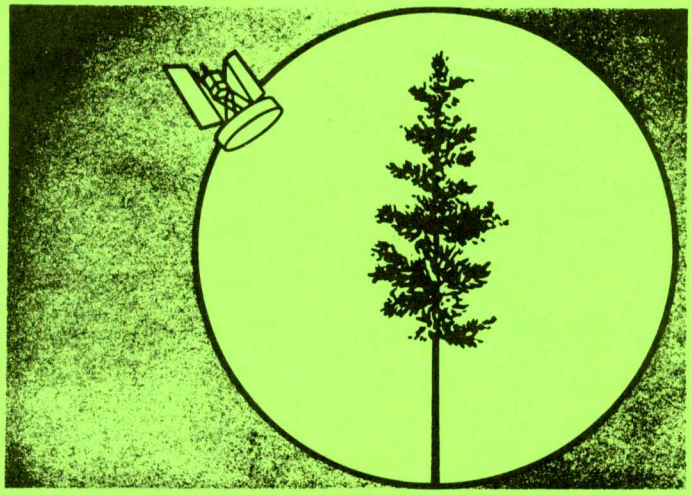


FRIS

FOREST RESOURCE INFORMATION SYSTEM

NASA ST. REGIS LARS

LARSFRIS USER'S MANUAL

Volume 5

Purdue University
Laboratory for Applications
of Remote Sensing

LARS Contract
Report No. 100580
October 1, 1980

Star Information Form

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
100580		

4. Title and Subtitle	5. Report Date
LARSFRIS User's Manual Volume 5	October 1, 1980
	6. Performing Organization Code
7. Author(s) LARS Staff, R.P. Mroczynski, editor	8. Performing Organization Report No.
9. Performing Organization Name and Address Laboratory for Applications of Remote Sensing Purdue University West Lafayette, IN 47906	10. Work Unit No.
	11. Contract or Grant No. NAS9-15325
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address R.E. Joosten/SF5 NASA/Johnson Space Center Houston, TX 77058	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This document contains user instructions for the proper use and application of the Software which comprises the LARSFRIS package. LARSFRIS represents a compilation of software developed over a number of years by the staff at Purdue University's Laboratory for Applications of Remote Sensing. The software packages are designed to help the user analyses digital image data, such as that collected by the Landsat Multispectral scanner. This is one of five documents that comprise the LARSFRIS package.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Landsat analysis Digital Image data User Documentation Software packages	

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

*For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA - JSC

ACKNOWLEDGEMENTS

An undertaking of the magnitude of the LARSFRIS documentation, albeit only a modification to existing materials, depended on the individual dedication of many people. A number of LARS staff contributed to updating LARSYS ver. 3.1 and integrating LARSYSDV (developmental software) into the final LARSFRIS software package.

LARS staff who made significant contributions to either creating new or updating existing program modules included; Sue Schwingendorf, Bill Shelley, Carol Jobusch, Joan Buis, Luis Bartolucci, Louis Lang, and John Cain. Kay Hunt deserves special thanks for coordinating and organizing staff efforts.

Typing of the manuscript for the LARSFRIS documentation was ably handled by; Dee Dee Dexter, Sylvia Johnston, Pam Burroff, and Bonnie Phibbs. Assistance in editorial matters was provided by Doug Morrison and Davida Parks, and Sue Ferringer provided graphic inputs.

Special thanks are also appropriate for members of the FRIS Steering Committee; especially G. R. Barker of the St. Regis Paper Company, and R. E. Joosten of the National Aeronautics and Space Administration, for their patience and sage council during the preparation of these volumes.

Preparation of this documentation was supported by NASA Contract NAS 9-15325.

PREFACE

The documentation of the LARSFRIS system closely parallels existing LARSYS Version 3.1 documentation. The major differences are in the addition of certain program modules which provide the user greater flexibility in the analysis of multispectral data. The LARSFRIS documentation exists in three parts: LARSFRIS Program Abstracts, LARSFRIS System Manual, and LARSFRIS User's Manual.

The first of these contains the documentation of each Fortran and Assembler routine and each CMS Executive routine in LARSFRIS. These program abstracts are provided for programmers who are required to revise and/or maintain these routines.

The second manual, LARSFRIS System Manual, is directed primarily to programmers and analysts who maintain or revise the system or write new functions that must be interfaced with LARSYS. It contains detailed information of (and references to) the hardware and software framework upon which the system was built, the internal organization of the software, the organization of the data fields, and a discussion of special techniques that were used in the implementation of LARSFRIS.

This manual, LARSFRIS User's Manual, contains a comprehensive description of the functional organization of the system, the processing functions provided, and the manner in which the

functions are invoked and controlled. While it is written primarily for the system's user, a good knowledge of its contents is essential for any individual who intends to work with the system -- be he a user, an analyst, or a programmer.

Table of Contents

SECTION 1

INTRODUCTION

SECTION 1

INTRODUCTION

This manual is directed primarily at the programmer or analyst who maintains the LARSFRIS system or writes new programs for it. It is assumed that the reader is familiar with the external characteristics and general operating concepts of the system. These topics are discussed in considerable detail in the LARSFRIS User's Manual, which is also a prerequisite to a good overall understanding of the LARSFRIS system.

The remainder of the System Manual is divided into four major sections. Section 2 describes the hardware and software environment that forms the basic framework for the design of the LARSFRIS system. Only a frief outline of these subjects is presented there, but an adequate bibliography which lists the other publications on these subjects is also included. The second section also contains a discussion of virtual machine techniques and the LARSFRIS virtual machine configuration.

Section 3 describes the physical and logical organization of the system. Flowcharts are presented which show the overall hierarchial organization of the system and the individual organization and flow of control for each major entity.

Section 4 describes a number of programming and system implementation topics that will provide a better understanding of the internal characteristics of the system as well as procedures to use in updating and revising it. It is strongly recommended that all of the topics in this section be read carefully by any programmer who is to modify or revise LARSFRIS programs.

Section 5 concludes the System Manual with a description of the data usage in LARSFRIS. Included in that section is a table of all Data Set Reference Numbers that are used (with the symbolic DSRN's and the associated FILEDEF statement), a detailed description of the contents and format of each file that is used at the processing level of the system, and equivalent descriptions of a number of special system files that support the user.

SECTION 2

THE LARSFRIS SYSTEM ENVIRONMENT

SECTION 2

THE LARSFRIS SYSTEM ENVIRONMENT

This section describes the computing environment that forms the basic framework for LARSFRIS. This environment consists of the computer equipment, the IBM-supplied system software, and the use of virtual machines. An understanding of these topics is essential to understanding the internal design and implementation of LARSFRIS. References for further information on all of these topics are included at the end of the section.

The subsections are:

2.1 LARS COMPUTER EQUIPMENT

The computer hardware currently used is centered around an IBM 3031 machine with two million bytes of main memory. Unit record equipment (card reader, card punch and line printer) are connected to the CPU via a byte multiplexor channel. Seven- and nine-track magnetic tape units are connected to the CPU via two block multiplexor channels. One block multiplexor channel is used for a special digital image display and editing unit. The operating system and user files are maintained on two spindles of IBM 3350 disk storage connected via a block multiplexor channel. Two spindles of CDC 3330-11 disk storage and two spindles of CDC 33502 disk storage used to maintain user files are connected via a block multiplexor channel. Figures 2-1 and 2-2 illustrate this configuration.

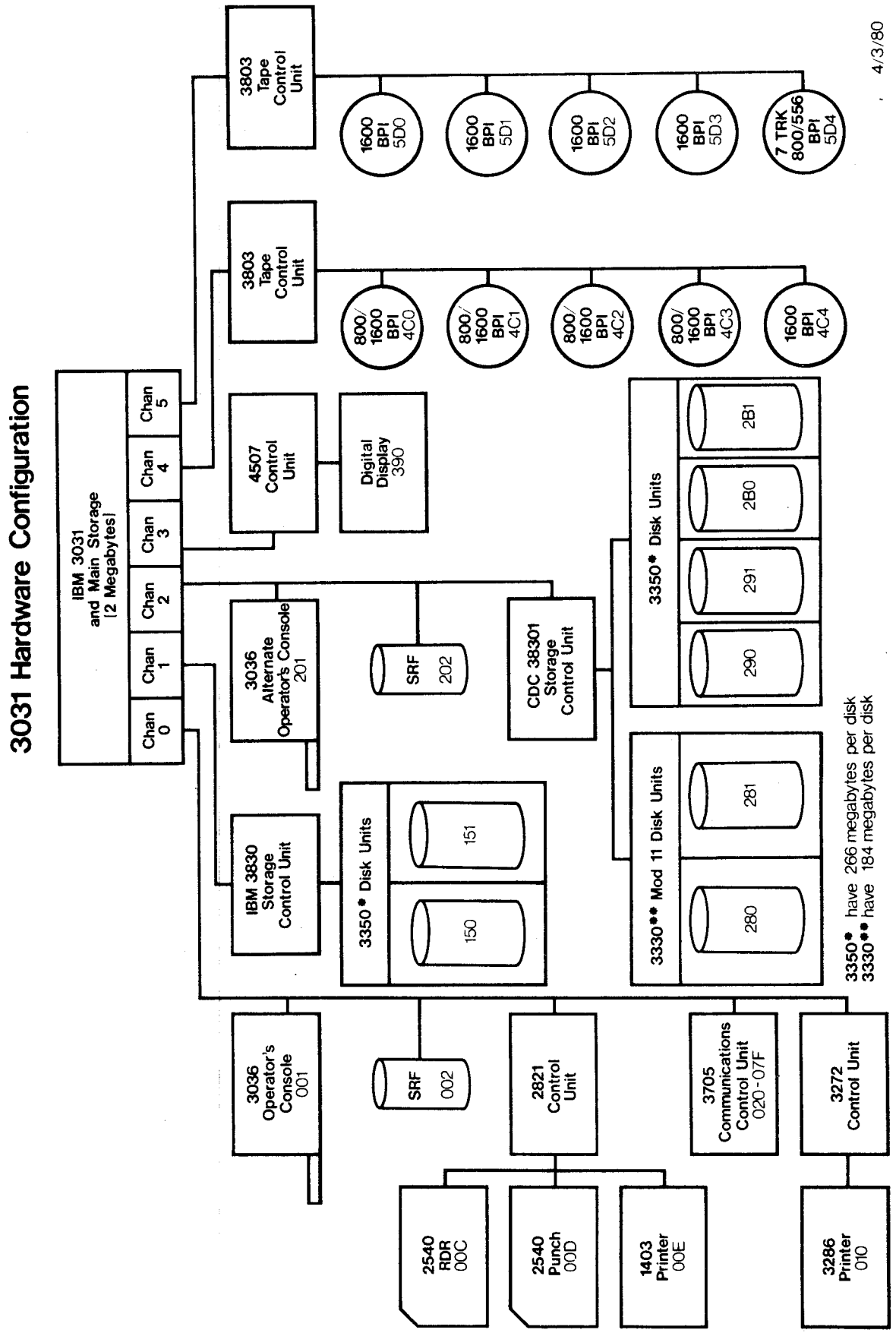


Figure 2-1. LARS Computer Configuration

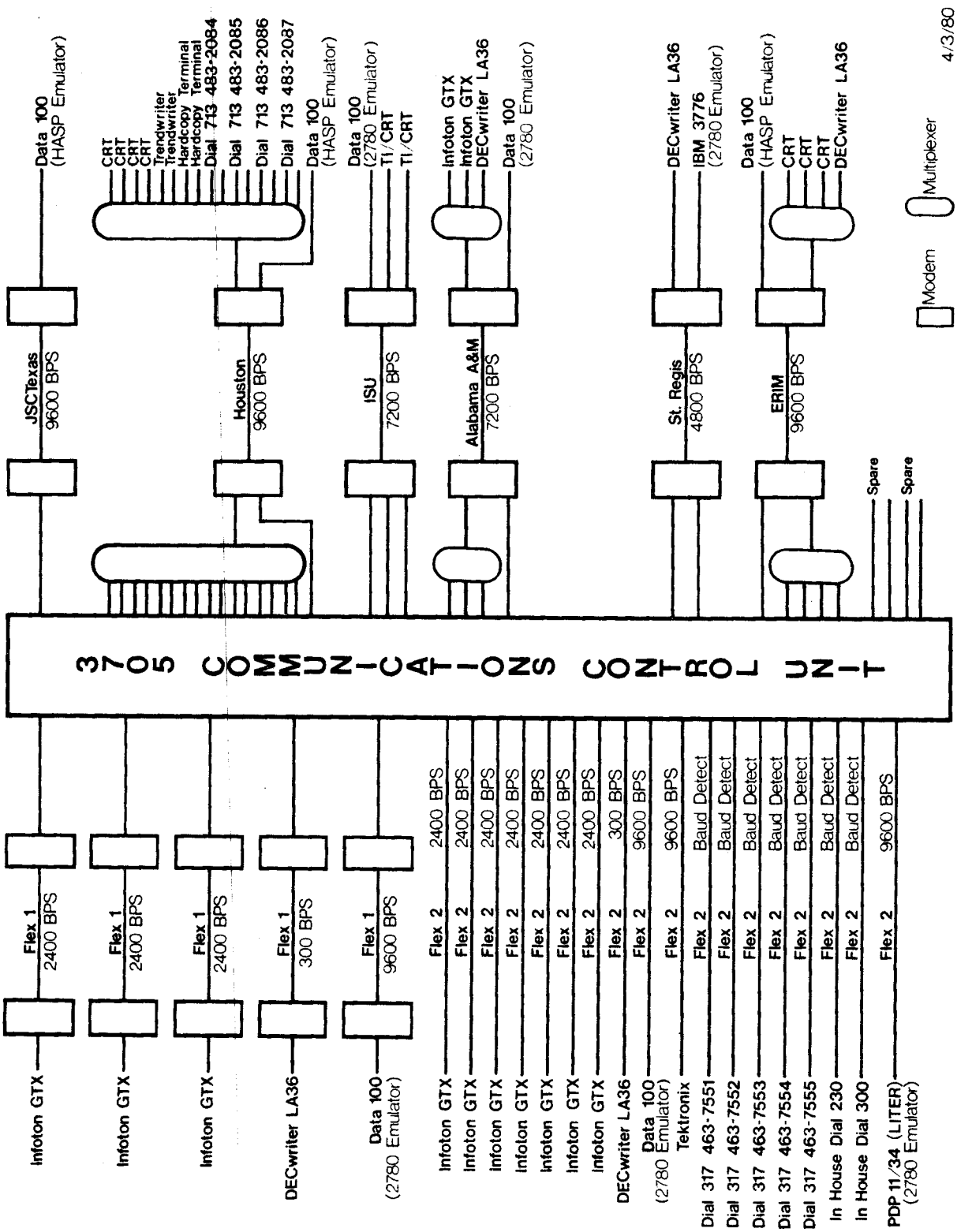


Figure 2-2. LARS Remote Terminal Configuration

2.2 IBM-SUPPLIED SOFTWARE

The LARSFRIS system uses the following IBM-supplied software components as its base: VM/370 CP and CMS, the FORTRAN IV compiler and the Assembler. These components are very briefly described below. The reader should consult the references listed at the end of this subsection for more detailed information.

VM/370 CP

The basic monitor program or control program under which the LARS computer operates is called VM/370. VM is a multi-programming package which uses special hardware features of the 3031 to create an environment in which it appears to each user that he has complete control of a dedicated machine, complete with I/O devices. These apparent machines are called virtual machines since they are created by software and do not exist in any physical sense. To the user and the program, the virtual machine is indistinguishable from a real system, but it is really one of many that VM is managing. VM allocates the resources of the real machine to each virtual machine in turn for a short slice of time, then moves on to the next user's virtual machine--thus time-sharing.

Since the real machine does not have sufficient real memory for all users' virtual memory, a technique called paging is

used by VM. Virtual memory is divided into 4,096-byte blocks called pages. All pages except those currently in use are stored by VM on secondary storage, and are called into and swapped out of real memory on a demand basis. In addition, all virtual machine input/output is handled by VM. However, all these operations are completely transparent to a user and the virtual machine.

VM also provides, as part of the virtual machine, commands that parallel the buttons and switches on the operator's console of a real machine. The user can issue these commands from the terminal, and thus the terminal becomes the pseudo-console for his virtual machine.

VM simulates card reader, punch and printer operations for a virtual machine. If a program running in a virtual machine is to process a card file, that card file must first be read into VM, headed by an ID card to identify the intended virtual machine. It is then stored as a disk file in what is called the VM spooling area. When the virtual machine requests card-reader input, VM supplies it with card images from the spooled input file. The same process works in reverse for printer and punch output; a disk spooling file is created, which is later transferred by VM from disk to a real printer or punch.

CMS

After the control program (CP) creates the virtual machine, that virtual machine must be equipped with its own operating system to provide support for the programs to be run. The programming system most commonly used at LARS is called the Conversational Monitor System (CMS). LARSFRIS uses CMS as its virtual machine operating system.

CMS is a single-user, conversational operating system designed to provide full use of a System 370 machine using a simple command language that can be entered at the terminal. CMS provides a full range of capabilities--creating and managing files, compiling, debugging and executing programs, utilities, control commands and library facilities.

The LARSFRIS programmer will normally use all the facilities of CMS, whereas the typical LARSFRIS user is insulated from CMS and is only aware of LARSFRIS. More advanced LARSFRIS users may effectively use some CMS facilities in conjunction with LARSFRIS.

FORTRAN

CMS provides a FORTRAN IV compiler which is identical to the OS G-level compiler. FORTRAN IV is a mathematically-oriented language useful in writing programs for applications that involve manipulation of numerical data. The majority of the LARSFRIS program modules are written in the FORTRAN

IV language.

Assembler

CMS supports the OS/370 Assembler language. The Assembler provides access to the full facilities of the hardware and operating system to the programmer, which FORTRAN does not. A small number of LARSFRIS program modules are written in Assembler language.

LARS Modifications to CP and CMS for LARSFRIS

CP and CMS both required several additional functions and minor modifications to permit LARSFRIS to operate as designed. Contact the LARS programming staff for detailed information about these changes.

References

The following collection of documents describes the programming supplied by IBM.

VM/370

GC20-1818 VM/370 CMS Command and Macro Reference

SD23-9023 BSEPP Supplement for GC20-1818

GC20-1819 VM/370 CMS User's Guide

SD23-9024 BSEPP Supplement for GC20-1819

GC20-1820 VM/370 CP Command Reference for General Users

FORTRAN

GC28-6515 FORTRAN IV Language

GC28-6817 FORTRAN IV Programmer's Guide

Assembler

GA22-7000 System/370 Principles of Operation

GC33-4010 Assembler Language (OS/VS-DOS/VS-VM/370)

GC33-4021 Assembler Programmer's Guide (OS/VS and VM/370)

2.3 THE LARSFRIS VIRTUAL MACHINE

CP implements the virtual machines. Since the virtual machines (VM) are simulated, their configurations may differ from each other and from the real machine. Most virtual machines, however, have the same configuration. This configuration is based on the requirements of LARSFRIS and CMS, which is the VM operating system required by LARSFRIS. Refer to Figure 2-3 for the configuration diagram. Note that the dashed lines represent optional devices and that the virtual device addresses are in parentheses. When the user completes the LOGON sequence, his virtual machine is automatically established as that represented by the solid lines. Certain optional devices may be attached to his virtual machine automatically by LARSFRIS depending on which LARSFRIS functions are selected.

Configuration

The VM/370 components are:

VM/370

CP provides virtual System/370 processors with an installation-specified amount of main storage. Because of the single level of overlay load modules used by LARSFRIS, 960K bytes is large enough to handle the largest functional load module.

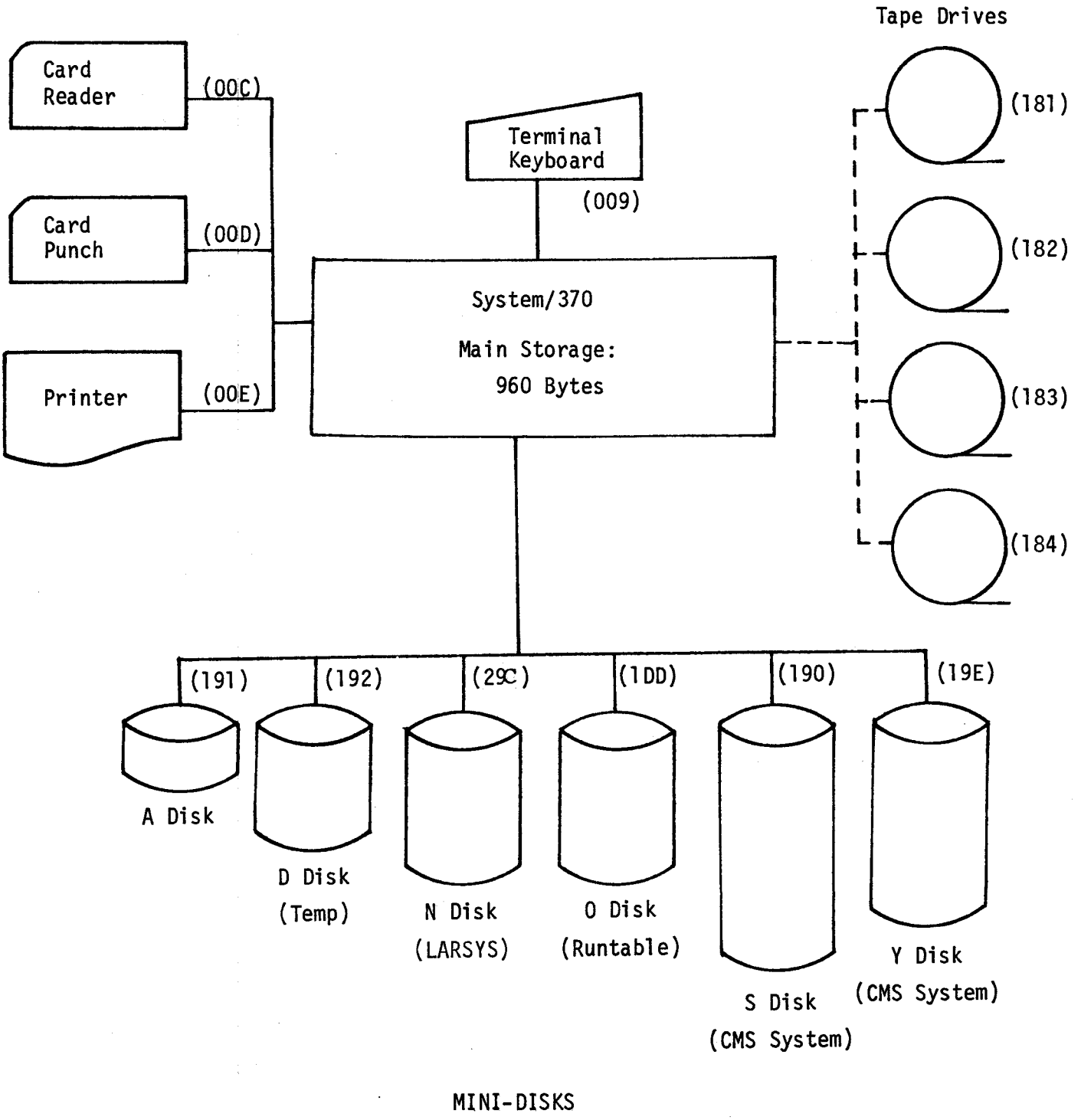


Figure 2-3. LARSFRIS Virtual Machine

Card Reader

This device is automatically specified as a spooled card reader. Therefore, the card reader in the computer room may be used or, for remote LARSFRIS users, any of the remote card readers may be used. The ID card specifies to which userid CP should spool the input deck.

Card Punch

This device is automatically specified as a spooled punch. Therefore, the punch in the computer room will be used unless a REMOTE command has been issued to specify another unit for punch output. The LARSFRIS module EXCOMD EXEC will automatically issue the proper REMOTE command for a remote user.

Console Keyboard

The terminal where the user LOGON occurred will be assigned as the console screen for the virtual machine (simulating the console of the real machine).

Disk Storage

CP and CMS implement the mini-disk concept. This permits many users to have distinct portions of a real disk assigned to their virtual machines. These mini-disks may be private or shared with other users. The LARSFRIS user automatically has access to five mini-disks. The five disks are:

A DISK - This is the user's private or permanent disk. The size of the A disk is defined in the CP user directory for each userid.

D DISK - This disk provides space to store intermediate results during a terminal session; thus, it is a temporary disk. The D disk is automatically assigned from a pool of termporary disk space set aside for that purpose. LARSFRIS currently acquires 3 million bytes of temp disk space when the user enters LARSFRIS. The temp disk is always cleared when I LARSYS is issued by the user.

S DISK - These are the standard system disks shared by all
Y DISK CMS users. The S and Y disks are a pair of shared read-only disks containing all of CMS and other system software. This includes the CMS nucleus, all transient routines, compilers, FORTRAN library, etc.

