ALEX(01)-TR-77-09

12 B.S.

THE EXTENDED INTERACTIVE SEISMIC PROCESSING SYSTEM (ISPSE)

TECHNICAL REPORT NO. 9

VELA NETWORK EVALUATION AND AUTOMATIC PROCESSING RESEARCH

Prepared by
Jeffrey S. Shaub and David G. Black

TEXAS INSTRUMENTS INCORPORATED
Equipment Group
Post Office Box 6015
Dallas, Texas 75222

Prepared for

AIR FORCE TECHNICAL APPLICATIONS CENTER
Alexandria, Virginia 22314

Sponsored by

ADVANCED RESEARCH PROJECTS AGENCY
Nuclear Monitoring Research Office
ARPA Program Code No. 7F10
ARPA Order No. 2551

5 December 1977

Acknowledgment: This research was supported by the Advanced Research Projects Agency, Nuclear Monitoring Research Office, under Project VELA-UNIFORM, and accomplished under the technical direction of the Air Force Technical Applications Center under Contract Number F08606-77-C-0009.

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED

ALEX(01)-TR-77-09

THE EXTENDED INTERACTIVE SEISMIC PROCESSING SYSTEM (ISPSE)

TECHNICAL REPORT NO. 9

TI-ALEX(01)-TR-77-09

VELA NETWORK EVALUATION AND AUTOMATIC PROCESSING RESEARCH

Prepared by

Jeffrey S. Shaub and David G. Black

TEXAS INSTRUMENTS INCORPORATED
Equipment Group
Post Office Box 6015
Dallas, Texas 75222

F08606-77-C-0009,
ARPA Order-2551

Prepared for

AIR FORCE TECHNICAL APPLICATIONS CENTER
Alexandria, Virginia 22314

Sponsored by

ADVANCED RESEARCH PROJECTS AGENCY
Nuclear Monitoring Research Office
ARPA Program Code No. 7F10
ARPA Order No. 2551

5 Dec 1977

183 P.

Acknowledgment: This research was supported by the Advanced Research Projects Agency, Nuclear Monitoring Research Office, under Project VELA-UNIFORM, and accomplished under the technical direction of the Air Force Technical Applications Center under Contract Number F08606-77-C-0009.

405 076

JOB

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE EXTENDED INTERACTIVE SEISMIC PROCESSING SYSTEM (ISPSE)		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER ALEX(01)-TR-77-09
7. AUTHOR(s) Jeffrey S. Shaub and David G. Black		8. CONTRACT OR GRANT NUMBER(s) F08606-77-C-0004
9. PERFORMING ORGANIZATION NAME AND ADDRESS Texas Instruments Incorporated Equipment Group Dallas, Texas 75222		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS VELA T/7705/B/ETR
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency Nuclear Monitoring Research Office Arlington, Virginia 22209		12. REPORT DATE 5 December 1977
		13. NUMBER OF PAGES 179
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Air Force Technical Applications Center VELA Seismological Center Alexandria, Virginia 22314		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

ARPA Order No. 2551

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

seismology	Programming Language
signal analysis	Command Language
event discrimination	Programmed Procedure
Interactive Processing Environment	Processing Module

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The primary intention of this report is to provide a comprehensive user manual for the Extended Interactive Seismic Processing System (ISPSE). ISPSE supports an interactive graphics environment on the PDP-15 computer located at the Seismic Data Analysis Center (SDAC) for the purpose of analyzing long-and short-period seismic waveforms. In addition to general signal analysis functions, ISPSE demonstrates the feasibility of utilizing interactive graphics for short-period event discrimination → next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

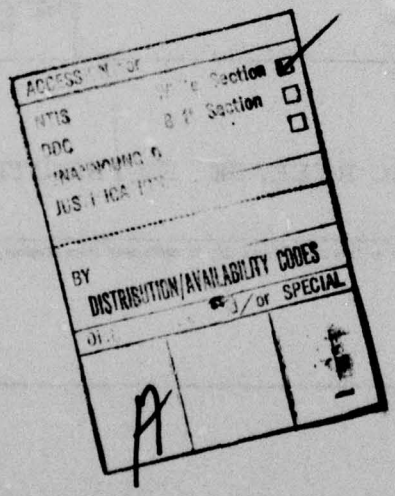
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (continued)

cont.

as described in the report entitled "Documentation of the Extended Interactive Seismic Processing System (ISPSE)" (Shaub, et al., 1977). This report discusses system features, supplemental to that documentation, to ensure optimal utilization of ISPSE for seismic problem solving.

These features include: a programmable mode of operation by which the user may define standard seismic processing tasks on-line; and an interpretive high level Interactive Seismic Programming Language (ISPL) with which a user may define functions that access, perform computation upon, and display seismic data in tabular and graphic form for evaluation and analysis.



ABSTRACT

The primary intention of this report is to provide a comprehensive user manual for the Extended Interactive Seismic Processing System (ISPSE). ISPSE supports an interactive graphics environment on the PDP-15 computer located at the Seismic Data Analysis Center (SDAC) for the purpose of analyzing long- and short-period seismic waveforms. In addition to general signal analysis functions, ISPSE demonstrates the feasibility of utilizing interactive graphics for short-period event discrimination as described in the report entitled "Documentation of the Extended Interactive Seismic Processing System (ISPSE)" (Shaub, et al., 1977). This report discusses system features, supplemental to that documentation, to ensure optimal utilization of ISPSE for seismic problem solving.

These features include: a programmable mode of operation by which the user may define standard seismic processing tasks on-line; and an interpretive high level Interactive Seismic Programming Language (ISPL) with which a user may define functions that access, perform computation upon, and display seismic data in tabular and graphic form for evaluation and analysis.

ACKNOWLEDGMENTS

Several Texas Instruments personnel have contributed to the development of ISPSE. The authors would like to acknowledge in particular Robert L. Sax and T. W. Harley for many valuable discussions concerning seismic implications of ISPSE design criteria. Also David R. Lashmit and David A. Eastburn provided considerable systems programming support during the implementation of ISPSE on the PDP-15/50 computer.

Finally we would like to thank A. William Schmidt who offered many constructive suggestions on the organization of this report and Kathy Vitale who typed all of the text and drafted the figures of this document.

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

LIST OF FIGURES

LIST OF FIGURES
(continued)

SECTION I
INTRODUCTION

The primary intention of this report is to provide a comprehensive user manual for the Extended Interactive Seismic Processing System (ISPSE). ISPSE supports an interactive graphics environment on the PDP-15 computer located at the Seismic Data Analysis Center (SDAC) for the purpose of analyzing long- and short-period seismic waveforms. In addition to general signal analysis functions, ISPSE demonstrates the feasibility of utilizing interactive graphics for short-period event discrimination as described in the report entitled "Documentation of the Extended Interactive Seismic Processing System (ISPSE)" (Shaub, et al., 1977). This report discusses system features, supplemental to that documentation, to ensure optimal utilization of ISPSE for seismic problem solving.

The main objective when extending the Interactive Seismic Processing System (ISPS) to the ISPSE is to develop software which would provide an operationally flexible, reliable, and attractive interactive environment for accomplishing standard seismic processing tasks. More specifically, the emphasis is to create a structure, which would free the geophysicist from the usual burdens of computer analysis (e.g., complex coding, batch compilation, and link editing) and permit him to concentrate on the solution of his seismic problems, without adopting the 'hard-wired' characteristics of many existing analysis packages.

With this objective in mind, ISPS was extended to support two new features, namely:

- A programmable mode of operation where standard seismic processing tasks may be defined (on-line, by the user) as a

sequence of functions and sub-functions to be executed within the ISPSE environment. This minimizes decision making at task execution time and improves reproducibility of results.

- An interpretive high level Interactive Seismic Programming Language (ISPL) with which a user may define his own functions. These functions access, perform computations upon, and display data in tabular and graphic form for evaluation and analysis.

This report consists of six sections and three appendices. Section II presents an overview of ISPSE, and discusses system components and the philosophy of combining these components to provide a seismic processing environment.

Section III offers a detailed description of the ISPSE command language. Command language syntax is described in a notation known as Backus Normal Form (BNF), (Backus, 1959), while semantics are presented by definition and example. Also system capabilities derived from the command language are summarized in this section.

Section IV is devoted entirely to the Interactive Seismic Programming Language (ISPL). It begins with an overview introducing the reader to the operational aspects of ISPL, continues with a detailed semantic description of the language including valid entries, valid statements, constants, variables, and expressions, and concludes with a presentation of ISPL syntax via BNF notation.

Section V discusses the ISPSE programmable feature and the construction and maintenance of standard seismic processing tasks via the creation, programming, and execution of named procedures.

Appendix A concentrates on the use of ISPSE for baseline discrimination, accomplished by executing the programmed procedure named

ISPED and the user function called MDCRM. Appendix B provides a summary of ISPSE error messages generated by the command language and ISPL. Finally, Appendix C is an index of key terms used throughout this report in describing ISPSE.

This report is written with the understanding that the reader is familiar with the ISPSE as described in the "Documentation of the Extended Interactive Seismic Processing System (ISPSE)" (Shaub, et al., 1977). Therefore, that report should be referred to for all user documentation not contained herein.

SECTION II
ISPSE OVERVIEW

A. SYSTEM COMPONENTS

The Extended Interactive Seismic Processing System (ISPSE) is designed for the PDP-15/50 computer located at SDAC. The computer provides several hardware features especially important to ISPSE. They are detailed in Figure II-1. Given these hardware features, together with the system design objectives noted in the Introduction, ISPSE is implemented to offer several important components which may be summarized as follows:

- A graphics console environment
- Interactive seismic processing modules
  - system functions
  - user functions
- Interactive languages
  - command language
  - programming language.

Figure II-2 describes some salient features provided by the graphics console environment. This environment is supported by the graphics display hardware and software of the PDP-15/50 together with ISPSE console support routines.

Under ISPSE seismic data are analyzed via the second ISPSE component, interactive processing modules. These modules are of two types: system functions and user functions, as shown in Figure II-3, hence the words function and module will be used interchangeably throughout this report. System functions are provided for the geophysicist and feature

- A CPU and independent Floating Point Processor
  - 32k executive memory
  - 32k user memory
- A card reader and line printer
- A system control teletype
- Two 7-track tape drives
- Two 9-track tape drives
- A ten megaword disk drive
- A 256k fixed head disk
- A graphics display processor
  - CRT console
  - input only keyboard
- A calcomp plotter.

FIGURE II-1
PDP-15/50 HARDWARE FEATURES

- Immediate CRT display of entered alphanumeric information
  - entry via console keyboard
  - entry on a per character basis with editing capability.
- Free format entry of numeric data with error checking.
- CRT display of system messages
  - error diagnostics
  - prompts to the user
  - processing status
  - system parameters
  - seismic parameters.
- Complete record keeping of all console communications on the line printer.
- Control of CRT screen scrolling via blue pushbutton #6- light on prevents scrolling - light off permits scrolling.
- CRT display of keyword menus within seismic processing modules
  - serve as a processing guide to the user
  - explain analysis options selected by designated push-button.
- CRT display of seismic waveforms and associated graphic data
  - point selections with moving cursor
  - time window selection with moving cursor.
- Calcomp hard copies of selected CRT displays.

FIGURE II-2

FEATURES OF THE GRAPHICS CONSOLE ENVIRONMENT

- **SYSTEM FUNCTIONS** (Provided by ISPSE)
  - **DPSCAN:** Catalogues and displays waveforms from disk
  - **SELEV :** Selects any event, station, component, and time window for analysis
  - **FILTER :** Performs general signal analysis functions
  - **GRVEL :** Generates long-period dispersion curves and fundamental mode source spectrum
  - **SPEED :** Performs short-period earthquake/explosion discrimination.

- **USER FUNCTIONS** (Defined on-line by geophysicist)
  - Access system data bases — Analysis
  - Perform simple computations on data — Analysis
  - Display tables of seismic parameters — Evaluation
  - Display data and x-y plots of data — Evaluation

FIGURE II-3

CAPABILITIES OF THE INTERACTIVE SEISMIC PROCESSING MODULES

advanced signal analysis software for event detection, parameter extraction, and discrimination. (Ringdal, et al., 1971 ; Shaub, et al., 1977) During execution of a system function, processing options may be selected interactively from displayed key word menus. Figure II-4 illustrates information flow for noted system functions. User functions, on the other hand, are defined online via an interpretive high level Interactive Seismic Programming Language (ISPL). That is, a user function is a named sequence of ISPL statements which, having been accepted and saved by ISPSE, may be executed in the same manner as a system function. Hence, the addition of user function capability to the ISPSE environment enables the system to offer a degree of flexibility uncommon to most seismic analysis packages.

The third ISPSE component, interactive languages, permit the geophysicist to control the definition and execution of seismic processing modules within the console environment.

As noted above, ISPL permits the definition of user functions. ISPL statements may be entered on a line-by-line (interactive editing) basis with immediate syntax error diagnostics provided. Moreover, ISPL features all arithmetic and logical operators required for formula translation, and several high-level numerical and I/O operators which simplify seismic programming tasks. These topics will be discussed in detail in Section IV.

Utilizing the ISPSE command language, processing modules may be executed either individually or in procedural mode. A procedure is a named sequence of modules to be executed with "INTERUP" 's inserted between modules in the sequence for branching purposes. Procedures may be programmed to accomplish standard seismic processing tasks and stored for future reference. The command language provides all the facilities for maintaining procedures and will be discussed in detail in Section III.

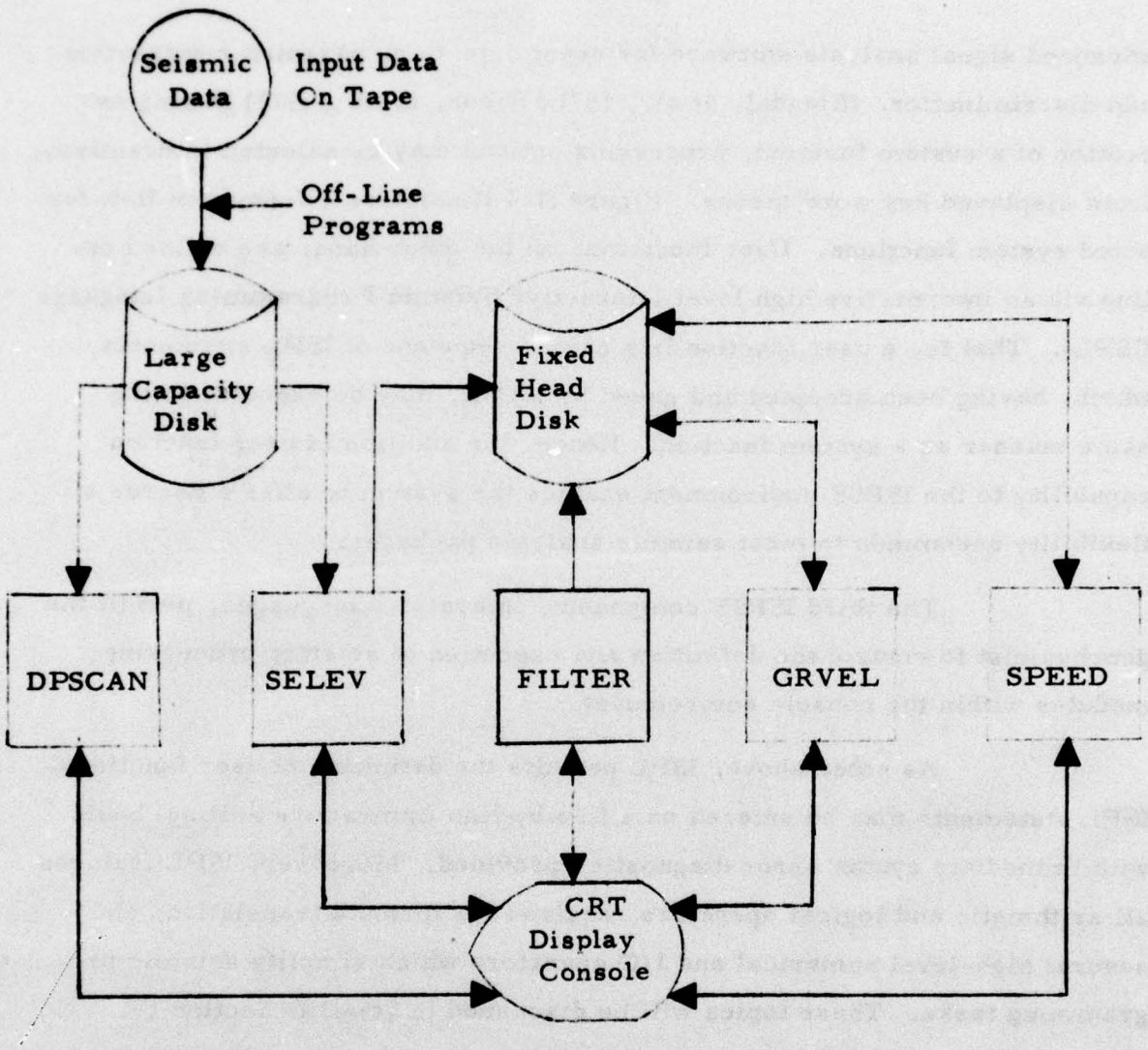


FIGURE II-4

ISPSE INFORMATION FLOW

B. MULTITASKING ENVIRONMENT

ISPSE is physically structured as a set of System Tasks which are controlled by a System Monitor. Communication between System Tasks is accomplished via a System Communication Block as shown in Figure II-5. The physical program structures of ISPSE are:

- System Monitor - a permanently active task that executes concurrently with each System Task. It switches system operation between tasks as required to satisfy ISPSE commands and it also monitors the status of each System Task.
- System Communication Block - an area of memory into which data are stored and retrieved for System Task intercommunication.
- System Tasks - a subdivision of ISPSE which performs command processing and processing modules.

A major advantage to this configuration is the automatic recoverability of ISPSE after a system error, which otherwise would be fatal. The actual physical boundaries of each System Task and the execution of the System Monitor is transparent to the analyst in that ISPSE appears to operate according to the logical organization as described below.

Figure II-6 illustrates the logical configuration of ISPSE whose program structures are:

- General Supervisor - which initializes the ISPSE data structures, and maintains control of the Command Supervisor and Interpreter Supervisor for deciphering and executing ISPSE commands.
- Command Supervisor - prompts the analyst to enter a command and then decodes it after it has been entered.

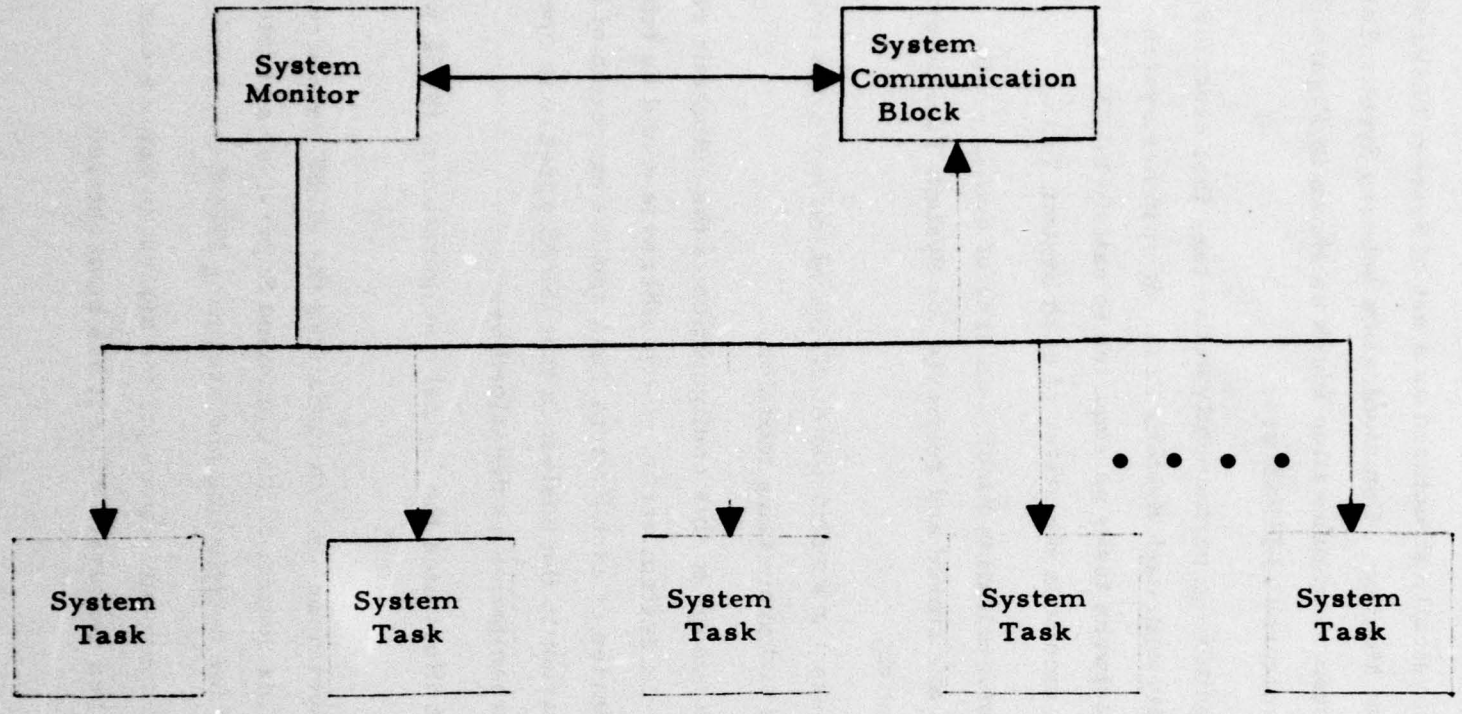


FIGURE II-5

PHYSICAL ORGANIZATION OF ISPSE

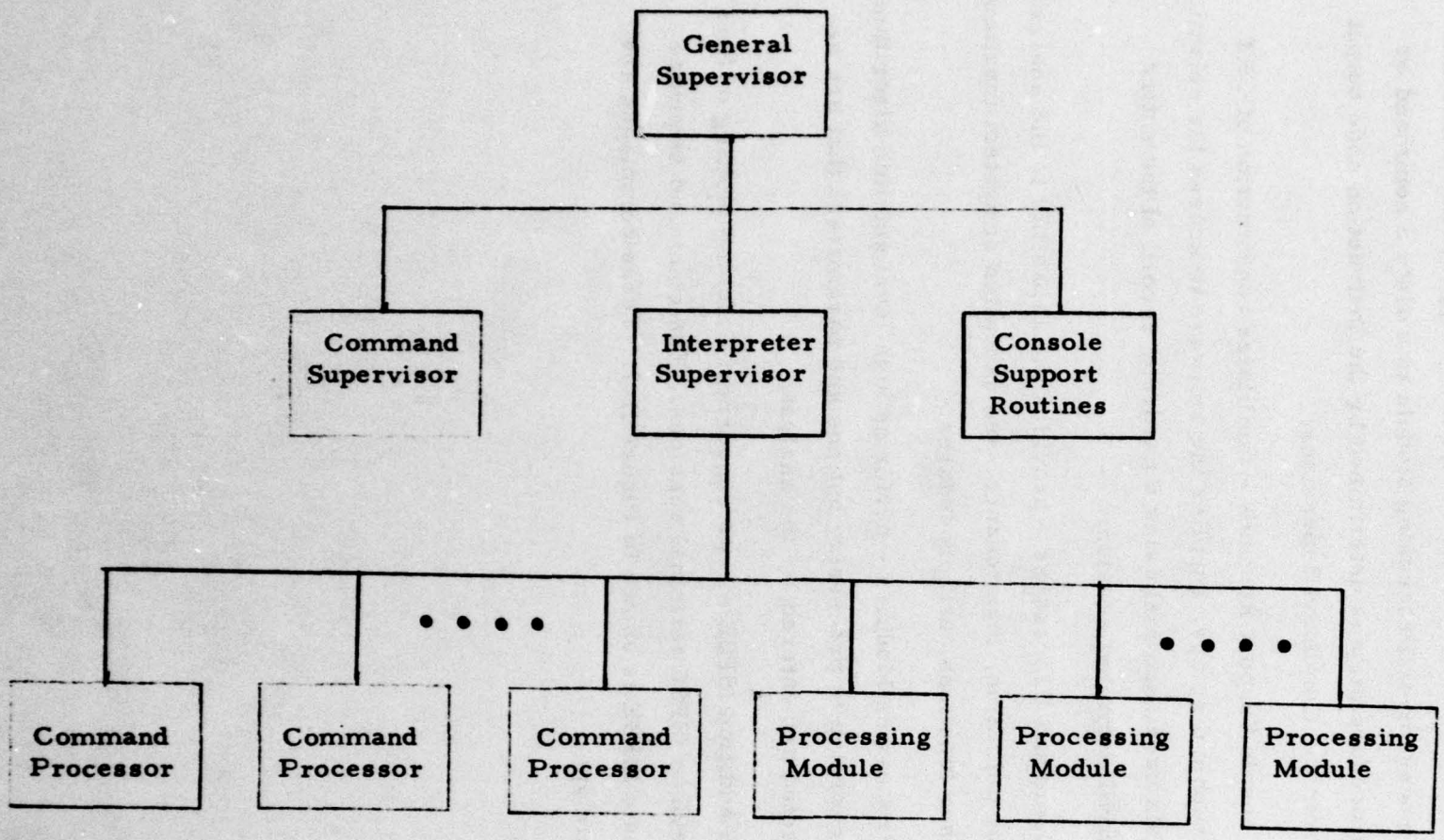


FIGURE II-6

LOGICAL ORGANIZATION OF ISPSE

- Interpreter Supervisor - request the appropriate Command Processor or Processing Module to satisfy a command or procedure step as determined by the instruction code output from the Command Supervisor.
- Console Support Routines - facilitates construction of CRT graphic displays, supplies the software to access the seismic data base, and maintains a hardcopy record of the entire ISPSE terminal session.
- Command Processors - provides the capability to the analyst for definition, maintenance, and execution of system functions, user functions, and procedures.
- Processing Modules - performs high level seismic algorithms according to processing options and parameters that are selected and entered by the analyst.

In addition ISPSE allows concurrent Calcomp plotting of files generated by previous ISPSE terminal sessions. The command sequence required to execute ISPSE is shown in Figure II-7. These commands are entered at the teletype.

```
DOS-15 V3B000
 ENTER DATE (MM/DD/YY) - 9/23/77

$MICLOG SYS

$PIP

DOSPIP V3B000

>I DK


>N DK <RSX>


>^C

DOS-15 V3B000
$LOGOUT


DOS-15 V3B000
$RSX

MCR>DSM RF
DISK IS READY FOR DISMOUNTING
MCR>MNT RF RSX
MCR>REQ ISPSE
MOUNT CALCOMP PLOT TAPE ON MT0
ENTER PLOT SCALE FACTOR (0 TO 1).  FORMAT=F4.2
.8
```

FIGURE II-7

PROCEDURE REQUIRED TO EXECUTE THE ISPSE SYSTEM

SECTION III
ISPSE COMMAND LANGUAGE

The ISPSE command language is described in detail in this section. As each command is discussed, the actions of the Interpreter Supervisor are described. The interaction of commands affecting user functions and their implications on procedures are presented along with several automatic protection features. Also, examples of the commands are provided.

The ISPSE Command Supervisor allows the user to enter either external or internal commands. The two types of commands are defined as follows:

- External command - an entry made when prompted by the four dots .... which initiates system function execution as well as procedure and user function creation, maintenance, and execution.
- Internal command - an entry made when prompted by a sequence number, e.g., 15.0, during the creation and editing of procedures.

A. FUNCTIONAL DESCRIPTION OF THE ISPSE COMMAND LANGUAGE

Each of the legal ISPSE commands is listed below, with a brief explanation of each command's function.

- .HELP - displays a brief explanation of any command or system function.

- .CREATE - enters a new procedure into the ISPSE system directory.
- .EDIT - permits the user to edit an existing procedure.
- .PROCS - displays a list of procedure names.
- .COMPILE - accesses a user function for creation, editing, or deletion.
- .UFUNC - displays a list of user function names.
- .LIST - lists the source lines of a procedure or user function, or the program matrices associated with a programmed procedure.
- .DELETE - removes a procedure or user function from the ISPSE system directory.
- .PERFORM - initiates the execution of a system function, procedure, or user function; inserts a system function, user function or an INTERUP in a procedure; or terminates an ISPSE terminal session.
- .MODE - sets the mode of the command supervisor or displays the current mode.

B. SYSTEM CAPABILITIES

1. Operational Modes

ISPSE can be used in any one of three operational modes, namely:

NORM - normal mode
PRGM - program mode
EXEC - execute mode

Normal mode is used whenever the analyst wishes to individually execute system functions, user functions, or procedures. User functions and procedures are also maintained in this mode. ISPSE is in NORM mode when a seismic processing session is initiated by requesting ISPSE from the teletype (Section II). All legal commands are valid in NORM mode.

A set of processing options selected from the keyword menus, during the execution of a procedure may be programmed into a procedure. ISPSE must be set to program mode in order to program a procedure. Programmed procedures are discussed in detail in Section V. The execution of a programmed procedure is accomplished with ISPSE set to execute mode. When ISPSE is in PRGM or EXEC mode, only the commands MODE, and .PERFORM in conjunction with a procedure name, are valid. All other commands cause an error message to be displayed on the video screen. See Appendix B for a complete list of error messages.

2. User Functions

Several ISPSE commands are provided to define and maintain user functions. They are:

- COMPILE
- UFUNC
- LIST
- DELETE.

