(passinfo,3,1);
```

```
INSERT(break,passinfo,3);
```

- The "Y" or "N" response to whether or not breakdown occurred is placed in the third character of the first line of PASSER.DAT.

```
recordit := 'Q';
```

```
WHILE NOT ((recordit = 'y') OR (recordit = 'Y') OR (recordit = 'n') OR  
            (recordit = 'N')) DO BEGIN
```

```
    WRITE('Do you want to keep this run? (Y or N) . . . ');
```

```
    READLN(recordit)
```

```
END;
```

- The operator is asked if he or she wants to store the data currently displayed on the 7612D.

```
IF ((recordit = 'y') OR (recordit = 'Y')) THEN BEGIN
```

```
    DELETE(passinfo,6,1);
```

```
    INSERT('Y',passinfo,6)
```

```
END
```

- If the data is to be kept, the sixth character in the first line of PASSER.DAT is set to "Y." This will cause PROCESS1 to read the data and place it in two files on the RAM disk similar to as was done in HARDWARE, Procedure Calibration. The character will also tell PROCESS2 to read those two files and calculate the Instantaneous Power and Energy.

```
ELSE BEGIN
```

```
    DELETE(passinfo,6,1);
```



```

INSERT('N',passinfo,6)
END;

```

- If the operator does not want the data, then "N" in position 6 of the first line of PASSER.DAT will tell both PROCESS1 and PROCESS2 not to execute.

```

recordit := 'Q';
WHILE NOT ((recordit = 'y') OR (recordit = 'Y') OR (recordit = 'n') OR
(recordit = 'N')) DO BEGIN
  WRITE('Do you want to measure the Base-Emitter Voltage? (Y or N) . . . ');
  READLN(recordit)
END;

```

- The operator is asked if he or she will measure  $V_{be}$ . If the answer is no, the second character in line one of PASSER.DAT will be set to "N" and the program will not execute. If the answer is yes, then the character is set to "Y" and PROCESS2 will pause as it finishes calculating P and E to allow the operator to acquire  $V_{be}$ . Then,  $V_{be}$  will be collected and written to a RAM disk file like  $V_{ce}$  and  $I_c$ .

```

IF ((recordit = 'y') OR (recordit = 'Y')) THEN BEGIN
  DELETE(passinfo,2,1);
  INSERT('Y',passinfo,2)
END
ELSE BEGIN
  DELETE(passinfo,2,1);
  INSERT('N',passinfo,2)
END;

```

```

pass_in_out(passinfo,device,timeout,ForBias,RevBias,keeper,stor,'O')

```

- The changes in the global variables are passed back to this procedure which then writes a new PASSER.DAT, destroying the old.

```

END. {MAIN PROGRAM}

```

APPENDIX F  
PROCESS1.PAS PROGRAM LISTING

```
PROGRAM Process_Level_One(INPUT,OUTPUT);
```

- This program reads the data captured on Channel A and B. It writes the data out to two separate files, one for each channel. This program is written for the MS-PASCAL compiler.

```
{*****}
```

```
[$INCLUDE: 'IODECL.EX']
```

- The file IODECL.EX contains the declarations necessary to the Command Library. These declarations are included as part of the code for this program by using this statement.

```
{*****}
```

```
CONST
```

```
bus = 7;
```

```
TYPE
```

```
STEXT = TEXT(45);
STR13 = STRING(13);
ST10 = STRING(10);
STR9 = STRING(9);
LST80 = LSTRING(80);
setting = ARRAY[1..2,1..10] OF REAL;
```

```
VAR
```

```
bookkeeper,storage : STRING(13);
passinfo : STRING(10);
answer,process : CHAR;
device : INTEGER;
ForBias,RevBias,timeout,Hscale : REAL;
```

```
{ ***** }
```

```
{ $INCLUDE: 'IOPROC.EX' }
```

- Includes all the external function declaration necessary to use the Command Library.

```
{ ***** }
```

```
PROCEDURE ENDXQQ; EXTERN;
```

- External procedure which is part of extended MS-PASCAL. Allows the program to be terminated prior to the normal ending. Performs the same function as HALT (not HALTER) in Turbo Pascal.

```
{ ***** }
```

```
PROCEDURE halter(errmsg : LST80);
```

- Same procedure as in the preceding programs. Halts the data acquisition software.

```
VAR
```

```
  name : STRING(12);
```

```
  going : TEXT(15);
```

```
BEGIN
```

```
  name := 'G:HALTER.PGM';
```

```
  ASSIGN(going,name);
```

```
  REWRITE(going);
```

```
  WRITELN(going,'There was some type of error in Program Process1.');
```

```
  WRITELN(going,errmsg);
```

```
  WRITELN(going,'PROGRAM HALTED.');
```

```
  CLOSE(going);
```

```
  ENDXQQ
```

```
END;
```

```
{ ***** }
```

```
PROCEDURE pass_in_out (VAR info : ST10; VAR DEV : INTEGER;
                      VAR timeo,fBias,rBias : REAL;
                      VAR bookkeeper,storage : STR13; IN_OUT : CHAR);
```

- Same as in STORDATA see that program for details.

```
VAR
```

```
  pass : STRING(12);
```

```
  passer : STEXT;
```

```
BEGIN
```

```
  pass := 'G:PASSER.DAT';
```

```
  ASSIGN(passer,pass);
```

```
  IF IN_OUT = 'I' THEN BEGIN
```

```
    RESET(passer);
```

```
    READLN(passer,info);
```

```
    READLN(passer,DEV);
```

```
    READLN(passer,timeo);
```

```
    READLN(passer,fBias,rBias);
```

```
    READLN(passer,bookkeeper);
```

```
    READLN(passer,storage);
```

```
    CLOSE(passer)
```

```
  END;
```

```
  IF IN_OUT = 'O' THEN BEGIN
```

```
    REWRITE(passer);
```

```
    WRITELN(passer,info);
```

```
    WRITELN(passer,DEV);
```

```
    WRITELN(passer,timeo);
```

```
    WRITELN(passer,fBias,rBias);
```

```
    WRITELN(passer,bookkeeper);
```

```
    WRITELN(passer,storage);
```

```
    CLOSE(passer)
```

```
  END
```

END; {End Pass\_In\_Out}

{\*\*\*\*\*}

PROCEDURE error\_handler (error : INTEGER; routine : STR9; A : CHAR);

- Same procedure as that found in HARDWARE. See that program for details.

VAR

estring : STRING(40);

BEGIN

IF error <> noerr THEN

BEGIN

Errstr(error,estring);

Writeln('Error in call to ',routine);

Writeln(error:6, ' ',estring);

IF A = 'N' THEN

BEGIN

WRITE('Press <RETURN> to continue . . .');

READLN

END;

IF A = 'Y' THEN

BEGIN

Writeln('CORRECT ERROR - Press <RETURN> to continue.');

READLN

END

END

END;

{\*\*\*\*\*}

PROCEDURE setup(tout : REAL);

- Sets the bus to a known state. Makes sure that the IOTIMEOUT is set.  
Sets the 7612D to remote so it is ready to receive later commands from the bus.

VAR

cmd :INTEGER;

endline : STRING(2);

BEGIN

cmd := IOEOI(7,1);

error\_handler(cmd,'IOEOI ','Y');

endline[1] := CHR(13);

endline[2] := CHR(10);

cmd := IOEOL(7,endline,1);

cmd := IOREMOTE(7);

error\_handler(cmd,'IOREMOTE ','Y');

cmd := IOTIMEOUT(7,tout)

END;

{\*\*\*\*\*}

PROCEDURE horscale (VAR Hsc : REAL);

- This procedure reads the horizontal scale of the both channels of the 7612D. The values must be the same or the program will abort.

VAR

response : STRING(25);

I,cmd : INTEGER;

HscA,HscB : REAL;

vsc : STRING(5);

BEGIN

vsc := 'HSFA?';

cmd := IOOUTPUTS(70200,vsc,5);

- Sends the query to the 7612D which causes it to return the Horizontal Scale Factor of Channel A. The value is in seconds between data samples.

I := 25;

cmd := IOENTERS(70200,response,I);

- The characters preceding the scale factor are read.

cmd := IOENTER(70200,HscA);

- The horizontal scale factor is read.

cmd := IOENTERS(70200,response,I);

- The trailing characters are read to release the 7612D.

vsc := 'HSFB?';

cmd := IOOUTPUTS(70200,vsc,5);

- The procedure is repeated for Channel B.

I := 25;

cmd := IOENTERS(70200,response,I);

cmd := IOENTER(70200,HscB);

cmd := IOENTERS(70200,response,I);

IF HscA <> HscB THEN BEGIN

- The scale factors are compared. If not equal, the program is halted.

WRITELN('Horizontal Scale Factor are not equal. They must be.');

WRITELN('Program Halted.');

HALTER('Horizontal Scale Factor are not equal.')

END

ELSE Hsc := HscA

- The horizontal scale factor is passed out to the main program.

END;

{\*\*\*\*\*}

PROCEDURE ChannelA(HscA,fBias,rBias : REAL; stor : STR13; breakdown : CHAR);

- Channel A's data is read and written out to RAM disk.

VAR

line : STRING(128);



```

store : STEXT;
state,I,R,C,cmd : INTEGER;
response : STRING(40);
incoming : CHAR;
vsc : STRING(5);
VV,probeA : REAL;
readit : STRING(6);
going : STRING(14);

```

```
BEGIN
```

```
  WRITELN('Reading data from Channel A.');
```

```
  probeA := 1;
```

- Multiplication factor for the probe attached to Channel A. In this case, the plug-ins on the 7612D recognize the attenuation factor on the probe and automatically compensate for it. So, no scaling is needed. If the probes are changed, then these values need to be re-examined.

```
  FOR I := 1 TO 9 DO
```

```
    going[I] := stor[I];
```

```
  going[10] := 'A';
```

```
  FOR I := 10 TO 13 DO
```

```
    going[I+1] := stor[I];
```

- These last five lines take the storage filename, "stor," which was read from PASSER.DAT and transferred to this procedure and makes a new storage filename called, "going." "Going" has an additional letter "A" after the inductor designation signifying that the file contains the data from Channel A, namely  $V_{ce}$ . For more information see Chapter 3, Figure 8.

```
  ASSIGN(store,going);
```

```
  REWRITE(store);
```

```
  WRITELN(store,breakdown);
```

```
  WRITELN(store,fBias,'rBias);
```

- The breakdown "Y" or "N" character is written in the first line of the file. Next, the forward bias and reverse bias settings are written in line two.

```
  vsc := 'VSL1?';
```

```
cmd := IOOUTPUTS(70200,vsc,5);
```

```
I := 5;
```

```
cmd := IOENTERS(70200,vsc,I);
```

```
cmd := IOENTER(70200,VV);
```

```
I := 40;
```

```
cmd := IOENTERS(70200,response,I);
```

- The last seven lines read the value of the Volts/Div setting on Channel A.

```
WRITELN(store,VV,' ',HscA,' ',probeA);
```

- The value of the vertical scale, the horizontal time scale and the probe multiplication factor are written to the storage file in line three.

```
readit := 'READ A';
```

```
cmd := IOOUTPUTS(70200,readit,6);
```

- The 7612D is ordered to transmit the data on Channel A.

```
FOR R := 1 TO 3 DO BEGIN
```

```
  I := 1;
```

```
  cmd := IOENTERS(70200,incoming,I)
```

```
END;
```

- The first three characters in the data stream sent by the 7612D are stripped. They are unused control characters.

```
state := 0;
```

```
cmd := IOMATCH(7,CHR(10),state);
```

- The end of line test is disabled. This prevents the data strings about to be read from being terminated before the required 128 characters have been read. The data is binary. If one of the bytes has the same value as the character used to sense the end of line, then the line would end prematurely if the checking were left on.

```
FOR R := 0 TO 15 DO BEGIN
```

```
  I := 128;
```

```
  cmd := IOENTERS(70200,line,I);
```

```
  WRITE(store,line)
```

- Sixteen sets of 128 characters are read from the 7612D and then written to the RAM disk. The process takes less than one second per channel this way. If the characters were read one at a time, instead of in strings, it would take over 30 seconds to read all of the data.

```

END;
CLOSE(store);
state := 1;
cmd := IOMATCH(7,CHR(10),state);

```

- The check for the end of line character is turned back on. It would be dangerous to leave it off except when absolutely necessary.

```

I := 40;
cmd := IOENTERS(70200,response,I)

```

- The trailing control characters are stripped of the output from Channel A.

```

END;

```

```

{ **** }

```

```

PROCEDURE ChannelB (HscB,fBias,rBias : REAL; stor : STR13);

```

- This procedure performs the same function as "ChannelA." Only the differences are noted.

```

VAR

```

```

line : STRING(128);
store : STEXT;
state,I,R,C,cmd : INTEGER;
incoming : CHAR;
vsc : STRING(5);
VI,probeB : REAL;
readit : STRING(6);
going : STRING(14);
response : STRING(40);

```

```

BEGIN

```

```

  WRITELN('Reading data from Channel B. ');
  probeB := 10;
  FOR I := 1 TO 9 DO
    going[I] := stor[I];
  going[10] := 'B';

```

- "B" is inserted as "A" was in the preceding procedure.

```

FOR I := 10 TO 13 DO
  going[I+1] := stor[I];
  vsc := 'VSR1?';
  cmd := IOOUTPUTS(70200,vsc,5);
  I := 5;
  cmd := IOENTERS(70200,vsc,I);
  cmd := IOENTER(70200,VI);
  I := 40;
  cmd := IOENTERS(70200,response,I);
  readit := 'READ B';

```

- The data from Channel B is read instead of Channel A.

```

cmd := IOOUTPUTS(70200,readit,6);
ASSIGN(store,going);
REWRITE(store);
WRITELN(store,fBias,' ',rBias);
WRITELN(store,VI,' ',HscB,' ',probeB);
FOR R := 1 TO 3 DO BEGIN
  I := 1;
  cmd := IOENTERS(70200,incoming,I)
END;
state := 0;
cmd := IOMATCH(7,CHR(10),state);
FOR R := 0 TO 15 DO BEGIN
  I := 128;
  cmd := IOENTERS(70200,line,I);
  WRITE(store,line)
END;
state := 1;
cmd := IOMATCH(7,CHR(10),state);
I := 40;
cmd := IOENTERS(70200,response,I);
cmd := IOLOCAL(7);
CLOSE(store)

```

END;

{\*\*\*\*\*}

PROCEDURE addone(VAR stor : STR13);

- This procedure adds one to the test series number at the end of the storage filename. See Chapter 3, Figure 8 for more on the series number.

VAR

I : INTEGER;

BEGIN

IF ORD(stor[13]) = 57 THEN BEGIN

IF ORD(stor[12]) = 57 THEN BEGIN

IF ORD(stor[11]) = 57 THEN BEGIN

- If the series number is 999, then ...

WRITELN('Too Many Indexed Devices. Program Halted.');

HALTER('Too Many Indexed Devices.')

END

ELSE BEGIN

I := ORD(stor[11]) + 1;

stor[11] := CHR(I);

stor[12] := CHR(48);

stor[13] := CHR(48)

- If the series number is x99 (where x is 0 to 8), then the "x" is increased by one and the two nines are set to zero.

END

END

ELSE BEGIN

I := ORD(stor[12]) + 1;

stor[12] := CHR(I);

stor[13] := CHR(48)

- If the series number is xy9 (where y is 0 to 8), then y is increased by one and the nine is set to zero.

```

END
END
ELSE BEGIN
  I := ORD(stor[13]) + 1;
  stor[13] := CHR(I)
  • If the series number is xyz (where z is 0 to 8), then z is increased by one.
END
END;

{ **** }

BEGIN
  pass_in_out(passinfo,device,timeout,ForBias,RevBias,bookkeeper,storage,'I');
  IF ((passinfo[6] = 'Y') OR (passinfo[6] = 'y')) THEN BEGIN
    addone(storage);
    setup(timeout);
    horscale(Hscale);
    ChannelA(Hscale,ForBias,RevBias,storage,passinfo[3]);
    • Passinfo[3] is the third character, first line, of PASSER.DAT. It is used
      to hold the "Y" or "N" indicator of breakdown stored in the first line of
      the Channel A storage file.
    ChannelB(Hscale,ForBias,RevBias,storage)
  END;
  pass_in_out(passinfo,device,timeout,ForBias,RevBias,bookkeeper,storage,'O');
  ENDXQQ
END. {End Program}

```

APPENDIX G  
PROCESS2.PAS PROGRAM LISTING

PROGRAM Process\_Level\_Two; {For use in Batch processing only!}

- This program is used to calculate the Instantaneous Power and Energy from the data stored on the RAM disk in PROCESS1. The program is written for the Turbo Pascal compiler with 8087 support.

TYPE

```
ST10 = STRING[10];
STR13 = STRING[13];
STR80 = STRING[80];
datarray = ARRAY[1..3,1..2048] OF INTEGER;
ene = ARRAY[1..2047] OF REAL;
```

VAR

```
passinfo : STRING[10];
device : INTEGER;
timeout,ForBias,RevBias : REAL;
bookkeeper,storage : STR13;
VIP : datarray;
globEnergy : ene;
```

{\*\*\*\*\*}

PROCEDURE halter(errmsg : STR80);

- This is the same procedure found in all the previous programs. See HARDWARE or GETDEVIC for more.

VAR

```
name : STRING[12];
going : TEXT[15];
```

BEGIN

```
name := 'G:HALTER.PGM';
ASSIGN(going,name);
```



```

REWRITE(going);
WRITELN(going,'There was some type of error in Program Process2. ');
WRITELN(going,errmsg);
WRITELN(going,'PROGRAM HALTED');
CLOSE(going);
HALT
END;

```

```

{ **** }

```

```

PROCEDURE pass_in_out (VAR info : ST10; VAR DEV : INTEGER;
                      VAR timeo,fBias,rBias : REAL;
                      VAR bk,stor : STR13; IN_OUT : CHAR);

```

- This is the same procedure seen in STORDATA and PROCESS1. See STORDATA for more.

```

VAR
  pass : STRING[13];
  passer : TEXT;

BEGIN
  pass := 'G:PASSER.DAT';
  ASSIGN(passer,pass);
  IF IN_OUT = 'T' THEN BEGIN
    RESET(passer);
    READLN(passer,info);
    READLN(passer,DEV);
    READLN(passer,timeo);
    READLN(passer,fBias,rBias);
    READLN(passer,bk);
    READLN(passer,stor);
    CLOSE(passer)
  END;
  IF IN_OUT = 'O' THEN BEGIN

```

```

REWRITE(passer);
Writeln(passer,info);
Writeln(passer,DEV);
Writeln(passer,timeo);
Writeln(passer,fBias,rBias);
Writeln(passer,bk);
Writeln(passer,stor);
CLOSE(passer)
END
END; {End Pass_In_Out}

```

```
{*****}
```

```
PROCEDURE Calculate (stor : STR13; VAR energy : ene);
```

- This procedure is the meat of this program.  $V_{ce}$  and  $I_c$  are read from the RAM disk, the values are multiplied together to produce the Instantaneous Power,  $P$ , and  $P$  is then integrated. Integration is started at the one time step back from the point where  $V_{ce}$  is 5% of its maximum value and continues until the time step just after the point where  $P$  is 3% of its maximum value on the descending edge. An energy array is passed to the main program. There are 2047 elements in the array. Each element represents the cumulative energy calculate to the corresponding time interval. The array contains zeros to the point where integration begins and zeros after integration ends.