N DISK - All LARSFRIS users have access to a single shared, read-only N disk that contains the LARSFRIS programs and necessary system support EXEC routines. Access to the disk is provided automatically when the user issues the I LARSYS command.

O DISK - This disk contains the LARSFRIS system runtable.

Tape Drives

Virtual tape drives must correspond one-for-one with actual tape drives in the computer room. Since there are a limited number of actual tape drives available, their usage must be carefully managed. LARSFRIS attempts to improve tape drive utilization by automatically and dynamically attaching and detaching drives based on the requirements of the processing function the user is currently running. For more information on how tapes are attached and detached, see Section 4 (Subsection 4.7). The VM may use up to four tapes concurrently; however, none of the present functions require more than two drives, and most of them require only one.

References

The following documents describe the LARSFRIS virtual machine:

GC20-1818	VM/370 CMS Command and Macro Reference
SD23-9023	BSEPP Supplement for GC20-1818
GC20-1819	VM/370 CMS User's Guide
SD23-9024	BSEPP Supplement for GC20-1819
---------	LARSFRIS User's Manual

SECTION 3

THE LARSFRIS SYSTEM ORGANIZATION

SECTION 3

THE LARSFRIS SYSTEM ORGANIZATION

This section describes the system organization of LARSFRIS. The organization is described mainly through a series of flowcharts, which document the physical and logical hierarchy of the system, and the flow of system control down to the individual program module level.

The subsections are:

- 3.1 The Overall LARSFRIS Hierarchy
- 3.2 Executive and Monitor Level Organization
- 3.3 Organization of the Functional Load Modules

Subsection 3.1 describes the LARSFRIS system control hierarchy. This includes the two levels of IBM-supplied software VM/370 and CMS) and the three levels within the LARSFRIS system itself. The accompanying figure shows these relationships graphically, with the flow shown from the highest level VM/370 at the top of the page, down to the lowest level (the individual supervisor) at the bottom of the page. Within LARSFRIS the individual CMS executive routines or processing function supervisors are identified by name.

Subsections 3.2 and 3.3 provide the detailed flowcharts for each of the three LARSFRIS levels that were identified in the first subsection. These flowcharts show the overall system logic and the major actions, the routines called, and data input and output; down to the level of the individual executive routine or program module. For more detailed descriptions of the logic of the individual routines and programs, see the LARSFRIS Program Documentation Manual.

The flowcharts in these two subsections all follow the same charting conventions. The actions that are performed by the executive routine or supervisor (or by a called routine) are briefly stated in the box on the left hand side of the page. Opposite this brief statement is a graphical representation of the action. Functional routines that are called are represented by boxes with the name of the routine printed in the top of the box. Support routines that are called are not represented by separate boxes; instead their names are simply listed inside the box for the functional routine that calls them. I/O actions are represented by the standard flowcharting symbols for reading and writing data. Appendix I contains a complete list of all modules used in the LARSFRIS System.

3.1 THE OVERALL LARSFRIS HIERARCHY

The organization chart in Figure 3-1 shows the hierarchy of control and, where applicable, the individual module names. The system runs under Control Program/370 and the Conversational Monitor System operating system (CP/370 and CMS). CP/370 is the highest level of software, with CMS running in a virtual machine under CP. The LARSFRIS modification of CMS causes the EXCOMD routine to be executed when the 'i larsys' command is issued. When RUN LARSYS is issued by the user, EXCOMD calls RUNLS EXEC which loads the root module and passes control to subroutine LARSMN (the FORTRAN main program). LARSMN reads Function Selector cards and passes control to the appropriate supervisor after loading the load module. All of the supervisors names shown in the process level are also the names of the load modules. The functional modules all load at the end of the root (LARSMN) and thus overlay each other.

Most load modules (and supervisors) execute only one function. The exceptions are RUNSUP (TRANSFERDATA, CHANNELTRANSFORMATION and IDLIST), GRHSUP (GRAPHHISTOGRAM, LINEGRAPH and COLUMNGRAPH) and RESSUP (COPYRESULTS, LISTRESULTS, and PUNCHSTATISTICS). The module CHASUP is called by RUNSUP to supervise the Channeltransformation function. This hierarchy is the same as that used on the batch machine except that the module PROFILE would replace EXCOMD.

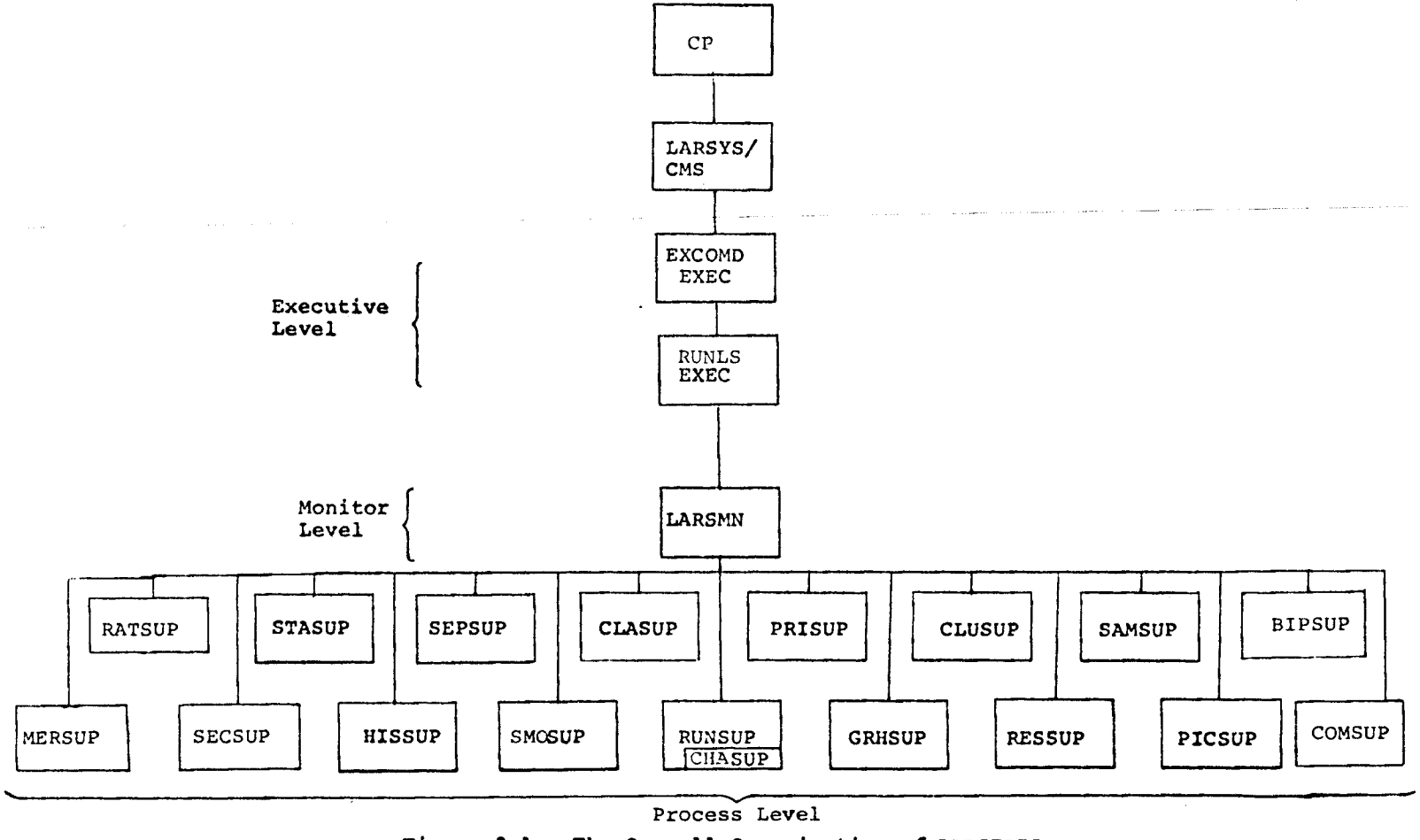


Figure 3-1. The Overall Organization of LARSFRIS.

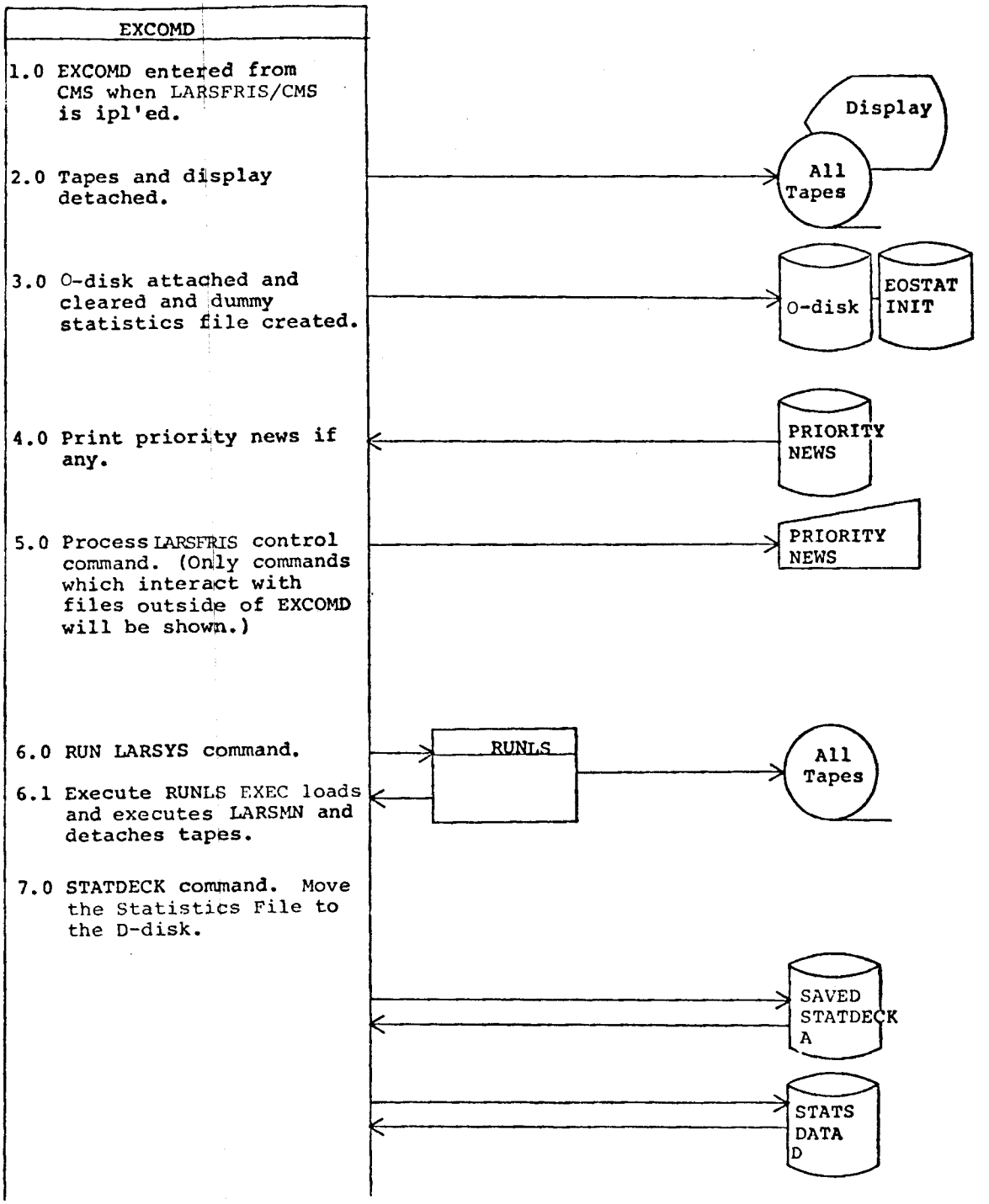
3.2 ORGANIZATION EXECUTIVE AND MONITOR LEVELS

Executive Level Flowchart

The executive level is entered when LARSFRIS/CMS is ipl'ed. A LARSFRIS modification to CMS causes EXCOMD EXEC to be executed automatically after IPL. EXCOMD detaches tapes and the display unit, acquires a T-disk, initializes the Statistics File and then goes into a loop which reads and processes LARSFRIS control commands. For a definition of the function of these commands, see the LARSFRIS User's Manual. The flowchart that follows indicates the processing of only those commands which involve files outside of EXCOMD. Steps 1.0-4.0 on the flowchart are executed only once before the command processing loop is entered.

Load Module Name: Executive

EXECUTIVE

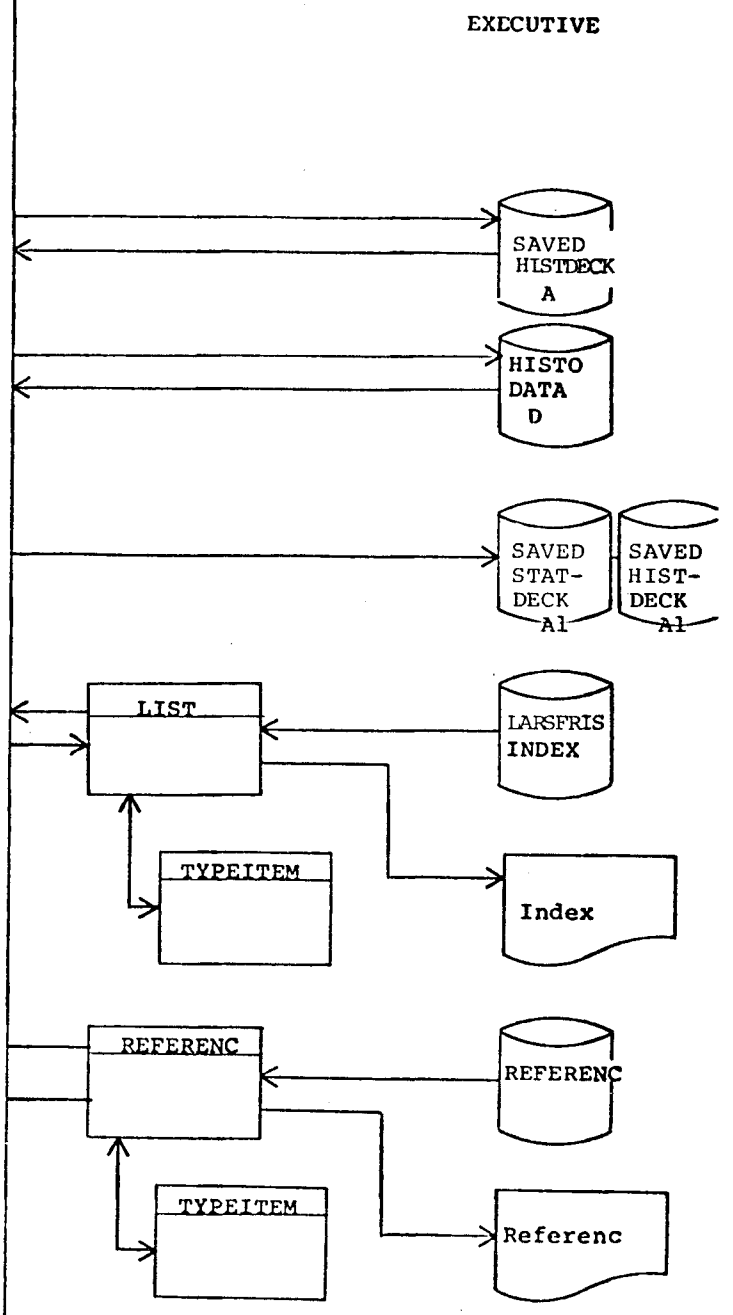


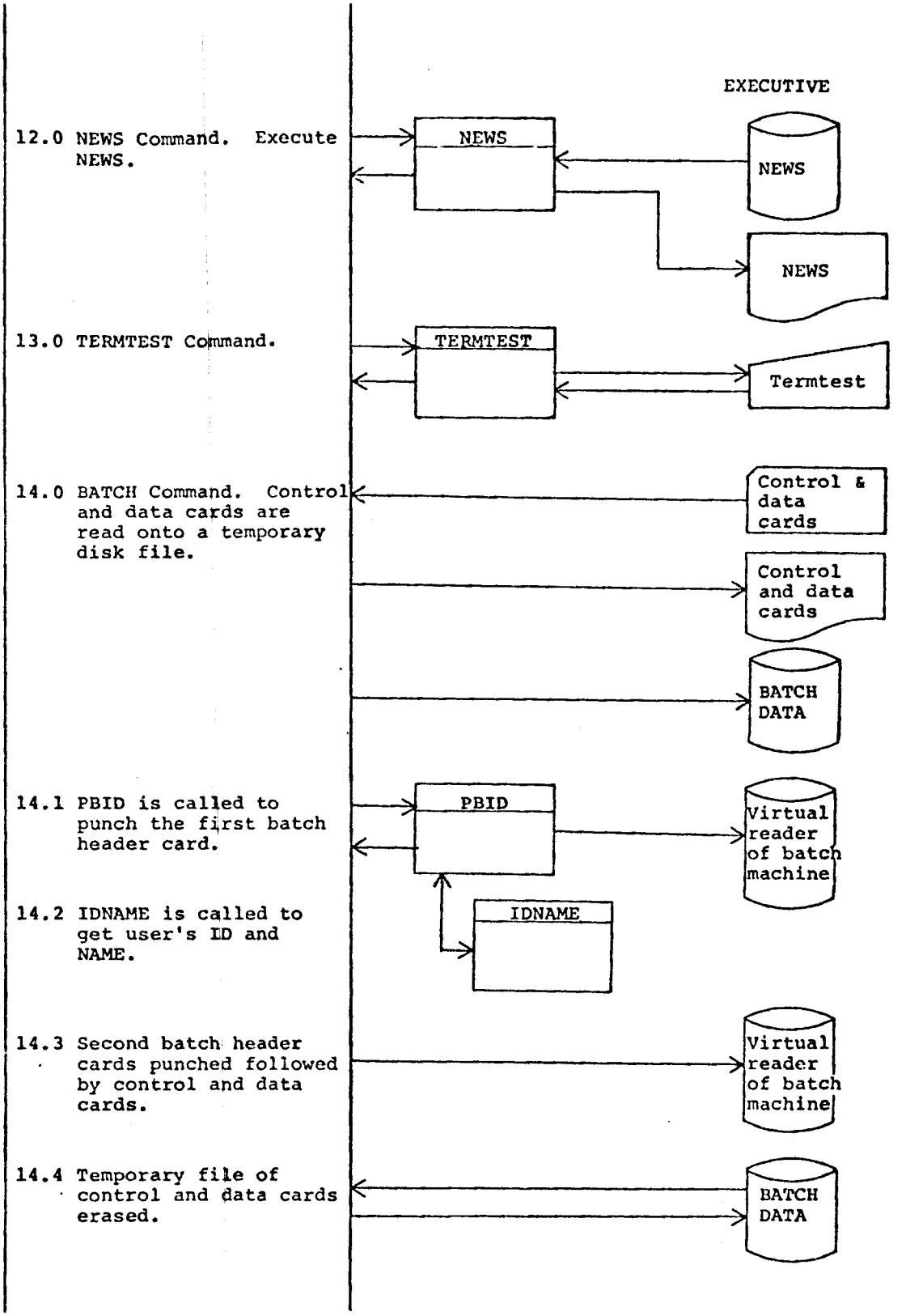
8.0 HISTDECK Command. Move the Histogram File to the D-disk.

9.0 CLEAR Command.

10.0 LIST Command. Execute LIST.

11.0 REFERENCE Command. Execute REFERENC.




EXECUTIVE
12.0 NEWS Command. Execute NEWS.
NEWS
NEWS
NEWS
13.0 TERMTEST Command.
TERMTEST
Termtest
14.0 BATCH Command. Control and data cards are read onto a temporary disk file.
Control & data cards
Control and data cards
BATCH DATA
14.1 PBID is called to punch the first batch header card.
PBID
Virtual reader of batch machine
14.2 IDNAME is called to get user's ID and NAME.
IDNAME
14.3 Second batch header cards punched followed by control and data cards.
Virtual reader of batch machine
14.4 Temporary file of control and data cards erased.
BATCH DATA

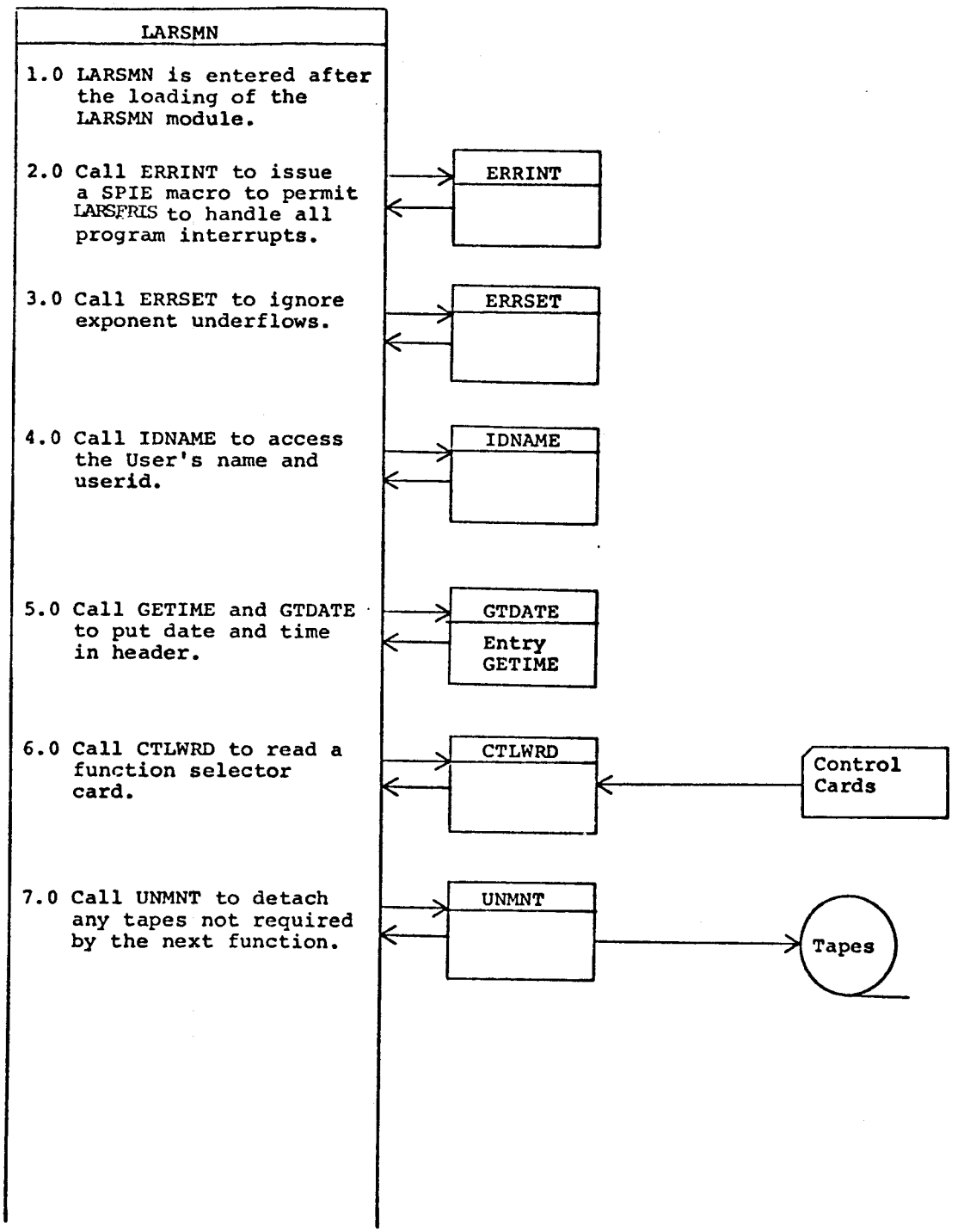
Monitor Level Flowchart

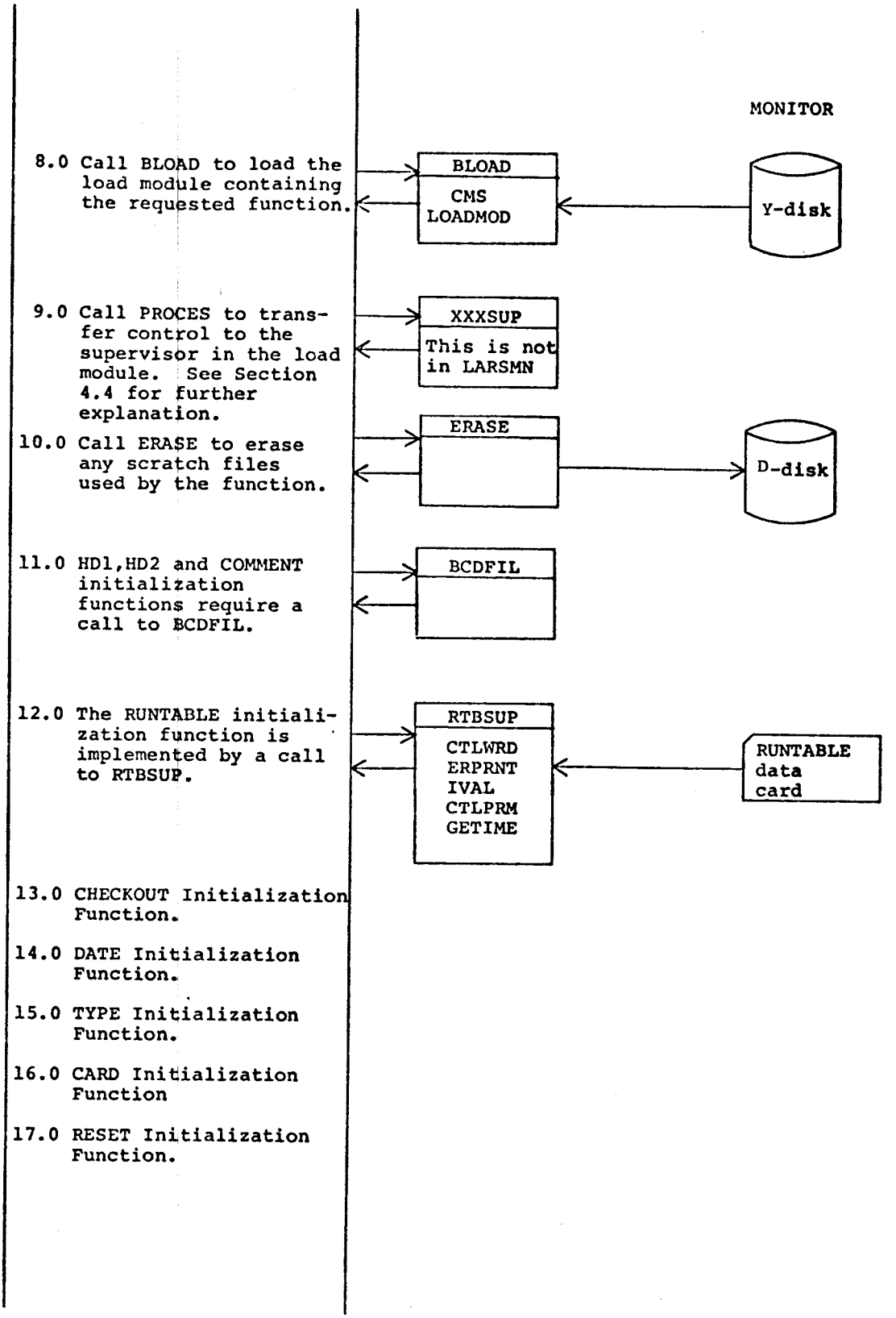
The monitor level is entered from RUNLS EXEC after loading the LARSMN load module (CMS file LARSMN MODULE). Control enters at LARSMN, the FORTRAN main program of LARSFRIS. The monitor level includes LARSMN, and the subroutines it calls as described on the load module flowchart. The Root Load Module which contains the monitor routines also contains a number of system support subroutines not actually used by LARSMN but used by many of the functions in other load modules, as well as the GLOCOM common block.