These commands may only be used in NORM mode. Two files are associated with each user function. The first file contains the source lines entered when the Interactive Seismic Programming Language (ISPL) command processor is executed. The second file contains the executable code and is only created if a user function is successfully compiled. ISPL is discussed in detail in Section IV.

The . COMPILE command directs the Interpreter Supervisor to invoke the ISPL command processor. A unique name with respect to procedures or system functions must be specified or an error is displayed and the command is ignored. User functions are defined, edited, and deleted via this command. If the user function does not already exist, then the name is entered into the system directory. Otherwise the source file is opened and positioned at the end of the user function. The sequence number displayed is the next available line in the user function. The executable code file, if any, is deleted at this time, i.e., the user function is decompiled. Also, see Section III-4 on programmed procedures.

The user may exit from the ISPL command processor in three separate ways. First, the user function may be deleted. Second, only the user function's source file is saved. And third, both the source and executable code file are saved if the user function is successfully compiled. When the user function is deleted, any procedure that references a user function is modified. This is discussed later in this section under programmed procedures.

A complete list of all user function names in the system directory is obtained by the . UFUNC command. An asterisk (*) appearing at the end of any name indicates that the user function is compiled and therefore may be executed. As many as 36 user functions may exist at one time.

The source file for any user function may be displayed by using the . LIST command and specifying an existing user function name. Any name that is not a user function (or procedure) name causes an error and the command is ignored.

As stated above, a user function may be deleted using the ISPL command processor. Alternatively, the . DELETE command may be used to delete a user function from the system directory. As before, when the user function to be deleted is referenced by a procedure, certain modifications are made to that procedure. This is discussed under programmed procedures.

3. Individual Processing Module Execution

In NORM mode, system functions or user functions may be executed via the .PERFORM command. At this time, system functions are limited to the following keywords:

DPSCAN
SELEV
FILTER
GRVEL
SPEED

which have been described by Ringdal, el al., (1975) and Shaub, et al., (1977). User functions are created by the analyst as described previously. A successfully compiled user function may be executed by means of the .PERFORM command. If the user function is not compiled, an error occurs and the command is terminated.

When a system function is .PERFORM'ed, the Interpreter Supervisor requests the appropriate processing module. Within each system function the analyst may select processing options from keyword menus. After the processing module is exited, the user is free to .PERFORM another system function or user function. When the analyst .PERFORM's a user function the Interpreter Supervisor initiates the processing module called the Interactive Seismic Programming Language Interpreter (ISPLI). The user may execute another function when ISPLI is finished executing the specified user function. If the analyst attempts to .PERFORM a name which is not a system function, user function, or procedure, an error is displayed and the command is ignored. ISPL is discussed in Section IV.

4. Procedures

Procedures consist of an ordered set of system functions, user functions, and INTERUP's. These internal commands, which are divided

into create and edit commands, are illustrated in the BNF description of the ISPSE command language. A procedure may contain no more than 25 statements. The ISPSE commands available for creation, maintenance, and execution of procedures are:

.CREATE
.EDIT
.PROCS
.LIST
.DELETE
.PERFORM.

Each of these commands, with the exception of .PERFORM, may only be entered in NORM mode. The use of .PERFORM in PGRM and EXEC mode is discussed later.

The .CREATE command enters a new procedure name into the system directory. The specified name must not duplicate any system function, user function, or previously defined procedure name. At this time, the Interpreter Supervisor invokes the .CREATE command processor which prompts the user with the sequence number, 1.0. The command processor only allows the user to enter create commands. As each command is placed in the procedure, the sequence number is incremented by one. No editing of procedures can occur within the .CREATE command processor. Any <user name>'s specified in a create command must be a user function contained in the system directory. Therefore a procedure cannot reference a non-existent user function. However, the user function need not be compiled or even completely defined. Placement of INTERUP's within a procedure is discussed in Section V. The #S command is used to exit from the command processor and causes the procedure to be saved on disk.

All procedure editing is accomplished via the .EDIT command. Any procedure that is to be edited must already exist in the system directory. Initially, the line pointer is positioned to the next available line, i.e., the end of the procedure.

Any edit command may be entered during an edit. The edit commands and their functions are:

- #N - resequence the procedure
- #P - display a line or the entire procedure
- #D - delete a line
- #I - insert a line
- #R - replace a line
- #S - exit from the editor.

The #N edit command causes the procedure to be resequenced beginning with one. Each subsequent sequence number is increased by 1.0. The #P command may be used with or without a sequence number. The entire procedure is displayed when #P is entered by itself. An edit command followed by a sequence number causes the appropriate action to occur only on that line. A #P <sequence number> or #D <sequence number> command is ignored if the specified line doesn't exist. The #I<sequence number> command is valid only if the sequence number refers to a new line. #R< sequence number > replaces an existing statement, or inserts a line if the sequence number is not present. The #S command is used to save the procedure and exit from the edit command processor. The procedure is automatically re-sequenced at this time.

The .PROCS external command is similar to the .UFUNC command in that it causes a list of all procedure names in the system directory to be displayed. The maximum number of procedures that are allowed at any one time is 36. A list of the commands in a procedure may be obtained with the .LIST command. If there is no procedure (or user function) in the

system directory by the specified name, then an error is displayed and the command is ignored. A procedure name is removed from the system directory via the .DELETE command. Here too, if the name is not in the system directory as either a procedure or user function an error occurs.

In NORM mode, a procedure may be executed with the .PERFORM command. If a non-existent procedure, user function, or system function name is supplied, the command does nothing but display an error. In the event that an uncompiled user function request is encountered during the execution of a procedure, an error occurs and the execution of the procedure is terminated.

5. Programmed Procedures

Procedures may be programmed to perform standard seismic processing tasks. The exact criteria for a procedure to be successfully programmed are described in Section V. The sequence of ISPSE commands required to program, reprogram, or execute a programmed procedure is presented in this section: To program a procedure, the analyst performs the following steps:

- Create a procedure that contains the .PERFORM commands necessary to accomplish the desired task. These statements include system functions, user functions, and INTERUP's.
- Set ISPSE to PRGM mode.
- .PERFORM the procedure.

The message, "PROGRAM OK FOR PROCEDURE <name> " is displayed and the program matrices (Section V) are saved if the user has successfully programmed the procedure. Otherwise, the general supervisor prompts the analyst with four dots and no message is displayed. No procedure containing references to uncompiled user functions can be successfully programmed. An attempt to program such a procedure is aborted when that user function request is

encountered. Procedures are reprogrammed by performing only the last two steps listed above. The only two steps necessary to execute a programmed procedure are:

- Set ISPSE to EXEC mode.
- . PERFORM the procedure.

Programmed procedures may also be executed in NORM mode without disturbing the associated program matrices.

When the . PROCS command is used, an asterisk appearing at the end of a procedure name indicates that the procedure is programmed. Furthermore, if an asterisk is keyed in as the seventh character following the . LIST command, then the procedure's program matrices are displayed. The meaning of these matrices is explained in Section V. If the procedure is not programmed, the command is ignored.

There are several automatic protection features which have been designed into ISPSE. These features are provided to insure that programmed procedure matrices, source code, and all user functions referenced within that procedure, are kept consistent regardless of any editing, deletion, creation, or compilation that is performed. For example, the procedure's program matrices are meaningless if they reference a deleted user function. These protection features are:

- Program matrices are not replaced unless a procedure is successfully reprogrammed.
- Procedures are immediately deprogrammed when they are EDIT'ed.
- Procedures are deprogrammed if a user function which it references is COMPILE'd or DELETE'd.

- All references to user functions in any procedure are replaced by INTERUP's when that user function is deleted.

A list of all procedures that have been deprogrammed is displayed when the user function is COMPILE'd. The message "LINE xx.x IN PROCEDURE name HAS BEEN REPLACED BY AN INTERUP" is displayed for every reference to a user function which is being deleted.

C. BNF DESCRIPTION OF COMMAND LANGUAGE

Up to now, only the actions of each command have been described without regard to the exact format. A formal definition of the ISPSE command language in BNF follows.

1. BNF for External Commands

```
<command>:: = .<system keyword>|
              .<user keyword><user name>|
              .PERFORM<system function>|
              .MODE<state>|
              .HELP .<keyword>|
              .HELP<function>|
              .LIST<user name>*
<system keyword>:: = MODE | PROCS | UFUNC | HELP
<user keyword>:: = CREATE|EDIT|DELETE|LIST|PERFORM|COMPILE
<user name>:: = <symbol>|<user name><symbol>
<symbol>:: = <letter>|<digit>
<letter> :: = A|B|...|Z
<digit> :: = 1|2|...|9|0
<system function>:: = DPSCAN|SELEV|FILTER|GRVEL|SPEED
<state> :: = NORM|EXEC|PRGM
<keyword>:: = <system keyword>|<user keyword>
<function>:: = <system function>| LOGOFF | INTERUP
```

2. BNF for Internal Commands

<create command>:: = .PERFORM <system function>|
.PERFORM INTERUP|.PERFORM
<user name>|#S

<edit command>:: = <edit keyword> <sequence number>|
<create command>|
#P|#N

<edit keyword>:: = #P|#D|#I|#R

<sequence number>:: = <integer>|<integer> .|<integer>.<digit>

<integer>:: = <digit>|<integer> <digit>

D. SEMANTICS AND EXAMPLES OF COMMAND LANGUAGE

The semantics of the ISPSE command language are described by four items, which are:

- <user names>, i.e. procedures and user functions, may contain no more than five characters.
- all <keywords> and <functions> may be abbreviated by the first five characters.
- comments may be appended to every command except .HELP, .MODE, and #P when they are used without their optional parameters.
- commands may be entered in free format with two exceptions. All internal commands beginning with a # must start in column one and the asterisk in the command, .LIST <user name>*, must be the eleventh character after the dot.

The only command that hasn't been discussed thus far is .HELP. This command allows the user to obtain a brief description of ISPSE, any command

in the ISPSE language, or any <function>. If .HELP .HELP is keyed, then all information pertinent to ISPSE is printed.

Examples of all the commands are illustrated in Figure III-1. An ISPSE session is presented from log on to log off with comments on each command. The first command is entered immediately after the initiation of ISPSE according to the directions described in Section II. In the following explanation of each command, the number in parentheses refers to the statement number in the figure.

(1) .HELP - displays a brief description of ISPSE including each command and its format plus the names of the system functions. No comment is allowed on this command because a parameter was not specified.

(2) .HELP .MODE - explains the .MODE command and the available states. A comment has been added to the statement.

(3) .MODE - displays the initial mode of ISPSE, which is NORM. No comment is allowed and it shows that a command doesn't have to start in any specific column.

(4) .UFUNC - displays the current system directory of user function names. The directory is empty.

(5) .COMPILE FUNC1 - enters the new user function name into the system directory. #Q exits from ISPL without compiling. (Section IV).

(6) .UFUN - shows that the new user function, FUNC1, is added to the system directory. Command is abbreviated.

(7) .LIST FUNC1 - displays source lines of user function, FUNC1.

(8) .COMP FUNC1 - allows user to edit, delete, or compile FUNC1. Short form of command is used.

```
  TI-ISPSE LOGGED ON:  9/29/77 AT  0:22:39, ENTER COMMAND OR .HELP

(1).... .HELP

  GENERAL INFORMATION ABOUT ISPSE

  THE FOLLOWING LIST CONTAINS THE COMMANDS AVAILABLE
  TO THE ISPSE USER.  ONLY THE FIRST FIVE CHARACTERS
  OF ANY COMMAND ARE REQUIRED.  ALL NAMES CONSIST OF
  ONE TO FIVE CHARACTERS.  ARGUMENTS IN BRACKETS ARE
  OPTIONAL.  FOR FURTHER INFORMATION CONCERNING EACH
  COMMAND TYPE ".HELP" FOLLOWED BY THE COMMAND.  FOR
  ALL PERTINENT INFORMATION TYPE ".HELP .HELP".

          .UFUN
          .PROC
          .CREATE <NAME>
          .DELETE <NAME>
          .EDIT <NAME>
          .LIST <NAME>
          .COMPILE <NAME>
          .MODE [<STATE>]
          .PERFORM LOGOF
          .PERFORM <NAME>
          .PERFORM INTERUP

  THE FOLLOWING LIST CONTAINS THE NAMES OF THE SYSTEM
  FUNCTIONS AVAILABLE TO THE ISPSE USER.  FOR MORE
  INFORMATION CONCERNING THE SYSTEM FUNCTION, TYPE
  ".HELP" FOLLOWED BY THE SYSTEM FUNCTION NAME OR
  REFER TO "THE DOCUMENTATION OF ISPS" (RINGDAL,
  SHAUB, BLACK, 1975).

       DPSCAN      SELEV      FILTER
       GRVEL       SPEED
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING
COMMAND LANGUAGE
(PAGE 1 OF 12)

```
(2)  .... .HELP .MODE          SYSTEM STARTS IN NORMAL MODE

     .MODE SYSTEM COMMAND

     THE ".MODE" COMMAND MAY BE USED IN TWO WAYS.
     THE ANALYST MAY CHANGE THE MODE BY TYPING IN
     ".MODE", FOLLOWED BY THE DESIRED MODE SETTING.

     IF ONLY ".MODE" IS TYPED, THEN THE ISPSE SYSTEM
     RESPONDS WITH THE CURRENT SETTING OF THE MODE.

     THE ACCEPTABLE MODE SETTINGS ARE: "NORM",
     "PRGM", AND "EXEC", WHICH CORRESPOND TO NORMAL,
     PROGRAM, AND EXECUTE MODE, RESPECTIVELY.

(3)  ....        .MODE
      MODE = NORM
(4)  .... .UFUNC               SYSTEM DIRECTORY OF USER FUNCTION NAMES
      USER FUNCTION LIST
(5)  .... .COMPILE FUNC1       DEFINE NEW USER FUNCTION
     COMPILE ENACTED
      1.0  LABEL ( LAB ( 5 ) )
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING
COMMAND LANGUAGE
(PAGE 2 OF 12)

```
  TI-ISPSE LOGGED ON:  9/29/77 AT  0:22:39, ENTER COMMAND OR .HELP

    2.0  LAB = " I = "
    3.0  I = 1
    4.0  DISPLAY ( LAB, I )
    5.0 #Q                      RETURN WITHOUT COMPILING
(6).... .UFUN                   FUNC1 ADDED TO SYSTEM DIRECTORY
    USER FUNCTION LIST
          FUNC1
(7).... .LIST FUNC1
    1.0  LABEL ( LAB ( 5 ) )
    2.0  LAB = " I = "
    3.0  I = 1
    4.0  DISPLAY ( LAB, I )
(8).... .COMP FUNC1             SHORT FORM OF COMPILE COMMAND
   COMPILE ENACTED
    5.0 #S                      GENERATE EXECUTABLE CODE FILE
    1.0  LABEL ( LAB ( 5 ) )
    2.0  LAB = " I = "
    3.0  I = 1
    4.0  DISPLAY ( LAB, I )
   NO ERRORS FLAGGED IN FUNC1, SUCCESSFUL COMPILATION
(9).... .UFUN                   FUNC1 IS NOW COMPILED (*)
    USER FUNCTION LIST
          FUNC1*
(10)... .PERFORM FUNC1          INDIVIDUAL EXECUTION OF USER FUNCTION
   USER FUNCTION  FUNC1  INITIATED
    I =     1.020000
(11)... .PERF DPSCAN            INDIVIDUAL EXECUTION OF SYSTEM FUNCTION
    SYSTEM FUNCTION DPSCAN INITIATED
    EVENT LIST REQUESTED, (Y,N)?
    .... YES
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING COMMAND LANGUAGE (PAGE 3 OF 12)

```
TI-ISPSE LOGGED ON:  9/29/77 AT  0:22:39, ENTER COMMAND OR .HELP

EV#   EVENT NAME     ORIG DATE/TIME     LAT  LON   MB   STATION CODES

  1 LX-MONOG-231   72  57/23.31. 9     49   105  5.1  1101101111111111
  2 EKAZ 797DEC2   72 345/ 4.26.58     49    78  5.7  0011001000101111
  3 EKAZ 699NOV2   72 307/ 1.26.58     49    78  6.2  0011101000011111
  4 ANM*EKZ*0958   76 161/ 3. 2.48     50    81  5.3  0111111111111111
  5 MAI*EKZ*0958   76 161/ 3. 2.48     50    81  5.3  1111110111111111
  6 NWA*EKZS1062   76 186/ 2.56.50     49    80  6.2  1111111101111111
  7 TAT*NOI*A245   76 245/16.11. 0     35   121  0.0  1111111111101111
  8 ANM*EKZS1062   76 186/ 2.56.50     49    80  6.2  0111111111111111
  9 MAI*EKZ*1368   76 241/ 2.56.43     49    81  6.0  1111110111111111
 10 GUM*EKZL1062   76 186/ 2.56.50     49    80  6.2  1111101111111111
 11 MAI*EKZS1062   76 186/ 2.56.50     49    80  6.2  1111110111111111
 12 MAI*EKZL1062   76 186/ 2.56.50     49    80  6.2  1111110111111111
 13 TAI*EKZL1062   76 186/ 2.56.50     49    80  6.2  1111111111101111
 14 ANM*EKZ*1368   76 241/ 2.56.43     49    81  6.0  0111111111111111
 15 NWA*EKZL1062   76 186/ 2.56.50     49    80  6.2  1111111101111111
 16 NMI*EKA*1368   76 241/ 2.56.43     49    81  6.0  0111111111111111
 17 MAI*SWRS1200   76 211/ 5. 0. 0     48    47  6.7  1111110111111111
 18 MAI*SWRL1200   76 211/ 5. 0. 0     48    47  6.7  1111110111111111
 19 ANM*SWRS1200   76 211/ 5. 0. 0     48    47  6.7  0111111111111111
 20 MAI*KTB*1217   76 213/15.46.46     35    80  3.7  1111110111111111
 21 MAI*KTB*1220   76 214/ 2. 5.29     35    79  4.5  1111110111111111
 22 MAI*NEC*1221   76 214/10.32.21     40   119  3.5  1111110111111111
 23 MAI*BUR*1224   76 214/18. 2.34     27    98  4.2  1111110111111111
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING
COMMAND LANGUAGE
(PAGE 4 OF 12)

```
24 MAI*NEC*1227  76 214/20.53.47  39  118  4.4  1111110111111111
25 MAI*NOI*A215  76 215/ 4. 7. 0  46   59  0.0  1111110111111111
26 MAI*HOK*1228  76 215/ 5. 8.57  44  144  4.4  1111110111111111
27 MAI*NOI*B215  76 215/22.24. 0  46   59  0.0  1111110111111111
28 MAI*KSB*1234  76 216/ 7.50.28  41   77  5.3  1111110111111111
29 MAI*HON*1238  76 216/23.37.48  37  138  4.8  1111110111111111
30 MAI*IBB*1021  76 175/15.38.50  23   88  4.8  1111110111111111
31 MAI*BUR*1240  76 217/ 9.28.17  27   98  4.0  1111110111111111
32 MAI*NOI*A218  76 218/ 3.29. 0  46   59  0.0  1111110111111111
33 MAI*ICG*1242  76 218/10.24.12  28   92  4.4  1111110111111111
34 MAI*NOI*B218  76 218/22.12. 0  46   59  0.0  1111110111111111
35 MAI*NOI*A219  76 219/ 7.37. 0  46   59  0.0  1111110111111111
36 MAI*NOI*B219  76 219/19.23. 0  46   59  0.0  1111110111111111
37 MAI*NOI*A220  76 220/ 4.25. 0  46   59  0.0  1111110111111111
38 MAI*NOI*B220  76 220/19. 6. 0  46   59  0.0  1111110111111111
39 MAI*NEC*1247  76 221/22.41.33  40  119  4.6  1111110111111111
40 MAI*NEC*1249  76 221/23.36. 8  40  121  3.6  1111110111111111
41 MAI*NOI*A222  76 222/12.30. 0  46   59  0.0  1111110111111111
42 MAI*NOI*B222  76 222/21.28. 0  46   59  0.0  1111110111111111
43 MAI*NOI*A223  76 223/ 0.13. 0  46   59  0.0  1111110111111111
44 MAI*NOI*B223  76 223/16.57. 0  46   59  0.0  1111110111111111
45 MAI*IND*1259  76 223/23.32.32  30   79  3.2  1111110111111111
46 MAI*NOI*A224  76 224/ 3.35. 0  46   59  0.0  1111110111111111
47 MAI*NOI*B224  76 224/18.14. 0  46   59  0.0  1111110111111111
48 MAI*NOI*A225  76 225/ 0.45. 0  46   59  0.0  1111110111111111
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING
COMMAND LANGUAGE
(PAGE 5 OF 12)

```
TI-ISPSE LOGGED ON:  9/29/77 AT  0:22:39, ENTER COMMAND OR .HELP

 49 MAI*BIB*1265   76 225/23.26.53   27   97  6.3  1111110111111111
 50 MAI*HOK*1266   76 225/23.28.18   43  144  5.0  1111110111111111
 51 MAI*NOI*A226   76 226/ 2.32. 0   46   59  0.0  1111110111111111
 52 MAI*NOI*B226   76 226/15.49. 0   46   59  0.0  1111110111111111
 53 MAI*ICB*1272   76 227/ 1.23.40   28   96  4.4  1111110111111111
 54 MAI*AFG*1275   76 227/ 8.22.54   32   62  3.7  1111110111111111
 55 MAI*SHR*1276   76 227/ 8.48.35   45   40  2.8  1111110111111111
 56 MAI*NEC*1279   76 227/16. 2.38   40  121  4.2  1111110111111111
 57 MAI*IRA*1280   76 227/22.56.22   28   61  3.5  1111110111111111
 58 MAI*NOI*A228   76 228/ 0. 5. 0   46   59  0.0  1111110111111111
 59 MAI*NOI*B228   76 228/19. 1. 0   46   59  0.0  1111110111111111
 60 MAI*NEC*1289   76 228/22.39.44   39  118  3.9  1111110111111111
 61 MAI*CHI*1296   76 229/14. 6.54   34  104  6.2  1111110111111111
 62 MAI*SZH*1300   76 229/15.26.14   33  105  4.4  1111110111111111
 63 MAI*SZH*1306   76 229/21.30.32   31  105  3.8  1111110111111111
 64 MAI*TUR*1308   76 229/22.41.26   41   39  3.1  1111110111111111
 65 MAI*UZB*1310   76 230/ 2. 3.33   39   64  3.8  1111110111111111
 66 MAI*HON*1314   76 230/17.19. 9   35  138  4.8  1111110111111111
 67 MAI*ION*1317   76 231/ 2.25.37   38   21  2.7  1111110111111111
 68 MAI*NOI*A231   76 231/20.37. 0   46   59  0.0  1111110111111111
 69 MAI*NOI*A232   76 232/ 1.17. 0   46   59  0.0  1111110111111111
 70 MAI*SZH*1321   76 232/12.49.45   33  105  5.2  1111110111111111
 71 MAI*NOI*A233   76 233/ 3.41. 0   46   59  0.0  1111110111111111
 72 MAI*KAS*1325   76 233/19. 5.25   36   74  3.4  1111110111111111
 73 TAT*NOI*A222   76 222/ 1.52. 0   35  121  0.0  1111111111110111

 74 TAT*NOI*B222   76 222/12.58. 0   35  121  0.0  1111111111110111
 75 ANM*NOI*B251   76 251/16.54. 0   45  106  0.0  0111111111111111
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING COMMAND LANGUAGE (PAGE 6 OF 12)

```
   TI-ISPSE LOGGED ON:  9/29/77 AT  0:22:39, ENTER COMMAND OR .HELP


(12).... .PROCS                  SYSTEM DIRECTORY OF PROCEDURES
     USER PROCEDURE LIST
           SPDCM
(13).... .CREATE TASK1           DEFINE NEW PROCEDURE
(14) 1.0 .PERFORM FILTER         INTERNAL CREATE COMMAND
(15) 2.0 .PERFORM SPEED
(16) 3.0 #S                      SAVE PROCEDURE
(17).... .PROCS                  TASK1 ADDED TO SYSTEM DIRECTORY
     USER PROCEDURE LIST
           SPDCM
           TASK1
(18).... .LIST TASK1
     1.0 .PERFORM FILTER         INTERNAL CREATE COMMAND
     2.0 .PERFORM SPEED
(19).... .HELP INTERUP           EXPLANATION OF AN INTERUP

   .PERFORM INTERUP SYSTEM COMMAND

   THE ".PERFORM INTERUP" SYSTEM COMMAND IS AN
   INTERNAL COMMAND TO BE USED DURING THE CREATION OR
   EDITING OF A USER PROCEDURE WHEN PROMPTED BY A
   SEQUENCE NUMBER.  IT INSERTS A PAUSE IN THE
   PROCESSING SEQUENCE, EFFECTIVE AT EXECUTION TIME,
   FROM WHICH POINT THE USER MAY RE-START, CONTINUE,
   OR STOP BY HITTING DESIGNATED PUSH BUTTONS.
   RE-START CAUSES A BRANCH TO THE NEAREST PRECEDING
   .PERFORM INTERUP IN THE SEQUENCE OR TO THE
   BEGINNING OF THE PROCEDURE, CONTINUE RESUMES
   EXECUTION, STOP TERMINATES EXECUTION.
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING COMMAND LANGUAGE (PAGE 7 OF 12)

```
(20)... .EDIT TASK1          OPEN PROCEDURE TASK1 FOR EDITING
(21) 3.0 #F 1.0              PRINT LINE ONE
     1.0 .PERFORM FILTER     INTERNAL CREATE COMMAND
(22) 3.0 #D 1.               DELETE LINE ONE
(23) 3.0 #R 2                REPLACE LINE TWO
(24) 2.0 .PERF SELEV
(25) 3.0 .PERF FUNC1
(26) 4.0 .PERF INTERUP
(27) 5.0 #1 0.5              INSERT A LINE A THE BEGINNING
(28) 0.5 .PERF INTER
(29) 0.6 #P
     0.5 .PERF INTER
     2.0 .PERF SELEV
     3.0 .PERF FUNC1
     4.0 .PERF INTERUP
(30) 5.0 #N                  RESEQUENCE THE PROCEDURE
(31) 5.0 #P
     1.0 .PERF INTER
     2.0 .PERF SELEV
     3.0 .PERF FUNC1
     4.0 .PERF INTERUP
(32) 5.0 #S                  SAVE THE PROCEDURE
(33)... .PERF TASK1          PROCEDURE EXECUTION IN NORM MODE
    USER PROCEDURE TASK1 INITIATED
    PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING
COMMAND LANGUAGE
(PAGE 8 OF 12)

```
   TI-ISPSE LOGGED ON:  9/29/77 AT  0:22:39, ENTER COMMAND OR .HELP

    SYSTEM FUNCTION SELEV INITIATED
   ENTER EVENT SEQUENCE NUMBER FROM LIST
   .... 1
   ENTER STATION NUMBER (1,15)
   .... 6
   ENTER COMPONENT NUMBER (1,3)
   .... 1
   ENTER NUMBER OF COPIES TO BE SAVED : TS =   421   TL =     0 SEC
   .... 1
   ENTER EVENT SEQUENCE NUMBER FROM LIST
   ....
   USER FUNCTION  FUNC1  INITIATED
    I =     1.020000
    PAUSE AT  4.0 : #4 RESTART, #5 CONTINUE, #6 STOP
(34).... .MODE PRGM          SET ISPSE TO PROGRAM MODE
    MODE = PRGM
(35).... .PERF TASK1         TO PROGRAM PROCEDURE
    USER PROCEDURE TASK1 INITIATED
    PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
    SYSTEM FUNCTION SELEV INITIATED
   ENTER EVENT SEQUENCE NUMBER FROM LIST
   .... 1
   ENTER STATION NUMBER (1,15)
   .... 6
   ENTER COMPONENT NUMBER (1,3)
   .... 1
   ENTER NUMBER OF COPIES TO BE SAVED : TS =   421   TL =     0 SEC
   .... 1
   ENTER EVENT SEQUENCE NUMBER FROM LIST
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING
COMMAND LANGUAGE
(PAGE 9 OF 12)

```
     ....
     USER FUNCTION  FUNC1  INITIATED
      I =     1.000000
      PAUSE AT   4.0 : #4 RESTART, #5 CONTINUE, #6 STOP
      PAUSE AT   1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
      AUTOLOOP REQUESTED, ENTER NUMBER OF TIMES TO LOOP
     .... 1
      PROGRAM OK FOR PROCEDURE TASK1
(36) .... .MODE NORM
      MODE = NORM
(37) .... .PROC                    TASK1 IS PROGRAMMED (*)
      USER PROCEDURE LIST
            SPDCM
            TASK1*
(38) .... .LIST TASK1*             DISPLAY PROGRAM MATRICES FOR TASK1
      PROCEDURE MATRIX
         7 -99   6   6   4
      NUMERIC INPUT MATRIX
       0.1000000E+01  0.6000000E+01  0.1000000E+01  0.1000000E+01
          8
      PROCEDURE FUNCTION ADDRESS TABLE
         1   2   0   5
      NUMERIC FUNCTION ADDRESS TABLE
         0   1   0   0
(39) .... .MODE EXEC.              SET ISPSE TO EXEC MODE
      MODE = EXEC
(40) .... .PERF TASK1              TO EXECUTE A PROGRAMMED PROCEDURE
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING COMMAND LANGUAGE (PAGE 10 OF 12)

```
   TI-ISPSE LOGGED ON:  9/29/77 AT  0:22:39, ENTER COMMAND OR .HELP

    USER PROCEDURE TASK1 INITIATED
    PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
    SYSTEM FUNCTION SELEV INITIATED
   ENTER EVENT SEQUENCE NUMBER FROM LIST
   ....   0.1000000E+01
   ENTER STATION NUMBER (1,15)
   ....   0.6000000E+01
   ENTER COMPONENT NUMBER (1,3)
   ....   0.1000000E+01
   ENTER NUMBER OF COPIES TO BE SAVED : TS =   421   TL =     0 SEC
   ....   0.1000000E+01
   ENTER EVENT SEQUENCE NUMBER FROM LIST
   ....
   USER FUNCTION  FUNC1  INITIATED
    I =    1.000000
    PAUSE AT  4.0 : #4 RESTART, #5 CONTINUE, #6 STOP
    PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
    AUTOLOOP COMPLETED, RETURNING TO SUPERVISOR
(41).... .MODE NORM
    MODE = NORM
(42).... .COMP FUNC1          TASK1 IS DEPROGRAMMED
   COMPILE ENACTED
    PROCEDURE TASK1 HAS BEEN DEPROGRAMMED.
(43) 5.0 #*                   ISPSE COMMAND, DELETE, COULD BE USED
   LINE  3.0 IN PROCEDURE TASK1 HAS BEEN REPLACED BY AN INTERUP
(44).... .UFUN                FUNC1 IS REMOVED FROM SYSTEM DIRECTORY
    USER FUNCTION LIST
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING
COMMAND LANGUAGE
(PAGE 11 OF 12)