```
VAR
```

```

R,I,Vmax,VstartPos,PstopPos : INTEGER;
going : STRING[14];
VIP : datarray;
incoming : CHAR;
Pmax,Pstop,eMax,Vstart,VscP,VI,VV,HscA,HscB,probeA,probeB,fBias,rBias,
intmed1,intmed2 : REAL;
store : TEXT[25];

```

BEGIN

eMax := 0;

FOR R := 1 TO 2047 DO energy[R] := 0;

- The energy array is initialized.

going := stor;

INSERT('A',going,10);

ASSIGN(store,going);

RESET(store);

- The storage filename held in "stor" is placed into the string "going" and an "A" is inserted after the inductor value. "stor" = G:BDxxxLL.yyy; "going" = G:BDxxxLLA.yyy. This is the file containing the data from Channel A ( $V_{ce}$ ).

READLN(store);

READLN(store,fBias,rBias);

READLN(store,VV,HscA,probeA);

- The first three lines of "store" (the file designation of "going") are read.

FOR R := 1 TO 2048 DO BEGIN

  READ(store,incoming);

  I := ORD(incoming);

  VIP[1,R] := I - 127

- The 2048 data points are read. The ASCII values are converted to integer values and then correction is made for the zero offset. The zero offset is explained in detail in Chapter 3, Data Acquisition Programs.

END;

CLOSE(store);

going := stor;

INSERT('B',going,10);

ASSIGN(store,going);

RESET(store);

- The file containing the data from Channel A is closed, and the file containing the data from Channel B is open.

READLN(store,fBias,rBias);

READLN(store,VI,HscB,probeB);

- The first two lines of the file are read. Reminder, the Channel A data file contains an extra line in the beginning; otherwise, the files are identical in structure.

IF HscA  $\neq$  HscB THEN BEGIN

WRITELN('Horizontal scale factors not equal. Program Halted.');

HALTER('Horizontal scale factors not equal.')

- The horizontal scale factors are compared. If they are not equal, the program is halted. It is absolutely imperative that the data have the same time scale as the data points are simply multiplied together to get P. If the time scales are different, then the data points on  $V_{ce}$  and  $I_c$  will not correspond to the same point in the test cycle, and a multiplication of the two values would be meaningless.

END;

FOR R := 1 TO 2048 DO BEGIN

READ(store,incoming);

I := ORD(incoming);

VIP[2,R] := I - 127

- The data points for  $I_c$  are read, converted the integers from ASCII and corrected for zero offset.

END;

CLOSE(store);

VscP := probeA\*probeB\*VV\*VI/1024;

- The vertical scale factor for P is calculated. There are 32 discrete levels in each Volt/Div setting. When the vertical scale factor is created this is taken into account by dividing by  $32 * 32 = 1024$ .

FOR R := 1 TO 2048 DO

VIP[3,R] := VIP[1,R]\*VIP[2,R];

- The Instantaneous Power is calculated.  $VIP[1,R] = V_{ce}$ ;  $VIP[2,R] = I_c$ ;  
 $VIP[3,R] = P$ .

Vmax := -128;

FOR R := 1 TO 2048 DO

IF Vmax < VIP[1,R] THEN Vmax := VIP[1,R];

- The maximum value of  $V_{ce}$  is found.

Vstart := 0.05\*Vmax;

```

VstartPos := 1;
WHILE Vstart > VIP[1,VstartPos] DO
  VstartPos := VstartPos + 1;
  • The starting position of the integration is found by finding the time step
    where 5% of the maximum value of  $V_{ce}$  is passed, and letting the start
    position be the point just prior to where this occurred.
Pmax := -100000.00;
FOR R := 1 TO 2048 DO
  IF Pmax < VIP[3,R] THEN Pmax := VIP[3,R];
  • The maximum value of P is found.
Pstop := 0.03*Pmax;
  • The threshold value of 3% of the maximum value of P is calculated.
PstopPos := 1;
WHILE Pmax <> VIP[3,PstopPos] DO
  PstopPos := PstopPos + 1;
  • The position of the maximum value of P is determined.
WHILE Pstop < VIP[3,PstopPos] DO
  PstopPos := PstopPos + 1;
PstopPos := PstopPos + 1;
  • The position of the time step where P descends passed the threshold
    value is determined.
FOR R := 1 TO 2047 DO
  energy[R] := 0;
FOR R := VstartPos TO PstopPos DO BEGIN
  IF (VIP[3,R]+VIP[3,(R+1)]) < (ABS(VIP[3,R])+ABS(VIP[3,(R+1)])) THEN BEGIN
    IF VIP[3,R] > VIP[3,(R+1)] THEN BEGIN
      intmed1 := VscP*VIP[3,R];
      intmed2 := VscP*VIP[3,(R+1)];
      eMax := eMax + (0.5*SQR(intmed1)*HscA/(ABS(intmed1)+ABS(intmed2)))
        - 0.5*SQR(intmed2)*HscA/(ABS(intmed1)+ABS(intmed2))
    END;
    IF VIP[3,R] < VIP[3,(R+1)] THEN BEGIN
      intmed1 := VscP*VIP[3,R];
      intmed2 := VscP*VIP[3,(R+1)];

```

```

eMax := eMax + (0.5*SQR(intmed1)*HscA/(ABS(intmed1)+ABS(intmed2)))
+ 0.5*SQR(intmed2)*HscA/(ABS(intmed1)+ABS(intmed2))
END
END
ELSE BEGIN
IF ABS(VIP[3,R]) > ABS(VIP[3,(R+1)]) THEN
eMax := eMax + VscP*(VIP[3,(R+1)]*HscA + 0.5*HscA*(VIP[3,R]
-VIP[3,(R+1)]));
IF ABS(VIP[3,R]) = ABS(VIP[3,(R+1)]) THEN
eMax := eMax + VscP*VIP[3,R]*HscA;
IF ABS(VIP[3,R]) < ABS(VIP[3,(R+1)]) THEN
eMax := eMax + VscP*(VIP[3,R]*HscA + 0.5*HscA*(VIP[3,(R+1)]-VIP[3,R]))
END;
energy[R] := eMax

```

- The energy is calculated. The formula used is simply the area of the rectangle formed by the time step and the lesser of the two vertical displacements, plus the triangle. The steps above are used to determine if the vertical displacement are equal or not, and if not, then which is greater. The following illustration (Figure 18.) gives a pictorial view of the integration scheme used.

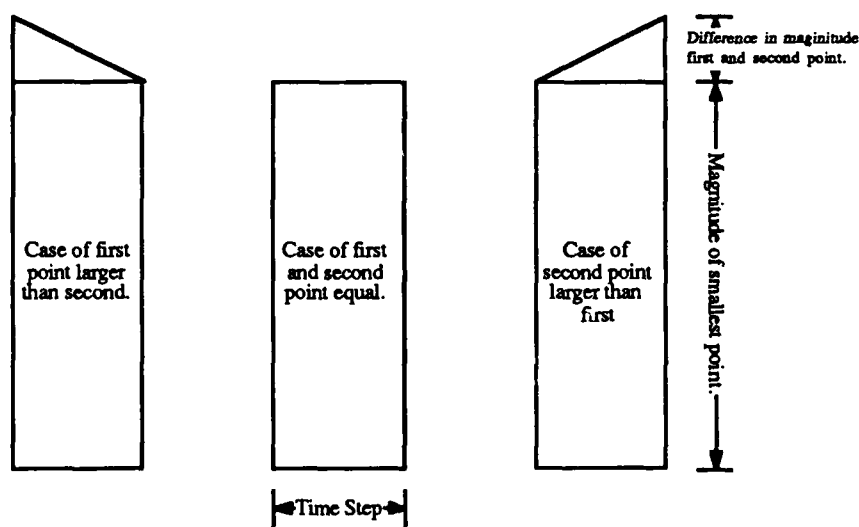


Figure 18. Integration Scheme.

The scheme also includes a check for P crossing negative where negative energy results (ie, the transistor is supplying energy), but this is not shown in the figure.

```
END;
going := stor;
going[1] := 'A';
INSERT('P',going,10);
ASSIGN(store,going);
REWRITE(store);
WRITELN(store,fBias,rBias);
WRITELN(store,eMax,HscA,VscP);
```

- The storage file for P is open on drive A:, the forward and reverse bias settings are written in line one, and the total energy, horizontal scale factor and vertical scale factor are written in line 2.

```
FOR R := 1 TO 2048 DO
```

```
  WRITE(store,VIP[3,R],' ');
```

- A series of 2048 integer values are then written, all separated by a space. The Instantaneous Power is written unscaled to disk (ie, the integer product of the integer multiplication of the Channel A and B data is written to disk). P is not converted to real numbers in an effort to save storage space on disk. To convert the values stored on this file into the actual watts calculated, simply multiply the individual integer values by the vertical scale factor.

```
  CLOSE(store)
END;
```

```
{ ***** }
```

```
PROCEDURE WriteToEnergy(stor : STR13; energy : ene);
```

- This procedure writes the energy array created in the last procedure to disk in its own file.

```
VAR
```

```
  going : STRING[14];
```

```

store : TEXT;
R : INTEGER;
fullstr : STRING[25];
outstr7 : STRING[7];
outstr5 : STRING[5];
outstr : STRING[12];

```

```
BEGIN
```

```

stor[1] := 'A';
going := stor;
INSERT('E',going,10);
ASSIGN(store,going);
REWRITE(store);

```

- The energy storage file, A:BDxxxLLE.yyy, is opened.

```
FOR R := 1 TO 2047 DO
```

```
IF energy[R] = 0 THEN WRITELN(store,'0')
```

- If the value in the energy array is zero, the character "0" (zero) is written to disk rather than the number zero. The number zero is written as " $\pm 0.0000000000000000E\pm000$ " rather than "0." By writing the value as a character, 24 bytes of storage space are saved per zero.

```
ELSE BEGIN
```

```

STR(energy[R],fullstr);
outstr7 := COPY(fullstr,2,7);
outstr5 := COPY(fullstr,19,5);
outstr := CONCAT(outstr7,outstr5);
WRITELN(store,outstr)

```

- For a value not equal to zero, the number is converted to string, and the middle numbers are removed. There is no loss of accuracy in doing this as the energy is accurate only to the second or third decimal place, and those values are kept. This saves an additional 13 bytes of storage space per number. The result is a file of around 28 kilobytes rather than 45 kilobytes or more.

```

END;
CLOSE(store)

```



END;

{\*\*\*\*\*}

BEGIN {MAIN PROGRAM}

pass\_in\_out(passinfo,device,timeout,ForBias,RevBias,bookkeeper,storage,T);

IF ((passinfo[6] = 'y') OR (passinfo[6] = 'Y')) THEN BEGIN

Calculate(storage,globEnergy);

WriteToEnergy(storage,globEnergy)

END;

IF ((passinfo[2] = 'Y') OR (passinfo[2] = 'y')) THEN BEGIN

- This statement is used to place a pause in execution so the operator can obtain the Base-Emitter Voltage on the 7612D. The pause only executes if the operator has previously indicated in STORDATA that this waveform is to be collected.

WRITELN('Obtain the Base-Emitter Voltage waveform on the 7612D and');

WRITE('press <RETURN> when ready to read it in . . .');

READLN

END;

pass\_in\_out(passinfo,device,timeout,ForBias,RevBias,bookkeeper,storage,'O')

END.

APPENDIX H  
PROCESS3.PAS PROGRAM LISTING

```
PROGRAM Process_Level_Three(INPUT,OUTPUT);
```

- This program reads data from Channel A and writes it out to file "BDxxxLLC.yyy" on the RAM disk. The data is expected to be  $V_{be}$  and is treated as such. The program is written for the MS-PASCAL compiler.

```
{*****}
```

```
($INCLUDE: 'IODECL.EX')
```

- This is the same declaration included in HARDWARE and PROCESS1. See HARDWARE for further information.

```
{*****}
```

```
CONST
```

```
bus = 7;
```

```
TYPE
```

```
STEXT = TEXT(45);
STR13 = STRING(13);
ST10 = STRING(10);
STR9 = STRING(9);
LST80 = LSTRING(80);
setting = ARRAY[1..2,1..10] OF REAL;
```

```
VAR
```

```
bookkeeper,storage : STRING(13);
passinfo : STRING(10);
answer,process : CHAR;
device : INTEGER;
ForBias,RevBias,timeout,Hscale : REAL;
```

```
{*****}
```

```
($INCLUDE: 'IOPROC.EX')
```

- This is the same procedures declaration included in **HARDWARE** and **PROCESS1**. See those programs for further information.

```
{*****}
```

```
PROCEDURE ENDXQQ; EXTERN;
```

- This extended level procedure is used to halt program execution prior to the normal program end. Its function is similar to that of **HALT** in Turbo Pascal.

```
{*****}
```

```
PROCEDURE halter(errmsg : LST80);
```

- This is the same procedure use in all the preceding programs. See **HARDWARE** or **GETDEVIC** for further information.

```
VAR
```

```
name : STRING(12);
```

```
going : TEXT(15);
```

```
BEGIN
```

```
name := 'G:HALTER.PGM';
```

```
ASSIGN(going,name);
```

```
REWRITE(going);
```

```
WRITELN(going,'There was some type of error in Program Process1.');
```

```
WRITELN(going,errmsg);
```

```
WRITELN(going,'PROGRAM HALTED.');
```

```
CLOSE(going);
```

```
ENDXQQ
```

```
END;
```

```
{ ***** }
```

```
PROCEDURE pass_in_out (VAR info : ST10; VAR DEV : INTEGER;
                      VAR timeo,fBias,rBias : REAL;
                      VAR bookkeeper,storage : STR13; IN_OUT : CHAR);
```

- Again, this is the same procedure used in previous programs. See STORDATA for further information.

```
VAR
```

```
  pass : STRING(12);
  passer : STTEXT;
```

```
BEGIN
```

```
  pass := 'G:PASSER.DAT';
  ASSIGN(passer,pass);
  IF IN_OUT = 'T' THEN BEGIN
    RESET(passer);
    READLN(passer,info);
    READLN(passer,DEV);
    READLN(passer,timeo);
    READLN(passer,fBias,rBias);
    READLN(passer,bookkeeper);
    READLN(passer,storage);
    CLOSE(passer)
```

```
  END;
```

```
  IF IN_OUT = 'O' THEN BEGIN
    REWRITE(passer);
    WRITELN(passer,info);
    WRITELN(passer,DEV);
    WRITELN(passer,timeo);
    WRITELN(passer,fBias,rBias);
    WRITELN(passer,bookkeeper);
    WRITELN(passer,storage);
    CLOSE(passer)
```

```
END
END; {End Pass_In_Out}
```

```
{*****}
```

```
PROCEDURE error_handler (error : INTEGER; routine : STR9; A : CHAR);
```

- This is the same procedure used in HARDWARE and PROCESS1. See HARDWARE for further information.

```
VAR
    estring : STRING(40);
```

```
BEGIN
    IF error <> noerr THEN
        BEGIN
            Errstr(error,estring);
            WRITELN('Error in call to ',routine);
            WRITELN(error:6,' ',estring);
            IF A = 'N' THEN
                BEGIN
                    WRITE('Press <RETURN> to continue . . . ');
                    READLN
                END;
            IF A = 'Y' THEN
                BEGIN
                    WRITELN('CORRECT ERROR - Press <RETURN> to continue. ');
                    READLN
                END
            END
        END
    END;
END;
```

```
{*****}
```

```
PROCEDURE setup(tout : REAL);
```

- This is the same procedure used in PROCESS1. See that program for further information.

VAR

cmd : INTEGER;  
endline : STRING(2);

BEGIN

cmd := IOEOI(7,1);  
error\_handler(cmd,'IOEOI ','Y');  
endline[1] := CHR(13);  
endline[2] := CHR(10);  
cmd := IOEOL(7,endline,1);  
cmd := IOREMOTE(7);  
error\_handler(cmd,'IOREMOTE ','Y');  
cmd := IOTIMEOUT(7,tout)

END;

{ \*\*\*\* }

PROCEDURE horscale (VAR Hsc : REAL);

- This procedure is used to obtain the horizontal scale factor from the 7612D.

VAR

response : STRING(25);  
I,cmd : INTEGER;  
HscA,HscB : REAL;  
vsc : STRING(5);

BEGIN

vsc := 'HSFA?';  
cmd := IOOUTPUTS(70200,vsc,5);

- The 7612D is queried for the horizontal scale factor.

```

I := 25;
cmd := IOENTERS(70200,response,I);
    • The characters preceding the desired number are read.
cmd := IOENTER(70200,HscA);
    • The horizontal scale factor is read.
cmd := IOENTERS(70200,response,I);
    • Trailing control characters are striped
vsc := 'HSFB?';
cmd := IOOUTPUTS(70200,vsc,5);
I := 25;
cmd := IOENTERS(70200,response,I);
cmd := IOENTER(70200,HscB);
cmd := IOENTERS(70200,response,I);
    • The process is repeated for Channel B. This is done only to be sure the
      scale factors are the same. Channel B should be unchanged and still
      contain the data from IC. With that data still there, this comparison en-
      sures the data has the same horizontal scale factor.
IF HscA <> HscB THEN BEGIN
    WRITELN('Horizontal Scale Factor are not equal. They must be. ');
    WRITELN('Program Halted. ');
    HALTER('Horizontal Scale Factor are not equal. ')
    • The program is halted if the Channel A and B scale factors are unequal.
END
ELSE Hsc := HscA
END;
```

```
{ ***** }
```

```
PROCEDURE ChannelA(HscA,fBias,rBias : REAL; stor : STR13);
```

- This procedure reads the data from Channel A and places it on the RAM disk. It is identical to the procedure of the same name used in PROCESS1 with the two exceptions. First, the output filename is different (BDxxxLLC.yyy instead of BDxxxLLA.yyy). Second, the first line



used to hold the "Y" or "N" indicator of breakdown in BDxxxLLA.yyy is deleted.

VAR

```
line : STRING(128);
store : STEXT;
state,I,R,C,cmd : INTEGER;
response : STRING(40);
incoming : CHAR;
vsc : STRING(5);
VV,probeA : REAL;
readit : STRING(6);
going : STRING(14);
```

BEGIN

```
probeA := 1;
FOR I := 1 TO 9 DO
  going[I] := stor[I];
going[10] := 'C';
FOR I := 10 TO 13 DO
  going[I+1] := stor[I];
ASSIGN(store,going);
REWRITE(store);
WRITELN(store,fBias,' ,rBias);
vsc := 'VSL1?';
cmd := IOOUTPUTS(70200,vsc,5);
I := 5;
cmd := IOENTERS(70200,vsc,I);
cmd := IOENTER(70200,VV);
I := 40;
cmd := IOENTERS(70200,response,I);
WRITELN(store,VV,' ,HscA,' ,probeA);
readit := 'READ A';
cmd := IOOUTPUTS(70200,readit,6);
```

```

FOR R := 1 TO 3 DO BEGIN
  I := 1;
  cmd := IOENTERS(70200,incoming,I)
END;
state := 0;
cmd := IOMATCH(7,CHR(10),state);
FOR R := 0 TO 15 DO BEGIN
  I := 128;
  cmd := IOENTERS(70200,line,I);
  WRITE(store,line)
END;
CLOSE(store);
state := 1;
cmd := IOMATCH(7,CHR(10),state);
I := 40;
cmd := IOENTERS(70200,response,I);
cmd := IOLOCAL(7)
END;

{ **** }

BEGIN
  pass_in_out(passinfo,device,timeout,ForBias,RevBias,bookkeeper,storage,'I');
  IF ((passinfo[2] = 'Y') OR (passinfo[2] = 'y')) THEN BEGIN
    setup(timeout);
    horscale(Hscale);
    ChannelA(Hscale,ForBias,RevBias,storage)
  END;
  pass_in_out(passinfo,device,timeout,ForBias,RevBias,bookkeeper,storage,'O');
ENDXQQ
END. { End Program }

```

APPENDIX I  
REPEATER.PAS PROGRAM LISTING

PROGRAM repeater;

- This program is used to query the operator as to his or her desire to continue. It, also, is used to update the file, INDEX.DEV, with the storage filename currently kept in the file PASSER.DAT. The test series number may have been incremented during the acquisition cycle, and rather than test for this condition, the name currently held in PASSER.DAT is loaded into the storage filename field in the record of the device under test.

TYPE

STR13 = STRING[13];

VAR

bk,stor : STR13;

DEV : INTEGER;

{\*\*\*\*\*}

PROCEDURE repeatprogram(VAR device : INTEGER; VAR storage : STR13);

- This procedure asks the operator if he or she wishes to take another measurement. If the answer is yes, then the file, CONTINUE.ANS, is created on the RAM disk. The existence of this file is used by the batch file to determine if it should enter into a loop.

VAR

ans : CHAR;

pass : STRING[12];

bookkeeper : STRING[13];

name : STRING[14];

passer,cont: TEXT;

passinfo : STRING[10];

timeout,ForBias,RevBias : REAL;

BEGIN

```
name := 'G:CONTINUE.ANS';
```

```
ASSIGN(cont,name);
```

- The file designator "cont" is assigned the value of "G:CONTINUE.ANS."

```
pass := 'G:PASSER.DAT';
```

```
ASSIGN(passer,pass);
```

```
RESET(passer);
```

```
READLN(passer,passinfo);
```

```
READLN(passer,device);
```

```
READLN(passer,timeout);
```

```
READLN(passer,ForBias,RevBias);
```

```
READLN(passer,bookkeeper);
```

```
READLN(passer,storage);
```

```
CLOSE(passer);
```

- The file, PASSER.DAT, is read.

```
ans := 'Q';
```

```
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO BEGIN
```

```
  WRITE('Do you want to continue taking measurements? (Y or N) . . . ');
```

```
  READLN(ans)
```

```
END;
```

```
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
```

```
  ans := 'Q';
```

```
  WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO BEGIN
```

```
    WRITELN('Repeat measurement on same device? {Reminder: New Inductance  
OR}');
```

```
    WRITE('Temperature = New Device} Same device? (Y or N) . . . ');
```

```
    READLN(ans)
```

- If the measurement to be taken will be on the same device, then the program GETDEVIC will not need to run; therefore, the above question is asked, and depending upon the answer, the first character in the first line of PASSER.DAT will be set so GETDEVIC will or will not execute as needed.

```
END;
```

```
IF ((ans = 'Y') OR (ans = 'y')) THEN BEGIN
```

```

    passinfo[1] := 'N';
    passinfo[7] := 'N'
END;
IF ((ans = 'N') OR (ans = 'n')) THEN BEGIN
    passinfo[1] := 'Y';
    passinfo[7] := 'Y'
END;
REWRITE(cont);
WRITELN(cont,'Continue.');
```

- The file, "CONTINUE.ANS", is created.

```

    CLOSE(cont)
END; { IF ((ans = 'Y'...}
REWRITE(passer);
WRITELN(passer,passinfo);
WRITELN(passer,device);
WRITELN(passer,timeout);
WRITELN(passer,ForBias,RevBias);
WRITELN(passer,bookkeeper);
WRITELN(passer,storage);
CLOSE(passer)
```

- The file, PASSER.DAT, is re-written with the new information.

```
END;
```

```
{ ***** }
```

```
PROCEDURE IndexUpdate (device : INTEGER; store : STR13);
```

- This procedure is used to update the file, INDEX.DEV, with the altered storage filename.

```
TYPE
```

```

    STR40 = STRING[40];
    STR13 = STRING[13];
    DEVICES = RECORD
```

```

DEV : INTEGER;
descrip : STR40;
book : STR13;
stor : STR13
END;

```

```

VAR

```

```

index : FILE OF DEVICES;
indexrec : DEVICES;
error,LineCount : INTEGER;

```

```

BEGIN

```

```

  ASSIGN(index,'C:\RBSOA\INDEX.DEV');

```

```

  {$I-}

```

```

  RESET(index);

```

```

  {$I+}

```

```

  IF IOresult <> 0 THEN BEGIN

```

```

    WRITELN('C:\RBSOA\INDEX.DEV not found. What happened?');

```

```

    WRITELN('Program Halted.')

```

- An error check is made to be certain the file INDEX.DEV is present. If it is not, the program will do nothing further.

```

  END

```

```

  ELSE BEGIN

```

```

    error := FileSize(index);

```

```

    IF device > error THEN BEGIN

```

- If the device number is larger than the number of devices stored in INDEX.DEV, then the program will do nothing further.