LARSMN functions in a loop reading Function Selector Cards requesting major processing functions and initialization functions and executing these functions. After all cards have been read by LARSMN and the functional load modules, control passes back to RUNLS EXEC in the executive level.

MONITOR

Load Module Name: LARSMN




MONITOR
8.0 Call BLOAD to load the load module containing the requested function.
BLOAD
CMS LOADMOD
Y-disk
9.0 Call PROCES to transfer control to the supervisor in the load module. See Section 4.4 for further explanation.
XXXSUP
This is not in LARSMN
10.0 Call ERASE to erase any scratch files used by the function.
ERASE
D-disk
11.0 HD1,HD2 and COMMENT initialization functions require a call to BCDFIL.
BCDFIL
12.0 The RUNTABLE initialization function is implemented by a call to RTBSUP.
RTBSUP
CTLWRD
ERPRNT
IVAL
CTLPRM
GETIME
RUNTABLE data card
13.0 CHECKOUT Initialization Function.
14.0 DATE Initialization Function.
15.0 TYPE Initialization Function.
16.0 CARD Initialization Function
17.0 RESET Initialization Function.

18.0 LARSMN module includes support routines.

TAPOP
ERPRNT
CHANEL
LARS12
TSTREQ
ERMNAM
CPFUNC
MOUNT
URADST
GADRUN
RUNERR
GADLIN

MONITOR

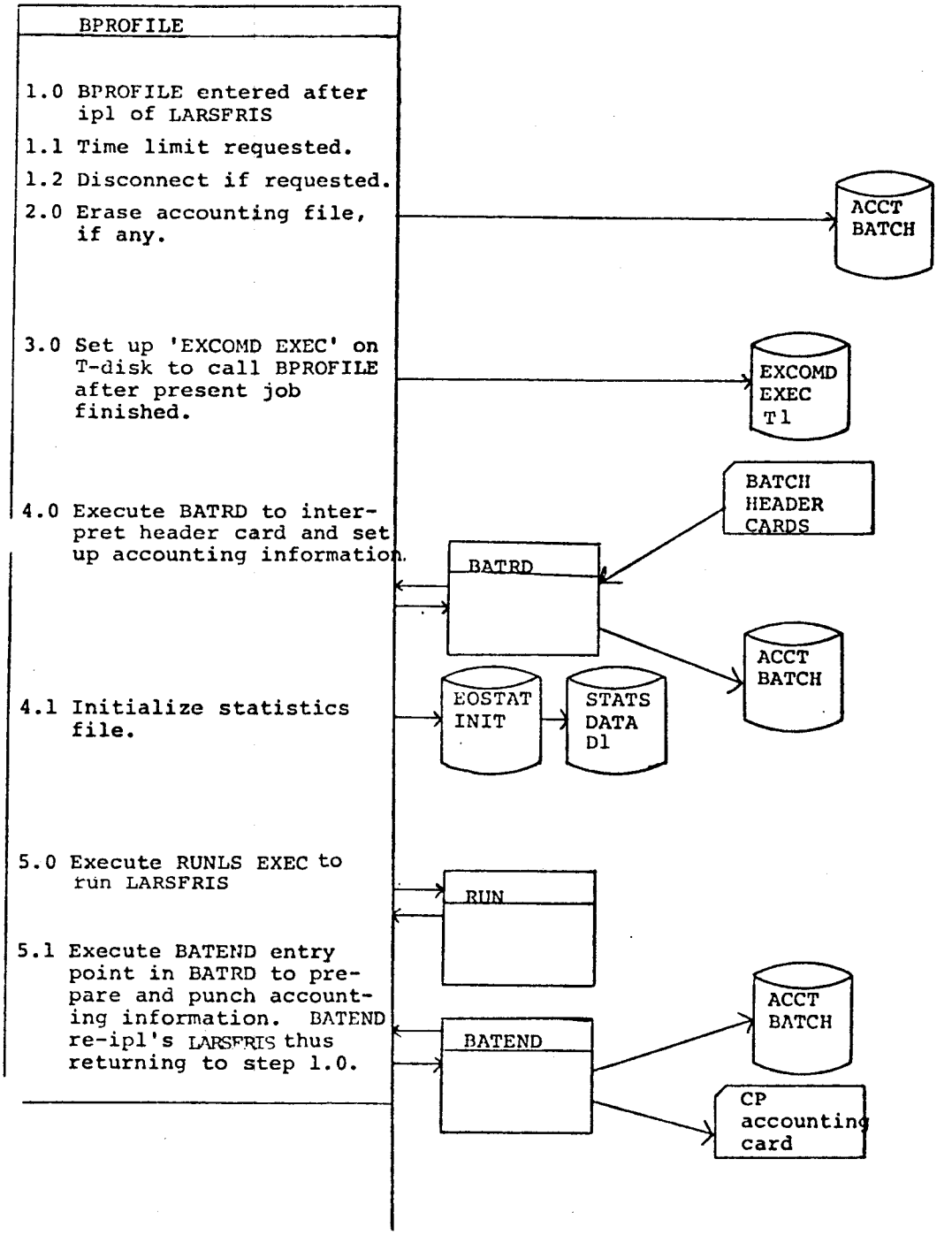
Batch Monitor Flowchart

The LARSFRIS facility that supports the batch mode of operation is implemented by two routines, a CMS executive routine called BPROFILE and an assembler language program called BATRD. BPROFILE is entered whenever LARSFRIS is ipl'ed, and again whenever a batch job is finished. (This is done with an EXEC file 'EXCOMD' on the N-disk.) The first time the batch machine is ipl'ed, BPROFILE will request a time limit to be applied to all jobs run by that machine in the future. It is exited only by the DRYUP command which will logout the batch machine.

The BATRD routine is called by BPROFILE to read and interpret the batch header cards, communicate with the operator on the status of the batch virtual machine, and generate accounting information. For detailed information on BATRD refer to the LARSFRIS Program Documentation Manual.

Load Module Name: Batch Monitor

BATCH MONITOR



3.3 ORGANIZATION OF THE FUNCTIONAL LOAD MODULES

The individual flowcharts for each of the seventeen LARSFRIS load modules are presented on the following pages. The flowcharts describe the flow of control within each load module, identify all program routines used, all use of input/output files, and all CALL's external to the load module.

The pages are organized and numbered by supervisor in the following order:

BIPSUP

CLASUP

CLUSUP

COMSUP

GRHSUP

HISSUP

MERSUP

PICSUP

PRISUP

RATSUP

RESSUP

RUNSUP

SAMSUP

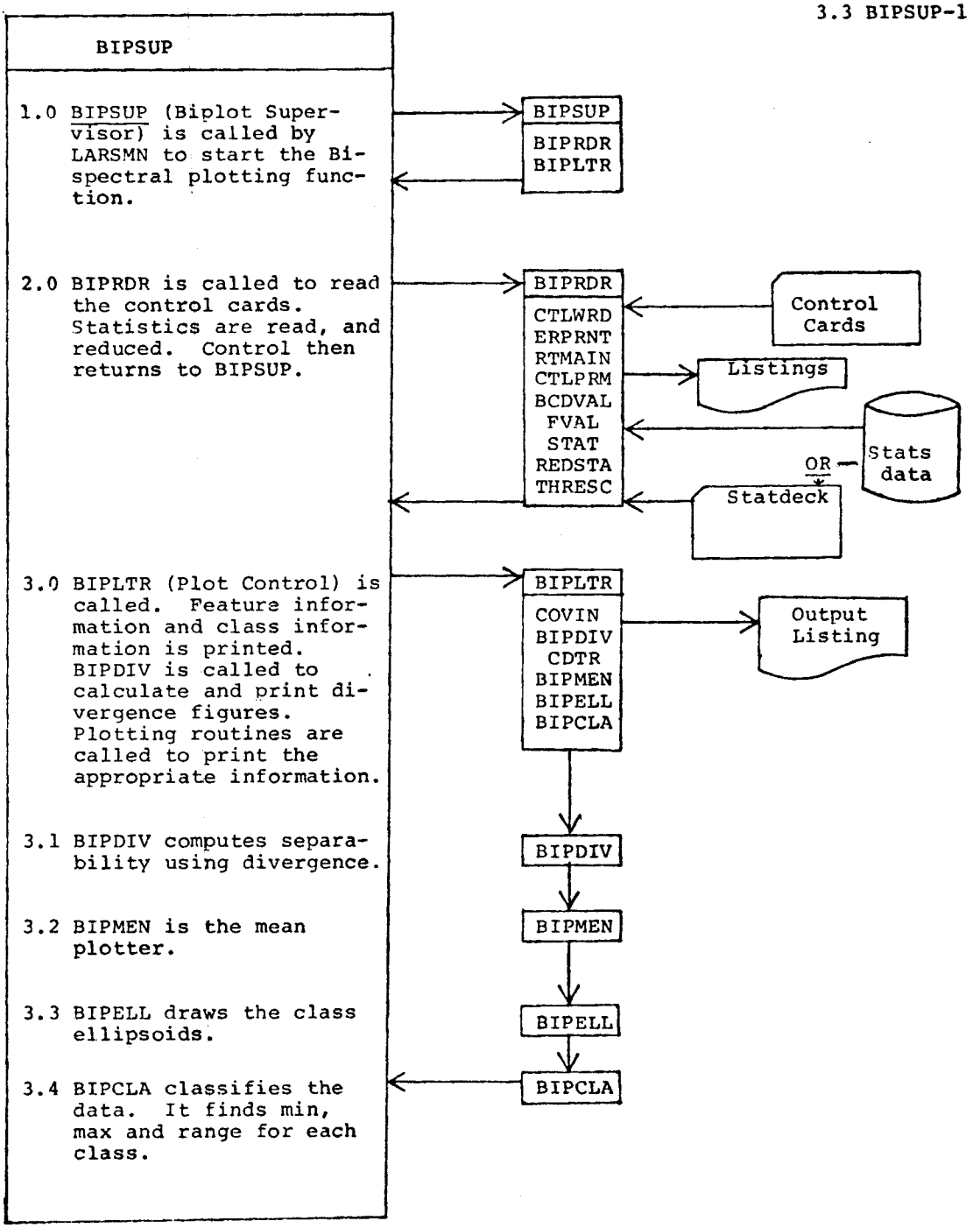
SECSUP

SEPSUP

SMOSUP

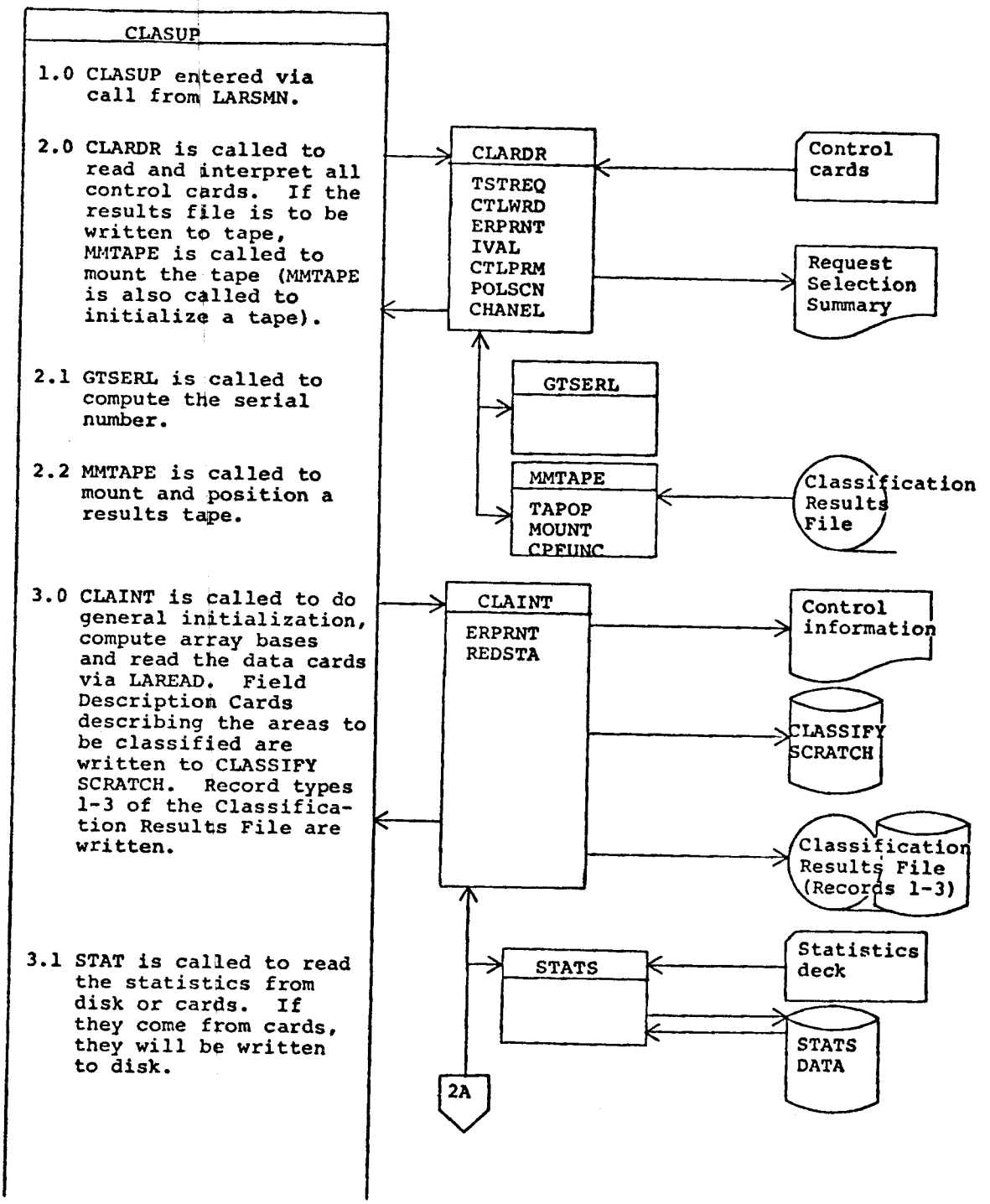
STASUP

3.3 BIPSUP-1


BIPSUP
1.0 BIPSUP (Biplot Supervisor) is called by LARSMN to start the Bispectral plotting function.
2.0 BIPRDR is called to read the control cards. Statistics are read, and reduced. Control then returns to BIPSUP.
3.0 BIPLTR (Plot Control) is called. Feature information and class information is printed. BIPDIV is called to calculate and print divergence figures. Plotting routines are called to print the appropriate information.
3.1 BIPDIV computes separability using divergence.
3.2 BIPMEN is the mean plotter.
3.3 BIPELL draws the class ellipsoids.
3.4 BIPCLA classifies the data. It finds min, max and range for each class.
BIPSUP
BIPRDR
BIPLTR
BIPRDR
CTLWRD
ERPRNT
RTMAIN
CTLPRM
BCDVAL
FVAL
STAT
REDSTA
THRESC
Control Cards
Listings
Stats data
OR
Statdeck
BIPLTR
COVIN
BIPDIV
CDTR
BIPMEN
BIPELL
BIPCLA
Output Listing
BIPDIV
BIPMEN
BIPELL
BIPCLA

Load Module Name: CLASUP

3.3 CLASUP-1



3.3 CLASUP-2

3.2 CLSCHK is called to check class (and pool) validity.

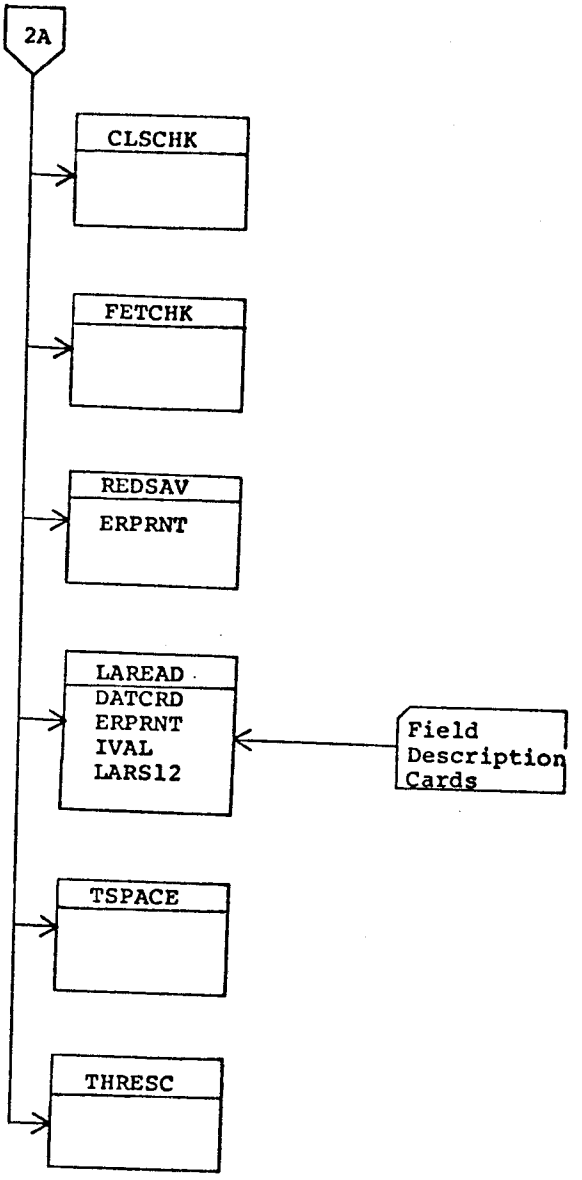
3.3 FETCHK is called to check channel validity.

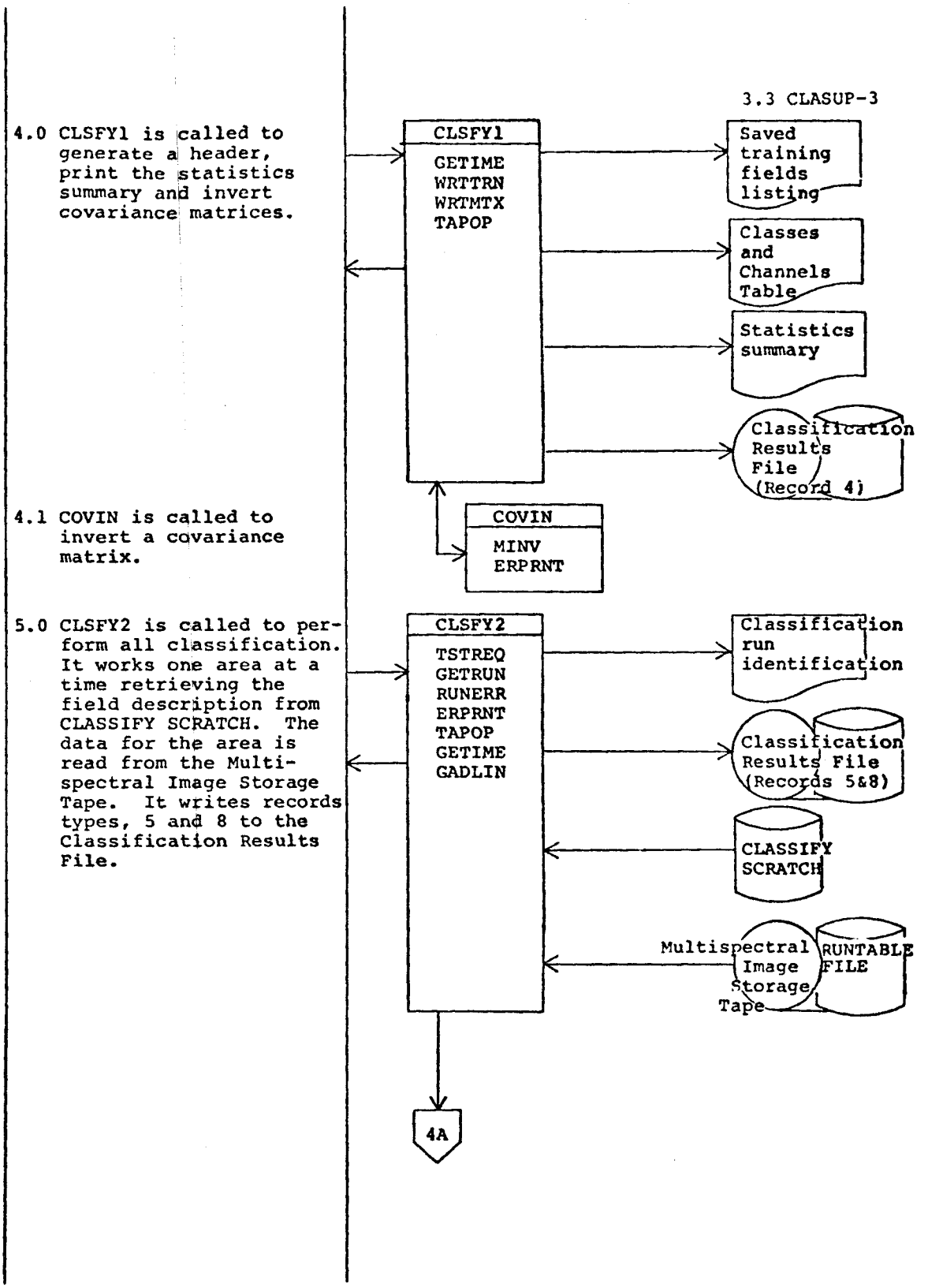
3.4 REDSAV is called to reduce statistics into classification pools.

3.5 LAREAD is called to read in Field Description Cards describing areas to be classified.

3.6 TSPACE is called to determine if the T-disk has space available for the results file (if RESULTS DISK was requested).