```
(45) .... .PROC                    TASK1 IS DEPROGRAMMED, NO ASTERISK
     USER PROCEDURE LIST
           SPDCM
           TASK1
(46) .... .LIST TASK1              .PERF FUNC1 CHANGED TO AN INTERUP
     1.0 .PERF INTER
     2.0 .PERF SELEV
     3.0 .PERFORM INTERUP - USER FUNCTION FUNC1 WAS DELETED
     4.0 .PERF INTERUP
(47) .... .DELETE TASK1            REMOVES TASK1 FORM SYSTEM DIRECTORY
(48) .... .PROC                    PROCEDURE NAME TASK1 IS DELETED
     USER PROCEDURE LIST
           SPDCM
(49) .... .PERF LOGOFF             END OF ISPSE TERMINAL SESSION




     TI-ISPS LOGGED OFF :  9/29/77 AT  0:47:41
```

FIGURE III-1

ISPSE TERMINAL SESSION ILLUSTRATING
COMMAND LANGUAGE
(PAGE 12 OF 12)

(9) . UFUN - The asterisk after the name, FUNC1, shows that the user function is compiled.

(10) . PERFORM FUNC1 - executes the user function, FUNC1. Output of the user function is displayed immediately below the command.

(11) . PERF DPSCAN - executes the system function, DPSCAN, with the abbreviated form of . PERFORM.

(12) . PROCS - displays the current system directory of procedure names. One procedure, SPDCM, exists at this time.

(13) . CREATE TASK1 - enters the procedure name, TASK1, into the system directory. Only create commands can be entered as long as the user is prompted by sequence numbers.

(14) . PERFORM FILTE - and

(15) . PERFORM SPEED - inserts instructions to execute the system functions FILTER and SPEED, into the procedure, TASK1. FILTER is shortened to FILTE.

(16) #S - saves the procedure, TASK1, and returns to the command supervisor. The procedure is written to disk.

(17) . PROC - shows that the procedure name, TASK1, has been added to the system directory. . PROCS has been abbreviated.

(18) . LIST TASK1 - lists the source lines associated with the procedure, TASK1.

(19) . HELP INTERUP - briefly describes the <function>, INTERUP.

(20) . EDIT TASK1 - allows the user to edit the procedure, TASK1. Both edit and create commands may be entered.

(21) #P 1.0 - displays line one of the procedure, TASK1. #P must be typed in columns 1 and 2.

(22) #D 1 - deletes line one of procedure, TASK1.

(23) #R 2 - replaces line two of procedure, TASK1, with the next line entered.

(24) .PERF SELEV - is the command that replaces line two.

(25) .PERF FUNC1 - adds the instructions to execute the user function, FUNC1, to the end of procedure, TASK1.

(26) .PERF INTERUP - inserts an INTERUP into the procedure, at the sequence number to the left of the statement.

(27) #I 0.5 - inserts the next create command at the beginning of the procedure.

(28) .PERF INTER - places an INTERUP at the front of TASK1, It is abbreviated.

(29) #P - prints the entire procedure. No comment is allowed.

(30) #N - resequences the procedure.

(31) #P - displays the new sequence numbers and the source lines of TASK1.

(32) #S - saves the procedure and exits from the command processor.

(33) .PERF TASK1 - executes the procedure in NORM mode. If TASK1 was programmed, the program matrices wouldn't be altered.

(34) .MODE PRGM - sets ISPSE to program mode, so that a procedure can be programmed. A comment is allowed on this statement since a parameter was specified.

(35) .PERF TASK1 - programs the procedure, TASK1, by constructing the program matrices.

(36) .MODE NORM - sets ISPSE to normal mode in order to use commands other than .PERFORM <user name>.

(37) .PROC - shows that TASK1 is programmed because an asterisk (*) follows the name.

(38) .LIST TASK1* - displays the program matrices for the programmed procedure, TASK1. The asterisk must immediately proceed the procedure name.

(39) .MODE EXEC - sets ISPSE to execute mode so that a programmed procedure may be .PERFORM'ed.

(40) .PERF TASK1 - the programmed procedure is executed. Procedure flow is determined by the program matrices.

(41) .MODE NORM - resets ISPSE to normal mode.

(42) .COMP FUNC1 - allows the user to edit FUNC1. Since the programmed procedure, TASK1, references this user function, it is deprogrammed.

(43) #W - deletes the user function and replaces the .PERFORM FUNC1 in TASK1 with an INTERUP. The command, .DELETE, could have been used without using .COMPILE.

(44) .UFUN - FUNC has been removed from the user function system directory.

(45) .PROC - shows that TASK1 is deprogrammed because the asterisk has been removed.

(46) .LIST TASK1 - displays the source lines of TASK1. Note that .PERFORM FUNC1 is removed and in its place is a PERFORM INTERUP.

(47) .DELETE TASK1 - removes the procedure from the system directory.

(48) .PROC - TASK1 is no longer on the procedure list.

(49) .PERF LOGOFF - ends the current ISPSE session.

SECTION IV

INTERACTIVE SEISMIC PROGRAMMING LANGUAGE

The Interactive Seismic Programming Language (ISPL) compiler is invoked when the user enters the ISPSE command, .COMPILE<user name>, as discussed in Section III. This command is used to define and modify user functions, which are named sequences of ISPL statements. After an error-free semantic analysis of the user function (Subsection C), it may be .PERFORM'ed individually or as part of a procedure (Section III).

The intent of this section is to present an overview of ISPL, describe the ISPL interactive editor, and explain ISPL statements in detail. Finally, a list of semantic errors is provided and the language is formally defined in BNF.

A. OVERVIEW

ISPL is an algorithmic oriented language, like FORTRAN and ALGOL, which is specifically attuned to seismic processing. ISPL has advantages over these other languages because it:

- provides a quick and easy method of displaying data (both graphically and in tabular form) and performing simple calculations.
- enables the geophysicist to test new algorithms without the time consuming job of task building (e.g., linkage editing).
- relieves the burden of overhead associated with accessing ISPSE seismic data structures, from the programmer.

- interfaces easily with existing ISPSE processing modules which include both system functions and user functions.
- decreases the time lost because of typing errors by performing immediate line-by-line syntax checking.
- reduces debugging effort by providing superior error diagnostics at execution time.
- contains high level vector processing statements for frequently used operations.

Execution time diagnostics include checking:

- the range of vector indices.
- function argument values.
- division by zero in arithmetic expressions.
- input values to the exponentiation operator.
- disk file record numbers.
- the number and types of arguments input to special vector processing statements
- the vector lengths input to special vector processing statements

At this time, only the vector processing statements designed to perform multivariate discrimination (Sax, 1976), are included as part of ISPL. However, new statements may be defined and added to ISPL at any time by the systems programmer, thereby expanding its capabilities. Figure IV-1 lists the capabilities of ISPL. When the analyst has defined a user function, either from the keyboard or from cards via the batch capability of the ISPL compiler, he must perform a semantic error analysis before it may be PERFORM'ed. This is done by means of the ISPL editing command, #S (Subsection B). If

- Access and modify ISPSE data files containing waveforms and parameters.
- Perform scalar and vector processing via arithmetic and logical operators.
- Accomplish multivariate discrimination using high level vector processing statements.
- Display tabular information on the CRT.
- Automatic X-Y plotting of vectors on the CRT.

FIGURE IV-1
ISPL CAPABILITIES

the semantic analysis of the user function reveals any errors, then control is returned to the ISPL interactive editor. Otherwise an executable code file is generated and control is transferred to the ISPSE command supervisor.

After an error-free semantic analysis has been performed and an executable code file has been generated, the user function may be executed individually by means of the .PERFORM <user name>, ISPSE command, or as part of a procedure.

The user should note that an infinite loop may be encountered during the execution of a user function due to programmer error. The user may halt the execution of a user function at anytime by typing in at the operator's console, the MCR command:

MCR >ABO UFUNC

ISPSE control is then transferred to the command supervisor. If the user function is being executed as part of a procedure, the procedure is also terminated.

B. ISPL INTERACTIVE EDITOR

Following the .COMPILE <user name> command the user may enter ISPL statements or edit commands. The edit commands are interpreted by the ISPL interactive editor.

The ISPL interactive editor allows the user to edit user functions, input user functions from the card reader (batch processing), and exit from the ISPL compiler. Edit commands must be entered as the first symbolic line entry following a statement number. They may not be entered as part of a continuation statement. A <statement number> is defined in BNF to be:

<statement number> :: = <integer> | <integer> . | <integer> . <digit>
<integer> :: = <digit> | <integer> <digit>

<digit>:: = 1|2 ... 9|0.

Edit commands must start in column one and their functions are:

#I <statement number>	- insert a statement at the specified position.
#D<statement number>	- delete the statement at the specified position.
#R<statement number>	- replace the statement at the specified position.
#P <statement number>	- print the next four statements starting at the specified position.
#P	- print the entire user function.
#N	- resequence the user function.
#B	- input the user function from the card reader.
#Q	- terminate the ISPL compiler and return to the ISPSE command processor.
#S	- perform semantic error analysis, generate the executable code file, and return to the ISPSE command processor.
#W	- delete the user function and return to the ISPSE command processor.

There are several restrictions which should be noted by the user. The insert command may only be used with a new <statement number>. The command is ignored if an existing <statement number> is specified. The delete command must be used with an existing <statement number> or it is ignored. The replace command may be used with any <statement number>. If the statement doesn't exist, the command operates identical to #I. Batch input of a user function, #B edit command, is processed until an edit command is encountered and then input is received from the CRT keyboard. Generally, the last card in the batch input should be a #S.

C. DETAILED DESCRIPTION OF ISPL

In this subsection, a detailed semantic description of the Interactive Seismic Programming Language (ISPL) is presented. Topics to be addressed are valid entries, constants, variables, declarations, arithmetic expressions, and statements. The following ISPL <statement>'s (see Section IV-D for syntax) are discussed: assignment statement, branch statement, label statement, conditional statement, I/O statements, and numerical operator statements. For each statement the general form followed by an explanation and example are given.

1. Valid Entries

Following the .COMPIL <name> command and/or an edit command (Section IV-B) the user is prompted with a statement number, after which one or more symbolic line entries is expected. A symbolic line entry is an alphanumeric character string containing at most 59 characters, blank characters included, and terminated by a line feed (@ character for cards) or a carriage return. Termination by a line feed indicates continuation and results in prompting with the same statement number after which an additional symbolic line entry is expected. (Exception: A symbolic line entry containing a <label constant> may not be continued (see IV-C-2).) Termination by a carriage return indicates end of entry, that is, no symbolic line entries will follow. Note also that a symbolic line entry is one of two types, a statement line (an ISPL <statement> or portion thereof) or a comment line (first character a 'C').

With these definitions in mind, a valid entry is defined as the concatenation of all symbolic line entries, excluding comment lines, for a statement number into a composite character string which may be classified as an ISPL <statement> (Section IV-d). If the composite string cannot be

so classified the entry is invalid and a syntax error message is immediately displayed. The following rules govern valid entries:

- Let N be the number of statement lines and M be the number of comment lines in a valid entry. Then $1 \leq N \leq 4$ and $0 \leq M \leq 6$.
- An identifier, number, mathematical function name, statement label or keyword (e.g., VECTOR, IF, THEN, JUMP, TO) may not be divided among symbolic line entries (Section IV-D).
- If a syntax error message is displayed following a line feed only the symbolic line entry in question need be re-entered.
- If a syntax error message is displayed following a carriage return, the entry (for that statement number) is invalid and must be re-typed in its entirety. Moreover, the null entry (just a carriage return) is invalid, but generates no error message. The user is simply prompted with the same statement number.
- Only symbolic line entries (statement lines and comment lines) associated with valid entries are recorded and saved under user function < name > by the system.
- A user function may contain at most 99 valid entries (i.e., 99 distinct statement numbers).
- A valid entry contains an ISPL <statement> of which there are two main classes, <declaration statement>'s and <executable statement>'s; <declaration statement>'s inform the system that specified < identifiers > are VECTOR's or LABEL's but they are not executed when the user function is .PERFORM'ed. <executable statement>'s, on the other hand, are executed

and control logical flow, computations, and I/O when the user function is .PERFORM'ed.

Figure IV-2 provides several examples of valid entries.

2. Constants

A constant is a data type which does not change when a user function is .PERFORM'ed. There are two classes of constants as defined in Section IV-D, <number>'s and <label constant>'s.

<number>'s are composed of digits, digits with a decimal point, or digits with a decimal point and an exponent. <number>'s without a decimal point may not exceed 131071 and <number>'s with decimal points may not exceed 10^{75}, be less than 10^{-75}, nor have more than 7 significant figures. Under ISPL, all <number>'s are stored internally as floating point, hence 175, 175., and 17.5E01 are equivalent representations of the quantity one-hundred and seventy five. Although negative <number>'s, as such, do not exist under ISPL, the occurrence of -<number> in an <arithmetic expression> means: change the sign of the constant <number>. Hence the value of an <arithmetic expression> may be negative. For example, x = -5 assigns the value minus five to the variable x.

A <label constant> is an ASCII character string preceded by a ". If the string is also followed by a second " then the <label constant> consists of all characters between the double quotes. Otherwise the <label constant> consists of all characters in that portion of the symbolic line entry following the ". A <label constant> may contain no more than 60 characters, hence a symbolic line entry containing a <label constant> may not be continued.

Note in addition that the number of constants in a single user function is restricted to 124 <number>'s and approximately 900 total characters for <label constant>'s. Figure IV-3 provides several examples of constants.

Valid entries	:	1.0 C THIS IS AN EXECUTABLE STATEMENT	@
		1.0 X=5	
Explanation	:	Assigns the constant 5 to the scalar x	
Valid entry	:	2.0 C THIS IS A DECLARATION STATEMENT	@
		2.0 VECTOR (A(10), B(20))	
Explanation	:	Informs the system that the ⟨identifier⟩ A is a 10 element vector and that the ⟨identifier⟩ B is a 20 element vector	
Valid entry	:	4.0 LABEL (L(20))	@
		4.0 C THIS IS ALSO A DECLARATION STATEMENT	
Explanation	:	Informs the system that ⟨identifier⟩ L is a 20 character label	
Valid entry	:	5.0 L = "A 20 CHARACTER LABEL"	
Explanation	:	Assigns the ⟨label constant⟩ "A 20 CHARACTER LABEL" to the 20 character label L (see 3rd example above and Section IV-d)	

FIGURE IV-2

EXAMPLES OF ENTRIES
(PAGE 1 OF 2)

Valid entry :	7.0 C THIS IS ALSO	@
	7.0 A(1) =	@
	7.0 C AN EXECUTABLE	@
	7.0 B(2)	@
	7.0 C STATEMENT	
Explanation :	Assigns the 2nd element of the vector B	
	to the first element of the vector A	
	(see preceding example and Section IV-D)	
Invalid entry :	9.0 JUMP TO $100	@
	9.0 5	
Explanation :	Entry is invalid, because the <statement label>	
	$1005 is divided between symbolic line entries	
	1 and 2, and will produce a syntax error	
Valid entry :	10.0 JUMP	@
	10.0 TO	@
	10.0 $1005	
Explanation :	This executable statement transfers control to	
	the <statement label> $1005	

FIGURE IV-2

EXAMPLES OF ENTRIES
(PAGE 2 OF 2)

Valid Constants	:	55	
		5555555. E-07	
		5555555.	
		1. E70	
		3. 1415	
		"THIS IS A 60 CHARACTER LABEL	
		"THIS IS A 30 CHARACTER LABEL. . "	
Invalid Constants	:	5555555	Exceeds 131071 and contains no decimal point
		. 12345678	Has too many significant figures
		1. E90	Has too large an exponent
		INVALID LABEL	Not preceded by "

FIGURE IV-3

EXAMPLES OF CONSTANTS

3. Variables and Declarations

A variable is a symbolic name called an <identifier> which represents one or more location(s) in memory and the values which are stored there during user function execution. An <identifier> is composed of a <letter> followed by from one to four <letter>'s or <digit>'s; longer <identifier>'s are truncated to five characters without warning. Also, an <identifier> cannot be a reserved keyword (e.g., IF, THEN, JUMP, TO). ISPL admits three classes of variables: scalars, vectors, and labels. (Refer to Section IV-D)

A scalar is an undeclared <identifier> representing a single memory location and the floating point value stored there. It may not be subscripted. An <identifier> is declared to be a vector by the VECTOR <declaration statement>. A vector represents a specified number N>0 of contiguous memory locations called <vector elements> and the floating point values stored there. <vector elements> are referred to by subscripting the <identifier> with a <number> or a scalar. The <number> or scalar must represent a quantity whose integer portion is at least 1 but not greater than N. Otherwise a fatal error message is displayed during execution. <vector elements> may appear in an <assignment statement>, <arithmetic expression>, or <variable list>. A vector may appear unsubscripted in a <variable list>, in which case an implied reference to all <vector elements> in the order 1 through N is understood.

An <identifier> is declared to be a label by the LABEL <declaration statement>. A label may, at any given moment, hold one <label constant> containing a specified number $0<M\leq 60$ or ASCII characters. The label may, of course, hold many different <label constant>'s during user function execution. A label is always referred to by its <identifier> alone

(no subscripting is permitted] and may appear only in an <assignment statement> or <variable list>.

The following restrictions concerning variables should be noted:

- The values stored in any scalar or vector element have at most 7 significant digits of accuracy and magnitudes bounded by 10^{-75} and 10^{75}. Values outside this range are clipped and cause OTS overflow-underflow messages at the DEC Writer console when they are generated.

- Let K <identifier> 's be declared vectors and L <identifier> 's labels in a user function,
  Let $\left\{N_i\right\}_{i=1}^{K}$ be the set of vector lengths and $\left\{M_i\right\}_{i=1}^{L}$ the set of label sizes,
  Let $\tilde{M}_i$ = integer part [[M_i+4]/5]
  Let A_S = 90-K-L and $A_{VL} = \sum_{i=1}^{K} N_i + \sum_{i=1}^{L} \tilde{M}_i$

  Then: A_S must be non-negative and is the number of user memory elements allocated for scalars (= # scalars available for use). And: A_{VL} must not exceed 1957 and is the number of user memory elements allocated for the declared vectors and labels.

  For example, if one 1957 element vector is declared then no labels can be declared and 89 scalars are available for use. Or more commonly, if seven 256 element vectors are declared then thirteen 60 character labels may be declared, 70 scalars would be available for use, and 9 user memory elements would remain unallocated. Attempts by the user to declare vectors and labels such that A_S and A_{VL} do not conform to

the above restrictions will result in semantic error diagnostics following a #S, to be discussed later in this section.

- Suppose a <label constant> containing $\hat{M}$ characters is stored in a label declared to be of size M (in characters). Then if
  $\hat{M} = M$, the <label constant> is stored as written,
  $\hat{M} > M$, the right-most $\hat{M} - M$ characters of the <label constant> are truncated before storing,
  $\hat{M} < M$, the <label constant> is stored as written followed by $M - \hat{M}$ unpredictable characters.

Figure IV-4 provides examples of variables and declarations.

4. Arithmetic Expressions

An <arithmetic expression> is a rule or formula for computing a numerical value. This value is obtained by executing the indicated binary operators on the <primary>'s of the expression (Section IV-D). A <primary> is a <number>, scalar, <vector element>, parenthesized <arithmetic expression>, or the result of a unary operation on a <primary>. Binary operators are addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (∧). Unary operators are minus (-) and <mathematical function> evaluation. The sequence of operations within one <arithmetic expression> is generally from left-to-right with the following additional rules of precedence determined by the syntax given in Section IV-D.

FIRST	:	parenthesis, unary minus (-), <mathematical function> evaluation
SECOND	:	exponentiation (∧)
THIRD	:	multiplication (*), division (/)
FOURTH	:	addition (+), subtraction (-)

For example, the expression: A + B * C ∧ - &cos [D] is evaluated as follows: The cosine of D is computed, its sign changed, and C is raised to

Valid Scalars:	NREC	**Invalid Scalars:**	1Q	First character a digit
	G1234		Z(J)	Subscript not permitted
	Z		J(5)	Subscript not permitted
			T*Q	Second character not letter or digit
Valid vectors:	VECTOR (X(100), Y(513), I(5))			Declares X to be a 100 element vector, Y to be a 513 element vector, and I to be a 5 element vector
	X(50) I(2)			
	Y(500) I(Z)			
	X(NREC) I(NREC)			
	Y(G1234) I(2.3)			
Invalid vectors:	VECTOR (7X(100), Y(513))			First character a digit
	7X (50)			First character a digit
	Y(Y)			Subscript not a <number> or scalar
	Y(Y(500))			Subscript not a <number> or scalar
	Y(JUMP)			Subscript a keyword, not a <number> or scalar
	Y(I+J)			Subscript not a<number> or scalar
Valid labels:	LABEL (TITLE(50), L(20))			Declares TITLE to be a 50 character label and L to be a 20 character label
	TITLE			
	L			
Invalid labels:	LABEL (60 CHT(60), TITLE(60))			First character a digit
	60CHT			First character a digit
	TITLE(40)			Subscript not permitted
	TITLE(NREC)			Subscript not permitted

FIGURE IV-4

EXAMPLES OF VARIABLES AND DECLARATIONS

that result. The result of that exponentiation is multiplied by B and the result of that multiplication is added to A, completing the evaluation.

Ten <mathematical function>'s are available. They are & SIN (sine), &COS (cosine), &TAN (tangent), &ATN (arctangent), &EXP (exponential: e^x), &LNN (natural log), &LOG (log base 10), &SQR (square root), and &PIM (Πmultiplied by). The argument of a <mathematical function> is enclosed in brackets instead of parenthesis.

There are several restrictions which should be noted concerning <arithmetic expressions>:

- Any formula which cannot be classified as an <arithmetic expression> under the rules of syntax is invalid and generates a syntax error.
- At most five levels of parenthesis are permitted in an <arithmetic expression>. Violation of this rule generates a syntax error.
- Intermediate values incurred during the evaluation of an <arithmetic expression> have at most 7 significant digits of accuracy and a magnitude bounded by 10^{-75} and 10^{75}. Values outside this range are clipped and cause OTS overflow or underflow message at the DEC Writer Console when they are generated.
- The <mathematical function>'s &LNN and &LOG must have a positive argument. A negative or zero argument for either function will generate a fatal execution error. Likewise the <mathematical function> &SQR must have a non-negative argument or a similar error will result.
- The binary operator / is defined as follows:

$$A/B \equiv \begin{cases} A/B, \text{ if } B \neq 0 \\ \text{undefined, otherwise} \end{cases}$$

The binary operator $\wedge$ is defined as follows:

$$A \wedge B \equiv \begin{cases} A^B, & \text{if B is an integer and } (A \neq 0 \text{ or } B \geq 0) \\ e^{B \ell n A}, & \text{if B is not an integer and } A > 0 \\ \text{undefined, otherwise.} \end{cases}$$

For the case where either operator is undefined a fatal execution error will be generated.

Figure IV-5 provides some examples of Arithmetic Expressions.

5. Executable Statements

Thus far, the basic elements of an ISPL user function have been discussed. These include entries, constants, variables, declarations, and arithmetic expressions. Now the <executable statement>'s previously mentioned will be discussed in detail. They include the assignment statement, branch statement and label statement, conditional statement, display statement, input statement, CREAD statement, GET and PUT statements, DREAD and DWRIT statements, plot statement, and numerical operator statements. With a knowledge of these <executable statement>'s and a mathematical formulation of his problem, the seismic user may be able to define one or more ISPL user functions to serve as a solution to the problem.

a. Assignment Statement

As defined syntactically in Section IV-D the <assignment statement> is one of two types, arithmetic assignment or text assignment. The general forms are:

arithmetic assignment : <variable> = <arithmetic expression>
text assignment : <identifier> = <label constant>

Both types permit the user to assign information to symbolic memory locations. Recall from Section IV-C-3 that for the arithmetic assignment statement, the

	VECTOR (X(25), Y(25))	
Valid Arithmetic Expressions:	THETA + &PIM [2] /25	
	R*&COS [THETA]	
	R+&SIN [THETA]	
	X(I)∧Z+Y(I)∧Z	
	&ATN [Y(I)/X(I)]	
	A+-B*-C∧-D	
Invalid Arithmetic Expressions: (Syntax)	A∧(B∧(C∧(D∧(E∧(F∧(G))))))	Too many levels of open parenthesis
	A*((B+C)/(E+F))∧3)	Too many right parenthesis
Invalid Arithmetic Expressions: (Execution)	&LNN [4-5]	Negative argument
	&SQR [1-2]	
	&LOG [&LNN [1]]	Argument is zero
	A/(2-2)	Denominator is zero
	O∧(4-5)	Negative power of zero
	-5∧2.5	Non-integer power of a negative value

FIGURE IV-5

EXAMPLES OF ARITHMETIC EXPRESSIONS

<variable> must be a scalar or vector element. The statement means: replace the value stored in memory location <variable> with the value obtained by evaluating the <arithmetic expression> (Section IV-C-4). Also from Section IV-C-3, the <identifier> in a text assignment statement must be a label. The statement means: replace all ASCII characters stored in memory locations(s) <identifier> with the ASCII character string <label constant> (Section IV-C-2). Figure IV-6 gives some examples of assignment statements.

b. Branch Statement and Label Statement

As defined syntactically in Section IV-D the <branch statement> has the general form:

JUMP TO <statement label>

The counterpart of the <branch statement> is the <label statement> which has general form:

<statement label> CONTINUE

The <statement label> consists of a dollar sign ($) followed by one to four digits (e.g., $1000, $2, $200, etc.). A particular <statement label> may not occur in more than one <label statement> within a given user function.

The purpose of the <branch statement> is to unconditionally transfer control to the corresponding <label statement>, for example, JUMP TO $1000 transfers control to the statement: $1000 CONTINUE. Hence the <label statement> simply acts as a control point in the user function and does not otherwise affect execution.

The following important points should be noted with respect to branch and label statements:

```
                    1.0 VECTOR (X(25), Y(25))
Valid Arithmetic
Assignment Statements:
                    4.0 I=I+1
                    5.0 THETA=THETA+&PIM [2] /25
                    5.5 X(I)=&COS [THETA]
                    7.0 Y(I)=&SIN [THETA]
                    9.0 R=X(I)∧2+Y(I)∧2
                   19.0 ABCD=A+-B*-C∧-D
                   40.0 J=25

                        LABEL (L0(30), L(60))
Valid Text
Assignment Statements:
                    5.0 L0="THIS IS A 30 CHARACTER   LABEL "
                   20.0 L ="THIS IS A 60 CHARACTER   LABEL