```

    WRITELN('FileSize ',error,' device ',device,' Error in update to device.');
```

```

    WRITELN('Program Halted');

```

```

    HALT

```

```

  END;

```

```

  SEEK(index,device);

```

```

  READ(index,indexrec);

```

- If no errors have been encountered, the record for the device under test is retrieved.

WITH indexrec DO

stor := store;

- The storage filename taken from PASSER.DAT replaces the storage filename in the device record.

SEEK(index,device);

WRITE(index,indexrec)

- The updated record is written to INDEX.DEV in direct access mode (no other records are disturbed).

END;

CLOSE(index)

END;

{ \*\*\*\*\* }

BEGIN {Main Program}

repeatprogram(DEV,stor);

IndexUpdate(DEV,stor)

END.



APPENDIX J  
PLOTDATA.PAS PROGRAM LISTING

PROGRAM plotdata2;

- This program is used to create plots on the HP-7470A Graphics Plotter. The waveforms of a single test run are printed in one of two formats. This program requires that the HPIB Peripheral Driver be installed with the plotter bus address (705) set equivalent to the output device "LPT2." The reader should be thoroughly familiar with the HP-7470A User's Manual.

TYPE

STR13 = STRING[13];  
 STR11 = STRING[11];  
 STR9 = STRING[9];  
 datarray = ARRAY[1..3,1..2048] OF INTEGER;

VAR

storage : STR13;  
 ts : STR11;  
 testmain : BOOLEAN;  
 Ignorit,answer,plotall : CHAR;  
 MaximumA,MaximumB,MaximumP,MinimumA,MinimumB,MinimumP,DEV,SER,  
 tI1,tI2,MaxC,MinC,PmaxPoint : INTEGER;  
 ForBias,RevBias,Ener,Hscale,Vscale,Ascale,Bscale,ScaleC : REAL;

{ \*\*\*\*\* }

PROCEDURE NumToStr9 (X : REAL; VAR Y : STR9);

- This procedure is used to prepare real numbers for output as strings. The values are limited to three significant figures in scientific notation format.

VAR

fullstr : STRING[25];  
 one : STRING[1];  
 thr : STRING[3];  
 four : STRING[4];

```
error,I : INTEGER;
num : REAL;
```

```
BEGIN
```

```
STR(X,fullstr);
DELETE(fullstr,1,1);
one := COPY(fullstr,6,1);
```

- The third digit to the right of the decimal place is loaded into the one character long string "one."

```
VAL(one,num,error);
```

- The character in "one" is translated into numeric format and loaded into "num."

```
DELETE(fullstr,6,12);
DELETE(fullstr,8,1);
Y := COPY(fullstr,1,9);
```

- The extra characters in the original number passed to the procedure are deleted. The shortened form of the number is placed in the nine character long string, "Y." The number is shortened from " $\pm x.xxxxxxxxxxxxxxxxxxxE\pm xxx$ " to " $\pm x.xx E\pm xx$ ."

```
IF num >= 5 THEN BEGIN
```

- If the digit in the third place to the right of the decimal point is equal to or over five, the digit to the left is increased by one. This serves to round the new number. The rest of this procedure is involved in performing this function.

```
four := COPY(fullstr,2,4);
VAL(four,num,error);
num := num + 0.01;
IF num = 10.00 THEN BEGIN
```

- This section increases the exponent if it is needed.

```
num := 1.00;
thr := COPY(Y,7,3);
VAL(thr,I,error);
I := I + 1;
STR(I,thr);
```

```

    IF ((I < 10) AND (I > -10)) THEN INSERT('0',thr,2);
    INSERT(thr,Y,7)
END;
STR(num:3:2,four);
DELETE(Y,2,4);
INSERT(four,Y,2)
END
END;

```

```

{ **** }

```

```

PROCEDURE PlotWhat(stor : STR13; VAR plotYorN : CHAR);

```

- This procedure is used to determine what files on drive A: are available to be plotted. It then compares the listing of files available to the user's request for plotting options. If the right files are not available, the user is told and asked to replace the disk in drive A: with one that should have all the necessary files.

```

VAR
  ans : INTEGER;
  ExA,ExB,ExC,ExD,ExE,ExP : CHAR;
  going : STRING[14];
  test : BOOLEAN;
  store : TEXT;

```

```

BEGIN

```

```

  WRITELN('Place the disk containing the data to be plotted in drive A:');
  WRITE('Press <RETURN> to continue . . . ');
  READLN;
  test := TRUE;
  WHILE test DO BEGIN

```

- This WHILE loop is used to look for available files and compare those files with the ones necessary to fulfill the operator's plotting request. If

all the files are present, the loop ends; otherwise, it continues until the files are found or the operator quits.

```
going := stor;
INSERT('A',going,10);
ASSIGN(store,going);
{$I-}
RESET(store);
IF IOresult <> 0 THEN ExA := 'N'
ELSE ExA := 'Y';
CLOSE(store);
```

- The last seven lines (excluding "{\$I-}") are used to check drive A: for BDxxxLLA.yyy. If the file is present the variable "ExA" is set to "Y." If the file is not present, "ExA" becomes "N."

```
going := stor;
INSERT('B',going,10);
ASSIGN(store,going);
RESET(store);
IF IOresult <> 0 THEN ExB := 'N'
ELSE ExB := 'Y';
CLOSE(store);
```

- A repeat of the previous check is done for BDxxxLLB.yyy.

```
going := stor;
INSERT('C',going,10);
ASSIGN(store,going);
RESET(store);
IF IOresult <> 0 THEN ExC := 'N'
ELSE ExC := 'Y';
CLOSE(store);
```

- Again, for BDxxxLLC.yyy.

```
going := stor;
INSERT('E',going,10);
ASSIGN(store,going);
RESET(store);
IF IOresult <> 0 THEN ExE := 'N'
```

```
ELSE ExE := 'Y';
CLOSE(store);
    • BDxxxLLE.yyy.
```

```
going := stor;
INSERT('P',going,10);
ASSIGN(store,going);
RESET(store);
IF IOresult = 0 THEN ExP := 'Y'
ELSE ExP := 'N';
CLOSE(store);
```

- Lastly, for BDxxxLLP.yyy.

```
{ $I+ }
```

```
ans := 3;
WHILE NOT ((ans = 0) OR (ans = 1) OR (ans = 2)) DO BEGIN
```

- This WHILE loop is part of the first loop. It is used to ask the operator for the particular plotting option required. It is primarily an "idiot" proofing loop.

```
CLRSCR;
WRITELN('With this program, you can plot in one of two different formats. ');
WRITELN('First, you can plot Collector-Emitter Voltage, Collector Current, ');
WRITELN('Instantaneous Power and Cumulative Energy up to the point of second ');
WRITELN('breakdown. Second, all of the above plus Base-Emitter Voltage can ');
WRITELN('also be plotted. ');
WRITELN;WRITELN;
WRITE('Enter 1 for the first case, 2 for the second and 0 to quit . . . ');
READLN(ans)
```

```
END;
```

```
IF ans = 0 THEN HALT;
```

```
IF ans = 1 THEN BEGIN
```

```
IF ((ExA = 'Y') AND (ExB = 'Y') AND (ExE = 'Y') AND (ExP = 'Y')) THEN
BEGIN
```

```
PlotYorN := 'N';
```

```
test := FALSE
```

- If the operator chooses option 1, then the files for  $V_{ce}$ ,  $I_C$ , E and P must be present (BDxxxLL{A,B,E,P}.yyy). If they are present, the main program is informed of the selection by "PlotYorN" and the first WHILE loop is ended.

END

ELSE BEGIN

WRITELN('Not all of the files need are on the disk in drive A:');

WRITELN('Replace the disk in drive A: with the correct one.');

WRITE('Press <RETURN> to continue . . .');

READLN

- If not all the files are present, the above message is printed, and the loop will be executed again from the beginning.

END

END;

IF ans = 2 THEN BEGIN

IF ((ExA = 'Y') AND (ExB = 'Y') AND (ExE = 'Y') AND (ExP = 'Y') AND  
(ExC = 'Y')) THEN BEGIN

PlotYorN := 'Y';

test := FALSE

- If option 2 is selected and all the files are present ( $V_{ce}$ ,  $I_C$ , E, P and  $V_{be}$ ), the main program is informed by "PlotYorN" and the loop is ended.

END

ELSE BEGIN

WRITELN('Not all of the files need are on the disk in drive A:');

WRITELN('Replace the disk in drive A: with the correct one.');

WRITE('Press <RETURN> to continue . . .');

READLN

- If all the files are not present, the above is printed to the screen, and the loop is executed, again.

END

END

END

END;

```
{ ***** }
```

```
PROCEDURE listdevice(VAR stor : STR13; VAR device,series : INTEGER);
```

- This procedure is common with some minor variation to all of the ancillary programs except EDITLIST. The procedure is responsible for listing the device records in INDEX.DEV upon request, querying for device and test series numbers which describes the particular test run to be plotted and returning the storage filename modified to contain the series number for the desired test.

```
TYPE
```

```
  STR40 = STRING[40];
  STR13 = STRING[13];
  DEVICES = RECORD
    DEV : INTEGER;
    descrip : STR40;
    book : STR13;
    store : STR13
  END;
```

```
VAR
```

```
  index : FILE OF DEVICES;
  indexrec : DEVICES;
  error,LineCount : INTEGER;
  name : STRING[18];
  one : STRING[1];
  two : STRING[2];
  thr : STRING[3];
  test : BOOLEAN;
  ans : CHAR;
  I,Count : INTEGER;
```

```
BEGIN
```



```

name := 'C:\RBSOAN\INDEX.DEV';
ASSIGN(index,name);
RESET(index);
WRITELN('You will be asked for the device number corresponding to the storage file');
WRITELN('for the data taken on the device of interest. You will also be asked for');
WRITELN('the storage series number which is the last three numbers in the file name. ');
WRITELN('The value shown in the listing is the highest currently stored. ');
ans := 'q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO
  BEGIN
    WRITELN('Do you want the see a listing of all currently cataloged devices?');
    WRITE('See list? (Y or N) . . . ');
    READLN(ans)
  END; { WHILE NOT ((ans = 'y'...}
IF ((ans = 'Y') OR (ans = 'y')) THEN BEGIN
  • If the operator wants to see a list of the devices in INDEX.DEV, this
    section will do it.
  LineCount := 1;
  Count := 0;
  SEEK(index,LineCount);
  WHILE NOT EOF(index) DO BEGIN
    READ(index,indexrec);
    • The record at the cursor location defined by "LineCount" is read.
  WITH indexrec DO BEGIN
    WRITE(DEV,' ');
    WRITE(descrip:40,' ');
    WRITE(book,' ');
    WRITELN(store)
    • The fields in the device record are printed on the screen.
  END;
  LineCount := LineCount + 1;
  • The cursor location pointer is increased by one.
  Count := Count + 1;

```

- "Count" is used to keep track of the number of records displayed on the screen.

```
IF Count = 21 THEN BEGIN
```

- When "Count" reaches 21, 21 lines are displayed on the screen. At this point, a pause is executed to give the operator a chance to review the records, and the "Count" is set to zero.

```
Count := 0;
```

```
WRITELN('Press <RETURN> for more devices.');
```

```
READLN
```

```
END
```

```
END;
```

```
WRITELN('End of indexed devices.')
```

```
END; {IF ((ans = 'Y'...}
```

```
test := TRUE;
```

```
WHILE test DO BEGIN
```

- This loop is designed as a type of "idiot" proofing. It prevents a device number which is too large from being accepted. It could have been done differently and more efficiently, but wasn't. Any upgrades might look at this.

```
WRITE('What is the device number? (Integer Value Only!) . . .');
```

```
READLN(device);
```

```
IF device < 1000 THEN test := FALSE
```

```
ELSE WRITELN('Integer too large. Try again.')
```

```
END;
```

```
test := TRUE;
```

```
WHILE test DO BEGIN
```

- "Idiot" proofing for the series number.

```
WRITE('What is the series number? (Integer Value Only!) . . .');
```

```
READLN(series);
```

```
IF series < 1000 THEN test := FALSE
```

```
ELSE WRITELN('Integer too large. Try again.')
```

```
END;
```

```
SEEK(index,device);
```

```
READ(index,indexrec);
```

- The specific device record is retrieved.

WITH indexrec DO

stor := store;

- The storage filename is placed in "stor."

CLOSE(index);

DELETE(stor,1,1);

INSERT('A',stor,1);

DELETE(stor,11,3);

INSERT('000',stor,11);

- The drive designation on the storage filename (previously G:) is changed to A:, and the series number is deleted and replaced with "000." The series number will be replaced with the one selected by the operator earlier in the lines below.

IF series < 10 THEN BEGIN

STR(series:1,one);

INSERT(one,stor,13)

- If the series number is less than 10 (one digit), it is transformed into a one character string and inserted into the last position of the storage filename.

END;

IF ((series > 9) AND (series < 100)) THEN BEGIN

STR(series:2,two);

INSERT(two,stor,12)

- Repeat of the above, but for the case of a two digit series number.

END;

IF ((series > 99) AND (series < 1000)) THEN BEGIN

STR(series:3,thr);

INSERT(thr,stor,11)

- For the case of a three digit series number.

END;

DELETE(stor,1,1);

INSERT('A',stor,1);

END; {Procedure listdevice}

```

VAR MaxA,MaxB,MaxP,MinA,MinB,MinP : INTEGER;
VAR HscA,VscP,ScaleA,ScaleB,energy,fBiasA,rBiasA : REAL;
VAR Ign : CHAR);

```

- This procedure plots the waveforms for  $V_{ce}$ ,  $I_c$  and  $P$ .  $V_{ce}$  and  $I_c$  will be superimposed in the upper left plot.

```

VAR
  R,I,S : INTEGER;
  going : STRING[14];
  incoming : CHAR;
  VIP : datarray;
  VI,VV,HscB,HscP,probeA,probeB,fBiasB,rBiasB,fBiasP,
  rBiasP,ZeroA,ZeroB,ZeroP,
  numouty,numoutx,DA,DB,DP : REAL;
  store,LPT2 : TEXT;

```

```

BEGIN

```

```

  ASSIGN(LPT2,'LPT2');
  REWRITE(LPT2);
  going := stor;
  INSERT('A',going,10);
  ASSIGN(store,going);
  incoming := 'Q';

```

```

{$I-}

```

```

  RESET(store);

```

- The file containing  $V_{ce}$  is opened for reading here. If the file is not present, the operator is given the opportunity to fix the problem.

```

  WHILE Ioresult <> 0 DO BEGIN

```

```

    WRITELN('There has been an error accessing file ',going);
    WRITELN('Please check to be sure the proper disk is in drive A:');
    WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
    READLN(incoming);
    IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
    IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT

```

```

END;
{$I+}
READLN(store,Ign);
READLN(store,fBiasA,rBiasA);
READLN(store,VV,HscA,probeA);

```

- The first three line of the file are read. Notice that the first character of the first line is placed in "Ign." The breakdown "Y" or "N" character is now in "Ign" which is passed to the main program. The forward and reverse bias settings are read from line two, and the Volts/Div, horizontal scale factor and probe scaling factors are read from line three.

```
ScaleA := VV*probeA/32;
```

- The vertical scale factor for  $V_{ce}$  is calculated.

```
FOR R := 1 TO 2048 DO BEGIN
```

```
  READ(store,incoming);
```

```
  VIP[1,R] := ORD(incoming)
```

- The data points are then read, but they are not corrected for the zero offset factor of 127. It is not needed in this application.

```
END;
```

```
CLOSE(store);
```

```
going := stor;
```

```
INSERT('B',going,10);
```

```
ASSIGN(store,going);
```

- A repeat of the above reading process is done for the file containing  $I_C$ .

The only difference is that the first line of this file corresponds to the second line of the preceding file.

```
{$I-}
```

```
RESET(store);
```

```
WHILE Ioresult <> 0 DO BEGIN
```

```
  WRITELN('There has been an error accessing file ',going);
```

```
  WRITELN('Please check to be sure the proper disk is in drive A:');

```

```
  WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
```

```
  READLN(incoming);
```

```
  IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
```

```
  IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT
```

```

END;
{$I+}
READLN(store,fBiasB,rBiasB);
READLN(store,VI,HscB,probeB);
ScaleB := VI*probeB/32;
IF HscA <> HscB THEN BEGIN
  WRITELN('Horizontal scale factors not equal. Program Halted.');
```

- The horizontal scale factors are compared. If they are not equal the program will halt and the data cannot be plotted.

```

END;
FOR R := 1 TO 2048 DO BEGIN
  READ(store,incoming);
  VIP[2,R] := ORD(incoming)
  • Ic is read.
```

```

END;
CLOSE(store);
going := stor;
INSERT('P',going,10);
ASSIGN(store,going);
```

- The read process is done one more time for the file containing P.

```

{$I-}
RESET(store);
WHILE Ioresult <> 0 DO BEGIN
  WRITELN('There has been an error accessing file ',going);
  WRITELN('Please check to be sure the proper disk is in drive A:');
  WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
  READLN(incoming);
  IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
  IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT
END;
```

```

{$I+}
READLN(store,fBiasP,rBiasP);
READLN(store,energy,HscP,VscP);
```

- The forward and reverse bias settings are read from line one, and the total energy during breakdown, the horizontal scale factor and the vertical scale factor are read from line two.

FOR R := 1 TO 2048 DO

  READ(store,VIP[3,R]);

- The Instantaneous Power is read.

CLOSE(store);

IF ((fBiasP <> fBiasA) OR (fBiasP <> fBiasB) OR (fBiasA <> fBiasB)) THEN BEGIN

  WRITELN('Error in reading in files. Forward Bias Settings not equal.');

  WRITELN('Program Halted.');

  HALT

- A check is made to be sure all the forward bias settings read from the file are the same; otherwise, the program stops and the data cannot be plotted.

END;

IF ((rBiasP <> rBiasA) OR (rBiasP <> rBiasB) OR (rBiasA <> rBiasB)) THEN BEGIN

  WRITELN('Error in reading in files. Reverse Bias Settings not equal.');

  WRITELN('Program Halted.');

  HALT

- Same as last check only for the reverse bias settings.

END;

IF ((HscA <> HscB) OR (HscA <> HscP)) THEN BEGIN

  WRITELN('Error in reading in files. Horizontal Scale Settings not equal.');

  WRITELN('Program Halted.');

  HALT

- The horizontal scale factors are compared. If they are not the same the program halts.

END;

MaxA := 0;

MaxB := 0;

MaxP := -32766;

MinA := 256;

MinB := 256;

MinP := 32767;

FOR R := 1 TO 2048 DO BEGIN

IF (MaxA < VIP[1,R]) THEN MaxA := VIP[1,R];

IF (MaxB < VIP[2,R]) THEN MaxB := VIP[2,R];

IF (MaxP < VIP[3,R]) THEN MaxP := VIP[3,R];

IF (MinA > VIP[1,R]) THEN MinA := VIP[1,R];

IF (MinB > VIP[2,R]) THEN MinB := VIP[2,R];

IF (MinP > VIP[3,R]) THEN MinP := VIP[3,R]

- The maximum and minimum values of  $V_{ce}$ ,  $I_c$  and P are found.

END;

DA := MaxA - MinA;

DB := MaxB - MinB;

DP := MaxP - MinP;

- The difference between the maximum and minimum value for each data set is calculated. This is used in producing a scaled plot which is as large as the space allows.

FOR R := 1 TO 2048 DO BEGIN

VIP[1,R] := VIP[1,R] - MinA;

VIP[2,R] := VIP[2,R] - MinB;

VIP[3,R] := VIP[3,R] - MinP

- The minimum values of  $V_{ce}$ ,  $I_c$  and P are now zero. All data points have been corrected for the new offset.

END;

DA := 2000/DA;

DB := 2000/DB;

DP := 2000/DP;

- The maximum vertical deflection of the plotted output is 2000 plotter units. "DA", "DB" and "DP" are now scale factors which when multiplied by the offset data points will convert the integer data point values to plotter units.

ZeroA := ABS((127-MinA)\*DA);

ZeroB := ABS((127-MinB)\*DB);

ZeroP := ABS(MinP\*DP);

- The above three lines calculate the zero crossing location of  $V_{ce}$ ,  $I_c$  and P.



IF MinA < 127 THEN BEGIN

WRITELN(LPT2,'LT 2,1');

numouty:= 4400+ZeroA;

WRITELN(LPT2,'PA 1240,',numouty:0:4);

WRITELN(LPT2,'PD 4800,',numouty:0:4)

- If the minimum value of  $V_{ce}$  is below 127 ( $\approx 0$ ), then the data goes negative and the zero crossing dashed line created by the above four lines is produced on the plot.

END;

IF MinB < 127 THEN BEGIN

WRITELN(LPT2,'LT 1,1');

numouty:= 4400+ZeroB;

WRITELN(LPT2,'PU 1240,',numouty:0:4);

WRITELN(LPT2,'PD 4800,',numouty:0:4)

- The same is done for  $I_C$ .

END;

IF MinP < 0 THEN BEGIN

WRITELN(LPT2,'LT 2,1');

numouty:= 1410+ZeroP;

WRITELN(LPT2,'PU 1360,',numouty:0:4);

WRITELN(LPT2,'PD 4800,',numouty:0:4)

- Again, for P.

END;

WRITELN(LPT2,'LT');

numouty := 4400+(VIP[1,1]\*DA);

- The value "numouty" is the vertical plot coordinate. The 4400 offset value defines where the plot is placed vertically on the sheet. Vertical placement is defined as the placement along the axis which stretches across the most narrow part of the paper or from long edge to long edge.

WRITELN(LPT2,'PU 1290,',numouty:0:4);

- The horizontal deflection will be described by "numoutx." The line above gives the starting point in the x-direction as 1290.

FOR I := 2 TO 513 DO BEGIN

```

R := (I-1)*4;
numoutx := 1290+(R-1)*1.7;
numouty := 4400+(VIP[1,R]*DA);
    • The location of the next point to be plotted are calculated in plotter units.
WRITELN(LPT2,'PD ',numoutx:0:4,',',numouty:0:4)
    • The data for  $V_{ce}$  is plotted. Only every fourth point are plotted to save
      time in plotting. The accuracy of the plot does not suffer as there are
      still over 100 data points per inch in the horizontal direction.
END;
numoutx := 1290+1024*1.7;
numouty := 4400+VIP[1,1024]*DA;
    • The location for the graph label for  $V_{ce}$  is calculated.
WRITELN(LPT2,'PU');
WRITELN(LPT2,'PA ',numoutx:0:4,',',numouty:0:4);
    • The plotter pen is moved to the location.
WRITELN(LPT2,'SI .12,.21');
IF numouty > 4600 THEN BEGIN
    WRITELN(LPT2,'CP 0,-.9');
    WRITELN(LPT2,'LBV')
END
ELSE BEGIN
    WRITELN(LPT2,'CP 0,+.9');
    WRITELN(LPT2,'LBV')
END;
    • Depending upon the value of the data point at the location the label, "V,"
      will be printed above or below the plot just made.
numouty := 4400+(VIP[2,1]*DB);
WRITELN(LPT2,'PU 1290,',numouty:0:4);
    • A repeat of the plotting process is done for  $I_c$ . Notice the offsets are
      1290 and 4400. This will superimpose the graphs of  $V_{ce}$  and  $I_c$ .
FOR I := 2 TO 513 DO BEGIN
    R := (I-1)*4;
    numoutx := 1290+(R-1)*1.7;
    numouty := 4400+VIP[2,R]*DB;

```

```

WRITELN(LPT2,'PD ',numoutx:0:4,',',numouty:0:4)
END;

```

- A repeat of the labeling process is done below, but the horizontal placement is moved from point 256 to 1024, and the label is "I."