3.7 THRESC is called to generate the THRTAB table of Chi-square values.




3.3 CLASUP-3
4.0 CLSFY1 is called to generate a header, print the statistics summary and invert covariance matrices.
CLSFY1
GETIME
WRTTRN
WRTMTX
TAPOP
Saved training fields listing
Classes and Channels Table
Statistics summary
Classification Results File (Record 4)
4.1 COVIN is called to invert a covariance matrix.
COVIN
MINV
ERPRNT
5.0 CLSFY2 is called to perform all classification. It works one area at a time retrieving the field description from CLASSIFY SCRATCH. The data for the area is read from the Multispectral Image Storage Tape. It writes records types, 5 and 8 to the Classification Results File.
CLSFY2
TSTREQ
GETRUN
RUNERR
ERPRNT
TAPOP
GETIME
GADLIN
Classification run identification
Classification Results File (Records 5&8)
CLASSIFY SCRATCH
Multispectral Image Storage Tape
RUNTABLE FILE
4A

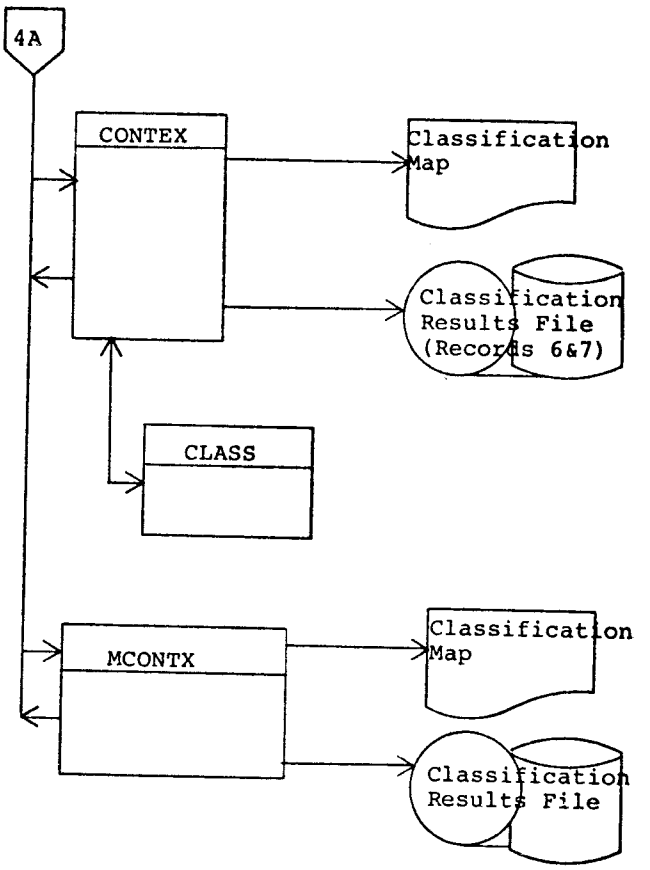
5.1 CONTEX is called to apply the maximum likelihood classification rule; and write one line of the Clasification Map. It writes Record types 6 and 7 to the Clasification Results File.

5.1.1 CLASS is an assembler routine which does the actual classification and computes the likelihood code.

5.2 MCONTX is called to compute the L1 or L2 (Euclidean) distance of each point to all classes; and to write one line of the Classification Map. It writes Record types 6 and 7 to the Results File.

6.0 Control is returned to LARSMN.

3.3 CLASUP-4



Load Module Name: CLUSUP

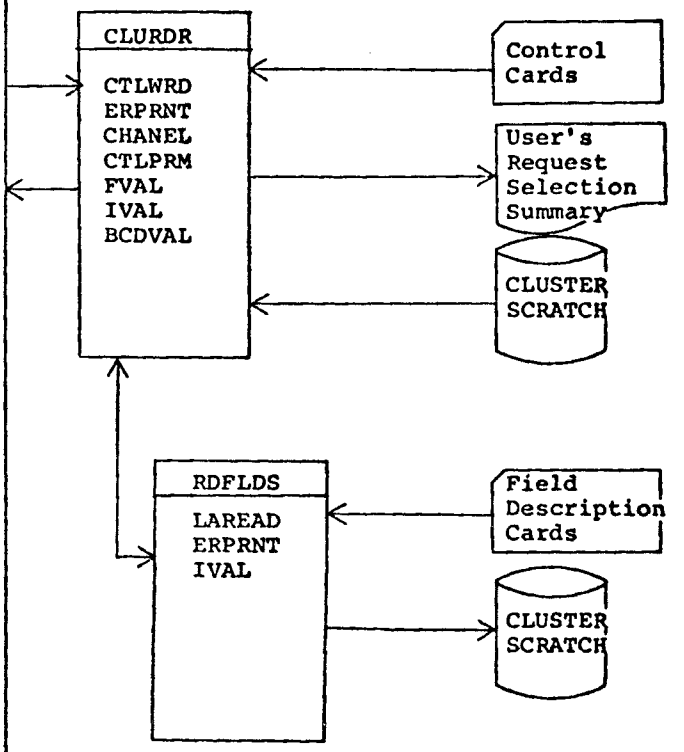
3.3 CLUSUP-1

CLUSUP

1.0 CLUSUP is entered from LARSMN to process the Cluster function request.

2.0 CLURDR is called to read control cards and perform initialization operations. The scratch data set is rewound. A summary of the user's requests are printed. RDFLDS is called to process the field description cards. Control is returned to CLUSUP.

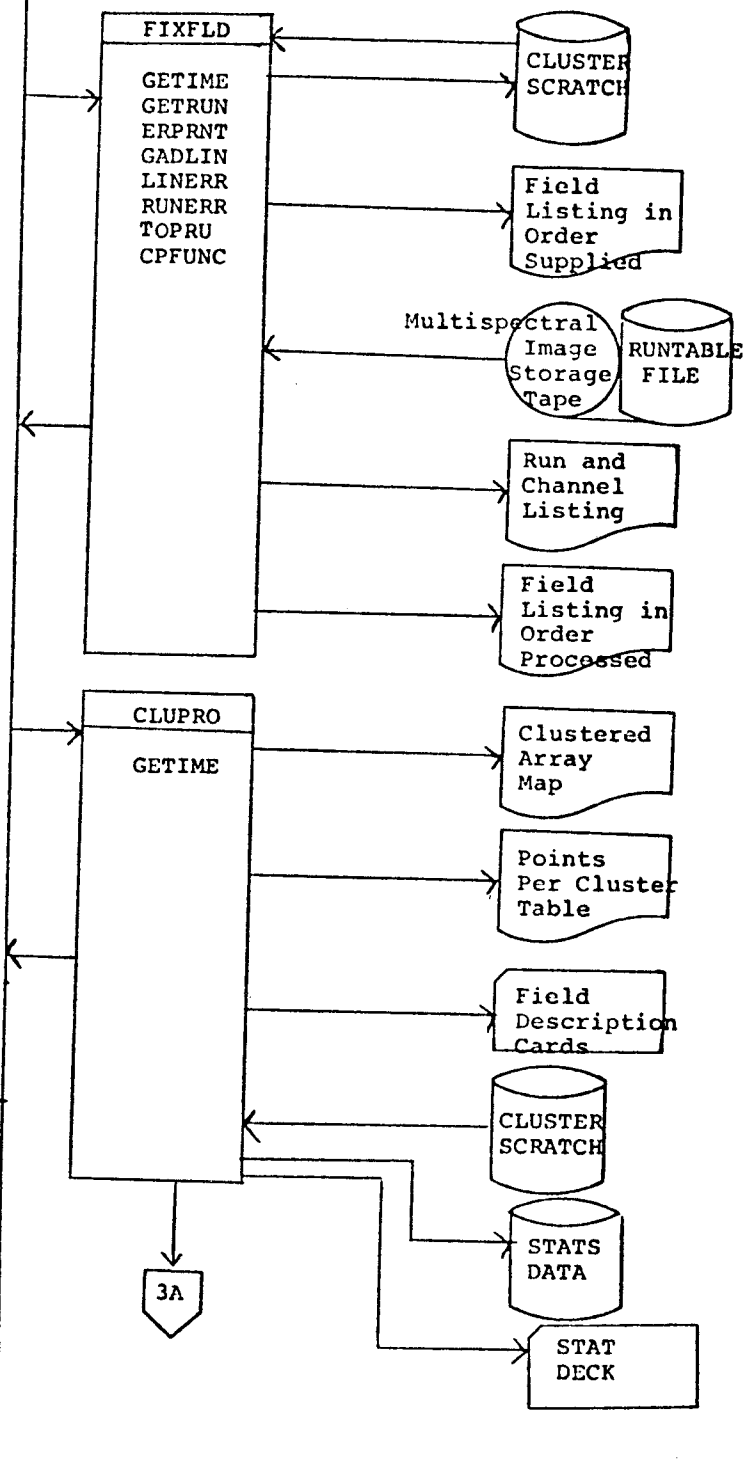
2.1 RDFLDS reads field description cards from the card reader, orders them by run number and line number and then writes the fields onto the cluster scratch disk data set in the order supplied. Return is made to CLURDR.



3.3 CLUSUP-2

3.0 FIXFLD is called to print the fields to be clustered and retrieve data from the Multispectral Image Storage Tape. Fields are listed on the printer in the order supplied. The fields are then written on the scratch disk area in the order to be processed. The Multispectral Image Storage Tape is mounted for each run requested and data is read as requested. Run and channel information is printed. After all data is read the fields are listed in the order processed. Return is made to CLUSUP.

4.0 CLUPRO establishes initial cluster centers and calls CLUMP to cluster the data. The array map of clustered points is printed. Then the table of number of points per cluster is printed, one for each field read from the cluster scratch data set. If punching of the fields was requested, then first card is punched and PCHFLD is called. Statistics are calculated and written on STATS DATA. If a punched statistics deck is requested it is punched out. A check is then made for further clustering. Return is made to CLUSUP.



3.3 CLUSUP-3

4.1 CLUMP is called to cluster the data and output on the printer means and variances. CLUMP1 is called to clump requested data. Then new means and variances are calculated and output on the printer. Return is made to CLUPRO.

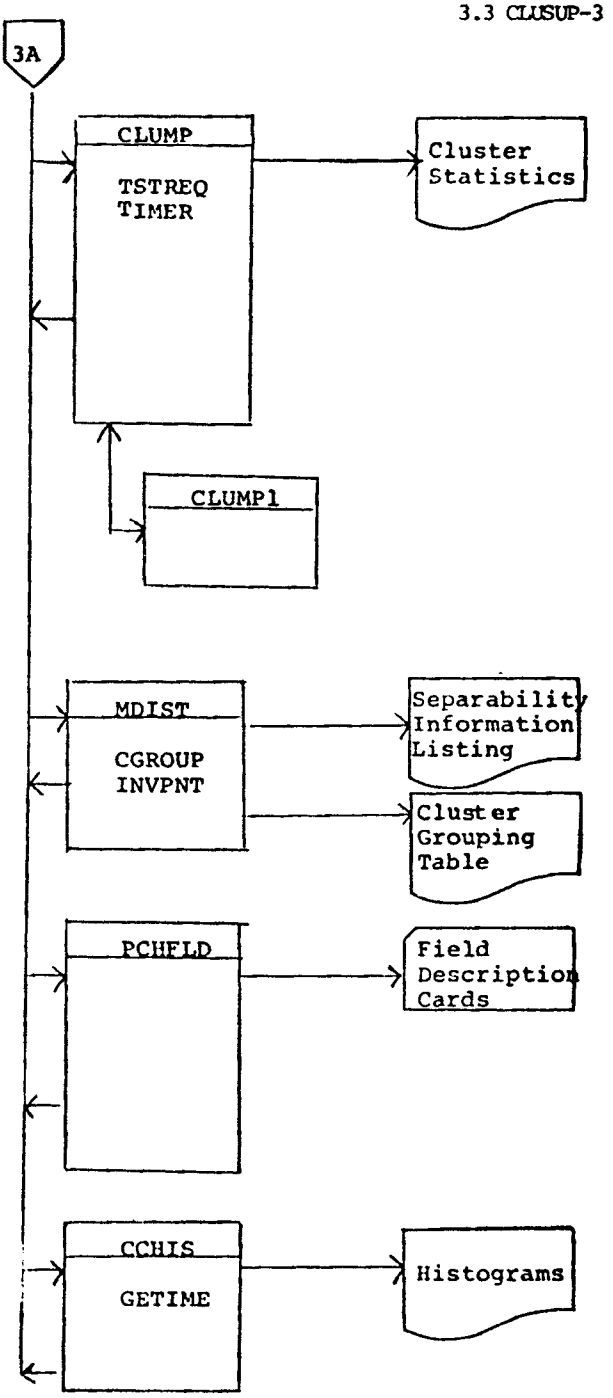
4.1.1 CLUMP1 is called to clump the data into clusters. Return is then made to CLUMP.

4.2 MDIST is called to compute separability information and print results. Return is made to CLUPRO.

4.3 PCHFLD is called to punch the field description. If FIELD specified on the PUNCH card then field description cards are punched. Control is then returned to CLUPRO.

4.4 CCHIS is called to print out histograms if requested.

5.0 CLUSUP returns control to LARSMN.



Load Module Name: COMSUP

COMSUP

1.0 COMSUP is entered via a call from LARSMN.

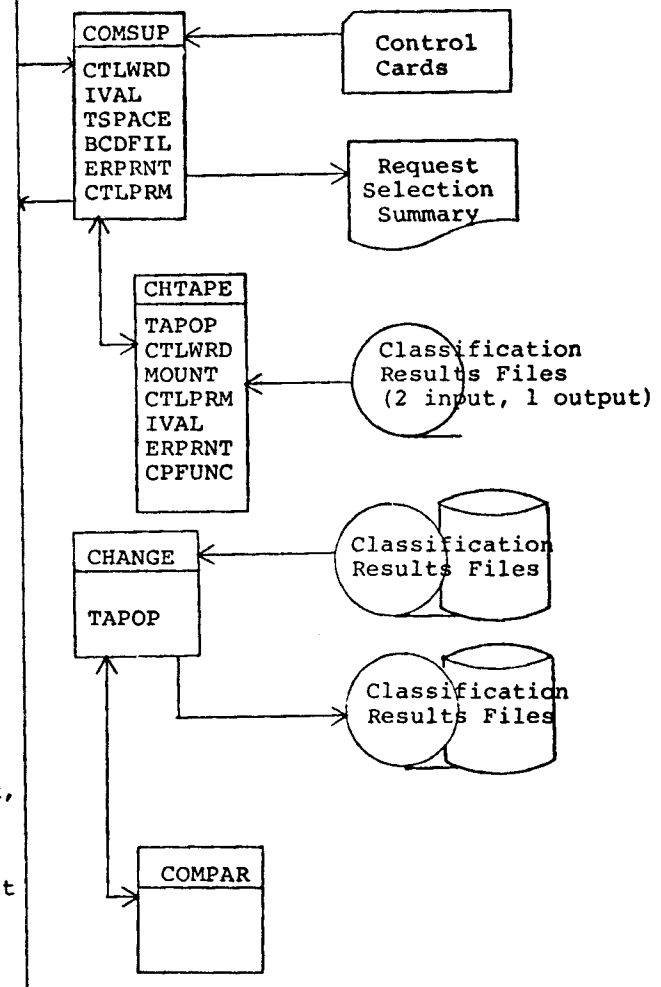
2.0 COMSUP is called to read and interpret all control cards.

2.1 CHTAPE is called to mount and position the Classification Results tapes.

3.0 CHANGE is called to read the header information from the two input files, locate the correct classification areas, and write the initial records to the output results file. It then reads the classified data, one line at a time from the two input files, calls COMPAR, and writes the output results.

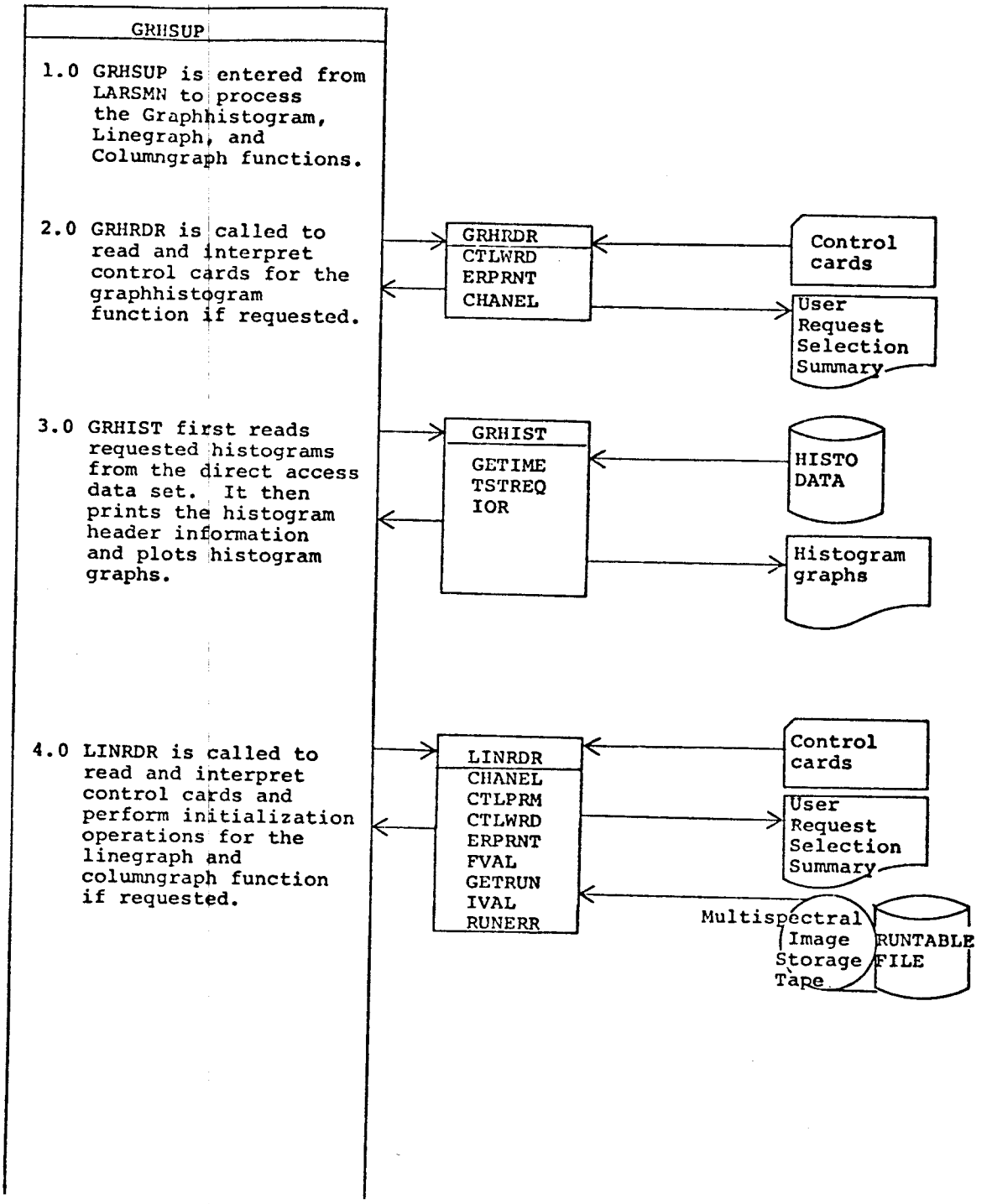
3.1 COMPAR compares each point of the two input lines using the CHANGE classes supplied by the user (on CLASS, FIRST, and SECOND cards) and assigns an output class number for each point of the line.

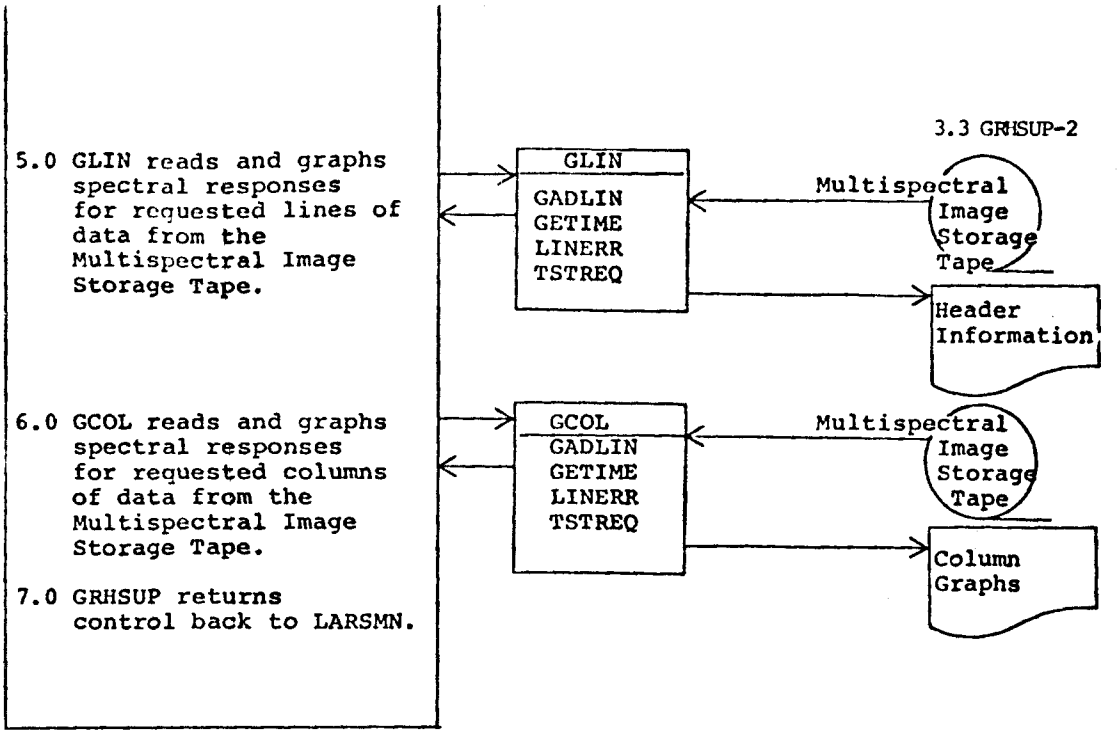
4.0 Control is returned to LARSMN.



Load Module Name: GRHSUP

3.3 GRHSUP-1




3.3 GRHSUP-2
5.0 GLIN reads and graphs spectral responses for requested lines of data from the Multispectral Image Storage Tape.
GLIN
GADLIN
GETIME
LINERR
TSTREQ
Multispectral Image Storage Tape
Header Information
6.0 GCOL reads and graphs spectral responses for requested columns of data from the Multispectral Image Storage Tape.
7.0 GRHSUP returns control back to LARSMN.
GCOL
GADLIN
GETIME
LINERR
TSTREQ
Multispectral Image Storage Tape
Column Graphs

Load Module Name: HISSUP

HISSUP

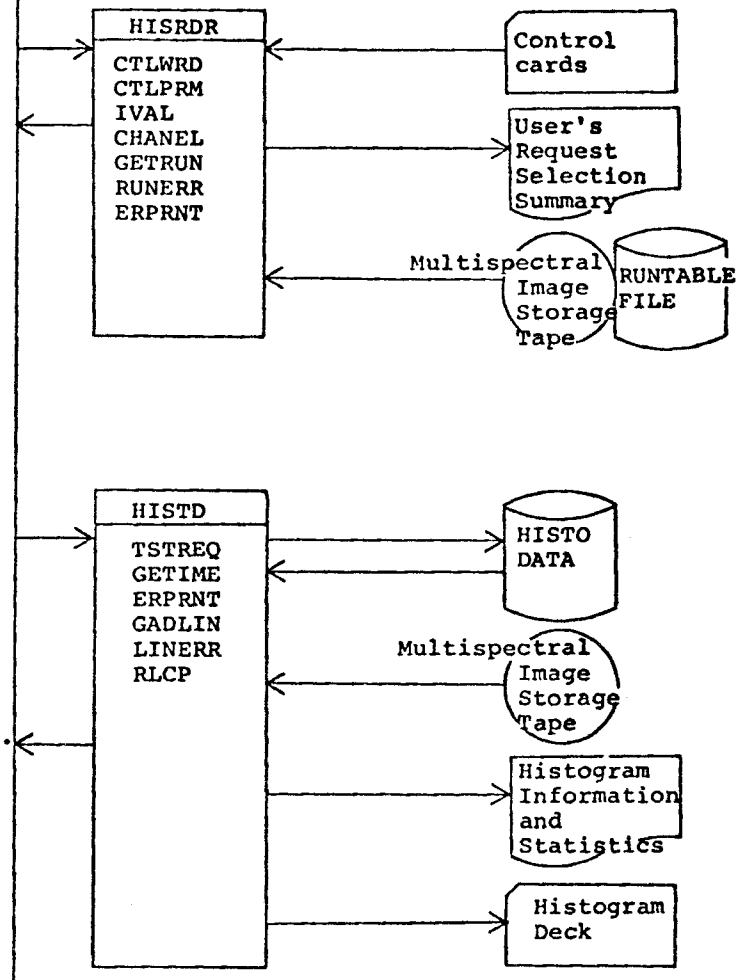
1.0 HISSUP is entered from LARSMN to process the Histogram function.

