TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.
FHWA/TX+464-1F

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle
ESTIMATING RESIDUAL FATIGUE LIFE OF BRIDGES

5. Report Date
March 1988

6. Performing Organization Code

7. Author(s)
G. Jeff Post, Karl H. Frank,
and Bahram (Alec) Tahmassebi

8. Performing Organization Report No.
Research Report 464-1F

9. Performing Organization Name and Address
Center for Transportation Research
The University of Texas at Austin
Austin, Texas 78712-1075

10. Work Unit No.

11. Contract or Grant No.
Research Study 3-5-86-464

12. Sponsoring Agency Name and Address
Texas State Department of Highways and Public Transportation; Transportation Planning Division
P. O. Box 5051
Austin, Texas 78763-5051

13. Type of Report and Period Covered
Final

14. Sponsoring Agency Code

15. Supplementary Notes
Study conducted in cooperation with the U. S. Department of Transportation, Federal Highway Administration
Research Study Title: "Estimating Residual Fatigue Life of Bridges"

16. Abstract

This manual describes the capabilities and operating procedures for an automated bridge testing system. The system was developed for the Texas State Department of Highways and Public Transportation to provide a portable, self-contained, and user-friendly means for evaluating the residual fatigue life of steel girder bridges. The bridge testing system has been designed so that it can be easily installed on a bridge in less than a day and can record data automatically for up to two weeks. The system has been enclosed to protect the electronic components from the environment and the entire system can be clamped onto a bridge girder.

The main components of the system are a Campbell Scientific eight channel datalogger and a Data General portable computer. The Data General computer is used to program the Campbell to record strains measured using conventional strain gages or special clamp-on strain transducers. The system is very flexible with respect to the types of data that can be collected, and programs have also been written to analyze the data.

This user's manual has been written to provide the information required to conduct a bridge test.

17. Key Words
automated, testing, residual fatigue life, steel girder bridges, data, strain gages, transducer, Campbell Scientific eight channel datalogger

18. Distribution Statement
No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	118	

Form DOT F 1700.7 (8-69)

ESTIMATING RESIDUAL FATIGUE LIFE OF BRIDGES

by

G. Jeff Post, Karl H. Frank and Bahram (Alec) Tahmassebi

Research Report No. 464-1F

Research Project 3-5-86-464

"Estimating Residual Fatigue Life of Bridges"

Conducted for

Texas
State Department of Highways and Public Transportation

In Cooperation with the
U.S. Department of Transportation
Federal Highway Administration

by

CENTER FOR TRANSPORTATION RESEARCH
BUREAU OF ENGINEERING RESEARCH
THE UNIVERSITY OF TEXAS AT AUSTIN

March 1988

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the Federal Highway Administration. This report does not constitute a standard, specification, or regulation.

There was no invention or discovery conceived or first actually reduced to practice in the course of or under this contract, including any art, method, process, machine, manufacture, design or composition of matter, or any new and useful improvement thereof, or any variety of plant which is or may be patentable under the patent laws of the United States of America or any foreign country.

PREFACE

Often when widening an existing highway or rehabilitating an older roadway, the bridge superstructure does not satisfy the existing AASHTO fatigue design requirements. The superstructure often shows no signs of distress. The decision is often made to replace the existing bridge due to its fatigue insufficiency. A more rational procedure to judge the remaining life of the bridge may show the existing structure can remain in service. The resulting savings may be considerable. In addition, an accurate fatigue life assessment method will allow the replacement of deficient bridges to be scheduled in an orderly and efficient manner based on the estimated remaining service life.

The determination of the residual fatigue life of an in service bridge requires an accurate means of estimating the stress history of the bridge. The procedures used in design of new bridges do not lend themselves to accurate prediction of fatigue lives. The design procedures do provide conservative designs by using simplified vehicle load distribution factors, impact fractions, and design load placements. Measured live load stresses are normally much less than the calculated design stresses. A more accurate fatigue life prediction can be made using the live load stresses measured on the bridge during normal traffic conditions.

The equipment and analysis procedures presented in this report are designed to allow the fatigue life of existing bridges to be determined based on measured stress. The equipment uses state of the art fatigue life assessment techniques. The equipment is designed to be used without extensive training of the users, to be rugged, and reliable.

The report is written in the form of a user's manual of the equipment developed in the study.

SUMMARY

The research effort produced a portable and easy to use computer based system to measure the fatigue stresses on a highway bridge. The equipment is light weight, protected from the elements, and powered by ordinary rechargeable storage batteries. The equipment is housed in weather proof aluminum boxes which mount directly to the bridge using ordinary C-clamps. The equipment is designed to stay in place on the bridge unattended during the data collection period. A portable lap-top type computer is used to program the data collection hardware and to retrieve the data in the field. The process is menu driven using software developed in the project. The system may also be used to capture the live load stress history of a single vehicle. This was done during the field testing of the unit to measure the stresses produced by a large special permit vehicle. The resulting stress time history of up to eight measurement point on the bridge can be viewed within minutes after the passage of the vehicle.

The fatigue analysis procedures use rainflow counting to determine the effective constant amplitude stress range on the bridge. The fatigue life assessment uses the stress range-cycle relationships of the standard AASHTO fatigue detail categories. Software to preform remaining life calculations was developed and sample calculations presented.

The equipment can use both clamp-on type strain gage transducers and single strain gages bonded to the bridge element to measure the stresses in the bridge. Calibration procedures and calibration equipment for the clamp-on transducers were developed. Individual bridge completion boxes are used for quarter bridge strain gages. All field wiring is done with prefabricated silicone covered wires with mating female and male connectors installed. Due to the modular arrangement of the wiring and the use of clamp-on transducers, the system may be installed on a typical bridge in less than four hours. The equipment has shown excellent reliability in field tests under both rain and extreme heat. The fatigue stress data collection of up to eight locations on the bridge is performed twenty four hours per day for a week or more.

IMPLEMENTATION

The equipment developed in this project provides a simple and reliable method to determine the fatigue damage occurring in a bridge due to service stresses. The equipment along with the analysis techniques presented in the report provide the user with the ability to accurately assess the estimated time to visible fatigue cracking. The use of the equipment and analysis techniques will allow the continued use of some structures which do not satisfy the current AASHTO fatigue design requirements. This will eliminate the unnecessary replacement of these structures.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE
INTRODUCTION

This manual describes the capabilities and operating procedures for an automated bridge testing system. The system was developed for the Texas State Department of Highways and Public Transportation to provide a portable, self-contained, and user-friendly means for evaluating the residual fatigue life of steel girder bridges. The bridge testing system has been designed so that it can be easily installed on a bridge in less than a day and can record data automatically for up to two weeks. The system has been enclosed to protect the electronic components from the environment and the entire system can be clamped onto a bridge girder.

The main components of the system are a Campbell Scientific eight channel datalogger and a Data General portable computer. The Data General computer is used to program the Campbell to record strains measured using conventional strain gages or special clamp-on strain transducers. The system is very flexible with respect to the types of data that can be collected. The Campbell can be programmed to record data continuously while a truck of known weight crosses a bridge and this data can be used to check analysis results. The Campbell can also be programmed to record and count stress cycles using the rainflow method for use in fatigue analysis. Other types of special tests can also be set up. Programs have also been written to analyze the data.

This user's manual has been written to provide the information required to conduct a bridge test. Chapter 2 describes the equipment that makes up the bridge testing system and Chapter 3 contains the procedures needed to prepare for a bridge test and to set up the equipment. Chapter 4 describes the use of the programs written for conducting a test and Chapter 5 discusses the program written for analyzing the test results. The appendices contain program listings and additional detailed information. Further information on the operation of the system can be found in the Campbell Scientific and Data General manuals.

CHAPTER TWO
BRIDGE TESTING EQUIPMENT

2.1 Equipment List

The following major equipment comprises the bridge testing system.

- Campbell Scientific 21X Micrologger
- aluminum box for micrologger
- two Stowaway 12 volt batteries
- two aluminum battery boxes
- Data General (DG) One computer
- computer carrying case
- five strain transducers
- five strain gage completion boxes
- carrying case for transducers and completion boxes
- 12 - 50 foot cables
- 4 - 90 foot cables
- 30 three inch C-clamps
- tool box for C-clamps
- transducer calibration specimen

2.2 Equipment Description

This section contains a general description of the major pieces of equipment. More detailed specifications for the equipment can be found in Appendix C.

2.2.1 **Campbell 21X Box.** The Campbell 21X Micrologger is mounted in an aluminum box which provides protection for the 21X during deployment. The box also provides connectors which greatly simplify the field hook-up of transducers, strain gages, and batteries to the Campbell. The box is constructed of 3/16-in. thickness aluminum and has outside dimensions of 25 x 10 x 11-in. and is shown in Figure 2.1. The top of the box is bolted on and is oversized to provide for ventilation.

The left side of the box has eight connectors corresponding to the eight input channels of the Campbell. Figure 2.2 shows the channel number corresponding to each connector. The right side of the box is equipped with two battery connectors and a connector for the Data General computer. The arrangement of these connectors is shown in Figure 2.3. A voltmeter is also provided to indicate the voltage level of the battery(s) connected to the Campbell. The small button to the left of the voltmeter is used to activate the meter. The small light below the voltmeter indicates whether or not the Campbell is currently taking data. If the light flashes when the button next to it is pushed, then the Campbell is taking data.

Also included in the Campbell box is a small black communication box which is required to establish communication between the Campbell and the DG computer. A schematic of the wiring arrangement in the Campbell box is shown in Figure 2.4. The specifications and complete operating instructions for the Campbell can be found in the Campbell operator's manual.

2.2.2 **Battery Boxes.** The two battery boxes are made of aluminum and have construction similar to the Campbell box. The outside dimensions of the box are 20 x 14 x 11-in. and the box is shown in Figure 2.5. The batteries used are Stowaway 12 volt sealed marine batteries rated at 154 amp-hours. A plastic battery box is mounted in the aluminum box to hold the battery in place and to contain any spilled battery acid in the event of a leak. The batteries must be slow charged and can be fully charged in about 12 hours using a 15 amp charger. A 1 amp fuse has been wired into the battery cable to protect the Campbell.

2.2.3 **Data General Computer and Case.** A lightweight carrying case made by the Zero Corporation has been provided for the Data General/One, Model Two portable computer. The Zero carrying case is filled with foam that has been cut to hold the DG firmly in place during transportation. The computer has 256 kilobytes of memory and uses the MS-DOS operating system. The computer can run off of an internal battery or from an external 7.5 volt power supply. The computer and carrying case are shown in Figure 2.6.

2.2.4 **Transducers, Strain Gage Completion Boxes, and Case.** Five clamp on strain transducers and five strain gage completion boxes have been provided. Figure 2.7 shows the Zero carrying case which has been provided for storing the transducers and

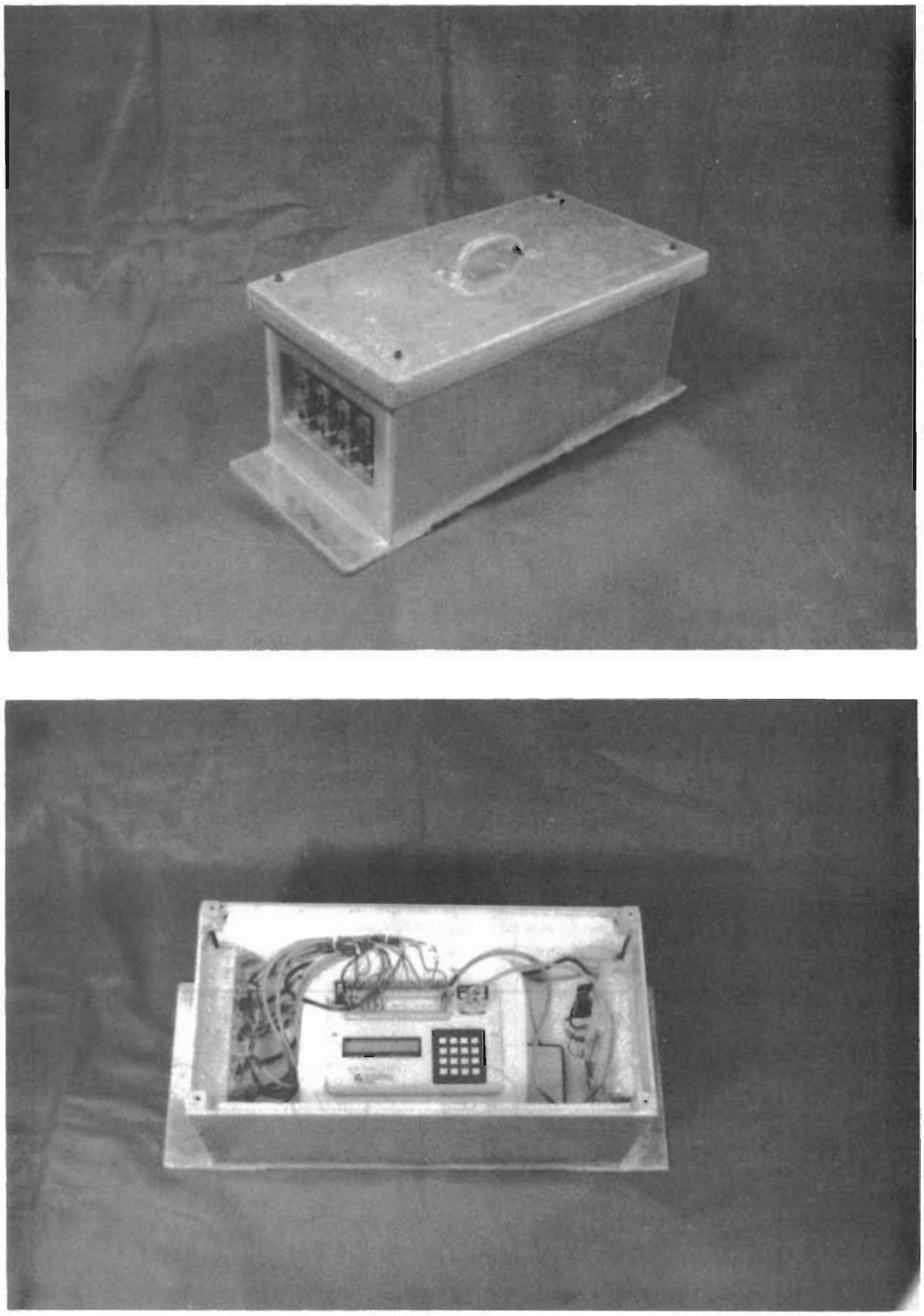


FIGURE 2.1 Campbell 21X Box

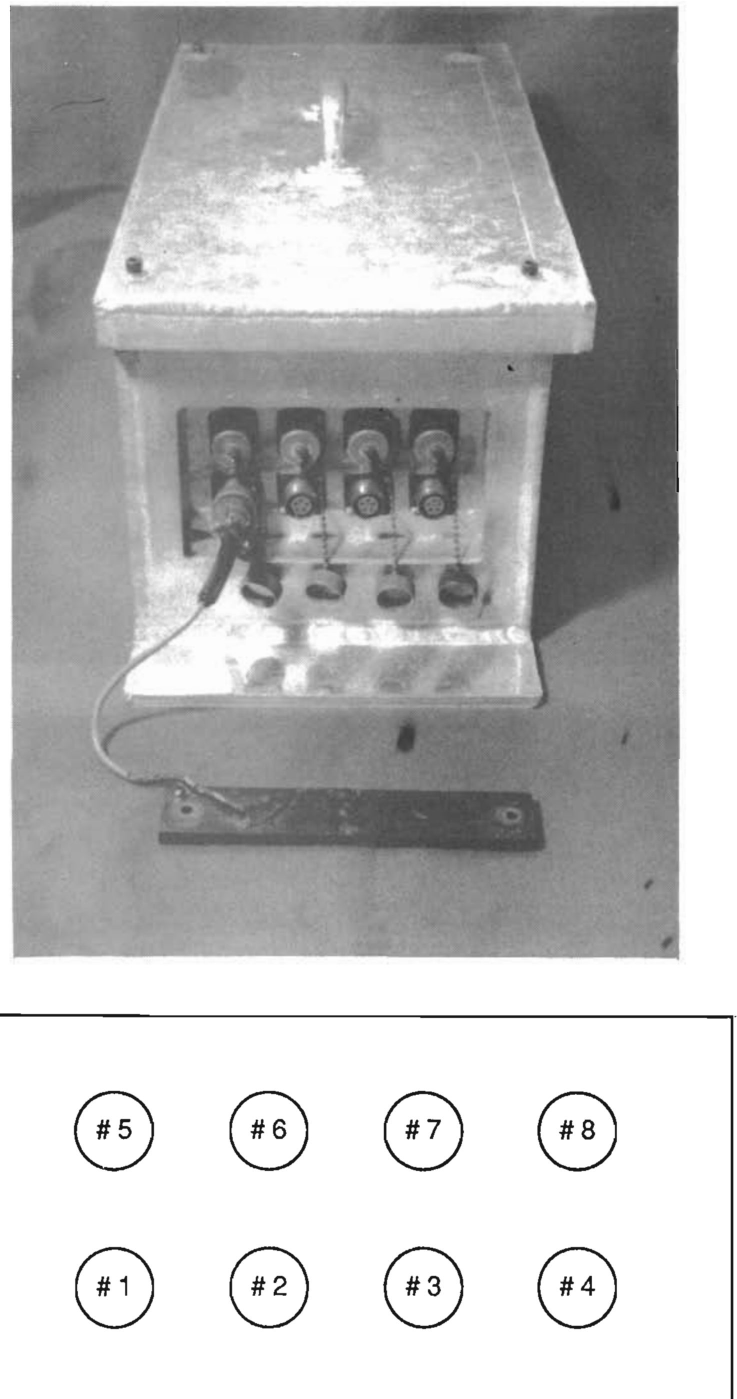
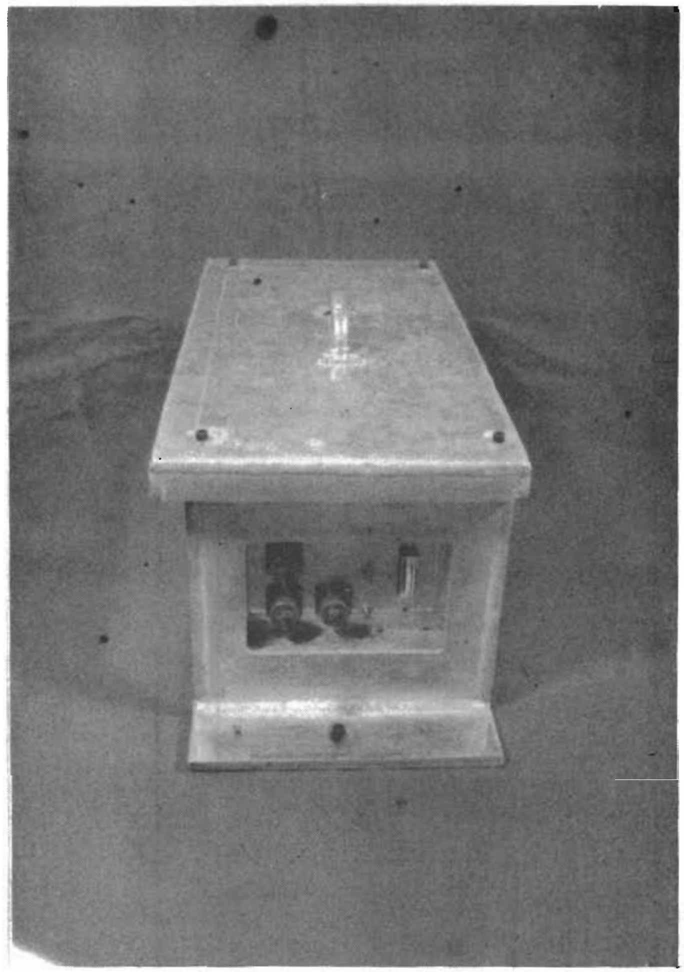


FIGURE 2.2 Campbell Box Connector Numbers



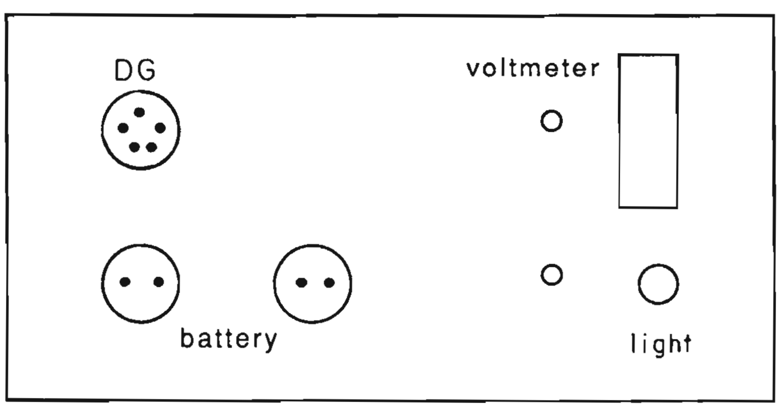


FIGURE 2.3 Right Side of Campbell Box

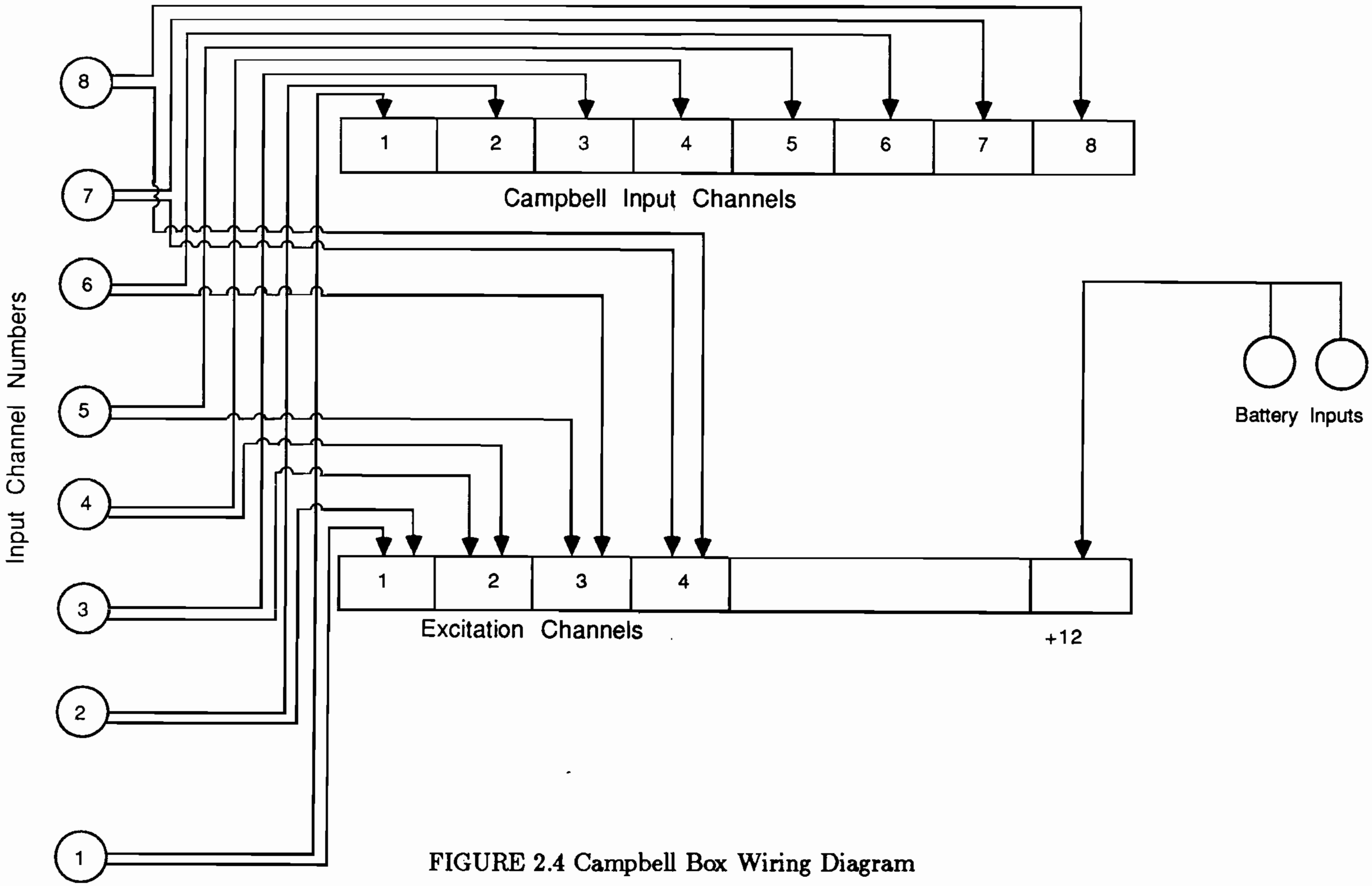


FIGURE 2.4 Campbell Box Wiring Diagram

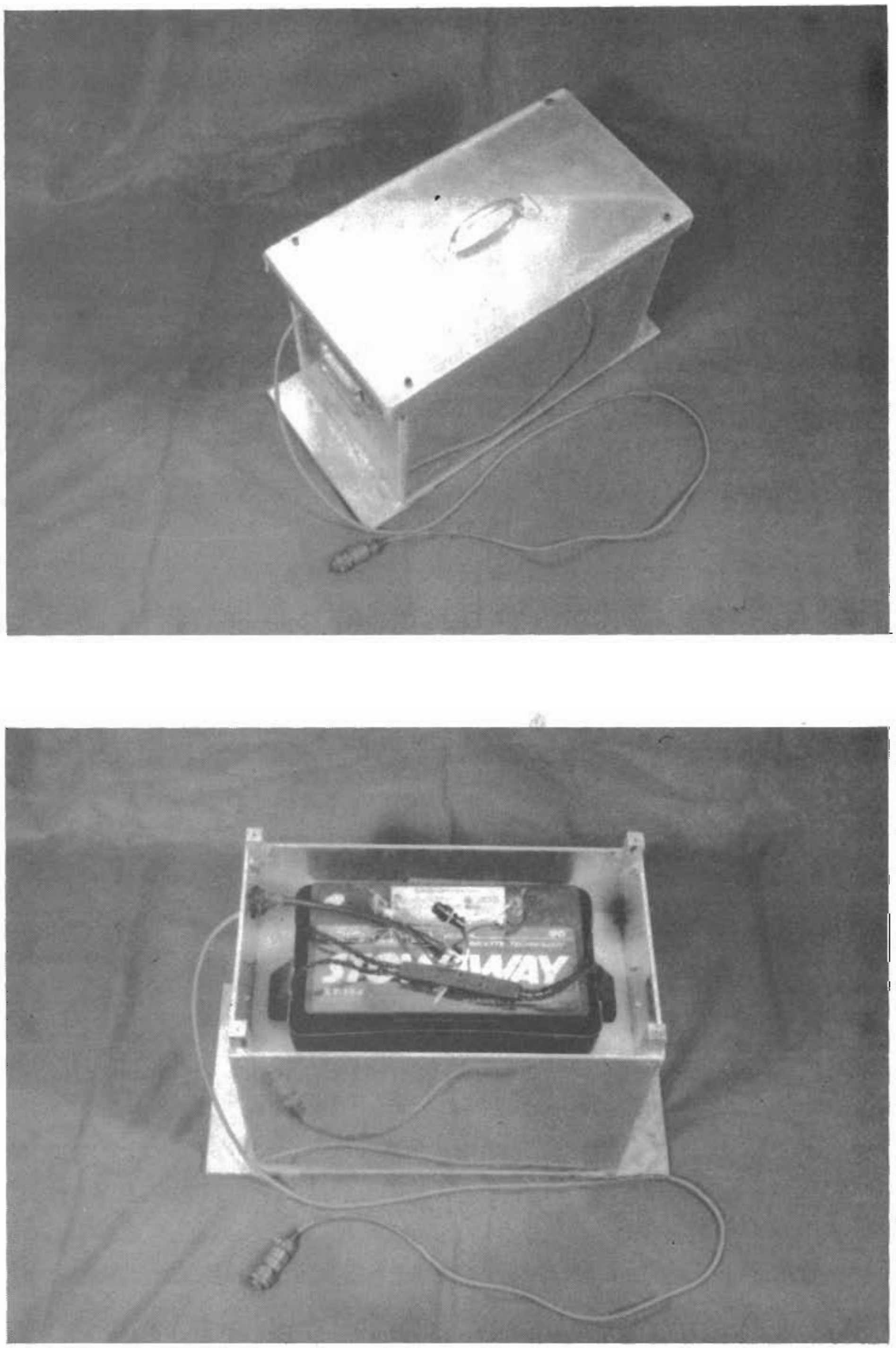


FIGURE 2.5 Battery Box

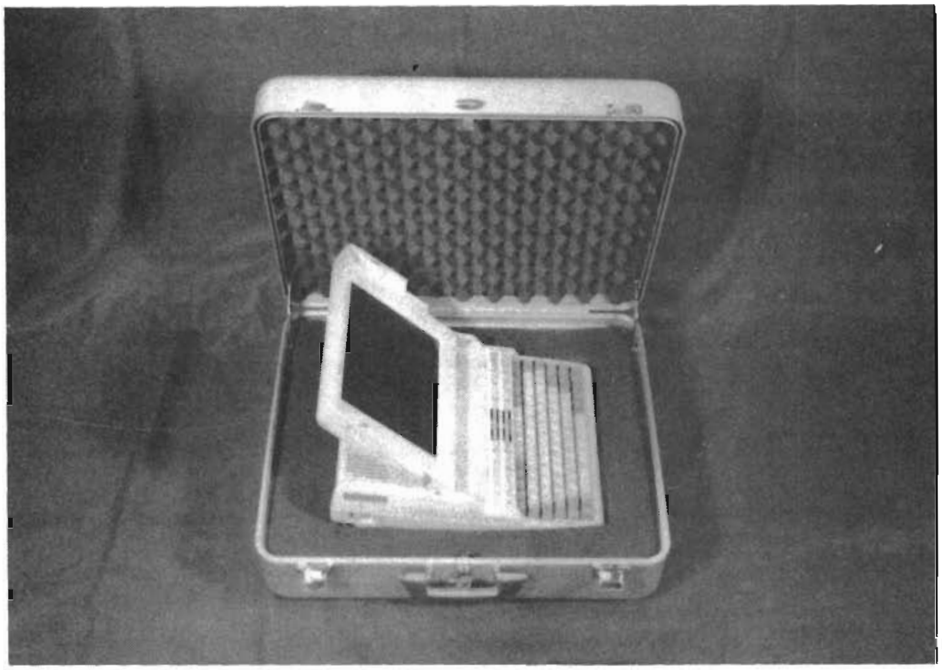


FIGURE 2.6 Data General Computer

FIGURE 2.7 Transducers, Completion Boxes, and Case

completion boxes. The transducers were manufactured by Bridge Weighing Systems, Inc. The transducers include four 350 ohm strain gages wired in a full bridge configuration and provide a mechanical amplification of approximately 7.5. The strain gage completion boxes contain three 120 ohm resistors for use with 120 ohm strain gages. The resistors are manufactured by Micro-Measurements and are guaranteed to have a resistance within .01% of 120 ohms. The completion boxes have been sealed to protect the circuit from moisture and should not be opened unless repairs are necessary.

2.2.5 **C-Clamps and Tool Box**. C-clamps are used to fasten the Campbell and battery boxes to the bridge girder. They are also used to clamp down the transducers and completion boxes. Thirty 3 inch clamps have been provided for this purpose. A tool box has also been provided for storing and carrying the clamps. Space is also available in the tool box for additional tools as required.

2.2.6 **Cables and Connectors**. Approximately 1000 feet of cable in various lengths has been provided to connect the instrumentation to the Campbell. The cable is manufactured by Belden and is insulated with teflon and has a silicon jacket. All of the cable is 4 wire except for the strain gage completion boxes which use 3 wire cable to connect to the strain gages. The connectors are standard Amphenol connectors. The transducers, strain gages, and DG computer use 5 pin connectors and the 12 volt batteries use 2 pin connectors. All of the connectors are sealed to prevent shorting due to moisture.

2.2.7 **Transducer Calibration Specimen**. The calibration specimen has been fabricated to fit in a standard tensile testing machine and can be used to calibrate the strain transducers. The specimen is shown in Figure 2.8 and is made of .375-inch thick A514 steel (100 ksi yield strength). The total length is 37 inches and holes have been drilled on the neck of the specimen for bolting on the transducers. The width of the neck is 2 inches and the cross sectional area is .75 in.^{2}. Strain gages have been mounted on both sides of the specimen and these can be used to measure the applied stress if desired. The strain gages have a resistance of 120 ohms ($\pm$.15%) and have a gage factor of 2.04. The procedures required for using the specimen to calibrate the transducers are given in Section 3.2.