```

numoutx := 1290+1024*1.7;
numouty := 4400+VIP[2,1024]*DB;
WRITELN(LPT2,'PU ',numoutx:0:4,',',numouty:0:4);
IF numouty > 4600 THEN BEGIN
  WRITELN(LPT2,'CP 0,-1.1');
  WRITELN(LPT2,'LBI')
END
ELSE BEGIN
  WRITELN(LPT2,'CP 0,1.1');
  WRITELN(LPT2,'LBI')
END;

```

- The last plot made in this procedure is P. It is made the same way the plots above were done. It, also, is labeled as the other plots were.

```

numouty := 1410+VIP[3,1]*DP;
WRITELN(LPT2,'PU 1290,',numouty:0:4);
FOR I := 2 TO 513 DO BEGIN
  R := (I-1)*4;
  numoutx := 1290+(R-1)*1.7;
  numouty := 1410+VIP[3,R]*DP;
  WRITELN(LPT2,'PD ',numoutx:0:4,',',numouty:0:4)
END;
numoutx := 1290+256*1.7;
numouty := 1410+VIP[3,256]*DP;
WRITELN(LPT2,'PU ',numoutx:0:4,',',numouty:0:4);
IF numouty > 1610 THEN BEGIN
  WRITELN(LPT2,'CP 0,-1.1');
  WRITELN(LPT2,'LBP')
END
ELSE BEGIN
  WRITELN(LPT2,'CP 0,1.1');

```

```

    WRITELN(LPT2,'LBP')
END;
WRITELN(LPT2,'SI');
WRITELN(LPT2,'PU');
CLOSE(LPT2)
END;

```

```

{ ***** }

```

```

PROCEDURE plot2 (stor : STR13; VAR MaxA,MinA : INTEGER;

```

```

    VAR HscA,ScaleA : REAL);

```

- This procedure is used to plot  $V_{be}$  in the upper right quadrant of the output. It is created exactly as the plots in "Plot1" were, except no label is plotted.

```

TYPE

```

```

    simplearr = ARRAY[1..2048] OF INTEGER;

```

```

VAR

```

```

    R,I,S : INTEGER;

```

```

    going : STRING[14];

```

```

    incoming : CHAR;

```

```

    VIP : simplearr;

```

```

    VV,probeA,ZeroA,fBiasA,rBiasA,numouty,numoutx,DA : REAL;

```

```

    store,LPT2 : TEXT;

```

```

BEGIN

```

```

    ASSIGN(LPT2,'LPT2');

```

```

    REWRITE(LPT2);

```

```

    going := stor;

```

```

    INSERT('C',going,10);

```

```

    ASSIGN(store,going);

```

```

    incoming := 'Q';

```

```

{$I-}

```

```

    RESET(store);

```

```

WHILE Ioresult <> 0 DO BEGIN
  WRITELN('There has been an error accessing file ',going);
  WRITELN('Please check to be sure the proper disk is in drive A:');
  WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
  READLN(incoming);
  IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
  IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT
END;

```

{ \$I+ }

- The file containing  $V_{be}$  is found and opened.

```

READLN(store,fBiasA,rBiasA);
READLN(store,VV,HscA,probeA);

```

- The first two lines are read. Forward bias and reverse bias settings from line one, and Volts/Div, horizontal scale factor and probe multiplication factor from line two.

```

ScaleA := VV*probeA/32;
FOR R := 1 TO 2048 DO BEGIN
  READ(store,incoming);
  VIP[R] := ORD(incoming)
END;

```

- The data points are read.

```

CLOSE(store);
MaxA := 0;
MinA := 256;
FOR R := 1 TO 2048 DO BEGIN
  IF (MaxA < VIP[R]) THEN MaxA := VIP[R];
  IF (MinA > VIP[R]) THEN MinA := VIP[R];
END;

```

- The minimum and maximums are calculated.

```
DA := MaxA - MinA;
```

- The difference between the minimum and maximum is calculated.

```

FOR R := 1 TO 2048 DO BEGIN
  VIP[R] := VIP[R] - MinA;
END;

```

- The data points are shifted so the minimum value in the array is zero.

DA := 2000/DA;

- "DA" becomes the conversion factor from integer values to plotter units.

IF MinA < 127 THEN BEGIN

ZeroA := ABS((127-MinA)\*DA);

WRITELN(LPT2,'LT 2,1');

numouty:= 4400+ZeroA;

WRITELN(LPT2,'PA 5780,',numouty:0:4);

WRITELN(LPT2,'PD 9340,',numouty:0:4);

WRITELN(LPT2,'LT')

END;

- If the plot goes negative the zero crossing location is marked by a dashed line.

numouty := 4400+(VIP[1]\*DA);

WRITELN(LPT2,'PU 5830,',numouty:0:4);

FOR I := 2 TO 513 DO BEGIN

R := (I-1)\*4;

numoutx := 5830+(R-1)\*1.7;

numouty := 4400+(VIP[R]\*DA);

WRITELN(LPT2,'PD ',numoutx:0:4,',',numouty:0:4)

END;

- Every fourth data point is plotted. Notice the offsets, 5830 and 4400. This places the plot in the upper right quadrant.

WRITELN(LPT2,'PU');

CLOSE(LPT2)

END;

{ \*\*\*\*\* }

PROCEDURE LabelChoice2;

- This procedure is used to create the text labels for the axes, the zero crossing line key and the graph titles for plotting option 2 made in "PlotWhat." The procedure is simply a series of WRITELN statements. To understand what is happening in this procedure the reader

should refer to the HP-7470A User's Manual. No further comments are included for this procedure.

VAR

LPT2 : TEXT;

BEGIN

```
  ASSIGN(LPT2,'LPT2');
  REWRITE(LPT2);
  WRITELN(LPT2,'SI .27,.375');
  WRITELN(LPT2,'PU 2790,6600');
  WRITELN(LPT2,'CP -5,0');
  WRITELN(LPT2,'LBV and I');
  WRITELN(LPT2,'PU 2790,6600');
  WRITELN(LPT2,'CP -4,-.3');
  WRITELN(LPT2,'LBCE C');
  WRITELN(LPT2,'PA 7330,6600');
  WRITELN(LPT2,'CP -10,0');
  WRITELN(LPT2,'LBBase-Emitter Voltage');
  WRITELN(LPT2,'PA 2790,3610');
  WRITELN(LPT2,'CP -4.5,0');
  WRITELN(LPT2,'LBP and E');
  WRITELN(LPT2,'PA 2790,3610');
  WRITELN(LPT2,'CP -3.5,-.3');
  WRITELN(LPT2,'LBI C');
  WRITELN(LPT2,'SI');
  WRITELN(LPT2,'DR 0,1');
  WRITELN(LPT2,'PA 5330,5450');
  WRITELN(LPT2,'CP -2.5,-1.2');
  WRITELN(LPT2,'LBVolts');
  WRITELN(LPT2,'PA 740,5450');
  WRITELN(LPT2,'CP -2.5,0');
  WRITELN(LPT2,'LBVolts');
  WRITELN(LPT2,'PA 740,5450');
```

```

WRITELN(LPT2,'CP -3.5,-1.2');
WRITELN(LPT2,'LBAmperes');
WRITELN(LPT2,'PA 740,2360');
WRITELN(LPT2,'CP -2.5,0');
WRITELN(LPT2,'LBWatts');
WRITELN(LPT2,'PA 740,2360');
WRITELN(LPT2,'CP -3,-1.2');
WRITELN(LPT2,'LBJoules');
WRITELN(LPT2,'DR 1,0');
WRITELN(LPT2,'PU 5500,1200');
WRITELN(LPT2,'LT 2,1');
WRITELN(LPT2,'PD 6800,1200');
WRITELN(LPT2,'LT 1,1');
WRITELN(LPT2,'PU 7050,1200');
WRITELN(LPT2,'LBZero Crossing V and P');
WRITELN(LPT2,'PU 5500,950');
WRITELN(LPT2,'PD 6800,950');
WRITELN(LPT2,'PU 7050,950');
WRITELN(LPT2,'LBZero Crossing Current');
WRITELN(LPT2,'LT');
CLOSE(LPT2)
END;

```

```
{ ***** }
```

```
PROCEDURE VerticalAxis1 (MaxA,MaxB,MaxP,MinA,MinB,MinP : INTEGER;
```

```
    VscP,ScaleA,ScaleB,energy,fBias,rBias : REAL);
```

- The vertical axes scales for  $V_{ce}$ ,  $I_c$  and  $P$  are calculated and plotted for plot option two with this procedure.

```
VAR
```

```
    fullstr : STRING[27];
```

```
    outstr : STRING[9];
```

```
    MinPlotA,MinPlotB,MinPlotP,MaxPlotA,MaxPlotB,MaxPlotP : REAL;
```



LPT2 : TEXT;

BEGIN

  ASSIGN(LPT2,'LPT2');

  REWRITE(LPT2);

  MinPlotA := ScaleA\*(MinA - 127);

  MinPlotB := ScaleB\*(MinB - 127);

  MinPlotP := VscP\*MinP;

- The minimum value of  $V_{ce}$  in volts,  $I_C$  in Amperes and P in Watts is calculated.

  MaxPlotA := ScaleA\*(MaxA - 127);

  MaxPlotB := ScaleB\*(MaxB - 127);

  MaxPlotP := VscP\*MaxP;

- The maximum value of  $V_{ce}$  in volts,  $I_C$  in Amperes and P in Wats is calculated.

  WRITELN(LPT2,'SI .12,.21');

  WRITELN(LPT2,'PU 750,6400');

  WRITELN(LPT2,'CP -3,+.5');

  NumToStr9(MaxPlotA,outstr);

  WRITELN(LPT2,'LB',outstr,'V');

- The plotter pen is moved to the location where the maximum value of  $V_{ce}$  is to be written. Depending upon the magnitude of the value the pen is moved a certain number of characters to the left to keep a constant right justification with the rest of the numbers to be printed below.

  WRITELN(LPT2,'PU 750,4400');

  WRITELN(LPT2,'CP -3,+.5');

  NumToStr9(MinPlotA,outstr);

  WRITELN(LPT2,'LB',outstr,'V');

- The minimum value of  $V_{ce}$  is printed. It is also right justified.

  WRITELN(LPT2,'PU 750,6400');

  WRITELN(LPT2,'CP -3,-.5');

  NumToStr9(MaxPlotB,outstr);

  WRITELN(LPT2,'LB',outstr,'A');

- The maximum value of  $I_C$  is printed (right justified).

```
WRITELN(LPT2,'PU 750,4400');
```

```
WRITELN(LPT2,'CP -3,-.5');
```

```
NumToStr9(MinPlotB,outstr);
```

```
WRITELN(LPT2,'LB',outstr,'A');
```

- The minimum value of  $I_C$  is printed (right justified).

```
WRITELN(LPT2,'PU 750,3410');
```

```
WRITELN(LPT2,'CP -3,.5');
```

```
NumToStr9(MaxPlotP,outstr);
```

```
WRITELN(LPT2,'LB',outstr,'W');
```

- The maximum value of P is printed (right justified).

```
WRITELN(LPT2,'PU 750,3410');
```

```
WRITELN(LPT2,'CP -3,-.5');
```

```
NumToStr9(energy,outstr);
```

```
WRITELN(LPT2,'LB',outstr,'J');
```

- The maximum value of the energy is printed (right justified). The energy was obtained when the file containing P was read. The total energy is contain on line two of that file, and was passed to this procedure. Since the total energy is the maximum energy, it is possible to use that value to create the scale label. Also, the power and energy plots are superimposed in both of the plot options so the scale value for the energy is written just below that for the power.

```
WRITELN(LPT2,'PU 750,1410');
```

```
WRITELN(LPT2,'CP -3,-.5');
```

```
NumToStr9(0,outstr);
```

```
WRITELN(LPT2,'LB',outstr,'J');
```

- The minimum value of the energy is printed.

```
WRITELN(LPT2,'PU 750,1410');
```

```
WRITELN(LPT2,'CP -3,.5');
```

```
NumToStr9(MinPlotP,outstr);
```

```
WRITELN(LPT2,'LB',outstr,'W');
```

- The minimum value of P is printed (right justified).

```
WRITELN(LPT2,'SI');
```

```
WRITELN(LPT2,'PU 7585,3100');
```

```
WRITELN(LPT2,'CP -11,0');
```

```
NumToStr9(energy,outstr);
```

```
WRITELN(LPT2,'LB ',outstr,'J');
```

- The value of the total energy is printed in the lower right quadrant in the "Breakdown Values" section.

```
WRITELN(LPT2,'PU 7585,1900');
```

```
WRITELN(LPT2,'CP -12,0');
```

```
NumToStr9(fBias,outstr);
```

```
WRITELN(LPT2,'LB ',outstr,'A');
```

```
WRITELN(LPT2,'PU 7585,1900');
```

```
WRITELN(LPT2,'CP 6.2,0');
```

```
NumToStr9(rBias,outstr);
```

```
WRITELN(LPT2,'LB ',outstr,'A');
```

- The forward and reverse bias settings are also printed over in the lower right quadrant.

```
CLOSE(LPT2)
```

```
END;
```

```
{*****}
```

```
PROCEDURE VerticalAxis2 (MaxA,MinA : INTEGER; ScaleA : REAL);
```

- This procedure is used to print the vertical axis scale for  $V_{be}$ .

```
VAR
```

```
fullstr : STRING[27];
```

```
outstr : STRING[9];
```

```
MinPlotA,MaxPlotA : REAL;
```

```
LPT2 : TEXT;
```

```
BEGIN
```

```
ASSIGN(LPT2,'LPT2');
```

```
REWRITE(LPT2);
```

```
MinPlotA := ScaleA*(MinA - 127);
```

```
MaxPlotA := ScaleA*(MaxA - 127);
```

- The maximum and minimum values of  $V_{be}$  are calculated.

```

WRITELN(LPT2,'SI .12,.21');
WRITELN(LPT2,'PU 5290,6400');
WRITELN(LPT2,'CP -2,0');
NumToStr9(MaxPlotA,outstr);
WRITELN(LPT2,'LB',outstr);
    • The maximum value of  $V_{be}$  is printed (right justified).
WRITELN(LPT2,'PU 5290,4400');
WRITELN(LPT2,'CP -2,-.5');
NumToStr9(MinPlotA,outstr);
WRITELN(LPT2,'LB',outstr);
    • The minimum value of  $V_{be}$  is printed (right justified).
CLOSE(LPT2)
END;

```

```
{*****}
```

```
PROCEDURE EnergyPlot(stor : STR13; Hsc : REAL; VAR PmaxPos : INTEGER);
```

- This procedure plots the Cumulative Energy superimposed on the Instantaneous Power.

```
TYPE
```

```
ene = ARRAY[1..2047] OF REAL;
```

```
VAR
```

```
store,LPT2 : TEXT;
```

```
energy : ene;
```

```
I,R,Estart,Estop : INTEGER;
```

```
DE,Emax,X,Y,t : REAL;
```

```
test : BOOLEAN;
```

```
going : STRING[14];
```

```
incoming : CHAR;
```

```
fullstr : STRING[25];
```

```
time : STRING[10];
```

```

BEGIN
  ASSIGN(LPT2,'LPT2');
  REWRITE(LPT2);
  going := stor;
  INSERT('E',going,10);
  ASSIGN(store,going);
  incoming := 'Q';
{$I-}
  RESET(store);
  WHILE Ioresult <> 0 DO BEGIN
    WRITELN('There has been an error accessing file ',going);
    WRITELN('Please check to be sure the proper disk is in drive A:');
    WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
    READLN(incoming);
    IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
    IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT
  END;
{$I+}

```

- The file containing the energy is found and opened.

```

test := TRUE;
FOR R := 1 TO 2047 DO
  READ(store,energy[R]);

```

- The cumulative energy values are read.

```

CLOSE(store);
FOR R := 1 TO 2047 DO
  IF test THEN
    IF energy[R] > 0 THEN BEGIN
      test := FALSE;
      Estart := R
    END;

```

- The point where the energy becomes non-zero is found. This is where the plot will begin.

```

Estart := Estart - 1;
Emax := -100000.0;

```

```
FOR R := 1 TO 2047 DO
```

```
  IF energy[R] > Emax THEN Emax := energy[R];
```

- The maximum value of the energy is found.

```
test := TRUE;
```

```
FOR R := 1 TO 2047 DO
```

```
  IF test THEN
```

```
    IF energy[R] = Emax THEN BEGIN
```

```
      test := FALSE;
```

```
      Estop := R
```

```
    END;
```

- The first occurrence (and theoretically the only) of the maximum value of E is found. This will be the end of the energy plot.

```
Estop := Estop + 1;
```

```
PmaxPos := Estop - 2;
```

```
DE := 2000/Emax;
```

- The vertical conversion factor from real numbers into plotting units is calculated.

```
X := 1.7*(Estart-1) + 1290;
```

```
Y := 1410;
```

- The start point for the energy plot is calculated in plotter units.

```
WRITELN(LPT2,'PU ',X:0:4,',',Y:0:4);
```

```
R := 0;
```

```
FOR I := Estart TO Estop DO BEGIN
```

```
  R := R + 1;
```

```
  IF R = 4 THEN BEGIN
```

```
    R := 0;
```

```
    X := 1.7*I + 1290;
```

```
    Y := 1410 + DE*energy[I];
```

```
    WRITELN(LPT2,'PD ',X:0:4,',',Y:0:4)
```

```
  END
```

```
END;
```

- The energy plot is created. Every fourth point is plotted.

```
X := X + 100;
```

```
Y := Y - 25;
```

```

WRITELN(LPT2,'PU ',X:0:4,',',Y:0:4);
WRITELN(LPT2,'SI .12,.21');
WRITELN(LPT2,'LBE');

```

- The energy plot is labeled.

```
t := (Estop - Estart)*Hsc;
```

- The time interval over which integration was conducted is calculated. This value is one of the values printed in the "Breakdown Values" section.

```
STR(t,fullstr);
```

- The time is converted from real number to string.

```

DELETE(fullstr,1,1);
DELETE(fullstr,7,11);
DELETE(fullstr,9,1);
time := COPY(fullstr,1,10);

```

- The time is placed into the string "time" as a 10 character representation of the time to breakdown in scientific notation. This step saves the printing of 25 characters to get the time, and it saves space on the paper.

```

WRITELN(LPT2,'SI');
WRITELN(LPT2,'PU 7585,3100');
WRITELN(LPT2,'CP 6,0');
WRITELN(LPT2,'LB',time,'S');

```

- The "time" string is printed.

```

CLOSE(LPT2)
END;

```

```
{ ***** }
```

```
PROCEDURE VIBreakdown (stor : STR13; PmaxPos : INTEGER);
```

- This procedure is used to calculate the values of  $V_{ce}$  and  $I_C$  at breakdown (specifically, at the protection circuit's firing).

```
TYPE
```

```
VIarray = ARRAY[1..10] OF INTEGER;
```

VAR

```
store : TEXT;
VIarr : VIarray;
coming : STRING[14];
outstr : STRING[9];
V,I,R : INTEGER;
test : BOOLEAN;
incoming : CHAR;
H,ScaleA,ScaleB,VV,VI,probeA,probeB,Vbd,Ibd : REAL;
```

BEGIN

```
coming := stor;
INSERT('A',coming,10);
ASSIGN(store,coming);
RESET(store);
READLN(store);
READLN(store);
READLN(store,VV,H,probeA);
ScaleA := VV*probeA/32;
```

- The file containing  $V_{ce}$  is opened the first lines are read and the vertical scale factor is calculated.

```
Vbd := 0;
```

```
FOR R := 1 TO PmaxPos-15 DO READ(store,incoming);
```

- The cursor position in the file is moved to the location 15 characters before the position where the energy is at its maximum. This corresponds to 15 time steps prior the region where breakdown occurred.

```
FOR R := 1 TO 10 DO BEGIN
```

```
  READ(store,incoming);
  VIarr[R] := ORD(incoming);
```

```
END;
```

- The next 10 characters are read and converted to integers.