2.0 HISRDR is called to read control cards and perform initialization operations. A summary of the user's requests are printed. The correct Multispectral Image Storage Tape that contains the data to be histogrammed is located and mounted.

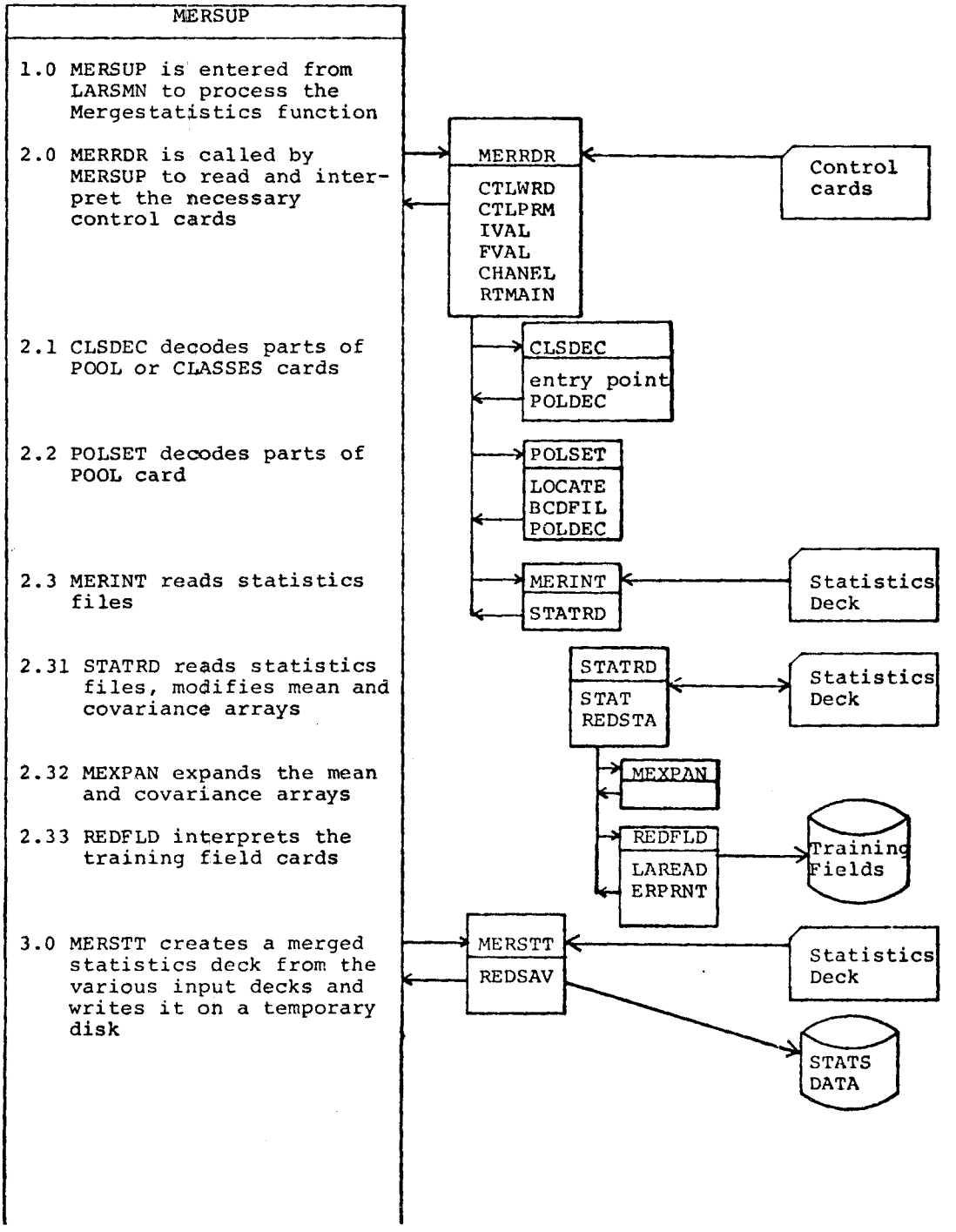
3.0 Space in ARRAY for 3 arrays is allocated.

4.0 HISTD is called to calculate the histograms. HISTD reads data from HISTO DATA to acquire previous histogram data if OPTIONS ACCUM was specified. The Multispectral Image Storage Tape is read for each line to be histogrammed. After calculations are complete, the HISTO DATA file is updated and the histogram information and statistics are printed. The histogram deck will be punched if PUNCH HIST was specified.

5.0 HISSUP returns control back to LARSMN.



Load Module Name: MERSUP



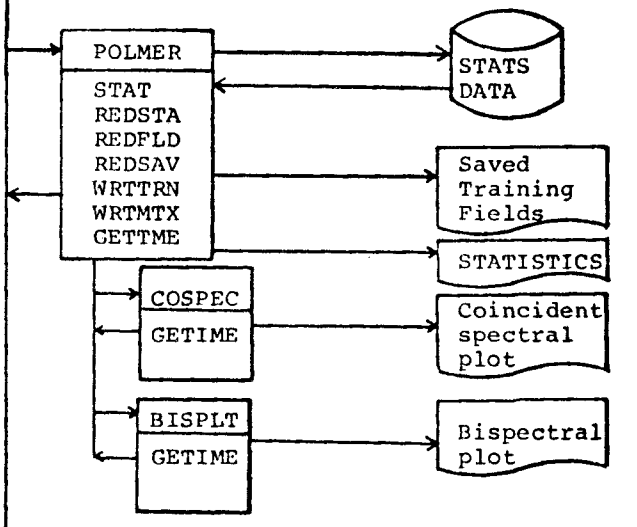
3.3 MERSUP-2

4.0 POLMER pools appropriate classes and writes new statdeck to disk, punches statistics if requested

4.1 COSPEC prints a coincident spectral plot of all the classes in a given statdeck

4.2 BISPLT prints a bispectral plot of all the classes in a given statdeck

5.0 MERSUP returns control to LARSMN



Load Module Name: PICSUP

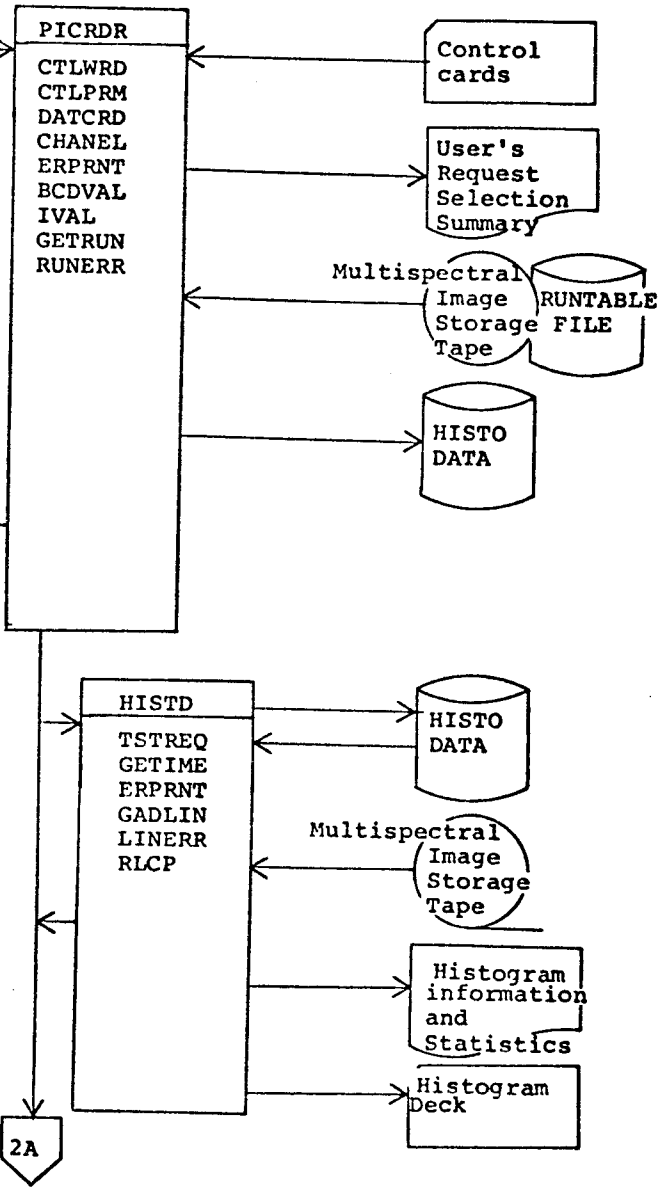
3.3 PICSUP-1

PICSUP

1.0 PICSUP is entered from LARSMN to process the Pictureprint function.

2.0 PICRDR is called to read control cards and perform initialization operations. A summary of the user's requests are printed. If HISTOGRAM COMPUTE was requested, the required tape must be mounted and then a CALL HISTD is performed. The correct data tape run to be displayed must also be mounted. An extra record is written to a disk file, HISTO DATA, to insure that it is formatted as a direct access file. If BOUNDARY DELETE or STORE was specified, a CALL BONDSU is executed.

2.1 HISTD will first read data from HISTO DATA to acquire previous histogram data. After the new data is histogrammed, the records are updated. The Multispectral Image Storage Tape is read for each line to be histogrammed. The histogram information and statistics are printed. The Histogram file will be punched if PUNCH HIST is specified and HISTOGRAM COMPUTE.

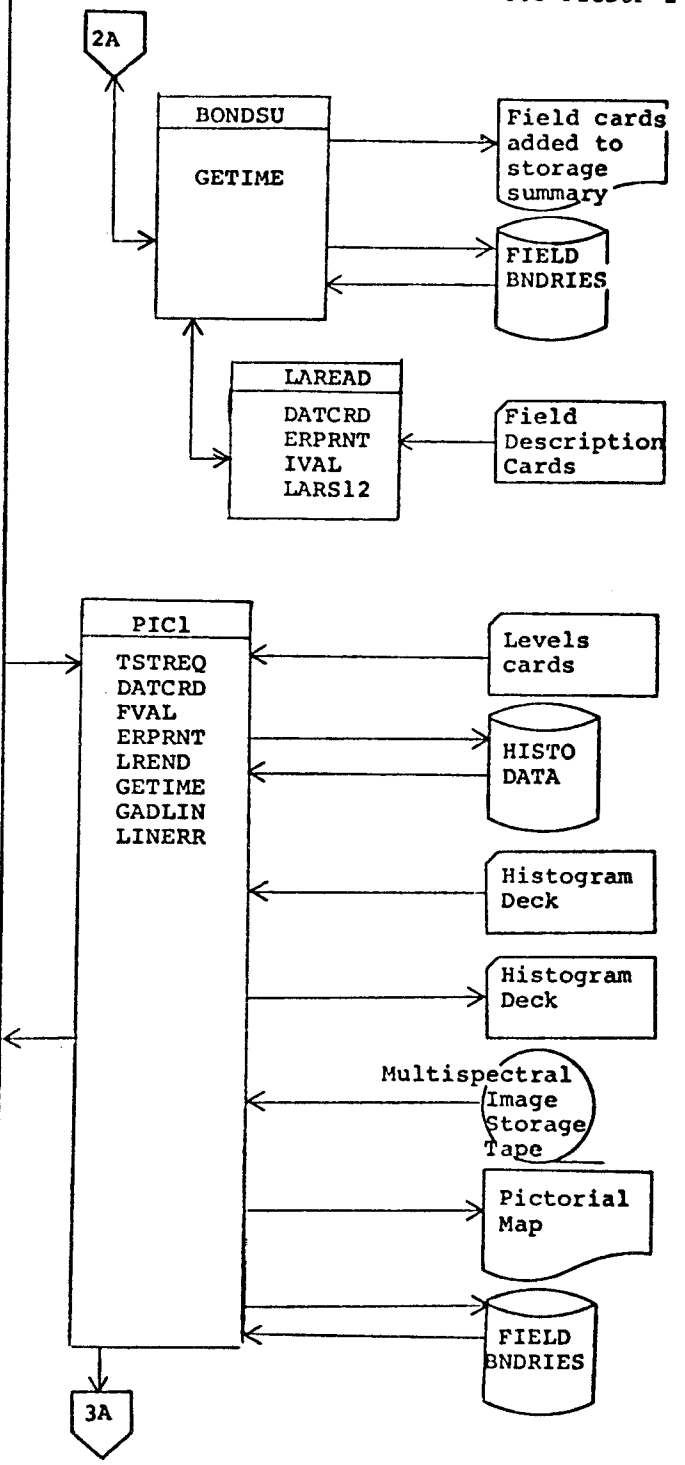


3.3 PICSUP-2

2.2 BONDSU will erase FIELD BNDRIES if DELETE is specified. If STORE is specified, existing records of FIELD BNDRIES are read and updated by Field Description Cards (the cards are read via LAREAD). A summary list of new boundaries is printed.

2.2.1 LAREAD reads the Field Description Cards.

3.0 PIC1 is called to generate the gray scale prints. If BOUNDARY OUTLINE is specified, FIELD BNDRIES is read. If HISTOGRAM LEVELS CARDS is specified, the levels cards are read. If HISTOGRAM HISTOCARDS is specified, the histograms deck is read. Otherwise the HISTO DATA file is read. If PRINT HIST is specified, a CALL GRHIST is performed. If PUNCH HIST is specified and not HISTOGRAM COMPUTE, the histogram file is punched. The Multispectral Image Storage Tape is read for each line to be displayed. If boundaries are to be displayed, a CALL FLDBOR is executed. The pictorial map is then printed.

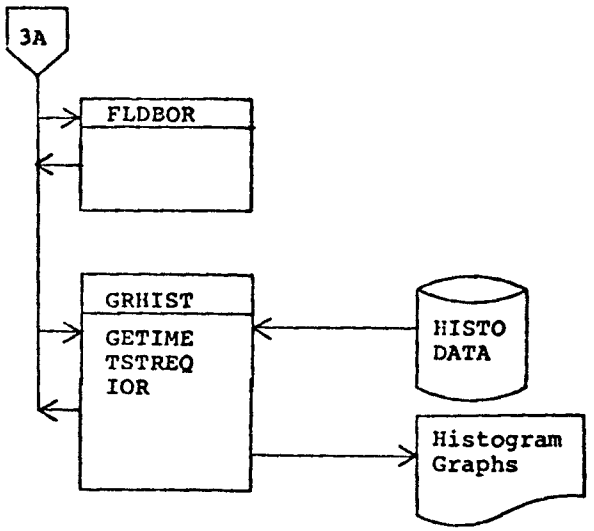


3.3 PICSUP-3

3.1 FLDBOR will insert boundary symbols into a given line of data to be printed.

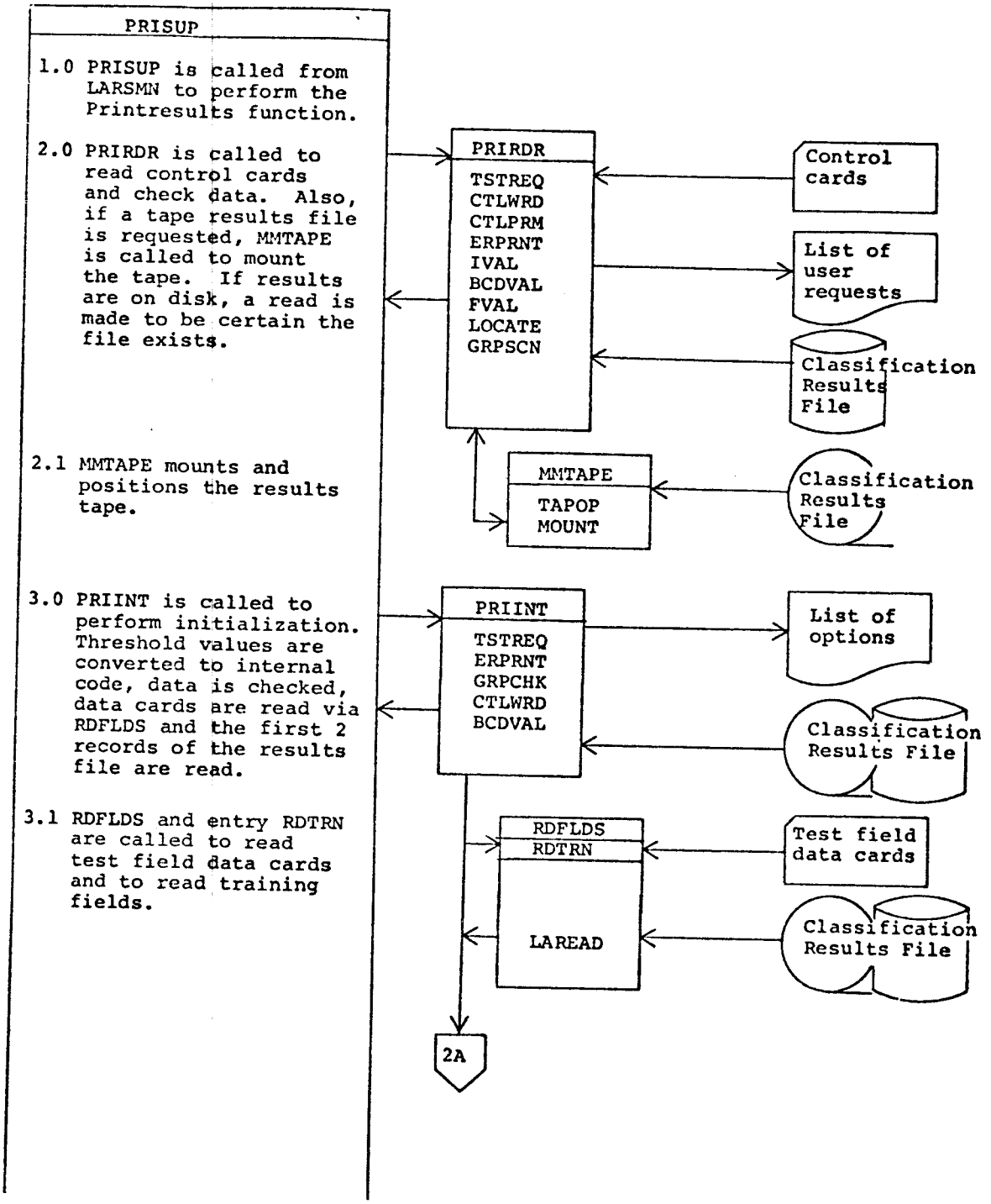
3.2 GRHIST reads the histogram data from HISTO DATA and prints the graphs.

4.0 PICSUP returns control back to LARSMN.



Load Module Name: PRISUP

3.3 PRISUP-1



3.3 PRISUP-2

3.2 STATS is called to read record type 4 from the results file and print the statistics summary if requested.

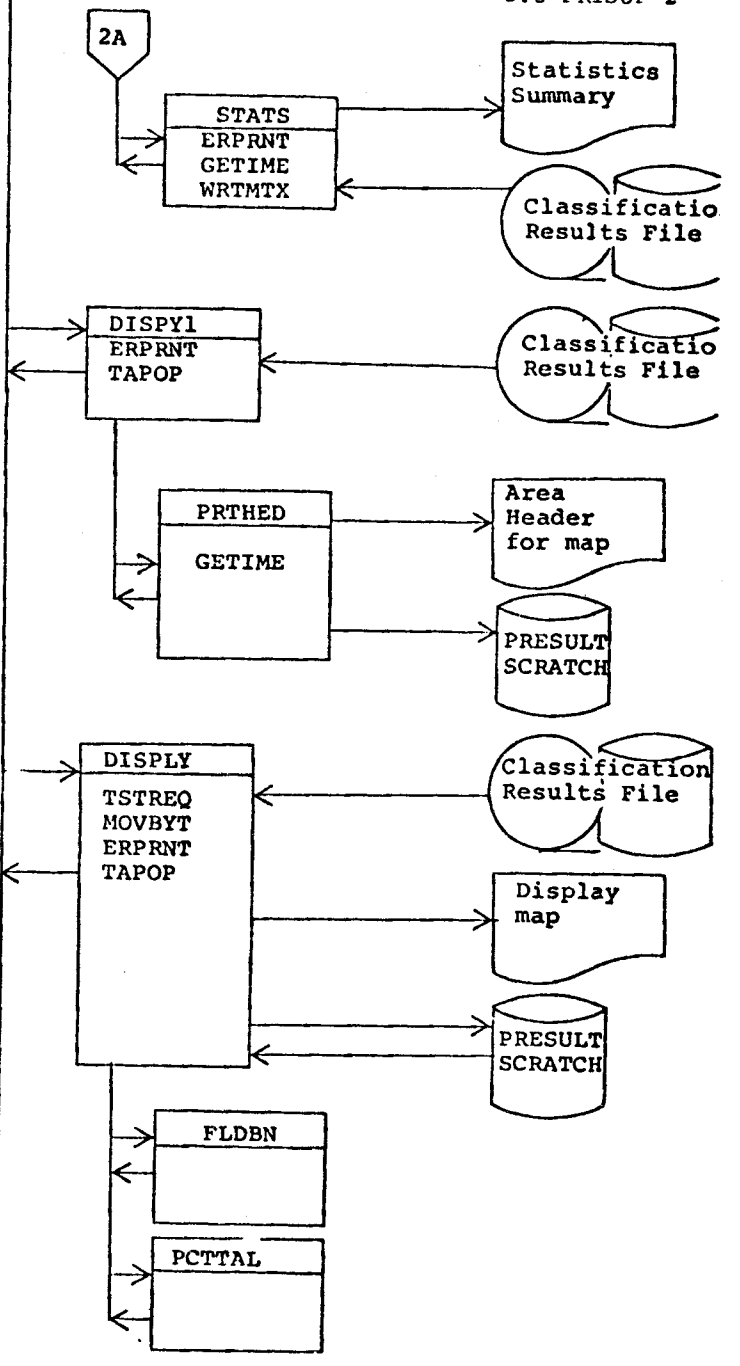
4.0 DISPY1 is called to position the results file to the next area of interest. Indexes are computed and the area header printed via PRTHED.

4.1 PRTHED prints (and may write on PRESULT SCRATCH) the header containing run, channel and classes information.

5.0 DISPLY generates (and may write on PRESULT SCRATCH) the display map for an area and calls PCTTAL to tally performance data.

5.1 FLDBN is called to place outline symbols in the display map (called for each line). Used only if outlining requested.

5.2 PCTTAL computes performance data for training on test fields (called for each line). Called only when performance tables requested.