Invalid Assignment
Statements:
(generate syntax errors)
                    1.0 A=B<C          B<C not an <arithmetic expression>
                    2.0 A=$1001        $1001 not an <arithmetic expression>
                    5.0 $1001=CONTINUE $1001 not a<variable>or<idendifier>
                   11.0 B<C=D          B<C not a <variable> or <identifier>
                   20.0 M*M=M∧2        M*M not a <variable> or < identifier>
```

FIGURE IV-6

EXAMPLES OF ASSIGNMENT STATEMENTS

- A <branch statement> must have one corresponding <label statement>, however many <branch statements>'s may refer to the same <label statement>.
- At most 25 unique <label statement>'s may exist simultaneously in one user function.

Failure to observe the above points will result in the generation of semantic errors following #S, to be discussed later in this section.

Figure IV-7 provides some examples of branch and label statements.

c. Conditional Statement

Before discussing the general form of the <conditional statement>, a <logical expression> must be defined. Consider two <arithmetic expression>'s joined by one of the logical operators (=,<,>,<=,>=) which are defined below. Further suppose each <arithmetic expression> is evaluated in turn, yielding two distinct values. Then the result is a relationship between those values, which may be TRUE or FALSE, and the construct that has been described is called a <logical expression>. For example, 2 + 3 < 4 + 5 is a <logical expression> which is TRUE, because 5 is less than 9. However A + 1 = A is a FALSE <logical expression>, regardless of the value of A, because 1 is not equal to 0.

With that background, the general form of the <conditional statement> according to Section IV-D is:

IF <logical expression> THEN <simple executable statement>

where the <simple executable statement> is any executable statement with the exception of a <conditional statement> or a <label statement>.

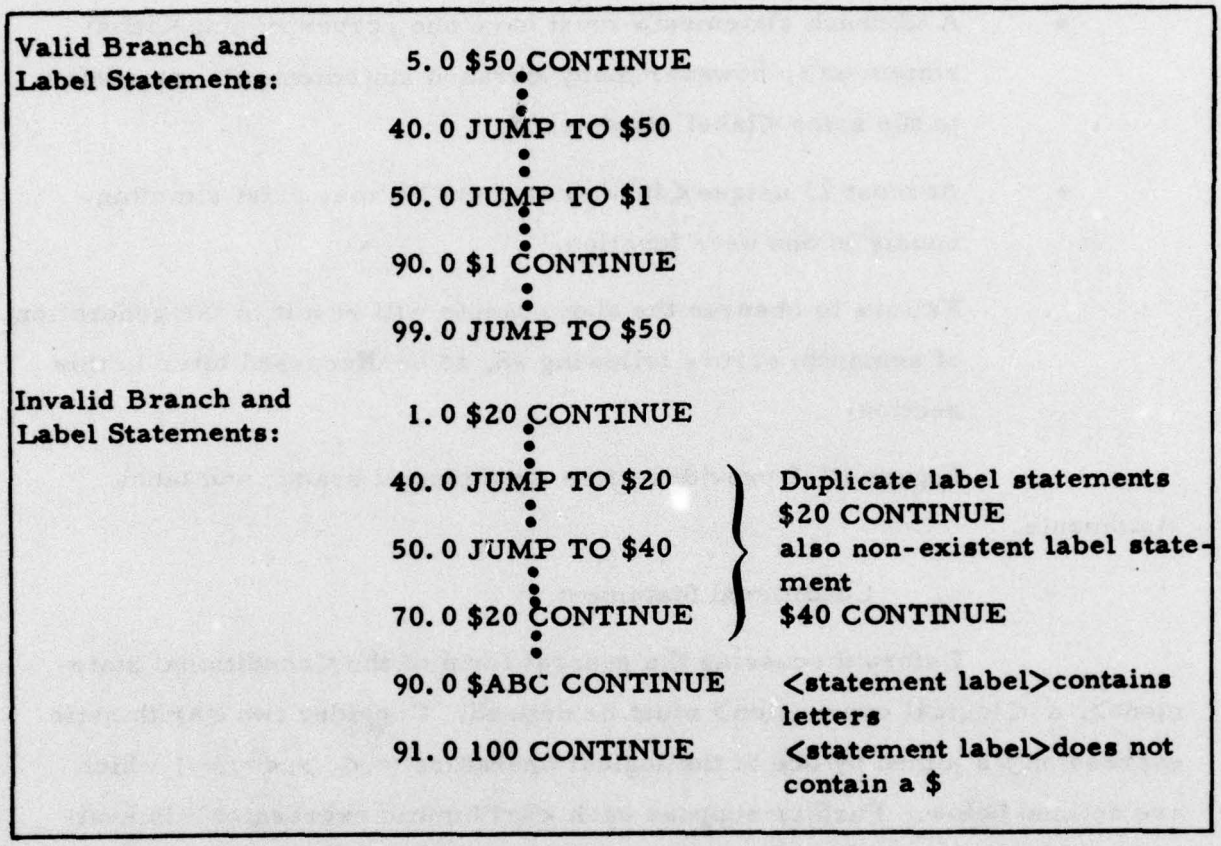


FIGURE IV-7

EXAMPLES OF BRANCH AND LABEL STATEMENTS

On execution of a <conditional statement>, the <logical expression> is first evaluated as being either TRUE or FALSE. If TRUE, then the <simple executable statement> is executed, if FALSE control passes immediately to the next sequential <executable statement> in the user function. For example, the ISPL statement sequence:

```
                .
                .
                .
54.0        I = 0
55.0        IF 2 + 3 < 4 + 5 THEN I=1
56.0        I = I + 1
                .
                .
                .
```

results in the value 2 being assigned to the variable I. On the other hand the sequence:

```
                .
                .
                .
34.0        I = 0
35.0        IF 4 + 5 < 2 + 3 THEN I = 1
36.0        I = I + 1
```

results in the value 1 being assigned to the variable I.

Some additional comments concerning <conditional statement>'s are in order. Specifically,

- The logical operators: =, <, >, <=, >= are read equal to, less than, greater than, less than or equal to, and greater than or equal to respectively, where equal to means mathematically equivalent to within five units in the seventh decimal place. (Recall that values under ISPL are accurate to at most 7 decimal places).

- A loop in which a counter I varies from 1 to N is easily constructed via the <conditional statement> in the following way (assume N has been initialized):

```
                .
                .
10.0            I = 0
11.0            $100 CONTINUE
12.0            I = I + 1
                .
                .
                .
40.             IF I<= N - 1 THEN JUMP TO $100.
```

Figure IV-8 provides some examples of conditional staments.

d. DISPLAY Statement

The DISPLAY statement is the first of the <I/0 statement>'s to be discussed in this section. Its general form from Section IV-D is:

DISPLAY (<variable list>)

where the <variable list> is a sequence of <variable>'s separated by commas. The list may include scalars, vectors, vector elements, and labels. On execution, display of the information begins on the next 65 character line on the CRT and continues for the number of lines required. Each scalar or vector element in the list represents one numerical value that is displayed in a 15 character format, automatically determined by the system and depending on the value's magnitude. Occurrence of a vector in the list implies display of all the vector's elements, hence the statements: VECTOR (A(20)), DISPLAY (A), generates 5 lines of numerical values on the CRT, 4 values per line. Occurrence of an M character label in the list implies display of the M characters currently held by the label. Therefore the statements: LABEL (L(15)), L = "THE TOTAL COST=", DISPLAY (L, COST), displays the <label constant>: "THE TOTAL COST=" followed by the numerical value represented by the scalar COST, on the CRT. Note that a full CRT screen (30 lines) is automatically scrolled to the line printer unless blue pushbutton #6 beneath the screen is on, preventing screen scroll (and user

```
                    VECTOR (X(25), Y(25))

Valid Conditional
Statements:         IF &COS [THETA] =&SIN [THETA] THEN THETA=&PIM[.25]
                    IF X(I)∧2+Y(I)∧2=1 THEN TAN=Y(I)/X(I)
                    IF K<7 THEN JUMP TO $100
                    IF &LNN [&COS [THETA ]]>-.5 THEN DISPLAY* (THETA)
                    IF 0=A*Z∧2+B*Z+C THEN JUMP TO $40
                    *Display statement to be discussed later in this section.

Invalid Conditional
Statements:         IF A<B JUMP TO $500                  missing keyword THEN
                    IF I>J>=K THEN RANGE=K-I             invalid <logical expression>
                                                         I>J>=K
                    IF (I+4>K+7) THEN L=I∧2+K∧2          parenthesis not permitted
                    IF C=D THEN $75 CONTINUE             $75 CONTINUE not a
                                                         <simple executable statement>
                    IF K<7 THEN IF K>0 THEN L=K∧Z        IF K>0 THEN L=K∧Z not a
                                                         <simple executable statement>
```

FIGURE IV-8

EXAMPLES OF CONDITIONAL STATEMENTS

function execution) until that button is turned off. Figure IV-9 provides some examples of DISPLAY statements.

e. INPUT and CREAD Statements

The INPUT and CREAD statements are two of the four ISPL <I/O statement>'s via which data may be input to the ISPSE system. Their general forms from Section IV-D are:

INPUT (<variable list>)
CREAD (<variable list>)

where the <variable list>, may include scalars, vectors, vector elements, and labels. On execution of the INPUT statement, the user is prompted with four dots (....) for each scalar or vector element in the list. In each case the system expects a valid <number> to be entered at the CRT keyboard. Entry of an invalid number generates an error message demanding that the number be re-entered. Occurrence of a vector in the list implies input of all the vector's elements. Occurrence of an M character label in the list also prompts the user with four dots, however in this case the system expects a <character string> of M≤60 characters to be entered.

Execution of the CREAD statement is identical to that of the INPUT statement, except that the user is not prompted with four dots. Instead information is read from the card reader, one <number> or <character string> per card, appearing free format in columns 1 through 60. Detection of an invalid number generates an error message and requires correction and re-reading of the bad card, and a carriage return at the CRT keyboard to continue execution.

Figure IV-10 gives some examples of INPUT and CREAD statements.

ISPL Statements Defining a User Function

```
1.0 VECTOR (X(4))
2.0 LABEL (L0(20), L1(55))
3.0 X(1)=1
4.0 X(2)=2
5.0 X(3)=3
6.0 X(4)=4
7.0 DISPLAY (X)
8.0 L0="THE VALUE OF X(1) IS"
9.0 L1="L1 IS A 55 CHARACTER LABEL; ITS DISPLAY
    TAKES ONE LINE*
10.0 DISPLAY (L0,X(1))
11.0 DISPLAY (L1)
12.0 DISPLAY (L0, X(1), X)
13.0 DISPLAY (L0, L0, L0, L0)
```

Output from User Function Execution:

```
   1.000000        2.000000        3.000000        4.000000
THE VALUE OF X(1) IS    1.000000
L1 IS A 55 CHARACTER LABEL; ITS DISPLAY TAKES ONE LINE
THE VALUE OF X(1) IS    1.000000        1.000000        2.000000
   3.000000        4.000000
THE VALUE OF X(1) ISTHE VALUE OF X(1) ISTHE VALUE OF X(1) ISTHE V
ALUE OF X(1) IS
```

* Should be written on one line.

FIGURE IV-9

EXAMPLES OF DISPLAY STATEMENTS

ISPL Statements

Defining a User Function:

```
1.0 VECTOR (X(4))
2.0 LABEL (L0(20), L1(55))
3.0 INPUT (X)
4.0 INPUT (L1)
5.0 CREAD (L0, X(1))
```

User Function Execution:

Entered by User at the VT Keyboard

```
.... 1
.... 2
.... 3
.... 4
.... L1 IS A 55 CHARACTER LABEL; ITS
     DISPLAY TAKES ONE LINE *
```

Punched on cards:

```
Card 1:  THE VALUE OF X(1) IS
Card 2:  1
```

* Should be written on one line.

FIGURE IV-10

EXAMPLES OF INPUT AND CREAD STATEMENTS

f. GET and PUT Statements

GET and PUT are special purpose <I/O statement>'s that enable the ISPSE user to access the temporary direct access data file resident on the fixed head disk (RF). A detailed discussion of the file may be found in Appendix B of "Documentation of the Interactive Seismic Processing System (ISPS)"(Ringdal, et al., 1975). Briefly this file consists of 2K logical records of identical format where $1 \leq K \leq 8$. Each record pair corresponds to one wave-form segment of at most 512 data points (memory elements). The first record of the pair contains raw waveform data, the second record contains processed data and is displayed as one trace on the CRT by system functions (SELEV, FILTER, SPEED etc.). (Note: In some cases a trace may contain multiple waveform s egments. However, that situation will not be addressed in this discussion.) Each logical record also contains an annotation trailer which specifies type of data, source information, and processing status.

With these facts in mind Section IV-D gives the general forms of GET and PUT as:

GET [<idn_int>] (<variable list>)
PUT [<idn_int>] (<variable list>)

where <idn_int> is any scalar or number whose value represents the RF file record number R ($1 \leq R \leq 16$), and the <variable list> must contain exactly two variables, the first being a 548 element vector and the second a 55 character label.

Now suppose SELEV and FILTER are utilized to select and display $K \leq 8$ traces on the CRT, and let $\tilde{K}$ be the trace number, beginning with $\tilde{K} = 1$ at the top of the screen. This means that K waveform segments will exist on the RF file, occupying R = 1, 2, .., 2K records. Then execution of the statement: GET[R] (X, EVSTA) with $R = 2\tilde{K}$ will return the data and annotation, associated with displayed trace $\tilde{K}$, in the vector X,

whose elements and their contents are described in Figure IV-11. Also, an alphanumeric event and station designation will be returned in the label EVSTA.

The inverse operation occurs on execution of the PUT[R] (X, EVSTA) statement. Here however, the user must ensure that X and EVSTA are properly initialized before the PUT statement is executed to overwrite trace $\tilde{K} = R/2$. Otherwise that trace will be incorrectly referenced by subsequently PERFORM'ed system functions (e.g., FILTER)

The following important points should be noted concerning GET and PUT:

- If the <variable list> for GET and PUT does not contain the precise variables described above, an execution error will be generated and no I/0 will be performed. An out-of-range record number will produce a similar error.
- Although the RF file record number R is usually even for GET and PUT statements, as described above, it may be odd for applications involving processing of data via user functions, before or between system functions. In that case, care must be taken that the raw waveform data held by the odd records may be conveniently restored, once overwritten. For example, it would probably not be convenient to re-PERFORM SELEV to restore the raw data.

Figure IV-12 provides an example of GET and PUT utilization.

g. DREAD and DWRIT Statements

DREAD and DWRIT are <I/0 statement>'s that enable the ISPSE user to automatically perform input and output to named direct access disk files, without being concerned with the associated systems bookkeeping

Element of X	Description	Sample Value
(1)-(512)	HOLDS THE DATA TRACE	
(513)	SEGMENT NUMBER:	1.000000
(514)	TOT # OF SEGS FOR TRACE:	4.000000
(515)	TOT # OF PTS FOR TRACE:	2048.000
(517)	MAX ABS DATA VALUE:	778.0598
(518)	SAMPLE RATE (SECS):	2.000000
(519)	STATION NUMBER (1-15):	3.000000
(520)	COMPONENT NUMBER (1-3):	1.000000
(522)	ORG YEAR:	72.00000
(523)	ORG DAY:	57.00000
(524)	ORG TIME (SECS IN DAY):	84669.00
(525)	DATA TS (SECS REL ORG):	421.0000
(526)	P-WV TS (SECS REL ORG):	557.0000
(527)	S-WV TS (SECS REL ORG):	1005.000
(528)	LR-WV TS (SECS REL ORG):	1359.000
(529)	LQ-WV TS (SECS REL ORG):	1511.000
(530)	LR LENGTH (SECS):	867.0000
(531)	SOURCE LATITUDE S(-) :	49.00000
(532)	SOURCE LONGITUDE W(-) :	105.0000
(533)	SOURCE DEPTH (KM):	0.0000000
(534)	PDE MB:	0.0000000
(535)	PDE MS:	0.0000000
(536)	OTHER MB:	5.100000
(537)	OTHER MS:	0.0000000
(538)	STATION LATITUDE S(-) :	64.89900
(539)	STATION LONGITUDE W(-):	-148.0050
(540)	STA AZIMUTH /EV (DEGS):	-51.79600
(541)	STA EPICNTRL DIS (DEGS):	52.98000
(544)	LOW BANDPASS FREQ (HZ):	7.0000000
(545)	HIGH BANDPASS FREQ (HZ):	10.00000
(546)	CHIRP CENTER FREQ (HZ):	0.4000000E-01
(547)	CHIRP LENGTH (SECS):	0.0000000
(548)	SCALE (NM/INCH) ACTUAL:	0.0000000

FIGURE IV-11

DESCRIPTION OF VECTOR X FOLLOWING GET [R] (X, EVSTA)

ISPL Statements Defining a User Function

```
1.0 C THIS USER FUNCTION RECTIFIES DATA TRACE #1   @
1.0 VECTOR (X(548))
2.0 LABEL (EVSTA(55))
3.0 GET[2](X,EVSTA)
4.0 C IF TRACE HAS > 1 WAVEFORM SEGMENT            @
4.0 C THEN ONLY RECTIFY THE FIRST SEGMENT          @
4.0 NPTS = X(515)
5.0 IF X(514)>=2 THEN NPTS=512
6.0 I=0
7.0 $10 CONTINUE
8.0 I=I+1
9.0 X(I)=&ABS[X(I)]
10.0 IF I<=NPTS-1 THEN JUMP TO $10
11.0 PUT[2] (X,EVSTA)
12.0 DISPLAY (EVSTA)
13.0 DISPLAY (X)
```

FIGURE IV-12

USE OF GET AND PUT

problem. Among other applications DREAD and DWRIT may be used to communicate parameters and data between user functions. According to Section IV-D their general forms are:

DREAD [<identifier>,<idn_int>] (<variable list>)
DWRIT [<identifier>,<idn_int>] (<variable list>)

where <identifier> is any five character filename label, <idn_int> is any scalar or number whose value represents a record number from 1 to 100, and the <variable list> may include scalars, vectors, vector elements, and labels. Each scalar or vector element in the list represents one memory element and the value stored there. Each M character label represents $\tilde{M}$ memory elements and the characters stored there where $\tilde{M}$ = integer part $[(M + 4)/5]$. Also, each occurrence of a vector in the list implies occurrence of all the vector's elements. On execution of DWRIT the contents of memory elements associated with variables in the <variable list> are written across the specified record within the specified file in the order in which the variables appear in the list. On execution of DREAD, the inverse operation occurs as elements are read from the specified record within the specified file and their contents stored sequentially in the variables appearing in the list.

The following important points concerning DREAD and DWRIT should be noted:

- As previously stated, the <identifier> in the DREAD and DWRIT syntax must be a five character label which contains a character string to be interpreted as a filename during execution. If the label is not initialized an error is flagged and no I/O is performed. A similiar error occurs if the record number specified via <idn_int> is outside the range 1-100.
- Writing information to a record of a file does not affect information in any other record. Moreover, at most 125 memory

elements may be read or written to any record. If the <variable list> for a DREAD or DWRIT exceeds 125 memory elements it is automatically truncated and a warning message is provided.

- In order for any record within a file to be referenced by DREAD, it must have been initially written by DWRIT, otherwise results are unpredictable. The only exception to this is the file SPEED PCM created by the system function SPEED and discussed in Appendix A. Of course, user function A may DWRIT a file to be DREAD by other user functions B, C, D, etc., as well as itself.

- Suppose a <variable list> containing W memory elements is written to a given record within a file and then a <variable list> containing R elements is read from that file and record. Then care must be taken that $R \leq W \leq 125$. Otherwise the last $(R - W) > 0$ elements of the R element <variable list> will be meaningless.

- If a disk file is created for scratch I/0 only, the user may not need to document its contents. However, it is recommended that all permanent files built via DREAD and DWRIT be documented as to their name and contants. This documentation may be maintained in a separate file.

Figure IV-13 illustrates the use of DREAD and DWRIT.

h. PLOT, PLOTP, and PLOTX Statements

The ISPSE user may graphically display vectors by means of the <I/0 statement>'s PLOT, PLOTP, and PLOTX. The CRT display resulting from the use of each operator consists of:

ISPL Statements Defining a User Function

```
1.0 C THIS FUNCTION READS SOME SAMPLE EVENT                    @
1.0 C PARAMETERS, INCLUDING THE EVENT NUMBER N                 @
1.0 C 1≤N≤100, FROM CARDS AND WRITES THEM TO A                 @
1.0 C DISK FILE CALLED "EVPAR". CARDS ARE READ UNTIL           @
1.0 C AN OUT OF RANGE EVENT NUMBER IS DETECTED                 @
1.0   LABEL (FNAME(5), EVNAM(15))
2.0   FNAME="EVPAR"
3.0   $10 CONTINUE
4.0   CREAD (N, EVNAM, LAT, LON, MB)
5.0   IF N<=0 THEN JUMP TO $100
6.0   IF N>=101 THEN JUMP TO $100
7.0   DWRIT [FNAME, N] (EVNAM, LAT, LON, MB)
8.0   JUMP TO $10
9.0   $100 CONTINUE
```

FIGURE IV-13

FUNCTIONS UTILIZING DREAD AND DWRIT
(PAGE 1 OF 2)

ISPL Statements Defining a User Function

```
1.0 C THIS FUNCTION READS EVENT NUMBERS FROM                          @
1.0 C CARDS.  FOR EACH EVENT NUMBER, ASSOCIATED                       @
1.0 C EVENT PARAMETERS ARE RETRIEVED FROM THE DISK                    @
1.0 C FILE "EVPAR" AND DISPLAYED ON THE CRT. CARDS                    @
1.0 C ARE READ UNTIL AN OUT OF RANGE EVENT NUMBER IS DETECTED         @
1.0   LABEL (FNAME(5), EVNAM(15), L(60))
2.0   L= "  EVENT NAME   EVENT LATITUDE EVENT LONGITUDE MAG(MB)
3.0   DISPLAY (L)
4.0   L="
5.0   DISPLAY (L)
6.0   FNAME="EVPAR"
7.0   $10 CONTINUE
8.0   CREAD (N)
9.0   IF N<=0 THEN JUMP TO $100
10.0  IF N>=101 THEN JUMP TO $100
11.0  DREAD [FNAME, N] (EVNAM, LAT, LON, MB)
12.0  DISPLAY (EVNAM, LAT, LON, MB)
13.0  JUMP TO $10
14.0  $100 CONTINUE
```

FIGURE IV-13

FUNCTIONS UTILIZING DREAD AND DWRIT
(PAGE 2 OF 2)

- PLOT - a line plot of ordered pairs of X and Y values.
- PLOTP - a point plot of ordered pairs of X and Y values.
- PLOTX - a line plot of Y values.

The X axis is displayed horizontally and the Y axis, vertically. These statements provide three significant advantages over similar FORTRAN call statements, in that they:

- allow one or more optional operands to be supplied by the user.
- perform automatic execution time checking of the number of arguments, the operand types (i.e. scalar, vector, or label), and the vector lengths.
- allow the user to interactively modify the plot scale factor, and to generate hard copies.

The BNF generalized format for these statements is the operator name followed by (<variable list>). Specifically, the format of each statement is:

```
PLOT    (X, Y, N[,XLAB[,YLAB]])
PLOTP   (X, Y, N[,XLAB[,YLAB[,INT]]])
PLOTX   (Y, N[,XLAB])
```

where the optional arguments are enclosed in brackets, [ ]. The arguments are described below:

X - a vector containing the X values.
Y - a vector containing the Y values.
N - a scalar whose value is the number of points to be plotted.
XLAB - a label containing a character string that is displayed at the bottom center of the graph.

YLAB - a label containing a character string that is displayed at the left center of the graph.

INT - a vector whose elements are indices of X and Y, and determine the points at which the intensity is increased.

The length of the INT vector determines the initial intensity of the plot. If INT is not specified in the <variable list>, then all points are plotted at the highest intensity. Each additional element in INT reduces the initial intensity of the plot. The values stored in INT specify the points at which the intensity is increased, for that point and all subsequent points. No more than three intensity changes are allowed. Therefore INT should not be longer than three elements, and the values stored in INT should be strictly increasing. Figure IV-14 clarifies the use of the INT argument.

Warning messages are displayed whenever too many arguments are specified, the value of N is greater than 512, or the length of either the X or Y vector is less than N (512 if N is greater than 512). In the latter two cases, the number of points that are plotted is the minimum of 512, the value of N, the length of the X vector, and the length of the Y vector. If no other errors are detected, a graph is constructed. No plot is displayed if too few arguments are supplied or if the argument types are incorrect. Regardless of any errors, execution of the user function continues after the plot statement, because data cannot be modified by plotting, and therefore further calculations involving these vectors are still meaningful.

When a plot operator is encountered in normal mode, execution of the user function is halted. At this time, a keyword menu is displayed and the user may rescale the plot, generate a hard copy, or exit from the plot statement, and continue executing the user function.

Finally, when a plot operator, within a user function, is contained in a programmed procedure, and this procedure is PERFORM'ed in

```
1.0 C THIS USER FUNCTION PLOTS ONE PERIOD OF A SINE WAVE                @
1.0 C USING SEVERAL FORMS OF THE PLOT STATEMENTS                        @
1.0   VECTOR (X(100), Y(100), INT(3))
2.0    LABEL (XLAB(15), YLAB(10))
3.0 C GENERATE THE SINE WAVE                                            @
3.0    N = 100
4.0    I = 0
5.0    $100 CONTINUE
6.0    I = I+1
7.0    X(I) = I-1
8.0    Y(I) = &SIN[&PIM[X(I)/50]]
9.0    IF I<N THEN JUMP TO $100
10.0 C THE NEXT THREE STATEMENTS PRODUCE IDENTICAL DISPLAYS             @
10.0 C EXCEPT THAT PLOTP IS A POINT PLOT                                @
10.0   PLOT (X, Y, N)
11.0   PLOT P (X, Y, N)
```

FIGURE IV-14

EXAMPLES OF PLOT, PLOTP, AND PLOTX
(PAGE 1 OF 2)