```
CLOSE(store);
```

```
test := TRUE;
```



```

R := 1;
WHILE test DO BEGIN
  Vbd := VIarr[R]/VIarr[R+1];
  IF Vbd < 1 THEN Vbd := 1 - Vbd;
  Vbd := FRAC(Vbd);
  IF Vbd > 0.10 THEN BEGIN
    V := R;
    test := FALSE
  END;
  R := R + 1;
  IF R = 10 THEN BEGIN
    V := R;
    test := FALSE
  END
END;

```

- A check is then made for breakdown occurring within those ten data points. If there is a 10% variation from one point to the next, then breakdown is said to have occurred. Only the points prior to this condition being met will be used to calculate the voltage at breakdown.

```

Vbd := 0;
FOR R := 1 TO V DO Vbd := Vbd + VIarr[R];
Vbd := Vbd/V;

```

- The average value of the 10 points or the points up to breakdown is calculated.

```
Vbd := (Vbd-127)*ScaleA;
```

- The average is converted to volts. This is now the breakdown voltage.

```
coming := stor;
```

- A repeat of the above is then done for the file containing  $I_C$ .

```

INSERT('B',coming,10);
ASSIGN(store,coming);
RESET(store);
READLN(store);
READLN(store,VI,H,probeB);
ScaleB := VI*probeB/32;

```

```

Ibd := 0;
FOR R := 1 TO PmaxPos-15 DO READ(store,incoming);
FOR R := 1 TO 10 DO BEGIN
  READ(store,incoming);
  VIarr[R] := ORD(incoming);
END;
CLOSE(store);
test := TRUE;
R := 1;
WHILE test DO BEGIN
  Ibd := VIarr[R]/VIarr[R+1];
  IF Ibd < 1 THEN Ibd := 1 - Ibd;
  Ibd := FRAC(Ibd);
  IF Ibd > 0.10 THEN BEGIN
    V := R;
    test := FALSE
  END;
  R := R + 1;
  IF R = 10 THEN BEGIN
    V := R;
    test := FALSE
  END
END;
Ibd := 0;
FOR R := 1 TO V DO Ibd := Ibd + VIarr[R];
Ibd := Ibd/V;
Ibd := (Ibd-127)*ScaleB;

```

- After both values have been calculated, they are printed out.

```

ASSIGN(store,'LPT2');
REWRITE(store);
WRITELN(store,'PU 7585,2700');
WRITELN(store,'CP -12,0');
NumToStr9(Vbd,outstr);
WRITELN(store,'LB',outstr,'V');

```

- The breakdown voltage is printed in the lower right quadrant.

```
WRITELN(store,'PU 7585,2700');
```

```
WRITELN(store,'CP 6,0');
```

```
NumToStr9(Ibd,outstr);
```

```
WRITELN(store,'LB',outstr,'A');
```

- The breakdown current is printed in the lower right quadrant.

```
WRITELN(store,'PU');
```

```
CLOSE(store)
```

```
END;
```

```
{ **** }
```

```
PROCEDURE plot (stor : STR13;
```

```
  VAR MaxA,MaxB,MaxP,MinA,MinB,MinP : INTEGER;
```

```
  VAR HscA,VscP,ScaleA,ScaleB,energy,fBiasA,rBiasA : REAL;
```

```
  VAR Ign : CHAR);
```

- This procedure is exactly the same as "Plot1" except the plots of  $V_{ce}$  and  $I_C$  are not superimposed.  $I_C$  is placed in the upper right quadrant and  $V_{ce}$  is in the upper left. This is plot option 1. Only changes between this procedure and "Plot1" will be commented.

```
VAR
```

```
  R,I,S : INTEGER;
```

```
  going : STRING[14];
```

```
  incoming : CHAR;
```

```
  VIP : datarray;
```

```
  VI,VV,HscB,HscP,probeA,probeB,fBiasB,rBiasB,fBiasP,
```

```
  rBiasP,ZeroA,ZeroB,ZeroP,
```

```
  numouty,numoutx,DA,DB,DP : REAL;
```

```
  store,LPT2 : TEXT[25];
```

```
BEGIN
```

```
  ASSIGN(LPT2,'LPT2');
```

```
  REWRITE(LPT2);
```

```

    IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
    IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT
END;
{$I+}
READLN(store,fBiasB,rBiasB);
READLN(store,VI,HscB,probeB);
ScaleB := VI*probeB/32;
IF HscA <> HscB THEN BEGIN
    WRITELN('Horizontal scale factors not equal. Program Halted. ');
    HALT
END;
FOR R := 1 TO 2048 DO BEGIN
    READ(store,incoming);
    VIP[2,R] := ORD(incoming)
END;
CLOSE(store);
going := stor;
INSERT('P',going,10);
ASSIGN(store,going);
{$I-}
RESET(store);
WHILE Ioresult <> 0 DO BEGIN
    WRITELN('There has been an error accessing file ',going);
    WRITELN('Please check to be sure the proper disk is in drive A:');
    WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
    READLN(incoming);
    IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
    IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT
END;
{$I+}
READLN(store,fBiasP,rBiasP);
READLN(store,energy,HscP,VscP);
FOR R := 1 TO 2048 DO
    READ(store,VIP[3,R]);

```

```

going := stor;
INSERT('A',going,10);
ASSIGN(store,going);
incoming := 'Q';
{$I-}
RESET(store);
WHILE Ioresult <> 0 DO BEGIN
    WRITELN('There has been an error accessing file ',going);
    WRITELN('Please check to be sure the proper disk is in drive A:');
    WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
    READLN(incoming);
    IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
    IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT
END;
{$I+}
READLN(store,Ign);
READLN(store,fBiasA,rBiasA);
READLN(store,VV,HscA,probeA);
ScaleA := VV*probeA/32;
FOR R := 1 TO 2048 DO BEGIN
    READ(store,incoming);
    VIP[1,R] := ORD(incoming)
END;
CLOSE(store);
going := stor;
INSERT('B',going,10);
ASSIGN(store,going);
{$I-}
RESET(store);
WHILE Ioresult <> 0 DO BEGIN
    WRITELN('There has been an error accessing file ',going);
    WRITELN('Please check to be sure the proper disk is in drive A:');
    WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
    READLN(incoming);

```

```

VIP[1,R] := VIP[1,R] - MinA;
VIP[2,R] := VIP[2,R] - MinB;
VIP[3,R] := VIP[3,R] - MinP
END;
DA := 2000/DA;
DB := 2000/DB;
DP := 2000/DP;
ZeroA := ABS((127-MinA)*DA);
ZeroB := ABS((127-MinB)*DB);
ZeroP := ABS(MinP*DP);
WRITELN(LPT2,'LT 2,1');
IF MinA < 127 THEN BEGIN
  numouty:= 4400+ZeroA;
  WRITELN(LPT2,'PA 1240,',numouty:0:4);
  WRITELN(LPT2,'PD 4800,',numouty:0:4)
END;
IF MinB < 127 THEN BEGIN
  numouty:= 4400+ZeroB;
  WRITELN(LPT2,'PU 5780,',numouty:0:4);
  WRITELN(LPT2,'PD 9340,',numouty:0:4)
END;
IF MinP < 0 THEN BEGIN
  numouty:= 1410+ZeroP;
  WRITELN(LPT2,'PU 1360,',numouty:0:4);
  WRITELN(LPT2,'PD 4800,',numouty:0:4)
END;
WRITELN(LPT2,'LT');
numouty := 4400+(VIP[1,1]*DA);
WRITELN(LPT2,'PU 1290,',numouty:0:4);
FOR I := 2 TO 513 DO BEGIN
  R := (I-1)*4;
  numoutx := 1290+(R-1)*1.7;
  numouty := 4400+(VIP[1,R]*DA);
  WRITELN(LPT2,'PD ',numoutx:0:4,',',numouty:0:4)

```

```
CLOSE(store);
IF ((fBiasP <> fBiasA) OR (fBiasP <> fBiasB) OR (fBiasA <> fBiasB)) THEN BEGIN
  WRITELN('Error in reading in files. Forward Bias Settings not equal.');
```

WRITELN('Program Halted.');

HALT

```
END;
```

IF ((rBiasP <> rBiasA) OR (rBiasP <> rBiasB) OR (rBiasA <> rBiasB)) THEN BEGIN

WRITELN('Error in reading in files. Reverse Bias Settings not equal.');

WRITELN('Program Halted.');

HALT

```
END;
```

IF ((HscA <> HscB) OR (HscA <> HscP)) THEN BEGIN

WRITELN('Error in reading in files. Horizontal Scale Settings not equal.');

WRITELN('Program Halted.');

HALT

```
END;
```

MaxA := 0;

MaxB := 0;

MaxP := -32766;

MinA := 256;

MinB := 256;

MinP := 32767;

FOR R := 1 TO 2048 DO BEGIN

IF (MaxA < VIP[1,R]) THEN MaxA := VIP[1,R];

IF (MaxB < VIP[2,R]) THEN MaxB := VIP[2,R];

IF (MaxP < VIP[3,R]) THEN MaxP := VIP[3,R];

IF (MinA > VIP[1,R]) THEN MinA := VIP[1,R];

IF (MinB > VIP[2,R]) THEN MinB := VIP[2,R];

IF (MinP > VIP[3,R]) THEN MinP := VIP[3,R]

```
END;
```

DA := MaxA - MinA;

DB := MaxB - MinB;

DP := MaxP - MinP;

FOR R := 1 TO 2048 DO BEGIN

```

END;
numouty := 4400+(VIP[2,1]*DB);
WRITELN(LPT2,'PU 5830,',numouty:0:4);
    • The offset in the x-direction for  $I_C$  is now 5830 instead on 1290.
FOR I := 2 TO 513 DO BEGIN
    R := (I-1)*4;
    numoutx := 5830+(R-1)*1.7;
    numouty := 4400+VIP[2,R]*DB;
    WRITELN(LPT2,'PD ',numoutx:0:4,',',numouty:0:4)
END;
numouty := 1410+VIP[3,1]*DP;
WRITELN(LPT2,'PU 1290,',numouty:0:4);
FOR I := 2 TO 513 DO BEGIN
    R := (I-1)*4;
    numoutx := 1290+(R-1)*1.7;
    numouty := 1410+VIP[3,R]*DP;
    WRITELN(LPT2,'PD ',numoutx:0:4,',',numouty:0:4)
END;
WRITELN(LPT2,'SI .12,.21');
numoutx := 1290+256*1.7;
numouty := 1410+VIP[3,256]*DP;
WRITELN(LPT2,'PU ',numoutx:0:4,',',numouty:0:4);
IF numouty > 1610 THEN BEGIN
    WRITELN(LPT2,'CP 0,-1.1');
    WRITELN(LPT2,'LBP')
END
ELSE BEGIN
    WRITELN(LPT2,'CP 0,1.1');
    WRITELN(LPT2,'LBP')
END;
    •  $V_{ce}$  and  $I_C$  are no longer labeled, but P is.
WRITELN(LPT2,'SI');
WRITELN(LPT2,'PU');
CLOSE(LPT2)

```



END;

{\*\*\*\*\*}

PROCEDURE baseplot(Hsc : REAL; device,series : INTEGER);

- This procedure produces the basic plotting elements. The elements are things like the axes, the time labels on the axes and the "Breakdown Values" labeling. The items created by this procedure are the same for both option 1 and 2.

VAR

t1,t2,t3,t4 : STRING[7];

fullstr : STRING[23];

time : REAL;

LPT2 : TEXT;

BEGIN

ASSIGN(LPT2,'LPT2');

REWRITE(LPT2);

time := 512\*Hsc;

- The first time mark is one-quarter of the way down the time axis. It is calculated by this line.

STR(time,fullstr);

DELETE(fullstr,7,12);

DELETE(fullstr,9,2);

DELETE(fullstr,1,2);

t1 := COPY(fullstr,1,7);

- The first time mark is converted to string and shortened.

time := 1024\*Hsc;

STR(time,fullstr);

DELETE(fullstr,7,12);

DELETE(fullstr,9,2);

DELETE(fullstr,1,2);

t2 := copy(fullstr,1,7);

- The second time mark is calculated and converted to string.

```
time := 1536*Hsc;
STR(time,fullstr);
DELETE(fullstr,7,12);
DELETE(fullstr,9,2);
DELETE(fullstr,1,2);
t3 := COPY(fullstr,1,7);
```

- The third time mark.

```
time := 2048*Hsc;
STR(time,fullstr);
DELETE(fullstr,7,12);
DELETE(fullstr,9,2);
DELETE(fullstr,1,2);
t4 := COPY(fullstr,1,7);
```

- The fourth time mark. All of the time marks are the same for all three graphs.

```
WRITELN(LPT2,'IP 520,770,9600,6750');
WRITELN(LPT2,'PA 520,770');
WRITELN(LPT2,'PU 1240,6400');
WRITELN(LPT2,'PD 1340,6400');
WRITELN(LPT2,'PU 1290,6400');
WRITELN(LPT2,'PD 1290,4400,4800,4400');
WRITELN(LPT2,'PU 5780,6400');
WRITELN(LPT2,'PD 5880,6400');
WRITELN(LPT2,'PU 5830,6400');
WRITELN(LPT2,'PD 5830,4400,9340,4400');
WRITELN(LPT2,'PU 1240,3410');
WRITELN(LPT2,'PD 1340,3410');
WRITELN(LPT2,'PU 1290,3410');
WRITELN(LPT2,'PD 1290,1410,4800,1410');
```

- The axes are plotted.
- This is the beginning of the time labeling section.

```
WRITELN(LPT2,'PU 2790,6500');
WRITELN(LPT2,'DT',CHR(10));
```

WRITELN(LPT2,'DR 1,0');  
WRITELN(LPT2,'SI .12,.21');  
WRITELN(LPT2,'PU 1290,4450');  
WRITELN(LPT2,'PD 1290,4350');  
WRITELN(LPT2,'PU 1290,4200');  
WRITELN(LPT2,'LB0');  
WRITELN(LPT2,'PU 1240,4400');  
WRITELN(LPT2,'PD 1300,4400');  
WRITELN(LPT2,'PU 2167.5,4450');  
WRITELN(LPT2,'PD 2167.5,4350');  
WRITELN(LPT2,'PU 2167.5,4200');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t1);  
WRITELN(LPT2,'PU 3045,4450');  
WRITELN(LPT2,'PD 3045,4350');  
WRITELN(LPT2,'PU 3045,4200');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t2);  
WRITELN(LPT2,'PU 3922.5,4450');  
WRITELN(LPT2,'PD 3922.5,4350');  
WRITELN(LPT2,'PU 3922.5,4200');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t3);  
WRITELN(LPT2,'PU 4800,4450');  
WRITELN(LPT2,'PD 4800,4350');  
WRITELN(LPT2,'PU 4800,4200');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t4);  
WRITELN(LPT2,'PU 1290,1460');  
WRITELN(LPT2,'PD 1290,1360');  
WRITELN(LPT2,'PU 1290,1210');  
WRITELN(LPT2,'LB0');  
WRITELN(LPT2,'PU 1240,1410');  
WRITELN(LPT2,'PD 1300,1410');

WRITELN(LPT2,'PU 21 7.5,1460');  
WRITELN(LPT2,'PD 2167.5,1360');  
WRITELN(LPT2,'PU 2167.5,1210');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t1);  
WRITELN(LPT2,'PU 3045,1460');  
WRITELN(LPT2,'PD 3045,1360');  
WRITELN(LPT2,'PU 3045,1210');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t2);  
WRITELN(LPT2,'PU 3922.5,1460');  
WRITELN(LPT2,'PD 3922.5,1360');  
WRITELN(LPT2,'PU 3922.5,1210');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t3);  
WRITELN(LPT2,'PU 4800,1460');  
WRITELN(LPT2,'PD 4800,1360');  
WRITELN(LPT2,'PU 4800,1210');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t4);  
WRITELN(LPT2,'PU 5830,4450');  
WRITELN(LPT2,'PD 5830,4350');  
WRITELN(LPT2,'PU 5830,4200');  
WRITELN(LPT2,'LB0');  
WRITELN(LPT2,'PU 5780,4400');  
WRITELN(LPT2,'PD 5850,4400');  
WRITELN(LPT2,'PU 6707.5,4450');  
WRITELN(LPT2,'PD 6707.5,4350');  
WRITELN(LPT2,'PU 6707.5,4200');  
WRITELN(LPT2,'CP -3.5,0');  
WRITELN(LPT2,'LB',t1);  
WRITELN(LPT2,'PU 7585,4450');  
WRITELN(LPT2,'PD 7585,4350');  
WRITELN(LPT2,'PU 7585,4200');

```

WRITELN(LPT2,'CP -3.5,0');
WRITELN(LPT2,'LB',t2);
WRITELN(LPT2,'PU 8462.5,4450');
WRITELN(LPT2,'PD 8462.5,4350');
WRITELN(LPT2,'PU 8462.5,4200');
WRITELN(LPT2,'CP -3.5,0');
WRITELN(LPT2,'LB',t3);
WRITELN(LPT2,'PU 9340,4450');
WRITELN(LPT2,'PD 9340,4350');
WRITELN(LPT2,'PU 9340,4200');
WRITELN(LPT2,'CP -3.5,0');
WRITELN(LPT2,'LB',t4);

```

- This is the end of the time labeling section. All three graphs have been labeled.
- The following section writes the "Breakdown Values" section minus the data.

```

WRITELN(LPT2,'PA 7585,3500');
WRITELN(LPT2,'SI .27,.375');
WRITELN(LPT2,'CP -8,0');
WRITELN(LPT2,'LBBreakdown Values');
WRITELN(LPT2,'SI');
WRITELN(LPT2,'PA 7585,3100');
WRITELN(LPT2,'CP -17,0');
WRITELN(LPT2,'LBEnergy:');
WRITELN(LPT2,'PA 7585,3100');
WRITELN(LPT2,'CP 2,0');
WRITELN(LPT2,'LBTime:');
WRITELN(LPT2,'PU 7585,2700');
WRITELN(LPT2,'CP -16,0');
WRITELN(LPT2,'LBV :');
WRITELN(LPT2,'PU 7585,2700');
WRITELN(LPT2,'CP -15,-.3');
WRITELN(LPT2,'LBCE');
WRITELN(LPT2,'PU 7585,2700');

```

```
WRITELN(LPT2,'CP 2,0');
WRITELN(LPT2,'LBI :');
WRITELN(LPT2,'PU 7585,2700');
WRITELN(LPT2,'CP 3,-.3');
WRITELN(LPT2,'LBC');
WRITELN(LPT2,'PA 7585,2300');
WRITELN(LPT2,'SI .27,.375');
WRITELN(LPT2,'CP -6.5,0');
WRITELN(LPT2,'LBBias Settings');
WRITELN(LPT2,'SI');
WRITELN(LPT2,'PU 7585,1900');
WRITELN(LPT2,'CP -18,0');
WRITELN(LPT2,'LBForward:');
WRITELN(LPT2,'PU 7585,1900');
WRITELN(LPT2,'LBReverse:');
WRITELN(LPT2,'SI .27,.375');
WRITELN(LPT2,'PA 7585,1500');
WRITELN(LPT2,'CP -12,0');
WRITELN(LPT2,'LBDevice ',device);
WRITELN(LPT2,'PA 7585,1500');
WRITELN(LPT2,'LBSeries ',series);
```

- This is ends the section where the "Breakdown Values" are written.
- The next section writes the time axes titles.

```
WRITELN(LPT2,'SI');
WRITELN(LPT2,'PA 3045,3925');
WRITELN(LPT2,'CP -7,0');
WRITELN(LPT2,'LBTime (Seconds)');
WRITELN(LPT2,'PA 3045,935');
WRITELN(LPT2,'CP -7,0');
WRITELN(LPT2,'LBTime (Seconds)');
WRITELN(LPT2,'PA 7585,3925');
WRITELN(LPT2,'CP -7,0');
WRITELN(LPT2,'LBTime (Seconds)');
CLOSE(LPT2)
```

END;

{ \*\*\*\* }

PROCEDURE LabelChoice1;

- This procedure is used to print the graph and vertical axis titles for plot option 1. The procedure is a series of WRITELN statements. See the HP-7470A User's Manual for more information.

VAR

LPT2 : TEXT;

BEGIN

ASSIGN(LPT2,'LPT2');

REWRITE(LPT2);

WRITELN(LPT2,'SI .27,.375');

WRITELN(LPT2,'PU 2790,6600');

WRITELN(LPT2,'CP -12.5,0');

WRITELN(LPT2,'LBCollector-Emitter Voltage');

WRITELN(LPT2,'PA 7330,6600');

WRITELN(LPT2,'CP -8.5,0');

WRITELN(LPT2,'LBCollector Current');

WRITELN(LPT2,'PA 2790,3610');

WRITELN(LPT2,'CP -4.5,0');

WRITELN(LPT2,'LBP and E');

WRITELN(LPT2,'PA 2790,3610');

WRITELN(LPT2,'CP -3.5,-.3');

WRITELN(LPT2,'LBI C');

WRITELN(LPT2,'SI');

WRITELN(LPT2,'DR 0,1');

WRITELN(LPT2,'PA 5430,5350');

WRITELN(LPT2,'CP -3.5,0');

WRITELN(LPT2,'LBAmperes');

WRITELN(LPT2,'PA 840,5350');

```

WRITELN(LPT2,'CP -2.5,0');
WRITELN(LPT2,'LBVolts');
WRITELN(LPT2,'PA 740,2360');
WRITELN(LPT2,'CP -2.5,0');
WRITELN(LPT2,'LBWatts');
WRITELN(LPT2,'PA 740,2360');
WRITELN(LPT2,'CP -3,-1.2');
WRITELN(LPT2,'LBJoules');
WRITELN(LPT2,'DR 1,0');
WRITELN(LPT2,'PU 5500,1000');
WRITELN(LPT2,'LT 2,1');
WRITELN(LPT2,'PD 6300,1000');
WRITELN(LPT2,'LT');
WRITELN(LPT2,'PU 6550,1000');
WRITELN(LPT2,'LBZero Crossing V, I and P');
CLOSE(LPT2)
END;

```

```
{ ***** }
```

```
PROCEDURE VerticalAxis (MaxA,MaxB,MaxP,MinA,MinB,MinP : INTEGER;
```

```
    VscP,ScaleA,ScaleB,energy,fBias,rBias : REAL);
```

- This procedure prints the vertical axis scales for plot option 1. It also writes the energy at breakdown and forward and reverse bias settings in the "Breakdown Values" section. It is exactly the same as "VerticalAxis1" except for the location where the vertical scales for  $I_C$  is concerned. Instead of the values for  $I_C$  being under the values for  $V_{ce}$  in the upper left quadrant, they are located in the upper right quadrant where the scale values for  $V_{be}$  would have been.