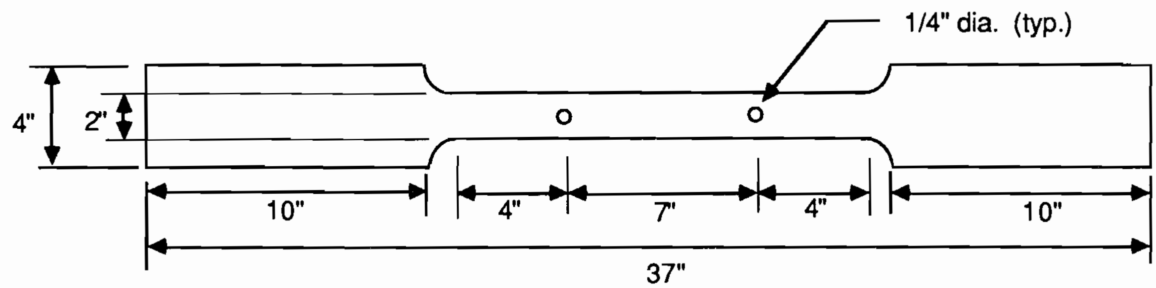


Fig. 2.8 Transducer Calibration Specimen

CHAPTER THREE
TEST PREPARATION

Prior to conducting a bridge fatigue test proper preparation is required to help insure that quality data is obtained. The major tasks involved in test preparation are the development of a test plan, calibration of the strain transducers, assembling and setting up the required equipment, and arranging for access to the bridge.

3.1 Test Plan

The initial step required in preparing for a bridge test is to clearly define a test plan. The test plan should identify the locations to be instrumented, the fatigue category of each location, the type and number of instruments (strain gages or transducers) at each location, the length of the test, and whether continuous or rainflow data is to be taken. A test plan must be developed which is within the limitations of the data acquisition system and which gives a clear picture of the fatigue condition of the bridge. Figure 3.1 is a data sheet which can be used in putting together a test plan.

The data acquisition system has been developed to be as flexible as possible, but the following limitations must be considered when developing the test plan.

- Number of Data Channels

  The maximum number of channels of data that can be taken at one time is eight. Any combination of strain gages and transducers can be used.

- Maximum Test Length

  The maximum test length depends on whether continuous or rainflow data is taken and is controlled by the memory capacity of the equipment. When taking rainflow data, the maximum test length depends on the rainflow period being used. The rainflow period is selected by the user and is discussed in Section 4.4. The rainflow period can range from one minute to one day. When rainflow data is taken on all eight channels, the maximum number of rainflow periods for which data can be taken is 18. For example, if a rainflow period of 24 hours is specified, then 18 days of data can be taken and stored. If the Campbell is allowed to

BRIDGE TEST DATA SHEET

Bridge Location : ____

Test Dates - Start: __________

Finish: __________

Instrumentation Description

Loc. #	Description	Fatigue Category	Instrument Type	Channel No.	Notes

Description Files

Channel Description File : ____________

Zero Description File : ____________

Rainflow Description File : ____________

Single Truck Tests

Data Files :

1. __________ .STK Notes: ______________________________
2. __________ .STK Notes: ______________________________
3. __________ .STK Notes: ______________________________
4. __________ .STK Notes: ______________________________

Rainflow Test

Rainflow Period : __________

Start Date : ____________

Time : ____________

Stop Date : ____________

Time : ____________

Data File : __________ .RFL

FIGURE 3.1 Bridge Test Data Sheet

take data for 19 days, then the first day of data will be overwritten and only the data for the last 18 days will be recoverable.

If a greater number of rainflow periods are desired, then the number of channels of data being taken must be reduced. The maximum number of rainflow periods can be calculated for a given number of data channels using the following formula:

$$\text{number of rainflow periods} = \frac{14{,}556}{(\textit{number of channels})(100)+3}$$

If the total length of time for which data is going to be taken exceeds 18 days, then the battery life needs to be considered (See section 3.3).

When taking continuous data (single truck test) the maximum length of the test depends on the rate at which data is being taken. A maximum of 660 scans can be taken for any given single truck test. One scan consists of one reading of each active channel. The maximum test length is then equal to the scan rate times 660. The scan rate is automatically set to .0125 seconds by the Data General. This is the fastest possible scan rate that the Campbell can use. With the .0125 second scan rate, the maximum length of a single truck test would be approximately 8.25 seconds. If several channels are being used, the Campbell may not be able to actually scan each of the channels every .0125 seconds and the maximum test time may actually be longer. If longer test lengths are desired then the scan rate must be increased. If a test length of 90 seconds is desired then the scan rate must be set to .136 seconds or greater. The procedure for adjusting the scan rate is discussed in Appendix B.

- Instrumentation Spacing

All of the locations that are to be instrumented must be connected by cable to the Campbell box. The maximum spacing between the instrumentation is therefore limited by the length of available cable. Four 90-foot cables and twelve 50-foot cables have been provided. These cables can be connected together to make longer lengths, if necessary. The cable lengths should be kept as short as possible to minimize noise and signal attenuation.

- Type of Instrumentation

Five clamp-on strain transducers have been provided with the system. If more channels of data are desired, then 120 ohm strain gages must be used or more strain transducers acquired. If strain gages are used they must be used with the strain gage completion boxes. Five completion boxes are provided.

Once the test plan has been developed, the channel description data as described in Section 4.3 can be entered and saved on the Data General computer. This will help reduce the time required to set up the test in the field.

3.2 Transducer Calibration

Each of the strain transducers provides a slightly different amplification of the actual strain values and therefore each transducer must be calibrated individually. The calibration data that must be input into the test program for each transducer is the output of the transducer in millivolts/volt for a specified stress level. The calibration is performed by bolting a transducer to the calibration bar and applying a known load. The calibration bar is shown in Fig. 2.8. The stress in the bar can be calculated using the bar cross sectional area of .75 in.^{2} and the output of the transducer can be read using the Campbell. The output of the transducer represents its calibration for the applied stress level. The following procedure should be used to obtain the transducer calibration data.

The calibration bar should be mounted in a test machine which is tall enough to handle the 37" long bar with a tension capacity of at least 10 kips. The load indicator on the test machine should be zeroed after clamping down the calibration bar. One transducer can then be bolted to the specimen and connected to channel 1 of the Campbell box. A 12V battery should also be connected to the box to provide power for the Campbell. The Campbell must then be programmed to read data from the transducer. For the calibration test it is best to program the Campbell directly using the Campbell keyboard, as opposed to using the DG for programming. The keystrokes required for the program are shown in Figure 3.2. After the battery is connected to the Campbell, the Campbell will come on, check the memory circuits, and then display "11:1111.11" on the screen. After the eight ones are displayed, programming can begin. Each entry in the program should be followed by an "A" to advance to the next program step. If a mistake is made in a program entry, the "B" key can be used to backup and the previous entry can be corrected. Alternatively, the

KEY	FUNCTION
*1 A	enter programming Table 1
D5 A	execution interval = 0.5 seconds
6 A	Instruction #6: full bridge measurements
1 A	1 strain transducer being read
13 A	±50 millivolt range
1 A	input channel number for transducer
1 A	use excitation channel 1
4000A	4000 mV excitation
1 A	use storage location 1
1 A	use multiplier of 1
0 A	use offset of 0
* 0	exit programming table, begin taking data
* 6A	display transducer reading in mV/V

FIGURE 3.2 Transducer Calibration Program

power can be disconnected from the Campbell and then reconnected and the programming started again from the beginning. Additional information on programming the Campbell can be found in the Campbell User's Manual.

After the Campbell has been programmed, the calibration test can begin. The transducer reading at zero load should be recorded and then load should be applied in 1 kip increments up to approximately 6 kips (8 ksi). The Campbell will continuously display the output of the transducer in MV/V. Transducer and load readings should be taken at each increment. The same procedure should then be followed for unloading. The test loading should not exceed six kips because the configuration of the transducers causes stress concentrations which can lead to local yielding of the transducers at higher load levels.

After the load has been removed the transducer should be moved to the other side of the calibration bar and the test repeated. The results of the two tests should be averaged in order to remove the effects of any bending that might be occurring in the specimen.

The required transducer calibration data can then be determined by plotting the transducer output in MV/V versus the stress in the calibration bar. The transducer output will be equal to the transducer reading minus the original transducer reading at zero load. The stress in the calibration bar will be equal to the load reading divided by the cross sectional area of .75 in.^{2}. A best fit line should be determined for the data and any point on the line can be taken as the calibration data for the transducer. It is recommended that the calibration test be run two or three times for each transducer and the results averaged.

The transducers should be recalibrated after every three or four tests or whenever a transducers has been subjected to stresses over approximately 10 ksi.

3.3 Power Supply

Power for running the Campbell during a test is provided by 12-volt marine batteries. If fully charged, one battery provides enough power to operate the Campbell for at least 18 days when all eight channels are being used. If fewer channels are being used, then the Campbell can operate longer. If a longer test is desired then a second battery can be connected to the Campbell and the operating duration will be doubled. Two battery connectors, wired in parallel, have been provided on the Campbell box for this purpose. Batteries can also be switched out during a test. A fully charged battery can be connected to the second battery connector and the old battery can then be disconnected. Two 12V batteries have been provided with the system.

The operating times given above are based on the performance of relatively new batteries operating at moderate temperature (approx. 70^oF). Consideration should be given to the drop in performance of the batteries with age and at lower temperature. Battery performance will drop considerably if the test is conducted at colder temperatures. Extreme care should be taken to insure that sufficient battery power is available for the full length of the test since all of the data will be lost if the battery voltage drops below the threshold needed to operate the Campbell.

A power supply is also required for the Data General during test set-up and data retrieval. The Data General has an internal battery which when fully charged can operate the computer for a maximum of two hours. Additional power can be provided by a portable generator or by the 12V batteries. A special adaptor has been provided which converts the 12V battery supply to the 7.5V used by the DG. The cable on this adapter is approximately 15 ft long and has a 2-pin connector for connecting to the battery. It should be noted that running the Data General off of the 12V batteries will reduce the length of time which the Campbell can run. Care should also be taken when running the Data General off of a portable generator since sudden power surges can damage the computer. An in-line voltage meter is useful in monitoring the output of the generator.

3.4 Desiccants

Inside the Campbell are several small desiccant packages. These packages contain material which absorbs moisture from the air to prevent possible damage to the Campbell circuits. At least once a year these packages need to be taken out of the Campbell and dried. To remove the desiccants the top of the Campbell must be removed. Special care should be taken not to disturb the wiring connected to the Campbell. The desiccant packages can then be placed in an oven to dry. The packages should be dried for 6 hours at 250^oF and then replaced in the Campbell.

3.5 Field Equipment Check List

The following equipment is required in the field to set-up and run a bridge test:

- Campbell 21X box
- 12V battery and box
- Data General computer
- strain transducers

- cables: computer to Campbell and transducers to Campbell
- C-clamps
- tool box

In addition, the following equipment may be required depending on the type of test planned:

- strain gages
- strain gage installation supplies
- strain gage completion boxes
- converter for running Data General off of 12V battery
- voltmeter
- camera

3.6 Equipment Set-up

Once the equipment has been transported to the test site, the actual set-up can be done fairly quickly. Figure 3.3 shows a typical set-up of the equipment. The order in which the equipment is mounted on the bridge is arbitrary and can be determined based on the method of access being used to get to the various locations on the bridge. The battery box must be mounted near the right side of the Campbell box and connected into one of the two pin connectors. The Data General also connects into the right side of the Campbell box. Figure 2.3 shows the location of these connectors on the box. Once the Data General and the battery have been connected, programming of the Campbell can begin regardless of whether the transducers have been connected.

The cables from the transducers and strain gages connect into the left side of the Campbell box. The eight connectors correspond to the eight input channels as shown in Figure 2.2. The cables should be wrapped around a C-clamp or a secondary bridge member near the Campbell box so that the cable weight will not be pulling on the connector. The same should be done for the connectors at the transducers and strain gages. The cables should be pulled tight to prevent them from hanging below the girder.

When installing the transducers, the C-clamps should be tightened by hand as tightly as possible to insure that no slipping occurs. When strain gages are used the strain

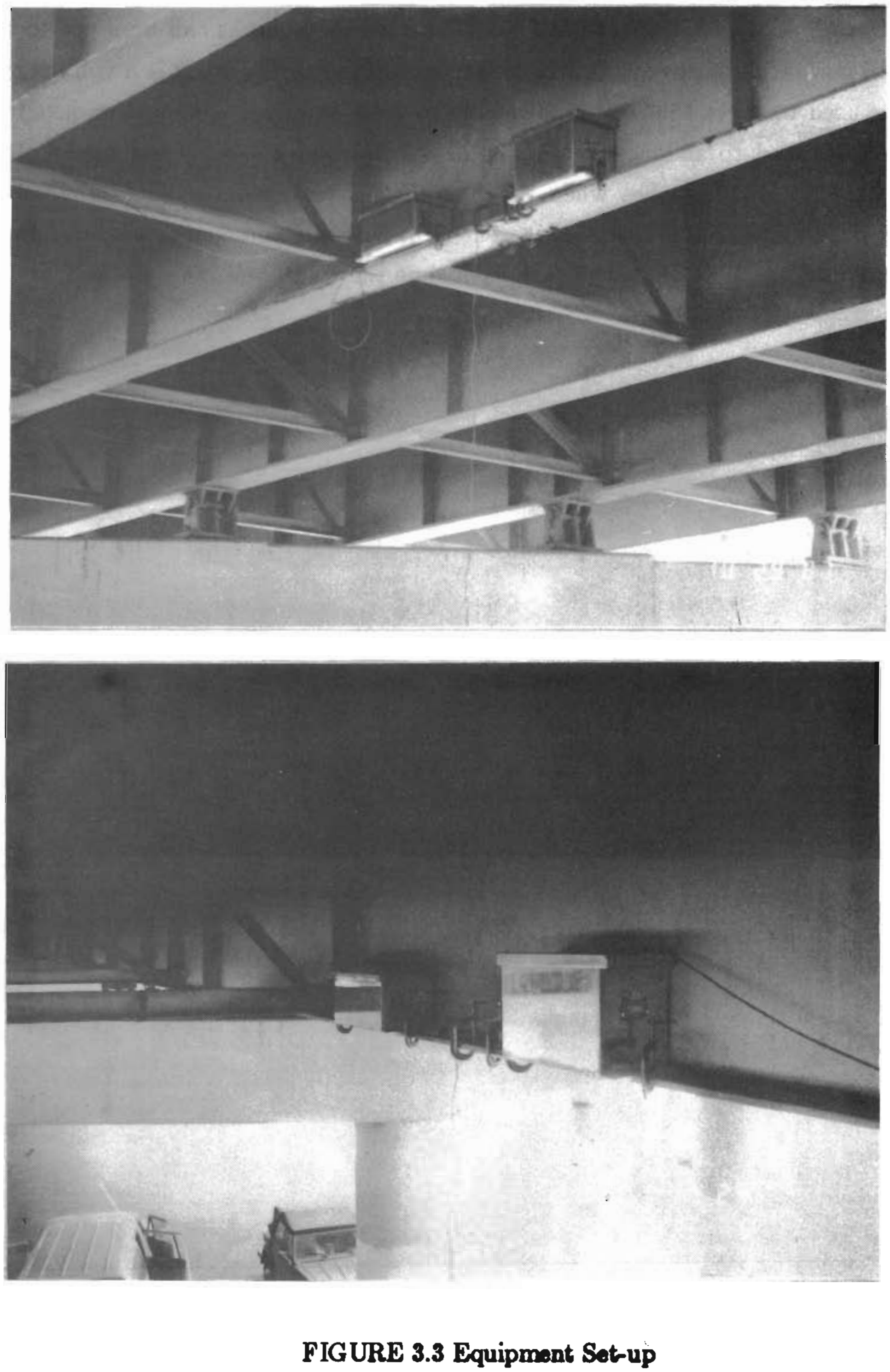


FIGURE 3.3 Equipment Set-up

gage completion boxes should be clamped to the girder and the connection to the gage should be insulated from moisture.

After all of the equipment has been connected, two checks can be made to see if the Campbell is working properly. On the right side of the campbell box is a voltage meter which can be used to check that power is getting to the Campbell. The button next to the meter must be pushed to activate it. Figure 2.3 shows the location of the meter. Below the meter is a light which flashes when the Campbell is taking data. The light must be activated by pushing the button next to it. The light will only come on after the Campbell has been programmed. The light will flash each time the Campbell takes data.

CHAPTER 4
TEST EXECUTION

Once the equipment has been set-up properly, a test can be controlled and executed directly from the Data General(DG) computer. The Campbell can be programmed using the DG and direct physical access to the Campbell is not required. The DG is connected to the Campbell by a cable which plugs into the serial port on the back of the DG.

The DG is an IBM compatible machine which uses the DOS operating system. It includes a 10-megabyte hard disk for permanent storage and a 3-1/2" floppy disk drive that can be used for transferring data to and from other IBM compatible machines. The programs supplied on the DG have been written specifically for the purpose of conducting bridge tests. The programs provide for a quick and simple means of programming the Campbell, recovering data from the Campbell, and performing preliminary analysis of the data. The specifics of using the programs for conducting a test are discussed in the following sections.

4.1 Program Overview

The primary program used to conduct bridge tests is titled 21X. This program was written by Alec Tahmassebi and a listing of the program code is included in Appendix D. The 21X program and others used for conducting bridge tests are stored on the DG hard disk in the CAMPBELL sub-directory. The program can be executed simply by typing 21X while in the CAMPBELL directory. The 21X program is menu driven and upon entering the program the main menu is displayed. The main menu is shown in Figure 4.1. From this menu three options can be selected using the function keys at the top of the DG keyboard or the program can be exited by pressing ESC. Pressing the F3 key enters the low level programming mode for the Campbell. This can be used to check the current program in the Campbell, to change the scanning rate, to modify the Campbell program, or to input a completely new program. It will not be necessary to use this mode for most tests. The F5 key brings up the channel description screen which is used to input the number and configuration of strain gages and transducers. Inputting the channel description data will normally be the first step in conducting a test. The F9 key accesses the data acquisition menu which is used to take and retrieve data. Each of these three secondary menus will be discussed further in the following sections. To return to the main menu from any of the

TEXAS STATE DEPARTMENT OF HIGHWAYS AND PUBLIC TRANSPORTATION

F3 : Low Level Programming

F5 : Channel Description

F9 : Data Acquisition Menu

ESC : Exit to DOS

Fig. 4.1 Main Menu

secondary menus use the ESC key. It is not possible to move directly from one secondary menu to another secondary menu.

To illustrate the use of the program, an example test will be discussed. The appropriate steps for conducting the example test will be given with the discussion of each of the program functions. The example test will involve taking both single truck (continuous) and rainflow type data at two locations on a bridge. The first location is a category C weld detail and will be instrumented with two strain transducers. The second location is a category E' weld detail and will be instrumented with one strain transducer and one strain gage. The test data sheet is shown in Figure 4.2. Assuming that the equipment is set-up properly as discussed in Section 3.6, the steps necessary to execute the example test will be discussed in the following sections and will be listed in Section 4.5.

4.2 Channel Description

The first step in executing a test is to input the channel description data. This is done using the channel description screen shown in Figure 4.3. The channel description data is used to tell the Campbell which of the eight input channels will be active and what type of instrumentation will be connected to each channel. The channel description screen is accessed from the main menu using the F5 key. The arrow keys at the bottom right corner of the keyboard can be used to move around the screen. The following input is required for each channel to be used.

1. Channel No.: The channel number corresponds to the input channel being used on the Campbell. The connectors for each Campbell channel are numbered on the outside of the Campbell box. If the cable from a strain gage or transducer is connected to the number 2 connector on the Campbell box, then the data for that gage or transducer should be entered into the channel number 2 line on the screen.

2. Channel Type: In this space a G should be entered if a strain gage is being used on this channel and a T should be entered if a transducer is being used. This information is used to set the expected input voltage range for the channel.

3. Calibration: This column is used to input the specific calibration data for the strain gage or transducer being used on this channel. Two related inputs are required. A stress level in ksi is entered in the S column. The stress level can be chosen up to a value of 99.99. A MV/V value is then entered in the adjacent

BRIDGE TEST DATA SHEET

Bridge Location : *Example Bridge Test*
Test Dates - Start: *1/20/88*
Finish: *1/28/88*

Instrumentation Description

Loc. #	Description	Fatigue Category	Instrument Type	Channel No.	Notes
1	*field splice , girder #4 , 2 nd span*	*C*	*trans #1* *trans #3*	*1* *2*	
2	*cover plate , girder #4 , north side , 1 st support*	*E'*	*trans #4* *S. G.*	*4* *5*	

Description Files

Channel Description File : *EXAMPLE*
Zero Description File : *EXZERO*
Rainflow Description File : *EXRFL*

Single Truck Tests

Data Files :

1. *EXA* .STK Notes: *fully loaded gravel truck in right lane*
2. *EXB* .STK Notes: *mobile home in center lane*
3. ______ .STK Notes: ______
4. ______ .STK Notes: ______

Rainflow Test

Rainflow Period : *1440 minutes*

Start Date : *1/20/88*
Time : *14:20*

Stop Date : *1/28/88*
Time : *9:55*

Data File : *EXRFL* .RFL

FIGURE 4.2 Example Test Data Sheet

Channel no.	Channel Type	Calibration S	MV/V	Fatigue Detail#	Sr max	Sr min
1	Undefined	?	?	Undefined	?	?
2	Undefined	?	?	Undefined	?	?
3	Undefined	?	?	Undefined	?	?
4	Undefined	?	?	Undefined	?	?
5	Undefined	?	?	Undefined	?	?
6	Undefined	?	?	Undefined	?	?
7	Undefined	?	?	Undefined	?	?
8	Undefined	?	?	Undefined	?	?

Message Line

F1: Load File   F2: Save File   F3: Send File   Del: Erase Channel   ESC: Exit

FIGURE 4.3 Channel Description Input Screen

column which corresponds to the stress level entered in the first column. A linear relationship between the stress level and the strain gage output is assumed and the MV/V value represents the voltage output of the gage (in millivolts) divided by the excitation voltage for the selected stress level. For a strain gage, the Campbell is programed to use an excitation voltage of 4 volts and the MV/V value can be calculated using the following formula:

$$MV/V = S * G.F. / (.004 * E)$$

where:

$$S = calibration\ stress\ level\ in\ ksi$$
$$G.F. = gage\ factor\ for\ strain\ gage$$
$$E = modulus\ of\ elasticity\ in\ ksi$$

The value should be entered with two significant figures to the right of the decimal point. If the value is less than one then the entry must include a zero before the decimal point (ex. 0.50). For a transducer, the MV/V value should be determined from a calibration test as described in Section 3.2.

4. Fatigue Detail No.: The AASHTO fatigue category for the detail to be instrumented should be determined in order to establish the maximum stress range that can be expected. The AASHTO categories are designated alphabetically from A to E'. The alphabetical category should be entered using the fatigue detail numbers shown on the message line at the bottom of the channel description screen. Category A is represented by 1, category B by 2, and so forth until category E' which is represented by 7. When a fatigue detail number between 1 and 7 is entered, default values for maximum and minimum stress ranges are automatically entered into the next two columns. The default maximum stress range value is the expected maximum stress for the specified type of detail and the minimum stress range is set at a level sufficient to prevent recording electronic noise. The default values may be modified if desired. When one channel is modified, the other channels with the same fatigue detail number are automatically updated with this modified value. F5 will restore the Sr Max and Sr Min values to the original default values. The default values are shown below.

CATEGORY	DETAIL NO.	Sr MAX	Sr MIN
A	1	28 ksi	3 ksi
B	2	20	2
B'	3	16	1
C	4	16	1
D	5	14	1
E	6	9	.5
E'	7	5	.5

For continuous data collection it is not necessary to enter a fatigue detail number based on the AASHTO categories. Any number from 1-99 may be entered. For numbers from 8-99, no values for maximum and minimum stress ranges will appear and these stress range limits should be entered as described in steps 5 and 6.

5. Sr Max: The maximum stress range that is expected during the test on this channel should be input in ksi. The default values based on the fatigue detail number may be changed if desired. In the rainflow mode, stress cycles with ranges up to the maximum stress range plus 5% will be recorded properly. Stress cycles exceeding this value will be recorded, but will be assigned the value of the maximum plus 5%. For continuous data tests, the Sr Max value is used for scaling only and does not actually limit the maximum stress that can be recorded.

   The absolute maximum stress values that can be recorded are governed by the maximum voltage range that the Campbell is set to and the calibration of the strain gage or transducer. The Campbell voltage range that is set automatically by the 21X program is ±5 mV for strain gages and ±50 mV for transducers. This allows a stress of greater than 50 ksi to be recorded when using transducers or strain gages with gage factors of approximately 2. The Campbell voltage range may be increased using the low level programming mode if higher voltage ranges are required.

6. Sr Min: In the rainflow mode it is necessary to enter a minimum stress cycle value that is to be recorded. This is to prevent the recording of low level cycles that are the result of electronic noise or are of no structural significance. A value of 0.5 ksi is generally sufficiently low to prevent recording of noise cycles. The default

values that are entered based on the fatigue detail number used are based on the best judgement of the authors as stress ranges which would produce insignificant fatigue damage. For continuous data tests, the Sr Min value has no effect on the test and any value may be entered.

The above information can be input in any order and as the cursor is moved from column to column a message is displayed near the bottom of the screen which gives helpful information about the data to be input in that particular column.

Below the message line is displayed the function keys which can be used from the channel description screen. The F1 and F2 keys can be used to load and save channel description files. These functions are useful because they allow the channel input data to be input prior to going into the field. The F1 key can be used to load a previously saved file into the channel description screen. When the F1 key is pressed, a prompt will be given requesting the filename of the file to be loaded. When the F2 key is pressed to save the channel data, a prompt will be given requesting a filename for the data to be saved. The filename should be in the standard DOS format. It is not necessary to enter a filename extension since the DG will automatically assign an extension of "21X" to the file. The same channel description file can then be retrieved by pressing F1 and entering the filename. It is not necessary to type the "21X" extension. If a mistake is made when entering a filename, the backspace key in the upper right corner of the keyboard can be used to go back and correct the entry.

The F3 key is used to send the channel description data to the Campbell datalogger. The DG is effectively programming the Campbell to take data on the specified channels. This programming takes 1 or 2 minutes and a message is flashed on the DG screen indicating that the Campbell is being programmed. It is recommended that the channel description data be saved using the F2 key prior to sending the program using the F3 key. The channel data remains in the 21X program until the program is exited, but it should be saved in case it is necessary to come back to it at a later time or if the program is abnormally exited due to a power failure. When the program is sent, the Campbell is turned on and begins to take data. However, the Campbell does not store the data taken until it is given instructions to do so using one of the commands on the data acquisition menu.

The other keys shown at the bottom of the screen are the DEL and ESC keys. The DEL key erases all of the information that has been entered for the channel that the pointer is currently on. The ESC key is used to return to the main menu.

The channel description screen for the example problem discussed earlier is shown in Figure 4.4. The two transducers to be used at the category C weld detail are connected into channels 1 and 2. The calibration for the transducers is 0.81 MV/V output at 7.0 ksi for the first transducer and 0.94 MV/V at 7.0 ksi for the second transducer. A category C detail corresponds to a fatigue detail number of 4 and the default values of 16.00 ksi and 1.00 ksi for the maximum and minimum stress ranges will be used. For the category E' detail, one transducer and one strain gage will be used and they are connected to channels 4 and 5. The calibration for the transducer is 0.95 MV/V at 7.0 ksi and the calibration for the strain gage is 0.35 MV/V at 20.0 ksi. Category E' corresponds to fatigue detail number 7 and again the default values for Sr Max and Sr Min will be used. Once the data has been entered as shown in the figure, the data should be saved using the F2 key and then sent to the Campbell using the F3 key. The channel description data was saved under the filename EXAMPLE.21X. After the data has been sent, the Campbell is programmed and is ready to take data. The ESC key can then be used to return to the main menu.

4.3 Data Acquisition

The data acquisition menu is accessed from the main menu using the F9 key and is shown in Figure 4.5. The main functions that are controlled from this menu include checking the data being received from the individual channels, zeroing the channels, initiating the collection of single truck (continuous mode) and rainflow data, and retrieving single truck or rainflow data from the Campbell. To use the functions on this menu the Campbell must have already been programmed previously, i.e. the channel description sent to the Campbell. The specific tasks carried out by each of the function keys are discussed below.

4.3.1 **F1: Check Channels.** This function and F2 are used to check that each of the active input channels are giving reasonable readings. When the F1 key is pushed, the Campbell is instructed to take data for a few seconds. After the data has been taken the DG then automatically retrieves the data from the Campbell and processes it. The scan number shown at the bottom of the screen indicates the number of readings for each channel that are still to be processed. After processing has been completed, the high, low, and average readings (in MV/V) are displayed for each of the channels which were specified on the channel description screen. If the Campbell has not been previously zeroed (see F3 and F4), then the data displayed will be raw data read directly from the strain gages or transducers. If the Campbell has previously been zeroed then the data displayed

Channel no.	Channel Type	Calibration S	Calibration MV/V	Fatigue Detail#	Sr max	Sr min
1	Transducer	7.00	0.81	4	16.00	1.00
2	Transducer	7.00	0.94	4	16.00	1.00
3	Undefined	?	?	Undefined	?	?
4	Transducer	7.00	0.95	7	5.00	0.50
5	Strain Gage	20.00	0.35	7	5.00	0.50
6	Undefined	?	?	Undefined	?	?
7	Undefined	?	?	Undefined	?	?
8	Undefined	?	?	Undefined	?	?

Message Line

F1: Load File   F2: Save File   F3: Send File   Del: Erase Channel   ESC: Exit

FIGURE 4.4 Example Channel Description Data

TEXAS STATE DEPARTMENT OF HIGHWAYS AND PUBLIC TRANSPORTATION

F1 = Check Channels

F2 = Take Single Reading

F3 = Take Zero Reading

F4 = Zero the Data Logger

F5 = Capture Truck
F6 = Retrieve Truck Data
F7 = Plot Truck Data
F8 = Save Truck Data

F9 = Start Rainflow Routine
F10 = Retrieve Rainflow Data

ESC = Exit to main menu

Waiting for command ...

Fig. 4.5 Data Acquisition Menu

will include the offsets used in the zeroing process. If the data are far from zero ($> \pm 1.0$ MV/V) then a good zero has not been obtained and the Campbell should be rezeroed.

4.3.2 **F2: Take Single Reading.** This function is similar to the Check Channels function. It instructs the Campbell to take data and then retrieve and display it. However, unlike the Check Channels function, it only retrieves one data point instead of retrieving several points and averaging them. The advantage of this function is that it is much quicker than the Check Channels. It only takes a few seconds to execute. The single reading function also differs from the check channels function in that even if the channels have been zeroed, the F2 function will always give the raw data read directly from the strain gages or transducers.

4.3.3 **F3: Take Zero Reading.** When the strain gages and transducers are installed, they will not produce a zero electronic output. It is necessary to subtract these initial non-zero outputs from each channel so that all of the channels will read zero stress under the same conditions and also to allow the full dynamic range of the Campbell to be utilized. This is referred to as zeroing and must be done before running a test. Zeroing is accomplished using the F3 and F4 functions. The F3 function instructs the Campbell to take data for a few seconds and the DG then retrieves the data for each active channel. The data are then averaged and displayed for each channel. These average values are saved by the DG and are used to zero the Campbell when the F4 function is used. The F3 key should be pressed when there is relatively little traffic on the bridge. Some automobile traffic is acceptable, but no truck traffic should be on the bridge when the zero readings are taken. If a truck should enter the bridge while zero readings are being taken, the zeroing function should be repeated by pressing the F3 key after the current zeroing operation is complete.

4.3.4 **F4: Zero The Data Logger.** After satisfactory zero readings have been taken using the F3 key, the Campbell must be reprogrammed using the zero values. The F4 function is used to accomplish this. Pressing the F4 key instructs the DG to reprogram the Campbell using a multiplier and offset which are derived from the zero values for each channel. The multiplier and offset values act on the raw data to give the zeroed values desired. Before reprogramming, the DG will ask for a filename to use for saving the description file. This description file will contain the data entered into the channel description screen, the zero readings, and the multiplier and offset values used. This information must be saved since it is used to retrieve and process the data. The filename should be different than the filename used to save the channel description data. The channel description data file contains default multiplier and offset values of 1 and 0. This channel description data file may be used again for another test at a later time, but the description file containing the actual zero, multiplier, and offset values should not be reused. No filename extension should be given, the DG will assign an extension of "21X". This is the same extension that is assigned for the channel description filename so care should be taken not to write over the channel description file. In the example problem the channel description data file was named EXAMPLE.21X. The description file containing the zero values will be named EXZERO.21X

4.3.5 **F5: Capture Truck**. The F5 through F8 functions are used for capturing a single truck crossing a bridge. The F5 function instructs the Campbell to start taking data continuously until instructed to stop. The Campbell must have been programmed and zeroed previously using the F4 function. When the F5 key is pressed there is a delay of a couple of seconds before data acquisition is actually begun because the Campbell must first be initialized. After the Campbell begins recording data, a message will be given at the bottom of the DG screen to press any key to stop taking data. Again there is a slight delay between pressing the key and stopping data collection. The maximum length of time that the test can cover is discussed in Section 3.1.