```
12.0   PLOTX (Y, N)
13.0 C ONLY ONE LABEL IS DISPLAYED ON THIS GRAPH                    @
13.0   XLAB = " SINE FUNCTION "
14.0   PLOT (X, Y, N, XLAB)
15.0 C POINTS 1-25 ARE THE DIMMEST                                  @
15.0 C        26-50 ARE AT THE SECOND INTENSITY                     @
15.0 C        51-75 ARE AT THE THIRD INTENSITY                      @
15.0 C        76-100 ARE THE BRIGHTEST                              @
15.0   INT(1) = 26
16.0   INT(2) = 51
17.0   INT(3) = 76
18.0   XLAB = "X TIMES 50 / PI"
19.0   YLAB = "SIN ( X ) "
20.0   PLOTP (X, Y, N, XLAB, YLAB, INT)
```

FIGURE IV-14

EXAMPLES OF PLOT, PLOTP, AND PLOTX
(PAGE 2 OF 2)

execute mode, the status of the blue pushbutton number five is of paramount importance. If the pushbutton light is on, then execution of the procedure is interrupted and the keyword menu, which allows rescaling and hardcopy generation, is displayed. After the ISPSE user exits from the plot operator, execution of the programmed procedure continues. If the pushbutton light is off, a hard copy of the display is automatically created, and procedure flow is uninterrupted.

The user function in Figure IV-14 illustrates typical usage of the plot <I/O statement>'s.

i. QSTAT, XSTAT, and DSTAT Statements

These <numerical operator>'s are especially designed to perform multivariate event discrimination based on mean detectabilities as described by Sax (1976). A detailed example of the entire solution to this problem is presented in Appendix A. Equivalent operators could be programmed using ISPL, but by including them as high level vector processing statements in ISPL, a substantial reduction in execution time is achieved.

These operators access a disk file containing event discriminant values that are calculated via the system function, SPEED. Event sequence numbers from the DPSCAN event list are input from the card reader, one event number per card. These operators, like the plot statements, perform automatic execution time checking of the number of arguments, the operand types, and the vector lengths. DSTAT also allows specification of two optional arguments.

The BNF description of the <numerical operator>'s is found in Section IV-D. The specific form for each statement is:

QSTAT (NQ, ND, EQ, SDQ)
XSTAT (NX, ND, EQ, SDQ, EDX)

DSTAT (NE, ND, EQ, SDQ, EDX1, EP1 ,[EDX2, EP2])

where the optional arguments are enclosed in brackets,[ ].

QSTAT calculates mean and standard deviation vectors, EQ and SDQ, of event discriminants obtained from the disk. The mathematical equations for these functions are:

$$EQ_j = \frac{1}{NQ} \sum_{i=1}^{NQ} D_{ij} \qquad j = 1, ND$$

$$SDQ_j = \frac{1}{NQ - 1} \sqrt{\sum_{i=1}^{NQ} D_{ij}^2} \qquad j = 1, ND$$

where ND is the number of discriminants per event, D is a matrix of discriminant values, EQ is the mean value vector and SDQ is the standard deviation vector of the discriminants over a sample population of NQ earthquakes.

The XSTAT operator calculates the mean detectability vector for a set of presumed explosions from a given region. Precisely, the operator is described by:

$$EDX_j = \frac{1}{NX \bullet SDQ_j} \left[\sum_{i=1}^{NX} D_{ij} - EQ_j \right] \qquad j = 1, ND$$

where the mean value vector, EQ, and the standard deviation vector, SDQ, are the outputs from the QSTAT operator. NX is the number of presumed explosions in the specified region. As before, ND and D are the number of discriminants and the matrix of actual discriminant values. The output, EDX, is the mean detectability vector for that region.

New events are then projected onto the mean detectability vector of a region for classification by the analyst. This is done with the DSTAT statement. Two projections can be calculated at the same time by

utilizing the optional arguments, EDX2 and EP2. Of course, in this case the XSTAT operator must be used twice to form the mean detectability vectors for both regions. EDX1 and EDX2 are the mean detectability vectors for region one and two, respectively. EP1 and EP2 are the projections onto these vectors. The equation describing the DSTAT operator is:

$$EP_i = \frac{1}{\sum_{j=1}^{ND} EDX_j^2} \left[\sum_{j=1}^{ND} \frac{EDX_j (D_{ij} - EQ_j)}{SDQ_j} \right] \quad i = 1, NE$$

where EP is the projection vector and NE is the total number of events, including the sample earthquake population, the presumed explosion population, and the events to be classified. The other parameters have all been described above.

An example of a user function that utilizes these operators is discussed in Appendix A. The event sequence numbers input from cards must be in the range, 1 - 100, or an error message is displayed. Undefined results are produced if the discriminants for a particular event are not calculated via SPEED prior to using QSTAT, XSTAT, or DSTAT. Unlike the plot statements, any error in the argument specification or event sequence numbers causes an immediate termination of the user function.

6. Semantic Error Analysis

On receiving a #S edit command (Section IV-B), the system makes a second pass over the ISPL statements that have been accepted as syntactically valid for the user function currently being .COMPILED. Besides generating and saving a matrix of interpretable instructions for the user function, #S also performs an analysis to ensure that no semantic ambiguities exist in the program as a whole. If any errors are flagged the user must correct them before the system will save the function and mark it .COMPILED. Some semantic considerations are discussed in the points which follow:

- A variable may be declared only once in a declaration statement. Re-occurrence of a variable in the same or another declaration statement is ignored and only the initial declaration applies. Note: this is the only semantic error which does not prevent the user function from being successfully .COMPILE'ed.

- Vector and label lengths must be at least 1 element and 1 character respectively. Zero (0) length allocations are not accepted and produce a semantic error message.

- All declaration statements must appear at the beginning of a user function and before any executable statements. Failure to observe this ordering will cause semantic errors to be flagged.

- Recall from Section IV-C-3 that $0 \leq A_S \leq 90$ and $0 \leq A_{VL} \leq 1957$ are requirements for acceptable declaration of vector and labels, where A_S is the number of scalars available for use and A_{VL} is the number of user memory elements allocated for declared vectors and labels. A semantic error diagnostic will be generated if these allocation requirements are not satisfied.

- Duplicate or non-existent label statements, or more than 25 unique label statements in a user function will result in semantic error messages.

- Utilization of more than 124 unique numbers or more than ~900 total characters for <label constant>'s will produce a semantic error message.

- The occurrence of a label in an arithmetic expression or arithmetic assignment statement, the occurrence of a scalar,

vector, or vector element in a text assignment statement, the occurrence of a subscripted label or scalar in any executable statement, the occurrence of a vector element whose subscript is a vector or label in any statement, or the occurrence of an unsubscripted vector in an arithmetic expression or arithmetic assignment statement will produce one or more semantic error messages following each violation.

- A user function must contain at least one executable statement, otherwise it will not be successfully .COMPILE'd.
- The bracketed item in a GET or PUT statement must be a scalar or number, otherwise semantic error messages will be produced.
- The first of two bracketed items in a DREAD or DWRIT statement must be a 5 character label and the second must be a scalar or number; otherwise semantic error messages will be produced.
- In general, variables should be initialized via assignment or I/O statements before they are used in other statements, otherwise semantic errors may be generated.

D. BNF DESCRIPTION OF ISPL

The following is a BNF description of ISPL syntax. No semantic considerations (e.g., magnitude limitations for numbers, vector lengths, variable list lengths, arithmetic expression extents, etc.) are addressed in this sub-section. The reader should refer to Section IV-C for this information. Note also that embedded blanks in ISPL syntax are ignored.

<blank>:: = the ASCII character (space)

<digit>:: = 0|1|2|3|...|9

<letter>:: = A|B|C|D|...|Z

<letter_digit>:: = <letter>|<digit>

<identifier>:: = <letter>|<letter><letter_digit>$_1^4$

<integer>:: = <digit>$_1^6$

<idn_int>:: = <identifier>|<integer>

<real>:: = <digit>$_1^7$.|.<digit>$_1^7$|<digit>$_1^K$.<digit>$_{K+1}^7$

<exponent>:: = E<digit>$_1^2$|E+<digit>$_1^2$|E-<digit>$_1^2$

<number>:: = <integer>|<real>|<real> <exponent>

<declaration list>:: = <identifier>(<integer>)|<identifier>(<integer>),
<declaration list>

<declaration statement>:: = LABEL(<declaration list>)| VECTOR(<declaration
list>)

<vector element>:: = <identifier>(<number>)|<identifier>(<identifier>)

<variable>:: = <identifier>|<vector element>

<variable list>:: = <variable>|<variable>, <variable list>

<mathematical function>:: = &SIN|&COS|&TAN|&ATN|&EXP|&LNN|&LOG|
&SQR|&ABS|&PIM

<add_subtract>:: = +|-

<multiply_divide>:: = *|/

<primary>:: = <number>|-<number>|<variable>|-<variable>| <mathematical function>[<arithmetic expression>]|-<mathematical function>[<arithmetic expression>]|(<arithmetic expression>)|-(<arithmetic expression>)

<factor>:: = <primary>|<factor>^<primary>

<term>:: = <factor>|<term> <multiply_divide><factor>

<arithmetic expression>:: = <term>|<arithmetic expression> <add_subtract> <term>

<label constant>:: = "<character string>"|"<character string>

<character string>:: = <character>|<character> <character string>

<character>:: = any ASCII character

<assignment statement>:: = <identifier>=<label constant>|<variable>= <arithmetic expression>

<statement label>:: = $<digit>$_1^4$

<branch statement>:: = JUMP<blank>TO<blank> <statement label>

<label statement>:: = <statement label> <blank>CONTINUE

<I/O statement>:: = DISPLAY(<variable list>)|INPUT(<variable list>)|
CREAD(<variable list>)|PLOT(<variable list>)|
PLOTX(<variable list>)|PLOTP(<variable list>)|
GET[<idn_int>](<variable list>)|
PUT[<idn_int>](<variable list>)|
DREAD[<identifier>,<idn_int>](<variable list>)|
DWRIT[<identifier>,<idn_int>](<variable list>)|

<numerical operator>:: = QSTAT(<variable list>)|XSTAT(<variable list>)|
DSTAT(<variable list>)

<simple executable statement>:: = <assignment statement>|<branch statement>
<I/O statement>|<numerical operator>

<logical operator>:: = =|<|>|<=|>=

<logical expression>:: = <arithmetic expression><logical operator><arithmetic expression>

<conditional statement>:: = IF<blank><logical expression><blank>THEN
<blank><simple executable statement>

<executable statement>:: = <simple executable statement>|<conditional statement>|<label statement>

<statement>:: = <declaration statement>|<executable statement>

SECTION V

CONSTRUCTION OF STANDARD SEISMIC PROCESSING TASKS VIA PROGRAMMED PROCEDURES

This section discusses the ISPSE programmable feature and the construction and maintenance of standard seismic processing tasks via the creation, programming, and execution of named procedures. Subsection A reviews procedure creation and the ordering of processing modules and INTERUP's within a procedure. Subsection B describes the details of programming a procedure including the structure of a procedure's program, system states, user decisions, alphanumeric input, and valid program termination. Subsection C describes the details of executing a programmed procedure, including interrupts in execution, manual branching, and automatic looping. Subsection D provides a scenario for the programming and execution of a sample procedure.

A. PROCEDURE CREATION

As noted in Section II, a procedure consists of an ordered set of system functions, user functions, and INTERUP's to be executed within the ISPSE console environment. Procedures are created and maintained by the ISPSE command language (Section III-B). They provide a mechanism for the user to assemble specific processing modules (system functions and user functions) needed to accomplish a seismic processing task. It is recommended that the user comment a procedure as to its purpose.

When a saved procedure is PERFORM'ed by name with the system in NORMal mode, the associated modules are executed in the order specified. Recall that within system functions, processing options are

selected interactively from displayed keyword menus by hitting the corresponding function keys (numbers 1 - 6) on the CRT keyboard. On encountering an INTERUP (i.e., . PERFORM INTERUP) during procedure execution the system will pause and permit the user to interactively re-start, continue, or stop the procedure by hitting designated keys, number 4, 5, or 6 respectively. Re-start causes a branch to the nearest preceding . PERFORM INTERUP in the sequence or to the beginning of the procedure if no preceding . PERFORM INTERUP exists. Continue resumes execution at the next sequence number. Stop terminates procedure execution.

The inclusion of INTERUP's in a procedure serves two main purposes which may be stated as follows:

- The occurrence of an INTERUP before and after a processing module in a procedure simplifies re-execution of that module in the event of user processing errors. Hence the first advantage of INTERUP's is recoverability. An example of this is shown in Figure V-1.

- The occurrence of an INTERUP before and after a sequence of modules in a procedure enables interactive processing in which that sequence may be executed several times by branching. One such application is the case where the first and last commands in a procedure are . PERFORM INTERUP. This configuration permits the user to establish an autoloop to be discussed later in this section. Briefly, a programmed procedure configured to autoloop may be automatically executed N times, where N is specified during programming. Hence the second advantage of INTERUP's is repetitive processing. An example of this is shown in Figure V-1.

PROCEDURE PROC1	.... . LIST PROC1	The INTERUP at step 2 may be used
(recoverability)	1.0 . PERFORM SELEV	to re-execute the SELEV module at
	2.0 . PERFORM INTERUP	step 1 in case of error. Also, in con-
	3.0 . PERFORM FILTER	junction with the INTERUP at step 4, it
	4.0 . PERFORM INTERUP	may be used to re-execute the module
		FILTER in case or error.
PROCEDURE PROC2	.... . LIST PROC2	This configuration permits the user
(repetitive processing)	1.0 . PERFORM INTERUP	to establish an autoloop for repetitive
	2.0 . PERFORM SELEV	event processing with the modules
	3.0 . PERFORM SPEED	SELEV and SPEED. That is, steps 2
	4.0 . PERFORM INTERUP	and 3 may be automatically executed
		N times to process N events.

FIGURE V-1

UTILIZATION OF INTERUPS IN PROCEDURES

B. PROGRAMMING PROCEDURES

Under ISPSE, a procedure may be programmed to perform a standard seismic processing task by instructing the system to automatically record the sequence of processing options selected interactively from key-work menus during the execution of each system function and INTERUP contained in the procedure. The mechanics for initiating the programming of a procedure are discussed in Section III-B-5 and will not be addressed here. Instead, the objective of this subsection is to define the program for a procedure in terms of its structure and to provide a set of rules for successfully programming the procedure. In this manner, the reader should gain an understanding of the options supported by the ISPSE programmable feature.

1. Program Structure

The program for a procedure is automatically saved as a disk-resident data structure which may be referenced at any time by the procedure's <user name>. The program consists of four sub-structures, namely:

- A Procedure Matrix (PM): the sub-structure in which user decisions (defined below) are sequentially recorded when programming a procedure and from which decisions are retrieved when executing a programmed procedure. The PM is partioned by the modules and INTERUP's contained in the procedure.
- A Procedure Function Address Table (PFAT): The sub-structure which defines the partitioning of the PM. More specifically, if a procedure contains processing modules and INTERUP's numbered 1 through N then the PFAT will contain N values, $PFAT_i$, i = 1, N, such that $PFAT_i$ is the PM index (element) at which decisions for processing module or INTERUP i begin. $PFAT_i$ = 0 implies no decisions exist for module i in the PM. This is always the case for user functions.

- A Numeric Input Matrix (NIM): The sub-structure in which alphanumeric input (defined below) is sequentially recorded when programming a procedure and from which that input is retrieved when executing a programmed procedure. The NIM is partitioned by modules and INTERUP's contained in the procedure.

- A Numeric Function Address Table (NFAT): The sub-structure which defines the partitioning of the NIM. More specifically, if a procedure contains processing modules and INTERUP's numbered 1 through N then the NFAT will contain N values, $NFAT_i$, i = 1, N, such that $NFAT_i$ is the NIM index (element) at which alphanumeric input for module i begins. $NFAT_i = 0$ implies no alphanumeric input exists for module i. This is always the case for user functions and INTERUP's.

Figure V-2a summarizes the role of these sub-structures in supporting the ISPSE programmable feature. Figure V-2b shows a sample listing of a procedure and the program associated with that procedure.

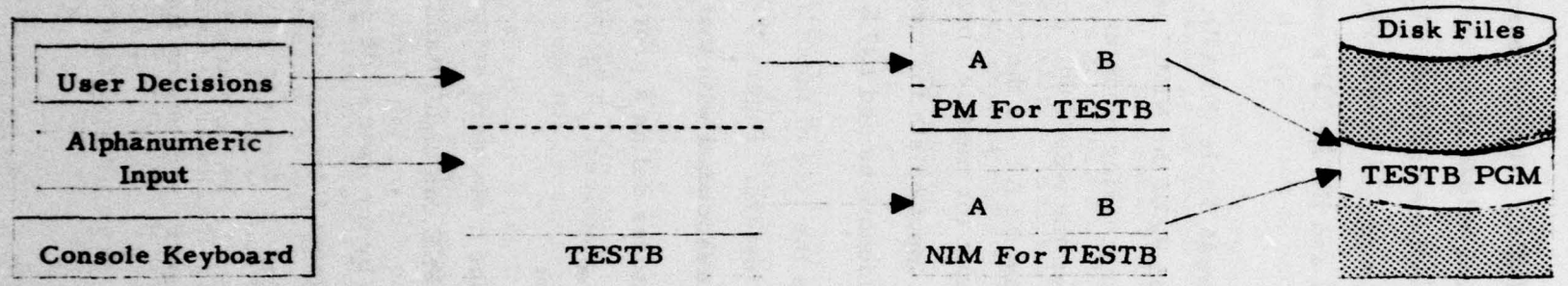
The discussed sub-structures define a procedure's program and are collectively called the program matrices. Together with the procedure itself they have a two-fold purpose:

- They unambiguously define a standard seismic processing task in a fashion which the ISPSE can understand.

- They may be referenced by the user to determine the sequence of a processing options (see: User Decisions, below) performed by the procedure. That is, the user may manually perform the decisions contained in the PM one at a time with the system in NORMal mode and observe processing flow. This technique

```
.... .MODE NORM
.... .PROCS
     TESTA
.... .CREATE TESTB
 1.0 .PERF A
 2.0 .PERF B
 3.0 #S
.... .MODE PRGM
.... .PERF TESTB
```

(Creation and programming of the procedure TESTB)




```
.... .PROCS
     TESTA
     TESTB*
.... .MODE EXEC
.... .PERF TESTB
```

(Execution of the programmed procedure TESTB)

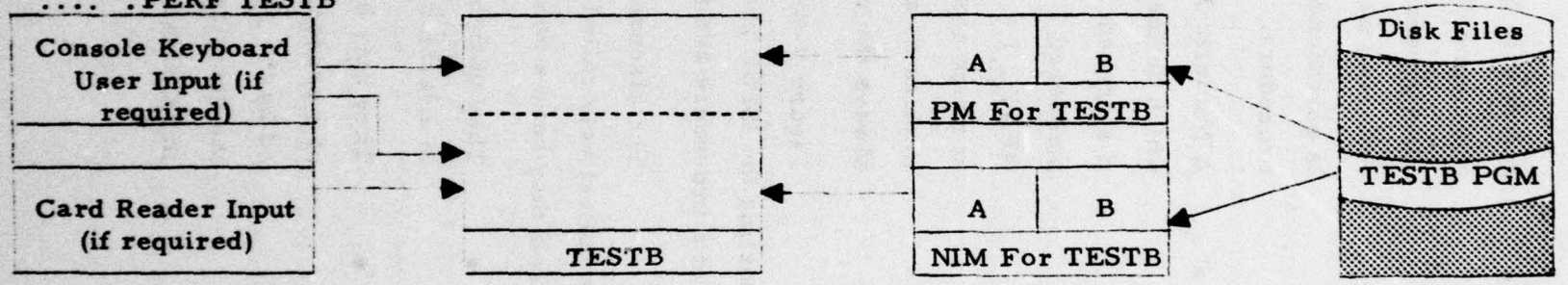


FIGURE V-2a

ISPSE PROGRAMMABLE FEATURE

```
.... .LIST SPECT
 1.0 .PERF INTERUP  THIS PROCEDURE COMPUTES & DISPLAYS THE POWER
 2.0 .PERF SELEV   SPECTRUM FOR A <=1024 PT LONG PERIOD WAVEFORM
 3.0 .PERF INTERUP  WITH AN 'INTERRUPT' IN FILTER FOR OPTIONAL
 4.0 .PERF FILTER  HARD COPY.  AN AUTOLOOP HAS BEEN ESTABLISHED
 5.0 .PERF INTERUP  BUT STEP 5 IS 'INTERRUPTED' FOR MANUAL EXIT.


.... .LIST SPECT*
 PROCEDURE MATRIX
   31 -99   4   1 -99   5   5   6   6   4   5   4   3   3   3
    3   5  -1   6   0
 NUMERIC INPUT MATRIX
    T                T                T           0.2000000E+01
    B           0.7500000E+02  0.2000000E+01  0.2000000E+01
 PROCEDURE FUNCTION ADDRESS TABLE
   1   2  10  11  20
 NUMERIC FUNCTION ADDRESS TABLE
   0   1   0   6   0
```

FIGURE V-2b

SAMPLE LISTING OF A PROCEDURE AND
ITS ASSOCIATED PROGRAM

is especially useful when re-programming a programmed procedure. In fact, it is always good practice to experiment with a procedure in NORMal mode before using PRGM mode to program the procedure.

2. System States

When the ISPSE is in PRGM mode and a procedure is PERFORMed the user is said to be programming the procedure. During this time the system may be in one of two states, 'continue' or 'interrupt', displayed on the CRT as 'CONTINUE PRGM' or 'INTERRUPT PRGM' respectively. The main purpose of these states is to control the recording of user decisions in the PM and alphanumeric input in the NIM. The discussion of user decisions and alphanumeric input which follow should clarify these concepts.

3. User Decisions

a. Functional Description

User decisions are actions taken by the user at the CRT keyboard or in some cases, actions taken by the system for the user. They control the programming of a procedure and may be categorized as follows:

- Within a system function, selection of a processing option from a displayed keyword menu by hitting the corresponding function key (number 1, 2, 3, 4, 5, or 6) on the CRT keyboard. When the system is in 'continue' state that number is recorded in the PM, otherwise it is not recorded. In either case the system then responds as if in NORMal mode.

- Selection of the re-start, continue, or stop option at an INTERUP by hitting the corresponding function key (number 4, 5, or 6) on the CRT keyboard. When the system is in 'continue' state that number is recorded in the PM. Otherwise, a 0 is recorded instead of the number and the system reverts to 'continue' state. In either case the system then responds as if in NORMal mode.

- Hitting the function key 'I' or 'C' on the CRT keyboard. These are called special decisions. To understand them note:

  - Within a system function, processing is guided by key-word menus; each menu is assigned a unique negative integer called a node; the node is displayed with the menu.

  - An INTERUP is assigned the node 0.

  With these facts in mind the following rules apply:

  - If the 'I' function key is hit the system state changes to 'interrupt'; this permits alternate decisions which are not recorded in the PM.

  - If the 'C' function key is hit with the system in 'interrupt' state two things happen; first the system state changes to 'continue'; second the displayed node value is recorded in the PM (unless the last decision before the 'C' was an 'I' or resulted from a dynamic cursor option described below).'

- Within a system function, if the state is 'continue' and an option enabling a dynamic cursor or starting dynamic cursor movement is selected, then the state automatically changes to 'interrupt'. An example of this is the time window selection process in SELEV.

- If the user exits from a system function when the state is 'interrupt' three things happen; first the state automatically changes to 'continue'; second, the node assigned to the menu from which the exit occured is recorded in the PM; third the number 6 (exit option) is recorded in the PM.

- If the user exits from SELEV the number 6 will automatically be recorded in the PM. This is for system reference only and should be ignored by the user.

b. Decisions at INTERUP's

Having described the types of user decisions that may be exercised during the programming of a procedure, some special applications involving decisions at INTERUP's will now be presented:

Referring to Figure V-1, assume that PROC1 is being programmed, all decisions have been made for SELEV, and the system is paused at step 2 (. PERFORM INTERUP) awaiting another decision; then the user may do one of the following:

- Select the re-start option (number 4) to re-program SELEV. This means all previous entries for SELEV in the PROC1 program matrices will be replaced by new entries. In general, re-programming within a procedure means:
  - programming through L steps of the procedure.
  - branching to step J<L via re-starts at INTERUP's.
  - re-executing all modules and INTERUP's from step J through step L.
  - replacing all previous entries for those modules and INTERUP's in the procedure's program matrices with new entries.
- Make two decisions; first, hit the function key 'I' to change the system state to 'interrupt'; second, select the re-start, continue, or stop option by hitting the corresponding function key (number 4, 5, or 6). Result: system state changes back to 'continue' and a 0 node value is recorded in the PM for the INTERUP at step 2. That INTERUP is then called 'interrupted'. In general any INTERUP in a procedure may be 'interrupted'.

During the execution of a programmed procedure (MODE=EXEC), 'interrupted' INTERUP's permit manual branching because they behave as if the system were in NORMal mode.

- Select the continue option (number 5). Result: a 5 is recorded in the PM for the INTERUP at step 2. In this case the INTERUP will be serviced from the PM when MODE=EXEC.

Again referring to Figure V-1 recall that PROC2 is configured to establish an autoloop because the first and last steps are .PERFORM INTERUP. Assume that PROC2 is being programmed, all decisions have been made through step 3, and the system is paused at step 4 (.PERFORM INTERUP) awaiting another decision; then the user may proceed as follows:

- Select the re-start option (number 4). Result: a 4 is recorded in the PM for the INTERUP at step 4; the system branches to the *INTERUP* at step *1*.
- Select the stop option (number 6). Result: the system exits and requests a value for the autoloop counter N where $1 \leq N \leq 99999$.
- Enter a value for N followed by a carriage return. Result: if $1 \leq N \leq 99999$ the system records the quantity N+6 in the PM for the INTERUP at step 1 and returns control to the Interpreter Supervisor. Otherwise another value for N must be entered.

In general, the user may establish an autoloop for any procedure whose first and last steps are INTERUP's by:

- programming (re-programming) through the next to last step.
- branching to step 1 via a sequence of one or more re-starts.

- selecting the stop option to exit from the procedure.
- entering a value for the autoloop counter.

Note: the autoloop counter specifies the number of times to repeat the procedure during execution (MODE=EXEC).

4. Alphanumeric Input

When programming a procedure, system functions frequently require the user to enter alphanumeric information in response to selected processing options. That alphanumeric information (except for user comments) is automatically recorded in the NIM if and only if the system is in 'continue' state. Moreover, permissible alphanumeric entries may be categorized as follows:

- A valid <number> as defined in Section IV-D. The corresponding value is recorded in the NIM.
- The character string 'Y' or 'YES' (in response to a question). A 'Y' is recorded in the NIM.
- The character string 'N' or 'NO' (in response to a question). An 'N' is recorded in the NIM.
- The null or default entry (i.e., a carriage return). A 'B' for blank is recorded in the NIM.
- The character string 'T$\not\beta$'<number>. The system accepts the number but records a 'T' in the NIM. This designates that the value will be requested from the keyboard instead of retrieved from the NIM when the procedure is executed (MODE=EXEC).
- The character string 'C$\not\beta$'<number>. The system accepts the number but records a 'C' in the NIM. This designates that

the value will be read from a card in the card reader instead of retrieved from the NIM when the procedure is executed (MODE =EXEC).

- A re-entry of any alphanumeric entry noted above if flagged invalid by the system.

5. Valid Program Termination

In order for the system to designate a procedure as programmed, to save the procedure's program matrices, and to display the message "PROGRAM OK FOR PROCEDURE <user name>", the program must be terminated by one of the following techniques:

- If the first step in the procedure is an INTERUP then the last step must be an INTERUP and the program is terminated in the manner described for establishing an autoloop (Section IV-B-3). If desired, a minor variation on that technique is permitted, that is, the INTERUP at the last step and/or step 1 may be 'interrupted' to establish a manual rather than an automatic loop.

- To change the autoloop counter N of a programmed procedure complete reprogramming is not required. One need only PERFORM the procedure with MODE=PRGM and immediately exit at step 1 via key number 6. Then only the first PM entry (=N+6) will be modified while all other program matrices remain undisturbed.

- If the first step in a procedure is not an INTERUP then the procedure is terminated by exiting at the last step in the procedure. If the last step is a . PERFORM <system function> then the system must be in 'continue' state before the exit occurs.

Termination by techniques other than those described above will result in loss of all recorded user decisions and alphanumeric input for the procedure. Complete re-programming would then be required.

C. EXECUTING PROGRAMMED PROCEDURES

Under ISPSE, a programmed procedure may be executed to perform a standard seismic processing task by instructing the system to retrieve the processing sequence from the procedure's program matrices. Recall (Section V-B) that the processing sequence consists of user decisions and alphanumeric input recorded in the matrices when the procedure was programmed. The mechanics for initiating the execution of a programmed procedure are discussed in Section III-B-5 and will not be addressed here. Instead, the objective of this subsection is to provide a set of operational rules for programmed procedure execution.

1. Interrupts in Execution

When ISPSE is in EXEC mode and a programmed procedure is PERFORMed the user is said to be executing the programmed procedure. During execution the system automatically retrieves user decisions from the PM to serve as menu selections within system functions and at INTERUP's. Likewise requests for alphanumeric input are serviced from the NIM. Thus, procedure execution will continue without the need for user intervention unless one of the following should occur:

- The system encounters a node (≤ 0) value in the PM.
- The system encounters a 'T' or 'C' in the NIM.

Each of these conditions cause an interrupt in execution and are discussed individually below:

a. Node Interrupt

Recall (Section V-B) that during the programming of a procedure, node values are recorded in the PM when the system state changes between 'continue' and 'interrupt' state. Conversely, the retrieval of a node value from the PM during the execution of a programmed procedures changes the system state from 'continue' to 'interrupt'. That is, execution will halt at a given menu and user intervention is then required. At this point the following options exist:

- For node value <0, the user may make alternate menu selections and numeric entries within the system function where execution halted, and the system will respond as if in NORMal mode. If the user exits from that system function then automatic execution will resume (from the PM) at the next step in the procedure.
- For node value =0, the user may perform a manual branch by selecting the re-start, continue, or stop option. This is the 'interrupted' INTERUP case discussed in Section V-B-3. Following the branch, automatic execution will resume at the appropriate step in the procedure.
- For node value = node value of currently displayed menu, the user may hit the function key 'C' to continue. This resumes automatic execution and retrieves the next PM entry to serve as the menu selection. If the node value of the currently displayed menu $\neq$ the node value which caused the halt in execution, the 'C' is ignored.

b. 'T' or 'C' Interrupt

When a 'T' or 'C' is retrieved from the NIM during the execution of a programmed procedure, the system will pause and accept a

<number> from the CRT keyboard or the card reader, respectively. Where as these numeric input options are incorporated into a procedure during programming (Section V-B-4), the user should be prepared to enter the information via the designated device (i.e., by monitoring the CRT console or stacking cards in the reader). Following the entry, automatic execution of the procedure is resumed.

2. Execution of the Autoloop

Recall (Section V-B-3) that the quantity N+6 is recorded as the first PM entry for a procedure programmed to autoloop. The execution of such a procedure is governed by the following rules:

- The quantity OP=N+6 is retrieved from the first PM entry, decremented by 1, and re-recorded in the first PM entry, each time step 1 is executed.
- If OP>6, then the INTERUP at step 1 is automatically serviced with a 5, to continue.
- If OP=6, then the INTERUP at step 1 is automatically serviced with a 6, to stop, terminating procedure execution and the autoloop.
- The PM entry for the INTERUP at the last step in the procedure is 4 to re-start, and all other INTERUP's between the first and last step are ignored.

Collectively, these rules imply that the procedure will automatically repeat N times when executed with MODE=EXEC.

D. PROGRAMMING AND EXECUTING A SAMPLE PROCEDURE

This subsection provided a scenario for the programming and execution of a sample procedure.

Assume that a procedure called SPECT has been CREATEd as described in Section III-B and that SPECT consists of the sequence: 1.0 . PERF INTERUP, 2.0 . PERF SELEV, 3.0 . PERF INTERUP, 4.0 . PERF FILTER, and 5.0 . PERF INTERUP. Further assume that SPECT has been programmed to compute and display the power spectra of selected long-period waveforms. Then a listing for SPECT and its associated program matrices is provided in Figure V-3. Note that boundary lines have been drawn to show the partioning of the Procedure Matrix (PM) and the Numeric Input Matrix (NIM) by the steps in the procedure. The PM partitioning is defined by the Procedure Function Address Table (PFAT) and the NIM partitioning is defined by the Numeric Function Address Table (NFAT).

More specifically, the PFAT for SPECT contains the indices 1, 2, 10, 11, and 20, indicating that user decisions for steps 1.0 . PERF INTERUP, 2.0 . PERF SELEV, 3.0 PERF INTERUP, 4.0 . PERF FILTER, and 5.0 . PERF INTERUP begin at PM elements 1, 2, 10, 11, and 20 respectively. Also, the NFAT for SPECT contains the indices 0, 1, 0, 6, 0, indicating that alphanumeric input for steps 2.0 . PERF SELEV and 4.0 . PERF FILTER begins at NIM elements 1 and 6 respectively. No alphanumeric input exists in the NIM for the INTERUP's at steps 1.0, 3.0, and 5.0.

With this background, scenarios will now be presented which describe how listed PM and NIM elements were recorded during the programming of SPECT and how they are utilized during an execution of SPECT. A step by step description of the various actions taken by the user and the resulting system response is given in the following format:

- Message (M): A message displayed on the CRT screen to perform a specified function (e.g., enter alphanumeric information via the CRT keyboard). M may also be a keyword menu which displays up to six processing options

```
..... .MODE NORM
 MODE = NORM

..... .LIST SPECT
 1.0 .PERF INTERUP  THIS PROCEDURE COMPUTES & DISPLAYS THE POWER
 2.0 .PERF SELEV    SPECTRUM FOR A <=1024 PT LONG PERIOD WAVEFORM
 3.0 .PERF INTERUP  WITH AN 'INTERRUPT' IN FILTER FOR OPTIONAL
 4.0 .PERF FILTER   HARD COPY.  AN AUTOLOOP HAS BEEN ESTABLISHED
 5.0 .PERF INTERUP  BUT STEP 5 IS 'INTERRUPTED' FOR MANUAL EXIT.