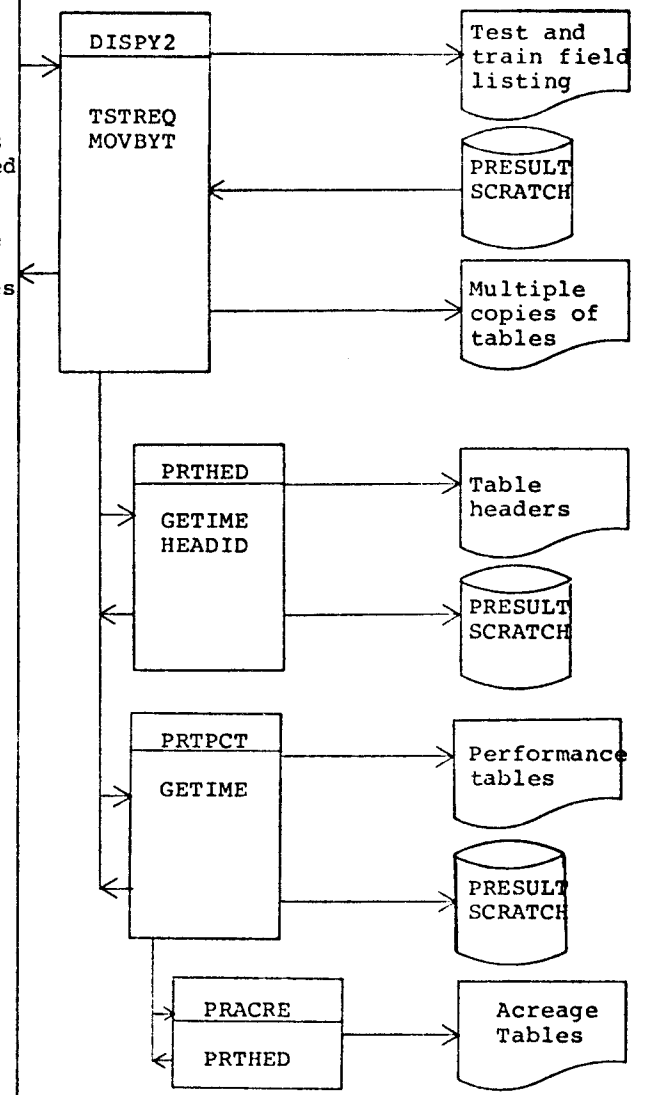
6.0 DISPY2 is called to control printing of the performance tables. The headers are printed by PRTHED and the tables are generated and printed by PRTPCT. If multiple copies were requested, PRTHED will write to the scratch disk and DISPY2 will print off the copies

6.1 PRTHED prints (or writes to disk) the headers for tables.

6.2 PRTPCT prints (or writes to disk) the performance tables. Called once for each kind of table.

6.2.1 PRACRE calculates and prints the tables of acres and hectares for test field.

7.0 PRISUP returns control to LARSMN.



Load Module Name: RATSUP

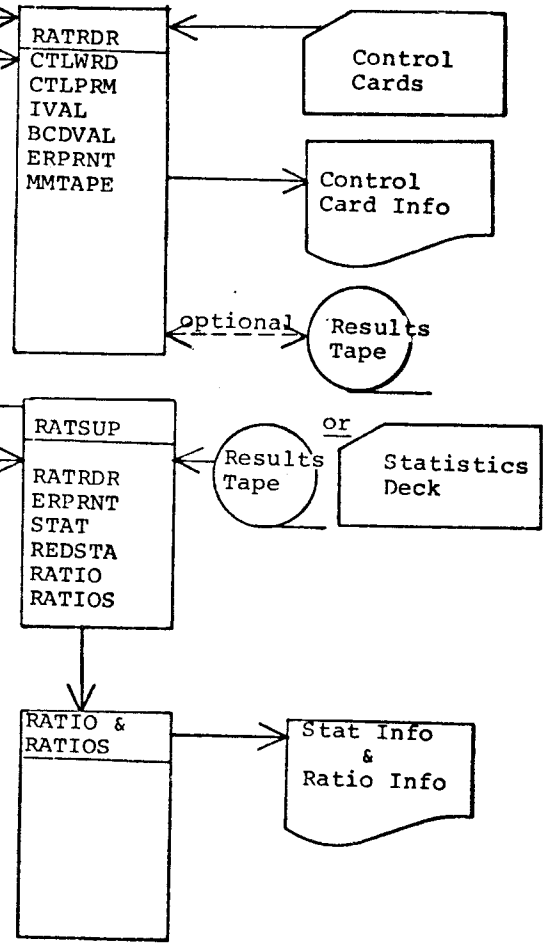
RATSUP

1.0 RATSUP is entered from LARSMN to process the Ratiomeans function.

2.0 RATRDR is called to read control cards. A list of the control cards along with some interpreted requests are printed. The results tape will be mounted if stats are on tape.

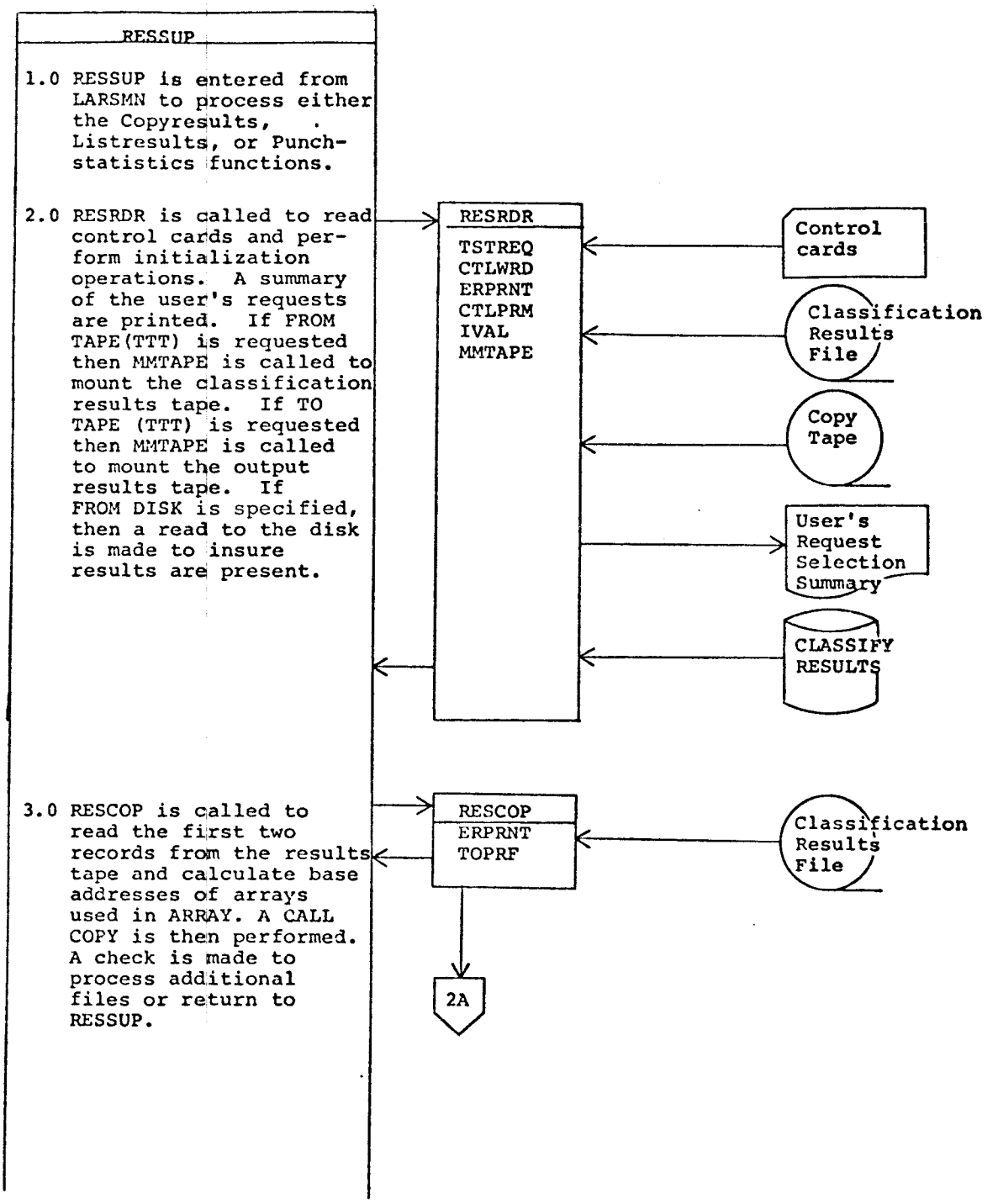
3.0 Control is passed back to RATSUP. Here the preliminary calculations are made and the statistics deck is read. REDSTA is called to reduce the statistics.

4.0 Routines RATIO & RATIOS calculate the ratio in the first four channels in the stat deck, and the channels in the visible region & infrared region respectively supplied by the user.

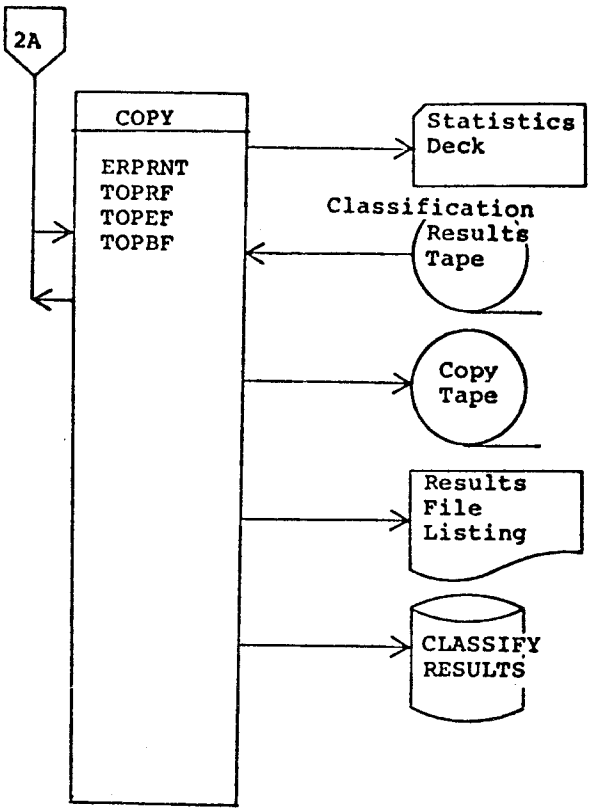


Load Module Name: RESSUP

3.3 RESSUP-1



3.3 RESSUP-2



3.1 COPY reads first two records from results tape. Header information is printed on the results file listing. If COPYRESULTS was requested, two records are written on Copy tape if TAPE specified and on disk if DISK specified. Statistics records are processed and copied if requested. If PUNCHSTATISTICS requested, then a deck is punched. The remainder of the file is then read or read and copied. If NOLIST specified, control is returned to RESCOP. If not, then a results file listing is printed. If COPYRESULTS requested, then the copy tape is terminated by 2 file marks, a check record, another file mark. Return is made to RESCOP.

4.0 RESSUP returns control back to LARSMN.

Load Module Name: RUNSUP

3.3 RUNSUP-1

RUNSUP

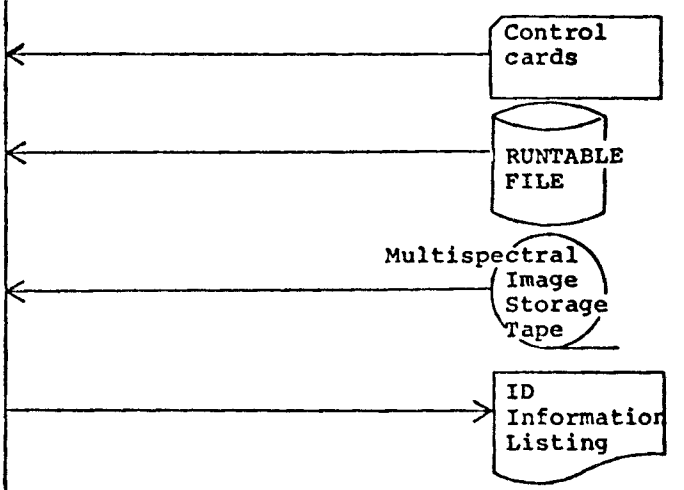
1.0 RUNSUP is entered from LARSMN to process the Idprint, Channeltransform and transferdata functions. The error handling routine for GADLIN (called in TRAXEQ) is contained in RUNSUP and calls these ROOT subroutine:

LINERR
ERPRNT
TOPBS.

2.0 The Idprint function is contained in RUNSUP and calls these ROOT subroutines:

TSTREQ
CTLWRD
ERPRNT
CTLPRM
IVAL
GETIME
TOPRW
TOPRU
MOUNT
TOPRD
TOPFF

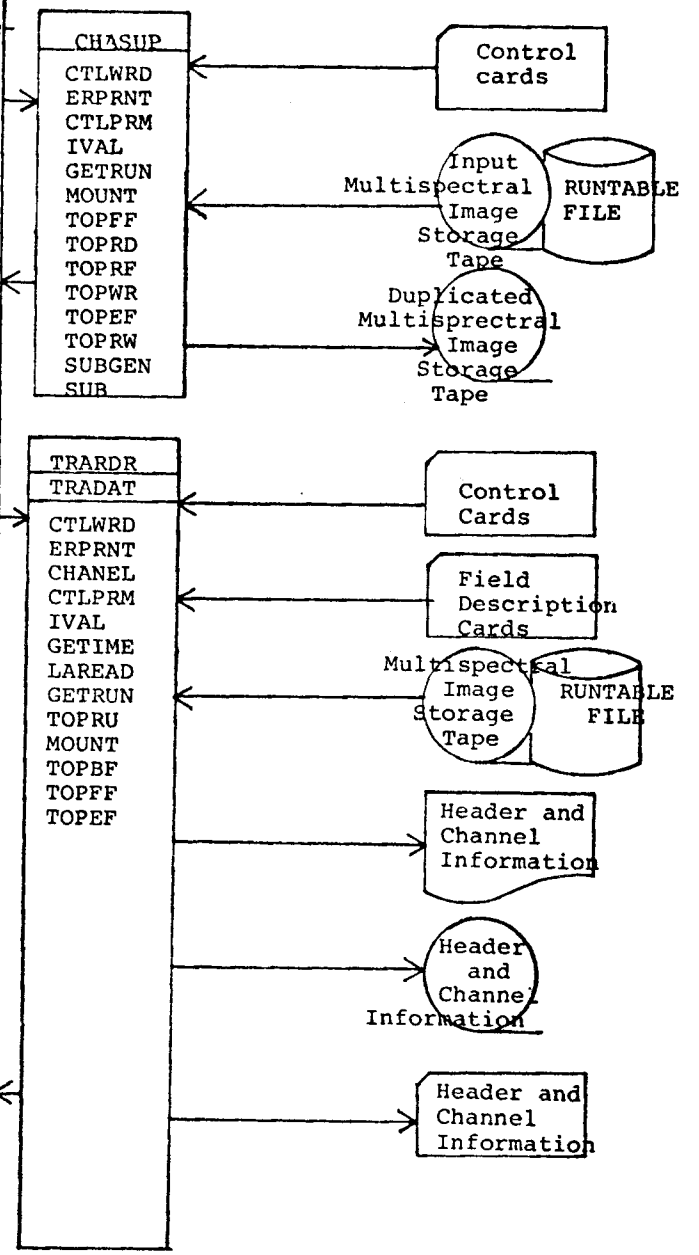
Control cards for the Idprint function are read in. If PRINT TAPE was requested, a Multispectral Image Storage Tape is mounted and the ID records on this tape are printed out. If PRINT RUN or PRINT ALL was requested, the system runtable is accessed to obtain the ID record(s), and the record or records are printed out.



3.0 CHASUP is entered if the Channeltransform function was requested. It is a combination card reader and supervisor. A Multispectral Image Storage Tape is mounted and an output tape is written which is a Multispectral Image Storage Tape containing the duplicated run.

4.0 TRARDR is entered if the Transferdata function was requested. It reads in control cards, reads the first field description card, mounts the Multispectral Image Storage Tape, and prints the header and channel information. If the TAPE and/or PUNCH options are specified, it also records on tape and/or punched cards the same information. Control is then returned to RUNSUP which will call TRAXEQ to output the roll parameters and/or the data values on the Multispectral Image Storage Tape.

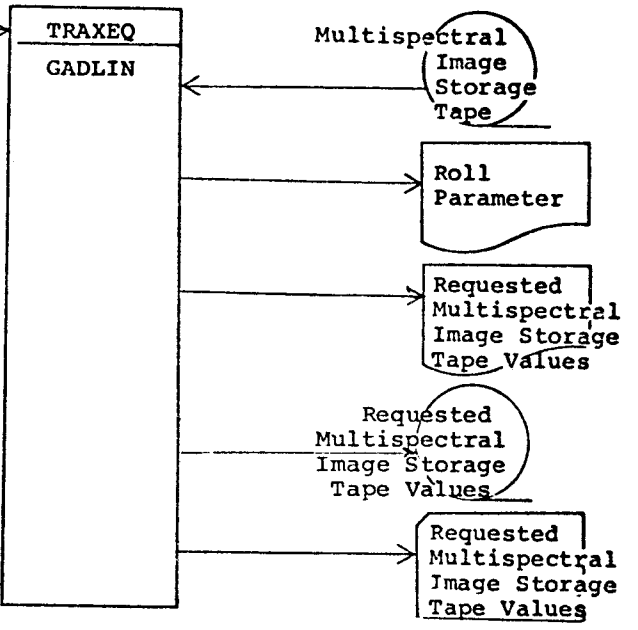
TRADAT, the second entry point, is entered each time control is returned to RUNSUP from TRAXEQ. It will read all subsequent Field Description cards and will mount tapes output the channel and header information for these runs. The only difference between this entry point and TRARDR



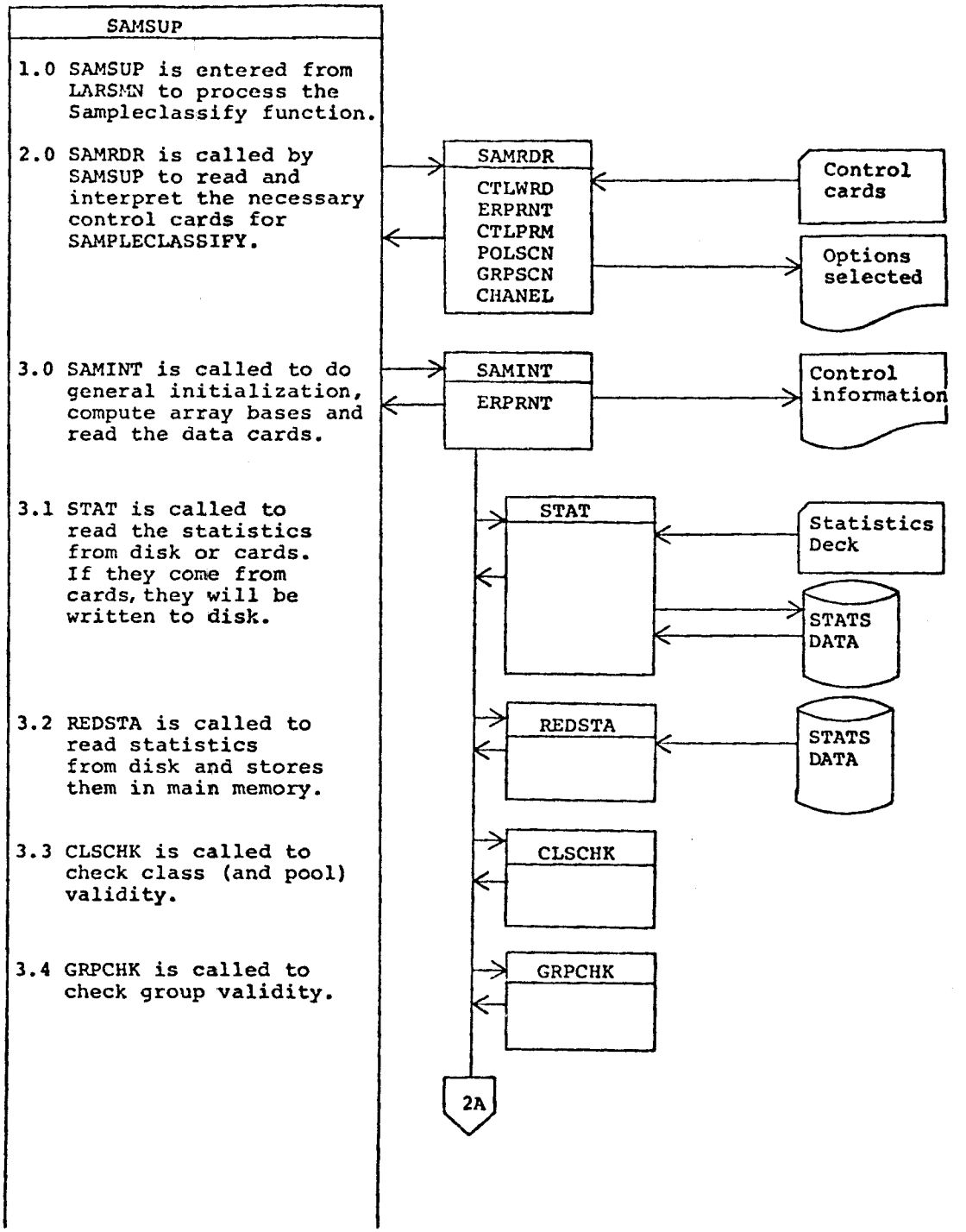
is that the latter reads the control cards. TRADAT will eventually read the END card and notify RUNSUP of the end of the input deck.

5.0 TRAXEQ is entered to print the roll parameter and print, punch, and record on tape the Multispectral Image Storage Tape values, depending upon the user's specifications. The Multispectral Image Storage Tape is already mounted and left positioned at the first data line by TRARDR or TRADAT. Control is returned to RUNSUP after the completion of each run.

6.0 RUNSUP returns control to LARSMN.



Load Module Name: SAMSUP



3.3 SAMSUP-2

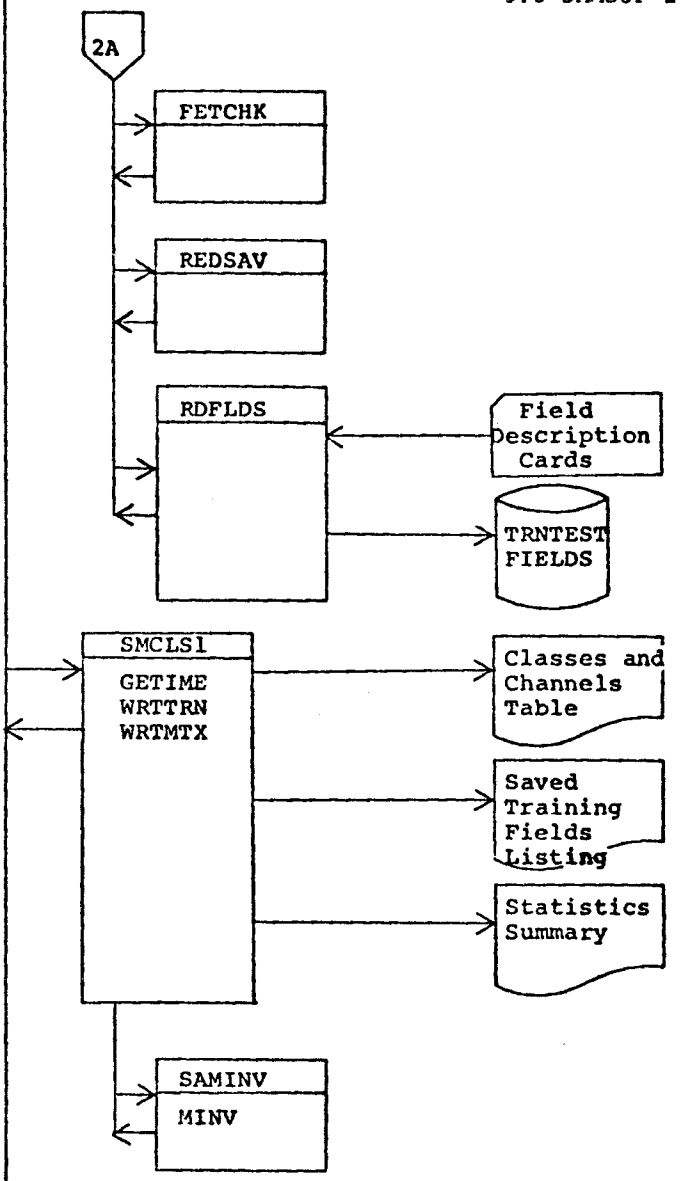
3.5 FETCHK is called to check channel validity.

3.6 REDSAV is called to reduce statistics into classification pools.

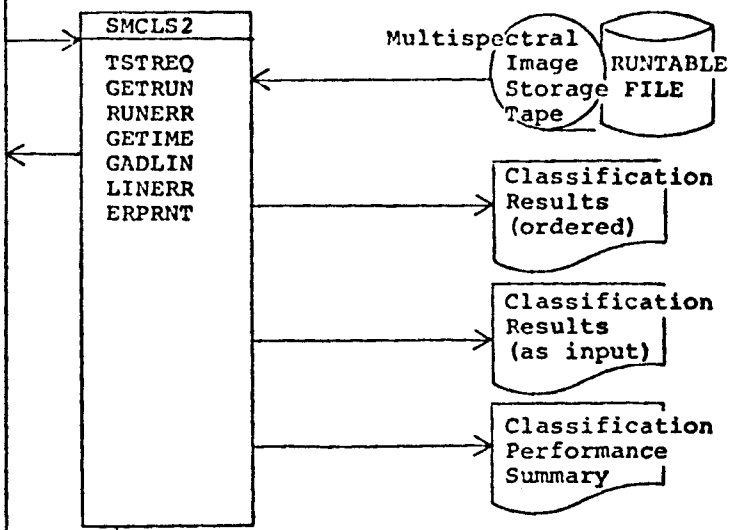
3.7 RDFLDS is called to read and sort the Field Description Cards.

4.0 SMCLS1 is called to output the statistics information and also to invert the covariance matrix and calculate the determinant.

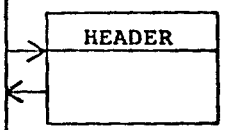
4.1 SAMINV is called to invert the covariance matrix and calculate the determinant.



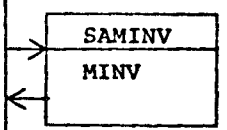
5.0 SMCLS2 is called to perform classification on a per field basis.



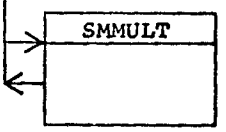
5.1 HEADER is called to print the run information header from the Multispectral Image Storage Tape.