4.3.6 **F6: Retrieve Truck Data**. After a truck crossing has been recorded using F5, F6 can be used to retrieve the data from the Campbell. Only the data recorded during the last execution of the F5 function are retrieved. While the DG is retrieving the data, the number of scans remaining to be retrieved is displayed in the bottom right corner of the screen.

4.3.7 **F7: Plot Truck Data**. The most recent data retrieved using the F6 function can be plotted using the F7 function. When the F7 function is used, the DG will ask for the channel numbers to be plotted. Any combination of the active channels may be specified. After the channel numbers have been entered, press the return key and the specified channels will be plotted on the screen. The plot will be of time on the horizontal axis and a scaled output on the vertical axis. The maximum and minimum output readings for the specified channels will also be displayed. The scaled output readings can be converted to stress using the following formula:

$$\text{Stress} = \text{Sr Max} * (\text{ output }) / 95$$

where Sr Max is the value entered on the channel description for the maximum stress expected for this particular channel. To remove the plot from the screen, press any key. The F7 function may be repeated as many times as desired.

4.3.8 **F8: Save Truck Data**. The F8 key is used to save the data onto the DG hard disk. If the data are not saved using the F8 function before the F6: Retrieve Truck Data function is used again or before the program is exited, the data will be lost. When the F8 key is pressed the DG will ask for a filename and a filename should be entered with no file extension. The DG will assign a file extension of ".STK" (Single TrucK). If a filename of EXA is entered, the data will be stored in file EXA.STK.

4.3.9 **F9: Start Rainflow Routine**. The F9 function is used to program the Campbell to take rainflow data for use in fatigue analysis. In the rainflow mode the Campbell counts the number of stress cycles measured during a specified period of time. Rainflow refers to the technique that is used for counting the cycles. The cycle counts are stored in a two dimensional histogram with the cycle amplitudes on one axis and the mean cycle magnitudes on the other axis. The histogram is 2 by 50 with 2 mean cycle rows and 50 amplitude columns. For specific details on the Campbell rainflow program see Instruction 81 in the Campbell manual. When F9 is pressed the DG will ask for a rainflow period. The rainflow period is the length of time over which the Campbell takes rainflow data before writing the rainflow histogram to final storage. If a period of 20 minutes is specified, the Campbell will take rainflow data for 20 minutes then write the histogram to storage and start taking data in a new histogram for the next 20 minute period. The maximum period that can be used is one day (1440 minutes). When entering the rainflow period, the period must be in whole minutes (a decimal point should not be entered). The DG will then ask for the current time. The time must be entered in military format (ie 4:30 am = 0430 and 4:30 pm = 1630). The two digit hour should be entered first followed by the two digit minute. The DG then asks for a filename for storing the channel description data. This file can have any name, but just as in the F4 function the filename should be different from the file used for the channel description data. Again a file extension of ".21X" is automatically assigned to the filename. After the file is saved, the DG will program the Campbell for taking rainflow data and will set the rainflow capture flag. The Campbell will then begin to take rainflow data.

Sometimes when the rainflow capture flag is being set, the DG keyboard will lock up. This is due to a bug in the Campbell processing unit which occurs occasionally when a very short rainflow period is used. If this occurs the DG should be turned off and then restarted. After the 21X program is reentered, the data file just saved in the Start Rainflow Routine should be loaded into the channel description screen. It is not necessary to send this file to the Campbell. The data acquisition menu can then be entered and the F9 function can be executed again.

The rainflow period is synchronized with the real time that is input by the user. If a 60 minute interval is used, the Campbell will store a rainflow histogram every hour on the hour. If the Campbell is instructed to begin taking rainflow data at 1630, a histogram will be written to storage at 1700 and then again at every hour. However, this first partial histogram (which included only 30 minutes of data) will not be retrieved when the rainflow data is retrieved using the F10 function. Regardless of the time interval used, the first time interval of each subsequent day will always begin at midnight. For this reason, if a time interval is entered which does not evenly divide into 1440 minutes, the last histogram of each day will have a different interval length than the specified time interval. For example, if a time interval of 300 minutes (5 hours) is used, a histogram will be written to storage at 0500, 1000, 1500, 2000, and 2400. The last interval will be only 240 minutes or 4 hours long. This is undesirable from a testing viewpoint and it will also lead to a problem when retrieving the rainflow data since the program will not know how many histograms are to be retrieved. For these reasons, only time intervals which divide evenly into 1440 should be used. A more detailed discussion of the procedure used by the Campbell for synchronizing the time interval can be found in Instruction 92 of the Campbell manual.

The most commonly used time interval will be 1440 minutes or one day. For this case, the first full time interval will always begin at midnight after data collection has begun. If data collection is begun on Wednesday, the first histogram that will be retrieved when the F10 function is used will be for Thursday.

4.3.10 **F10: Retrieve Rainflow Data.** When returning to the test site to retrieve rainflow data, the DG must first be reconnected to the Campbell and then the F10 function is used to initiate the data retrieval. Upon entering the 21X program, no other tasks should be attempted before executing the retrieve data function. Executing some of the other functions could result in the rainflow data being erased. In addition, data should not be entered on the channel description screen. When the F10 function is pressed, the DG will request the channel description file to be entered. This must be the same file that was saved when the F9 function was used to start the rainflow data collection. Again the filename extension does not need to be entered.

The DG will determine the number of input channels being used and the number of elapsed rainflow intervals. The DG will then begin to retrieve the rainflow data from the Campbell. The total number of intervals being retrieved will be displayed along with the interval number of the current interval being retrieved. The intervals are retrieved one at a time and are then automatically saved on the DG hard disk before the next interval is retrieved.

The data is saved in a file that has the same filename as the channel description file used in F9 and F10. However, the file extension will be "RFL" instead of "21X". If a rainflow channel description file of EXRFL is used, then the rainflow data will be stored in EXRFL.RFL. The number of intervals retrieved will not include any partial intervals at the beginning and end of the test period. For example, if a test begins at 10:00 a.m. on Monday and ends at 4:00 p.m. on Friday and has a rainflow period of 1440 minutes (1 day), only three intervals will be retrieved. These will be for Tuesday, Wednesday, and Thursday. The data taken on Monday and Friday will be only partial intervals and will not be recorded. Retrieving rainflow data takes approximately 45 seconds per interval per channel.

4.4 Low Level Programming Mode

The low level programming mode allows the user to program the Campbell directly through the DG. In this mode the DG acts only as a communication link to the Campbell and the Campbell programming is done just as it would be done directly on the Campbell keyboard. This feature has been included to make the system as flexible as possible, but for most applications it will not be necessary to use this mode. The 21X program has been developed to program the Campbell automatically for typical tests. An example of a standard Campbell program generated by the 21X program is included in Appendix A. If some changes to the standard Campbell program are desired, they can be made using the low level programming mode. An example of this might be to change the sampling rate being used by the Campbell. The 21X program automatically sets the sample rate at the fastest possible (0.0125 seconds). For a single truck test it might be useful to use a slower sample rate. The sample rate can be adjusted using the low level programming mode. An example showing the keystrokes necessary to change the scan rate are given in Appendix B.

To use this mode it is necessary to understand the Campbell programming procedures. These are discussed extensively in the Campbell manual. Additional commands that are used by the DG in the low level programming mode are discussed in Appendix B. When this mode is entered there are no prompts or menus provided. To exit to the main menu, press the ESC key.

4.5 Example Test

To illustrate the steps required in a typical test, the computer entries required to execute the example test discussed in the previous sections are shown in Figure 4.6. The final Campbell program generated is shown in Appendix A.

ENTRY	DESCRIPTION
cd campbell	change to campbell directory
21X	execute 21X program
F5	enter channel description screen
	enter channel data as shown in Figure 4.4
F2	save channel description data
example	channel description filename
"return"	
F3	send channel description data to Campbell
ESC	exit to main menu
F9	enter data acquisition mode
F1	check channels; verify that all active channels are reading properly
F3	take data for zeroing process
F4	send zeroing values to Campbell
exzero	save zeroing values in file named exzero.21x
"return"	
F1	check channels; verify each channel is reading approximately zero
F5	begin taking data for first single truck test
"any key"	end single truck test
F6	retrieve single truck test data
F7	plot single truck test data
1, 2, 4, 5	plot channels 1, 2, 4 and 5
"return"	
"any key"	erase plots from screen
F8	save single truck test data
exa	save data in file exa.stk
"return"	
F5	begin taking data for second single truck test
"any key"	end single truck test
F6	retrieve second single truck test data
F7	plot second single truck test data
1, 2, 4, 5	plot channels 1, 2, 4 and 5
"return"	
"any key"	
F8	save data from second single truck test
exb	save data in file exb.stk
"return"	

FIGURE 4.6 Example Test

ENTRY	DESCRIPTION
F9	begin rainflow test
1440	use rainflow period of 1440 minutes (1 day)
1420	current time (start time of test)
exrfl	save rainflow description in file exrfl.21x
"return"	
ESC	return to main menu
ESC	exit 21X program

disconnect DG
return at end of rainflow test and reconnect

cd campbell	change to campbell directory
21x	execute 21X program
F9	enter data acquisition mode
F10	retrieve rainflow data
exrfl	read rainflow description file, retrieve and save data
"return"	
ESC	return to main menu
ESC	exit 21X program
copy exrfl.rfl, a:	make backup copy of rainflow data on floppy disk

FIGURE 4.6 Example Test (cont.)

CHAPTER FIVE
ESTIMATION OF REMAINING FATIGUE LIFE

5.1 Background

The data gathered during a field study of a bridge can be used to provide a realistic estimate of a structure's fatigue life. The stress cycles measured in the field are stored in a two-dimensional array for each period of collection and data channel in the Campbell. These arrays are then transferred to Data General microcomputers where the data is retrieved. The array contains the number of stress range cycles which occurred at each of the fifty stress range increments and two mean stress levels for each channel and period.

The stress range level and number of cycles can be used to estimate the fatigue damage using a Miner's rule summation to calculate the effective stress range as shown below:

$$S_{Re} \;=\; (\sum \gamma_i \; S_{Ri}^3)^{1/3} \tag{5.1}$$

where:

$\gamma_i = \; n_i/T_i$

$n_i = \;$ the number of cycles at stress range S_{Ri}

$T = \;$ the total number of cycles recorded $= \sum n_i$

Note that mean stress is not included in equation 5.1. Fatigue research on welded structural steel details indicate that mean stress is not a significant variable. The number of cycles at the two mean stress levels should be added to obtain n_i for each S_{Ri}.

The effective stress range represents the stress range which produces the same fatigue damage as the variable stress cycles measured on the bridge. The estimated fatigue life in cycles can be calculated using Eq. 5.2.

$$N_{life} \;=\; A \; S_{Re}^{-3} \tag{5.2}$$

The constant A in equation 5.2 is obtained from the fatigue life equation of the detail on the bridge where the stresses were measured. The value of A can be obtained from the

AASHTO fatigue design stress ranges for redundant load path members at 2 million cycles in Table 10.3.1A using the equation below:

$$A = 2 \times 10^6 \times S_{RD}^3 \tag{5.3}$$

where S_{RD} is the stress range in Table 10.31A for the detail under consideration.

In order to relate the cyclic life from equation 5.2 into the structure life in years, an estimate of the number of cycles per year is required. The number of cycles in a year can be estimated by annualizing the cycles gathered in the field collection period and adjusting this estimate for past and future traffic volume differences. Methods of adjusting the number of cycles using traffic surveys and estimated traffic volumes are presented later.

The measured stresses may often be so low that no fatigue damage is occurring at the location. The stress ranges listed in Tables 10.31A and 10.31B of the AASHTO Specification for over two million cycles represent the estimated fatigue limit or threshold stress range of each fatigue category. The values listed in Table 10.31A for redundant members are based on laboratory studies. The values in 10.31B are reduced stress ranges to provide increased reliability for non-redundant members. If the largest measured stress range gathered in the field study is less than the values listed for over two million cycles for the detail instrumented, no fatigue damage occurred at the detail during the period the data was collected. If the largest measured stress range is less than or equal to 75% of the threshold value and the data gathering period is representative of typical traffic, at least five days of data, then it is reasonable to assume that the location instrumented on the bridge will not exhibit fatigue cracking. No fatigue life estimate is required since the fatigue life is infinity. The 75% limit on the threshold stress range suggested above is based on the authors judgement. Higher cutoffs, but less than or equal to 100% of the threshold, can be justified if the user is satisfied that present or future loadings will not cause an increase in the measured stresses. A longer sampling period, for example two weeks versus one, or sampling another week will allow a more refined analysis and justify an increase in the percentage of threshold stress range to be used.

5.2 Program RFLO

In order to facilitate reduction of the stress range data gathered in the field, an additional program is provided. The program title is RFLO. This program is written in Turbo Pascal. RFLO can be used to plot the stress range and the fatigue damage factor for each channel and collection interval on the screen. In addition, the user can print out the data for further study and documentation. The program will also create comma separated files in which the printed data is written to a disk file with each value separated by a comma. This comma separated file can then be used as input into other programs such as commercial spreadsheet programs. A detailed description of how to use this program and interpretation of its output is given in the next section.

Data File : I35.RFL
Save File : C:I35.11
Number of intervals : 7
Interval Length : 1440 Minutes
Number of channels : 5

	SR Max	SR Min
Channel 1	9.000	0.500
Channel 2	9.000	0.500
Channel 3	9.000	0.500
Channel 4	9.000	0.500
Channel 5	9.000	0.500

Interval : 1
Channel : 1

F1=SUM MSL vs. SR   F2=D.F. vs SR   F4=PRINT   F5=SAVE   ESC = END
F7=SAVE ALL   F8=PRINT ALL

FIGURE 5.1 RFLO Input Screen

5.2.1 Using Program RFLO. The program is executed from DOS by typing RFLO followed by a carriage return. The prompt "Data File:" will then appear on the screen. Enter the data file you wish to work with by typing in the characters. RFLO automatically adds the extension RFL. Only files with the RFL extension can be accessed by this program. The RFL extension was added to your file name in program 21X when you retrieved the data. After the file name is typed and a carriage return pressed, RFLO will search the disk for the file and retrieve the header information in the file. The screen should look like Figure 5.1 after the file has been accessed. The number of channels, intervals, interval length, and the minimum and maximum stress ranges of each channel are displayed. Also, the program defaults the save file to File name.11. The numbered extension refers interval and channel. For example, I35.21 refers to the data gathered during the second interval on channel one. The cursor position is at the bottom of the screen adjacent to "Interval:". The number of the current interval and channel are displayed. The data from this interval and channel are the data the program will graph or print out on your command using the function keys. The interval and channel can be changed by moving the cursor using the arrow keys and typing in the desired quantities. The save file extension is automatically changed to match the current values.

The save file is a comma separated file created by RFLO if you press the F5 key. You can also create comma separated files for all intervals and channels by pressing the F8 key. A printout of the data and an analysis of the data can be obtained by pressing the F4 key. F4 will print out the current interval and channel. The printout for all intervals and channels is obtained by pressing the F7 key. A typical print is shown in Figure 5.2. The headings on the printout are defined as follows:

SRL = stress range level

MSL1 = number of cycles recorded in mean stress level 1

MSL2 = number of cycles recorded in mean stress level 2

SUM = total number of cycles recorded in both mean stress levels

SR = stress range in ksi for the given SRL

%MSL1 = percent of the total number of cycles recorded that are in the mean stress level 1

%MSL2 = percent of the total number of cycles recorded that are in the mean stress level 2

%ALL = percent of the total number of cycles recorded that are in the particular SRL

D.F. = damage factor, see Eq. 5.4

In addition to the printing, analysis, and saving functions, the program provides for a graphical display of the data on the screen. Pressing F1 produces a bar graph histogram of the level of occurrence of the stress ranges. The levels at the two mean stress levels are added together to produce this plot. Pressing the F2 key produces a plot of the

Interval = 1
Channel = 1

SRL	MSL1	MSL2	SUM	SR	% MSL1	% MSL2	% ALL	D.F.
1	0	0	0	.38	0	0	0	0
2	95	602	697	.76	5.88	37.23	43.10	.19
3	23	400	423	1.14	1.42	24.74	26.16	.38
4	1	207	208	1.52	.06	12.80	12.86	.45
5	0	99	99	1.89	0	6.12	6.12	.42
6	0	53	53	2.27	0	3.28	3.28	.39
7	0	64	64	2.65	0	3.96	3.96	.74
8	0	58	58	3.03	0	3.59	3.59	1.00
9	0	13	13	3.41	0	.80	.80	.32
10	0	2	2	3.79	0	.12	.12	.07
11	0	0	0	4.17	0	0	0	0
12	0	0	0	4.55	0	0	0	0
13	0	0	0	4.93	0	0	0	0
14	0	0	0	5.31	0	0	0	0
15	0	0	0	5.68	0	0	0	0
16	0	0	0	6.06	0	0	0	0
17	0	0	0	6.44	0	0	0	0
18	0	0	0	6.82	0	0	0	0
19	0	0	0	7.20	0	0	0	0
20	0	0	0	7.58	0	0	0	0
21	0	0	0	7.96	0	0	0	0
22	0	0	0	8.34	0	0	0	0
23	0	0	0	8.72	0	0	0	0
24	0	0	0	9.09	0	0	0	0
25	0	0	0	9.47	0	0	0	0
26	0	0	0	9.85	0	0	0	0
27	0	0	0	10.23	0	0	0	0
28	0	0	0	10.61	0	0	0	0
29	0	0	0	10.99	0	0	0	0
30	0	0	0	11.37	0	0	0	0
31	0	0	0	11.75	0	0	0	0
32	0	0	0	12.13	0	0	0	0
33	0	0	0	12.51	0	0	0	0
34	0	0	0	12.88	0	0	0	0
35	0	0	0	13.26	0	0	0	0
36	0	0	0	13.64	0	0	0	0
37	0	0	0	14.02	0	0	0	0
38	0	0	0	14.40	0	0	0	0
39	0	0	0	14.78	0	0	0	0
40	0	0	0	15.16	0	0	0	0
41	0	0	0	15.54	0	0	0	0
42	0	0	0	15.92	0	0	0	0
43	0	0	0	16.29	0	0	0	0
44	0	0	0	16.67	0	0	0	0
45	0	0	0	17.05	0	0	0	0
46	0	0	0	17.43	0	0	0	0
47	0	0	0	17.81	0	0	0	0
48	0	0	0	18.19	0	0	0	0
49	0	0	0	18.57	0	0	0	0
50	0	0	0	18.95	0	0	0	0

Cycles per Hours = 67.4
Effective Stress Range = 1.58

FIGURE 5.2 Printout from RFLO Program

fatigue damage factor versus the stress range. The damage factor is also the last column of the printout and is defined as:

$$\gamma_i \; S_{Ri}^3 \tag{5.4}$$

This term is the function summed in Eq. 5.1, the effective stress range calculation. The larger values of this damage factor indicate the levels of stress range producing significant fatigue damage. The two plots, histogram and fatigue damage, can be used to determine the significance of the measured stress ranges relative to the fatigue calculations. Figures 5.3 and 5.4 show the stress range histograms for all intervals and each relevant individual interval. Figure 5.5 shows the stress cycles in each interval. Figure 5.6 shows the fatigue damage for each interval. Most of the fatigue damage occurs at a stress range of 3 ksi. Intervals 1, 3 and 4 show significantly lower number of stress cycles.

5.3 Calculating Fatigue Life

It is recommended that a minimum of seven days of data be used in assessing a bridge. This one week period should prevent a biases in the analysis due to daily truck traffic fluctuations. A more refined analysis is obtained if the number of days of collection is increased. Based on our experience, however, care should be exercised in extending the testing for periods more than one week. Daily variations in traffic can be considerable. If, for example, eight days of data are collected and the repeated day of the week is a low traffic day, the resulting fatigue life estimate will be biased and possibly unconservative.

Before a fatigue life is estimated, the collected data should be screened to determine if it looks reasonable. The effective stress range for each collection interval of a channel should be reasonably constant. This daily effective stress range is given in the printout from RFLO. The number of cycles for each interval for all the channels should show reasonable correlation. That is, the ratio of the number of cycles between two channels from one day to another should be reasonably constant. If large differences occur that cannot be explained, then the field test should be repeated to determine the cause of the differences. Switching transducer is suggested to determine if the differences are caused by a faulty transducer.

5.3.1 Fatigue Life Calculation Example. The data from a seven-day (one interval equal to a day) test of an AASHTO category E′ fatigue detail are used to estimate the fatigue life of a bridge. The data was analyzed using a commercial spreadsheet program, SuperCalc 4. A copy of the SuperCalc template (example.cal) file is provided on a diskette.

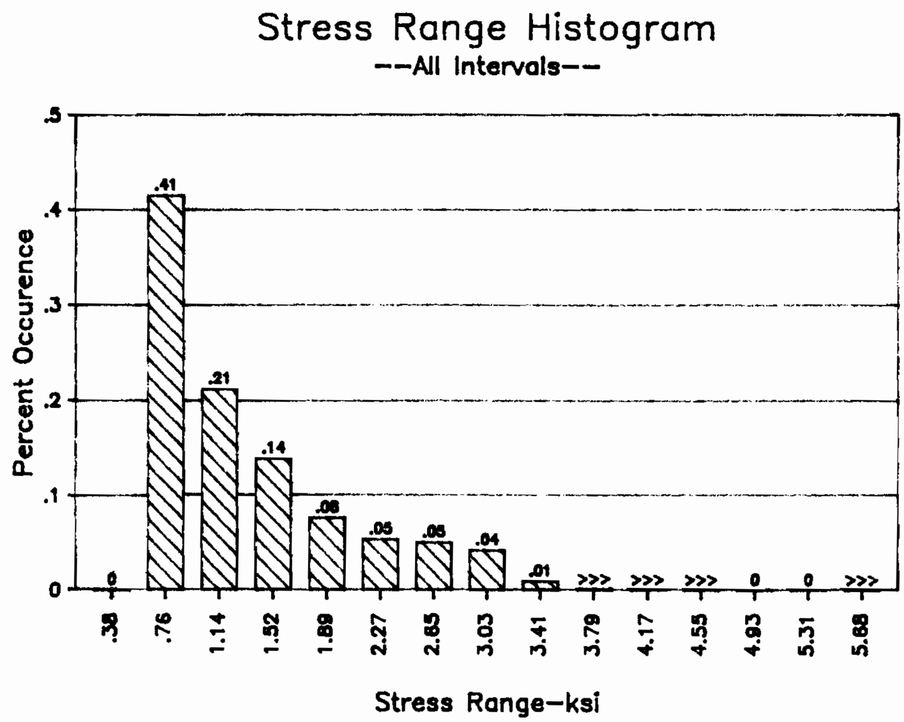


Figure 5.3

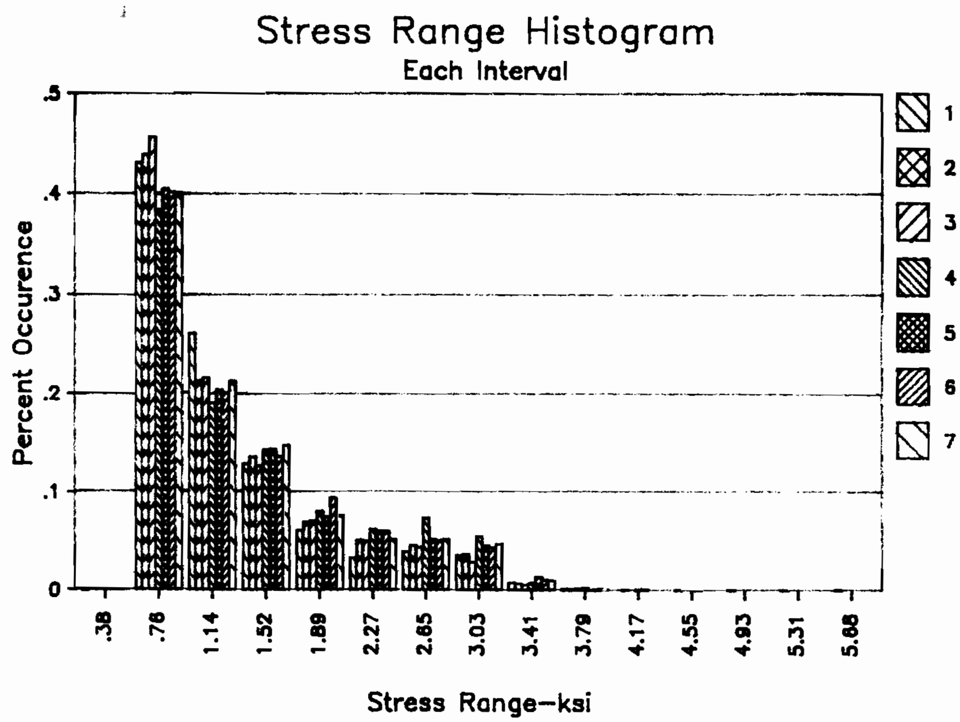


Figure 5.4

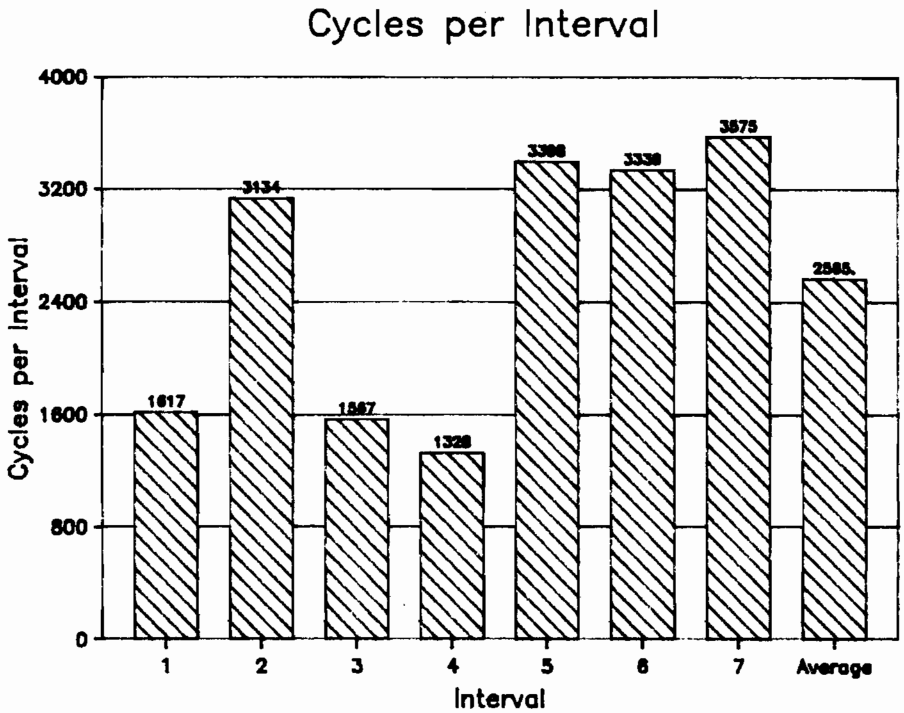


Figure 5.5

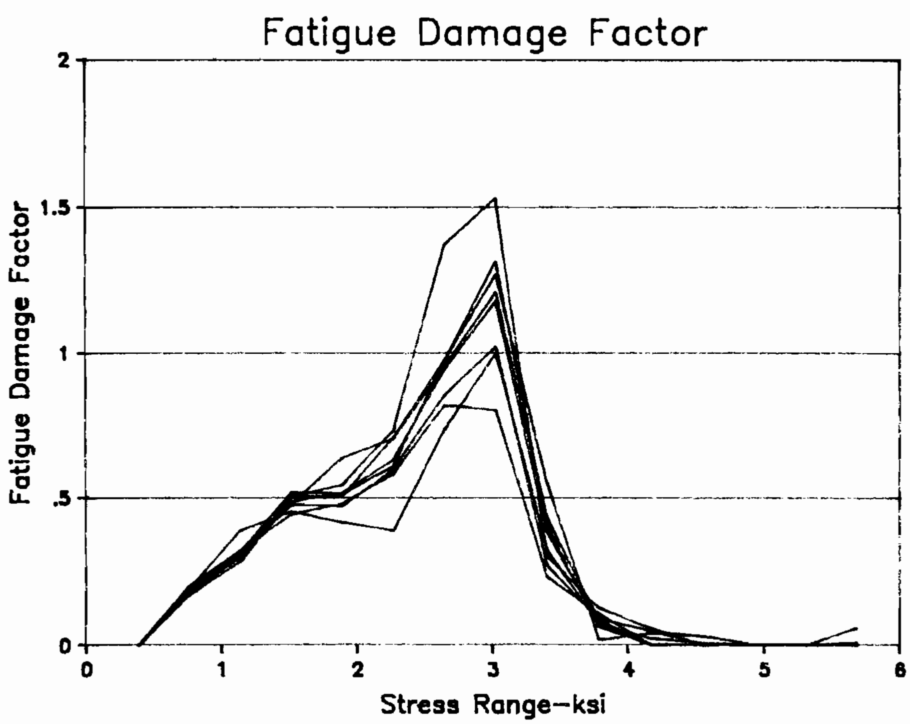


Figure 5.6

The data was first processed through program RFLO to create comma separated files. The seven files for Channel 1, the channel connected to the transducer at the E′ detail, were loaded into the SuperCalc template. The template was used to analyze the data and to produce hard copy plots using a pen plotter. The data in Fig 5.2 is one of the files used. Figures 5.3 through 5.6 were made using this spreadsheet program.

Figure 5.7 shows a printout from the spread sheet program. The number of cycles, effective stress range, maximum stress range, and the fatigue life based on the effective stress range cycles are shown.

Fatigue Life Analysis

Load Comma Separated Files from RFLO
Starting at Cell A20

Fatigue Detail Sr @ 2x10^6 cycles: 5.8ksi (enter value)

A: 3.9022e8

Interval	Cycles	Sr Eff.	Max. Sr	Life-Yrs	Graphs
1	1,617	1.58	3.79	167.6	1 - Sr Histogram All Interval
2	3,134	1.64	5.68	77.8	2 - Sr Histogram Ea. Interval
3	1,567	1.58	3.79	171.6	3 - Cycles per Interval
4	1,328	1.78	4.17	143.6	4 - Fatigue Damage Factor
5	3,398	1.72	4.55	62.0	
6	3,339	1.71	3.79	64.5	
7	3,575	1.70	3.79	61.3	
All	17,958	1.68	5.68	88.0	

Cycles/Year = 936,381

FIGURE 5.7 Spreadsheet Output

The values for all the intervals taken together is also shown. The overall values should be used for the fatigue life calculation. The estimated fatigue life based on the traffic conditions during the study is 88 years. Since the structure is 35 years old, one estimate of the remaining fatigue life is 43 years.

The calculated remains life of 43 years assumes that past and future traffic volume and distribution of trucks within the traffic remain stationary. A more realistic estimate of the fatigue life can be made by using actual historical traffic counts on the roadway. As an example of the use of measured traffic counts, assume the following traffic data is available:

YEAR	ADT	
1953	5,000	(bridge opening)
1960	7,500	
1970	10,000	
1975	15,000	
1980	20,000	

This measured traffic data can be used to estimate the traffic for the life of the structure using a simple compound traffic growth model shown below:

$$ADTj = \overline{ADT}\ (1+R)^j \tag{5.5}$$

where: ADTj = ADT after j years

R = rate of growth per year

In order to use this formula the value of R must be estimated from the measured data. A SuperCalc spreadsheet was used to determine the best fit R value. This spreadsheet was also used for some of the ratio calculations. The template file is named fatigue. cal and is on the diskette provided.