..... .LIST SPECT*
                                SELEV           INTERUP          FILTER
 IN-    PROCEDURE MATRIX
 TERUP  31  -99   4   1  -99   5   5   6   6     4     5   4   3   3   3
         3   5  -1   6   0
 NUMERIC INPUT MATRIX  INTERUP                  SELEV
         1          1             1                        0.2000000E+01
         8           0.7500000E+02  0.2000000E+01   0.2000000E+01
 PROCEDURE FUNCTION ADDRESS TABLE
    1   2  10  11  20                            FILTER
 NUMERIC FUNCTION ADDRESS TABLE
    0   1   0   6   0
```

FIGURE V-3

LISTING OF THE PROGRAMMED PROCEDURE SPECT
AND ITS ASSOCIATED PROGRAM MATRICES

corresponding to the function keys (number 1, 2, 3, 4, 5, and 6) on the CRT keyboard.

- Action (A): Is the subsequent action taken by the analyst (e.g., alphanumeric input or a user decision).
- Explanation (E): Gives an explanation as to the result of an analyst action.

In the scenarios which follow it is assumed that ISPSE has been initiated according to the directions described in Section II and that the user is working at the CRT console. All dialogue refers to that CRT display and associated CRT keyboard. Also, for a detailed description of the system functions SELEV and FILTER which appear in the scenarios, refer to "Documentation of the ISPS" (Ringdal, et al., 1975).

1. Programming SPECT

Referring to the keyword menus for SELEV and FILTER illustrated in Figure V-5 and V-6, respectively, to the console communications for the programming of SPECT shown in Figure V-4, and to the SPECT program matrices in Figure V-3:

A: Type: .MODE PRGM
E: The system goes to PRGM (program)mode.

A: Type: .PERF SPECT
E: The system initiates execution of the procedure SPECT and pauses at step 1, .PERF INTERUP.

M: PAUSE AT 1.0: #4 RESTART, #5 CONTINUE, #6 STOP
A: Hit function key number 5 on the CRT keyboard, to continue.

```
.... .MODE PRGM
 MODE = PRGM
.... .PERF SPECT
 USER PROCEDURE SPECT INITIATED
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST
.... T 1
ENTER STATION NUMBER (1,15)
.... T 6
ENTER COMPONENT NUMBER (1,3)
.... T 1
ENTER NUMBER OF COPIES TO BE SAVED : TS =  1253   TL =   814 SEC
.... 2
ENTER EVENT SEQUENCE NUMBER FROM LIST
....
 PAUSE AT  3.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION FILTER INITIATED
 PAUSE AT  5.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  3.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 AUTOLOOP REQUESTED, ENTER NUMBER OF TIMES TO LOOP
.... 25
 PROGRAM OK FOR PROCEDURE SPECT
```

FIGURE V-4

CONSOLE COMMUNICATIONS FOR
PROGRAMMING OF SPECT

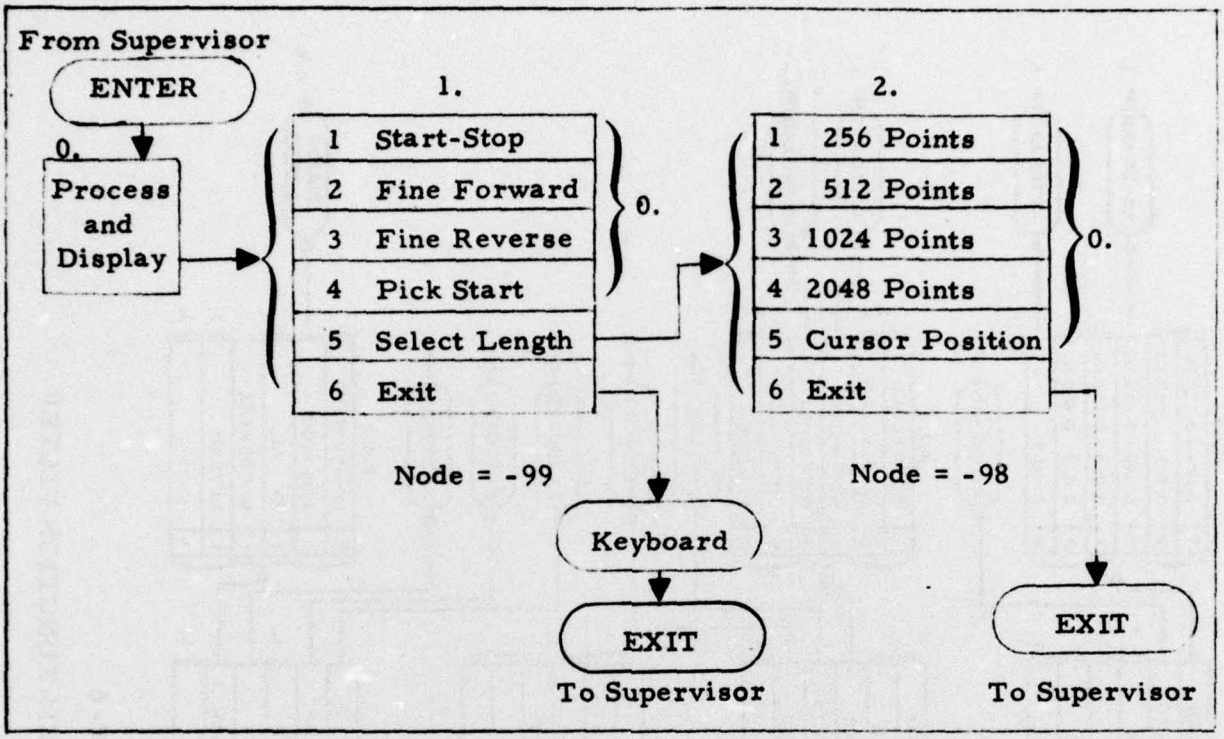


FIGURE V-5

KEYWORD MENUS FOR
SYSTEM FUNCTION SELEV

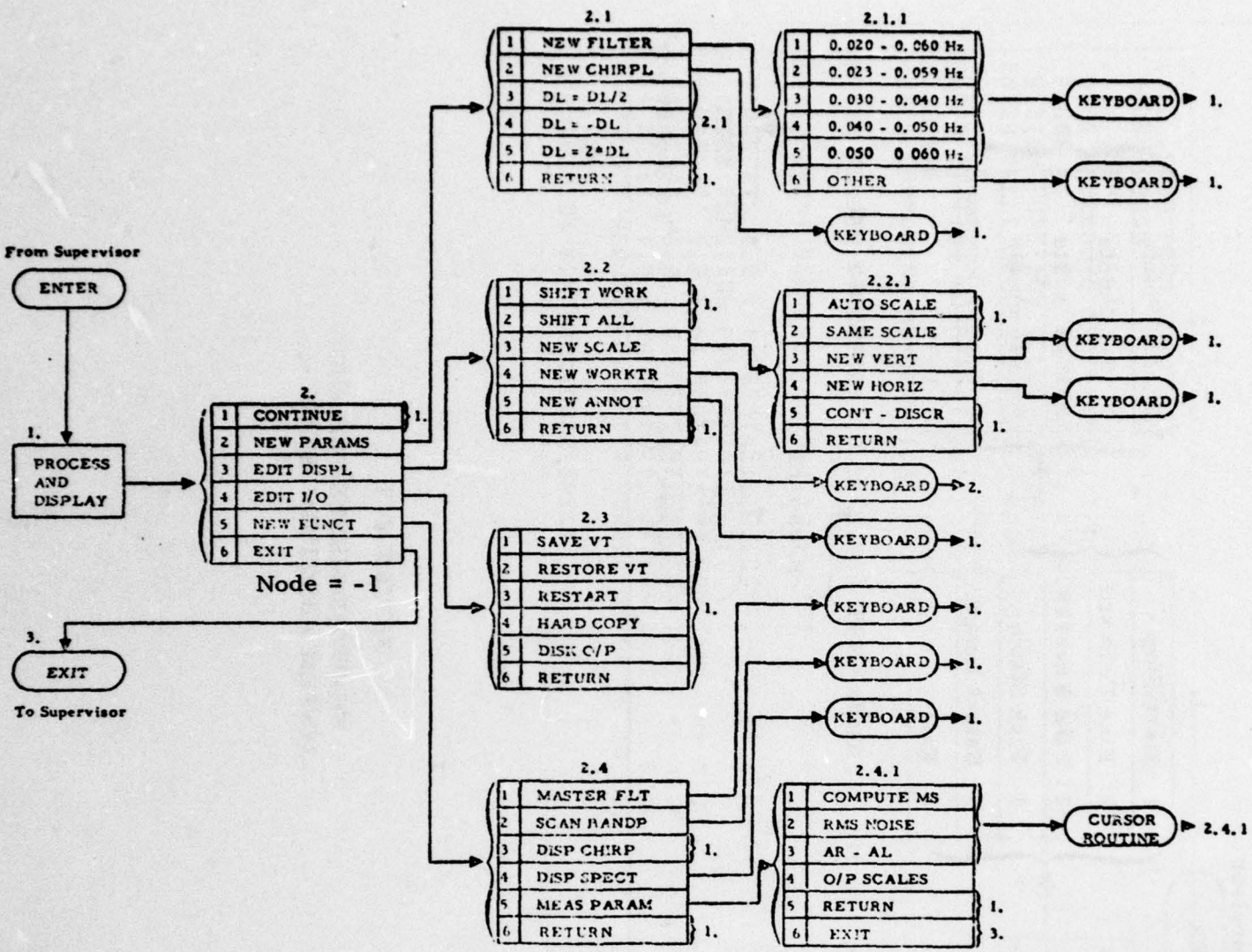


FIGURE V-6

KEYWORD MENUS FOR SYSTEM FUNCTION FILTER

E: Execution of SPECT is continued at step 2 where SELEV is initiated and the system state 'CONTINUE PRGM' is displayed.

M: SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST

A: Type: T01

E: Selects event number 1 for processing and records a 'T' in the SELEV partition of the NIM.

M: ENTER STATION NUMBER (1, 15)

A: Type T06

E: Selects station number 6 and records another 'T' in the NIM.

M: ENTER COMPONENT NUMBER (1, 3)

A: Type: T01

E: Selects component number 1 and records a third 'T' in the NIM. Then 2048 points of the long-period waveform for event #1, station #6 and component #1 are displayed.

M: SELEV keyword menu 1 for dynamic cursor operation

A: System state automatically changes to 'INTERRUPT PRGM'

E: System enters 'interrupt' state because a dynamic cursor is enabled.

M: SELEV keyword menu 1.

A: Hit function key number 1 (START-STOP)

E: Cursor begins moving across waveform.

M: SELEV keyword menu 1.

A: Hit function key number 1 (also possibly number 2 and/or number 3).

E: Stops cursor movement to position cursor at beginning of time window.

M: SELEV keyword menu 1.

A: Hit function key 'C'

E: System state changes to 'CONTINUE PRGM' and the node -99 is recorded in the SELEV partition of the PM.

M: SELEV keyword menu 1.

A: Hit function key number 4 (PICK START)

E: Selects point at which cursor is positioned as the start of the time window and records a 4 in the PM.

M: SELEV keyword menu 1.

A: Hit function key number 1.

E: Cursor begins moving, a 1 is recorded in the PM, and the system state automatically changes to 'INTERRUPT PRGM' because a dynamic cursor is being moved.

M: SELEV keyword menu 1.

A: Hit function key number 1 (also possibly number 2 and/or number 3)

E: Stops cursor movement to position cursor at end of time window.

M: SELEV keyword menu 1.

A: Hit function key 'C'

E: System state changes to 'CONTINUE PRGM' and the node -99 is recorded in the PM.

M: SELEV keyword menu 1.

A: Hit function key number 5 (SELECT LENGTH)

E: Records a 5 in the PM and displays a new menu.

M: SELEV keyword menu 2.

A: Hit function key number 5 (CURSOR POSITION)

E: Selects the point at which the cursor is positioned as the end of the time window and records a 5 in the PM.

M: SELEV keyword menu 1.

A: Hit function key number 6 (EXIT)

E: Exits from the window selection phase of SELEV and records a 6 in the PM.

M: ENTER NUMBER OF COPIES TO BE SAVED: TS=1253 TL=814 SEC.

A: Type: 2

E: Saves two copies of the edited waveform and records a 2 in the NIM.

M: ENTER EVENT SEQUENCE NUMBER FROM LIST

A: Type: carriage return (i.e., the null entry)

E: The system exits from SELEV, records a 'B' in the NIM and a 6 in the PM, and pauses at step 3, .PERF INTERUP.

M: PAUSE AT 3.0: #4 RESTART, #5 CONTINUE, #6 STOP

A: Hit function key number 5 on the CRT keyboard, to continue

E: Execution of SPECT is continued at step 4 where the system function *FILTER in initiated* and two copies of the edited waveform are displayed.

M: FILTER keyword menu 2.

A: Hit function key number 5 (NEW FUNCT)

E: Records a 5 in the FILTER partition of the PM and displays a new menu.

M: FILTER keyword menu 2.4.

A: Hit function key number 4 (DISP SPECT).

E: Computes the spectrum of trace number 2 - work trace and records a 4 in the PM.

M: ENTER DESIRED RESOLUTION IN DB-DEFAULT=40

A: Type: 75

E: Specifies a resolution of 75 DB, records a 75 in the FILTER partition of the NIM, and displays the computed spectrum.

M: FILTER keyword menu 2.

A: Hit function key number 3 (EDIT DISPL)

E: Records a 3 in the PM and displays a new menu.

M: FILTER keywork menu 2.2.

A: Hit function key number 3 (NEW SCALE)

E: Records a 3 in the PM and displays a new menu.

M: FILTER keyword menu 2.2.1.

A: Hit function key number 3 (NEW VERT)

E: Records a 3 in the PM and initiates request for a new vertical scale.

M: ENTER NEW VERTICAL SCALE - CURRENT VALUE IS 2.4 INCHES

A: Type: 2

E: The vertical scale factor is decreased to 2 inches (improving trace separation), the traces are re-displayed accordingly, and a 2 is recorded in the NIM.

M: FILTER keyword menu 2.

A: Hit function key number 3 (EDIT DISPLAY)

E: Records a 3 in the PM and displays a new menu.

M: FILTER *keyword menu 2. 2.

A: Hit function key 5 (NEW ANNOT)

E: Records a 5 in the PM and initiates request for new annotation.

M: ENTER NEW ANNOTATION INDEX - CURRENT VALUE IS 1.

A: Type: 2

E: More complete annotation is provided for each trace and a 2 is recorded in the NIM.

M: FILTER keyword menu 2.

A: Hit function key 'I'

E: System state changes to 'INTERRUPT PRGM'. The objective here is to provide an interrupt in execution for optional hard copy.

M: FILTER keyword menu 2.

A: Hit function key 6 (EXIT)

E: The system exits from SELEV, records the node -1 followed by a 6 in the PM, reverts to the state 'CONTINUE PRGM', and pauses at step 5, .PERF INTERUP.

M: PAUSE AT 5. 0: #4 RESTART, #5 CONTINUE, #6 STOP

A: Hit function key 'I'

E: System state changes to 'INTERRUPT PRGM'. The objective here is to provide an 'interrupted' INTERUP for manual branching during execution.

M: PAUSE AT 5.0: #4 RESTART, #5 CONTINUE, #6 STOP

A: Hit function key number 4 to restart.

E: A 0 is recorded in the PM for the INTERUP at step 5, the system state reverts to 'CONTINUE PRGM' and control is transferred to step 3, .PERF INTERUP, where the system pauses.

M: PAUSE AT 3.0: #4 RESTART, #5 CONTINUE, #6 STOP

A: Hit function key number 4 to restart

E: A 4 is recorded in the PM for the INTERUP at step 3 and control is transferred to step 1, .PERF INTERUP, where the system pauses.

M: PAUSE AT 1.0: #4 RESTART, #5 CONTINUE, #6 STOP

A: Hit function key number 6 to stop

E: The objective here is to terminate the programming of SPECT by establishing an autoloop.

M: AUTOLOOP REQUESTED, ENTER NUMBER OF TIMES TO LOOP

A: Type: 25

E: Specifies an autoloop counter = 25 (providing for the processing of 25 waveforms via SPECT), records a 31 = 25+6 in the PM

for the INTERUP at step 1, and returns control to the ISPSE Interpreter Supervisor with the message: PROGRAM OK FOR PROCEDURE SPECT. This means that the program for SPECT has been terminated in a valid fashion and that the procedure SPECT is now designated as programmed.

2. Execution of SPECT

Referring to the keyword menus for SELEV and FILTER illustrated in Figure V-5 and Figure V-6, respectively, to the console communications for the execution of SPECT shown in Figure V-7, and to the SPECT program matrices in Figure V-3:

(1) A: Type: . MODE EXEC

E: The system goes to EXEC (execute) mode

(2) A; Type: . PERF SPECT

E: The system initiates execution of the programmed procedure SPECT

(3) M: PAUSE AT 1.0: #4 RESTART, #5 CONTINUE, #6 STOP

E: System automatically services this INTERUP from the PM, according to autoloop conventions, and execution continues at step 2.

(4) M: SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST

A: Type: 1

E: User intervention is required at this point because a 'T' is retrieved from the SELEV partition of the NIM. Hence the user selects event number 1 via the CRT keyboard.

```
.... .MODE EXEC
 MODE = EXEC
.... .PERF SPECT
 USER PROCEDURE SPECT INITIATED
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST
.... 1
ENTER STATION NUMBER (1,15)
.... 6
ENTER COMPONENT NUMBER (1,3)
.... 1
ENTER NUMBER OF COPIES TO BE SAVED : TS =  1251   TL =   816 SEC
....   0.2000000E+01
ENTER EVENT SEQUENCE NUMBER FROM LIST
....
 SYSTEM FUNCTION FILTER INITIATED
 PAUSE AT  5.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST
.... 1
ENTER STATION NUMBER (1,15)
.... 6
ENTER COMPONENT NUMBER (1,3)
.... 2
ENTER NUMBER OF COPIES TO BE SAVED : TS =  1499   TL =   862 SEC
....   0.2000000E+01
ENTER EVENT SEQUENCE NUMBER FROM LIST
....
 SYSTEM FUNCTION FILTER INITIATED
 PAUSE AT  5.0 : #4 RESTART, #5 CONTINUE, #6 STOP
```

FIGURE V-7

CONSOLE COMMUNICATIONS FOR
EXECUTION OF SPECT

(5) M: ENTER STATION NUMBER (1, 15)

A: Type: 6

E: Here again user intervention is required because another 'T' is retrieved from the NIM. Hence the user selects station number 6 via the CRT keyboard.

(6) M: ENTER COMPONENT NUMBER (1,3)

A: Type: 1

E: User intervention is required once more as a third 'T' is retrieved from the NIM. Here the user selects component number 1 and then 2048 points of the long-period waveform for event #1, station #6, and component #1 are displayed.

(7) M: SELEV keyword menu 1 for dynamic curser operation

A: Hit function key number 1 (START-STOP)

E: User intervention is required at this point because a -99 node is retrieved from the SELEV partition of the PM. This action starts the cursor moving across the waveform.

(8) M: SELEV key word menu 1.

A: Hit function key number 1 (also possibly number 2 and/or number 3)

E: Stops cursor movement to position curser at beginning of window.

(9) M: SELEV keyword menu 1 .

A: Hit function key 'C'

E: Automatic execution is continued. The sequence 4, 1, -99 is retrieved from the PM resulting in the selection of the cursor position as the start of the time window, followed by an interrupt execution.

(10) M: SELEV keyword menu 1.

A: Hit function key number 1

E: User intervention is required to begin cursor moving across the waveform.

(11) M: SELEV keyword menu 1:

A: Hit function key number 1 (also possibly number 2 and/or number 3)

E: Stops cursor movement to position cursor at end of time window

(12) M: SELEV keyword menu 1

A: Hit function key 'C'

E: Automatic execution is continued. The sequence 5, 5, 6, 6 is retrieved from the PM and the sequence 2, 'B' from the NIM to select the cursor position as the end of the time window, to save two copies of the edited waveform, and to exit from SELEV.

(13) M: SYSTEM FUNCTION FILTER INITIATED

E: Automatic execution continues at step 4 because the INTERUP at step 3 is ignored during the execution of the SPECT auto-loop. Then the sequence 5, 4, 3, 3, 3, 3, 5, -1 is retrieved from the FILTER partition of the PM and the sequence 75, 2, 2 from the FILTER partition of the NIM to compute, display, re-scale, and re-annotate the spectrum for the edited waveform. Finally, retrieval of the -1 interrupts execution.

(14) M: FILTER keyword menu 2.

A: Hit function key number 3 (EDIT I/0)

E: User intervenes here to display a new menu

(15) M: FILTER keyword menu 2.3

A: Hit function key number 4 (HARD COPY)

E: User intervenes to request optional hard copy shown in Figure V-8.

(16) M: FILTER keyword menu 2.

A: Hit function key 'C'

E: Automatic execution is continued. The system retrieves a 6 from the PM to exit from FILTER, followed by a 0 for the INTERUP at step 5, causing an interrupt in execution.

(17) M: PAUSE AT 5.0: #4 RESTART, #5 CONTINUE, #6 STOP

A: Hit function key #4 to restart

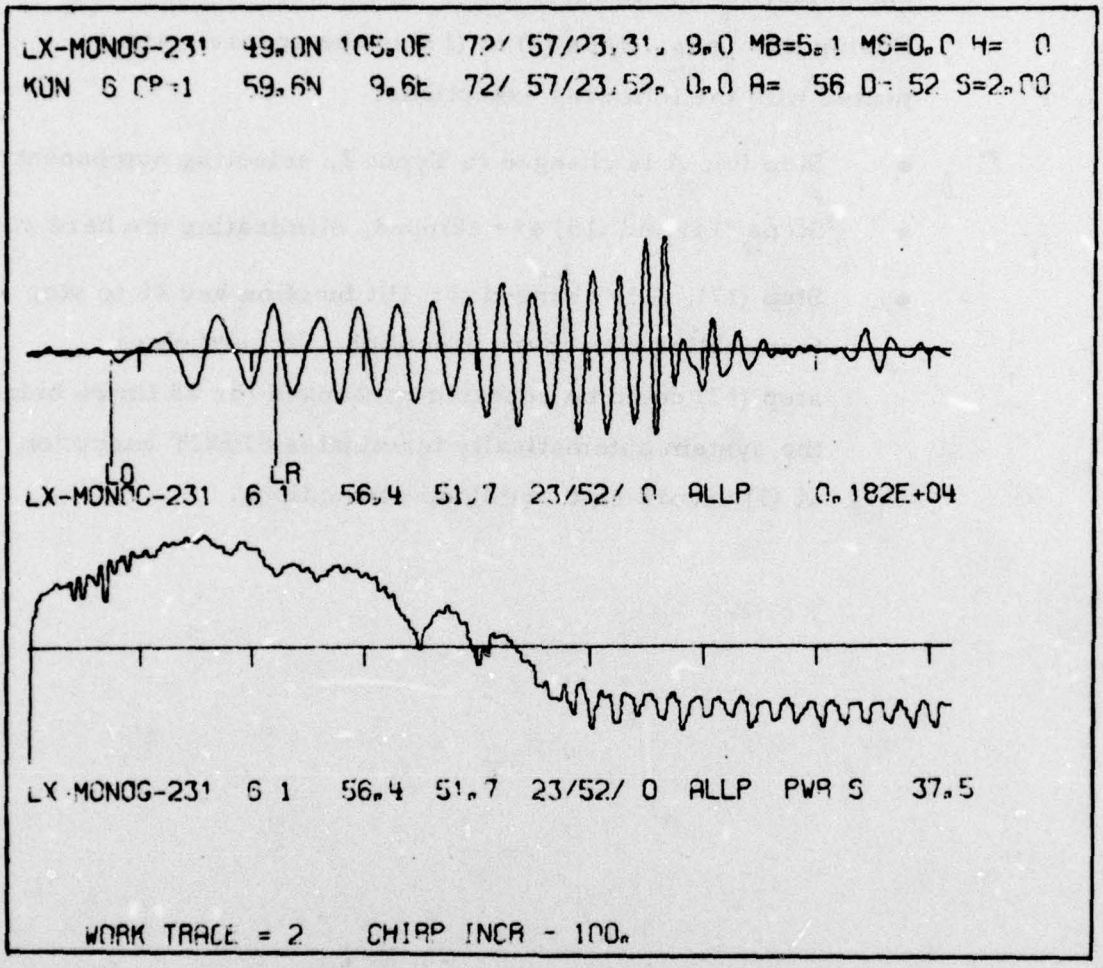


FIGURE V-8

HARDCOPY OUTPUT FROM EXECUTION OF SPECT

E: Automatic execution is resumed as the user intervenes and performs a manual branch to re-execute the procedure. During this pass, steps (3) to (17) in the scenario are repeated with the following exceptions:

- Step (6), A is changed to Type: 2, selecting component 2
- Steps (14) and (15) are skipped, eliminating the hard copy
- Step (17), A is changed to: Hit function key #6 to stop terminating procedure execution. Nevertheless, step (17) could be repeated unchanged for 25 times before the system automatically terminates SPECT execution at (3) according to autoloop conventions.

SECTION VI
REFERENCES

Backus, J.W., 1959, The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference, Proc. Internat. Conf. Inf. Proc. UNESCO, Paris.

Ringdal, Frode, Jeffrey S. Shaub, and David G. Black, 1975, Documentation of the Interactive Seismic Processing System (ISPS), AFTAC Contract Number F08606-75-C-0029, Texas Instruments Incorporated, Equipment Group, Dallas, Texas.

Sax, Robert L., 1976, Design, Simulated Operation, and Evaluation of a Short-Period Seismic Discrimination Processor in the Context of a World-wide Seismic Surveillance System, Technical Report No. 9, AFTAC Contract Number F08606-76-C-0011, Texas Instruments Incorporated, Equipment Group, Dallas, Texas.

Shaub, Jeffrey S., David A. Eastburn, David R. Lashmit, and Robert L. Sax, 1977, Documentation of the Extended Interactive Seismic Processing System (ISPSE), AFTAC Contract Number F08606-77-C-0004, Texas Instruments Incorporated, Equipment Group, Dallas, Texas.

APPENDIX A

USE OF ISPSE FOR BASELINE DISCRIMINATION

This appendix describes the operation of the Interactive Short-Period Event Discriminator (ISPED) procedure and the Multivariate Event Discriminator (MDCRM) user function. This procedure has been programmed to perform a standard processing sequence for baseline discrimination, and the user function has been designed to accomplish multivariate discrimination by means of the high level numerical operator statements, QSTAT, XSTAT, and DSTAT (see Section IV). The baseline and multivariate discrimination techniques have been discussed by Sax (1976).

The ISPED programmed procedure performs the operations as shown in Figure A-1. It is noted here that the special disk file named SPEED PCM which contains the event discriminant values, may be accessed using the ISPL I/O operator, DREAD (see Section IV). The event number, from the DPSCAN (see Section II) event list, is used to specify the record number containing the discriminant values which are calculated within the SPEED system function (see Section II). Figure A-2 lists the operations of the user function, MDCRM. This function automatically retrieves the discriminant values from SPEED PCM to calculate the mean detectability vectors and the projections onto these vectors.

The steps used to create and program ISPED are shown in Figure A-3. An autoloop is programmed into the procedure with an autoloop counter equal to four. The autoloop counter may be changed according to the instructions discussed in Section V-B-5 without reprogramming the procedure. The program matrices for the programmed procedure, ISPED, are listed

- Select displacement (velocity, acceleration) wavelet from disk.
- Decimate wavelet.
- Exponentially taper wavelet.
- Smooth spectra of deconvolved signal and echo.
- Inverse taper the reconstructed signal.
- Compute variable frequency magnitude discriminants using log-log spectral values and store them on a special disk file called SPEED PCM for later access by MDCRM.
- Display log-log spectrum, tabulate the discriminant values on the CRT, and generate a hard copy of the display.
- Select the number of events to process with the autoloop feature.

FIGURE A-1
ISPED OPERATIONS

- Select sample earthquake population from the event list.
- Calculate the mean and standard deviation vectors, EQ and SDQ from the earthquake population.
- Select two regional presumed explosion populations.
- Calculate the mean detectability vectors, EDX1, and EDX2, for each region.
- Select event population to be classified.
- Calculate detectability vectors for all events and project them onto EDX1 and EDX2.
- Display projections as a point plot with different intensities for each population.

FIGURE A-2
MDCRM OPERATIONS

```
.... .PROCS
       EVNTL*
       SPECT*
       BANDP*
.... .CREATE ISPED
 1.0 .PERF INTERUP  THIS PROCEDURE CALCULATES A SET OF VFM
 2.0 .PERF SELEV    EARTHQUAKE/EXPLOSION DISCRIMINANT VALUES.
 3.0 .PERF INTERUP  A HARD COPY IS MADE FOR EACH EVENT SHOWING
 4.0 .PERF SPEED    ITS LOG LOG SPECTRUM AND THE DISCRIMINANT
 5.0 .PERF INTERUP  PARAMETERS (SAVED ON FILE = SPEED PCM).
 6.0 #S
.... .MODE PRGM
 MODE = PRGM
.... .PERF ISPED
 USER PROCEDURE ISPED INITIATED
 PAUSE AT  1.2 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST
.... C 6
ENTER STATION NUMBER (1,15)
.... 1
ENTER COMPONENT NUMBER (1,3)
.... 1
ENTER NUMBER OF COPIES TO BE SAVED : TS =  2280   TL =    12 SEC
.... 1
ENTER EVENT SEQUENCE NUMBER FROM LIST
....
 PAUSE AT  3.2 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SPEED INITIATED
T/O (TOTAL) VALUE IN ABSORB =   0.000E+00   HIT CR TO CONTINUE
....
DO YOU WISH TO DECIMATE? (Y,N)
.... Y
ENTER REFLECTION COEFICIENT
.... Ø
ENTER  SCATTERING COEFFICIENT   (0.0<SC<0.5)
.... 0
ENTER ECHO DELAY IN POINTS (NDEL)
.... Ø
DO YOU WISH TO PROCESS SECONDARY REFLECTION? (Y OR N)
.... N
ANALYZE RESIDUAL ?  (Y,N)
.... N
```

FIGURE A-3

CREATION AND PROGRAMMING OF ISPED
(PAGE 1 OF 2)

```
ENTER POINT AT WHICH TO ZERO DATA (<128)
.... 100
ENTER 'Y' IF FREQS OK,OTHERWISE ENTER INDEX
.... Y
 ENTER LOWEST FREQUENCY TO BE DISPLAYED IN SPECTRUM (LOG FREQ)
.... -.4
ENTER HIGHEST FREQ TO BE DISPLAYED (MUST BE < 0.6990)
.... .69
 PAUSE AT  5.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  3.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 AUTOLOOP REQUESTED, ENTER NUMBER OF TIMES TO LOOP
.... 4
 PROGRAM OK FOR PROCEDURE ISPED
```

FIGURE A-3

CREATION AND PROGRAMMING OF ISPED
(PAGE 2 OF 2)

in Figure A-4. Note that there are no "interrupts" in the procedure because all of the values in the Procedure Matrix are positive. Therefore no analyst interaction is required during the execution of ISPED.

Figure A-5 provides a list of the user function, MDCRM. The discriminant values must be calculated through the execution of ISPED, for all events used in the multivariate discrimination analysis, before MDCRM is .PERFORM'ed.

Figure A-6 provides a list of the output of ISPED when executed for 4 events. Figure A-7 is the hardcopy Calcomp plots of the CRT displays from the same run. Figure A-8 displays the output of MDCRM when executed for twenty earthquakes, ten presumed explosions from region 1, three presumed explosions from region 2, and two events to be discriminated. The input to MDCRM consists of two card decks. The first deck contains the event numbers for the earthquake sample population, and the presumed explosions from regions 1 and 2, in that order. The second deck includes all of the first deck plus the event numbers of the events to be discriminated. Remember that the programmed procedure, ISPED, must be executed for all 35 events before MDCRM is .PERFORM'ed.

Figure A-9 is the Calcomp plot of the projection vectors. The circles, in increasing diameters, represent the sample earthquake population, the presumed explosions from the first region, the presumed explosions from region 2, and the events to be discriminated, respectively.

Once an event data base has been defined (Appendix B, Documentation of the Extended Interactive Seismic Processing System (ISPSE), (Shaub, et. al., 1977)), the discrimination task may be performed by the following procedure:

- Request the execution of ISPSE from the DEC teletype as shown in Figure II-7.