```
VAR
```

```
    fullstr : STRING[27];
```

```
    outstr : STRING[9];
```

```
    MinPlotA,MinPlotB,MinPlotP,MaxPlotA,MaxPlotB,MaxPlotP : REAL;
```



LPT2 : TEXT;

BEGIN

  ASSIGN(LPT2,'LPT2');

  REWRITE(LPT2);

  MinPlotA := ScaleA\*(MinA - 127);

  MinPlotB := ScaleB\*(MinB - 127);

  MinPlotP := VscP\*MinP;

- The minimum values of  $V_{ce}$ ,  $I_c$  and P are calculated.

  MaxPlotA := ScaleA\*(MaxA - 127);

  MaxPlotB := ScaleB\*(MaxB - 127);

  MaxPlotP := VscP\*MaxP;

- The maximum values of  $V_{ce}$ ,  $I_c$  and P are calculated.

  WRITELN(LPT2,'SI .12,.21');

- In the following section, the vertical scales are printed.

  WRITELN(LPT2,'PU 750,6400');

  WRITELN(LPT2,'CP -2,-.5');

  NumToStr9(MaxPlotA,outstr);

  WRITELN(LPT2,'LB',outstr);

  WRITELN(LPT2,'PU 750,4400');

  WRITELN(LPT2,'CP -2,-.5');

  NumToStr9(MinPlotA,outstr);

  WRITELN(LPT2,'LB',outstr);

  WRITELN(LPT2,'PU 5290,6400');

  WRITELN(LPT2,'CP -2,-.5');

  NumToStr9(MaxPlotB,outstr);

  WRITELN(LPT2,'LB',outstr);

  WRITELN(LPT2,'PU 5290,4400');

  WRITELN(LPT2,'CP -2,-.5');

  NumToStr9(MinPlotB,outstr);

  WRITELN(LPT2,'LB',outstr);

  WRITELN(LPT2,'PU 750,3410');

  WRITELN(LPT2,'CP -3,.5');

  NumToStr9(MaxPlotP,outstr);

```

WRITELN(LPT2,'LB',outstr,'W');
WRITELN(LPT2,'PU 750,3410');
WRITELN(LPT2,'CP -3,-.5');
NumToStr9(energy,outstr);
WRITELN(LPT2,'LB',outstr,'J');
WRITELN(LPT2,'PU 750,1410');
WRITELN(LPT2,'CP -3,-.5');
NumToStr9(0,outstr);
WRITELN(LPT2,'LB', outstr,'J');
WRITELN(LPT2,'PU 750,1410');
WRITELN(LPT2,'CP -3,.5');
NumToStr9(MinPlotP,outstr);
WRITELN(LPT2,'LB',outstr,'W');

```

- End of the section where the vertical scales are printed.
- Below the energy at breakdown and the forward and reverse bias settings are printed in the "Breakdown Values" section.

```

WRITELN(LPT2,'SI');
WRITELN(LPT2,'PU 7585,3100');
WRITELN(LPT2,'CP -11,0');
NumToStr9(energy,outstr);
WRITELN(LPT2,'LB ',outstr,'J');
WRITELN(LPT2,'PU 7585,1900');
WRITELN(LPT2,'CP -12,0');
NumToStr9(fBias,outstr);
WRITELN(LPT2,'LB ',outstr,'A');
WRITELN(LPT2,'PU 7585,1900');
WRITELN(LPT2,'CP 6.2,0');
NumToStr9(rBias,outstr);
WRITELN(LPT2,'LB ',outstr,'A');
CLOSE(LPT2)
END;

```

```
{ ***** }
```

PROCEDURE plotend;

- This procedure is used to return the plotting pen to its stall and present the finished plot to the operator.

VAR

LPT2 : TEXT;

BEGIN

ASSIGN(LPT2,'LPT2');

REWRITE(LPT2);

Writeln(LPT2,'PU 0,0');

Writeln(LPT2,'SP 0');

CLOSE(LPT2)

END;

{ \*\*\*\*\* }

PROCEDURE Ignorproc;

- If breakdown did not occur, this procedure comes into play. If there was no breakdown, then the "Breakdown Values" section is not needed and not accurate. Rather than modify the program to print a different format for non-breakdown, this procedure was written. It prints the word "IGNORE" across the "Breakdown Values" section as a reminder that the numbers written there are irrelevant.

VAR

LPT2 : TEXT;

BEGIN

ASSIGN(LPT2,'LPT2');

REWRITE(LPT2);

Writeln(LPT2,'DR 86.67,25');

Writeln(LPT2,'PU 5700,2700');

Writeln(LPT2,'SI 1,1');

```

WRITELN(LPT2,'LBIGNORE');
WRITELN(LPT2,'DR');
WRITELN(LPT2,'SI');
CLOSE(LPT2)
END;

```

```
{ ***** }
```

```

BEGIN {MAIN PROGRAM}
REPEAT

```

- This loop is used to allow the operator to plot multiple test runs without having to load the program over and over.

```

Ignorit := 'Q';
listdevice(storage,DEV,SER);
PlotWhat(storage,plotall);
plotinitialize;
IF plotall = 'N' THEN BEGIN

```

- If the operator selected option 1 in "PlotWhat", then this section is executed.

```

plot(storage,MaximumA,MaximumB,MaximumP,MinimumA,MinimumB,MinimumP,
      Hscale,Vscale,Ascale,Bscale,Ener,ForBias,RevBias,Ignorit);
LabelChoice1;
baseplot(Hscale,DEV,SER);
VerticalAxis(MaximumA,MaximumB,MaximumP,MinimumA,MinimumB,
             MinimumP,Vscale,Ascale,Bscale,Ener,ForBias,RevBias);
EnergyPlot(storage,Hscale,PmaxPoint);
VIBreakdown(storage,PmaxPoint)

```

- End of the plot option 1 section.

```

END;
IF plotall = 'Y' THEN BEGIN

```

- Plot option 2 begins here.

```

plot1(storage,MaximumA,MaximumB,MaximumP,MinimumA,MinimumB,
       MinimumP,Hscale,Vscale,Ascale,Bscale,Ener,ForBias,RevBias,Ignorit);
plot2(storage,MaxC,MinC,Hscale,ScaleC);

```

```

LabelChoice2;
baseplot(Hscale,DEV,SER);
VerticalAxis1(MaximumA,MaximumB,MaximumP,MinimumA,MinimumB,
              MinimumP,Vscale,Ascale,Bscale,Ener,ForBias,RevBias);
VerticalAxis2(MaxC,MinC,ScaleC);
EnergyPlot(storage,Hscale,PmaxPoint);
VIBreakdown(storage,PmaxPoint)

```

- Plot option 2 ends here.

```

END;
IF ((Ignorit = 'N') OR (Ignorit = 'n')) THEN Ignorproc;
plotend;
answer := 'Q';
WHILE NOT ((answer = 'y') OR (answer = 'Y') OR (answer = 'n') OR (answer = 'N'))
DO BEGIN
  WRITE('Plot another? (Y or N) . . . ');
  READLN(answer)
END;

```

- The operator is asked if he or she wants to plot another test run. If yes, the loop is executed again; otherwise, the loop ends and so does the program.

```

IF ((answer = 'Y') OR (answer = 'y')) THEN testmain := FALSE;
IF ((answer = 'N') OR (answer = 'n')) THEN testmain := TRUE
UNTIL testmain
END.

```

APPENDIX K  
TABULATE.PAS PROGRAM LISTING

PROGRAM Tabulate;

- This program is used to create the file BDxxxLL.TAB which is a compilation of all the breakdown values on a given device, "xxx." The output is written to the hard disk directory, "C:\RBSOA." The program is written for the Turbo Pascal compiler.

TYPE

Rarray = ARRAY[1..6,1..999] OF REAL;

STR13 = STRING[13];

VAR

storage : STRING[13];

Ppos,rep,SER : INTEGER;

ForB,RevB,Voltage,Current,TotEnergy,Time : REAL;

BreakDown : CHAR;

{ \*\*\*\*\* }

PROCEDURE listdevice(VAR stor : STR13; VAR series : INTEGER);

- This is the same general procedure used in PLOTDATA. It serves to provide the operator with a list of the devices currently stored on the file, INDEX.DEV, as well as allowing the operator to choose the device to be tabulated. It returns the storage filename and the number of test runs taken on the device.

TYPE

STR40 = STRING[40];

STR13 = STRING[13];

DEVICES = RECORD

DEV : INTEGER;

descrip : STR40;

book : STR13;

store : STR13

END;

VAR

index : FILE OF DEVICES;  
 indexrec : DEVICES;  
 error,LineCount : INTEGER;  
 name : STRING[18];  
 one : STRING[1];  
 two : STRING[2];  
 thr : STRING[3];  
 test : BOOLEAN;  
 ans : CHAR;  
 I,Count,err,device : INTEGER;

BEGIN

name := 'C:\RBSOAN\INDEX.DEV';  
 ASSIGN(index,name);  
 RESET(index);  
 Writeln('You will be asked for the device number corresponding to the storage file');  
 Writeln('for the data taken on the device of interest. ');  
 Writeln;

ans := 'q';

WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO

BEGIN

Writeln('Do you want the see a listing of all currently cataloged devices?');

WRITE('See list? (Y or N) . . . ');

READLN(ans)

END; { WHILE NOT ((ans = 'y'...}

IF ((ans = 'Y') OR (ans = 'y')) THEN BEGIN

- If the operator wants to see a listing of the devices in INDEX.DEV, then the following is executed.

LineCount := 1;

Count := 0;

SEEK(index,LineCount);



```
WHILE NOT EOF(index) DO BEGIN
```

```
  READ(index,indexrec);
```

- A device record is read.

```
  WITH indexrec DO BEGIN
```

```
    WRITE(DEV,' ');
```

```
    WRITE(descrip:40,' ');
```

```
    WRITE(book,' ');
```

```
    WRITELN(store)
```

```
  END;
```

- The record fields are displayed on the screen.

```
  LineCount := LineCount + 1;
```

- The record pointer is increased by one.

```
  Count := Count + 1;
```

- The count of lines displayed since last pause or beginning of list is increased by one.

```
  IF Count = 21 THEN BEGIN
```

- If the number of lines displayed is 21, then the program pauses to allow the operator to read the device records displayed.

```
    Count := 0;
```

```
    WRITELN('Press <RETURN> for more devices.');
```

```
    READLN
```

```
  END
```

```
END;
```

```
  WRITELN('End of indexed devices.')
```

```
END; { IF ((ans = 'Y'...) }
```

```
test := TRUE;
```

```
WHILE test DO BEGIN
```

```
  WRITE('What is the device number? (Integer Value Only!) . . . ');
```

```
  READLN(device);
```

```
  IF device < 1000 THEN test := FALSE
```

```
  ELSE WRITELN('Integer too large. Try again.')
```

```
END;
```

- The device number is entered.

```
  SEEK(index,device);
```

READ(index,indexrec);

- The record of the device of interest is read.

WITH indexrec DO

stor := store;

- The storage filename is loaded into the string variable, "stor."

CLOSE(index);

DELETE(stor,1,1);

INSERT('A',stor,1);

- The drive designation is changed from G: to A:.

thr := COPY(stor,11,3);

VAL(thr,series,err);

IF err <> 0 THEN BEGIN

WRITELN('Error in reading maximum series value. Program Halted.');

HALT

END;

- The series number (last three characters in "stor") is copied into the string, "thr." The string is converted to an integer by the built-in procedure "VAL." An error check is made on the conversion. The program is halted if there is an error.

DELETE(stor,11,3);

INSERT('000',stor,11)

- The last three characters on "stor" are changed to "000." The string is then passed to the main program.

END; {Procedure listdevice}

{\*\*\*\*\*}

PROCEDURE addone(VAR stor : STR13);

- This procedure is use to add a value of one to the series number in the string, "stor." The new string is then passed back to the main program.

VAR

I : INTEGER;

```

BEGIN
  IF ORD(stor[13]) = 57 THEN BEGIN
    IF ORD(stor[12]) = 57 THEN BEGIN
      IF ORD(stor[11]) = 57 THEN BEGIN
        • If the series number is 999, then ...
        WRITELN('Too Many Indexed Devices. Program Halted.');
```

HALT

```

      END
    ELSE BEGIN
      I := ORD(stor[11]) + 1;
      stor[11] := CHR(I);
      stor[12] := CHR(48);
      stor[13] := CHR(48)
      • If the series number is x99 (where x is 0 to 8), then the "x" is increased
        by one and the two nines are set to zero.

    END
  END
  ELSE BEGIN
    I := ORD(stor[12]) + 1;
    stor[12] := CHR(I);
    stor[13] := CHR(48)
    • If the series number is xy9 (where y is 0 to 8), then y is increased by one
      and the nine is set to zero.

  END
END
  ELSE BEGIN
    I := ORD(stor[13]) + 1;
    stor[13] := CHR(I)
    • If the series number is xyz (where z is 0 to 8), then z is increased by one.

  END
END;
```

```
{ ***** }
```

```
PROCEDURE VIBreakdown (stor : STR13; PmaxPos : INTEGER;
```

```
VAR ForBias,RevBias,Vbd,Ibd : REAL; GoNoGo : CHAR);
```

- This procedure is used to find the breakdown values of  $V_{ce}$  and  $I_c$ .

```
TYPE
```

```
VIarray = ARRAY[1..10] OF INTEGER;
```

```
VAR
```

```
store : TEXT;
```

```
coming : STRING[14];
```

```
VIarr : VIarray;
```

```
test : BOOLEAN;
```

```
V,I,R : INTEGER;
```

```
incoming : CHAR;
```

```
H,ScaleA,ScaleB,VV,VI,probeA,probeB : REAL;
```

```
BEGIN
```

```
IF ((GoNoGo = 'Y') OR (GoNoGo = 'y')) THEN BEGIN
```

- The "GoNoGo" variable is has the first character of the BDxxxLLA.yyy file in it. If breakdown occurred, then the value is "Y."

```
coming := stor;
```

```
INSERT('A',coming,10);
```

```
ASSIGN(store,coming);
```

```
RESET(store);
```

```
READLN(store);
```

```
READLN(store,ForBias,RevBias);
```

```
READLN(store,VV,H,probeA);
```

```
ScaleA := VV*probeA/32;
```

- The file containing  $V_{ce}$  is opened the bias settings are read, the Volts/Div, horizontal scale factor and probe multiplication factor are read and the vertical scale factor is calculated.

```
Vbd := 0;
```

```
FOR R := 1 TO PmaxPos-15 DO READ(store,incoming);
```

```
FOR R := 1 TO 10 DO BEGIN
```

```
  READ(store,incoming);
```

```
  VIarr[R] := ORD(incoming);
```

```
END;
```

- The file cursor is move to a point 15 time steps before the time step where the maximum energy is found. The next 10 data points are read.

```
CLOSE(store);
```

```
test := TRUE;
```

```
R := 1;
```

```
WHILE test DO BEGIN
```

```
  Vbd := VIarr[R]/VIarr[R+1];
```

```
  IF Vbd < 1 THEN Vbd := 1 - Vbd;
```

```
  Vbd := FRAC(Vbd);
```

```
  IF Vbd > 0.10 THEN BEGIN
```

```
    V := R;
```

```
    test := FALSE
```

```
  END;
```

```
  R := R + 1;
```

```
  IF R = 10 THEN BEGIN
```

```
    V := R;
```

```
    test := FALSE
```

```
  END
```

```
END;
```

- The data points are checked for the occurrence of breakdown. If there is a change of greater than 10% in the data points magnitudes from one step to the next, breakdown is said to have occurred.

```
Vbd := 0;
```

```
FOR R := 1 TO V DO Vbd := Vbd + VIarr[R];
```

```
Vbd := Vbd/V;
```

```
Vbd := (Vbd-127)*ScaleA;
```

- The average of the 10 data points (or as many data points as there were prior to the point where breakdown was judged to have occurred) are averaged and converted to volts.
- The same procedure is now done for  $I_C$ .

```

coming := stor;
INSERT('B',coming,10);
ASSIGN(store,coming);
RESET(store);
READLN(store);
READLN(store,VI,H,probeB);
ScaleB := VI*probeB/32;
Ibd := 0;
FOR R := 1 TO PmaxPos-15 DO READ(store,incoming);
FOR R := 1 TO 10 DO BEGIN
  READ(store,incoming);
  VIarr[R] := ORD(incoming);
END;
CLOSE(store);
test := TRUE;
R := 1;
WHILE test DO BEGIN
  Ibd := VIarr[R]/VIarr[R+1];
  IF Ibd < 1 THEN Ibd := 1 - Ibd;
  Ibd := FRAC(Ibd);
  IF Ibd > 0.10 THEN BEGIN
    V := R;
    test := FALSE
  END;
  R := R + 1;
  IF R = 10 THEN BEGIN
    V := R;
    test := FALSE
  END
END;
Ibd := 0;
FOR R := 1 TO V DO Ibd := Ibd + VIarr[R];
Ibd := Ibd/V;
Ibd := (Ibd-127)*ScaleB

```

END  
END;

{\*\*\*\*\*}

PROCEDURE GetValues (stor : STR13; VAR incoming : CHAR;  
VAR PmaxPos : INTEGER;  
VAR Emax,t : REAL);

- This procedure locates the time step where breakdown occurred. It, also, returns the energy at breakdown and the time to breakdown. Lastly, it reads the first character of the first line of BDxxxLLA.yyy and returns that to the main program (indicates whether or not breakdown occurred during the particular test run).

TYPE

earr = ARRAY[1..2047] OF REAL;

VAR

store : TEXT;

energy : earr;

R,Estart,Estop,I : INTEGER;

going : STRING[14];

Hscale : REAL;

test : BOOLEAN;

BEGIN

going := stor;

INSERT('A',going,10);

ASSIGN(store,going);

{ \$I- }

RESET(store);

WHILE Ioresult <> 0 DO BEGIN

WRITELN('There has been an error accessing file ',going);

WRITELN('Please check to be sure the proper disk is in drive A:');

```

incoming := 'Q';
WHILE NOT ((incoming = 'Y') OR (incoming = 'y') OR (incoming = 'N') OR
(incoming = 'n')) DO BEGIN
    WRITE('Enter "Y" to continue, or "N" to halt program. . . ');
    READLN(incoming)
END;
IF ((incoming = 'y') OR (incoming = 'Y')) THEN RESET(store);
IF ((incoming = 'N') OR (incoming = 'n')) THEN HALT
END;
{$I+}
READLN(store,incoming);
READLN(store);
READLN(store,Emax,Hscale);
CLOSE(store);
    • The file, BDxxxLLA.yyy is opened, the first character is read, parts of
      the second and third lines are read and the file is closed.
IF ((incoming = 'Y') OR (incoming = 'y')) THEN BEGIN
    • If breakdown occurred, this will execute.
going := stor;
INSERT('E',going,10);
ASSIGN(store,going);
RESET(store);
FOR R := 1 TO 2047 DO
    READ(store,energy[R]);
    • The file containing the cumulative energy is read.
CLOSE(store);
FOR R := 1 TO 2047 DO
    IF test THEN
        IF energy[R] > 0 THEN BEGIN
            test := FALSE;
            Estart := R
        END;
Estart := Estart - 1,

```



- The position where the energy becomes non-zero is found. This is the start point for determining the time to breakdown.

Emax := -100000.0;

FOR R := 1 TO 2047 DO

IF energy[R] > Emax THEN Emax := energy[R];

- The maximum value of the energy (total energy) is found.

test := TRUE;

FOR R := 1 TO 2047 DO

IF test THEN

IF energy[R] = Emax THEN BEGIN

test := FALSE;

Estop := R

END;

Estop := Estop + 1;

- The location of the maximum energy values occurrence is found.

PmaxPos := Estop - 2;

- The location where breakdown is said to happen is calculated.

t := Hscale\*(Estop-Estart)

- The time to breakdown is calculated.

END

END;

{ \*\*\*\* }

PROCEDURE WriteOutput(stor : STR13; Ind : INTEGER; Fb,Rb,V,I,E,t : REAL;

GoNoGo : CHAR);

- This procedure outputs the breakdown values from one test run to the BDxxxLL.TAB file on the hard disk.

TYPE

earr = ARRAY[1..2047] OF REAL;

DEVICETABLE = RECORD

series : INTEGER;

ForBias : REAL;

```

RevBias : REAL;
Vce  : REAL;
Ic   : REAL;
energy : REAL;
time  : REAL
END;

```

- The breakdown values are stored in records consisting of the above fields.

```
VAR
```

```

tab : FILE OF DEVICETABLE;
tabrecord : DEVICETABLE;
Values : Rarray;
goinglong : STRING[20];
MaxRec : INTEGER;

```

```
BEGIN
```

```
IF ((GoNoGo = 'Y') OR (GoNoGo = 'y')) THEN BEGIN
```

- If breakdown did not occur, then the rest of this procedure will not be executed.

```

goinglong := stor;
DELETE(goinglong,11,3);
INSERT('TAB',goinglong,11);
DELETE(goinglong,1,2);
INSERT('C:\RBSOA\',goinglong,1);
ASSIGN(tab,goinglong);

```

```
[$I-]
```

```
RESET(tab);
```

```
[$I+]
```

```
IF IOresult <> 0 THEN REWRITE(tab);
```

- The tabulated data file is opened to receive new data. If the file does not exist, it is created.