Figure 5.8 shows the absolute error of the predicted ADT versus the measured values for R from 0.04 to 0.063. The lowest error, best fit R value, is R=0.051. The estimated fatigue life is sensitive to value of R. Figure 5.9 shows how the predicted fatigue life changes with R, with all other parameters remaining constant. Figure 5.10 shows the

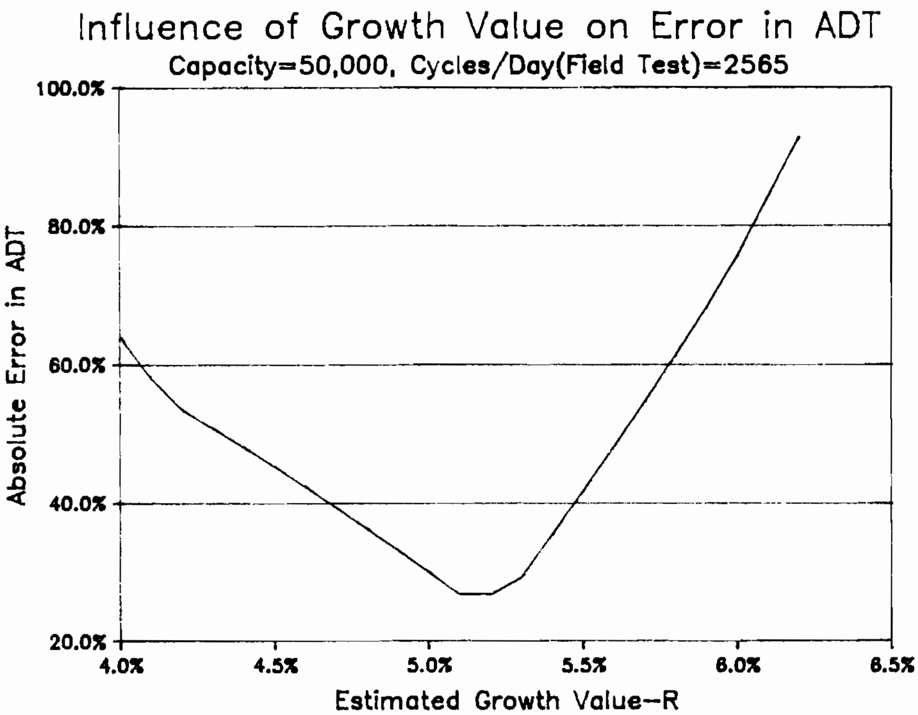


Figure 5.8

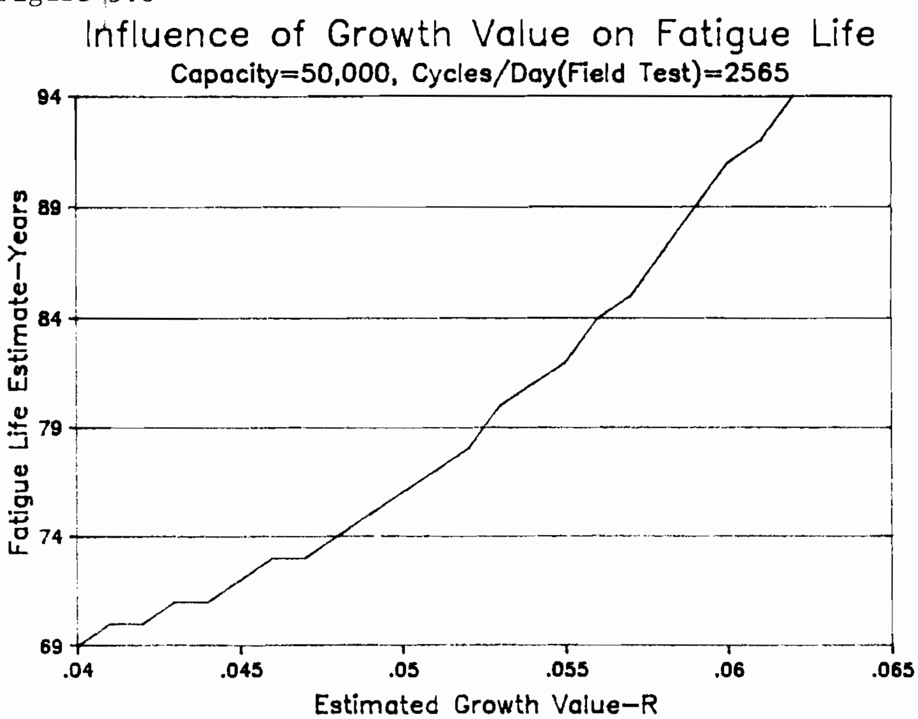


Figure 5.9

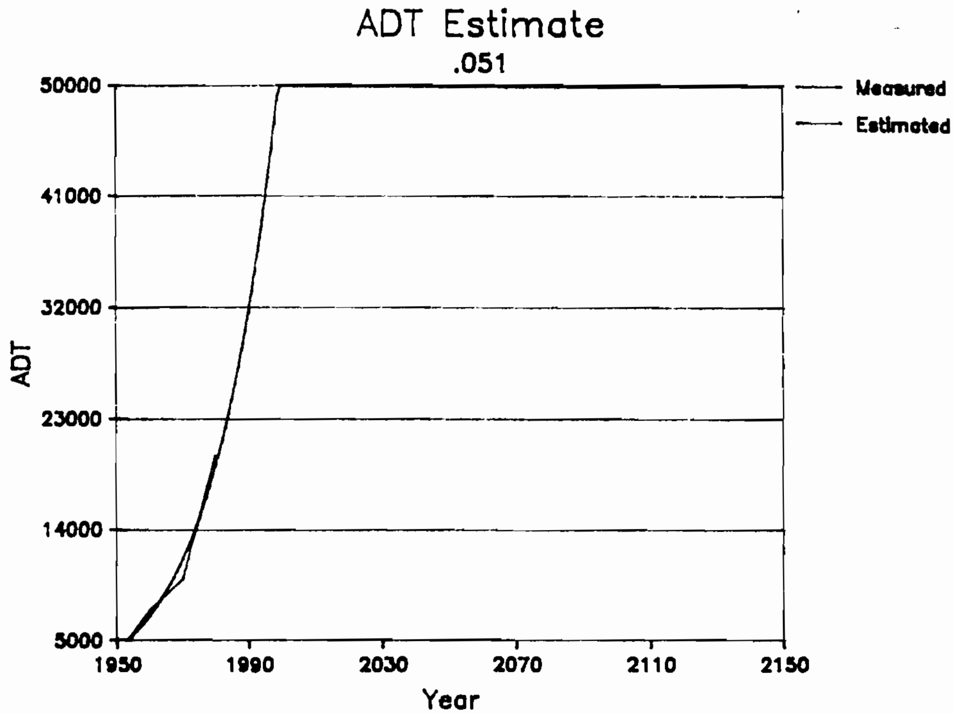


Figure 5.10

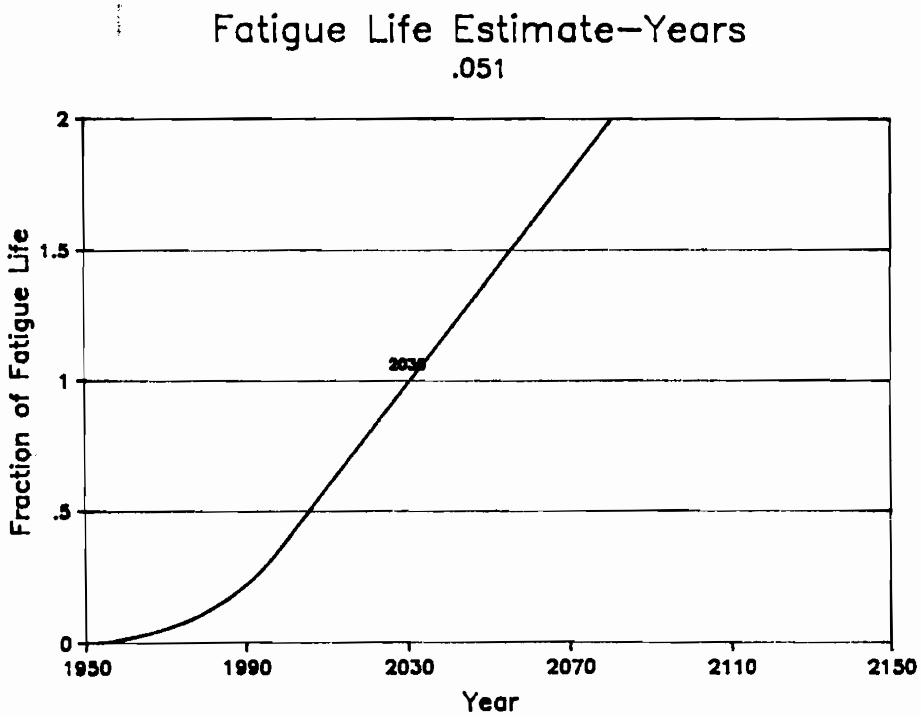


Figure 5.11

estimated ADT versus the ADT calculated from Eq. 5.5 using R=0.051. The agreement is seen to be fairly reasonable.

In addition to employing the measured ADT, a capacity limit upon the ADT should be used. This is necessary to prevent the ADT predicted from Eq. 5.5 to exceed the absolute capacity of the highway. This limit can be obtained from highway rating procedures or estimated based on observed conditions at peak traffic hours. An estimated maximum value of 50,000 was used in Figs. 8 through 10. The resulting fatigue life prediction is shown in Fig. 5.11. The estimated end of life is 2030, a fatigue life of 77 years. The ADT limit of 50,000 is reached after 47 years, the year 2000. The life estimates assume the ratio of stress cycles to number of vehicles remains constant over the life of the bridge. The ratio used in the spreadsheet analysis is based on the average number of stress range cycles from the seven-day field test, 17,958/7 = 2565 cycles divided by the estimated ADT for 1988 for an R value of 0.051. The estimated 1988 ADT is 28,514 which results in 0.08996 stress cycles per vehicle. This ratio is extremely important in determining the estimated fatigue life. Figure 5.12 shows how the fatigue life estimate changes as the average number of stress cycles per day is changed. A low of 1,317 and a high of 3,575 were measured during the field test. The result is a two-fold difference in fatigue life. A field test duration should be long enough to insure that daily variations in traffic do not cause the number stress cycles counted to be biased.

The last figure, Fig. 5.13, shows how the estimated fatigue life varies with the estimated ADT capacity. In this example, capacities above 75,000 do not significantly change the fatigue life since the majority of the fatigue damage occurs before the ADT reaches this capacity.

The estimated life for this example, based on steady state number of fatigue cycles equal to the average measured in the field test for the life of the bridge, is 88 years. Using a best-fit R of 0.051 and a ADT capacity of 50,000 yield a life of 77 years. Using the same value of R and ADT capacity, but increasing the number of stress cycles per day to the maximum in the field test yields a fatigue life of 62 years. Therefore, based on these estimates, the Category E$'$ detail in the example bridge would be expected to have significant cracks after a life of 60-80 years, or between the years 2013 and 2033.

The bridge should be inspected and retrofitted prior to this date. Further future field studies can be performed to redefine the number of stress cycles and effective stress range produced by traffic to compare with the values estimated in this analysis.