```
.... .PROCS
       EVNTL*
       SPECT*
       BANDP*
       ISPED*
.... .LIST ISPED
 1.0 .PERF INTERUP   THIS PROCEDURE CALCULATES A SET OF VFM
 2.0 .PERF SELEV     EARTHQUAKE/EXPLOSION DISCRIMINANT VALUES.
 3.0 .PERF INTERUP   A HARD COPY IS MADE FOR EACH EVENT SHOWING
 4.0 .PERF SPEED     ITS LOG LOG SPECTRUM AND THE DISCRIMINANT
 5.0 .PERF INTERUP   PARAMETERS (SAVED ON FILE = SPEED PCM).
.... .LIST ISPED*
PROCEDURE MATRIX
  10   4   5   1   6   6   4   1   1   1   1   1   6   2   3
   4   1   1   4   1   1   4   6   3   5   4   1   1   1   1
   2   1   6   5   1   1   1   6   6   1   4   5   6   4
NUMERIC INPUT MATRIX
    C              0.1000000E+01  0.1000000E+01  0.1000000E+01
    B                             Y              0.0000000E+00
 0.0000000E+00  0.0000000E+00     N                 N
 0.1000000E+03     Y             -0.6000000E+00  0.6900000E+00
PROCEDURE FUNCTION ADDRESS TABLE
   1   2   7   8  44
NUMERIC FUNCTION ADDRESS TABLE
   0   1   0   6   0
```

FIGURE A-4

PROGRAM LISTING OF ISPED

```
TI-ISPSE LOGGED ON: 10/29/77 AT  0: 3:58, ENTER COMMAND OR .HELP

.... .LIST MDCRM
 1.0 C    THIS FUNCTION IS CALLED  MDCRM  .     IT PERFORMS A     *
 1.0 C    MULTIVARIATE EVENT DISCRIMINATION BASED ON MEAN         *
 1.0 C    DETECTABILITIES AS DESCRIBED BY  (SAX,1976).            *
 1.0 C    A SAMPLE EARTHQUAKE POP. TOGETHER WITH TWO REGIONAL     *
 1.0 C    PRESUMED EXPLOSION POPULATIONS ARE SPECIFIED BY CARDS.*
 1.0        VECTOR ( EQ( 10 ), SDQ( 10 ), EDX1( 10 ), EDX2( 10 ))
 2.0 C    EACH CARD CONTAINS ONE EVENT SEQUENCE NUMBER FROM THE   *
 2.0 C    DPSCAN EVENT LIST.  EQ AND SDQ VECTORS ARE COMPUTED     *
 2.0 C    FROM THE EARTHQUAKES AND USED TO COMPUTE THE MEAN       *
 2.0 C    DETECTABILITY VECTORS EDX1 (PRESUMED EXPLOSIONS FROM    *
 2.0 C    REGION 1) AND EDX2 (PRESUMED EXPLOSIONS FROM REGION 2)*
 2.0        VECTOR ( EP1( 100 ), EP2( 100 ), INT( 3 ) )
 3.0 C    DETECTABILITIY VECTORS FOR ALL EVENTS NOTED PLUS NEW    *
 3.0 C    EVENTS TO BE CLASSIFIED ARE THEN COMPUTED AND PROJECTED*
 3.0 C    ON EDX1 AND EDX2, YIELDING TWO VECTORS (EP1,EP2) THAT   *
 3.0 C    MAY BE PLOTTED (X,Y) FASHION, 1 POINT PER EVENT, THUS   *
 3.0 C    OBTAINING A MULTIVARIATE DISCRIMINANT PLOT (SAX,1976).*
 3.0        LABEL ( BLANK ( 1 ), LBL ( 10 ) )
 4.0        LABEL ( XLAB ( 10 ), YLAB ( 10 ) )
 5.0 C    NOTE: THE PROCEDURE  ISPED  MUST HAVE BEEN PERFORMED    *
 5.0 C    TO GENERATE DISCRIMINANTS FOR ALL EVENTS TO BE          *
 5.0 C    REFERENCED BY MDCRM PRIOR TO PERFORMING MDCRM.          *
 5.0        LABEL ( MESS ( 45 ) )
 6.0        BLANK = " "
 7.0        YLAB = "REGION 1  "
 8.0        XLAB = "REGION 2  "
 9.0        ND = 10
```

FIGURE A-5

LIST OF USER FUNCTION MDCRM
(PAGE 1 OF 4)

```
10.0        MESS = "INPUT NUMBER OF EARTHQUAKES - NQ
11.0       DISPLAY ( MESS )
12.0       MESS = "FOR AN EXPLAINATION OF MDCRM ENTER  0  FOR NQ
13.0       DISPLAY (MESS)
14.0       MESS = "THEN TYPE: ,LIST MDCRM  BEFORE RE-PERFORMING
15.0       DISPLAY (MESS)
16.0       INPUT ( NQ )
17.0       IF NQ=0 THEN JUMP TO $500
18.0       MESS = "INPUT NUMBER OF EVENTS IN REGION 1 - NX1
19.0       DISPLAY ( MESS )
20.0       INPUT ( NX1 )
21.0       MESS = "INPUT NUMBER OF EVENTS IN REGION 2 - NX2
22.0       DISPLAY ( MESS )
23.0       INPUT ( NX2 )
24.0       MESS = "INPUT OF EVENTS TO DISCRIMINATE - NED
25.0       DISPLAY ( MESS )
26.0       INPUT ( NED )
27.0       NE = NQ + NX1 + NX2 + NED
28.0       INT(1) = NQ + 1
29.0       INT(2) = INT(1) + NX1
30.0       INT(3) = INT(2) + NX2
31.0       MESS = "PLACE NQ + NX1 + NX2 CARDS IN CARD READER
32.0       DISPLAY ( MESS )
33.0       MESS = " HIT CARRIAGE RETURN TO CONTINUE
34.0       DISPLAY ( MESS )
35.0       INPUT (MESS)
36.0       QSTAT ( NQ, ND, EQ, SDQ )
37.0       DISPLAY ( BLANK )
```

FIGURE A-5

LIST OF USER FUNCTION MDCRM
(PAGE 2 OF 4)

```
TI-ISPSE LOGGED ON: 10/29/77 AT  0: 3:50; ENTER COMMAND OR .HELP

38.0        LBL = " EQ VECTOR"
39.0        DISPLAY ( LBL )
40.0        DISPLAY ( BLANK )
41.0        DISPLAY ( EQ )
42.0        DISPLAY ( BLANK )
43.0        LBL = "SDQ VECTOR"
44.0        DISPLAY ( LBL )
45.0        DISPLAY ( BLANK )
46.0        DISPLAY ( SDQ )
47.0        XSTAT ( NX1, ND, EQ, SDQ, EDX1 )
48.0        DISPLAY ( BLANK )
49.0        LBL = "EDX1 VECTR"
50.0        DISPLAY ( LBL )
51.0        DISPLAY ( BLANK )
52.0        DISPLAY ( EDX1 )
53.0        XSTAT ( NX2, ND, EQ, SDQ, EDX2 )
54.0        DISPLAY ( BLANK )
55.0        LBL = "EDX2 VECTR"
56.0        DISPLAY ( LBL )
57.0        DISPLAY ( BLANK )
58.0        DISPLAY ( EDX2 )
59.0        DISPLAY (BLANK)
```

FIGURE A-5

LIST OF USER FUNCTION MDCRM
(PAGE 3 OF 4)

```
60.0        MESS = "PUT NE = NQ+NX1+NX2+NED <=100 CARDS IN READER
61.0        DISPLAY ( MESS )
62.0        MESS = "HIT CARRIAGE RETURN TO CONTINUE
63.0        DISPLAY ( MESS )
64.0        INPUT (MESS)
65.0        DSTAT (NE, ND, EQ, SDQ, EDX1, EP1, EDX2, EP2 )
66.0        DISPLAY ( BLANK )
67.0        MESS = "EP1 VECTOR  (READ ONLY FIRST NE ELEMENTS)
68.0        DISPLAY (MESS)
69.0        DISPLAY ( BLANK )
70.0        DISPLAY ( EP1 )
71.0        DISPLAY ( BLANK )
72.0        MESS = "EP2 VECTOR  (READ ONLY FIRST NE ELEMENTS)
73.0        DISPLAY (MESS)
74.0        DISPLAY ( BLANK )
75.0        DISPLAY ( EP2 )
76.0        PLOTP ( EP2, EP1, NE, XLAB, YLAB, INT )
77.0         $500 CONTINUE
78.0        DISPLAY ( BLANK )
79.0        BLANK = "END  "
80.0        DISPLAY ( BLANK )
....
```

FIGURE A-5

LIST OF USER FUNCTION MDCRM
(PAGE 4 OF 4)

```
.... .MODE EXEC
 MODE = EXEC
.... .PERF ISPED
 USER PROCEDURE ISPED INITIATED
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST
.... 8      EARTHQUAKE
ENTER STATION NUMBER (1,15)
....    0.1000000E+01
ENTER COMPONENT NUMBER (1,3)
....    0.1000000E+01
ENTER NUMBER OF COPIES TO BE SAVED : TS =  2280   TL =    12 SEC
....    0.1000000E+01
ENTER EVENT SEQUENCE NUMBER FROM LIST
....
 SYSTEM FUNCTION SPEED INITIATED
T/Q (TOTAL) VALUE IN ABSORB =   0.000E+00   HIT CR TO CONTINUE
....
DO YOU WISH TO DECIMATE? (Y,N)
.... YES
ENTER REFLECTION COEFICIENT
....    0.2000000E+00
ENTER  SCATTERING COEFFICIENT   (0.0<SC<0.5)
....    0.5000000E+00
ENTER ECHO DELAY IN POINTS (NDEL)
....    0.2000000E+00
DO YOU WISH TO PROCESS SECONDARY REFLECTION? (Y OR N)

.... NO
ANALYZE RESIDUAL ?  (Y,N)
.... NO

ENTER POINT AT WHICH TO ZERO DATA (<128)
....    0.1000000E+03
ENTER 'Y' IF FREQS OK,OTHERWISE ENTER INDEX
.... YES
 ENTER LOWEST FREQUENCY TO BE DISPLAYED IN SPECTRUM (LOG FREQ)
....   -0.6000000E+00
ENTER HIGHEST FREQ TO BE DISPLAYED (MUST BE < 0.6990)
....    0.6900000E+00
 PAUSE AT  5.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST
.... 9      EARTHQUAKE
```

FIGURE A-6

EXECUTION OF ISPED
(PAGE 1 OF 4)

```
ENTER STATION NUMBER (1,15)
....    0.1020002E+01
ENTER COMPONENT NUMBER (1,3)
....    0.1030000E+01
ENTER NUMBER OF COPIES TO BE SAVED : TS =  2134   TL =    12 SEC
....    0.1000000E+01
ENTER EVENT SEQUENCE NUMBER FROM LIST
....
 SYSTEM FUNCTION SPEED INITIATED
T/Q (TOTAL) VALUE IN ABSORB =   0.0000E+00   HIT CR TO CONTINUE
....
DO YOU WISH TO DECIMATE? (Y,N)
.... YES
ENTER REFLECTION COEFICIENT
....    0.7000000E+00
ENTER  SCATTERING COEFFICIENT   (0.0<SC<0.5)
....    0.0000000E+00
ENTER ECHO DELAY IN POINTS (NDEL)
....    0.0000000E+00
DO YOU WISH TO PROCESS SECONDARY REFLECTION? (Y OR N)
.... NO
ANALYZE RESIDUAL ?  (Y,N)
.... NO

ENTER POINT AT WHICH TO ZERO DATA (<128)
....    0.1000000E+03
ENTER 'Y' IF FREQS OK,OTHERWISE ENTER INDEX
.... YES
 ENTER LOWEST FREQUENCY TO BE DISPLAYED IN SPECTRUM (LOG FREQ)
....   -0.6000000E+20
ENTER HIGHEST FREQ TO BE DISPLAYED (MUST BE < 0.6990)
....    0.6900000E+00
 PAUSE AT  5.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST
.... 31    PRESUMED EXPLOSION
ENTER STATION NUMBER (1,15)
....    0.1020000E+01

ENTER COMPONENT NUMBER (1,3)
....    0.1030000E+01
ENTER NUMBER OF COPIES TO BE SAVED : TS =   440   TL =    12 SEC
....    0.1000000E+01
ENTER EVENT SEQUENCE NUMBER FROM LIST
....
 SYSTEM FUNCTION SPEED INITIATED
```

FIGURE A-6

EXECUTION OF ISPED
(PAGE 2 OF 4)

```
T/S (TOTAL) VALUE IN ABSORB =   0.000E+00   HIT CR TO CONTINUE
....
DO YOU WISH TO DECIMATE? (Y,N)
.... YES
ENTER REFLECTION COEFICIENT
....   0.7000000E+00
ENTER  SCATTERING COEFFICIENT   (0.0<SC<0.5)
....   0.3000000E+00
ENTER ECHO DELAY IN POINTS (NDEL)
....   0.0020002E+02
DO YOU WISH TO PROCESS SECONDARY REFLECTION? (Y OR N)
.... NO
ANALYZE RESIDUAL ?  (Y,N)
.... NO

ENTER POINT AT WHICH TO ZERO DATA (<128)
....   3.1000000E+03
ENTER 'Y' IF FREQS OK,OTHERWISE ENTER INDEX
.... YES
 ENTER LOWEST FREQUENCY TO BE DISPLAYED IN SPECTRUM (LOG FREQ)
....  -0.6020000E+00
ENTER HIGHEST FREQ TO BE DISPLAYED (MUST BE < 0.6990)
....   0.6900000E+00
 PAUSE AT  5.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 SYSTEM FUNCTION SELEV INITIATED
ENTER EVENT SEQUENCE NUMBER FROM LIST
.... 35    PRESUMED EXPLOSION
ENTER STATION NUMBER (1,15)
....   0.1000000E+01
ENTER COMPONENT NUMBER (1,3)
....   0.1000000E+01
ENTER NUMBER OF COPIES TO BE SAVED : TS =   439   TL =    12 SEC
....   0.1000000E+01
ENTER EVENT SEQUENCE NUMBER FROM LIST
....
 SYSTEM FUNCTION SPEED INITIATED
T/S (TOTAL) VALUE IN ABSORB =   0.000E+00   HIT CR TO CONTINUE
....
DO YOU WISH TO DECIMATE? (Y,N)
.... YES
ENTER REFLECTION COEFICIENT
....   0.7000000E+00
ENTER  SCATTERING COEFFICIENT   (0.0<SC<0.5)
....   0.3000000E+00
ENTER ECHO DELAY IN POINTS (NDEL)
....   0.0033000E+02
DO YOU WISH TO PROCESS SECONDARY REFLECTION? (Y OR N)
.... NO
```

FIGURE A-6

EXECUTION OF ISPED
(PAGE 3 OF 4)

```
ANALYZE RESIDUAL ?  (Y,N)
.... NO

ENTER POINT AT WHICH TO ZERO DATA (<128)
....    0.1000000E+03
ENTER 'Y' IF FREQS OK,OTHERWISE ENTER INDEX
.... YES
 ENTER LOWEST FREQUENCY TO BE DISPLAYED IN SPECTRUM (LOG FREQ)
....  -2.6000000E+00
ENTER HIGHEST FREQ TO BE DISPLAYED (MUST BE < 0.6990)
....    0.6990120E+00
 PAUSE AT  5.4 : #4 RESTART, #5 CONTINUE, #6 STOP
 PAUSE AT  1.0 : #4 RESTART, #5 CONTINUE, #6 STOP
 AUTOLOOP COMPLETED, RETURNING TO SUPERVISOR
```

FIGURE A-6

EXECUTION OF ISPED
(PAGE 4 OF 4)

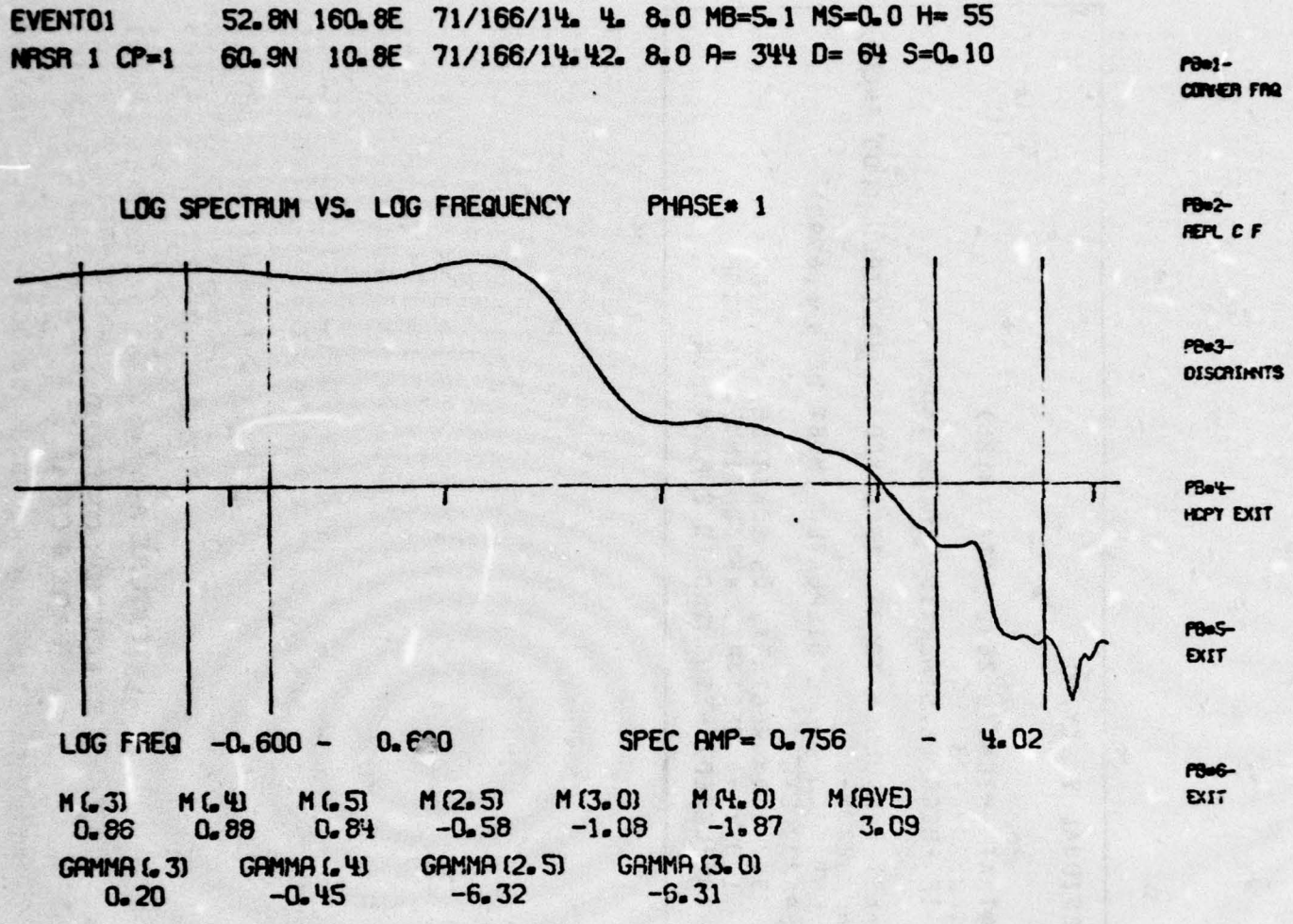
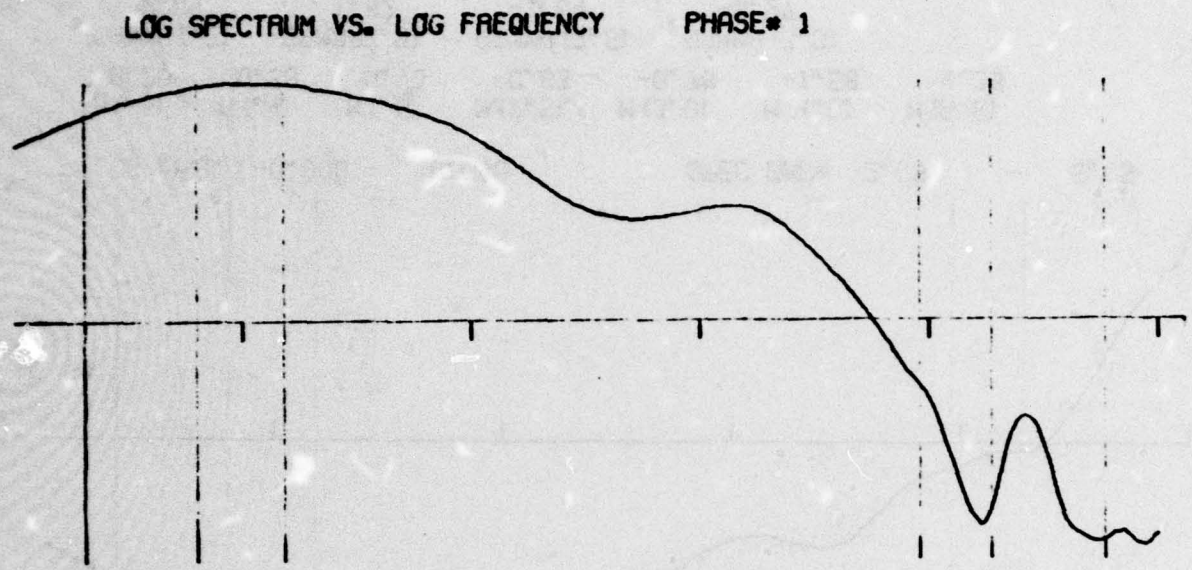


FIGURE A-7

HARDCOPY FROM ISPED EXECUTION
(PAGE 1 OF 4)

EVENT02 41.5N 78.3E 71/170/17.23. 2.0 MB=5.2 MS=0.0 H= 33
NRSR 1 CP=1 60.9N 10.8E 71/170/17.58.36.0 A= 360 D= 44 S=0.10



PB=1- CORNER FRQ
PB=2- REPL C F
PB=3- DISCRIMNTS
PB=4- HCPY EXIT
PB=5- EXIT
PB=6- EXIT

M(.3)	M(.4)	M(.5)	M(2.5)	M(3.0)	M(4.0)	M(AVE)
0.66	0.83	0.83	-0.95	-1.91	-2.08	3.79

GAMMA(.3)	GAMMA(.4)	GAMMA(2.5)	GAMMA(3.0)
1.32	0.06	-12.06	-1.36

FIGURE A-7

HARDCOPY FROM ISPED EXECUTION
(PAGE 2 OF 4)

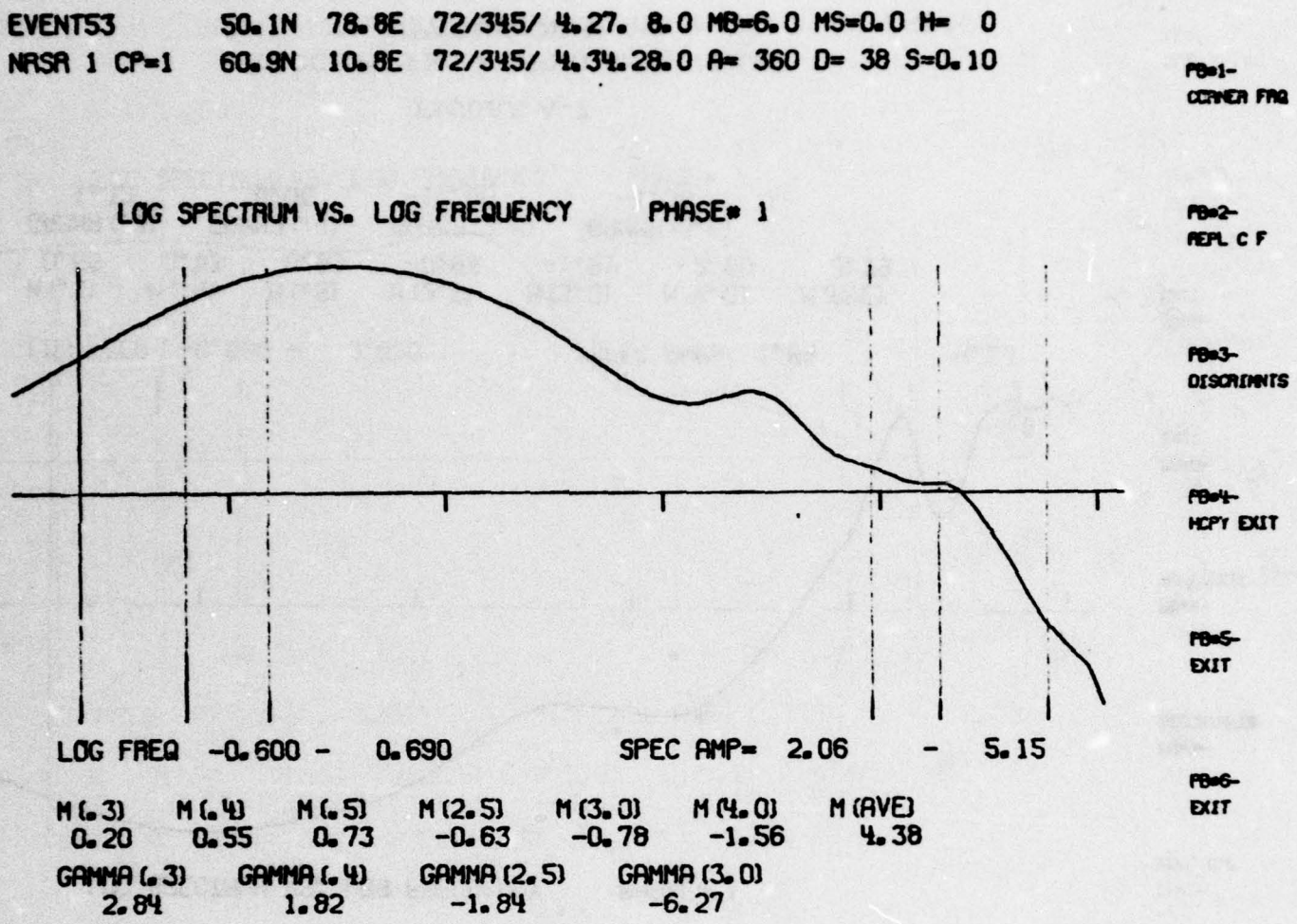


FIGURE A-7

HARDCOPY FROM ISPED EXECUTION
(PAGE 3 OF 4)

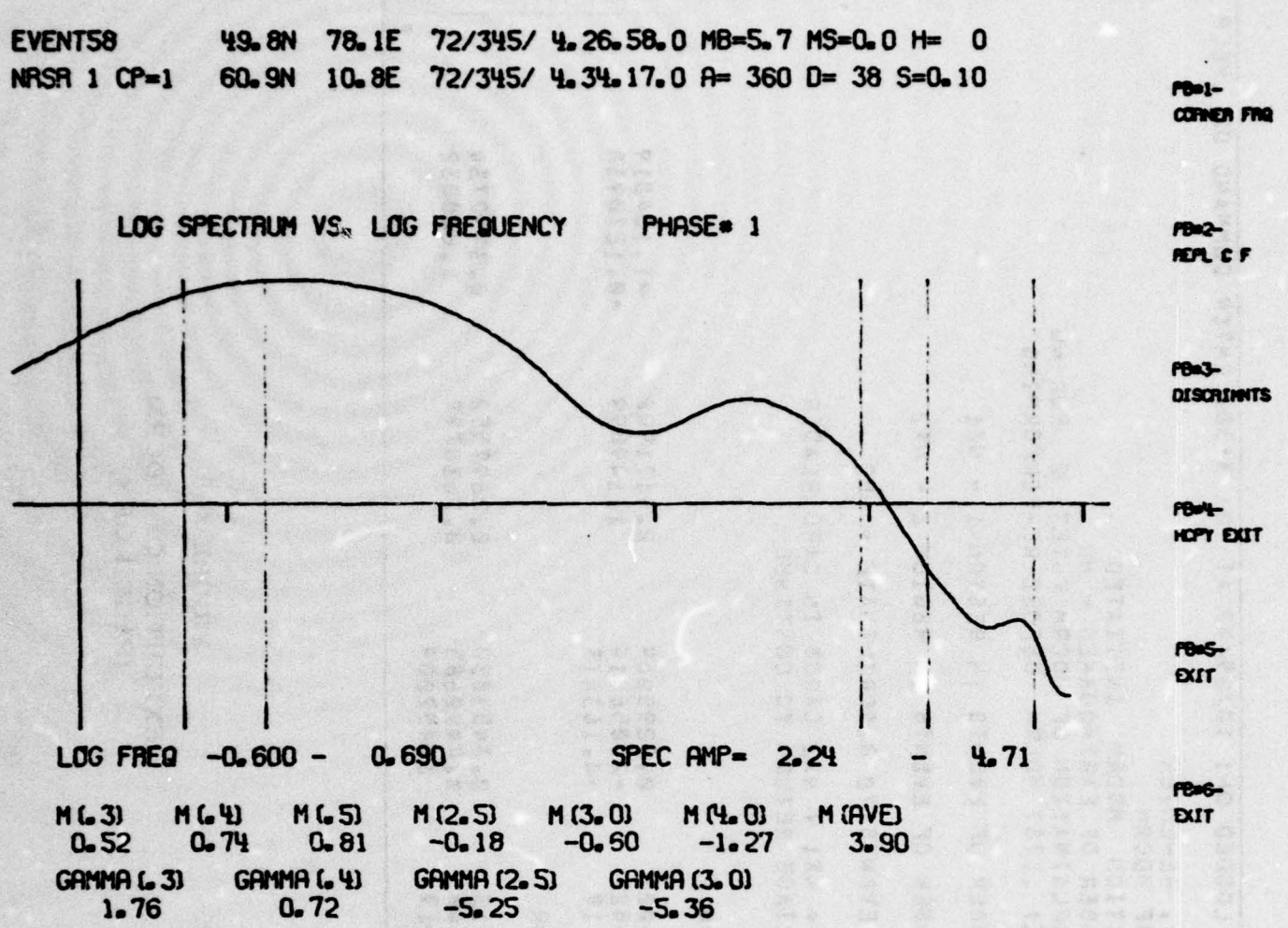


FIGURE A-7

HARDCOPY FROM ISPED EXECUTION
(PAGE 4 OF 4)

```
TI-ISPSE LOGGED ON: 10/29/77 AT  0: 3:58, ENTER COMMAND OR .HELP

 NO ENTRY, RE-ENTER
.... .PERF MDCRM
USER FUNCTION  MDCRM  INITIATED
INPUT NUMBER OF EARTHQUAKES - NQ
FOR AN EXPLAINATION OF MDCRM ENTER  0  FOR NQ
THEN TYPE: .LIST MDCRM  BEFORE RE-PERFORMING
.... 20
INPUT NUMBER OF EVENTS IN REGION 1 - NX1
.... 10
INPUT NUMBER OF EVENTS IN REGION 2 - NX2
.... 3
INPUT OF EVENTS TO DISCRIMINATE - NED
.... 2
PLACE NQ + NX1 + NX2 CARDS IN CARD READER
 HIT CARRIAGE RETURN TO CONTINUE
....