5.2 SAMINV is called to find the inverse and determinant of a matrix.

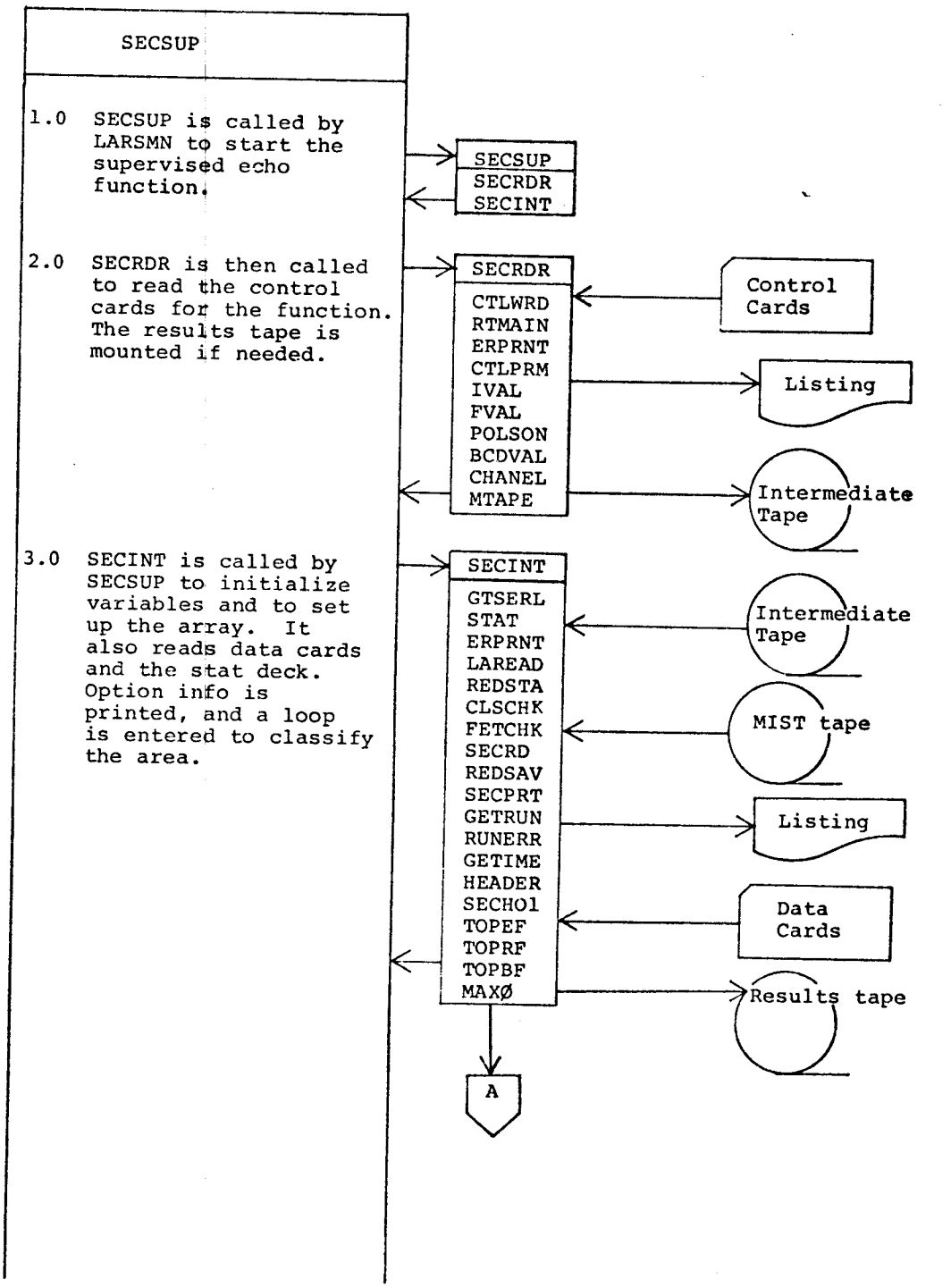


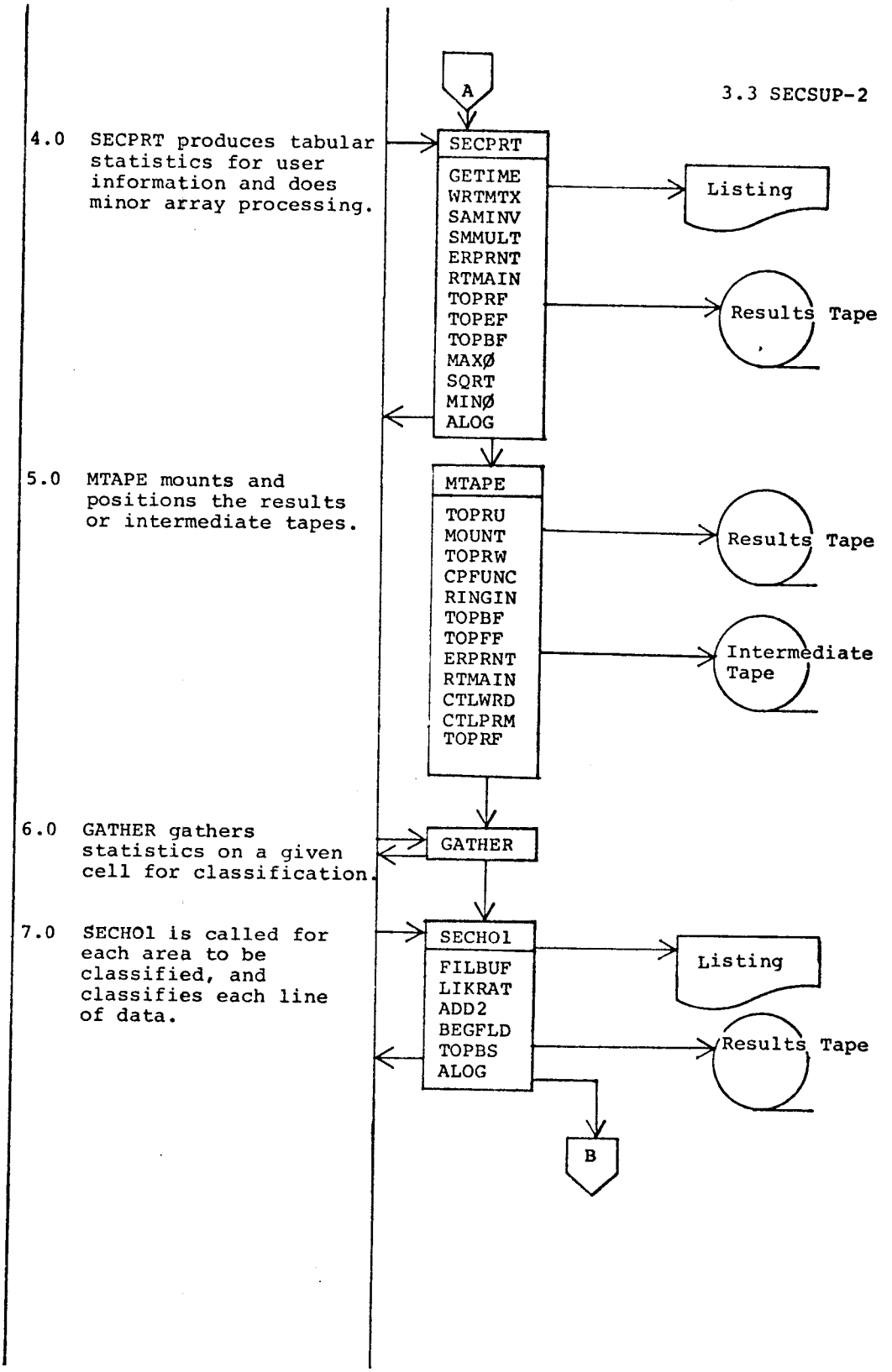
5.3 SMMULT is called to compute vector and scalar products of a matrix.

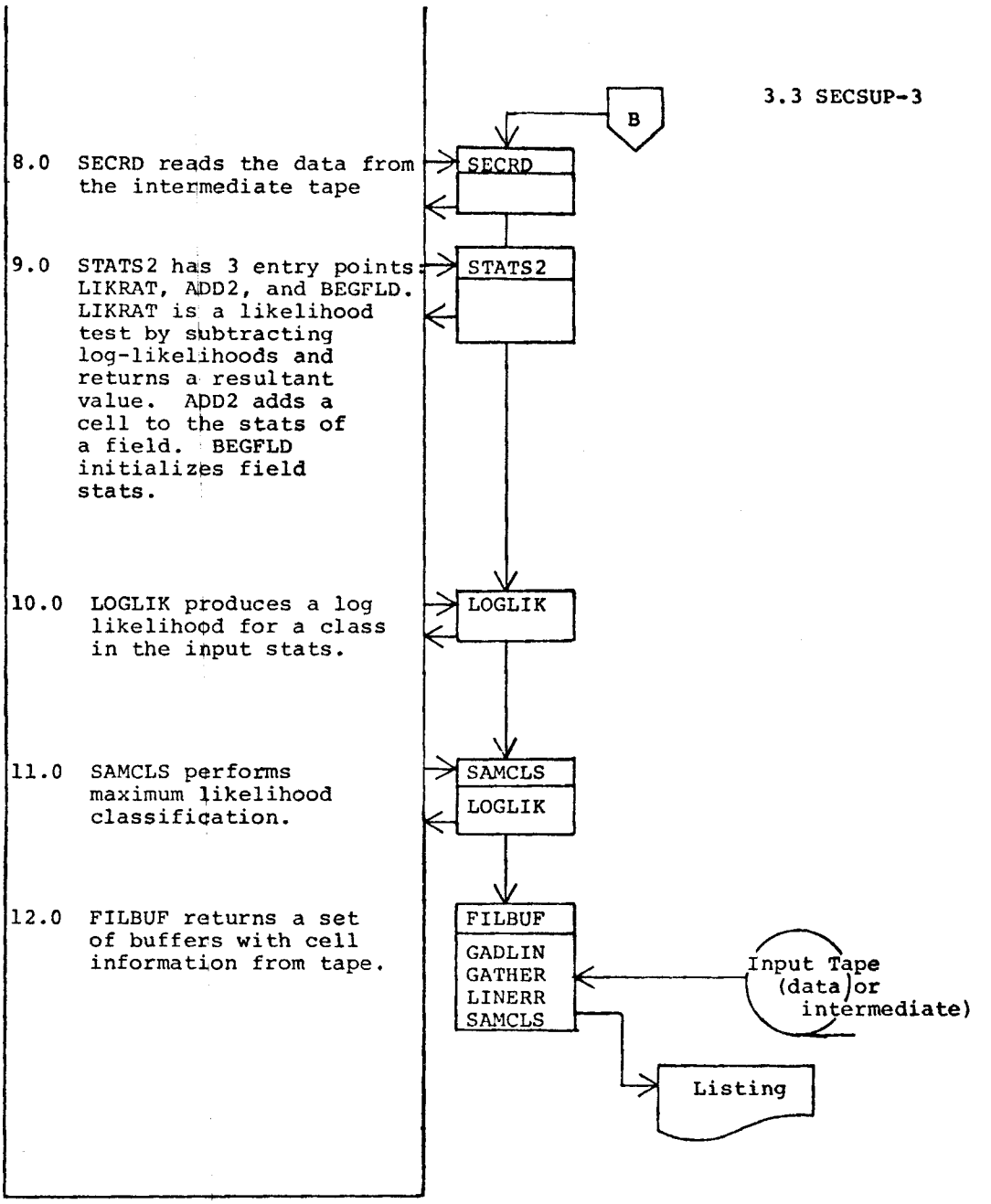


6.0 SAMSUP returns control back to LARSMN.

Load Module Name: SECSUP

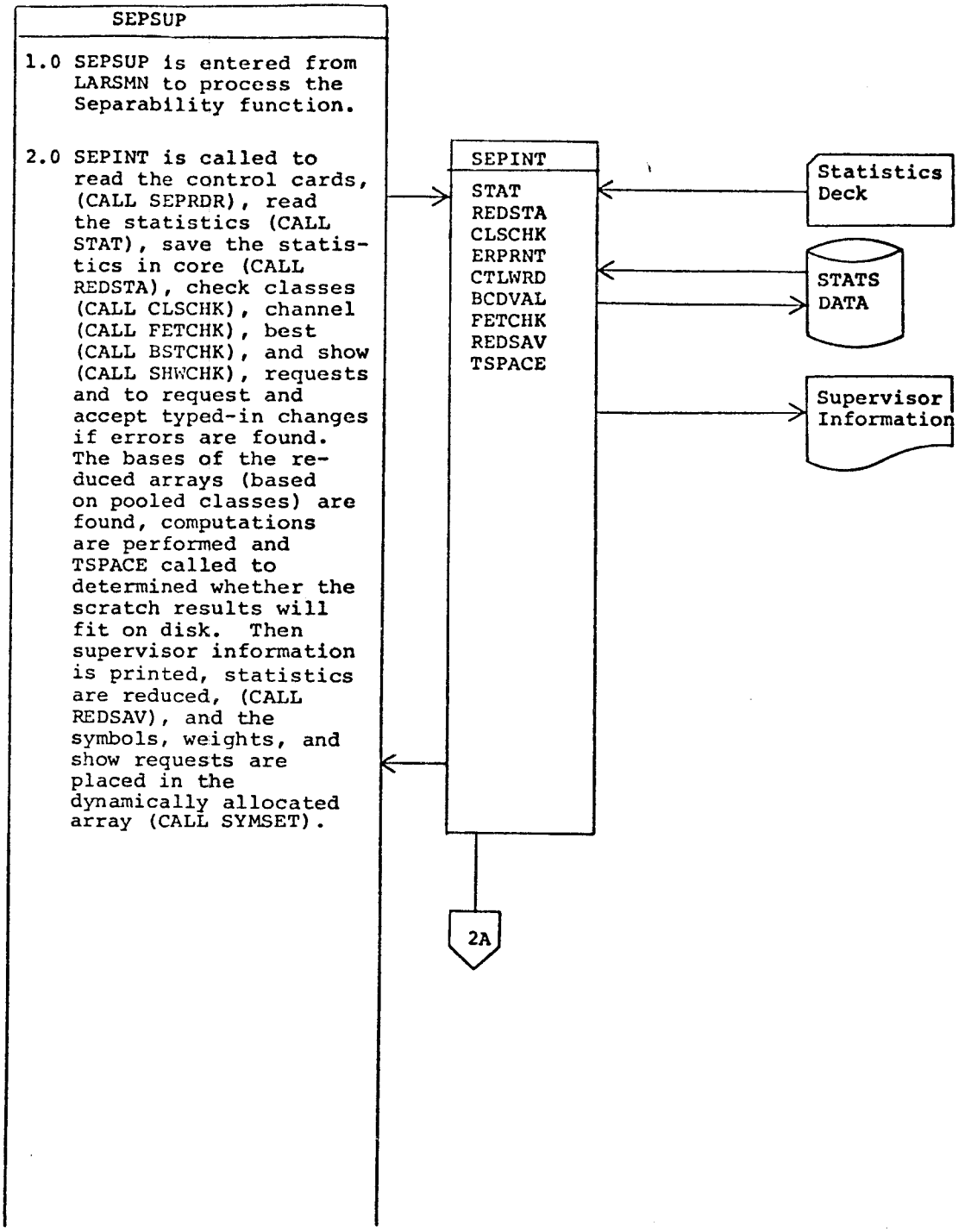



A
3.3 SECSUP-2
4.0 SECPRT produces tabular statistics for user information and does minor array processing.
SECPRT
GETIME
WRTMTX
SAMINV
SMMULT
ERPRNT
RTMAIN
TOPRF
TOPEF
TOPBF
MAXØ
SQRT
MINØ
ALOG
Listing
Results Tape
5.0 MTAPE mounts and positions the results or intermediate tapes.
MTAPE
TOPRU
MOUNT
TOPRW
CPFUNC
RINGIN
TOPBF
TOPFF
ERPRNT
RTMAIN
CTLWRD
CTLPRM
TOPRF
Results Tape
Intermediate Tape
6.0 GATHER gathers statistics on a given cell for classification.
GATHER
7.0 SECHO1 is called for each area to be classified, and classifies each line of data.
SECHO1
FILBUF
LIKRAT
ADD2
BEGFLD
TOPBS
ALOG
Listing
Results Tape
B


3.3 SECSUP-3
B
8.0 SECRD reads the data from the intermediate tape
SECRD
9.0 STATS2 has 3 entry points LIKRAT, ADD2, and BEGFLD. LIKRAT is a likelihood test by subtracting log-likelihoods and returns a resultant value. ADD2 adds a cell to the stats of a field. BEGFLD initializes field stats.
STATS2
10.0 LOGLIK produces a log likelihood for a class in the input stats.
LOGLIK
11.0 SAMCLS performs maximum likelihood classification.
SAMCLS
LOGLIK
12.0 FILBUF returns a set of buffers with cell information from tape.
FILBUF
GADLIN
GATHER
LINERR
SAMCLS
Input Tape (data or intermediate)
Listing

Load Module Name: SEPSUP

3.3 SEPSUP-1

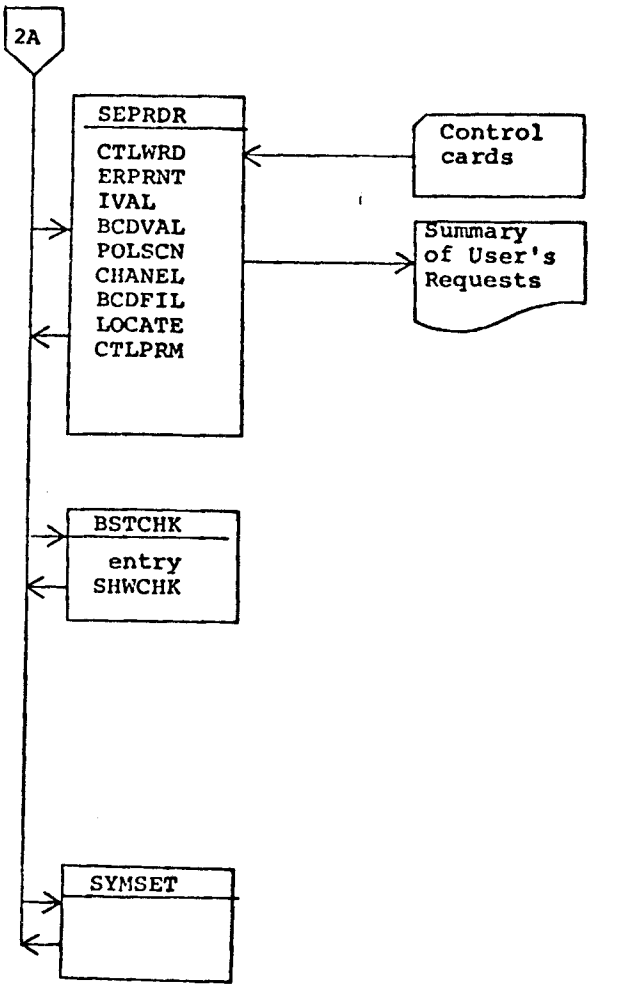


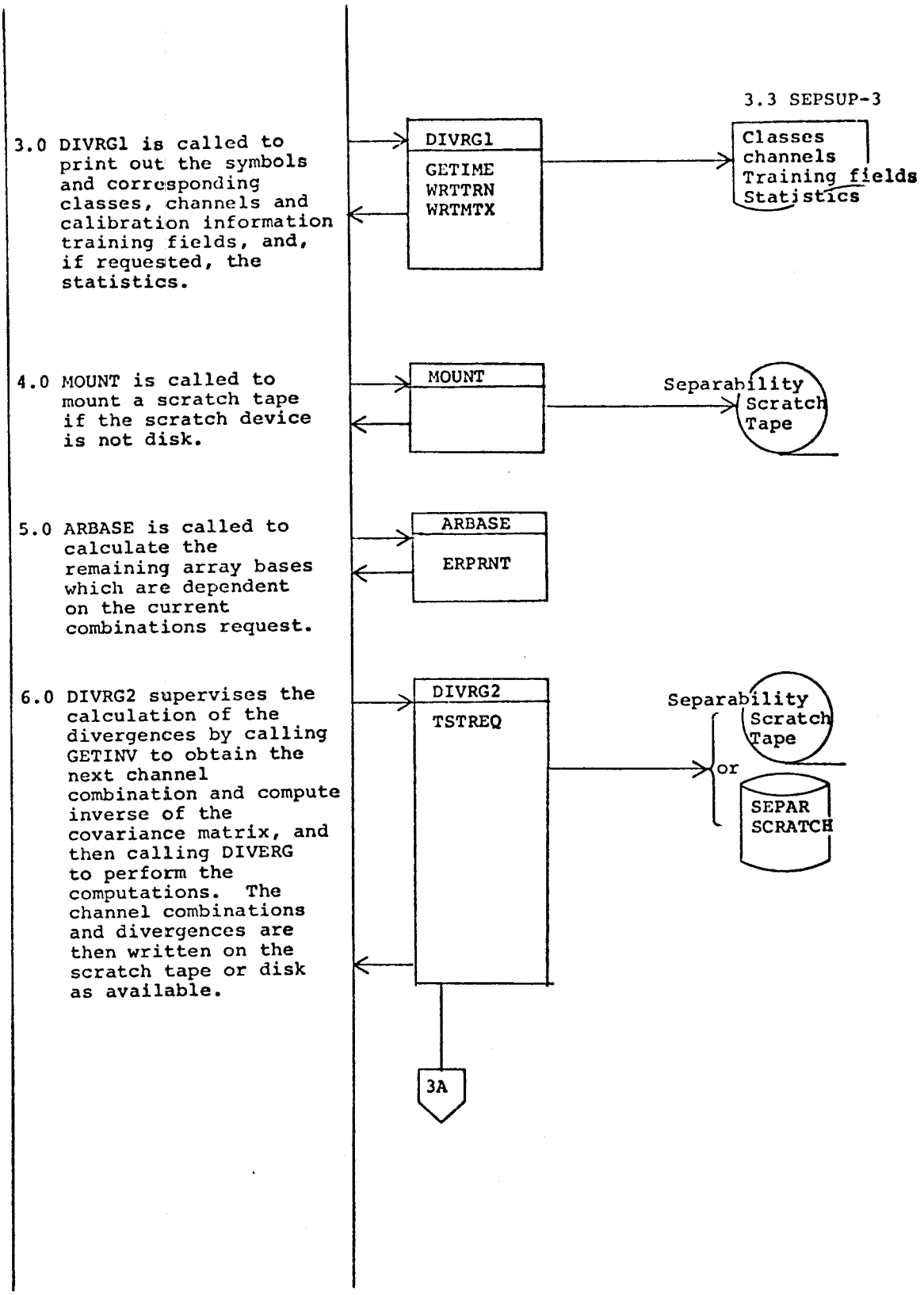
3.3 SEPSUP-2

2.1 SEPRDR is called to perform initialization operations and read and decode the control cards. The control cards and options requested are printed on the printer. If errors are detected, the card and error message will appear both at the terminal and on the printer.

2.2 BSTCHK deletes any combinations request which is larger than the number of channels in the given data. Entry point SHWCHK examines each show request for having channels in ascending order and valid channels listed.

2.3 SYMSET is called to set up all the symbol combinations and place valid weight requests and show requests in the dynamically allocated array. The weight is ignored for an invalid class combination and this message appears on the printer and typewriter.