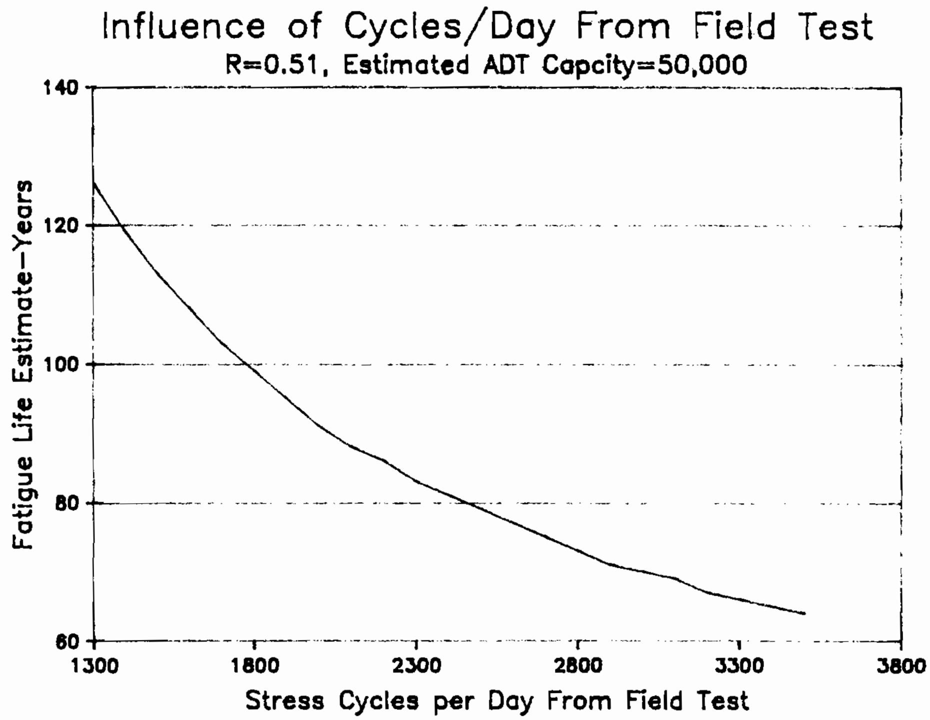


Figure 5.12

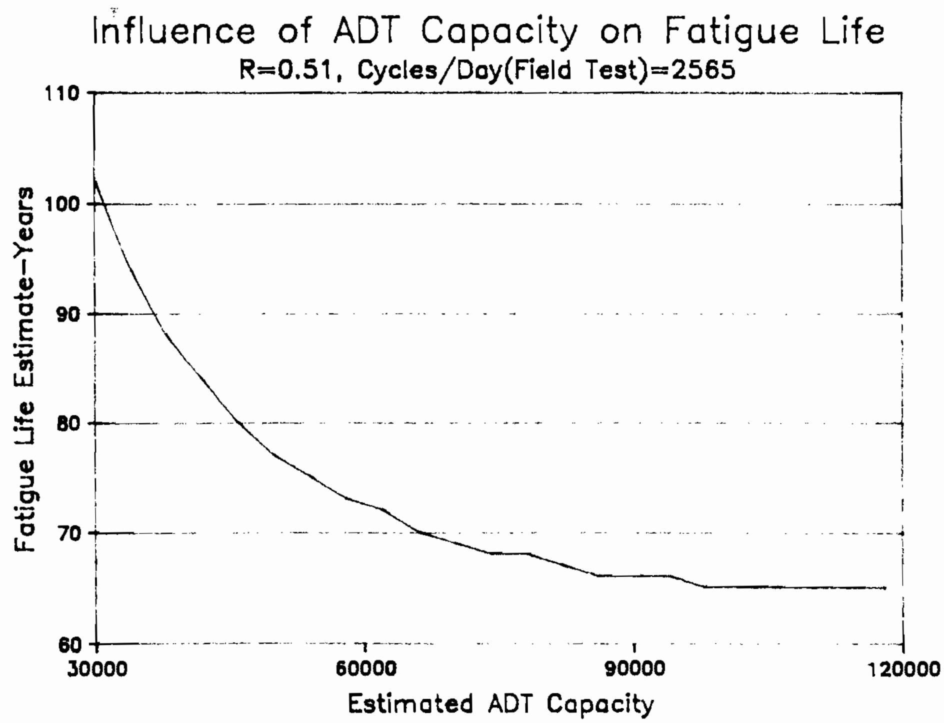


Figure 5.13

APPENDIX

DATA GENERAL PROGRAM LISTING

```
PROGRAM CR21X(INPUT,OUTPUT) ;
{ AUTHOR : BAHRAM ALEC TAHMASSEBI                                          }
{          DEVELOPED AT FERGUSON STRUCTURAL ENGINEERING LABORATORY         }
{          BALCONES RESEARCH CENTER                                        }
{          UNIVERSITY OF TEXAS AT AUSTIN                                   }
{          VERSION 1.1              7-1-88                                 }
{ ........................................................................ }

  CONST
     F1         = 59  ;
     F2         = 60  ;
     F3         = 61  ;
     F4         = 62  ;
     F5         = 63  ;
     F6         = 64  ;
     F7         = 65  ;
     F8         = 66  ;
     F9         = 67  ;
     F10        = 68  ;

     UP_ARROW   = 72  ;
     DN_ARROW   = 80  ;
     LT_ARROW   = 75  ;
     RT_ARROW   = 77  ;

     CARRIAGE   = 28  ;
     ESC        = 1   ;
     DEL        = 83  ;

     DEFAULT_DRIVE = 'C' ;

     ERR_ROW    = 25  ;
     ERR_COL    = 0   ;

     FKEY_ROW   = 24  ;
     MESSAGE_ROW         = 22 ;
     FILE_ROW            = 22 ;
     FILE_COL            = 10 ;
     TABLE_START_COLUMN  = 6  ;
     TABLE_START_ROW     = 2  ;
     ACTIVITY_MESS_ROW   = 24 ;

     MAX_PLOT_POINTS     = 650 ;
     MAX_INTERVALS       = 14  ;

     DEFAULT_SG_EXCITATION = 4000  ;  { 4000 milivolts = 4 volts }
     DEFAULT_SG_RANGE      = 11    ;  { 5 milivolt range         }
     DEFAULT_DT_EXCITATION = 4000  ;  { 4000 milivolts = 4 volts }
     DEFAULT_DT_RANGE      = 13    ;  { 50 milivolt range        }

     HILITE     = $40 ;
     BLINK      = $83 ;

{ ........................................................................ }

  TYPE
     MESSAGE                  = STRING[80] ;
     MESS_ARRAY               = ARRAY[0..10] OF MESSAGE ;
     REGISTERS                = RECORD
                                  AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS : INTEGER
                                END ;
     CHANNEL_RECORD           = RECORD
                                  ID              : INTEGER ;
                                  TRANSDUCER      : INTEGER ;   { 1 = sg , 2 = dt , 0 = undefined }
                                  TRANSDUCER_COL  : INTEGER ;
                                  DETAIL          : INTEGER ;
                                  DETAIL_COL      : INTEGER ;
                                  EXCITATION      : INTEGER ;
                                  EX_CHANNEL      : INTEGER ;
                                  ROW             : INTEGER ;
                                  SR_MAX          : REAL    ;
                                  SR_MIN          : REAL    ;
                                  S_CALIB         : REAL    ;
```

```
                                 MVPERV_CALIB     : REAL     ;
                                 OFFSET           : REAL     ;
                                 MULTIPLIER       : REAL     ;
                                 ZERO , CURRENT   : REAL     ;
                                 HI   , LO        : REAL     ;
                                 RANGE            : INTEGER  ;
       ;                      END ;
    VAL_ARRAY               = ARRAY[1..8] OF RECORD
                                                 INDEX : INTEGER ;
                                                 VAL   : REAL    ;
                                             END ;

    PLOT_ARRAY              = ARRAY[1..MAX_PLOT_POINTS] OF REAL ;
    PLOT_CHANNELS           = ARRAY[1..8] OF INTEGER ;
    TIME                    = RECORD
                                 HOUR   : INTEGER ;
                                 MINUTE : INTEGER ;
                                 SECOND : INTEGER ;
                              END ;

{ ....................................................................... }

  VAR
    CURR_PAGE               : BYTE          ;
    MESS                    : MESSAGE       ;
    ROW , COL               : INTEGER       ;
    CUR_ROW , CUR_COL       : BYTE          ;
    COLOR                   : INTEGER       ;
    KEY_PRESSED             : BYTE          ;
    KEY                     : BYTE          ;

    FNAME                   : STRING[50]    ;
    PF_FOUND                : BOOLEAN       ;
    NEW_PF                  : BOOLEAN       ;
    SOLID_LINE              : MESSAGE       ;
    ERRORS                  : MESS_ARRAY    ;
    DONE                    : BOOLEAN       ;
    SCREEN_SET              : BOOLEAN       ;
    ACQ_SCREEN_SET          : BOOLEAN       ;
    LAST_NUM                : REAL          ;
    LAST_CHAR               : CHAR          ;
    SERIAL_PORT             : TEXT          ;
    CH                      : CHAR          ;
    CHAR_AVAIL              : BOOLEAN       ;
    FULL_DUPLEX             : BOOLEAN       ;
    I                       : INTEGER       ;
    LINE , MODEM            : BYTE          ;

    DESCRIPTION_FILE        : TEXT          ;
    DATA_FILE               : TEXT          ;
    TRUCK_DATA              : TEXT          ;
    STK_FILE                : TEXT          ;

    REGS                    : REGISTERS     ;

    BAUD                    : INTEGER       ;
    PARITY                  : INTEGER       ;
    NSTOP                   : INTEGER       ;
    NDATA                   : INTEGER       ;

    CANCELLED               : BOOLEAN       ;
    EMPTY                   : BOOLEAN       ;
    PROGRAMMED              : BOOLEAN       ;
    ZEROED                  : BOOLEAN       ;

    TITLE                   : MESSAGE       ;
    TABLE_TOP               : MESSAGE       ;
    TABLE_TOP_2             : MESSAGE       ;
    TABLE_MIDDLE            : MESSAGE       ;
    TABLE_LINE              : MESSAGE       ;
    TABLE_BOTTOM            : MESSAGE       ;

    FKEYS                   : MESSAGE       ;
    S_CALIB_MESSAGE         : MESSAGE       ;
    MVPERV_CALIB_MESSAGE    : MESSAGE       ;
```

```
     TRANS_MESSAGE          : MESSAGE     ;
     DETAIL_MESSAGE         : MESSAGE     ;
     SR_MAX_MESSAGE         : MESSAGE     ;
     SR_MIN_MESSAGE         : MESSAGE     ;
     SEND_MESSAGE           : MESSAGE     ;
     READ_MESSAGE           : MESSAGE     ;
     DEFINE_MESSAGE         : MESSAGE     ;
     EMPTY_MESSAGE          : MESSAGE     ;
     CHANNEL_INFO           : ARRAY[1..8] OF CHANNEL_RECORD ;
     RAW_VOLTAGES           : ARRAY[1..8] OF REAL ;
     ACTIVE_CHANNELS        : ARRAY[1..8] OF INTEGER ;
     RAIN_ARRAY             : ARRAY[1..201] OF MESSAGE ;

     SRMAX , SRMIN          : ARRAY[1..7] OF REAL ;

     CURRENT_TIME           : TIME        ;
     CR21X_TIME             : TIME        ;
     START_TIME             : TIME        ;
     DAYS_ELAPSED           : INTEGER     ;
     INTERVALS_ELAPSED      : INTEGER     ;
     RAIN_INTERVAL          : INTEGER     ;

     CURRENT_CHANNEL        : INTEGER     ;
     CURRENT_FIELD          : INTEGER     ;

     MEANS_BINS             : INTEGER     ;
     AMPLITUDE_BINS         : INTEGER     ;
     LOW_LIMIT              : INTEGER     ;
     HIGH_LIMIT             : INTEGER     ;
     PEAK_VALLEY_DISTANCE   : REAL        ;
     CR_LF                  : STRING[2]   ;
     CR                     : CHAR        ;
     CHANNELS               : INTEGER     ;
     CR21X_CHANNELS         : INTEGER     ;
     DSP                    : INTEGER     ;
     MPTR                   : INTEGER     ;
     NSCAN                  : INTEGER     ;

     PLOT_DATA              : ARRAY[1..8] OF PLOT_ARRAY ;
     NPOINTS                : INTEGER     ;

{-----------------------------------------------------------------------------}

PROCEDURE CLS (STROW,STCOL,ENDROW,ENDCOL,COLOR : BYTE  ) ;
   VAR
     REGS : REGISTERS ;
   BEGIN
     WITH REGS DO
        BEGIN
          AX := $0700 ;
          BX := (COLOR  * 256) + 0      ;
          CX := (STROW  * 256) + STCOL ;
          DX := (ENDROW * 256) + ENDCOL
        END ;
      INTR($10,REGS)
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE SCROLL_UP (STROW,STCOL,ENDROW,ENDCOL : BYTE) ;
   VAR
     REGS : REGISTERS ;
   BEGIN
     WITH REGS DO
        BEGIN
          AX := $0601 ;
          BX := $0700 ;
          CX := (STROW  * 256) + STCOL ;
          DX := (ENDROW * 256) + ENDCOL
        END ;
      INTR($10,REGS)
   END ;

{-----------------------------------------------------------------------------}
```

```
PROCEDURE SCROLL_DN (STROW,STCOL,ENDROW,ENDCOL : BYTE) ;
   VAR
      REGS : REGISTERS ;
   BEGIN
      WITH REGS DO
         BEGIN
         ; AX :  $0701 ;
           BX := $0700 ;
           CX := (STROW  * 256) + STCOL ;
           DX := (ENDROW * 256) + ENDCOL
         END ;
       INTR($10,REGS)
   END ;

{----------------------------------------------------------------------------}

PROCEDURE CURSOR (ROW,COL : BYTE) ;
   VAR
      REGS : REGISTERS ;
   BEGIN
      WITH REGS DO
         BEGIN
           AX := $0200 ;
           BX := CURR_PAGE   * 256 ;
           DX := (ROW * 256) + COL
         END ;
       INTR($10,REGS)
   END ;

{----------------------------------------------------------------------------}

PROCEDURE SEL_PAGE ;
   VAR
      REGS : REGISTERS ;
   BEGIN
      REGS.AX := (5 * 256) + CURR_PAGE ;
      INTR($10,REGS)
   END ;

{----------------------------------------------------------------------------}

PROCEDURE GETC (VAR KEY : BYTE) ;
   VAR
      REGS : REGISTERS ;
   BEGIN
      REGS.AX := $0000 ;
      INTR($16,REGS)    ;
      KEY         := REGS.AX AND $00FF ;
      KEY_PRESSED := (REGS.AX AND $FF00) DIV 256
   END ;

{----------------------------------------------------------------------------}

PROCEDURE WRCOL(ROW, COL, COLOR : INTEGER ; MESS : MESSAGE) ;
   VAR
      I, J, K : INTEGER    ;
      REGS    : REGISTERS  ;
   BEGIN
      K := COL ;
      J := ORD(MESS[0]) ;
      FOR I := 1 TO J DO
         BEGIN
            CURSOR(ROW,K) ;
            WITH REGS DO
               BEGIN
                  AX   := (9 * 256) + ORD(MESS[I]) ;
                  BX   := (CURR_PAGE * 256) + COLOR ;
                  CX   := $0001 ;
                  INTR($10,REGS) ;
                  K    := K + 1
               END
         END
   END ;

{----------------------------------------------------------------------------}
```

```
PROCEDURE GET_CURSOR(VAR ROW, COL : BYTE) ;
   VAR
      REGS     : REGISTERS  ;
   BEGIN
      REGS.AX := $0300 ;
      REGS.BX := (CURR_PAGE * 256) + 0 ;
      INTR($10,REGS) ;
      ROW := (REGS.DX AND $FF00) DIV 256 ;
      COL :=  REGS.DX AND $00FF
   END ;

{ ---------------------------------------------------------------------}

PROCEDURE VDO_STAT(VAR MODE, WIDTH, PAGE : BYTE) ;
   VAR
      REGS     : REGISTERS  ;
   BEGIN
      REGS.AX := $0F00 ;
      INTR($10,REGS) ;
      MODE   := LO(REGS.AX) ;
      WIDTH  := HI(REGS.AX) ;
      PAGE   := HI(REGS.BX) ;
   END ;

{----------------------------------------------------------------------}

PROCEDURE VDO_MODE(MODE : BYTE) ;
   VAR
      REGS     : REGISTERS  ;
   BEGIN
      REGS.AX := $0000 + MODE ;
      INTR($10,REGS) ;
   END ;

{----------------------------------------------------------------------}

FUNCTION NUMERIC(CH : BYTE) : BOOLEAN ;
   BEGIN
      NUMERIC := FALSE ;
      IF ((CH >= 48) AND (CH <= 57))
         THEN NUMERIC := TRUE
   END ;

{----------------------------------------------------------------------}

FUNCTION SPECIAL(CB : BYTE) : BOOLEAN ;
   BEGIN
      SPECIAL := FALSE ;
      IF    (CB = 13) OR (CB = 44) OR
            (KEY_PRESSED = UP_ARROW) OR (KEY_PRESSED = DN_ARROW) OR
            (KEY_PRESSED = LT_ARROW) OR (KEY_PRESSED = RT_ARROW) OR
            (KEY_PRESSED = F1)  OR  (KEY_PRESSED = F2)  OR
            (KEY_PRESSED = F3)  OR  (KEY_PRESSED = F4)  OR
            (KEY_PRESSED = F5)  OR  (KEY_PRESSED = F6)  OR
            (KEY_PRESSED = F7)  OR  (KEY_PRESSED = F8)  OR
            (KEY_PRESSED = F9)  OR  (KEY_PRESSED = F10) OR
            (KEY_PRESSED = ESC) OR  (KEY_PRESSED = DEL)
        THEN SPECIAL := TRUE
   END ;

{----------------------------------------------------------------------}

PROCEDURE DISP_ERR(ERR : INTEGER) ;
   VAR
      CUR_ROW , CUR_COL : BYTE ;
   BEGIN
      GET_CURSOR(CUR_ROW,CUR_COL) ;
      WRCOL(ERR_ROW,ERR_COL,4,ERRORS[ERR]) ;
      WRCOL(ERR_ROW,ERR_COL,8,ERRORS[0]) ;
      CURSOR(CUR_ROW,CUR_COL)
   END ;

{----------------------------------------------------------------------}
```

```
PROCEDURE GETD(VAR DIG : INTEGER) ;
   VAR
      DONE  : BOOLEAN ;
      KEY   : BYTE  ;
   BEGIN
      DONE  := FALSE ;
      WHILE NOT DONE DO
         BEGIN
            GETC(KEY) ;
            IF (NUMERIC(KEY))
               THEN
                  BEGIN
                     DONE := TRUE ;
                     DIG  := ORD(CHR(KEY)) - 48
                  END
               ELSE IF SPECIAL(KEY)
                       THEN DONE := TRUE
                       ELSE DISP_ERR(5)
         END
   END ;

{-----------------------------------------------------------------------------}

FUNCTION LEGAL_C(CH : BYTE) : BOOLEAN ;
   BEGIN
      LEGAL_C := FALSE ;
      IF ((CH >= 48) AND (CH <= 57))  OR  ((CH >= 64) AND (CH <= 90))  OR
         ((CH >= 95) AND (CH <= 123)) OR  ((CH >= 35) AND (CH <= 39))  OR
          (CH  = 33) OR  (CH  = 125)  OR   (CH  = 126)
        THEN LEGAL_C := TRUE
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE GETNAM ;
   VAR
      CB   , ROW , COL   : BYTE        ;
      I , FLEN , ELEN    : INTEGER     ;
      DONE               : BOOLEAN     ;
      EDONE , FDONE      : BOOLEAN     ;
      NAME               : STRING[50]  ;
   BEGIN
      GET_CURSOR(ROW,COL) ;
      FOR I   := 3 TO 50 DO NAME[I] := ' ' ;
      DONE    := FALSE ;
      FDONE   := FALSE ;
      NEW_PF  := FALSE ;
      FLEN    := 0 ;
      ELEN    := 0 ;
      WHILE NOT DONE DO
        BEGIN
          ELEN   := 0 ;
          EDONE  := FALSE ;
          WHILE NOT FDONE DO
             BEGIN
                GETC(CB) ;
                IF LEGAL_C(CB)
                  THEN
                     BEGIN
                        WRCOL(ROW,COL,14,CHR(CB)) ;
                        COL := COL + 1 ;
                        CURSOR(ROW,COL) ;
                        FLEN := FLEN + 1 ;
                        NAME[FLEN+2] := CHR(CB) ;
                        IF FLEN = 8
                           THEN
                              BEGIN
                                 FDONE := TRUE ;
                                 NAME[FLEN+3] := '.' ;
                                 WRCOL(ROW,COL,14,'.') ;
                                 COL := COL + 1 ;
                                 CURSOR(ROW,COL) ;
                              END
                     END
                  ELSE
```

```
            IF (FLEN > 0) AND (CB = 46)
               THEN
                  BEGIN
                      FDONE := TRUE          ;
                      WRCOL(ROW,COL,14,'.')  ;
                      COL := COL + 1         ;
                      CURSOR(ROW,COL)        ;
                      NAME[FLEN+3] := '.'    ;
                  END
               ELSE
                  IF (CB = 8) AND (FLEN > 0)
                      THEN
                         BEGIN
                            COL := COL - 1          ;
                            WRCOL(ROW,COL,14,' ')  ;
                            CURSOR(ROW,COL)         ;
                            FLEN := FLEN - 1        ;
                         END
                      ELSE
                         IF SPECIAL(CB)
                           THEN
                              BEGIN
                                 FDONE := TRUE ;
                                 EDONE := TRUE ;
                                 DONE  := TRUE ;
                                 ELEN  := 3 ;
                                 NAME[FLEN+4] := '2' ;
                                 NAME[FLEN+5] := '1' ;
                                 NAME[FLEN+6] := 'X'
                              END
                           ELSE DISP_ERR(1)
   END ;
WHILE NOT EDONE DO
   BEGIN
      GETC(CB) ;
      IF LEGAL_C(CB)
        THEN
           BEGIN
              WRCOL(ROW,COL,14,CHR(CB))    ;
              COL := COL + 1               ;
              CURSOR(ROW,COL)              ;
              ELEN := ELEN + 1             ;
              NAME[FLEN+3+ELEN] := CHR(CB) ;
              IF ELEN = 3
                 THEN
                    BEGIN
                       EDONE := TRUE ;
                       DONE  := TRUE
                    END
           END
        ELSE
           IF SPECIAL(CB)
              THEN
                 BEGIN
                    EDONE := TRUE ;
                    DONE  := TRUE ;
                    IF ELEN = 0
                       THEN
                          BEGIN
                             ELEN  := 3 ;
                             NAME[FLEN+4] := '2' ;
                             NAME[FLEN+5] := '1' ;
                             NAME[FLEN+6] := 'X'
                          END
                 END
              ELSE
                 IF (CB = 8)
                    THEN
                       IF ELEN > 0
                          THEN
                             BEGIN
                                COL := COL - 1          ;
                                WRCOL(ROW,COL,14,' ')  ;
                                CURSOR(ROW,COL)         ;
                                ELEN := ELEN - 1        ;
```

```
                                       END
                                    ELSE
                                       BEGIN
                                          COL := COL -  2                    ;
                                          WRCOL(ROW,COL,14,'   ') ;
                                          CURSOR(ROW,COL)                    ;
                                          FDONE := FALSE                     ;
                                          FLEN := FLEN - 1                   ;
                                          EDONE := TRUE
                                       END
                              ELSE DISP_ERR(1)
             END
      END ;
      IF FLEN > 0
         THEN
            BEGIN
               NEW_PF    := TRUE   ;
               PF_FOUND  := FALSE  ;
               NAME[FLEN+3] := '.' ;
               NAME[0] := CHR(FLEN+ELEN+3) ;
               FNAME[0] := NAME[0] ;
               FNAME[1] := DEFAULT_DRIVE ;
               FNAME[2] := ':' ;
               FOR I := 3 TO ORD(NAME[0]) DO FNAME[I] := NAME[I]
            END
END ;

{-----------------------------------------------------------------------------}

PROCEDURE GETNUM(VAR NUMBER : REAL ; WL, FL : INTEGER)  ;
   VAR
      ENTRY                    : ARRAY [1..10] OF CHAR ;
      I , J                    : INTEGER ;
      WHOLE , FRAC , MUL       : REAL ;
      WLEN , FLEN              : INTEGER ;
      DONE , WDONE , FDONE     : BOOLEAN ;
      CB                       : BYTE ;
      CH                       : CHAR ;
      ROW , COL                : BYTE ;
   BEGIN
      GET_CURSOR(ROW,COL) ;
      FOR I := 1 TO 10 DO ENTRY[I] := ' ' ;
      WLEN := 0 ;
      IF (WL > 0) THEN WDONE := FALSE  ELSE WDONE := TRUE ;
      DONE  := FALSE ;
      WHILE NOT DONE DO
         BEGIN
            FLEN := 0 ;
            IF FL > 0 THEN FDONE := FALSE ELSE FDONE := TRUE ;
            WHILE NOT WDONE DO
               BEGIN
                  GETC(CB) ;
                  IF NUMERIC(CB)
                     THEN
                        BEGIN
                           WLEN := WLEN + 1 ;
                           ENTRY[WLEN] := CHR(CB) ;
                           WRCOL(ROW,COL,14,ENTRY[WLEN]) ;
                           COL := COL + 1 ;
                           CURSOR(ROW,COL) ;
                           IF WLEN = WL THEN IF FDONE THEN BEGIN
                                                              WDONE := TRUE ;
                                                              DONE  := TRUE
                                                           END
                                                      ELSE BEGIN
                                                              WDONE := TRUE ;
                                                              WRCOL(ROW,COL,14,'.') ;
                                                              COL := COL + 1 ;
                                                              CURSOR(ROW,COL) ;
                                                           END
                        END
                     ELSE IF SPECIAL(CB)
                             THEN BEGIN
                                     WDONE := TRUE ;
```

```
                                          FDONE  := TRUE ;
                                          DONE   := TRUE
                                       END
                                  ELSE IF (CB = 8) AND (WLEN > 0)
                                          THEN BEGIN
                                                  COL := COL - 1 ;
                                                  WRCOL(ROW,COL,14,' ')   ;
                                                  CURSOR(ROW,COL) ;
                                                  WLEN := WLEN - 1
                                               END
                                          ELSE IF (CB = 46) AND (WLEN > 0)
                                                  THEN BEGIN
                                                          IF (FL > 0)
                                                            THEN BEGIN
                                                                    WRCOL(ROW,COL,14,'.') ;
                                                                    COL := COL + 1 ;
                                                                    CURSOR(ROW,COL) ;
                                                                 END ;
                                                          WDONE := TRUE
                                                       END
                                                  ELSE DISP_ERR(5)
         END ;
      WHILE NOT FDONE DO
         BEGIN
            GETC(CB) ;
            IF NUMERIC(CB) THEN BEGIN
                                   FLEN := FLEN + 1 ;
                                   ENTRY[WLEN+FLEN] := CHR(CB) ;
                                   WRCOL(ROW,COL,14,ENTRY[WLEN+FLEN]) ;
                                   COL := COL + 1 ;
                                   CURSOR(ROW,COL) ;
                                   IF FLEN = FL THEN BEGIN
                                                        FDONE := TRUE ;
                                                        DONE  := TRUE
                                                     END
                                END
            ELSE
               IF SPECIAL(CB)
                  THEN BEGIN
                          FDONE := TRUE ;
                          DONE  := TRUE
                       END
                  ELSE
                     IF (CB = 8)
                        THEN
                           IF FLEN > 0
                              THEN
                                 BEGIN
                                    COL := COL - 1 ;
                                    WRCOL(ROW,COL,14,' ') ;
                                    CURSOR(ROW,COL) ;
                                    FLEN := FLEN - 1
                                 END
                              ELSE
                                 BEGIN
                                    COL := COL - 2 ;
                                    WRCOL(ROW,COL,14,'  ') ;
                                    CURSOR(ROW,COL) ;
                                    WDONE := FALSE ;
                                    WLEN := WLEN - 1 ;
                                    FDONE := TRUE
                                 END
                        ELSE DISP_ERR(5)
         END
   END ;
IF (WL > 0) AND (WLEN > 0)
   THEN
      BEGIN
         J := WLEN ;
         WHOLE := 0.0 ;
         MUL := 1.0 ;
         WHILE J > 0 DO
```

```
              BEGIN
                 WHOLE := WHOLE + (TRUNC(ORD(ENTRY[J])-48))*MUL ;
                 MUL := MUL * 10.0 ;
                 J := J - 1
              END;
           FRAC := 0.0 ;
           MUL := 10.0 ;
           J := WLEN + 1;
           WHILE (J <= WLEN+FLEN) DO
              BEGIN
                 FRAC := FRAC + (TRUNC(ORD(ENTRY[J])-48))/MUL ;
                 MUL := MUL * 10.0 ;
                 J := J + 1
              END ;
           NUMBER := WHOLE + FRAC ;
           LAST_NUM := NUMBER ;
        END ;
  END ;

{------------------------------------------------------------------------}

FUNCTION F_TO_A(VLU : REAL ; WL , FLEN : INTEGER) : MESSAGE ;
  VAR
     SIGN , I , J , K , WLEN  : INTEGER ;
     TEMP1 , TEMP2 , FACTOR  : REAL     ;
     ALPHA , ALPHA2 : MESSAGE ;
  BEGIN
     SIGN  := 1 ;
     IF VLU < 0.0 THEN BEGIN
                          VLU  := - VLU ;
                         SIGN := -1
                       END ;
     TEMP1 := VLU ;
     WLEN  := 0 ;
     WHILE NOT (TEMP1 < 1.0) DO BEGIN
                                  TEMP1 := TEMP1 / 10.0 ;
                                  WLEN  := WLEN + 1
                                END ;
     TEMP1 := VLU ;
     TEMP2 := VLU ;
     FOR J := 1 TO WLEN DO BEGIN
                              K := J ;
                              FACTOR := 1.0 ;
                              WHILE K < WLEN DO BEGIN
                                                   FACTOR := FACTOR * 10.0 ;
                                                   K := K + 1
                                                 END ;
                              TEMP1 := TEMP2 / FACTOR ;
                              ALPHA[J] := CHR(TRUNC(TEMP1)+48) ;
                              TEMP2 := TEMP2 - (FACTOR * TRUNC(TEMP1))
                            END ;
     IF WLEN = 0
       THEN BEGIN
               FOR J := 1 TO WL DO
                 BEGIN
                   WLEN := WLEN + 1 ;
                   ALPHA[WLEN] := '0'
                 END
            END ;
     ALPHA[WLEN+1] := '.' ;
     FOR I := 1 TO FLEN DO BEGIN
                              TEMP2 := TEMP2 * 10.0 ;
                              ALPHA[WLEN+1+I] := CHR(TRUNC(TEMP2)+48) ;
                              TEMP2 := TEMP2 - TRUNC(TEMP2)
                            END ;
     ALPHA[0] := CHR(WLEN+1+FLEN) ;
     IF SIGN = -1 THEN BEGIN
                          FOR I := (WLEN+1+FLEN) DOWNTO 1 DO ALPHA[I+1] := ALPHA[I] ;
                          ALPHA[1] := '-' ;
                          ALPHA[0] := CHR(WLEN+1+FLEN+1)
                       END ;
     IF SIGN = -1
       THEN I := 2
       ELSE I := 1 ;
```

```
      WHILE ((ALPHA[I] = '0') AND (ALPHA[I+1] = '0')) DO I := I + 1 ;
      IF SIGN = -1
         THEN J := 2
         ELSE J := 1 ;
      FOR K := J TO ORD(ALPHA[0]) DO ALPHA[K] := ALPHA[K+I-J] ;
      ALPHA[0] := CHR(ORD(ALPHA[0]) + J - I) ;

      I := 1 ;
      WHILE ALPHA[I] <> '.' DO I := I + 1 ;
      J := WL - I + 1 ;
      FOR I := 1 TO J DO ALPHA2[I] := ' ' ;
      FOR I := 1 TO ORD(ALPHA[0]) DO ALPHA2[I+J] := ALPHA[I] ;
      ALPHA2[0] := CHR(ORD(ALPHA[0]) + J) ;
      F_TO_A    := ALPHA2 ;
   END ;

{---------------------------------------------------------------------------}

FUNCTION I_TO_A(IVLU : INTEGER) : MESSAGE ;
   VAR
      SIGN  , I , J , K , WLEN  : INTEGER ;
      TEMP1 , TEMP2 , FACTOR  : INTEGER    ;
      ALPHA : MESSAGE ;
   BEGIN
      IF IVLU < 0 THEN BEGIN
                         IVLU := - IVLU ;
                         SIGN := -1
                      END ;
      IF IVLU = 0 THEN BEGIN
                         ALPHA[0] := CHR(1) ;
                         ALPHA[1] := '0' ;
                         I_TO_A   := ALPHA
                      END
                 ELSE BEGIN
                         TEMP1 := IVLU ;
                         WLEN  := 0 ;
                         WHILE NOT (TEMP1 < 1) DO BEGIN
                                                     TEMP1 := TEMP1 DIV 10 ;
                                                     WLEN  := WLEN + 1
                                                  END ;
                         TEMP1 := IVLU ;
                         TEMP2 := IVLU ;
                         FOR J := 1 TO WLEN DO BEGIN
                                                  K := J ;
                                                  FACTOR := 1 ;
                                                  WHILE K < WLEN DO BEGIN
                                                                      FACTOR := FACTOR * 10 ;
                                                                      K := K + 1
                                                                   END ;
                                                  TEMP1 := TEMP2 DIV FACTOR ;
                                                  ALPHA[J] := CHR(TEMP1+48) ;
                                                  TEMP2 := TEMP2 - (FACTOR * TEMP1)
                                               END ;
                         ALPHA[0] := CHR(WLEN) ;
                         IF SIGN = -1 THEN BEGIN
                                              FOR I := WLEN DOWNTO 1 DO ALPHA[I+1] := ALPHA[I] ;
                                              ALPHA[1] := '-' ;
                                              ALPHA[0] := CHR(WLEN+1)
                                           END ;
                         I_TO_A   := ALPHA
                      END
   END ;

{---------------------------------------------------------------------------}

FUNCTION A_TO_F(ASCII : MESSAGE) : REAL ;
   VAR
      I , J , K , L , M : INTEGER ;
      INDEX : INTEGER ;
      SIGN  : REAL    ;
      WSTRG : STRING[5] ;
```

```
      FSTRG : STRING[5] ;
      WHOLE : INTEGER ;
      FRAC  : INTEGER ;
      FRACR : REAL    ;
      TEMP  : INTEGER ;
   BEGIN
      J := 1 ;
      SIGN := 1.0 ;
      IF ((ASCII[J] = '+') OR (ASCII[J] = '-'))
         THEN
            BEGIN
               IF ASCII[J] = '-' THEN SIGN := -1.0 ;
               J := J + 1 ;
            END ;
      K := 0 ;
      WHILE ASCII[J] <> '.' DO
         BEGIN
            K := K + 1 ;
            WSTRG[K] := ASCII[J] ;
            J := J + 1 ;
         END ;
      WHOLE := 0 ;
      FOR L := 1 TO K DO
         BEGIN
            TEMP := ORD(WSTRG[L]) - 48 ;
            FOR  M := L TO K - 1 DO TEMP := TEMP * 10 ;
            WHOLE := WHOLE + TEMP
         END ;
      J := J + 1 ;
      K := 0 ;
      WHILE (J <= ORD(ASCII[0])) DO
         BEGIN
            K := K + 1 ;
            FSTRG[K] := ASCII[J] ;
            J := J + 1 ;
         END ;
      FRAC  := 0 ;
      FOR L := 1 TO K DO
         BEGIN
            TEMP := ORD(FSTRG[L]) - 48 ;
            FOR  M := L TO K - 1 DO TEMP := TEMP * 10 ;
            FRAC  := FRAC  + TEMP
         END ;
      FRACR := FRAC ;
      FOR L := 1 TO K DO FRACR := FRACR / 10.0 ;
      A_TO_F := SIGN * (WHOLE + FRACR)  ;
   END ;

{----------------------------------------------------------------------------}

{----------------------------------------------------------------------------}

FUNCTION COM_READY   : BOOLEAN ;
   VAR
      LINE,MODEM : BYTE ;
      I , J : INTEGER ;
      REGS  : REGISTERS ;
   BEGIN
      COM_READY := FALSE ;
      REGS.AX := $0300 ;
      REGS.DX := $0000 ;
      INTR($14,REGS) ;
      LINE    := HI(REGS.AX) ;
      MODEM   := LO(REGS.AX) ;
{
      WRITELN ('COM READY : LINE = ',LINE:3,' MODEM = ',MODEM:3) ;

      LINE := LINE AND $01 ;
      WRITELN(LINE) ;
 }
      IF (LINE AND $01) = 1
         THEN COM_READY := TRUE ;
   END ;

{----------------------------------------------------------------------------}
```

```
PROCEDURE COM_PAR(BAUD , PARITY , STOPS , NDATA : INTEGER) ;
   VAR
      PARAMETERS : BYTE ;
      I , J : INTEGER ;
      REGS  : REGISTERS ;
   BEGIN
      PARAMETERS := 0 ;
      CASE BAUD OF
          110 : PARAMETERS := $00 ;
          150 : PARAMETERS := $20 ;
          300 : PARAMETERS := $40 ;
          600 : PARAMETERS := $60 ;
         1200 : PARAMETERS := $80 ;
         2400 : PARAMETERS := $A0 ;
         4800 : PARAMETERS := $C0 ;
         9600 : PARAMETERS := $E0 ;
      END ; { case }
      CASE PARITY OF
          0 : PARAMETERS := PARAMETERS + $00 ;   { none }
          1 : PARAMETERS := PARAMETERS + $08 ;   { odd  }
          2 : PARAMETERS := PARAMETERS + $18 ;   { even }
      END ; { case }
      CASE STOPS OF
          1 : PARAMETERS := PARAMETERS + $00 ;
          2 : PARAMETERS := PARAMETERS + $04 ;
      END ; { case }
      CASE NDATA OF
          7 : PARAMETERS := PARAMETERS + $02 ;
          8 : PARAMETERS := PARAMETERS + $03 ;