```
MaxRec := FileSize(tab);
```

- The location of the next record to be written is determined. It will be place after the last entry.

```
IF MaxRec < 0 THEN MaxRec := 0;
```

```
WITH tabrecord DO BEGIN
```

```
  series := Ind;
```

```
  ForBias := Fb;
```

```
  RevBias := Rb;
```

```
  Vce := V;
```

```
  Ic := I;
```

```
  energy := E;
```

```
  time := t;
```

- The breakdown values are placed into the record fields of the record about to be added to the tabulated data file.

```
  WRITELN('  Series  ',series);
```

```
  WRITELN('  Bias Settings');
```

```
    WRITELN('Forward:',ForBias:9:6,'A',' Reverse:',RevBias:9:6,'A');
```

```
  WRITELN('  Breakdown Values');
```

```
    WRITELN('Voltage:',Vce:9:4,'V',' Current:',Ic:9:4,'A');
```

```
    WRITELN('Energy:',energy:9:4,'J',' Time:',time:14:10,'Sec')
```

- The above six lines are displayed on the screen to let the operator know that data is being written to the disk and what that data is.

```
END;
```

```
SEEK(tab,MaxRec);
```

```
WRITE(tab,tabrecord);
```

- The record is written to disk.

```
CLOSE(tab)
```

```
END
```

```
END;
```

```
{ ***** }
```

```
BEGIN { Main Program }
```

```
  ListDevice(storage,SER);
```

```
  FOR rep := 1 to SER DO BEGIN
```

- The following steps will only be taken as many times as the total number of test taken on the device. This way there are no repeat measurements nor any searches for files that do not exist.

addone(storage);

GetValues(storage,Breakdown,Ppos,TotEnergy,Time);

VIBreakdown(storage,Ppos,ForB,RevB,Voltage,Current,Breakdown);

WriteOutput(storage,rep,ForB,RevB,Voltage,Current,TotEnergy,Time,Breakdown)

END

END.

APPENDIX L  
SORTABLE.PAS PROGRAM LISTING

PROGRAM SortTabulatedData;

- This program takes a file created by TABULATE.PAS, sorts the data contained in the file and outputs the sorted information to either a line printer or a text file on disk.

TYPE

```
STR40 = STRING[40];
STR13 = STRING[13];
STR9 = STRING[9];
Iarray = ARRAY[1..2,1..999] OF REAL;
Rarray = ARRAY[1..6,1..999] OF REAL;
```

VAR

```
storage : STR13;
PathN : STR40;
DK1,DK2,Recs,K1,K2 : INTEGER;
Valuarr : Rarray;
Posarr : Iarray;
```

{ \*\*\*\*\* }

PROCEDURE NumToStr9 (X : REAL; VAR Y : STR9);

- This procedure is used to prepare real numbers for output as strings. The values are limited to three significant figures in scientific notation format.

VAR

```
fullstr : STRING[25];
one : STRING[1];
thr : STRING[3];
four : STRING[4];
error,I : INTEGER;
num : REAL;
```

BEGIN

STR(X,fullstr);

DELETE(fullstr,1,1);

one := COPY(fullstr,6,1);

- The third digit to the right of the decimal place is loaded into the one character long string "one."

VAL(one,num,error);

- The character in "one" is translated into numeric format and loaded into "num."

DELETE(fullstr,6,12);

DELETE(fullstr,8,1);

Y := COPY(fullstr,1,9);

- The extra characters in the original number passed to the procedure are deleted. The shortened form of the number is placed in the nine character long string, "Y." The number is shortened from " $\pm x.xxxxxxxxxxxxxxxxxE\pm xxx$ " to " $\pm x.xx E\pm xx$ ."

IF num >= 5 THEN BEGIN

- If the digit in the third place to the right of the decimal point is equal to or over five, the digit to the left is increased by one. This serves to round the new number. The rest of this procedure is involved in performing this function.

four := COPY(fullstr,2,4);

VAL(four,num,error);

num := num + 0.01;

IF num = 10.00 THEN BEGIN

- This section increases the exponent if it is needed.

num := 1.00;

thr := COPY(Y,7,3);

VAL(thr,I,error);

I := I + 1;

STR(I,thr);

IF ((I < 10) AND (I > -10)) THEN INSERT('0',thr,2);

INSERT(thr,Y,7)

END;

STR(num:3:2,four);

```

DELETE(Y,2,4);
INSERT(four,Y,2)
END
END;

```

```

{ **** }

```

```

PROCEDURE listdevice(VAR pName : STR40);

```

- This is a modified version of the procedure of the same name used in TABULATE and PLOTDATA. It returns the location and filename of the tabulated file for the device the user selects.

```

CONST

```

```

FilePresent : BOOLEAN = FALSE;

```

```

TYPE

```

```

TabArray = ARRAY[1..999] OF INTEGER;

```

```

STR40 = STRING[40];

```

```

STR13 = STRING[13];

```

```

DEVICES = RECORD

```

```

    DEV : INTEGER;

```

```

    descrip : STR40;

```

```

    book : STR13;

```

```

    store : STR13

```

```

END;

```

```

DEVICETABLE = RECORD

```

```

    series : INTEGER;

```

```

    ForBias : REAL;

```

```

    RevBias : REAL;

```

```

    Vce : REAL;

```

```

    Ic : REAL;

```

```

    energy : REAL;

```

```

    time : REAL

```

```

END;

```



VAR

```

indextest : FILE OF DEVICETABLE;
index : FILE OF DEVICES;
indexrec : DEVICES;
pNameLen,I,device,LineCount : INTEGER;
TabFilePres : TabArray;
one : STRING[1];
two : STRING[2];
thr : STRING[3];

```

BEGIN

```
FOR I := 1 TO 999 DO TabFilePres[I] := 0;
```

- The array "TabFilePres" is initialized. The array will contain the results of a search on disk for the existence of a tabulated data file.

```
ASSIGN(index,'C:\RBSOA\INDEX.DEV');
```

```
RESET(index);
```

```
LineCount := FileSize(index);
```

```
WRITELN('Enter the pathname of the directory containing the tabulated file.');
```

```
WRITELN('Example: A: or C:\RBSOA');
```

```
WRITE('Pathname . . . ');
```

```
READLN(pName);
```

- The operator enters the pathname for the directory to be searched for tabulated data files.

```
pNameLen := LENGTH(pName);
```

- The length in characters of the pathname is determined.

```
FOR I := 1 TO LineCount-1 DO BEGIN
```

- The program will search for all possible device numbers on the path specified. The FOR loop does this by iterating through the possible device numbers defined by the size of the file, INDEX.DEV.
- The tabulated data filename is added to the pathname supplied by the operator. The name varies as the FOR loop executes. These IF THEN statements control the filename.

```
IF I < 10 THEN BEGIN
```

```

INSERT('\BD00xL.TAB',pName,pNameLen+1);
DELETE(pName,pNameLen+6,1);
STR(I,one);
INSERT(one,pName,pNameLen+6)
END;

```

- The base filename is added to the pathname and then the device number replaces the "x" in the name. The process is repeated for increasing device numbers.

```

IF ((I > 9) AND (I < 100)) THEN BEGIN
  INSERT('\BD0xxL.TAB',pName,pNameLen+1);
  DELETE(pName,pNameLen+5,2);
  STR(I,two);
  INSERT(two,pName,pNameLen+5)
END;

```

```

IF ((I > 99) AND (I < 1000)) THEN BEGIN
  INSERT('\BDxxxL.TAB',pName,pNameLen+1);
  DELETE(pName,pNameLen+4,2);
  STR(I,thr);
  INSERT(thr,pName,pNameLen+4)
END;

```

- At this point the basic filename has been created; however, the character which describes the inductor has not been determined.

```

SEEK(index,I);
READ(index,indexrec);
one := COPY(indexrec.store,9,1);

```

- The character which describes the inductor is placed into the one character string, "one."

```

INSERT(one,pName,pNameLen+8);

```

- The letter is then inserted into the proper place in the filename.

```

ASSIGN(indextest,pName);
{$I-}
RESET(indextest);
IF IOresult = 0 THEN BEGIN
  TabFilePres[I] := 1;

```

```

CLOSE(indextest)
END
{$I+}
    • A search is made for the file. If it is found, the "TabFilePres" array
      location corresponding to the device number of the file just searched for
      is set to 1.

END;
    • This ends the FOR loop search.

LineCount := 0;
FOR I := 1 TO 999 DO
  IF TabFilePres[I] = 1 THEN BEGIN
    • A listing of the tabulated data files present on the specified path is displayed.

    FilePresent := TRUE;
    LineCount := LineCount + 1;
    WRITELN('The Tabulated file for device 'I' is present on the specified path. ');
    IF LineCount = 20 THEN BEGIN
      LineCount := 0;
      WRITE('Press <RETURN> for more files . . . ');
      READLN
    END
  END;
  IF NOT FilePresent THEN BEGIN
    • If no files were present on the specified path, this message is printed, and
      the program is halted.

    WRITELN('No tabulated files found on specified path. Program Halted. ');
    HALT
  END;
  DELETE(pName,pNameLen+1,40);
    • The filename is cleared from the pathname string, but the pathname remains.

  LineCount := device;
  WRITELN('End of Tabulated files on this directory. ');
  WRITE('Please enter the device number of the file to be sorted . . . ');

```

READLN(device);

- The operator enters the device number corresponding to the file to be sorted.
- The filename is reconstructed for the appropriate device.

IF device < 10 THEN BEGIN

INSERT('\BD00xL.TAB',pName,pNameLen+1);

DELETE(pName,pNameLen+6,1);

STR(device,one);

INSERT(one,pName,pNameLen+6)

END;

IF ((device > 9) AND (device < 100)) THEN BEGIN

INSERT('\BD0xxL.TAB',pName,pNameLen+1);

DELETE(pName,pNameLen+5,2);

STR(device,two);

INSERT(two,pName,pNameLen+5)

END;

IF ((device > 99) AND (device < 1000)) THEN BEGIN

INSERT('\BDxxxL.TAB',pName,pNameLen+1);

DELETE(pName,pNameLen+4,2);

STR(device,thr);

INSERT(thr,pName,pNameLen+4)

END;

SEEK(index,device);

READ(index,indexrec);

one := COPY(indexrec.store,9,1);

INSERT(one,pName,pNameLen+8);

- The complete pathname/filename combination is passed to the main program.

CLOSE(index)

END;

{ \*\*\*\* }

PROCEDURE ReadTabulatedData (pName : STR40; VAR Values : Rarray;

VAR MaxRecords : INTEGER);

- This procedure reads the data stored on disk in the tabulated data file into a 6 by 999 array of real numbers. The array is passed to the main program along with the number of records in the file which tell the program how much of the array contains valid information.

TYPE

DEVICETABLE = RECORD

series : INTEGER;

ForBias : REAL;

RevBias : REAL;

Vce : REAL;

Ic : REAL;

energy : REAL;

time : REAL

END;

VAR

tab : FILE OF DEVICETABLE;

tabrecord : DEVICETABLE;

going : STRING[20];

R,I : INTEGER;

one : STRING[1];

two : STRING[2];

thr : STRING[3];

BEGIN

FOR I := 1 TO 999 DO

FOR R := 1 TO 6 DO

Values[R,I] := 0;

- The array is initialized.

ASSIGN(tab,pName);

RESET(tab);

MaxRecords := FileSize(tab);

```
FOR I := 0 TO MaxRecords-1 DO BEGIN
```

- The record fields are read into the array.

```
  SEEK(tab,I);
```

```
  READ(tab,tabrecord);
```

```
  WITH tabrecord DO BEGIN
```

```
    Values[1,I+1] := ForBias;
```

```
    Values[2,I+1] := RevBias;
```

```
    Values[3,I+1] := Vce;
```

```
    Values[4,I+1] := Ic;
```

```
    Values[5,I+1] := energy;
```

```
    Values[6,I+1] := time
```

```
  END
```

```
END
```

```
END;
```

```
{ ***** }
```

```
PROCEDURE SortValues (Values : Rarray; VAR Posarray : Iarray; VAR MaxRec,key1,  
  key2,DifKey1,DifKey2 : INTEGER);
```

- This procedure sorts the data read from the tabulated data file. A maximum of two sorting keys can be used. The procedure outputs an array of real numbers 2 by 999 (thought of as 2 columns of 999 rows while the 6 by 999 array is 6 columns by 999 rows). The numbers in the first column are the minimum to maximum values of the corresponding data in the 6 by 999 array based upon the first sorting key. The numbers in the second column are the minimum to maximum values of the corresponding data in the 6 by 999 array based upon the second sorting key.
- If, for example, a series of data had 3 different values in the set of array element described by sort key 1 and there were a total of 10 data sets, then the first column in the 2 by 999 array only have three values followed by 996 elements with zeros. If there were four different values among those kept in the column in the 6 by 999 array referenced by the second sorting key, there would be four values in the first four ele-

ments of the second column of the 2 by 999 array followed by 995 elements with a zero. This is shown more clearly in Figure 19.

- With these two arrays, the printing routine can provide the necessary outputs.

	6 by 999 Array					2 by 999 Array	
1		0.2		80		0.2	3.12
2		1		4		1	4
3		5		3.12		5	7
4		1		7		0	80
5		0.2		7		0	0
6		0.2		4		0	0
7		5		80		0	0
8		1		7		0	0

Sort Key 1

Sort Key 2

The Sort Keys refer to particular rows. These are examples.

Figure 19. Example of sorting function.

VAR

LowToHigh,I,R,BrPt,PrevBrPt,ChCnt : INTEGER;

MinKey1,MinKey1A : REAL;

test,NoChange : BOOLEAN;

```

BEGIN
  FOR I := 1 TO 2 DO
    FOR R := 1 TO 999 DO
      Posarray[I,R] := 120000.0;
      • The 2 by 999 array is initialized.
    clscr;
    WRITELN('Here is a list of the possible sorting keys. ');
    WRITELN('Forward Base Current ..... 1');
    WRITELN('Reverse Base Current ..... 2');
    WRITELN('Collector-Emitter Voltage .... 3');
    WRITELN('Collector Current ..... 4');
    WRITELN('Energy at Breakdown ..... 5');
    WRITELN('Time to Breakdown ..... 6');
    WRITELN('No Sort key ..... 0');
    WRITE('Enter Sort key one . . . ');
    READLN(key1);
    WRITE('Enter Sort key two . . . ');
    READLN(key2);
    • The operator is asked what values he or she wants to sort on.
  IF key1 = key2 THEN BEGIN
    WRITELN('Keys are the same. Key 2 set to zero. ');
    key2 := 0;
  END;
  MinKey1A := -1.0E+9;
  • This value is the rising threshold of minimum values. Below this value
    the program will not search for a minimum.
  test := TRUE;
  LowToHigh := 0;
  WHILE test DO BEGIN
    • This loop will determine the unique values in the first sort key.
    LowToHigh := LowToHigh + 1;
    MinKey1 := 1.00000E+09;
    FOR I := 1 TO MaxRec DO BEGIN

```



- Each time the WHILE loop is executed a search is made for the minimum value in the sort key, but the new minimum value in each pass must be greater than the previous minimum value.

```
IF ((MinKey1 > Values[key1,I]) AND (MinKey1A < Values[key1,I])) THEN
  MinKey1 := Values[key1,I]
```

- The new minimum value above the threshold "MinKey1A" is found.

```
END;
```

```
IF MinKey1A = MinKey1 THEN test := FALSE
```

- As the WHILE loop executes, there eventually are no more new minimum values and this case will be met. At this point, the WHILE loop will be stopped.

```
ELSE BEGIN
```

```
  MinKey1A := MinKey1;
```

- The new minimum value threshold is the minimum value above the old threshold that was just found.

```
  Posarray[1,LowToHigh] := MinKey1
```

- The minimum value is placed into the 2 by 999 array at the row below the last minimum value (See Figure 15.).

```
END
```

```
END;
```

```
Posarray[1,LowToHigh-1] := 0;
```

```
DifKey1 := LowToHigh-2;
```

- The value "DifKey1" is determined. This value represents the number of rows in column one of the 2 by 999 which contain non-zero numbers. It is the number of unique data points found for sort key 1.

```
IF key2 <> 0 THEN BEGIN
```

- If there is a second sort key then the following is executed. It is exactly the same as the above except where the term "key1" is used above now "key2" is used, and the minimum values are loaded into the second column of the 2 by 999 array.

```
  MinKey1A := -1.0E+9;
```

```
  test := TRUE;
```

```
  LowToHigh := 0;
```

```
  WHILE test DO BEGIN
```

```

LowToHigh := LowToHigh + 1;
MinKey1 := 1.00E+09;
FOR I := 1 TO MaxRec DO BEGIN
  IF ((MinKey1 > Values[key2,I]) AND (MinKey1A < Values[key2,I])) THEN
    MinKey1 := Values[key2,I]
  END;
  IF MinKey1A = MinKey1 THEN test := FALSE
  ELSE BEGIN
    MinKey1A := MinKey1;
    Posarray[2,LowToHigh] := MinKey1
  END
END;
Posarray[2,LowToHigh-1] := 0;
DifKey2 := LowToHigh-2;
  • This is the number of unique data points found for the second sort key.
END;
END;

{ **** }

PROCEDURE WriteOutPut(Values : Rarray; Posarray : Iarray; MaxRec,key1,
  key2,DifKey1,DifKey2 : INTEGER);
  • This procedure is responsible for producing the output.
TYPE
  STR25 = STRING[25];
  Lab = ARRAY[1..6] OF STR25;

VAR
  going : STRING[60];
  LPT : TEXT;
  L : Lab;
  fullstr : STRING[25];
  time : STRING[9];
  P,S,T,I,R : INTEGER;

```

```
test : BOOLEAN;
```

```
BEGIN
```

```
  CLRSCR;
```

```
  L[1] := 'Forward Base Current';
```

```
  L[2] := 'Reverse Base Current';
```

```
  L[3] := 'Collector-Emitter Voltage';
```

```
  L[4] := 'Collector Current';
```

```
  L[5] := 'Energy at Breakdown';
```

```
  L[6] := 'Time at Breakdown';
```

- The above values are assign to the "L" array.

```
  WRITELN('Enter the output destination. "LPT1" routes output to line printer,');
  WRITELN('or any valid file name routes output to disk as a text file.');
```

```
  WRITE('Destination: ');
```

```
  READLN(going);
```

- The operator provides the destination to the program.

```
test := TRUE;
```

```
WHILE test DO BEGIN
```

```
{ $I- }
```

```
  ASSIGN(LPT,going);
```

```
  REWRITE(LPT);
```

```
  IF IOresult <> 0 THEN BEGIN
```

```
    WRITELN('Something is wrong with that destination. Try Again.');
```

```
    WRITE('Destination: ');
```

```
    READLN(going)
```

```
  END
```

```
  ELSE test := FALSE
```

```
{ $I+ }
```

- The output device/file is opened with error checking to be sure the device/file is ready to receive data.

```
END;
```

```
FOR P := 1 TO DifKey1 DO BEGIN
```

- The following is done iteratively for "DifKey1" times.

```
  WRITE(LPT,'For primary key of ',L[key1], ' = ');
```

```
NumToStr9(Posarray[1,P],time);
```

```
WRITE(LPT,time);
```

- The value in the first row first column of the 2 by 999 array is printed. It is written as a shortened number using scientific notation. The variable "time" is a string and the name is basically meaningless.

```
CASE key1 OF
```

```
  1 : WRITELN(LPT,' A');
```

```
  2 : WRITELN(LPT,' A');
```

```
  3 : WRITELN(LPT,' V');
```

```
  4 : WRITELN(LPT,' A');
```

```
  5 : WRITELN(LPT,' J');
```

```
  6 : WRITELN(LPT,' Sec')
```

```
END;
```

- The proper units for the first sorting key are added to the output along with a <CR> <LF>.

```
IF key2 <> 0 THEN BEGIN
```

- If there is a second sorting key, the following is done.

```
  FOR S := 1 TO DifKey2 DO BEGIN
```

- The following is repeated once for each unique value of sort key 2.

```
    test := TRUE;
```

```
    FOR T := 1 TO MaxRec DO
```

```
      IF ((Posarray[1,P] = Values[key1,T]) AND (Posarray[2,S]  
        = Values[key2,T]) AND test) THEN BEGIN
```

- A test is made for the first occurrence of a match of both sort key 1 and sort key 2 (sort key 1 is held in Posarray[1,P] and sort key 2 is held in Posarray[2,S]).

```
      test := FALSE;
```

```
      WRITELN(LPT);
```

```
      WRITE(LPT,'   For secondary key of ',L[key2],' = ');
```

```
      NumToStr9(Posarray[2,S],time);
```

```
      WRITE(LPT,time);
```

```
      CASE key2 OF
```

```
        1 : WRITELN(LPT,' A');
```

```
        2 : WRITELN(LPT,' A');
```

```

3 : Writeln(LPT,' V');
4 : Writeln(LPT,' A');
5 : Writeln(LPT,' J');
6 : Writeln(LPT,' Sec')

```

```
END;
```

- On the first occurrence of a match of sort key 1 and sort key 2, the above is printed. It is printed only once even if more matches are encountered.

```
Writeln(LPT)
```

```
END;
```

```
FOR T := 1 TO MaxRec DO BEGIN
```

```
  IF ((Posarray[1,P] = Values[key1,T]) AND (Posarray[2,S] = Values[key2,T]))
  THEN
```

- On every occurrence of a match to sort key 1 and sort key 2, the following is printed.

```
  FOR R := 1 TO 6 DO BEGIN
```

```
    IF ((R <> key1) AND (R <> key2)) THEN BEGIN
```

```
      WRITE(LPT,'      ',L[R], ' ..... ');
```

```
      NumToStr9(Values[R,T],time);
```

```
      WRITE(LPT,time);
```

```
      CASE R OF
```

```
        1 : Writeln(LPT,' A');
```

```
        2 : Writeln(LPT,' A');
```

```
        3 : Writeln(LPT,' V');
```

```
        4 : Writeln(LPT,' A');
```

```
        5 : Writeln(LPT,' J');
```

```
        6 : Writeln(LPT,' Sec')
```

```
      END
```

- The above prints the values of the 6 by 999 array that are not referenced by the sort keys.