 EQ VECTOR

  0.7858400       0.9295069       0.9171902       -1.134819
  -1.536642       -2.056814        1.149899       -0.1270938
  -5.074719       -4.163415

SDQ VECTOR

  0.4098013       0.3453830       0.2697552       0.3752734
  0.4608360       0.4498685       0.7638799        1.028832
   3.839619        3.452809
```

FIGURE A-8

EXECUTION OF MDCRM
(PAGE 1 OF 4)

```
EDX1 VECTR

 -0.3342091      -0.4214653      -0.5621967       2.055851
  1.794391        1.556355      -0.9019071E-01 -0.6106611E-01
  0.1822653      -0.2968068

EDX2 VECTR

  0.3595534       0.9046804       1.686616      -0.6004922
 -0.2889020       1.781709        1.730081       1.429350
  0.3033041       2.163850

PUT NE = NQ+NX1+NX2+NED <=100 CARDS IN READER
HIT CARRIAGE RETURN TO CONTINUE
....

EP1 VECTOR  (READ ONLY FIRST NE ELEMENTS)

  0.5536903      -0.7113819E-01  0.3537094       0.4063516
 -0.6456922      -0.2816944E-01  0.2975856      -0.9553750
 -0.9923344E-02  -0.3659358     -0.2145738       0.3801536E-01
  0.2685915       0.5087083       0.1729497       0.6669283
 -0.1574269E-01  -0.4743943E-01 -0.5218757      -0.3906645
  1.214968        1.131430        0.8358501       0.8915140
  0.7552553       1.032534        0.8250048       0.9215746
  1.151375        1.240495        0.3746443      -0.3362205
 -0.3947599       0.9527778E-01   0.3641596       0.0000000
```

FIGURE A-8

EXECUTION OF MDCRM
(PAGE 2 OF 4)

```
TI-ISPSE LOGGED ON: 10/29/77 AT  0: 3:58, ENTER COMMAND OR .HELP

 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
 0.0000000      0.0000000      0.0000000      0.0000000
```

FIGURE A-8

EXECUTION OF MDCRM
(PAGE 3 OF 4)

```
EP2 VECTOR  (READ ONLY FIRST NE ELEMENTS)

 -0.2909914       0.4684671E-01 -0.6027189       0.1952799
  0.2649158      -0.6172361E-03  0.3140514E-01   0.1271972
  0.2753761E-02  -0.2175316     -0.1973341E-01  -0.1285973
 -0.1716325      -0.1867232      0.2077613       0.1222428
 -0.4942878       0.6070854      0.3139115       0.1934337
  0.1797988E-01  -0.6527032      0.3235601E-02  -0.3658457
  0.1329124      -0.1438661      0.2006909       0.1310275
  0.8421580E-01  -0.1392404       1.342357       0.6230428
   1.034600       0.2022359E-02 -0.4089877       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000
  0.0000000       0.0000000      0.0000000       0.0000000

END
.... .PERF LOGOFF



 TI-ISPS LOGGED OFF : 10/29/77 AT  0:11:52
```

FIGURE A-8

EXECUTION OF MDCRM
(PAGE 4 OF 4)

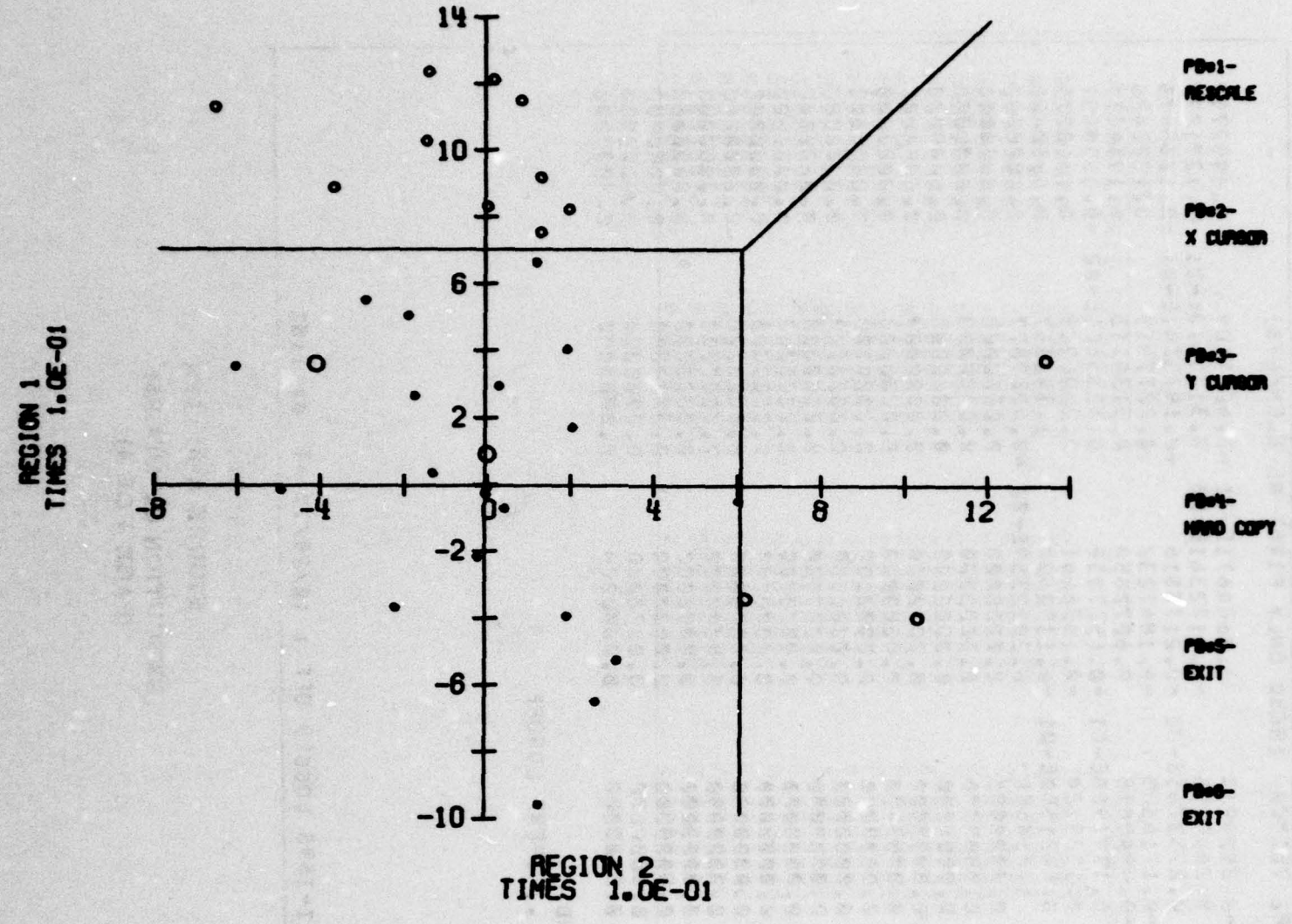


FIGURE A-9

HARDCOPY FROM MDCRM EXECUTION

- Move to the CRT console and proceed as shown, user entries following .... and terminating with a carriage return:

  .... .MODE EXEC

  MODE = EXEC

  .... .PERF EVNTL (A procedure which produces an event list)

  USER PROCEDURE EVNTL INITIATED

  SYSTEM FUNCTION DPSCAN INITIATED

  (This gives a listing of the event data base.)

  .... .MODE PRGM

  MODE = PRGM

  .... .PERF ISPED

  USER PROCEDURE ISPED INITIATED

  PAUSE AT 1.0: #4 RESTART, #5 CONTINUE, #6 STOP

  (Immediately hit key #6 to STOP; the system will then prompt:)

  AUTOLOOP REQUESTED, ENTER NUMBER OF TIMES TO LOOP.

  4

  PROGRAM OK FOR PROCEDURE ISPED

  (This reflects the number (N) of events to process; here N=4 was entered.)

- Place N cards indicating sequence numbers (one per card) of events to process in the card reader and push RESET. These sequence numbers are obtained from the event data base listing. At this point on the CRT:

  .... .MODE EXEC

  MODE = EXEC

  .... .PERF ISPED

  (This will cause the procedure to execute (N) times: i.e., (N) events will be processed. Each event requires approximately

1 minute 45 seconds to process; hence total execution time in seconds is 105*N.)

When ISPED completes executing (i.e., when the message "AUTOLOOP COMPLETE, RETURNING TO SUPERVISOR" is received on the CRT), the MDCRM user function may be .PERFORM'ed. At this time, all events that are to be used in MDCRM must have been processed, but not necessarily at this time. They may be processed anytime prior to the execution of MDCRM. Proceed as follows:

```
.... .MODE NORM
MODE = NORM
.... .PERF MDCRM
USER FUNCTION MDCRM INITIATED.
INPUT NUMBER OF EARTHQUAKES - NQ
.... 20
(This is the number of earthquakes in the sample population.)
INPUT NUMBER OF EVENTS IN REGION 1 - NX1.
.... 10
(This is the number of presumed explosions from region 1,)
INPUT NUMBER OF EVENTS IN REGION 2 - NX2
.... 3
(This is the number of presumed explosions from region 2.)
INPUT NUMBER OF EVENTS TO DISCRIMINATE - NED
.... 2
(This number represents the unknown events.)
PLACE NQ+NX1+NX2 CARDS IN CARD READER
HIT CARRIAGE RETURN TO CONTINUE
....
```

- Place input cards in the card reader and push RESET. Each card contains one event number from the event data base listing.
  (A table of the vectors EQ, SDQ, EDX1, and EDX2 as described in Section III is displayed on the CRT.)
  At this point on the CRT:
  PUT NE = NQ+NX1+NX2+NED<= 100 CARDS IN READER
  HIT CARRIAGE RETURN TO CONTINUE
  ....

- Take the cards from the output hopper on the card reader, add the cards containing the event numbers of the events to be discriminated, and replace them in the input hopper. Push RESET.
  (A table of the projection vectors, EP1 and EP2, is displayed on the CRT.)

- When the X-Y plot of the projection vectors appears on the CRT push key #4 to obtain a hardcopy.

At this time the Calcomp plot tape generate by ISPED and MDCRM may be plotted using the DEC console command:

MCR>REQ PLTPDP

(Note: the generated plot tape should be mounted on tape drive MT1.) If desired, ISPSE may be terminated via the .PERF LOGOFF command.

For a detailed description and interpretation of the techniques for analysis and the discriminant parameters provided on the hardcopy output, refer to Technical Report No. 9 (Sax, 1975).

APPENDIX B

ISPSE ERROR SUMMARY

This appendix presents a summary of error messages, their cause, and the remedy. The errors are divided into five sections which are:

- ISPSE general errors
- ISPSE command language errors
- ISPL syntax errors
- ISPL semantic errors
- ISPL execution time errors

Unless otherwise noted, all errors are displayed on the CRT screen.

A. ISPSE GENERAL ERRORS

These errors may occur any time the user is entering numbers. This could be within a processing module, as an edit command, or as part of an ISPL statement.

Error: INTEGER TOO LARGE, RE-ENTER
Cause: The user has typed an integer whose value is greater than 131071 or less than -131071.
Remedy: Correct the number.

Error: TOO MANY SIGNIFICANT DIGITS, RE-ENTER
Cause: More than seven significant digits were entered.
Remedy: Enter only seven significant digits.

Error: TOO LARGE AN EXPONENT, RE-ENTER

Cause: The exponent part of a real number is larger than 75.

Remedy: Correct the number.

Error: INVALID NUMBER, RE-ENTER

Cause: A number contains a symbol other than a digit, sign, decimal point, or letter E, or the number is improperly constructed.

Remedy: Correct the number.

Error: NO ENTRY, RE-ENTER.

Cause: A carriage return was keyed in without any other characters being typed.

Remedy: Enter the correct number.

B. ISPSE COMMAND ERRORS

The errors listed in this sub-section occur when the user enters ISPSE commands.

Error: NON-EXISTENT USER PROCEDURE, RE-ENTER.

Cause: The user has attempted to EDIT a non-existent procedure, DELETE or LIST a non-existent procedure or user function.

Remedy: Correct the name of the procedure or user function.

Error: NON-EXISTENT PROCESSING MODULE OR USER PROCEDURE, RE-ENTER.

Cause: The user has tried to . PERFORM a non-existent system function, user function, or procedure.

Remedy: Correct the specified name.

Error: PROCEDURE CREATION LIMIT EXCEEDED, COMMAND IGNORED

Cause:	The user has attempted to create more than 36 procedures.
Remedy:	Delete or modify an existing procedure.
Error:	MNEMONIC ALREADY IN USE BY SYSTEM OR USER, RE-ENTER
Cause:	The user has tried to create a procedure or define a user function with the same name as an existing procedure, user function, or system function.
Remedy:	Choose another name for the procedure or user function.
Error:	USER FUNCTION CREATION LIMIT EXCEEDED, COMMAND IGNORED
Cause:	The user has attempted to define more than 36 user functions.
Remedy:	Delete or modify an existing user function.
Error:	USER PROCEDURE IS NOT PROGRAMMED, CHANGE MODE OR PROGRAM
Cause:	The user has tried to execute a procedure in EXEC mode that is not programmed.
Remedy:	Change mode to PRGM and program the procedure or execute the procedure in NORM mode.
Error:	USER FUNCTION ISN'T COMPILED SUCCESSFULLY-COMMAND IGNORED.
Cause:	The user has attempted to PERFORM a non-compiled user function individually or as part of a procedure.
Remedy:	. COMPILE the user function and exit via the #S command.
Error:	INCORRECT MODE - RESET MODE TO NORM
Cause:	The user has entered a command other than . PERFORM procedure name or . MODE in PRGM or EXEC mode.
Remedy:	Change the mode to NORM.

Error:	BAD COMMAND, RE-ENTER
Cause:	The user has entered an unrecognizable command or used .MODE or .HELP with a bad keyword.
Remedy:	Correct the command.
Error:	NO ENTRY, RE-ENTER
Cause:	The user has keyed in a carriage return without typing a command.
Remedy:	Enter a valid command.
Error:	MAXIMUM PROCEDURE LENGTH ATTAINED, .CREATE EXITS
Cause:	The user has tried to enter more than 25 statements in a procedure.
Remedy:	Subdivide the procedure into two procedures.

C. ISPL SYNTAX ERRORS

The errors listed in this subsection occur as the user enters ISPL statements.

Error:	INVALID LABEL
Cause:	An invalid <statement label> was entered, e.g., $12A.
Remedy:	Correct the <statement label>.
Error:	INVALID FUNCTION
Cause:	An invalid <mathematical function> was entered, i.e., not &SIN, &COS, &TAN, &ATN, &EXP, &LNN, &LOG, &SQR, &ABS, or, &PIM.
Remedy:	Correct the <mathematical function> name.
Error:	INVALID IDENTIFIER OR KEYWORD
Cause:	An <identifier> contains a character other than a letter or digit.

Remedy:	Correct the <identifier>.
Error:	INVALID CONDITIONAL STATEMENT
Cause:	A <conditional statement> is syntactically invalid.
Remedy:	Correct the statement.
Error:	INVALID ASSIGNMENT STATEMENT
Cause:	An <assignment statement> is in error.
Remedy:	Correct the statement.
Error:	ILLEGAL ARITHMETIC EXPRESSION - SYNTAX ERROR
Cause:	An <arithmetic expression> in an <assignment statement> or <conditional statement> is invalid.
Remedy:	Correct the statement.
Error:	SYNTAX ERROR IN VARIABLE LIST
Cause:	The <variable list> in an <I/O statement> or <numerical operator> is invalid or the <declaration list> in a <declaration statement> is incorrect.
Error:	TOO MANY CONTINUATION LINES, MAX CMNT=6, MAX CODE=4
Cause:	More than 6 comment or 4 statement lines are entered as part of one ISPL statement.
Remedy:	Construct two or more statements from the one large statement.
Error:	ILLEGAL EDIT COMMAND - NOT #P, #D, #I, #N, #Q, #S, #B, or #W
Cause:	The first character entered is a # sign but it is not an edit command.
Remedy:	Re-enter the correct command.

Error: (displayed on the operator's console) LEXICL TABLE SPACE EXHAUSTED, SYSTEM ERROR=125

Cause: More than 100 uniform symbols have been entered as part of one ISPL statement.

Remedy: Divide the statement into two or more smaller statements.

Error: UNRECOGNIZABLE STATEMENT - SYNTAX ERROR

Cause: An ISPL statement is syntactically incorrect but the error does not fit any of the above error messages.

Remedy: Correct the statement.

D. ISPL SEMANTIC ERRORS

These errors appear during semantic error analysis of a user function, i.e., #S. The <statement number> refers to the absolute statement rather than the executable statement as in execution time errors.

Error: <statement number> VARIABLE <identifier> NOT A SCALAR OR NOT INITIALIZED

Cause: The first executable statement in a user function contains a variable on the right hand side of an equal sign which is not a scalar or which hasn't been initialized.

Remedy: Correct the statement or insert an <assignment statement> which initializes the scalar.

Error: <statement number> VARIABLE <identifier> NOT A VECTOR

Cause: A statement contains a subscripted <identifier> which is not declared as a vector or a statement should have a vector name and it is a scalar or label

Remedy: Correct the statement or declare the variable to be a vector.

Error:	<statement number> VARIABLE <identifier> PREVIOUSLY DECLARED, ALLOCATION IGNORED
Cause:	A label or vector name appears in two or more declaration statements.
Remedy:	Delete one of the declarations of the variable.
Error:	<statement number> DECLARATION ERROR, ALLOCATION MUST BE > 0.
Cause:	A vector or label was declared to be length 0.
Remedy:	Correct the statement.
Error:	<statement number> INSUFFICIENT MEMORY TO A ALLOCATE <identifier>
Cause:	The total vector lengths and labels in declaration statements require more than 1957 memory elements.
Remedy:	Correct the declaration statements.
Error:	<statement number> DUPLICATE STATEMENT LABEL <statement label>.
Cause:	A <statement label> appears in more than one <label statement>.
Remedy:	Change one of the <statement labels> or delete one <label statement>.
Error:	<statement number> VARIABLE <identifier> DIFFERS IN MODE FROM EXPRESSION
Cause:	The left hand side of an <assignment statement> is not the same type as the right hand side. A vector or scalar is set equal to a <label constant> or a label or a non-subscripted vector is set equal to an <arithmetic expression>.
Remedy:	Correct the variable type or the statement.

Error:	<statement number> VARIABLE <identifier> NOT A SCALAR
Cause:	A statement which expects a scalar variable contains a vector or label variable name.
Remedy:	Correct the variable type or the statement.
Error:	<statement number> VARIABLE <identifier> NOT A 5 CHARACTER FILENAME LABEL
Cause:	The filename parameter in a DREAD or DWRIT statement is not a 5 character label.
Remedy:	Correct the variable type or the <I/O statement>.
Error:	<statement number> MIS-PLACED DECLARATION STATEMENT
Cause:	A declaration statement appears after the first executable statement.
Remedy:	Place the declaration statement at the beginning of the user function.
Error:	<statement number> NON-EXISTENT STATEMENT LABEL <statement label>
Cause:	A <branch statement> refers to a non-existent <statement label>.
Remedy:	Correct the <branch statement> or add a <label statement>.
Error:	(displayed on the operator's console) LEXICL TABLE SPACE EXHAUSTED, SYSTEM ERROR = <integer>
Cause:	System error=122, more than 25 <statement label>'s have been used. System error=123, more than 124 <number>'s have been used.

System error=124, more than 90 <identifier>'s have been used.

System error=126, more than 920 total characters have been used in <label> constants.

Remedy: Decrease the number of <statement labels>, constants, or variables.

Error: UNSUCCESSFUL COMPILATION FOR <user name> RETURNING TO EDITOR

Cause: A semantic error was discovered in the user function.

Remedy: Correct the error and recompile or save without compiling, #Q.

Message: NO ERRORS FLAGGED IN <user name>, SUCCESSFUL COMPILATION

Cause: The user function is successfully compiled, the executable code file is saved. The user function can then be .PERFORM'ed.

E. ISPL EXECUTION TIME ERRORS

The errors listed in this subsection may occur during the execution of user functions, whether performed individually or as a part of a procedure. <statement number> refers to the executable statement number where the error occurred, excluding declaration statements. Therefore, it is generally less than the absolute statement number displayed when a user function is listed.

Error: (displayed on the operator's console.) SYSTEM ERROR - PROGRAM LOAD SPACE EXHAUSTED. REQUESTED USER FUNCTION WILL EXECUTE THRU STATEMENT <statement number>

Cause:	The user function is too large to load into the allocated memory.
Remedy:	Divide the user function into two smaller ones. Communication between the two new user functions may be accomplished via DREAD and DWRIT statements.
Error:	FATAL ERROR... ... ... SUBSCRIPT EXCEEDS VECTOR LENGTH IN ASIGMNT, EXECUTABLE STMN# <statement number>
Cause:	Vector index on left hand side of an assignment statement is less than one or greater than the vector length.
Remedy:	Increase vector allocation in declaration statement or correct program logic.
Error:	FATAL ERROR... ... ...
Cause:	SUBSCRIPT EXCEEDS VECTOR LENGTH IN ARITH. EXPR, EXECUTABLE STMN# <statement number>
Cause:	Vector index on right hand side of an assignment statement is less than one or greater than the vector length.
Remedy:	Increase the vector allocation in declaration statement or correct program logic.
Error:	FATAL ERROR... ... ... DIVISION BY ZERO ATTEMPTED IN ARITH. EXPR., EXECUTABLE STMN# <statement number>
Cause:	The second argument to the binary operator, /, is zero.
Remedy:	Correct program logic.
Error:	FATAL ERROR... ... ... ARGUMENT NEGATIVE FOR MATH FUNCTION, EXECUTABLE STMN# <statement number>

Cause: The argument to the math function, &SQR, is negative.

Remedy: Correct program logic.

Error: FATAL ERROR... ... ...
ARGUMENT NEG OR ZERO FOR MATH FUNCTION,
EXECUTABLE STMN# <statement number>

Cause: The argument to one of the math functions, &LNN or &LOG, is out of range; the first operand to the exponentiation operator, ^, is not positive and the second operand is not an integer; or the first operand is zero and the second is not a positive integer.

Remedy: Correct program logic.

The following list of errors may occur during the use of <I/O statement>'s or <numerical operator>'s. They are categorized under the name of the particular statement or operator where they occur.

GET and PUT:

Error: BAD ARGUMENT FOR PUT OR GET, NO DISK I/C PERFORMED

Cause: The specified record number is not in the range (1-16); the length of the specified vector is less than 548 elements; or the length of the specified label is less than 55 characters.

Remedy: Increase the vector or label allocation in declaration statement or correct program logic.

DREAD and DWRIT:

Error: NO NAME IN LABEL FOR DREAD OR DWRIT, NO I/O PERFORMED

Cause: The label argument for DREAD or DWRIT is not initialized via an assignment statement.

Remedy: Initialize the 5 character label variable to a <label constant >.

Error: BAD RECORD # FOR DREAD OR DWRIT, NO I/O PERFORMED

Cause: The record number argument input to DREAD or DWRIT is not in the range (1-100).

Remedy: Correct program logic.

PLOT, PLOTP, and PLOTX

<name> in the following errors is replaced by PLOT, PLOTP, or PLOTX depending on the particular operator where the error occurred.

Error: INCORRECT NUMBER OF ARGUMENTS INPUT TO <name>

Cause: Too few or too many arguments are in the variable list.

Remedy: Correct the variable list.

Error: ARGUMENTS OF <name> ARE THE WRONG TYPE

Cause: The arguments of the operator are not specified in the correct order or a vector, scalar, or label variable is specified as a different type.

Remedy: Correct the variable list.

Error: N=<integer>, TRUNCATED TO 512 POINTS FOR <name>

Cause: More than 512 points were requested to be plotted. <integer> is the current value specified.

Remedy; Correct the value of the argument.

Error: VECTOR LENGTHS FOR <name> ARE TOO SHORT, SET TO <integer>

Cause: One or more of the vectors is shorter than the number of points requested to be plotted. <integer> is the number of points actually plotted.

Remedy: Increase the allocation for the vector, or change the number of points to be plotted.

DSTAT, QSTAT, and XSTAT

<name> is replaced by DSTAT, QSTAT, or XSTAT, according to which operator caused the error.

Error: SYSTEM ERROR <name>: EVENT SEQ # NOT IN RANGE (1-100)

Cause: An event sequence number on the card input is not valid.

Remedy: Correct the card in error and re-execute the user function.

Error: INCORRECT NUMBER OF ARGUMENTS INPUT TO <name>

Cause: Too few or too many arguments are specified in the particular statement.

Remedy: Correct the variable list.

Error: ARGUMENTS OF <name> ARE THE WRONG TYPE

Cause: A vector or scalar argument was specified as a different type.

Remedy: Correct the variable list.

Error: VECTORS INPUT TO <name> ARE TOO SHORT

Cause:	The vector lengths input to the operator are too short to perform the function.
Remedy:	Increase the allocation for the vector.
Error:	NUMBER OF DISCRIMINANTS IS ZERO OR GREATER THAN 125 IN <name>
Cause:	The user has requested an invalid number of discriminant values for this operator. At this time only 10 values are calculated by SPEED.
Remedy:	Correct the specified values.
Error:	NUMBER OF EVENTS IS ZERO OR GREATER THAN 100 IN DSTAT.
Cause:	Invalid number of events specified for DSTAT.
Remedy:	Correct the specified value.
Error:	NUMBER OF EVENTS INPUT TO QSTAT IS LESS THAN 2
Cause:	Invalid number of events specified. At least two are required to calculate the standard deviation vector.
Remedy:	Correct the specified value.
Error:	NUMBER OF PRESUMED EXPLOSIONS IS ZERO IN XSTAT
Cause:	Invalid number of presumed explosions specified.
Remedy:	Correct the specified value.

APPENDIX C
ISPSE INDEX