      END ; { case }
      REGS.AX := $0000 + PARAMETERS ;
      REGS.DX := $0000 ;
      INTR($14,REGS) ;
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE SET_PAR ;
   BEGIN
      CURR_PAGE := 1 ;
      SEL_PAGE ;
      CLS(0,0,24,79,0) ;
      WRCOL(1,40,3,'Communications Parameters') ;
      WRCOL(2,40,3,'_________________________') ;
      WRCOL(24,1,3,'F1 : Baud      F3 : Parity      F5 : Stop Bits     F7 : Data Bits      ESC = Exit') ;
      WRCOL(5,40,3,'Baud Rate = ') ;
      CASE BAUD OF
          110 : WRCOL(5,52,3,'110 ') ;
          150 : WRCOL(5,52,3,'150 ') ;
          300 : WRCOL(5,52,3,'300 ') ;
          600 : WRCOL(5,52,3,'600 ') ;
         1200 : WRCOL(5,52,3,'1200') ;
         2400 : WRCOL(5,52,3,'2400') ;
         4800 : WRCOL(5,52,3,'4800') ;
         9600 : WRCOL(5,52,3,'9600') ;
      END ; { case }
      WRCOL(6,40,3,'Parity = ') ;
      CASE PARITY OF
         0 : WRCOL(6,49,3,'None') ;
         1 : WRCOL(6,49,3,'Odd ') ;
         2 : WRCOL(6,49,3,'Even') ;
      END ; { case }
      WRCOL(7,40,3,'Stop Bits = ') ;
      CASE NSTOP OF
         1 : WRCOL(7,52,3,'1') ;
         2 : WRCOL(7,52,3,'2') ;
      END ; { case }
      WRCOL(8,40,3,'Data Bits = ') ;
      CASE NDATA OF
         7 : WRCOL(8,52,3,'7') ;
         8 : WRCOL(8,52,3,'8') ;
      END ; { case }
```

```
      KEY := 127 ;
      GETC(KEY) ;
      WHILE KEY_PRESSED <> ESC DO
         BEGIN
            CASE KEY_PRESSED OF
               F1 : BEGIN
                       CASE BAUD OF
                            110 : BAUD := 150   ;
                            150 : BAUD := 300   ;
                            300 : BAUD := 600   ;
                            600 : BAUD := 1200  ;
                           1200 : BAUD := 2400  ;
                           2400 : BAUD := 4800  ;
                           4800 : BAUD := 9600  ;
                           9600 : BAUD := 110   ;
                       END ; { case }
                       CASE BAUD OF
                            110 : WRCOL(5,52,3,'110 ') ;
                            150 : WRCOL(5,52,3,'150 ') ;
                            300 : WRCOL(5,52,3,'300 ') ;
                            600 : WRCOL(5,52,3,'600 ') ;
                           1200 : WRCOL(5,52,3,'1200') ;
                           2400 : WRCOL(5,52,3,'2400') ;
                           4800 : WRCOL(5,52,3,'4800') ;
                           9600 : WRCOL(5,52,3,'9600') ;
                       END ; { case }
                    END ; { F1 }
               F3 : BEGIN
                       CASE PARITY OF
                          0 : PARITY := 1 ;
                          1 : PARITY := 2  ;
                          2 : PARITY := 0  ;
                       END ;
                       CASE PARITY OF
                          0 : WRCOL(6,49,3,'None') ;
                          1 : WRCOL(6,49,3,'Odd ') ;
                          2 : WRCOL(6,49,3,'Even') ;
                       END ;
                    END ; { F3 }
               F5 : BEGIN
                       CASE NSTOP OF
                          1 : NSTOP := 2 ;
                          2 : NSTOP := 1 ;
                       END ; { case }
                       CASE NSTOP OF
                          1 : WRCOL(7,52,3,'1') ;
                          2 : WRCOL(7,52,3,'2') ;
                       END ; { case }
                    END ; { F5 }
               F7 : BEGIN
                       CASE NDATA OF
                          7 : NDATA := 8 ;
                          8 : NDATA := 7 ;
                       END ; { case }
                       CASE NDATA OF
                          7 : WRCOL(8,52,3,'7') ;
                          8 : WRCOL(8,52,3,'8') ;
                       END ; { case }
                    END ;
               ELSE BEGIN END ;
            END ; { case key of }
            GETC(KEY)
        END ;
      COM_PAR(BAUD,PARITY,NSTOP,NDATA) ;
      CURR_PAGE := 0 ;
      SEL_PAGE ;
   END ;

{-------------------------------------------------------------------------------}

PROCEDURE COM_STAT(VAR LINE , MODEM : BYTE)  ;
   VAR
      REGS  : REGISTERS ;
   BEGIN
      REGS.AX := $0300 ;
```

```
      REGS.DX := $0000 ;
      INTR($14,REGS) ;
      LINE := HI(REGS.AX) ;
      MODEM := LO(REGS.AX) ;
   END ;

{----------------------------------------------------------------------}

PROCEDURE PUTC_COM(CH : CHAR )  ;
   VAR
      REGS   : REGISTERS ;
      DONE   : BOOLEAN   ;
   BEGIN
      REGS.AX := $0100 ;
      REGS.AX := REGS.AX + ORD(CH) ;
      REGS.DX := $0000 ;
      INTR($14,REGS) ;
   END ;

{----------------------------------------------------------------------}

PROCEDURE GETC_COM(VAR CH : CHAR ; VAR AVAIL : BOOLEAN)  ;
   VAR
      REGS   : REGISTERS ;
      DONE   : BOOLEAN   ;
   BEGIN
      IF COM_READY
         THEN
            BEGIN
               REGS.AX := $0200 ;
               REGS.DX := $0000 ;
               INTR($14,REGS) ;
               CH  := CHR(LO(REGS.AX) AND $7F) ;
               AVAIL := TRUE ;
            END
         ELSE AVAIL := FALSE ;
      IF AVAIL
         THEN
            IF FULL_DUPLEX
               THEN PUTC_COM(CH) ;
   END ;

{----------------------------------------------------------------------}

PROCEDURE WRS_COM(MESS : MESSAGE) ;
   VAR
      I  : INTEGER    ;
      CH : CHAR       ;
   BEGIN
      FOR I := 1 TO ORD(MESS[0]) DO
         BEGIN
            PUTC_COM(MESS[I]) ;
            GETC_COM(CH , CHAR_AVAIL) ;
            DELAY(200) ;
         END ;
   END ;
{----------------------------------------------------------------------}

PROCEDURE INIT_CAMPBELL ;
   VAR
      CH1 , CH2 : CHAR ;
      DONE : BOOLEAN   ;
   BEGIN
      COM_PAR(BAUD,PARITY,NSTOP,NDATA) ;
      CH1 := CHR(13) ;
      CHAR_AVAIL := FALSE ;
      DONE := FALSE ;
      WHILE NOT DONE DO
         BEGIN
            PUTC_COM(CH1) ;
            GETC_COM(CH2 , CHAR_AVAIL) ;
            IF CHAR_AVAIL
               THEN
                  BEGIN
                     WRITE(CH2) ;
```

```
                        DONE := TRUE ;
                     END
                  ELSE DELAY(100)
            END ;
      END ;

{-----------------------------------------------------------------------------}

PROCEDURE GO_REMOTE ;
   VAR
      CH : CHAR ;
   BEGIN
      INIT_CAMPBELL ;
      WRS_COM('2718H') ;
      CH := CHR(13) ;
      PUTC_COM(CH) ;
      GETC_COM(CH , CHAR_AVAIL) ;
      DELAY(200) ;
      PUTC_COM('*') ;
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE OPEN_COM ;
   BEGIN
      ASSIGN(SERIAL_PORT,'COM1') ;
      REWRITE(SERIAL_PORT)
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE CLOSE_COM ;
   BEGIN
      CLOSE(SERIAL_PORT)
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE SEND_MODE_1 ;
   BEGIN
      GO_REMOTE ;
      DELAY(200) ;
{
      PUTC_COM('1') ;
      PUTC_COM('A') ;
      PUTC_COM('D') ;
      PUTC_COM('0') ;
      PUTC_COM('1') ;
      PUTC_COM('2') ;
      PUTC_COM('5') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('6') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('2') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('1') ;
      PUTC_COM('1') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('1') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('1') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('2') ;
      PUTC_COM('0') ;
      PUTC_COM('0') ;
      PUTC_COM('0') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('1') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('7') ;
      PUTC_COM(CHR(13)) ;
      PUTC_COM('2') ;
      PUTC_COM('.') ;
      PUTC_COM('5') ;
      PUTC_COM(CHR(13)) ;
```

```
}
  END ;
{-----------------------------------------------------------------------------}


{-----------------------------------------------------------------------------}
FUNCTION DEFINED(CHANNEL : INTEGER) : BOOLEAN ;
   BEGIN
      IF ((CHANNEL_INFO[CHANNEL].TRANSDUCER <> 0)    AND
          (CHANNEL_INFO[CHANNEL].DETAIL     <> 0)    AND
          (CHANNEL_INFO[CHANNEL].SR_MAX      > 0.0) AND
          (CHANNEL_INFO[CHANNEL].SR_MIN      > 0.0))
         THEN DEFINED := TRUE
         ELSE DEFINED := FALSE
   END ;
{-----------------------------------------------------------------------------}
PROCEDURE LOAD_FILE(ROW , COL : INTEGER) ;
   VAR
      DONE : BOOLEAN ;
      TEMP_NAME : STRING[50] ;
      CHANNEL : INTEGER ;
      I : INTEGER ;
      CH : CHAR    ;
   BEGIN
      DONE := FALSE ;
      CLS(ROW,0,ROW,79,0) ;
      REPEAT
         IF (FNAME[0] <> CHR(0))
            THEN
               BEGIN
                  FOR I := 1 TO ORD(FNAME[0])-2 DO TEMP_NAME[I] := FNAME[I+2] ;
                  TEMP_NAME[0] := CHR(ORD(FNAME[0])-2)
               END ;
         WRCOL(ROW,COL,3,'Data File : ') ;
         CURSOR(ROW,COL+12) ;
         FNAME[0] := CHR(0) ;
         GETNAM ;
         IF (FNAME[0] <> CHR(0)) { file specified }
            THEN
               BEGIN
                  ASSIGN(DESCRIPTION_FILE,FNAME) ;
                  {$I-}
                  RESET(DESCRIPTION_FILE)
                  {$I+} ;
                  DONE := (IOresult = 0) ;
                  IF NOT DONE
                     THEN WRCOL(ROW,COL+25,4, 'File Not Found ')
                     ELSE
                        BEGIN
                           WRCOL(ROW,COL+25,4,'Reading Data.. ') ;
                           CHANNELS := 0 ;
                           READLN(DESCRIPTION_FILE,START_TIME.HOUR,START_TIME.MINUTE,RAIN_INTERVAL) ;
                           FOR CHANNEL := 1 TO 8 DO ACTIVE_CHANNELS[CHANNEL] := -1 ; { inactive }
                           FOR CHANNEL := 1 TO 8 DO
                              BEGIN
                                 READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].ID)          ;
                                 READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].TRANSDUCER)  ;
                                 READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].DETAIL)      ;
                                 READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].EXCITATION)  ;
                                 READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].EX_CHANNEL)  ;
                                 READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].ROW)         ;
```

```
                    READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].SR_MAX)          ;
                    READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].SR_MIN)          ;
                    CASE CHANNEL_INFO[CHANNEL].TRANSDUCER OF
                       0 : CHANNEL_INFO[CHANNEL].EXCITATION := -1 ;
                       1 : BEGIN { strain gage }
                              CHANNEL_INFO[CHANNEL].EXCITATION := DEFAULT_SG_EXCITATION ;
                              CHANNEL_INFO[CHANNEL].RANGE      := DEFAULT_SG_RANGE      ;

                           END ;
                        2 : BEGIN { transducer }
                               CHANNEL_INFO[CHANNEL].EXCITATION := DEFAULT_DT_EXCITATION ;
                               CHANNEL_INFO[CHANNEL].RANGE      := DEFAULT_DT_RANGE      ;
                            END ;
                     END ; { case }

                     IF ((CHANNEL_INFO[CHANNEL].DETAIL < 7) AND
                        (CHANNEL_INFO[CHANNEL].DETAIL > 0))
                        THEN
                           BEGIN
                              SRMAX[CHANNEL_INFO[CHANNEL].DETAIL] := CHANNEL_INFO[CHANNEL].SR_
                              SRMIN[CHANNEL_INFO[CHANNEL].DETAIL] := CHANNEL_INFO[CHANNEL].SR_
                           END ;

                          READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].OFFSET)         ;
                          READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].MULTIPLIER)     ;
                          READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].S_CALIB)        ;
                          READLN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].MVPERV_CALIB)   ;
                          IF DEFINED(CHANNEL)
                             THEN
                                BEGIN
                                   CHANNELS := CHANNELS + 1          ;
                                   ACTIVE_CHANNELS[CHANNELS] := CHANNEL ;
                                END ;
                       END ;
                    PROGRAMMED := FALSE ;   { new program needs to be loaded }
                    ZEROED     := FALSE ;   { new zeroes need be taken       }
                    CLOSE(DESCRIPTION_FILE) ;
                    WRCOL(ROW,COL+25,4,'                   ') ;
                    DONE := TRUE ;
                 END ; { else }
           END
        ELSE DONE := TRUE ;
     UNTIL DONE ;
     IF (FNAME[0] <> CHR(0))
        THEN
           BEGIN
              FOR I := 1 TO ORD(FNAME[0])-2 DO TEMP_NAME[I] := FNAME[I+2] ;
              TEMP_NAME[0] := CHR(ORD(FNAME[0])-2) ;
              WRCOL(ROW,COL+12,3,'                ') ;
              WRCOL(ROW,COL+12,3,TEMP_NAME) ;
           END ;
  END ;

{------------------------------------------------------------------------}

PROCEDURE SAVE_DESCRIPTION(ROW , COL : INTEGER) ;
  VAR
     DONE , DN , FOUND : BOOLEAN ;
     TEMP_NAME : STRING[50] ;
     I : INTEGER ;
     CH : CHAR   ;
  PROCEDURE RECORD_INFO ;
     VAR
```

```
         CHANNEL : INTEGER ;
      BEGIN
         WRCOL(ROW,COL+25,4,'Please wait ... Saving file to disk. ') ;
         REWRITE(DESCRIPTION_FILE) ;
         WRITELN(DESCRIPTION_FILE,START_TIME.HOUR:2,'   ',START_TIME.MINUTE:2,'   ',RAIN_INTERVAL:4)
         FOR CHANNEL := 1 TO 8 DO
            BEGIN
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].ID:1)             ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].TRANSDUCER)       ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].DETAIL)           ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].EXCITATION)       ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].EX_CHANNEL)       ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].ROW)              ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].SR_MAX)           ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].SR_MIN)           ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].OFFSET)           ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].MULTIPLIER)       ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].S_CALIB)          ;
               WRITELN(DESCRIPTION_FILE,CHANNEL_INFO[CHANNEL].MVPERV_CALIB)     ;
            END ;
         CLOSE(DESCRIPTION_FILE) ;
         WRCOL(ROW,COL+25,4,'                                              ') ;
      END ; { record }

   BEGIN
      DONE := FALSE ;
      REPEAT
         IF (FNAME[0] <> CHR(0))
            THEN
               BEGIN
                  FOR I := 1 TO ORD(FNAME[0])-2 DO TEMP_NAME[I] := FNAME[I+2] ;
                  TEMP_NAME[0] := CHR(ORD(FNAME[0])-2)
               END ;
         CLS(ROW,0,ROW,79,0) ;
         WRCOL(ROW,COL,3,'Save File : ') ;
{
         IF (FNAME[0] <> CHR(0))
            THEN WRCOL(ROW,COL+12,4,TEMP_NAME) ;
}
         CURSOR(ROW,COL+12) ;
         FNAME[0] := CHR(0) ;
         GETNAM ;
         IF (FNAME[0] <> CHR(0)) { file specified }
            THEN
               BEGIN
                  ASSIGN(DESCRIPTION_FILE,FNAME) ;
                  {$I-}
                  RESET(DESCRIPTION_FILE) ;
                  CLOSE(DESCRIPTION_FILE) ;
                  {$I+} ;
                  FOUND := (IOresult = 0) ;
                  IF FOUND
                     THEN  { duplicate file exists }
                        BEGIN
                           WRCOL(ROW,COL+25,4,'File exists ... Overwrite (Y or N) ? ') ;
                           DN := FALSE ;
                           WHILE NOT DN DO
                              BEGIN
                                 GETC(KEY) ;
                                 IF CHR(KEY) IN ['Y','y','N','n'] THEN DN := TRUE ;
                              END ;
                           IF CHR(KEY) IN ['Y','y']
                              THEN { overwite file }
                                 BEGIN
                                    RECORD_INFO ;
                                    DONE := TRUE ;
                                 END ;
                        END { then }
                     ELSE
                        BEGIN
                           RECORD_INFO ;
                           DONE := TRUE ;
```

```
                        END ;
                 END
              ELSE DONE := TRUE ;
        UNTIL DONE ;
        IF (FNAME[0] <> CHR(0))
           THEN
              BEGIN
                 FOR I := 1 TO ORD(FNAME[0])-2 DO TEMP_NAME[I] := FNAME[I+2] ;
                 TEMP_NAME[0] := CHR(ORD(FNAME[0])-2) ;
                 WRCOL(ROW,COL+12,3,'            ') ;
                 WRCOL(ROW,COL+12,3,TEMP_NAME) ;
              END ;
     END ;

{---------------------------------------------------------------------------}

PROCEDURE SHOW_VALUES ;
     VAR
        ROW , CHANNEL , COL : INTEGER ;
        CAT : STRING[4] ;
     BEGIN
        ROW := TABLE_START_ROW + 4 ;
        FOR CHANNEL := 1 TO 8 DO
           BEGIN
              CASE CHANNEL_INFO[CHANNEL].TRANSDUCER OF
                 0 : WRCOL(ROW,CHANNEL_INFO[CHANNEL].TRANSDUCER_COL,3,'  Undefined  ') ;
                 1 : WRCOL(ROW,CHANNEL_INFO[CHANNEL].TRANSDUCER_COL,3,' Strain Gage ') ;
                 2 : WRCOL(ROW,CHANNEL_INFO[CHANNEL].TRANSDUCER_COL,3,'  Transducer ') ;
              END ;
              COL := CHANNEL_INFO[CHANNEL].DETAIL_COL ;
              WRCOL(ROW,COL-15,3,'       ') ;
              IF CHANNEL_INFO[CHANNEL].S_CALIB > -9999.0
                 THEN WRCOL(ROW,COL-15,3,F_TO_A(CHANNEL_INFO[CHANNEL].S_CALIB,2,2))
                 ELSE WRCOL(ROW,COL-13,3,'?') ;
              WRCOL(ROW,COL-7,3,'       ') ;
              IF CHANNEL_INFO[CHANNEL].MVPERV_CALIB > -9999.0
                 THEN WRCOL(ROW,COL-7,3,F_TO_A(CHANNEL_INFO[CHANNEL].MVPERV_CALIB,2,2))
                 ELSE WRCOL(ROW,COL-5,3,'?') ;
              IF CHANNEL_INFO[CHANNEL].DETAIL = 0
                 THEN BEGIN
                         WRCOL(ROW,COL,3,'Undefined') ;
                      END
                 ELSE BEGIN
                         WRCOL(ROW,COL,3,'         ') ;
                         WRCOL(ROW,COL+4,3,I_TO_A(CHANNEL_INFO[CHANNEL].DETAIL)) ;
                      END ;
              WRCOL(ROW,COL+12,3,'       ') ;
              IF CHANNEL_INFO[CHANNEL].SR_MAX > 0.0
                 THEN WRCOL(ROW,COL+12,3,F_TO_A(CHANNEL_INFO[CHANNEL].SR_MAX,2,2))
                 ELSE WRCOL(ROW,COL+12,3,'   ?   ') ;
              WRCOL(ROW,COL+20,3,'       ') ;
              IF CHANNEL_INFO[CHANNEL].SR_MIN > 0.0
                 THEN WRCOL(ROW,COL+20,3,F_TO_A(CHANNEL_INFO[CHANNEL].SR_MIN,2,2))
                 ELSE WRCOL(ROW,COL+20,3,'   ?   ') ;
              ROW := ROW + 2 ;
           END ;
     END ;
{---------------------------------------------------------------------------}

PROCEDURE SHOW_TABLE ;
     VAR
        CHANNEL , ROWS , ROW , COL : INTEGER ;
        CAT : STRING[4] ;
     BEGIN
        CLS(0,0,24,79,0) ;
        WRCOL(0,0,HILITE,TITLE) ;
        WRCOL(FKEY_ROW-1,0,3,SOLID_LINE) ;
        WRCOL(FKEY_ROW,0,3,FKEYS)       ;
        ROW := TABLE_START_ROW          ;
        COL := TABLE_START_COLUMN       ;
```

```
      WRCOL(ROW,COL,3,TABLE_TOP)        ;
      ROW := ROW + 1                    ;
      TABLE_MIDDLE[32] := ' '           ;
      WRCOL(ROW,COL,3,TABLE_MIDDLE)     ;
      ROW := ROW + 1                    ;
      WRCOL(ROW,COL,3,TABLE_MIDDLE)     ;
      TABLE_MIDDLE[32] := CHR(179)      ;
      ROW := ROW + 1                    ;
      WRCOL(ROW,COL,3,TABLE_TOP_2)      ;
      ROW := ROW + 1                    ;
      FOR ROWS := 1 TO 7 DO
         BEGIN
            WRCOL(ROW,COL,3,TABLE_MIDDLE) ;
            WRCOL(ROW+1,COL,3,TABLE_LINE) ;
            ROW := ROW + 2 ;
         END ;
      WRCOL(ROW,COL,3,TABLE_MIDDLE) ;
      WRCOL(ROW+1,0,3,TABLE_BOTTOM) ;

      WRCOL(TABLE_START_ROW+1 ,TABLE_START_COLUMN+1,3,'Channel') ;
      WRCOL(TABLE_START_ROW+2 ,TABLE_START_COLUMN+1,3,'  no.   ') ;
      WRCOL(TABLE_START_ROW+1 ,TABLE_START_COLUMN+11,3,'  Channel ') ;
      WRCOL(TABLE_START_ROW+2 ,TABLE_START_COLUMN+11,3,'    Type    ') ;
      WRCOL(TABLE_START_ROW+1 ,TABLE_START_COLUMN+26,3,'Calibration') ;
      WRCOL(TABLE_START_ROW+2 ,TABLE_START_COLUMN+27,3,'S     *MV/V') ;
      WRCOL(TABLE_START_ROW+1 ,TABLE_START_COLUMN+41,3,'Fatige') ;
      WRCOL(TABLE_START_ROW+2 ,TABLE_START_COLUMN+41,3,'Detail #') ;
      WRCOL(TABLE_START_ROW+1 ,TABLE_START_COLUMN+53,3,'Sr ') ;
      WRCOL(TABLE_START_ROW+2 ,TABLE_START_COLUMN+53,3,'max') ;
      WRCOL(TABLE_START_ROW+1 ,TABLE_START_COLUMN+60,3,'Sr ') ;
      WRCOL(TABLE_START_ROW+2 ,TABLE_START_COLUMN+60,3,'min') ;
      WRCOL(TABLE_START_ROW+4 ,TABLE_START_COLUMN+4,3,'1') ;
      WRCOL(TABLE_START_ROW+6 ,TABLE_START_COLUMN+4,3,'2') ;
      WRCOL(TABLE_START_ROW+8 ,TABLE_START_COLUMN+4,3,'3') ;
      WRCOL(TABLE_START_ROW+10,TABLE_START_COLUMN+4,3,'4') ;
      WRCOL(TABLE_START_ROW+12,TABLE_START_COLUMN+4,3,'5') ;
      WRCOL(TABLE_START_ROW+14,TABLE_START_COLUMN+4,3,'6') ;
      WRCOL(TABLE_START_ROW+16,TABLE_START_COLUMN+4,3,'7') ;
      WRCOL(TABLE_START_ROW+18,TABLE_START_COLUMN+4,3,'8') ;
      SCREEN_SET := TRUE ;

   END ;

{---------------------------------------------------------------------------}

PROCEDURE UPDATE_SCREEN_SR ;
   VAR
      I : INTEGER ;
   BEGIN
      FOR I := 1 TO 8 DO
         BEGIN
            IF ((CHANNEL_INFO[I].DETAIL < 7) AND
               (CHANNEL_INFO[I].DETAIL > 0))
               THEN
                  BEGIN
                     CHANNEL_INFO[I].SR_MAX := SRMAX[CHANNEL_INFO[I].DETAIL] ;
                     CHANNEL_INFO[I].SR_MIN := SRMIN[CHANNEL_INFO[I].DETAIL] ;
                  END ;
         END ;
      SHOW_VALUES ;
   END ;

{---------------------------------------------------------------------------}

PROCEDURE GET_TRANSDUCER(VAR TRANSDUCER : INTEGER ; ROW , COL : INTEGER) ;
   VAR
      DONE : BOOLEAN ;
   BEGIN
      WRCOL(MESSAGE_ROW,0,3,TRANS_MESSAGE) ;
      CASE TRANSDUCER OF
         0 : WRCOL(ROW,COL,HILITE,'  Undefined  ') ;
         1 : WRCOL(ROW,COL,HILITE,' Strain Gage ') ;
         2 : WRCOL(ROW,COL,HILITE,'  Transducer ') ;
      END ; { case }
      CURSOR(ROW,COL) ;
```

```
      DONE := FALSE ;
      WHILE NOT DONE DO
         BEGIN
            GETC(KEY) ;
            IF CHR(KEY) IN ['g','G','t','T']
               THEN
                  BEGIN
                     IF (CHR(KEY) = 'G') OR (CHR(KEY) = 'g')
                        THEN TRANSDUCER := 1
                        ELSE TRANSDUCER := 2 ;
                     DONE := TRUE ;
                     PROGRAMMED := FALSE ; { parameter change needs to be programmed }
                  END
               ELSE IF SPECIAL(KEY) THEN DONE := TRUE ;
         END ;
      CASE TRANSDUCER OF
         0 : WRCOL(ROW,COL,3,'  Undefined  ') ;
         1 : WRCOL(ROW,COL,3,' Strain Gage ') ;
         2 : WRCOL(ROW,COL,3,'  Transducer ') ;
      END ; { case }
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE GET_SR_MAX(CHANNEL : INTEGER) ;
   VAR
      TEMPREAL  : REAL    ;
      DONE      : BOOLEAN ;
      ROW , COL : INTEGER ;
   BEGIN
      ROW := CHANNEL_INFO[CHANNEL].ROW ;
      COL := CHANNEL_INFO[CHANNEL].DETAIL_COL + 12 ;
      WRCOL(MESSAGE_ROW,0,3,SR_MAX_MESSAGE) ;
      IF CHANNEL_INFO[CHANNEL].SR_MAX > 0.0
         THEN WRCOL(ROW,COL,HILITE,F_TO_A(CHANNEL_INFO[CHANNEL].SR_MAX,2,2))
         ELSE WRCOL(ROW,COL+2,HILITE,'?') ;
      CURSOR(ROW,COL) ;
      DONE := FALSE ;
      TEMPREAL := -9999.0 ;
      GETNUM(TEMPREAL,2,2) ;
      IF TEMPREAL > 0.0
         THEN
            BEGIN
               CHANNEL_INFO[CHANNEL].SR_MAX := TEMPREAL ;
               IF ((CHANNEL_INFO[CHANNEL].DETAIL < 7) AND
                   (CHANNEL_INFO[CHANNEL].DETAIL > 0))
                  THEN
                     BEGIN
                        SRMAX[CHANNEL_INFO[CHANNEL].DETAIL] := CHANNEL_INFO[CHANNEL].SR_MAX ;
                        UPDATE_SCREEN_SR ;
                     END ;
               PROGRAMMED := FALSE ; [ parameter change needs to be programmed }
            END ;
      WRCOL(ROW,COL,3,'       ') ;
      IF CHANNEL_INFO[CHANNEL].SR_MAX > 0.0
         THEN WRCOL(ROW,COL,3,F_TO_A(CHANNEL_INFO[CHANNEL].SR_MAX,2,2))
         ELSE WRCOL(ROW,COL+2,3,'?') ;
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE GET_SR_MIN(CHANNEL : INTEGER) ;
   VAR
      TEMPREAL  : REAL    ;
      DONE      : BOOLEAN ;
      ROW , COL : INTEGER ;
   BEGIN
      ROW := CHANNEL_INFO[CHANNEL].ROW ;
      COL := CHANNEL_INFO[CHANNEL].DETAIL_COL + 20 ;
      WRCOL(MESSAGE_ROW,0,3,SR_MIN_MESSAGE) ;
      IF CHANNEL_INFO[CHANNEL].SR_MIN > 0.0
         THEN WRCOL(ROW,COL,HILITE,F_TO_A(CHANNEL_INFO[CHANNEL].SR_MIN,2,2))
```

```
         ELSE WRCOL(ROW,COL+2,HILITE,'?') ;
      CURSOR(ROW,COL) ;
      DONE := FALSE ;
      TEMPREAL := -9999.0 ;
      GETNUM(TEMPREAL,2,2) ;
      IF TEMPREAL > 0.0
         THEN
            BEGIN
               CHANNEL_INFO[CHANNEL].SR_MIN := TEMPREAL ;
               IF ((CHANNEL_INFO[CHANNEL].DETAIL < 7) AND
                   (CHANNEL_INFO[CHANNEL].DETAIL > 0))
                  THEN
                     BEGIN
                        SRMIN[CHANNEL_INFO[CHANNEL].DETAIL] := CHANNEL_INFO[CHANNEL].SR_MIN ;
                        UPDATE_SCREEN_SR ;
                     END ;
               PROGRAMMED := FALSE ; { parameter change needs to be programmed }
            END ;
      WRCOL(ROW,COL,3,'        ') ;
      IF CHANNEL_INFO[CHANNEL].SR_MIN > 0.0
         THEN WRCOL(ROW,COL,3,F_TO_A(CHANNEL_INFO[CHANNEL].SR_MIN,2,2))
         ELSE WRCOL(ROW,COL+2,3,'?') ;
   END ;
{-----------------------------------------------------------------------------}

PROCEDURE GET_S_CALIB(VAR S_CALIB : REAL ; ROW , COL : INTEGER) ;
   VAR
      TEMPREAL   : REAL    ;
      DONE       : BOOLEAN ;
   BEGIN
      WRCOL(MESSAGE_ROW,0,3,S_CALIB_MESSAGE) ;
      IF S_CALIB > -9999.0
         THEN WRCOL(ROW,COL,HILITE,F_TO_A(S_CALIB,2,2))
         ELSE WRCOL(ROW,COL+2,HILITE,'?') ;
      CURSOR(ROW,COL) ;
      DONE := FALSE ;
      TEMPREAL := -9999.0 ;
      GETNUM(TEMPREAL,2,2) ;
      IF TEMPREAL > -9998.0
         THEN
            BEGIN
               S_CALIB := TEMPREAL ;
               PROGRAMMED := FALSE ; { parameter change needs to be programmed }
            END ;
      WRCOL(ROW,COL,3,'        ') ;
      IF S_CALIB > -9999.0
         THEN WRCOL(ROW,COL,3,F_TO_A(S_CALIB,2,2))
         ELSE WRCOL(ROW,COL+2,3,'?') ;
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE GET_MVPERV_CALIB(VAR MVPERV_CALIB : REAL ; ROW , COL : INTEGER) ;
   VAR
      TEMPREAL   : REAL    ;
      DONE       : BOOLEAN ;
   BEGIN
      WRCOL(MESSAGE_ROW,0,3,MVPERV_CALIB_MESSAGE) ;
      IF MVPERV_CALIB > -9999.0
         THEN WRCOL(ROW,COL,HILITE,F_TO_A(MVPERV_CALIB,2,2))
         ELSE WRCOL(ROW,COL+2,HILITE,'?') ;
      CURSOR(ROW,COL) ;
      DONE := FALSE ;
      TEMPREAL := -9999.0 ;
      GETNUM(TEMPREAL,2,2) ;
      IF TEMPREAL > -9998.0
         THEN
            BEGIN
               MVPERV_CALIB := TEMPREAL ;
               PROGRAMMED := FALSE ; { parameter change needs to be programmed }
            END ;
      WRCOL(ROW,COL,3,'        ') ;
      IF MVPERV_CALIB > -9999.0
         THEN WRCOL(ROW,COL,3,F_TO_A(MVPERV_CALIB,2,2))
```

```
         ELSE WRCOL(ROW,COL+2,3,'?') ;
   END ;

{--------------------------------------------------------------------------}

PROCEDURE GET_FATIGUE_DETAIL(VAR CHANNEL : CHANNEL_RECORD) ;
   VAR
      ROW , COL   : INTEGER    ;
      DONE        : BOOLEAN    ;
      CAT         : STRING[4]  ;
      TEMP_REAL   : REAL       ;
      OLD_DETAIL  : INTEGER    ;
   BEGIN
      ROW := CHANNEL.ROW ;
      COL := CHANNEL.DETAIL_COL ;
      WRCOL(MESSAGE_ROW,0,3,DETAIL_MESSAGE) ;
      OLD_DETAIL := CHANNEL.DETAIL ;
      IF CHANNEL.DETAIL = 0
         THEN BEGIN
                 WRCOL(ROW,COL,HILITE,'Undefined') ;
              END
         ELSE BEGIN
                 WRCOL(ROW,COL,HILITE,'            ') ;
                 WRCOL(ROW,COL+4,HILITE,I_TO_A(CHANNEL.DETAIL)) ;
                 IF ((CHANNEL.DETAIL > 0) AND (CHANNEL.DETAIL <= 7))
                    THEN
                       BEGIN
                          CHANNEL.SR_MAX := SRMAX[CHANNEL.DETAIL] ;
                          CHANNEL.SR_MIN := SRMIN[CHANNEL.DETAIL] ;
                          WRCOL(ROW,COL+12,3,'        ') ;
                          WRCOL(ROW,COL+12,3,F_TO_A(CHANNEL.SR_MAX,2,2)) ;
                          WRCOL(ROW,COL+20,3,'        ') ;
                          WRCOL(ROW,COL+20,3,F_TO_A(CHANNEL.SR_MIN,2,2)) ;
                       END ;
              END ;
      CURSOR(ROW,COL+4) ;
      TEMP_REAL := -9999.0 ;
      GETNUM(TEMP_REAL,2,0) ;
      IF TEMP_REAL > 0.0
         THEN
            BEGIN
               CHANNEL.DETAIL := ROUND(TEMP_REAL) ;
               PROGRAMMED := FALSE ; { parameter change needs to be programmed }
            END ;

      IF CHANNEL.DETAIL = 0
         THEN BEGIN
                 WRCOL(ROW,COL,3,'Undefined') ;
              END
         ELSE BEGIN
                 WRCOL(ROW,COL,3,'            ') ;
                 WRCOL(ROW,COL+4,3,I_TO_A(CHANNEL.DETAIL)) ;
                 IF (OLD_DETAIL <> CHANNEL.DETAIL)
                    THEN
                       BEGIN
                          CHANNEL.SR_MAX := -1.0 ;
                          CHANNEL.SR_MIN := -1.0 ;
                       END ;
                 IF ((CHANNEL.DETAIL > 0) AND (CHANNEL.DETAIL <= 7))
                    THEN
                       BEGIN
                          CHANNEL.SR_MAX := SRMAX[CHANNEL.DETAIL] ;
                          CHANNEL.SR_MIN := SRMIN[CHANNEL.DETAIL] ;
                          WRCOL(ROW,COL+12,3,'        ') ;
                          WRCOL(ROW,COL+12,3,F_TO_A(CHANNEL.SR_MAX,2,2)) ;
                          WRCOL(ROW,COL+20,3,'        ') ;
                          WRCOL(ROW,COL+20,3,F_TO_A(CHANNEL.SR_MIN,2,2)) ;
                       END ;
              END ;
   END ;

{--------------------------------------------------------------------------}

PROCEDURE GET_TIME(ROW,COL : INTEGER) ;
   VAR
```

```
      TEMP_REAL : REAL ;
      STR : MESSAGE ;
   BEGIN
      WITH REGS DO
         BEGIN
            AX := $2C00 ;
            MSDOS(REGS) ;
            CURRENT_TIME.HOUR    := HI(CX) ;
            CURRENT_TIME.MINUTE  := LO(CX) ;
            CURRENT_TIME.SECOND  := HI(DX) ;
         END ;
      WRCOL(ROW,COL,3,'Current time is ') ;
      WITH CURRENT_TIME DO
         BEGIN
            STR := I_TO_A(HOUR) ;
            IF ORD(STR[0]) = 1
               THEN BEGIN
                       STR[2] := STR[1] ;
                       STR[1] := '0'    ;
                       STR[0] := CHR(2) ;
                    END ;
            WRCOL(ROW,COL+16,3,STR) ;
            WRCOL(ROW,COL+18,3,':') ;
            STR := I_TO_A(MINUTE) ;
            IF ORD(STR[0]) = 1
               THEN BEGIN
                       STR[2] := STR[1] ;
                       STR[1] := '0'    ;
                       STR[0] := CHR(2) ;
                    END ;
            WRCOL(ROW,COL+19,3,STR) ;
         END ;
      TEMP_REAL := -9999.0 ;
      WRCOL(ROW,COL+25,3,'Enter new hour : ') ;
      CURSOR(ROW,COL+42) ;
      GETNUM(TEMP_REAL,2,0) ;
      IF TEMP_REAL > -9999.0 THEN CURRENT_TIME.HOUR := ROUND(TEMP_REAL) ;
      TEMP_REAL := -9999.0 ;
      WRCOL(ROW,COL+50,3,'Enter new minute : ') ;
      CURSOR(ROW,COL+69) ;
      GETNUM(TEMP_REAL,2,0) ;
      IF TEMP_REAL > -9999.0 THEN CURRENT_TIME.MINUTE := ROUND(TEMP_REAL) ;
      CURRENT_TIME.SECOND := 0 ;
      WITH REGS DO
         BEGIN
            AX := $2D00 ;
            CX := CURRENT_TIME.HOUR *  256 + CURRENT_TIME.MINUTE ;
            DX := CURRENT_TIME.SECOND * 256 ;
            MSDOS(REGS) ;
         END ;
      CLS(ROW,COL,ROW,COL+72,0) ;
   END ;

{--------------------------------------------------------------------------}

PROCEDURE INIT_SR_TABLE ;
   BEGIN
      SRMAX[1] := 28.0 ;
      SRMAX[2] := 20.0 ;
      SRMAX[3] := 16.0 ;
      SRMAX[4] := 16.0 ;
      SRMAX[5] := 14.0 ;
      SRMAX[6] :=  9.0 ;
      SRMAX[7] :=  5.0 ;

      SRMIN[1] := 3.0 ;
      SRMIN[2] := 2.0 ;
      SRMIN[3] := 1.0 ;
      SRMIN[4] := 1.0 ;
      SRMIN[5] := 1.0 ;
      SRMIN[6] := 0.5 ;
      SRMIN[7] := 0.5 ;
   END ;

{--------------------------------------------------------------------------}
```

```
PROCEDURE INITIALIZE ;
   VAR
      ROWS , ROW , COL : INTEGER ;
      CHANNEL : INTEGER ;
   BEGIN
     'INIT_SR_TABLE ;

      CURRENT_CHANNEL := 1 ;