```
    END;
```

```
  IF R = 6 THEN Writeln(LPT)
```

```
END
```

```
END
```

```

END
END
ELSE BEGIN
    • If there is no second sort key, then the following is executed.
    FOR T := 1 TO MaxRec DO BEGIN
        IF Posarray[1,P] = Values[key1,T] THEN
            FOR R := 1 TO 6 DO BEGIN
                IF R <> key1 THEN BEGIN
                    WRITE(LPT,'      'L[R],' ..... ');
                    NumToStr9(Values[R,T],time);
                    WRITE(LPT,time);
                    CASE R OF
                        1 : Writeln(LPT,' A');
                        2 : Writeln(LPT,' A');
                        3 : Writeln(LPT,' V');
                        4 : Writeln(LPT,' A');
                        5 : Writeln(LPT,' J');
                        6 : Writeln(LPT,' Sec')
                    END
                • The above writes all of the values in a given row except the one referred by sort key 1.
                END;
                IF R = 6 THEN Writeln(LPT)
            END
        END
    END
END;
CLOSE(LPT)
END;

{ **** }

BEGIN {Main Program}
    ListDevice(PathN);

```

```
ReadTabulatedData(PathN,Valuarr,Recs);  
SortValues(Valuarr,Posarr,Recs,K1,K2,DK1,DK2);  
WriteOutPut(Valuarr,Posarr,Recs,K1,K2,DK1,DK2)  
END.
```

APPENDIX M  
EDITLIST.PAS PROGRAM LISTING



PROGRAM EditListDevice;

- This program is used to create (if needed), maintain and correct the file, INDEX.DEV. It consists of a single main program body.

LABEL

loop;

CONST

first : BOOLEAN = FALSE;

TYPE

STR40 = STRING[40];

STR13 = STRING[13];

DEVICES = RECORD

DEV : INTEGER;

descrip : STR40;

book : STR13;

stor : STR13

END;

VAR

index : FILE OF DEVICES;

indexrec : DEVICES;

LineCount,series,Count : INTEGER;

test : BOOLEAN;

induct,ans : CHAR;

Three : STRING[3];

Two : STRING[2];

One,One1,One2 : STRING[1];

BEGIN

ASSIGN(index,'C:\RBSOA\INDEX.DEV');

{ \$I- }

RESET(index);

{ \$I+ }

IF IOresult <> 0 THEN BEGIN

    WRITELN('C:INDEX.DEV not found. Assumed first time used. ');

    WRITELN('Creating C:\RBSOA\INDEX.DEV.');

    REWRITE(index);

    first := TRUE

END

- The file, INDEX.DEV, is opened with error checking. If an error occurs, it is assumed the file does not exist and a new one is created.

ELSE BEGIN

- If the file exists, the following is executed.

    loop:

- The "loop:" is a LABEL. It can be used as the destination point in a GOTO. This point is the top of a control loop which allows for double checking of whatever editing is performed.
- The following provides a list of the devices stored in the file, INDEX.DEV.

    LineCount := 1;

    Count := 0;

    SEEK(index, LineCount);

- Record number one is sought.

    WHILE NOT EOF(index) DO BEGIN

- This loop will be executed until the end of file mark is reached in INDEX.DEV.

        READ(index, indexrec);

- The record at the current cursor position is read.

        WITH indexrec DO BEGIN

            WRITE(DEV, ' ');

            WRITE(descrip:40, ' ');

            WRITE(book, ' ');

            WRITELN(stor)

        END;

- The record fields are printed to the screen.

    LineCount := LineCount + 1;

- The cursor position pointer is increased by one.

Count := Count + 1;

- The count of the number of records displayed since the last pause or the beginning of the file is increased by one.

IF Count = 21 THEN BEGIN

- If there has been 21 lines displayed since the beginning or since the last pause, the program pauses to allow the operator to review the device records.

Count := 0;

WRITELN('Press <RETURN> for more devices.');

READLN

END

END;

WRITELN('End of indexed devices.')

END;

ans := 'Q';

- If this had been a new INDEX.DEV file, the following would be the first question asked.

WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO BEGIN

WRITE('Do you want to add a new device? (Y or N) . . . ');

READLN(ans)

END;

IF ((ans = 'Y') OR (ans = 'y')) THEN BEGIN

test := TRUE;

IF NOT first THEN LineCount := FileSize(index)

- If this is not the first entry, then the record pointer is set so the new device will be added to the end of the file.

ELSE BEGIN

LineCount := 1;

first := FALSE

END;

test := TRUE;

WHILE test DO BEGIN

WITH indexrec DO BEGIN

```

WRITELN('Enter the Device Description for Device Number ',LineCount);
WRITELN('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx - X"s indicate 40
      Characters.');
```

- The operators is asked to provide a 40 character description of the device.

```

DEV := LineCount;
      • The device number is set equal to the record pointer value.
WRITELN('The following is a list of inductor value codes. Please enter the ');
WRITELN('the SECOND letter corresponding to the value of the inductor being
      used.');
```

```

WRITELN('LG . . . 44  $\mu$ H');
WRITELN('LH . . . 146  $\mu$ H');
WRITELN('LI . . . 267  $\mu$ H');
WRITELN('LJ . . . 426  $\mu$ H');
WRITELN('LK . . . 1 mH');
WRITELN('LL . . . 2.16mH');
```

```

READLN(induct);
induct := UPCASE(induct);
      • The inductor value is input.
INSERT(induct,One1,1);
      • The following creates the bookkeeping and storage filenames for various
        device numbers.
IF DEV < 10 THEN BEGIN
  STR(LineCount,One2);
  book := CONCAT('G:DEV00',One2,'L',One1,'.BK');
  stor := CONCAT('G:BD00',One2,'L',One1,'.000')
END;
IF ((DEV > 9) AND (DEV < 100)) THEN BEGIN
  STR(LineCount,Two);
  book := CONCAT('G:DEV0',Two,'L',One1,'.BK');
  stor := CONCAT('G:BD0',Two,'L',One1,'.000')
END;
IF ((DEV > 99) AND (DEV < 1000)) THEN BEGIN
  STR(LineCount,Three);
```

```
book := CONCAT('G:DEV',Three,'L',One1,'.BK');
```

```
stor := CONCAT('G:BD0',Three,'L',One1,'.000')
```

```
END;
```

- The bookkeeping and storage filenames are completed.

```
WRITELN('Here is the entry for Device Number ',DEV);
```

```
WRITELN(DEV:3,' ',descrip:40,' ',book,' ',stor);
```

- The complete device record is displayed for the operator's approval.

```
END;
```

```
ans := 'Q';
```

```
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO BEGIN
```

```
  WRITE('Is the entry alright? (Y or N) . . . ');
```

```
  READLN(ans)
```

```
END;
```

```
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
```

```
  SEEK(index,LineCount);
```

```
  WRITE(index,indexrec);
```

- If the entry is correct, then it is added to the INDEX.DEV file.

```
  ans := 'Q';
```

```
  WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO
```

```
  BEGIN
```

```
    WRITE('Do you want to add another Device to the file? (Y or N) . . . ');
```

```
    READLN(ans)
```

```
  END;
```

```
  IF ((ans = 'n') OR (ans = 'N')) THEN test := FALSE
```

- If the operator does not want to add another device, then the WHILE loop is ended; otherwise, the LineCount is increased by one and the loop is executed, again.

```
  ELSE LineCount := LineCount + 1;
```

```
END
```

```
ELSE BEGIN
```

- If the device record is not correct, then . . .

```
  ans := 'Q';
```

```
  WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'N') OR (ans = 'n')) DO
```

```
  BEGIN
```

```

WRITE('Do you want to try again? (Y or N) . . . ');
READLN(ans)
END;

```

- A "Y" answer will automatically re-execute the loop.

```

IF ((ans = 'n') OR (ans = 'N')) THEN HALT
END
END

```

- End of the WHILE loop to add a record.

```

END;
ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO BEGIN
  WRITE('Do you want to edit any of these devices? (Y or N) . . . ');
  READLN(ans)
END;

```

- The program now gives the operator the option to edit any of the device records. The device description can be altered, and the series number can be changed, only.

```

test := FALSE;
IF ((ans = 'y') OR (ans = 'Y')) THEN REPEAT
  WRITE('Enter the device number (Integer Value Only!) . . . ');
  READLN(LineCount);
  SEEK(index,LineCount);
  READ(index,indexrec);

```

- The record for the device of interest is entered and the appropriate record is retrieved.

```

WITH indexrec DO BEGIN
  WRITELN('Here is the current device description. ');
  WRITELN(descrip);
  ans := 'Q';
  WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO BEGIN
    WRITE('Do you want to change this entry? (Y or N) . . . ');
    READLN(ans)
  END;

```

- The operator is shown the device description and asked if a change is desired.

```
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
  WRITELN('Enter a new device description. ');
  WRITELN('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx - X"s show 40
    Characters');
  READLN(descrip)
END;
```

- If the description is to be changed, then the operator supplies the new description here.
- The following concerns the correction/alteration of the series number in the storage filename.

```
WRITELN('Here is the current storage file, ',stor);
ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO BEGIN
  WRITE('Do you want to change the series number for this entry? (Y or N). . . ');
  READLN(ans)
END;
IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
  WRITE('Enter the new series number (Integer Value Less Than 1000 Only) . . . ');
  READLN(series)
END;
```

- The current series name is displayed, and the operator is asked to verify the series number. If the number is not correct and the operator wants to change it, he or she is asked to supply a new series number.
- The following places the new series number into the storage filename.

```
DELETE(stor,11,3);
INSERT('000',stor,11);
IF series < 10 THEN BEGIN
  STR(series,One);
  INSERT(One,stor,13)
END;
IF ((series > 9) AND (series < 100)) THEN BEGIN
  STR(series,Two);
```

```

    INSERT(Two,stor,12)
END;
IF ((series > 99) AND (series < 1000)) THEN BEGIN
    STR(series,Three);
    INSERT(Three,stor,11)
END;

```

- The new series number is in place.

```

IF series > 999 THEN BEGIN
    WRITELN('Series value greater than 1000. Program Halted. ');
    HALT
END;

```

- If the series number supplied by the operator is 1000 or greater, the program halts with no change made to the device record.

```

CLRSCR;
WRITELN('Here is the new device entry. ');
WRITELN(DEV, ' ',descrip, ' ',book, ' ',stor);

```

- The revised device entry is displayed for the operator's approval.

```

ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO BEGIN
    WRITE('Is this entry correct? (Y or N) . . . ');
    READLN(ans)
END;

```

```

IF ((ans = 'y') OR (ans = 'Y')) THEN BEGIN
    SEEK(index,LineCount);
    WRITE(index,indexrec)
END

```

```

ELSE WRITELN('No changes made. ');

```

- If the entry is correct, then it is incorporated in the file; otherwise, nothing is done.

```

ans := 'Q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO BEGIN
    WRITE('Edit another entry? (Y or N) . . . ');
    READLN(ans)
END;

```



IF ((ans = 'n') OR (ans = 'N')) THEN test := TRUE

- If the operator is done editing, and negative response drops him or her from the editing loop.

ELSE BEGIN

ans := 'Q';

WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO

BEGIN

WRITE('See list again? (Y or N) . . . ');

READLN(ans)

END;

IF ((ans = 'y') OR (ans = 'Y')) THEN GOTO loop

END

- If the operator wants to edit something else, he or she is also given the opportunity to see the list, again.

END;

UNTIL test;

END.

APPENDIX N  
PRCSLTWO.PAS PROGRAM LISTING

PROGRAM Process\_Level\_Two; {For use as a stand alone program in conjunction with FASTDATA.BAT as the follow on processing step}

- This program is to be used to calculate or recalculate the Instantaneous Power and Energy. It is for use in case of a program crash or in case PROCESS2 was not executed in the data acquisition cycle in an attempt to speed the process up. (This second case is not recommended.)

TYPE

```
ST10 = STRING[10];
STR13 = STRING[13];
STR80 = STRING[80];
datarray = ARRAY[1..3,1..2048] OF INTEGER;
ene = ARRAY[1..2047] OF REAL;
```

VAR

```
EnergyPass : ene;
tnum,device : INTEGER;
storage : STR13;
```

```
{*****}
```

PROCEDURE listdevice(VAR stor : STR13);

- This procedure returns the basic storage filename for the data to be used as input and for the output destination.

TYPE

```
STR40 = STRING[40];
STR13 = STRING[13];
DEVICES = RECORD
    DEV : INTEGER;
    descrip : STR40;
    book : STR13;
    store : STR13
END;
```

VAR

```

index : FILE OF DEVICES;
indexrec : DEVICES;
error, LineCount : INTEGER;
name : STRING[18];
one : STRING[1];
two : STRING[2];
thr : STRING[3];
test : BOOLEAN;
ans : CHAR;
I, Count, series, device : INTEGER;

```

BEGIN

```

name := 'C:\RBSOA\INDEX.DEV';
ASSIGN(index, name);
RESET(index);
WRITELN('You will be asked for the device number corresponding to the storage file');
WRITELN('for the data taken on the device of interest. You will also be asked for');
WRITELN('the storage series number which is the last three numbers in the file name. ');
WRITELN('The value shown in the listing is the highest currently stored. ');
ans := 'q';
WHILE NOT ((ans = 'y') OR (ans = 'Y') OR (ans = 'n') OR (ans = 'N')) DO
  BEGIN
    WRITELN('Do you want to see a listing of all currently cataloged devices?');
    WRITE('See list? (Y or N) . . . ');
    READLN(ans)
  END; { WHILE NOT ((ans = 'y'...}

```

- If the operator wants to see a listing the following is executed; otherwise, the next section is skipped. See PLOTDATA or TABULATE for a description of this section.

```

IF ((ans = 'Y') OR (ans = 'y')) THEN BEGIN
  LineCount := 1;
  Count := 0;

```

```

SEEK(index,LineCount);
WHILE NOT EOF(index) DO BEGIN
  READ(index,indexrec);
  WITH indexrec DO BEGIN
    WRITE(DEV,' ');
    WRITE(descrip:40,' ');
    WRITE(book,' ');
    WRITELN(store)
  END;
  LineCount := LineCount + 1;
  Count := Count + 1;
  IF Count = 21 THEN BEGIN
    Count := 0;
    WRITELN('Press <RETURN> for more devices. ');
    READLN
  END
END;
WRITELN('End of indexed devices.')
END; { IF ((ans = 'Y'...) }
test := TRUE;
WHILE test DO BEGIN
  WRITE('What is the device number? (Integer Value Only!) . . . ');
  READLN(device);
  IF device < 1000 THEN test := FALSE
  ELSE WRITELN('Integer too large. Try again.')
END;

```

- The operator is asked to supply the device number. Some error checking is performed.

```

test := TRUE;
WHILE test DO BEGIN
  WRITE('What is the series number? (Integer Value Only!) . . . ');
  READLN(series);
  IF series < 1000 THEN test := FALSE
  ELSE WRITELN('Integer too large. Try again.')

```

END;

- The operator is asked to supply the series number of the particular test run to be processed. Some error checking is performed.

SEEK(index,device);

READ(index,indexrec);

WITH indexrec DO

stor := store;

- The appropriate record is obtained and the storage filename is place in the string variable, "stor."

CLOSE(index);

DELETE(stor,1,1);

INSERT('A',stor,1);

- The drive designation is changed from G: to A:.

DELETE(stor,11,3);

INSERT('000',stor,11);

IF series < 10 THEN BEGIN

STR(series:1,one);

INSERT(one,stor,13)

END;

IF ((series > 9) AND (series < 100)) THEN BEGIN

STR(series:2,two);

INSERT(two,stor,12)

END;

IF ((series > 99) AND (series < 1000)) THEN BEGIN

STR(series:3,thr);

INSERT(thr,stor,11)

END

- The new series number is inserted into "stor."

END; {Procedure listdevice}

{ \*\*\*\*\* }

PROCEDURE Calculate (stor : STR13; VAR energy : ene);

- This is exactly the same procedure as that used in PROCESS2. See that program for more information.

VAR

```
R,I,Vmax,VstartPos,PstopPos : INTEGER;
going : STRING[14];
VIP : datarray;
incoming : CHAR;
Pmax,Pstop,eMax,Vstart,VscP,VI,VV,HscA,HscB,probeA,probeB,fBias,rBias,
intmed1,intmed2 : REAL;
store : TEXT[25];
```

BEGIN

```
eMax := 0;
FOR R := 1 TO 2047 DO energy[R] := 0;
going := stor;
INSERT('A',going,10);
ASSIGN(store,going);
RESET(store);
READLN(store);
READLN(store,fBias,rBias);
READLN(store,VV,HscA,probeA);
FOR R := 1 TO 2048 DO BEGIN
  READ(store,incoming);
  I := ORD(incoming);
  VIP[1,R] := I - 127
END;
CLOSE(store);
going := stor;
INSERT('B',going,10);
ASSIGN(store,going);
RESET(store);
READLN(store,fBias,rBias);
READLN(store,VI,HscB,probeB);
```

```

IF HscA <> HscB THEN BEGIN
  WRITELN('Horizontal scale factors not equal. Program Halted.');
```

    HALT

```

END;
FOR R := 1 TO 2048 DO BEGIN
  READ(store,incoming);
  I := ORD(incoming);
  VIP[2,R] := I - 127
END;
CLOSE(store);
VscP := probeA*probeB*VV*VI/1024;
FOR R := 1 TO 2048 DO
  VIP[3,R] := VIP[1,R]*VIP[2,R];
Vmax := -128;
FOR R := 1 TO 2048 DO
  IF Vmax < VIP[1,R] THEN Vmax := VIP[1,R];
Vstart := 0.05*Vmax;
VstartPos := 1;
WHILE Vstart > VIP[1,VstartPos] DO
  VstartPos := VstartPos + 1;
Pmax := -100000.00;
FOR R := 1 TO 2048 DO
  IF Pmax < VIP[3,R] THEN Pmax := VIP[3,R];
Pstop := 0.03*Pmax;
PstopPos := 1;
WHILE Pmax <> VIP[3,PstopPos] DO
  PstopPos := PstopPos + 1;
WHILE Pstop < VIP[3,PstopPos] DO
  PstopPos := PstopPos + 1;
PstopPos := PstopPos + 1;
FOR R := 1 TO 2047 DO
  energy[R] := 0;
FOR R := VstartPos TO PstopPos DO BEGIN
  IF (VIP[3,R]+VIP[3,(R+1)]) < (ABS(VIP[3,R])+ABS(VIP[3,(R+1)])) THEN BEGIN
```



```

IF VIP[3,R] > VIP[3,(R+1)] THEN BEGIN
    intmed1 := VscP*VIP[3,R];
    intmed2 := VscP*VIP[3,(R+1)];
    eMax := eMax + (0.5*SQR(intmed1)*HscA/(ABS(intmed1)+ABS(intmed2)))
    - 0.5*SQR(intmed2)*HscA/(ABS(intmed1)+ABS(intmed2))
END;
IF VIP[3,R] < VIP[3,(R+1)] THEN BEGIN
    intmed1 := VscP*VIP[3,R];
    intmed2 := VscP*VIP[3,(R+1)];
    eMax := eMax + (0.5*SQR(intmed1)*HscA/(ABS(intmed1)+ABS(intmed2)))
    + 0.5*SQR(intmed2)*HscA/(ABS(intmed1)+ABS(intmed2))
END
END
ELSE BEGIN
    IF ABS(VIP[3,R]) > ABS(VIP[3,(R+1)]) THEN
        eMax := eMax + VscP*(VIP[3,(R+1)]*HscA + 0.5*HscA*(VIP[3,R]
            -VIP[3,(R+1)]));
    IF ABS(VIP[3,R]) = ABS(VIP[3,(R+1)]) THEN
        eMax := eMax +VscP*VIP[3,R]*HscA;
    IF ABS(VIP[3,R]) < ABS(VIP[3,(R+1)]) THEN
        eMax := eMax + VscP*(VIP[3,R]*HscA + 0.5*HscA*(VIP[3,(R+1)]-VIP[3,R]))
    END;
    energy[R] := eMax
END;
going := stor;
going[1] := 'A';
INSERT('P',going,10);
ASSIGN(store,going);
REWRITE(store);
WRITELN(store,fBias,rBias);
WRITELN(store,eMax,HscA,VscP);
FOR R := 1 TO 2048 DO
    WRITE(store,VIP[3,R], ' ');
CLOSE(store)

```

END;

{\*\*\*\*\*}

PROCEDURE WriteToEnergy(stor : STR13; energy : ene);

- This is exactly the same procedure as that used in PROCESS2. See that program for more information.

VAR

going : STRING[14];

store : TEXT;

R : INTEGER;

fullstr : STRING[25];

outstr7 : STRING[7];

outstr5 : STRING[5];

outstr : STRING[12];

BEGIN

stor[1] := 'A';

going := stor;

INSERT('E',going,10);

ASSIGN(store,going);

REWRITE(store);

FOR R := 1 TO 2047 DO

IF energy[R] = 0 THEN WRITELN(store,'0')

ELSE BEGIN

STR(energy[R],fullstr);

outstr7 := COPY(fullstr,2,7);

outstr5 := COPY(fullstr,19,5);

outstr := CONCAT(outstr7,outstr5);

WRITELN(store,outstr)

END;

CLOSE(store)

END;

{ \*\*\*\*\* }

BEGIN {MAIN PROGRAM}

listdevice(storage);

Calculate(storage,EnergyPass);

WriteToEnergy(storage,EnergyPass)

END.

PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copy-right infringement.

Disagree (Permission not granted)

Agree (Permission granted)

\_\_\_\_\_  
Student's signature

John R. Ottain  
\_\_\_\_\_  
Student's signature

\_\_\_\_\_  
Date

4 December 1987  
\_\_\_\_\_  
Date