      CURRENT_FIELD   := 1 ;

      SCREEN_SET := FALSE ;
      ACQ_SCREEN_SET := FALSE ;
      CURR_PAGE := 0 ;
      SEL_PAGE ;
      CLS(0,0,24,79,0) ;

      CR_LF[1] := CHR(13) ;
      CR_LF[2] := CHR(10) ;
      CR_LF[0] := CHR(0)  ;
      CR       := CHR(13) ;

      TITLE :='           TEXAS STATE DEPARTMENT OF HIGHWAYS AND PUBLIC TRANSPORTATION          ' ;
      WRCOL(0,0,HILITE,TITLE) ;
      WRCOL(10,10,3,'F3  : Low Level Programming') ;
      WRCOL(13,10,3,'F5  : Channel Description') ;
      WRCOL(16,10,3,'F9  : Data Acquisition Menu') ;
      WRCOL(19,10,3,'ESC : Exit to DOS ') ;

      FKEYS                  := 'F1: Load File   F2: Save File   F3: Send File   Del: Erase Channel    ES
      TRANS_MESSAGE          := '               Press G for Strain gage , or T for Strain Transducer
      DETAIL_MESSAGE         := '   Enter 1..7 for category A,B,B ,C,D,E, and E , or 8 thru 99 for user d
      DETAIL_MESSAGE[33]     := CHR(39) ;
      DETAIL_MESSAGE[47]     := CHR(39) ;
      SR_MAX_MESSAGE         := ' Enter desired value for Sr max  (if less than 1 precede decimal point w
      SR_MIN_MESSAGE         := ' Enter desired value for Sr min  (if less than 1 precede decimal point w
      SEND_MESSAGE           := '          PLEASE STAND BY ... TRANSMITTING INTRUCTIONS TO MICROLOGGER.
      READ_MESSAGE           := '                PLEASE STAND BY ... Reading Channel(s).
      DEFINE_MESSAGE         := 'Channel x is incompletely defined. Delete channel (Y or N or ESC to canc
      EMPTY_MESSAGE          := '                  No channels to send!  Press any key ...
      S_CALIB_MESSAGE        := '                            Enter Calibration S value
      MVPERV_CALIB_MESSAGE := '                             Enter Calibration MV/V value
      FULL_DUPLEX            := FALSE ;
      BAUD                   := 300   ;
      PARITY                 := 0     ;
      NSTOP                  := 1     ;
      NDATA                  := 8     ;

      COM_PAR(BAUD,PARITY,NSTOP,NDATA) ;


      FOR COL := 1 TO 80 DO
         BEGIN
            SOLID_LINE[COL] := CHR(196) ;
            TABLE_BOTTOM[COL] := CHR(196) ;

     END ;

  SOLID_LINE[0]   := CHR(80) ;
  TABLE_BOTTOM[0] := CHR(80) ;

  FOR COL := 2 TO 66 DO
     BEGIN
```

```
      TABLE_TOP[COL]      := CHR(196) ;
      TABLE_TOP_2[COL]    := CHR(196) ;
      TABLE_MIDDLE[COL]   := ' ' ;
      TABLE_LINE[COL]     := CHR(196) ;
      TABLE_BOTTOM[COL]   := CHR(196) ;
   END ;

TABLE_TOP[1]       := CHR(218) ;
TABLE_TOP[9]       := CHR(194) ;
TABLE_TOP[24]      := CHR(194) ;
TABLE_TOP[40]      := CHR(194) ;
TABLE_TOP[51]      := CHR(194) ;
TABLE_TOP[59]      := CHR(194) ;
TABLE_TOP[67]      := CHR(191) ;

TABLE_TOP_2[1]     := CHR(195) ;
TABLE_TOP_2[9]     := CHR(197) ;
TABLE_TOP_2[24]    := CHR(197) ;
TABLE_TOP_2[32]    := CHR(194) ;
TABLE_TOP_2[40]    := CHR(197) ;
TABLE_TOP_2[51]    := CHR(197) ;
TABLE_TOP_2[59]    := CHR(197) ;
TABLE_TOP_2[67]    := CHR(180) ;

TABLE_MIDDLE[1]    := CHR(179) ;
TABLE_MIDDLE[9]    := CHR(179) ;
TABLE_MIDDLE[24]   := CHR(179) ;
TABLE_MIDDLE[32]   := CHR(179) ;
TABLE_MIDDLE[40]   := CHR(179) ;
TABLE_MIDDLE[51]   := CHR(179) ;
TABLE_MIDDLE[59]   := CHR(179) ;
TABLE_MIDDLE[67]   := CHR(179) ;

TABLE_LINE[1]      := CHR(195) ;
TABLE_LINE[9]      := CHR(197) ;
TABLE_LINE[24]     := CHR(197) ;
TABLE_LINE[32]     := CHR(197) ;
TABLE_LINE[40]     := CHR(197) ;
TABLE_LINE[51]     := CHR(197) ;
TABLE_LINE[59]     := CHR(197) ;
TABLE_LINE[67]     := CHR(180) ;

TABLE_BOTTOM[7]    := CHR(193) ;
TABLE_BOTTOM[15]   := CHR(193) ;
TABLE_BOTTOM[30]   := CHR(193) ;
TABLE_BOTTOM[38]   := CHR(193) ;
TABLE_BOTTOM[46]   := CHR(193) ;
TABLE_BOTTOM[57]   := CHR(193) ;
TABLE_BOTTOM[65]   := CHR(193) ;
TABLE_BOTTOM[73]   := CHR(193) ;

TABLE_TOP[0]     := CHR(67) ;
TABLE_TOP_2[0]   := CHR(67) ;
TABLE_MIDDLE[0]  := CHR(67) ;
TABLE_LINE[0]    := CHR(67) ;

FOR CHANNEL := 1 TO 8 DO
   BEGIN
      CHANNEL_INFO[CHANNEL].ID                := CHANNEL ;
      CHANNEL_INFO[CHANNEL].TRANSDUCER        := 0         ; { undefined }
      CHANNEL_INFO[CHANNEL].TRANSDUCER_COL    := TABLE_START_COLUMN + 9 ;
      CHANNEL_INFO[CHANNEL].DETAIL            := 0         ; { undefined }
      CHANNEL_INFO[CHANNEL].DETAIL_COL        := TABLE_START_COLUMN + 40 ;
      CHANNEL_INFO[CHANNEL].EXCITATION        := 4000      ; { 4 volts }
      IF ((CHANNEL = 1) OR (CHANNEL = 2))
         THEN CHANNEL_INFO[CHANNEL].EX_CHANNEL := 1   ;
      IF ((CHANNEL = 3) OR (CHANNEL = 4))
         THEN CHANNEL_INFO[CHANNEL].EX_CHANNEL := 2   ;
      IF ((CHANNEL = 5) OR (CHANNEL = 6))
         THEN CHANNEL_INFO[CHANNEL].EX_CHANNEL := 3   ;
      IF ((CHANNEL = 7) OR (CHANNEL = 8))
         THEN CHANNEL_INFO[CHANNEL].EX_CHANNEL := 4   ;
      CHANNEL_INFO[CHANNEL].ROW               := TABLE_START_ROW + CHANNEL*2 + 2 ;
      CHANNEL_INFO[CHANNEL].SR_MAX            := -1.0      ; { undefined }
```

```
         CHANNEL_INFO[CHANNEL].SR_MIN           := -1.0     ; { undefined }
         CHANNEL_INFO[CHANNEL].ZERO             :=  0.0     ;
         CHANNEL_INFO[CHANNEL].CURRENT          :=  0.0     ;
         CHANNEL_INFO[CHANNEL].S_CALIB          := -9999.9  ;
         CHANNEL_INFO[CHANNEL].MVPERV_CALIB     := -9999.9  ;
         CHANNEL_INFO[CHANNEL].MULTIPLIER       := 1.0      ;
         CHANNEL_INFO[CHANNEL].OFFSET           := 0.0      ;
      END ;

      START_TIME_HOUR        := 99    ;
      START_TIME_MINUTE      := 99    ;
      RAIN_INTERVAL          := 9999  ;
      MEANS_BINS             := 2     ;
      AMPLITUDE_BINS         := 50    ;
      LOW_LIMIT              := -50   ;
      HIGH_LIMIT             := 50    ;
      PEAK_VALLEY_DISTANCE   := 0.2   ; { 0.2 mv = 1.62 ksi }
      RAIN_INTERVAL          := 1440  ;
      PROGRAMMED             := FALSE ;

   END ;

{-----------------------------------------------------------------------------}

PROCEDURE ELIMINATE(CHANNEL : INTEGER) ;
   BEGIN
      CHANNEL_INFO[CHANNEL].TRANSDUCER      := 0        ; { undefined }
      CHANNEL_INFO[CHANNEL].DETAIL          := 0        ; { undefined }
      CHANNEL_INFO[CHANNEL].SR_MAX          := -1.0     ; { undefined }
      CHANNEL_INFO[CHANNEL].SR_MIN          := -1.0     ; { undefined }
      CHANNEL_INFO[CHANNEL].S_CALIB         := -9999.9  ; { undefined }
      CHANNEL_INFO[CHANNEL].MVPERV_CALIB    := -9999.9  ; { undefined }
      SHOW_VALUES ;
   END ;

{-----------------------------------------------------------------------------}

FUNCTION ALL_DEFINED : BOOLEAN ;
   VAR
      CHANNEL : INTEGER ;
      DEFINED : BOOLEAN ;
   BEGIN
      DEFINED := TRUE ;
      CHANNEL := 1    ;
      WHILE (CHANNEL <= 8) AND DEFINED DO
         BEGIN
            IF (((CHANNEL_INFO[CHANNEL].TRANSDUCER   =  0)       AND
                 (CHANNEL_INFO[CHANNEL].DETAIL       =  0)       AND
                 (CHANNEL_INFO[CHANNEL].SR_MAX       <  0.0)     AND
                 (CHANNEL_INFO[CHANNEL].SR_MIN       <  0.0)     AND
                 (CHANNEL_INFO[CHANNEL].S_CALIB      < -9999.0)  AND
                 (CHANNEL_INFO[CHANNEL].MVPERV_CALIB < -9999.0))
               OR
               ((CHANNEL_INFO[CHANNEL].TRANSDUCER   <> 0)       AND
                (CHANNEL_INFO[CHANNEL].DETAIL       <> 0)       AND
                (CHANNEL_INFO[CHANNEL].SR_MAX       >  0.0)     AND
                (CHANNEL_INFO[CHANNEL].SR_MIN       >  0.0)     AND
                (CHANNEL_INFO[CHANNEL].S_CALIB      > -9999.0)  AND
                (CHANNEL_INFO[CHANNEL].MVPERV_CALIB > -9999.0)))
             THEN DEFINED := TRUE
             ELSE
                BEGIN
                  DEFINED := FALSE ;
                  IF (CHANNEL_INFO[CHANNEL].TRANSDUCER = 0)
                     THEN BEGIN
                             CURRENT_CHANNEL := CHANNEL ;
                             CURRENT_FIELD   := 1 ;
                          END
                     ELSE IF (CHANNEL_INFO[CHANNEL].DETAIL = 0)
                             THEN BEGIN
                                     CURRENT_CHANNEL := CHANNEL ;
                                     CURRENT_FIELD   := 4 ;
                                  END
                             ELSE IF (CHANNEL_INFO[CHANNEL].SR_MAX < 0.0)
                                     THEN BEGIN
```

```
                                        CURRENT_CHANNEL := CHANNEL ;
                                        CURRENT_FIELD   := 5 ;
                                     END
                              ELSE IF (CHANNEL_INFO[CHANNEL].SR_MIN < 0.0)
                                    THEN BEGIN
                                              CURRENT_CHANNEL := CHANNEL ;
                                              CURRENT_FIELD   := 6 ;
                                         END
                                    ELSE IF (CHANNEL_INFO[CHANNEL].S_CALIB < -9999.0)
                                          THEN BEGIN
                                                    CURRENT_CHANNEL := CHANNEL ;
                                                    CURRENT_FIELD   := 2 ;
                                               END
                                          ELSE IF (CHANNEL_INFO[CHANNEL].MVPERV_CALIB < -999
                                                THEN BEGIN
                                                          CURRENT_CHANNEL := CHANNEL ;
                                                          CURRENT_FIELD   := 3 ;
                                                     END
              END ; { else }
          CHANNEL := CHANNEL + 1 ;
        END ; { while }
      ALL_DEFINED := DEFINED ;
    END ;

{----------------------------------------------------------------------------}

{----------------------------------------------------------------------------}

PROCEDURE JUMP_TO(LOCATION : INTEGER) ;
    VAR
      I : INTEGER ;
      RESPONSE1 : MESSAGE ;
    BEGIN
      INIT_CAMPBELL ;
      WRS_COM(I_TO_A(LOCATION)) ;
      PUTC_COM('G') ;
      PUTC_COM(CR) ;
      READLN(AUX,RESPONSE1) ;
      READLN(AUX,RESPONSE1) ;
    END ;

{----------------------------------------------------------------------------}

PROCEDURE ERASE_PROGRAM ;
    VAR
      I : INTEGER ;
    BEGIN
      GO_REMOTE ;
      WRS_COM('AAAA978A') ;
      FOR I := 1 TO 200 DO
        DELAY(100) ;
      GO_REMOTE ;
    END ;

{----------------------------------------------------------------------------}

PROCEDURE SEND_PROGRAM ;
    VAR
      CHANNEL        : INTEGER ;
      INPUT_LOCATION : INTEGER ;
      INITIAL_LOC    : INTEGER ;
    BEGIN
      ERASE_PROGRAM ;
      GO_REMOTE ;
      WRS_COM('AA2434A') ;
      WRS_COM('*1A') ;
      WRS_COM('D0125A') ;
```

```
{ write # of channels to input location 1 }
WRS_COM('30A') ;
WRS_COM(I_TO_A(CHANNELS)) ;
WRS_COM('A') ;
WRS_COM('1A') ;
    ;
INPUT_LOCATION := 2 ;
CHANNELS := 0 ;
FOR CHANNEL := 1 TO 8 DO ACTIVE_CHANNELS[CHANNEL] := -1 ; { inactive }
FOR CHANNEL := 1 TO 8 DO
    BEGIN
        IF DEFINED(CHANNEL)
            THEN
                BEGIN
                    CHANNELS := CHANNELS + 1 ;
                    ACTIVE_CHANNELS[CHANNELS] := CHANNEL ;
                    WRS_COM('6A') ;        { full bridge measurment }
                    WRS_COM('1A') ;        { param. # 1 = # of repetitions = 1 for 1 channel }
                    WRS_COM(I_TO_A(CHANNEL_INFO[CHANNEL].RANGE))        ; { param. # 2 = full scale range
                    WRS_COM('A') ;
                    WRS_COM(I_TO_A(CHANNEL)) ; { param. # 3 = input channel # }
                    WRS_COM('A') ;
                    WRS_COM(I_TO_A(CHANNEL_INFO[CHANNEL].EX_CHANNEL)) ; { param. # 4 = excitation channel
                    WRS_COM('A') ;
                    WRS_COM(I_TO_A(CHANNEL_INFO[CHANNEL].EXCITATION)) ; { param. # 5 = excitation level }
                    WRS_COM('A') ;
                    WRS_COM(I_TO_A(INPUT_LOCATION)) ; { param. # 6 = input storage location }
                    WRS_COM('A') ;
                    WRS_COM(F_TO_A(CHANNEL_INFO[CHANNEL].MULTIPLIER,4,4)) ; { param. # 7 = multiplier }
                    WRS_COM('A') ;
                    WRS_COM(F_TO_A(CHANNEL_INFO[CHANNEL].OFFSET,4,4)) ;
                    IF CHANNEL_INFO[CHANNEL].OFFSET < 0.0 THEN WRS_COM('C') ;
                    WRS_COM('A') ;
                    INPUT_LOCATION := INPUT_LOCATION + 1 ;
                END ;
    END ;

WRS_COM('91A')  ;  { if flag set control instruction - used to  }
                   { start and stop data processing subroutines }
WRS_COM('11A')  ;  { parameter 1 : do if flag 1 is set          }
WRS_COM('1A')   ;  { parameter 2 : call subroutine 1            }

WRS_COM('91A')  ;  { if flag set control instruction - used to  }
                   { start and stop data processing subroutines }
WRS_COM('12A')  ;  { parameter 1 : do if flag 2 is set          }
WRS_COM('2A')   ;  { parameter 2 : call subroutine 2            }

WRS_COM('30A') ;   {******** flashing light ********}
WRS_COM('3000A') ;
WRS_COM(I_TO_A(INPUT_LOCATION)) ;
WRS_COM('A') ;
WRS_COM('21A') ;
WRS_COM('1A') ;
WRS_COM(I_TO_A(INPUT_LOCATION)) ;
WRS_COM('A') ;
WRS_COM('30A') ;
WRS_COM('0A') ;
WRS_COM(I_TO_A(INPUT_LOCATION+1)) ;
WRS_COM('A') ;
WRS_COM('21A') ;
WRS_COM('1A') ;
WRS_COM(I_TO_A(INPUT_LOCATION+1)) ;
WRS_COM('A') ;

WRS_COM('*3A') ;   {******* ENTER_MODE_3 ******                 }

WRS_COM('85A') ;   { label subroutine                           }
WRS_COM('1A')  ;   { subroutine 1                               }
```

```
WRS_COM('92A') ;    { if time output instruction - sets time      }
                    { interval for output of data.                }
WRS_COM('0A')  ;    { parameter 1 : start at time = 0             }
WRS_COM(I_TO_A(RAIN_INTERVAL)) ;
                    { parameter 2 : rainflow time interval        }
WRS_COM('A')   ;    { advance                                     }
WRS_COM('10A') ;    { parameter 3 : set output flag for end of    }
                    { time interval                               }

WRS_COM('77A') ;    { output time tagging instruction             }
WRS_COM('110A') ;   { parameter 1 : day , hour , minute format    }

INITIAL_LOC := 2 ;
FOR CHANNEL := 1 TO 8 DO
   BEGIN
      IF DEFINED(CHANNEL)
         THEN
            BEGIN
               WRS_COM('81A') ;    { rainflow intermediate processing inst.      }
               WRS_COM('1A')  ;    { parameter 1 : # of transducers              }
               WRS_COM(I_TO_A(INITIAL_LOC)) ;
                                   { parameter 2 : initial location of input     }
               WRS_COM('A') ;      { advance }
               WRS_COM('1A') ;     { parameter 3 : on-line cont. processing      }
               WRS_COM(I_TO_A(MEANS_BINS)) ;
                                   { parameter 4 : # of means beans              }
               WRS_COM('A') ;      { advance }
               WRS_COM(I_TO_A(AMPLITUDE_BINS)) ;
                                   { parameter 5 : # of amplitude beans          }
               WRS_COM('A') ;      { advance }
               WRS_COM(I_TO_A(LOW_LIMIT)) ;
                                   { parameter 6 : low limit of input data mv    }
               IF LOW_LIMIT < 0 THEN WRS_COM('C') ;
               WRS_COM('A') ;      { advance }
               WRS_COM(I_TO_A(HIGH_LIMIT)) ;
                                   { parameter 7 : high limit of input data mv   }
               WRS_COM('A') ;      { advance }
               PEAK_VALLEY_DISTANCE := 95.0 * (CHANNEL_INFO[CHANNEL].SR_MIN /
                                     CHANNEL_INFO[CHANNEL].SR_MAX) ;
               WRS_COM(F_TO_A(PEAK_VALLEY_DISTANCE,2,2)) ;
                                   { parameter 8 : minimum distance between      }
                                   { peak and valley                             }
               WRS_COM('A') ;      { advance }
               WRS_COM('11A') ;    { parameter 9 : open form , counts recorded   }
               WRS_COM('0A') ;     { parameter 10: send output to final storage  }
               INITIAL_LOC := INITIAL_LOC + 1 ;
            END ; { if defined }
   END ; { for }
WRS_COM('95A') ;    { end instruction                             }

WRS_COM('85A') ;    { label subroutine instruction                }
WRS_COM('2A')  ;    { parameter 1 : subroutine # 2                }

WRS_COM('86A') ;    { do command instruction                      }
WRS_COM('10A') ;    { parameter 1 : set output flag               }

WRS_COM('70A') ;    { sample and output data in input storage     }
WRS_COM(I_TO_A(CHANNELS)) ;
WRS_COM('A') ;      { advance }
```

```
                         { parameter 1 : # of transducers              }
    WRS_COM('2A') ;      { location of initial input storage location }

    WRS_COM('95A') ;     { end instruction                             }

    WRS_COM('*0')  ;     { compile program and execute                 }
    CLS(MESSAGE_ROW,0,MESSAGE_ROW,79,0) ;

    PROGRAMMED := TRUE ;

  END ;
{--------------------------------------------------------------------------}

PROCEDURE WAIT(LENGTH : INTEGER) ;
  VAR
    START , CURRENT : TIME     ;
    DONE , AVAIL    : BOOLEAN  ;
    CH              : CHAR     ;
    LAPS            : INTEGER  ;
    RESPONSE1       : MESSAGE  ;
  BEGIN
    DONE := FALSE ;
    WITH REGS DO
      BEGIN
        AX := $2C00 ;
        MSDOS(REGS) ;
        START.MINUTE := LO(CX) ;
        START.SECOND := HI(DX) ;
      END ;
    WHILE NOT DONE DO
      BEGIN
        WITH REGS DO
          BEGIN
            AX := $2C00 ;
            MSDOS(REGS) ;
            CURRENT.MINUTE := LO(CX) ;
            CURRENT.SECOND := HI(DX) ;
          END ;
        LAPS := (CURRENT.MINUTE*60 + CURRENT.SECOND) - (START.MINUTE*60 + START.SECOND) ;
        IF  LAPS >= LENGTH
          THEN DONE := TRUE
          ELSE BEGIN
                  PUTC_COM(CR) ;
                  READLN(AUX,RESPONSE1) ;
               END ;
      END ;
  END ;
{--------------------------------------------------------------------------}

PROCEDURE LOAD_STK(ROW , COL : INTEGER) ;
  VAR
    DONE : BOOLEAN ;
    TEMP_NAME : STRING[50] ;
    CHANNEL : INTEGER ;
    I , J , SCAN : INTEGER ;
    CH : CHAR    ;
  BEGIN
            ASSIGN(STK_FILE,'C:TR2.STK') ;
            {$I-}
            RESET(STK_FILE)
            {$I+} ;
            DONE := (IOresult = 0) ;
            IF NOT DONE
              THEN WRCOL(ROW,COL+25,4, 'File Not Found ')
              ELSE
                BEGIN
                  WRCOL(ROW,COL+25,4,'Reading Data.. ') ;
                  READLN(STK_FILE,NSCAN) ;
                  READLN(STK_FILE) ;
                  FOR SCAN := 1 TO NSCAN DO
                    BEGIN
                      FOR I := 1 TO CR21X_CHANNELS DO
```

```
                                       BEGIN
                                          J := ACTIVE_CHANNELS[I] ;
                                          READ(STK_FILE,PLOT_DATA[J,SCAN]) ;
                                       END ;
                                    READLN(STK_FILE) ;
                                 END ;
                           END ; { else }
              ;       CLOSE(STK_FILE)
   END ;

{------------------------------------------------------------------------}

PROCEDURE PLOT_CHANNEL(N_CHAN , NPOINTS : INTEGER ; PCHAN : PLOT_CHANNELS) ;
   VAR
      I , J                : INTEGER ;
      XMAX , XMIN          : INTEGER ;
      XLEN , YLEN          : REAL    ;
      XSCALE , YSCALE      : REAL    ;
      X1 , X2 , Y1 , Y2    : INTEGER ;
      CHAN                 : INTEGER ;
      YMAX , YMIN          : REAL    ;
      ROW , COL            : INTEGER ;
   BEGIN
      XMAX := NPOINTS ;
      XMIN := 1 ;
      XLEN := XMAX ;
      YMAX := -9999.9 ;
      YMIN :=  9999.9 ;
      FOR I := 1 TO N_CHAN DO
         BEGIN
            CHAN := PCHAN[I] ;
            FOR J := 1 TO NPOINTS DO
               BEGIN
                  IF PLOT_DATA[CHAN,J] > YMAX THEN YMAX := PLOT_DATA[CHAN,J] ;
                  IF PLOT_DATA[CHAN,J] < YMIN THEN YMIN := PLOT_DATA[CHAN,J] ;
               END ;
         END ;
      YLEN := ABS(YMAX - YMIN) ;
      IF YLEN = 0.0 THEN YLEN := ABS(YMAX) ;
      XSCALE := 639.0 / XLEN ;
      YSCALE := 199.0 / YLEN ;
      IF YMAX <> YMIN THEN WRCOL(24,0,3,F_TO_A(YMIN,5,4)) ;
      WRCOL(0,0,3,F_TO_A(YMAX,5,4)) ;
      FOR I := 1 TO N_CHAN DO
         BEGIN
            CHAN := PCHAN[I] ;
            FOR J := 1 TO NPOINTS - 1 DO
               BEGIN
                  X1 := J * ROUND(XSCALE) ;
                  Y1 := ROUND((YMAX - PLOT_DATA[CHAN,J]) * YSCALE) ;
                  X2 := (J + 1) * ROUND(XSCALE) ;
                  Y2 := ROUND((YMAX - PLOT_DATA[CHAN,J+1]) * YSCALE) ;
                  DRAW(X1,Y1,X2,Y2,3) ;
                  DELAY(25) ;
               END ;
         END ;
   END ;

{------------------------------------------------------------------------}

PROCEDURE TRANSLATE(LINE : MESSAGE ; VAR NVAL : INTEGER ; VAR VALUES : VAL_ARRAY) ;
   VAR
      I , J : INTEGER ;
      PTR   : INTEGER ;
      INDEX : INTEGER ;
      ASCII : STRING[6] ;
   BEGIN
      PTR  := 1 ;
      NVAL := 0 ;
      WHILE PTR < ORD(LINE[0]) DO
         BEGIN
            NVAL := NVAL + 1 ;
            VALUES[NVAL].INDEX := ((ORD(LINE[PTR]) - 48) * 10) + (ORD(LINE[PTR+1]) - 48) ;
```

```
          FOR J := 1 TO 6 DO ASCII[J] := LINE[PTR + 1 + J] ;
          ASCII[0] := CHR(6) ;
          VALUES[NVAL].VAL := A_TO_F(ASCII)  ;
          PTR := PTR + 10 ;
       END ;
   END ;

{-----------------------------------------------------------------------}

PROCEDURE GET_POINTERS      ;
   VAR
      RESPONSE1 , RESPONSE2 : MESSAGE ;
      I , J : INTEGER ;
      OK    : BOOLEAN ;
   BEGIN
      OK := FALSE ;
      WHILE NOT OK DO
         BEGIN
            PUTC_COM('A')          ;
            PUTC_COM(CHR(13))      ;
            READLN(AUX,RESPONSE1)  ;
            READLN(AUX,RESPONSE2)  ;
            IF RESPONSE2[2] = 'R'
               THEN OK := TRUE
               ELSE WAIT(1)        ;
         END ;
      I := 1 ;
      WHILE (NOT (RESPONSE2[I] IN ['1','2','3','4','5','6','7','8','9','0'])) AND
            (I < 80)
      DO I := I + 1 ;
      J := 1 ;
      WHILE RESPONSE2[I] <> ' ' DO
         BEGIN
            RESPONSE1[J] := RESPONSE2[I] ;
            J := J + 1 ;
            I := I + 1 ;
         END ;
      RESPONSE1[0] := CHR(J-1) ;
      DSP := ROUND(A_TO_F(RESPONSE1)) ;
      WHILE RESPONSE2[I] <> 'L' DO I := I + 1 ;
      I := I + 2 ;
      J := 1 ;
      WHILE RESPONSE2[I] <> ' ' DO
         BEGIN
            RESPONSE1[J] := RESPONSE2[I] ;
            J := J + 1 ;
            I := I + 1 ;
         END ;
      RESPONSE1[0] := CHR(J-1) ;
      MPTR := ROUND(A_TO_F(RESPONSE1)) ;
      IF MPTR > DSP
         THEN NSCAN := (14588 + DSP - MPTR) DIV (CR21X_CHANNELS + 1)
         ELSE NSCAN := (DSP - MPTR) DIV (CR21X_CHANNELS + 1) ;
   END ;

{-----------------------------------------------------------------------}

PROCEDURE TOGGLE_RECORD(MODE : INTEGER) ;
   VAR
      RESPONSE1 , RESPONSE2 : MESSAGE ;
      CH : CHAR ;
   BEGIN
      GO_REMOTE              ;
      READLN(AUX,RESPONSE1)  ;
      PUTC_COM('6')          ;
      PUTC_COM('A')          ;
      READLN(AUX,RESPONSE1)  ;
      PUTC_COM('D')          ;
      READLN(AUX,RESPONSE1)  ;
      IF MODE = 1
         THEN PUTC_COM('1')
         ELSE PUTC_COM('2')  ;
      READLN(AUX,RESPONSE1)  ;
      PUTC_COM('*')          ;
```

```
      READLN(AUX,RESPONSE1) ;
      PUTC_COM('0')         ;
      READLN(AUX,RESPONSE1) ;
      READLN(AUX,RESPONSE2) ;
{      WAIT(5) ;}
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE DUMP_SINGLE(VAR DATA_POINTS : VAL_ARRAY) ;

   VAR
      RESPONSE1 , RESPONSE2 , RESPONSE3 : MESSAGE ;
      I , J , K : INTEGER ;
   BEGIN
      PUTC_COM('D')          ;    { get a dump of readings }
      PUTC_COM(CHR(13))      ;
      READLN(AUX,RESPONSE1)  ;

      READLN(AUX,RESPONSE1) ;
      READLN(AUX,RESPONSE2) ;
      IF CR21X_CHANNELS = 8 THEN READLN(AUX,RESPONSE3) ;

      FOR I := 1 TO 70 DO RESPONSE3[I] := RESPONSE1[I+10] ;
      RESPONSE3[0] := CHR(CR21X_CHANNELS*10) ;
      IF CR21X_CHANNELS = 8
         THEN
            BEGIN
               FOR I := 71 TO 80 DO RESPONSE3[I] := RESPONSE2[I-70] ;
               RESPONSE3[0] := CHR(80) ;
            END ;

      TRANSLATE(RESPONSE3,K,DATA_POINTS) ;

   END ;

{-----------------------------------------------------------------------------}

PROCEDURE ACQ_SCREEN ;
   BEGIN
      CLS(0,0,24,79,0) ;
      TITLE :='          TEXAS STATE DEPARTMENT OF HIGHWAYS AND PUBLIC TRANSPORTATION          ' ;
      WRCOL(0,0,HILITE,TITLE) ;
      WRCOL( 4,4,3,'F1  = Check Channels')       ;
      WRCOL( 6,4,3,'F2  = Take Single reading')  ;
      WRCOL( 8,4,3,'F3  = Take zero readings')   ;
      WRCOL(10,4,3,'F4  = Zero the data logger') ;
      WRCOL(12,4,3,'F5  = Capture Truck')        ;
      WRCOL(13,4,3,'F6  = Retrieve Truck data')  ;
      WRCOL(14,4,3,'F7  = Plot Truck data')      ;
      WRCOL(15,4,3,'F8  = Save Truck data')      ;
      WRCOL(17,4,3,'F9  = Start Rainflow routine') ;
      WRCOL(18,4,3,'F10 = Retrieve Rainflow Data') ;
      WRCOL(20,4,3,'ESC = Exit to main menu')    ;
      WRCOL(ACTIVITY_MESS_ROW-1,0,3,SOLID_LINE)  ;
      ACQ_SCREEN_SET := TRUE ;
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE RETRIEVE_SINGLE_TRUCK ;
   VAR
      CHANNEL         : INTEGER   ;
      DATA_POINTS     : VAL_ARRAY ;
      COUNTER         : INTEGER   ;
      SCAN , I , J    : INTEGER   ;
   BEGIN
      WRCOL(ACTIVITY_MESS_ROW,5,3,'Please wait ... Retrieving truck data') ;
      COUNTER := NSCAN ;
      JUMP_TO(MPTR) ;
      IF NSCAN > MAX_PLOT_POINTS         {************}
         THEN NSCAN := MAX_PLOT_POINTS ;
{**}  ACQ_SCREEN ;
{**}  WRCOL(ACTIVITY_MESS_ROW,5,3,'Please wait ... Retrieving truck data') ;
```

```
    FOR SCAN := 1 TO NSCAN DO
       BEGIN
          WRCOL(ACTIVITY_MESS_ROW,60,3,'(reading  scan     )') ;
          WRCOL(ACTIVITY_MESS_ROW,75,3,'    ') ;
          WRCOL(ACTIVITY_MESS_ROW,75,3,I_TO_A(COUNTER)) ;
          COUNTER := COUNTER - 1 ;
       ; DUMP_SINGLE(DATA_POINTS) ;
          FOR I := 1 TO CR21X_CHANNELS DO
             BEGIN
                J := ACTIVE_CHANNELS[I] ;
                PLOT_DATA[J,SCAN] := DATA_POINTS[I].VAL ;
             END ;
       END ;
    CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
 END ; { retrieve single truck }

{----------------------------------------------------------------------------}

PROCEDURE SAVE_SINGLE_TRUCK ;
 VAR
    DONE , DN , FOUND   : BOOLEAN    ;
    TEMP_NAME           : STRING[50] ;
    I , J , K , SCAN    : INTEGER    ;
    CH                  : CHAR       ;
 PROCEDURE RECORD_INFO ;
    VAR
       CHANNEL          : INTEGER    ;
       DATA_POINTS      : VAL_ARRAY  ;
       COUNTER          : INTEGER    ;
    BEGIN
       WRCOL(ACTIVITY_MESS_ROW,FILE_COL+25,3,'Please wait ... Saving file to disk. ') ;
       REWRITE(TRUCK_DATA) ;
       WRITELN(TRUCK_DATA,NSCAN:5) ;
       FOR I := 1 TO CR21X_CHANNELS DO
          BEGIN
             J := ACTIVE_CHANNELS[I] ;
             WRITE(TRUCK_DATA,J:1,' , ') ;
          END ;
       WRITELN(TRUCK_DATA) ;
       COUNTER := NSCAN ;
       FOR SCAN := 1 TO NSCAN DO
          BEGIN
             WRCOL(ACTIVITY_MESS_ROW,FILE_COL+62,3,'    ') ;
             WRCOL(ACTIVITY_MESS_ROW,FILE_COL+62,3,I_TO_A(COUNTER)) ;
             COUNTER := COUNTER - 1 ;
             FOR I := 1 TO CR21X_CHANNELS DO
                BEGIN
                   J := ACTIVE_CHANNELS[I] ;
                   WRITE(TRUCK_DATA,PLOT_DATA[J,SCAN]:10:5,' , ') ;
                END ;
             WRITELN(TRUCK_DATA) ;
          END ;
       CLOSE(TRUCK_DATA) ;
       CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
    END ; { record }

 BEGIN
    DONE := FALSE ;
    REPEAT
       IF (FNAME[0] <> CHR(0))
          THEN
             BEGIN
                FOR I := 1 TO ORD(FNAME[0])-2 DO TEMP_NAME[I] := FNAME[I+2] ;
                TEMP_NAME[0] := CHR(ORD(FNAME[0])-2)
             END ;
       CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
       WRCOL(ACTIVITY_MESS_ROW,FILE_COL,3,'Save File : ') ;
       CURSOR(ACTIVITY_MESS_ROW,FILE_COL+12) ;
       FNAME[0] := CHR(0) ;
       GETNAM ;
       IF (FNAME[0] <> CHR(0)) { file specified }
          THEN
             BEGIN
                I := 1 ;
```

```
                WHILE FNAME[I] <> '.' DO I := I + 1 ;
                FNAME[I+1] := 'S' ;
                FNAME[I+2] := 'T' ;
                FNAME[I+3] := 'K' ;
                FOR I := 1 TO ORD(FNAME[0])-2 DO TEMP_NAME[I] := FNAME[I+2] ;
                TEMP_NAME[0] := CHR(ORD(FNAME[0])-2) ;
                WRCOL(ACTIVITY_MESS_ROW,FILE_COL+12,3,'              ') ;
                WRCOL(ACTIVITY_MESS_ROW,FILE_COL+12,3,TEMP_NAME) ;
                ASSIGN(TRUCK_DATA,FNAME) ;
                {$I-}
                RESET(TRUCK_DATA) ;
                CLOSE(TRUCK_DATA) ;
                {$I+} ;
                FOUND := (IOresult = 0) ;
                IF FOUND
                  THEN  { duplicate file exists }
                    BEGIN
                      WRCOL(ACTIVITY_MESS_ROW,FILE_COL+25,4,'File exists ... Overwrite (Y or N) ? ')
                      DN := FALSE ;
                      WHILE NOT DN DO
                        BEGIN
                          GETC(KEY) ;
                          IF CHR(KEY) IN ['Y','y','N','n'] THEN DN := TRUE ;
                        END ;
                      IF CHR(KEY) IN ['Y','y']
                        THEN { overwrite file }
                          BEGIN
                            RECORD_INFO ;
                            DONE := TRUE ;
                          END ;
                    END { then }
                  ELSE
                    BEGIN
                      RECORD_INFO ;
                      DONE := TRUE ;
                    END ;
            END
          ELSE DONE := TRUE ;
      UNTIL DONE ;
    END ;

{-----------------------------------------------------------------------}

PROCEDURE MOVE_BACK ;
    VAR
      RESPONSE  : MESSAGE ;
      I , J , K : INTEGER ;
      CH : CHAR ;
    BEGIN
      INIT_CAMPBELL  ;
      PUTC_COM('B')   ;
      PUTC_COM(CHR(13)) ;
      READLN(AUX,RESPONSE) ;
      READLN(AUX,RESPONSE) ;
      READLN(AUX,RESPONSE) ;
    END ;

{-----------------------------------------------------------------------}

PROCEDURE SET_CR21X_TIME(CR21X_TIME : TIME) ;
    VAR
      RESPONSE  : MESSAGE ;
      I , J , K : INTEGER ;
      CH : CHAR ;
    BEGIN
      INIT_CAMPBELL  ;
      RESPONSE := I_TO_A(CR21X_TIME.HOUR) ;
      IF CR21X_TIME.HOUR < 10
        THEN BEGIN
                RESPONSE[2] := RESPONSE[1] ;
                RESPONSE[1] := '0' ;
                RESPONSE[0] := CHR(2) ;
             END ;
```

```
      WRS_COM(RESPONSE) ;
      PUTC_COM(':') ;
      RESPONSE := I_TO_A(CR21X_TIME.MINUTE) ;
      IF CR21X_TIME.MINUTE < 10
         THEN BEGIN
                 RESPONSE[2] := RESPONSE[1] ;
                 RESPONSE[1] := '0' ;
                 RESPONSE[0] := CHR(2) ;
              END ;
      WRS_COM(RESPONSE) ;
      PUTC_COM(':') ;
      RESPONSE := I_TO_A(CR21X_TIME.SECOND) ;
      IF CR21X_TIME.MINUTE < 10
         THEN BEGIN
                 RESPONSE[2] := RESPONSE[1] ;
                 RESPONSE[1] := '0' ;
                 RESPONSE[0] := CHR(2) ;
              END ;
      WRS_COM(RESPONSE) ;
      PUTC_COM('C') ;
      PUTC_COM(CHR(13)) ;
      READLN(AUX,RESPONSE) ;
      READLN(AUX,RESPONSE) ;
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE GET_CR21X_TIME ;
   VAR
      RESPONSE  : MESSAGE ;
      I , J , K : INTEGER ;
      CH : CHAR ;
   BEGIN
      INIT_CAMPBELL  ;
      PUTC_COM('C')   ;
      PUTC_COM(CHR(13)) ;
      READLN(AUX,RESPONSE) ;
      READLN(AUX,RESPONSE) ;
      READLN(AUX,RESPONSE) ;
      DAYS_ELAPSED := ((ORD(RESPONSE[3]) - 48) * 1000) + ((ORD(RESPONSE[4]) - 48) *  100) +
                      ((ORD(RESPONSE[5]) - 48) *   10) + ((ORD(RESPONSE[6]) - 48) *    1) ;
      CR21X_TIME.HOUR   := ((ORD(RESPONSE[10]) - 48) * 10) + (ORD(RESPONSE[11]) - 48)     ;
      CR21X_TIME.MINUTE := ((ORD(RESPONSE[13]) - 48) * 10) + (ORD(RESPONSE[14]) - 48)     ;
      CR21X_TIME.SECOND := ((ORD(RESPONSE[16]) - 48) * 10) + (ORD(RESPONSE[17]) - 48)     ;
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE GET_NUMBER_OF_CHANNELS ;
   VAR
      RESPONSE  : MESSAGE ;
      I , J , K : INTEGER ;
      CH : CHAR ;
      ASCII : STRING[7] ;
   BEGIN
      GO_REMOTE  ;
      WRS_COM('6AA')   ;
      READLN(AUX,RESPONSE) ;
      READLN(AUX,RESPONSE) ;
      FOR I := 1 TO 7 DO ASCII[I] := RESPONSE[I+3] ;
      ASCII[0] := CHR(7) ;
      CR21X_CHANNELS := ROUND(A_TO_F(ASCII)) ;
      WRS_COM('*0')   ;
      READLN(AUX,RESPONSE) ;
      READLN(AUX,RESPONSE) ;
      READLN(AUX,RESPONSE) ;
   END ;

{-----------------------------------------------------------------------------}
```

```
PROCEDURE SAVE_RAINFLOW_DATA ;
   VAR
      DONE , DN , FOUND : BOOLEAN     ;
      TEMP_NAME         : STRING[50]  ;
      I                 : INTEGER     ;
      MINUTES           : INTEGER     ;
      CH                : CHAR        ;

   PROCEDURE RETRIEVE ;
      VAR
         RESPONSE                 : MESSAGE   ;
         DATA_POINTS              : VAL_ARRAY ;
         I , J , K , INTERVALS    : INTEGER   ;
         CH                       : CHAR      ;
      BEGIN
         REWRITE(DATA_FILE) ;
         WRITELN(DATA_FILE,MEANS_BINS:2,'   ',AMPLITUDE_BINS:2) ;
         WRITELN(DATA_FILE,INTERVALS_ELAPSED:2) ;
         WRITELN(DATA_FILE,RAIN_INTERVAL:4) ;
         WRITELN(DATA_FILE,LOW_LIMIT:4,'   ',HIGH_LIMIT:4) ;
         WRITELN(DATA_FILE,CR21X_CHANNELS:1) ;
         FOR I := 1 TO CR21X_CHANNELS DO
            BEGIN
               J := ACTIVE_CHANNELS[I] ;
               WRITELN(DATA_FILE,CHANNEL_INFO[J].ID:2,'   ',CHANNEL_INFO[J].SR_MAX:5:5,'   ',CHANNEL_INFO[
            END ;
{**}     ACQ_SCREEN ;
         CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
         WRCOL(ACTIVITY_MESS_ROW,0,3,'Positioning memory pointer ') ;
         FOR INTERVALS := 1 TO INTERVALS_ELAPSED DO MOVE_BACK ;
{**}     ACQ_SCREEN ;
         CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
         FOR INTERVALS := 1 TO INTERVALS_ELAPSED DO
            BEGIN
               WRCOL(ACTIVITY_MESS_ROW,0,3,'Intervals elapsed = ') ;
               WRCOL(ACTIVITY_MESS_ROW,20,3,I_TO_A(INTERVALS_ELAPSED)) ;
               WRCOL(ACTIVITY_MESS_ROW,28,3,'Retrieving data collected on interval ') ;
               WRCOL(ACTIVITY_MESS_ROW,66,3,'    ') ;
               WRCOL(ACTIVITY_MESS_ROW,66,3,I_TO_A(INTERVALS)) ;
               CURSOR(ACTIVITY_MESS_ROW,75) ;
               PUTC_COM('D')    ;
               PUTC_COM(CHR(13)) ;
               READLN(AUX,RESPONSE) ;
               FOR I := 1 TO (((CR21X_CHANNELS * MEANS_BINS * AMPLITUDE_BINS) DIV 8) + 1) DO
                  BEGIN
                     READLN(AUX,RAIN_ARRAY[I]) ;
                  END ;
               READLN(AUX,RESPONSE) ;
               WRCOL(ACTIVITY_MESS_ROW,28,3,'     Saving data collected on interval ') ;
               CURSOR(ACTIVITY_MESS_ROW,75) ;
               TRANSLATE(RAIN_ARRAY[1],K,DATA_POINTS) ;
                  FOR J := 4 TO 8 DO
                     WRITE(DATA_FILE,ROUND(DATA_POINTS[J].VAL),'   ') ;
                  WRITELN(DATA_FILE) ;
               FOR I := 2 TO ((CR21X_CHANNELS * MEANS_BINS * AMPLITUDE_BINS) DIV 8)  DO
                  BEGIN
                     TRANSLATE(RAIN_ARRAY[I],K,DATA_POINTS) ;
                     FOR J := 1 TO 8 DO
                        WRITE(DATA_FILE,ROUND(DATA_POINTS[J].VAL),'   ') ;
                     WRITELN(DATA_FILE) ;
                  END ;
               I := (((CR21X_CHANNELS * MEANS_BINS * AMPLITUDE_BINS) DIV 8) + 1) ;
               TRANSLATE(RAIN_ARRAY[I],K,DATA_POINTS) ;
                  FOR J := 1 TO 7 DO
                     WRITE(DATA_FILE,ROUND(DATA_POINTS[J].VAL),'   ') ;
                  WRITELN(DATA_FILE) ;
            END ; { for days }
         CLOSE(DATA_FILE) ;
{**}     ACQ_SCREEN ;
```

```
      CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
   END ;

BEGIN
   DONE := FALSE ;
   WRCOL(ACTIVITY_MESS_ROW,0,3,'Verifying channels ... ') ;
   WAIT(2) ;
   GET_NUMBER_OF_CHANNELS ;
   WAIT(2) ;
   IF CR21X_CHANNELS = 0
      THEN
         BEGIN
            WRCOL(ACTIVITY_MESS_ROW,0,3,'No channels to read ... Press any key ') ;
            GETC(KEY) ;
            DONE := TRUE ;
         END ;
   CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
{**}  ACQ_SCREEN ;
   WRCOL(ACTIVITY_MESS_ROW,0,3,'Verifying number of elapsed intervals ... ') ;
   IF NOT DONE
      THEN
         BEGIN
            GET_CR21X_TIME ;
            MINUTES := ((DAYS_ELAPSED - 1) * 24 * 60) + (CR21X_TIME.HOUR * 60) +
                       (CR21X_TIME.MINUTE) + 1440 -
                       (START_TIME.HOUR * 60  +  START_TIME.MINUTE) ;
            INTERVALS_ELAPSED := MINUTES DIV RAIN_INTERVAL ;
            IF INTERVALS_ELAPSED > MAX_INTERVALS THEN INTERVALS_ELAPSED := MAX_INTERVALS ;
            IF INTERVALS_ELAPSED = 0
               THEN
                  BEGIN
                     WRCOL(ACTIVITY_MESS_ROW,0,3,'No data to read ... Press any key ...') ;
                     GETC(KEY) ;
                     DONE := TRUE ;
                  END ;
         END ;
   WHILE NOT DONE DO
      BEGIN
         IF (FNAME[0] <> CHR(0))
            THEN
               BEGIN
                  FOR I := 1 TO ORD(FNAME[0])-2 DO TEMP_NAME[I] := FNAME[I+2] ;
                  TEMP_NAME[0] := CHR(ORD(FNAME[0])-2)
               END ;
         CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
         WRCOL(ACTIVITY_MESS_ROW,FILE_COL,3,'Save File : ') ;
         CURSOR(ACTIVITY_MESS_ROW,FILE_COL+12) ;
         FNAME[0] := CHR(0) ;
         GETNAM ;
         IF (FNAME[0] <> CHR(0)) { file specified }
            THEN
               BEGIN
                  I := 1 ;
                  WHILE FNAME[I] <> '.' DO I := I + 1 ;
                  FNAME[I+1] := 'R' ;
                  FNAME[I+2] := 'F' ;
                  FNAME[I+3] := 'L' ;
                  ASSIGN(DATA_FILE,FNAME) ;
                  {$I-}
                  RESET(DATA_FILE) ;
                  CLOSE(DATA_FILE) ;
                  {$I+} ;
                  FOUND := (IOresult = 0) ;
                  IF FOUND
                     THEN  { duplicate file exists }
                        BEGIN
                           WRCOL(ACTIVITY_MESS_ROW,FILE_COL+25,4,'File exists ... Overwrite (Y or N) ') ;
                           DN := FALSE ;
```

```
                    WHILE NOT DN DO
                       BEGIN
                          GETC(KEY) ;
                          IF CHR(KEY) IN ['Y','y','N','n'] THEN DN := TRUE ;
                       END ;
                    IF CHR(KEY) IN ['Y','y']
                       THEN { overwite file }
                          BEGIN
                             RETRIEVE ;
                             DONE := TRUE ;
                          END ;
                 END { then }
              ELSE
                 BEGIN
                    RETRIEVE ;
                    DONE := TRUE ;
                 END ;
           END
        ELSE DONE := TRUE ;
     END ;
  END ;

{------------------------------------------------------------------------------}

PROCEDURE ACQUISITION ;
  VAR
     DELAY          : INTEGER    ;
     DATA_POINTS    : VAL_ARRAY  ;
     I , J , K      : INTEGER    ;
     OLD_PAGE       : BYTE       ;
     DONE , AVAIL   : BOOLEAN    ;
     CH             : CHAR       ;
     RESPONSE1      : MESSAGE    ;
     COUNTER        : INTEGER    ;
     TEMP_REAL      : REAL       ;
     N_CHAN         : INTEGER    ;
     PCHAN          : PLOT_CHANNELS ;
  PROCEDURE GET_PCHAN(VAR N_CHAN : INTEGER ; VAR PCHAN : PLOT_CHANNELS) ;
     VAR
        ROW , COL : INTEGER ;
        DONE : BOOLEAN ;
        FOUND : BOOLEAN ;
        TEMP_REAL : REAL ;
        CHAN : INTEGER ;
     BEGIN
        ROW := ACTIVITY_MESS_ROW ;
        COL := 31 ;
        CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
        WRCOL(ACTIVITY_MESS_ROW,0,3,'Enter channels to be plotted : ') ;
        DONE := FALSE ;
        N_CHAN := 0 ;
        WHILE NOT DONE DO
           BEGIN
              TEMP_REAL := -9999.9 ;
              CURSOR(ROW,COL) ;
              GETNUM(TEMP_REAL,1,0) ;
              IF TEMP_REAL > 0.0
                 THEN
                    BEGIN
                       CHAN := ROUND(TEMP_REAL) ;
                       FOUND := FALSE ;
                       FOR I := 1 TO CR21X_CHANNELS DO
                          BEGIN
                             IF CHAN = ACTIVE_CHANNELS[I]
                                THEN FOUND := TRUE ;
                          END ;
                       FOR I := 1 TO N_CHAN DO
                          BEGIN
                             IF CHAN = PCHAN[I]
                                THEN FOUND := FALSE ;
                          END ;
                       IF FOUND
                          THEN
                             BEGIN
```

```
                                        N_CHAN := N_CHAN + 1 ;
                                        PCHAN[N_CHAN] := CHAN ;
                                        WRCOL(ROW,COL,3,I_TO_A(CHAN)) ;
                                        WRCOL(ROW,COL+1,3,',') ;
                                        COL := COL + 2 ;
                                     END
                                 ELSE WRCOL(ROW,COL,3,'  ') ;
                              END
                          ELSE DONE := TRUE ;
                   END ;
            END ;
      BEGIN
            DONE := FALSE ;
            CLS(0,0,24,79,0) ;
{
            OLD_PAGE := CURR_PAGE ;
            CURR_PAGE := 2 ;
            SEL_PAGE ;
            IF NOT ACQ_SCREEN_SET THEN
}
            ACQ_SCREEN ;
            WHILE NOT DONE DO
               BEGIN
                   CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                   WRCOL(ACTIVITY_MESS_ROW,0,3,'Waiting for command ...') ;
                   GETC(KEY) ;
                   CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                   CASE KEY_PRESSED OF
                      F1  : BEGIN  { check     }
                               WRCOL(ACTIVITY_MESS_ROW,4,3,'Checking channels ...... ') ;
                               WRCOL(ACTIVITY_MESS_ROW,60,3,'(start scanner)') ;
                               TOGGLE_RECORD(2)        ;
                               WRCOL(ACTIVITY_MESS_ROW,60,3,'(stop  scanner)') ;
                               TOGGLE_RECORD(2)        ;
{**}                           ACQ_SCREEN ;
{**}                           WRCOL(ACTIVITY_MESS_ROW,4,3,'Checking channels ...... ') ;
                               WRCOL(ACTIVITY_MESS_ROW,60,3,'(get  pointers)') ;
                               GET_POINTERS            ;
                               COUNTER := NSCAN        ;
                               FOR I := 1 TO CR21X_CHANNELS DO
                                  BEGIN
                                     J := ACTIVE_CHANNELS[I] ;
                                     CHANNEL_INFO[J].CURRENT :=  0.0      ;
                                     CHANNEL_INFO[J].HI      := -99999.9 ;
                                     CHANNEL_INFO[J].LO      := +99999.9 ;
                                  END ;
{**}                           ACQ_SCREEN ;
{**}                           WRCOL(ACTIVITY_MESS_ROW,4,3,'Checking channels ...... ') ;
                               WRCOL(ACTIVITY_MESS_ROW,60,3,'(checking scan     )') ;
                               FOR K := 1 TO NSCAN DO
                                  BEGIN
                                     WRCOL(ACTIVITY_MESS_ROW,75,3,'    ') ;
                                     WRCOL(ACTIVITY_MESS_ROW,75,3,I_TO_A(COUNTER)) ;
                                     COUNTER := COUNTER - 1 ;
                                     DUMP_SINGLE(DATA_POINTS) ;
                                     FOR I := 1 TO CR21X_CHANNELS DO
                                        BEGIN
                                           J := ACTIVE_CHANNELS[I] ;
                                           CHANNEL_INFO[J].CURRENT := DATA_POINTS[I].VAL + CHANNEL_INFO[J].CURREN
                                           IF CHANNEL_INFO[J].HI < DATA_POINTS[I].VAL
                                              THEN CHANNEL_INFO[J].HI := DATA_POINTS[I].VAL ;
                                           IF CHANNEL_INFO[J].LO > DATA_POINTS[I].VAL
                                              THEN CHANNEL_INFO[J].LO := DATA_POINTS[I].VAL ;
                                        END ;
                                  END ; { for k }
                               WRCOL(ACTIVITY_MESS_ROW,60,3,'                    ') ;
                               CLS(9,40,17,79,0) ;
                               WRCOL(9,44,3,'  LO   ') ;
                               WRCOL(9,57,3,'AVERAGE') ;
```

```
                  WRCOL(9,68,3,'  HI    ') ;
                  FOR I := 1 TO CR21X_CHANNELS DO
                     BEGIN
                        J := ACTIVE_CHANNELS[I] ;
                        CHANNEL_INFO[J].CURRENT := CHANNEL_INFO[J].CURRENT / NSCAN ;
                        WRCOL(9+I,40,3,I_TO_A(J)) ;
                        WRCOL(9+I,41,3,' : ') ;
                        WRCOL(9+I,44,3,F_TO_A(CHANNEL_INFO[J].LO,2,4)) ;
                        WRCOL(9+I,56,3,F_TO_A(CHANNEL_INFO[J].CURRENT,2,4)) ;
                        WRCOL(9+I,68,3,F_TO_A(CHANNEL_INFO[J].HI,2,4)) ;
                     END ;
                  CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                  CURSOR(ACTIVITY_MESS_ROW,0) ;
               END ;
         F2  : BEGIN { single reading }
                  WRCOL(ACTIVITY_MESS_ROW,4,3,'Taking single reading .. ') ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(start scanner)') ;
                  TOGGLE_RECORD(2)         ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(stop  scanner)') ;
                  TOGGLE_RECORD(2)         ;
{**}              ACQ_SCREEN ;
{**}              WRCOL(ACTIVITY_MESS_ROW,4,3,'Taking single reading .. ') ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(get  pointers)') ;
                  GET_POINTERS             ;
{**}              ACQ_SCREEN ;
{**}              WRCOL(ACTIVITY_MESS_ROW,4,3,'Taking single reading .. ') ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(memory search)') ;
                  JUMP_TO(DSP)             ;
{**}              ACQ_SCREEN ;
{**}              WRCOL(ACTIVITY_MESS_ROW,4,3,'Taking single reading .. ') ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(memory position)') ;
                  MOVE_BACK                ;
{**}              ACQ_SCREEN ;
{**}              WRCOL(ACTIVITY_MESS_ROW,4,3,'Taking single reading .. ') ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(retrieve data)  ') ;
                  DUMP_SINGLE(DATA_POINTS) ;
                  FOR I := 1 TO CR21X_CHANNELS DO
                     BEGIN
                        J := ACTIVE_CHANNELS[I] ;
                        CHANNEL_INFO[J].CURRENT := (DATA_POINTS[I].VAL - CHANNEL_INFO[J].OFFSET) /
                                                   CHANNEL_INFO[J].MULTIPLIER ;
                     END ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'                    ') ;
                  CLS(9,40,17,79,0) ;
                  WRCOL(9,60,3,'Current') ;
                  FOR I := 1 TO CR21X_CHANNELS DO
                     BEGIN
                        J := ACTIVE_CHANNELS[I] ;
                        WRCOL(9+I,55,3,I_TO_A(J)) ;
                        WRCOL(9+I,59,3,F_TO_A(CHANNEL_INFO[J].CURRENT,2,4)) ;
                     END ;
                  CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                  CURSOR(ACTIVITY_MESS_ROW,0) ;
               END ; { F2 }
         F3  : BEGIN  { zero       }
                  WRCOL(ACTIVITY_MESS_ROW,4,3,'Taking zero readings ... ') ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(start scanner)') ;
                  TOGGLE_RECORD(2)         ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(stop  scanner)') ;
                  TOGGLE_RECORD(2)         ;
{**}              ACQ_SCREEN ;
{**}              WRCOL(ACTIVITY_MESS_ROW,4,3,'Taking zero readings ... ') ;
                  WRCOL(ACTIVITY_MESS_ROW,60,3,'(get  pointers)') ;
```

```
                    GET_POINTERS          ;
                    COUNTER := NSCAN      ;
                    FOR I := 1 TO CR21X_CHANNELS DO
                       BEGIN
                          J := ACTIVE_CHANNELS[I] ;
                          CHANNEL_INFO[J].ZERO := 0.0 ;
                       END ;
{**}                ACQ_SCREEN ;
{**}                WRCOL(ACTIVITY_MESS_ROW,4,3,'Taking zero readings ... ') ;
                    WRCOL(ACTIVITY_MESS_ROW,60,3,'(averaging scan     )') ;
                    FOR K := 1 TO NSCAN DO
                       BEGIN
                          WRCOL(ACTIVITY_MESS_ROW,76,3,'    ') ;
                          WRCOL(ACTIVITY_MESS_ROW,76,3,I_TO_A(COUNTER)) ;
                          COUNTER := COUNTER - 1 ;
                          DUMP_SINGLE(DATA_POINTS) ;
                          FOR I := 1 TO CR21X_CHANNELS DO
                             BEGIN
                                J := ACTIVE_CHANNELS[I] ;
                                CHANNEL_INFO[J].ZERO := DATA_POINTS[I].VAL + CHANNEL_INFO[J].ZERO ;
                             END ;
                       END ; { for k }
                    WRCOL(ACTIVITY_MESS_ROW,60,3,'                        ') ;
                    CLS(9,40,17,79,0) ;
                    WRCOL(9,62,3,'ZERO') ;
                    FOR I := 1 TO CR21X_CHANNELS DO
                       BEGIN
                          J := ACTIVE_CHANNELS[I] ;
                          CHANNEL_INFO[J].ZERO := CHANNEL_INFO[J].ZERO / NSCAN ;
                          CHANNEL_INFO[J].ZERO := (CHANNEL_INFO[J].ZERO - CHANNEL_INFO[J].OFFSET) /
                                                   CHANNEL_INFO[J].MULTIPLIER ;
                          WRCOL(9+I,55,3,I_TO_A(J)) ;
                          WRCOL(9+I,59,3,F_TO_A(CHANNEL_INFO[J].ZERO,2,4)) ;
                       END ;
                    CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                    CURSOR(ACTIVITY_MESS_ROW,0) ;
                 END ; { F3 }
         F4    : BEGIN { zero the campbell }
                    FOR I := 1 TO CR21X_CHANNELS DO
                       BEGIN
                          J := ACTIVE_CHANNELS[I] ;
                          CHANNEL_INFO[J].MULTIPLIER := (95.0 * CHANNEL_INFO[J].S_CALIB) /
                                                  (CHANNEL_INFO[J].SR_MAX * CHANNEL_INFO[J].MVPE
                          CHANNEL_INFO[J].OFFSET     := CHANNEL_INFO[J].ZERO * CHANNEL_INFO[J].MULTIPL
                       END ;
                    WRCOL(ACTIVITY_MESS_ROW,0,3,'Channel description must be saved (updated). Press an
                    GETC(KEY) ;
                    CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                    SAVE_DESCRIPTION(ACTIVITY_MESS_ROW,0) ;
                    WRCOL(ACTIVITY_MESS_ROW,0,BLINK,SEND_MESSAGE) ;
                    SEND_PROGRAM ;
                    CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                    CURSOR(ACTIVITY_MESS_ROW,0) ;
{**}                ACQ_SCREEN ;
                 END ; { F4 }
         F5    : BEGIN  { capture truck }
                    WRCOL(ACTIVITY_MESS_ROW,4,3,'Initializing the data logger ... ') ;
                    TOGGLE_RECORD(2)    ;
{**}                ACQ_SCREEN ;
                    CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                    CURSOR(ACTIVITY_MESS_ROW,0) ;
                    WRCOL(ACTIVITY_MESS_ROW,4,3,'Capturing Data ... Press any key to stop ... ') ;
                    REPEAT
                       BEGIN
```

```
                    PUTC_COM(CR) ;
                    READLN(AUX,RESPONSE1) ;
                 END ;
              UNTIL KEYPRESSED ;
{**}          ACQ_SCREEN ;
              CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
              CURSOR(ACTIVITY_MESS_ROW,0) ;
              WRCOL(ACTIVITY_MESS_ROW,4,3,'Capture stopped ... Updating pointers ...     ') ;
              TOGGLE_RECORD(2)       ;
              GET_POINTERS           ;
{**}          ACQ_SCREEN ;
              CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
              CURSOR(ACTIVITY_MESS_ROW,0) ;
           END ; { F5 }
      F6  : BEGIN  { retrieve single truck }
              RETRIEVE_SINGLE_TRUCK ;
{**}          ACQ_SCREEN ;
              CURSOR(ACTIVITY_MESS_ROW,0) ;
           END ;
      F7  : BEGIN  { plot data }
              GET_PCHAN(N_CHAN , PCHAN) ;
              HIRES ;
              PLOT_CHANNEL(N_CHAN , NSCAN , PCHAN) ;
              GETC(KEY) ;
              TEXTMODE ;
              ACQ_SCREEN ;
           END ;
      F8  : BEGIN  { save data }
              SAVE_SINGLE_TRUCK ;
              CURSOR(ACTIVITY_MESS_ROW,0) ;
           END ;
      F9  : BEGIN  { prepare rainflow }
              WRCOL(ACTIVITY_MESS_ROW,0,3,'Enter Rainflow period in minutes (24 hrs=1440 mins, '
              CURSOR(ACTIVITY_MESS_ROW,66) ;
              TEMP_REAL := -1.0 ;
              WHILE ((TEMP_REAL < 0.0) OR (TEMP_REAL > 1440.0)) DO
                 BEGIN
                    WRCOL(ACTIVITY_MESS_ROW,61,3,'      ') ;
                    CURSOR(ACTIVITY_MESS_ROW,61) ;
                    GETNUM(TEMP_REAL,4,0) ;
                 END ;
              RAIN_INTERVAL := ROUND(TEMP_REAL) ;
              IF RAIN_INTERVAL > 0
                 THEN
                    BEGIN
                       CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                       GET_TIME(ACTIVITY_MESS_ROW,0) ;
                       START_TIME.HOUR := CURRENT_TIME.HOUR ;
                       START_TIME.MINUTE := CURRENT_TIME.MINUTE ;
                       CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                       WRCOL(ACTIVITY_MESS_ROW,0,3,
                             'Channel description must be saved (updated). Press any key ... ')
                       GETC(KEY) ;
                       CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                       SAVE_DESCRIPTION(ACTIVITY_MESS_ROW,0) ;
                       CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                       WRCOL(ACTIVITY_MESS_ROW,0,BLINK,SEND_MESSAGE) ;
                       SEND_PROGRAM ;
{**}                   ACQ_SCREEN ;
                       CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                       WAIT(1) ;
                       WRCOL(ACTIVITY_MESS_ROW,0,3,'Setting data logger time ... ');
                       SET_CR21X_TIME(CURRENT_TIME) ;
{**}                   ACQ_SCREEN ;
                       WRCOL(ACTIVITY_MESS_ROW,0,3,'Setting Rainflow capture flag ... ') ;
```

```
                                          WAIT(2) ;
                                          TOGGLE_RECORD(1) ;
                                          CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
{**}                                      ACQ_SCREEN ;
                                       END ;
                           END ; { F9 }
                    F10 : BEGIN { save rainflow data }
                              WRCOL(ACTIVITY_MESS_ROW,0,3,'Setting Rainflow capture flag off ... ') ;
                              INIT_CAMPBELL ;
                              WAIT(5) ;
                              TOGGLE_RECORD(1) ;
{**}                          ACQ_SCREEN ;
                              CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                              WRCOL(ACTIVITY_MESS_ROW,0,3,'Channel description must read. Press any key ...  ') ;
                              GETC(KEY) ;
                              CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                              LOAD_FILE(ACTIVITY_MESS_ROW,0) ;
                              CLS(ACTIVITY_MESS_ROW,0,ACTIVITY_MESS_ROW,79,0) ;
                              SAVE_RAINFLOW_DATA ;
                              CURSOR(ACTIVITY_MESS_ROW,0) ;
                          END ; { F10 }
                    ESC : DONE := TRUE ;
                END ; { case }
            END ; { while }
{
        CURR_PAGE := OLD_PAGE ;
        SEL_PAGE ;
}
        CLS(0,0,24,79,0) ;
    END ;

{-----------------------------------------------------------------------------}

PROCEDURE DESCRIBE ;
    VAR
        DONE     : BOOLEAN ;
        DN       : BOOLEAN ;
        MESS     : MESSAGE ;
        CHANNEL  : INTEGER ;
    BEGIN
{
        CURR_PAGE := 1 ;
        SEL_PAGE ;
}

{
        IF NOT SCREEN_SET
            THEN
                BEGIN
}
        CLS(0,0,24,79,0) ;
                    SHOW_TABLE ;
                    SHOW_VALUES ;
{
                END ;
}
        WHILE NOT DONE DO
            BEGIN
                ROW := CHANNEL_INFO[CURRENT_CHANNEL].ROW ;
                CASE CURRENT_FIELD OF
                    1 : BEGIN
                            COL := CHANNEL_INFO[CURRENT_CHANNEL].TRANSDUCER_COL ;
                            GET_TRANSDUCER(CHANNEL_INFO[CURRENT_CHANNEL].TRANSDUCER,ROW,COL) ;
                            CASE CHANNEL_INFO[CURRENT_CHANNEL].TRANSDUCER OF
                                0 : CHANNEL_INFO[CURRENT_CHANNEL].EXCITATION := -1 ;
                                1 : BEGIN { strain gage }
                                        CHANNEL_INFO[CURRENT_CHANNEL].EXCITATION := DEFAULT_SG_EXCITATION ;
                                        CHANNEL_INFO[CURRENT_CHANNEL].RANGE      := DEFAULT_SG_RANGE      ;
                                    END ;
```

```
                2 : BEGIN { transducer }
                     CHANNEL_INFO[CURRENT_CHANNEL].EXCITATION := DEFAULT_DT_EXCITATION ;
                     CHANNEL_INFO[CURRENT_CHANNEL].RANGE      := DEFAULT_DT_RANGE      ;
                    END ;
            END ; { case }
        END ;
    2 : GET_S_CALIB(CHANNEL_INFO[CURRENT_CHANNEL].S_CALIB,CHANNEL_INFO[CURRENT_CHANNEL].ROW,
                   CHANNEL_INFO[CURRENT_CHANNEL].DETAIL_COL-15)  ;
    3 : GET_MVPERV_CALIB(CHANNEL_INFO[CURRENT_CHANNEL].MVPERV_CALIB,CHANNEL_INFO[CURRENT_CHANN
                   CHANNEL_INFO[CURRENT_CHANNEL].DETAIL_COL-7)  ;
    4 : GET_FATIGUE_DETAIL(CHANNEL_INFO[CURRENT_CHANNEL]) ;
    5 : GET_SR_MAX(CURRENT_CHANNEL) ;
    6 : GET_SR_MIN(CURRENT_CHANNEL) ;
END ; { case }
CASE KEY_PRESSED OF
   RT_ARROW : BEGIN
                IF CURRENT_FIELD = 6
                   THEN CURRENT_FIELD := 1
                   ELSE CURRENT_FIELD := CURRENT_FIELD + 1 ;
              END ;
  LT_ARROW  : BEGIN
                IF CURRENT_FIELD = 1
                   THEN CURRENT_FIELD := 6
                   ELSE CURRENT_FIELD := CURRENT_FIELD - 1 ;
              END ;
  DN_ARROW  : BEGIN
                IF CURRENT_CHANNEL = 8
                   THEN CURRENT_CHANNEL := 1
                   ELSE CURRENT_CHANNEL := CURRENT_CHANNEL + 1 ;
              END ;
  UP_ARROW  : BEGIN
                IF CURRENT_CHANNEL = 1
                   THEN CURRENT_CHANNEL := 8
                   ELSE CURRENT_CHANNEL := CURRENT_CHANNEL - 1 ;
              END ;
  F1        : BEGIN
                LOAD_FILE(FILE_ROW,FILE_COL) ;
                SHOW_VALUES ;
              END ;
  F2        : SAVE_DESCRIPTION(FILE_ROW,FILE_COL) ;
  F3        : BEGIN
                CANCELLED := FALSE ;
                WHILE (NOT ALL_DEFINED) AND (NOT CANCELLED) DO
                   BEGIN
                      MESS := I_TO_A(CURRENT_CHANNEL)         ;
                      DEFINE_MESSAGE[9] := MESS[1]            ;
                      WRCOL(MESSAGE_ROW,0,3,DEFINE_MESSAGE) ;
                      DN := FALSE ;
                      WHILE NOT DN DO
                         BEGIN
                            GETC(KEY) ;
                            IF (CHR(KEY) IN ['Y','y','N','n']) OR (KEY = 27)
                               THEN DN := TRUE
                         END ;
                      IF KEY = 27
                         THEN CANCELLED := TRUE
                         ELSE IF (CHR(KEY) IN ['Y','y'])
                                 THEN ELIMINATE(CURRENT_CHANNEL)
                                 ELSE CANCELLED := TRUE
                   END ;
                IF NOT CANCELLED
                   THEN
                      BEGIN
                         EMPTY := TRUE ;
                         FOR I := 1 TO 8 DO
                            IF DEFINED(I) THEN EMPTY := FALSE ;
                         IF EMPTY
                            THEN BEGIN
                                    WRCOL(MESSAGE_ROW,0,BLINK,EMPTY_MESSAGE) ;
```

```
                                                      GETC(KEY) ;
                                                      CLS(MESSAGE_ROW,0,MESSAGE_ROW,79,0) ;
                                                   END
                                              ELSE BEGIN
                                                      WRCOL(MESSAGE_ROW,0,BLINK,SEND_MESSAGE) ;
                                                      SEND_PROGRAM ;
                                                      CLS(MESSAGE_ROW,0,MESSAGE_ROW,79,0) ;
                                                   END ;
                                        END ;
                              END ;
               F5           : BEGIN
                                 INIT_SR_TABLE ;
                                 UPDATE_SCREEN_SR ;
                              END ;
               ESC          : BEGIN
                                 DONE := TRUE ;
                                 CR21X_CHANNELS := 0 ;
                                 FOR CHANNEL := 1 TO 8 DO
                                    BEGIN
                                       IF DEFINED(CHANNEL)
                                          THEN
                                             BEGIN
                                                CR21X_CHANNELS := CR21X_CHANNELS + 1 ;
                                                ACTIVE_CHANNELS[CHANNELS] := CHANNEL ;
                                             END ;
                                    END ;
{
                                 CURR_PAGE := 0 ;
                                 SEL_PAGE ;
}
      CLS(0,0,24,79,0) ;
                              END ;
               DEL          : ELIMINATE(CURRENT_CHANNEL) ;
               ELSE           BEGIN
                                 CASE CURRENT_FIELD OF
                                    1 : CURRENT_FIELD := 2 ;
                                    2 : BEGIN
                                           IF CHANNEL_INFO[CURRENT_CHANNEL].DETAIL > 7
                                              THEN CURRENT_FIELD := 3
                                              ELSE CURRENT_FIELD := 1
                                        END ;
                                    3 : CURRENT_FIELD := 4 ;
                                    4 : CURRENT_FIELD := 1 ;
                                 END { case }
                              END ;
            END
         END ; { while not done }
   END ;

{-----------------------------------------------------------------------------}

PROCEDURE DIRECT ;
   VAR
      DONE : BOOLEAN ;
   BEGIN
      CLS(0,0,24,79,0) ;
      DONE := FALSE ;
      WHILE NOT DONE DO
         BEGIN
            WHILE NOT KEYPRESSED DO
               BEGIN
                  GETC_COM(CH , CHAR_AVAIL) ;
                  WHILE CHAR_AVAIL DO
                     BEGIN
                        IF ORD(CH) <> 17 THEN WRITE(CH) ;
                        GETC_COM(CH , CHAR_AVAIL) ;
                     END ;
               END ;
            GETC(KEY) ;
```

```
         CASE KEY_PRESSED OF
            F1 : SET_PAR            ;
            F2 : INIT_CAMPBELL      ;
            F3 : GO_REMOTE          ;
            ESC: DONE := TRUE
            ELSE
               BEGIN
                  CH := CHR(KEY) ;
                  PUTC_COM(CH) ;
               END ;
         END ; { case }
      END ;
   CLS(0,0,24,79,0) ;
END ;

{-----------------------------------------------------------------------------}

BEGIN
   INITIALIZE ;
   DONE := FALSE ;
   WHILE NOT DONE DO
      BEGIN
         TITLE :='           TEXAS STATE DEPARTMENT OF HIGHWAYS AND PUBLIC TRANSPORTATION
         WRCOL(0,0,HILITE,TITLE) ;
         WRCOL(10,10,3,'F3   : Low Level Programming') ;
         WRCOL(13,10,3,'F5   : Channel Description') ;
         WRCOL(16,10,3,'F9   : Data Acquisition Menu') ;
         WRCOL(19,10,3,'ESC  : Exit to DOS ') ;
         GETC(KEY) ;
         CASE KEY_PRESSED OF
            F3 : DIRECT               ;
            F5 : DESCRIBE             ;
            F9 : { IF PROGRAMMED THEN } ACQUISITION ;
            ESC: BEGIN
                    GET_CURSOR(CUR_ROW,CUR_COL) ;
                    WRCOL(24,55,3,'EXIT TO DOS (Y or N) ? ') ;
                    REPEAT
                       GETC(KEY) ;
                    UNTIL KEY IN [89,121,78,110] ;
                    IF KEY IN [89,121]
                       THEN DONE := TRUE
                       ELSE BEGIN
                               WRCOL(24,55,3,'                         ') ;
                               CURSOR(CUR_ROW,CUR_COL) ;
                            END ;
                 END ;
         END ; { case }
      END ; { while }
END.
```