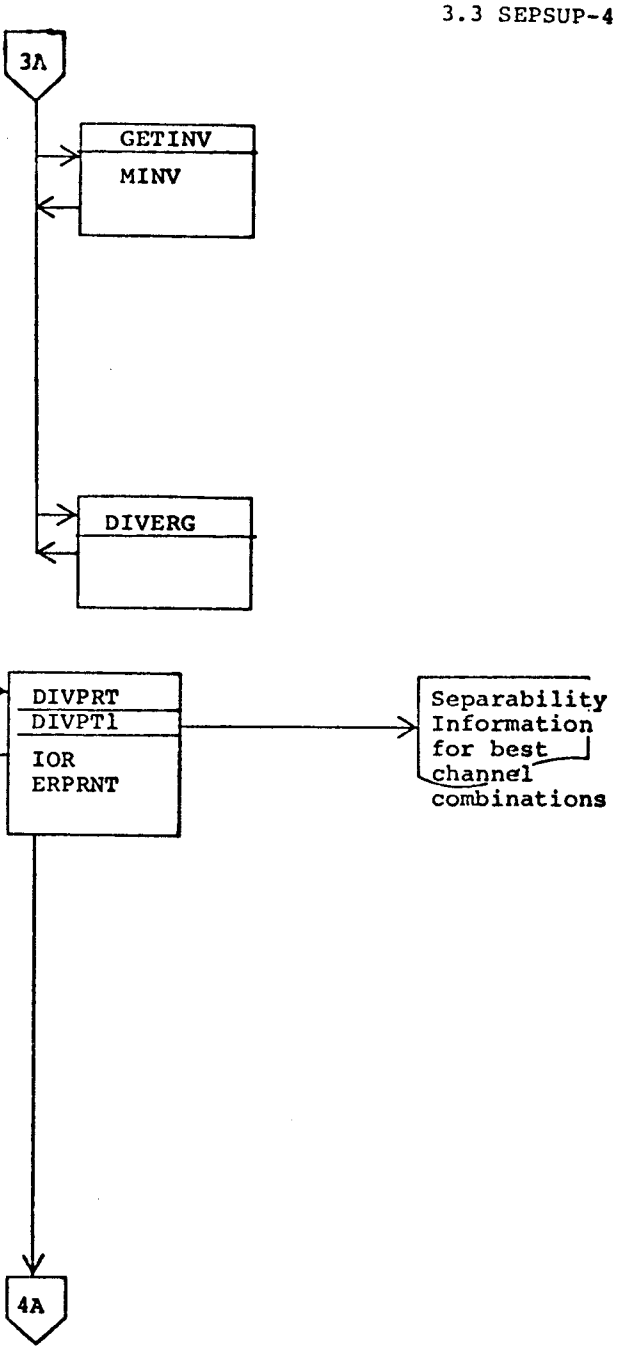



3.0 DIVRG1 is called to print out the symbols and corresponding classes, channels and calibration information training fields, and, if requested, the statistics.
DIVRG1
GETIME
WRTTRN
WRTMTX
3.3 SEPSUP-3
Classes
channels
Training fields
Statistics
4.0 MOUNT is called to mount a scratch tape if the scratch device is not disk.
MOUNT
Separability
Scratch
Tape
5.0 ARBASE is called to calculate the remaining array bases which are dependent on the current combinations request.
ARBASE
ERPRNT
6.0 DIVRG2 supervises the calculation of the divergences by calling GETINV to obtain the next channel combination and compute inverse of the covariance matrix, and then calling DIVERG to perform the computations. The channel combinations and divergences are then written on the scratch tape or disk as available.
DIVRG2
TSTREQ
Separability
Scratch
Tape
or
SEPAR
SCRATCH
3A

6.1 GETINV is called to obtain the next set of channels and compute the inverse of the covariance matrix. If the determinant is zero, a message that the channel statistics for some class are ill-conditioned is printed at the printer and typewriter.

6.2 DIVERG is called to compute the divergence for each class combination.

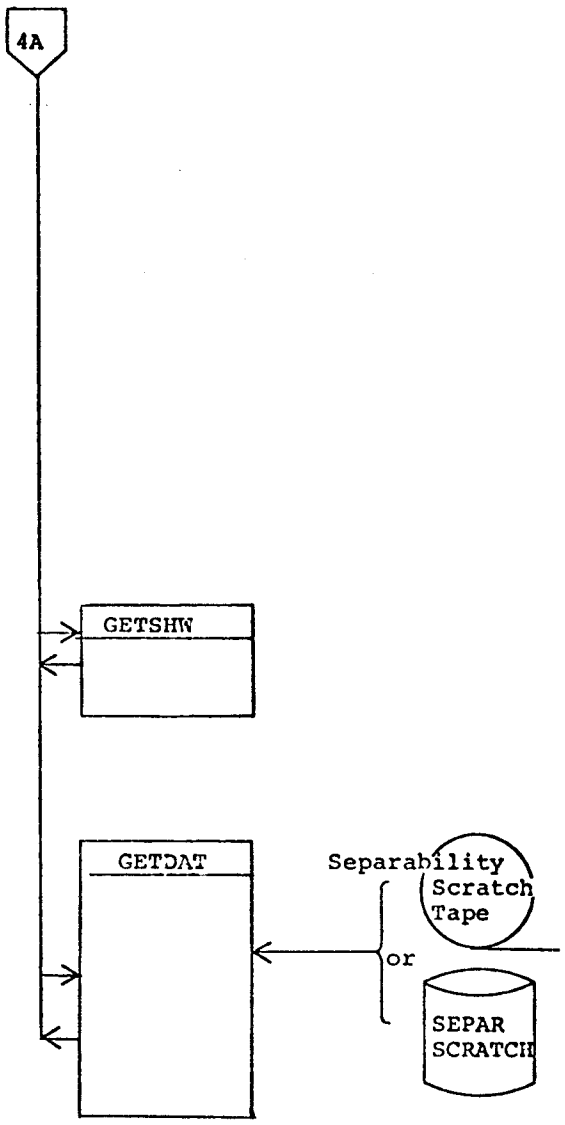
7.0 DIVPRT is called to print the separability of the classes by use of the divergences just calculated. The show requests are obtained (CALL GETSHW), the weights are saved, and page formats set up. GETDAT is called to obtain the needed divergences from the scratch device and to apply options requested from control cards. The results are ordered, headings printed, and numbers set up and printed 1 line at a time. The best set of



channels is saved before a return is executed. Entry DIVPT1 is called after the user has entered options at the typewriter, and the above sequence is followed as required by changes in options requested.

7.1 GETSHW searches through SPLCOM array for all show requests with the correct number of channels and places them in SAVQUE array.

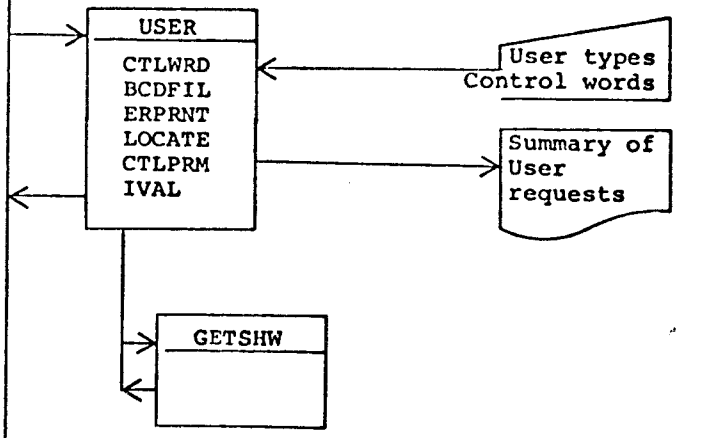
7.2 GETDAT obtains the separability information from the scratch device, applies the saturating transform if SATFLG=0, checks for excluded combinations and show requests, orders results by DIJ(MIN) if MINFLG=1, and returns the infomation to DIVPRT.



3.3 SEPSUP-6

8.0 USER allows the user to type in different sets of options at the terminal, using the same control words as for card input, plus the following control words: RESET to get back to the initial set of options, and HELP and TABLE to get information at the terminal. If a new SHOW request is entered, GETSHW adds it to the set of SHOW requests with the proper number of channels.

9.0 SEPSUP calls entry DIVPT1 if the user did not type STOP. Otherwise, the best set of channels is saved and control returns to LARSMN.



Load Module Name: SMOSUP

3.3 SMOSUP-1

SMOSUP

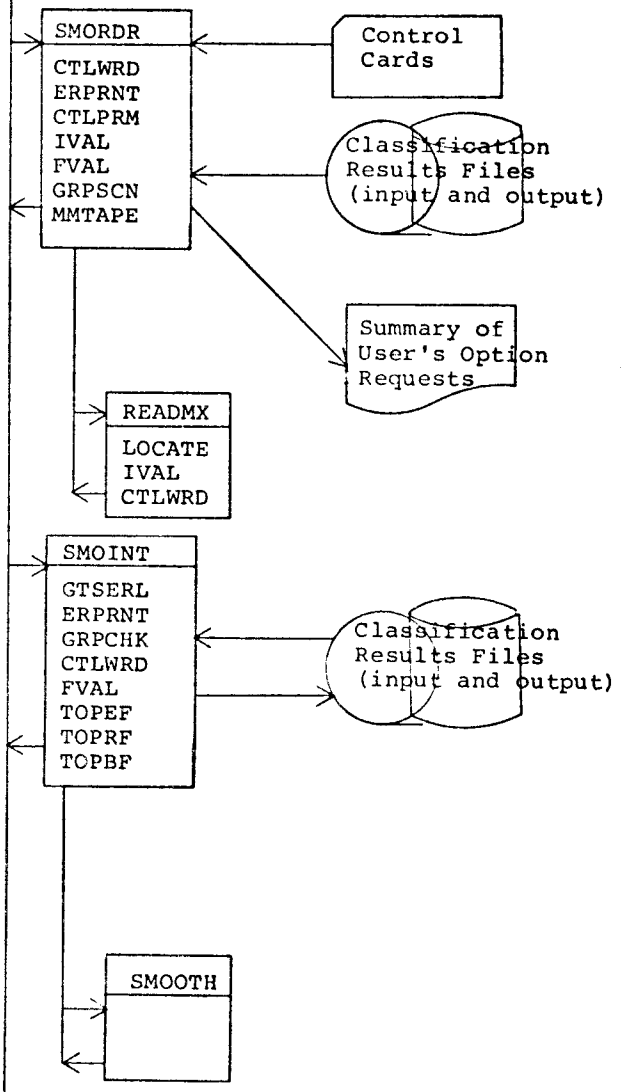
1.0 SMOSUP is entered from LARSMN to process the SMOOTHRESULTS function.

2.0 SMORDR is called to read control cards and to initialize variables and arrays. GRPSCN is called to interpret a group card and READMX interprets the user defined classes from a MIXCLASS card. MMTAPE is called to mount the input and/or output results tape. If INRESULTS DISK is specified, then a read to the disk is made to be sure results are present. A summary of the options requested is printed.

2.1 READMX scans and interprets the MIXCLASS control card.

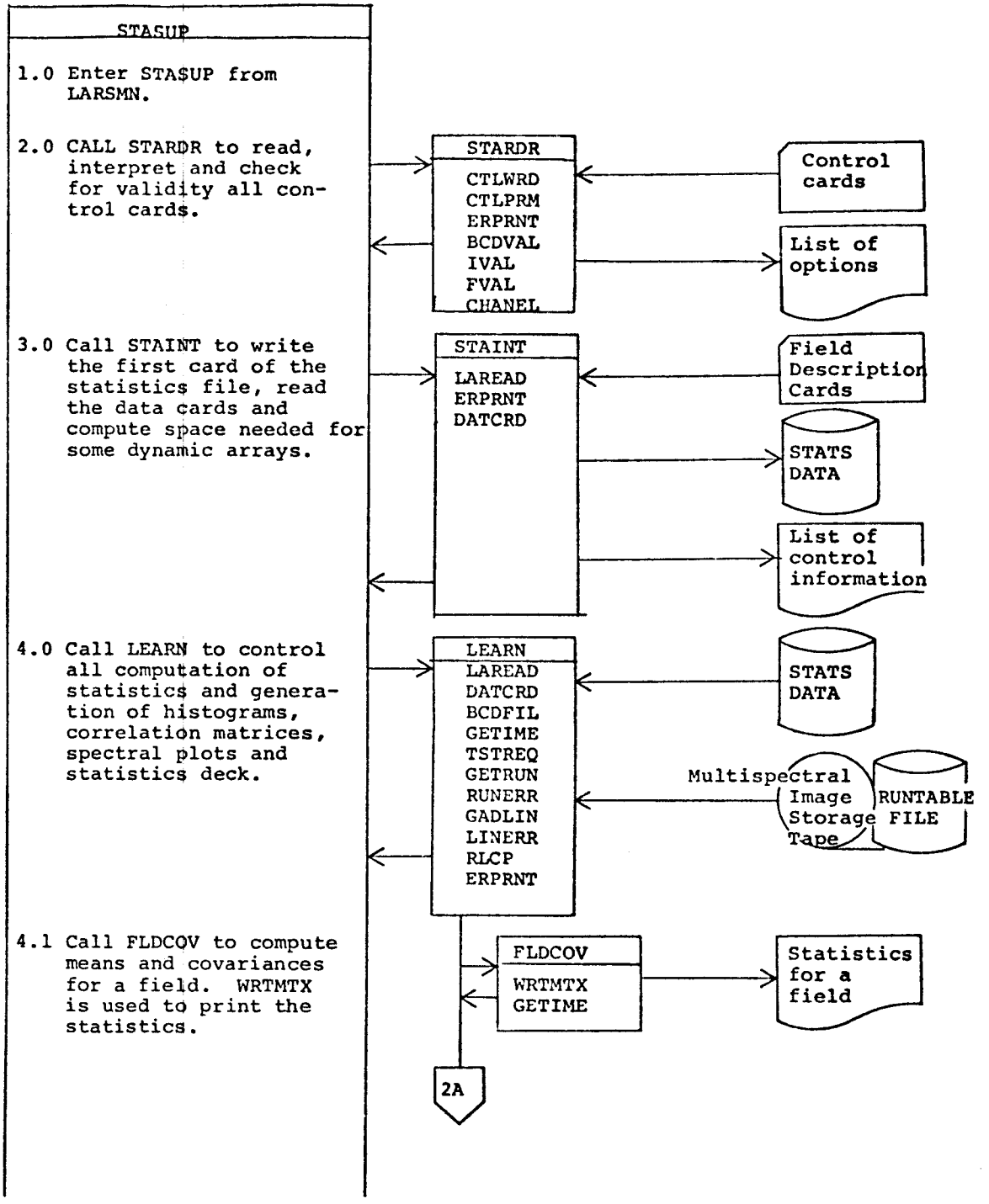
3.0 SMOINT reads Record Types 1 and 2 from the input results, calls GRPCHK to update or fill the group arrays, transfers information to the pool arrays and writes the new record types 1 and 2 to the output results file. The statistics file (record type 3) is then read, marked as invalid and copied to the output file. The requested area is located on the input file, data lines are read, and shifted if necessary. SMOOTH is called to modify the classified points and the final records are written to the output tape.

3.1 SMOOTH tallys the number of points in each class for each group (cell) of pixels and assigns the majority class to all pixels. Priority classes remain unchanged, and the data buffer is reloaded with the modified cells.



Load Module Name: STASUP

3.3 STASUP-1



3.3 STASUP-2

4.2 Call FLDSPC to print the spectral plot for a field (entry point in CLSSPC).

4.3 Call FLDHIS to print the histogram for a field. (Entry point in CLSHIS).

4.4 Call CLSCOV to compute mean and covariance for a class and print if requested. (Entry point in FLDCOV)

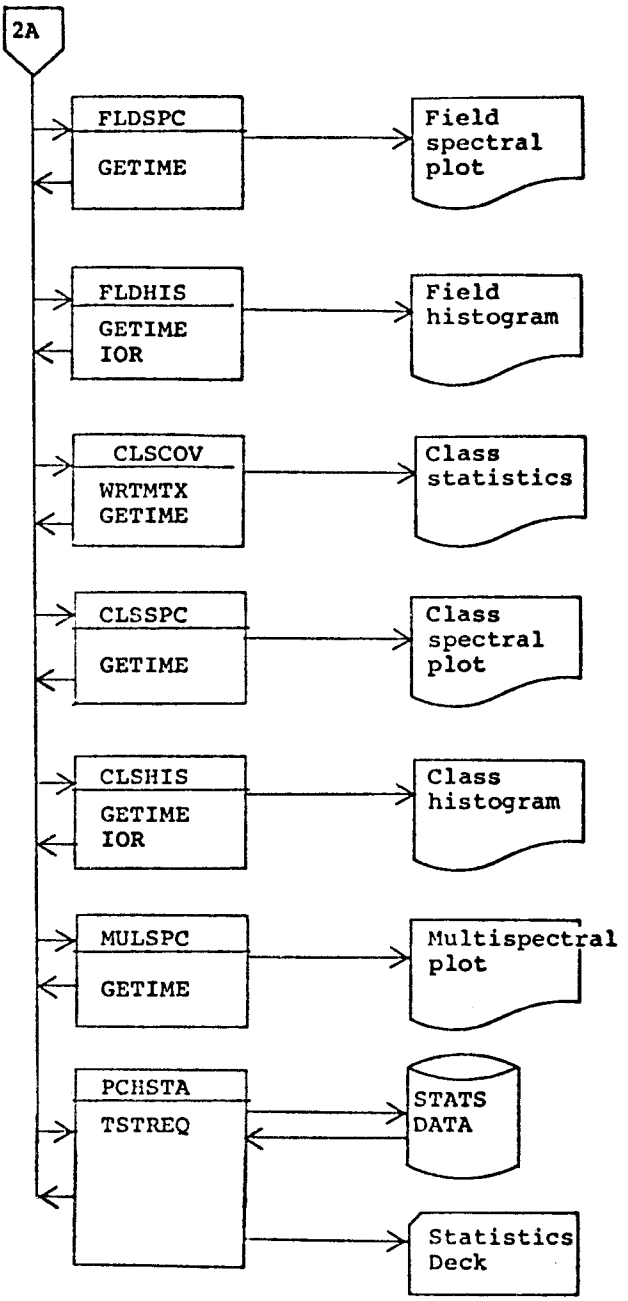
4.5 Call CLSSPC to print the spectral plot for a class.

4.6 Call CLSHIS to print the histogram for a class.

4.7 Call MULSPC to print one multispectral plot. (Entry in CLSSPC).

4.8 Call PCHSTA to write the statistics on the Statistics File and if a deck is requested, punch it.

5.0 Control is returned to LARSMN.



SECTION 4

LARSFRIS IMPLEMENTATION TECHNIQUES

SECTION 4

LARSFRIS IMPLEMENTATION TECHNIQUES

This section describes a number of implementation techniques that were used in the development of LARSFRIS Version 3 as well as discussions of some of the internal characteristics of the system that are of particular interest to the programmer or system analyst. The topics that are covered are:

4.1 COMMON Block Usage

4.2 Use of Object-Time Dimensions

4.3 Programming LARSFRIS Supervisors, Readers and Initiators

4.4 Generating Functional Load Modules

4.5 LARSFRIS Error Handling

4.6 Use of the LARSFRIS System for Test Runs

4.7 Attaching and Detaching Tape Drives

4.8 Implementation of the Control Card Checkout Feature

4.1 COMMON BLOCK USAGE

This section discusses the requirements for BLOCK DATA subroutines, describes how to change to a new version of a COMMON block, and gives a table of which program modules contain which COMMON blocks.

A BLOCK DATA subroutine must exist for each COMMON block that is used in LARSFRIS. Without the BLOCK DATA subroutine, there would be no CMS TEXT file for the COMMON block and thus no way to explicitly load the block. Even if no variables are initialized in the BLOCK DATA subroutine, it is necessary to explicitly load it in order to force the COMMON to load at the correct location (see Subsection 4.4 on Generating Functional Load Modules).

Procedure for Changing COMMON Blocks

An automatic procedure for changing to a new version of a COMMON block has been developed. The procedure requires the following:

Input Requirements

1. The number of lines in the old COMMON must be known. These number of lines will be deleted before inserting the new COMMON block. The count of lines starts with the COMMON statement and includes the specification statements, since they frequently change when the COMMON block is changed.

2. An EXEC file which lists the names of the FORTRAN programs containing the COMMON block. This file should have a first card of CONTROL OFF plus one card of the form shown below for each Fortran routine that is affected.

   &1 &2 name FORTRAN &3 &4

   where "name" is the name of the FORTRAN program.

3. Finally, the file containing the new COMMON block must be ready. This file should contain the COMMON statement and all specification statements. The filename, filetype of the file must be 'cname COMMON' where 'cname' is the name of the COMMON block.

Procedure Restrictions

The following restrictions apply to the procedure:

1. All FORTRAN files to be changed to include the new COMMON and the COMMON file must be on the A-disk.
2. The name of the COMMON block must be 6 characters long.

The Change Procedure

The procedure will replace the old COMMON with the new COMMON in each requested program that uses it.

To change the COMMON blocks use the following command:

list EXEC COMCHNGE cname lines

where:

list = the name of the EXEC file with the names of FORTRAN programs

cname = the name of the COMMON block

lines = the number of lines in the old COMMON. It is extremely important that lines be correct since this many lines will be deleted from the file before adding the new COMMON block.

The EXEC COMCHNGE executive routine executes an Assembler program called CHNCA which creates a new file containing the FORTRAN routine and the new COMMON. It does this by directly copying from the old source file to a new source file until the character string specified by "cname" is found. The new COMMON block is then read and copied into the new file. The counter that is used for reading the old file is then increased by the number of lines specified by "lines" in the command. This effectively skips the old COMMON, and the remainder of the old file is copied into the new one. COMCHNGE then erases the old file and gives the new one its name.

Four error messages can be produced by COMCHNGE. When they are produced, the FORTRAN routine being processed at the time of the error is left unchanged, and the entire procedure is terminated. The errors are:

1. cname COMMON does not exist

2. The FORTRAN file typed above does not exist.
3. An I/O error occurred reading or writing.
4. The FORTRAN program typed above does not contain the COMMON block.

4.2 USE OF OBJECT TIME DIMENSIONS (UTILIZATION OF ARRAY)

The LARSFRIS system provides each functional load module with storage space which it can access by use of the FORTRAN object-time dimension facility. The space is provided in the form of the array "ARRAY" in the COMMON block GLOCOM. GLOCOM is located at the end of the LARSMN(root) load module. ARRAY is specified as REAL*8, and the number of bytes it contains is given by the variable TOP (I*4) also in GLOCOM. Thus, the functional load module can determine the amount of space available in ARRAY. ARRAY is never used to pass information from one functional load module to another (The vector TEMPAS in GLOCOM is provided for this purpose).

There are three ways in which ARRAY can be used. The simplest ways are to use it directly or to "equivalence" other variables to it. Both of these methods have a limitation in that they fix the amount of space allocated to a given variable (e.g., if A is equivalenced to ARRAY(1) and B is equivalenced to ARRAY(25), then A can have no more than 24 doublewords). This can be a severe restriction when working with a variable number of classes, channels and samples in a line. Therefore, the most common usage of ARRAY is the third way; the use of object-time dimensions.

The use of object-time dimensions is achieved by passing arguments to subroutines which are, in fact, parts of the array "ARRAY". For example:

```
CALL SUB(ARRAY(I1),ARRAY(I2))
          :
SUBROUTINE SUB(COVAR,MEAN)
COMMON /EXAMPLE/ N1,N2
DIMENSION COVAR(N1),MEAN(N2)
```

In the example shown, the CALL statement will pass two arguments to subroutine SUB (in this case ARRAY(I1) and ARRAY(I2)) which are the starting locations in ARRAY for the storage of two arrays identified in SUB (in this case COVAR and MEAN). Since both I1 and I2 are variables, they are computed (prior to the CALL) according to the space requirements needed for the options selected in any given run. This is useful because there may not be enough space in array ARRAY if the user selects all the channels and maximum classes for instance, even though there is enough space for all channels and half the classes or vice-versa. Note that the functional programs should check to be certain that there is sufficient space in ARRAY for their requirements. In the subroutine being called (SUB) the arrays (in this case COVAR and MEAN) have variable dimensions (in this case N1 and N2 respectively). Any variable dimension must be an INTEGER*4 and non-subscripted. It may be defined in COMMON as shown above or in the calling list as shown here:

```
SUBROUTINE SUB(COVAR,MEAN,N1,N2)
DIMENSION COVAR(N1),MEAN(N2)
```

Note that the FORTRAN - G compiler will permit the program to simply use the dimensions COVAR(1) and MEAN(1) but this technique should not be used for two reasons. One is that it is very poor documentation of the program and the other is that the FORTRAN - G debug facility cannot be used unless variables are properly dimensioned.

4.3 PROGRAMMING LARSFRIS SUPERVISORS, READERS, AND INITIATORS

This subsection provides information which will aid a programmer in writing the supervisor, card reader or initiator of a processor. It contains common message structures and documentation of the usage of LARSFRIS support subroutines. First is a section on the general functions of supervisors, card readers and initiators, and some of the general programming features which may be applicable to any kind of subroutine. Then a section is provided with descriptions of the programming logic used in supervisors and card readers. Refer to the LARSFRIS Program Documentation Manual for specific subroutine uses. See the documentation of CTLWRD, BCDVAL, and ERPRNT since these subroutines are used extensively by card reading subroutines. Also see documentation for LAREAD which is used by initiators to read Field Description Cards.

General Functions of Supervisors, Card Readers and Initiators

The supervisor is the highest level subroutine of any load module. It receives control from LARSMN, executes a function, and returns control to LARSMN. The supervisor consists primarily of calls to other subroutines. In a load module with more than one LARSFRIS processing function, the supervisor may be the supervisor for all of the functions, or it may call a subroutine to control each function, and this subroutine is itself called a supervisor.

The card reader reads and interprets the function control cards. It usually performs checks for complete and accurate control information after the control cards have been read.

The function of an initiator is less well defined than that of a supervisor or card reader. In general, an initiator performs initialization operations for the processor other than control card reading. This may include reading data cards, reading stored information from a disk data set, computing base addresses of variable-dimension arrays, and printing header information.

The naming convention is a hierarchical one. That is, if initiator functions are in the card reader, the subroutine is called a card reader, and if the card reading is done in the supervisor, the subroutine is called a supervisor. Sometimes these functions are combined into one subroutine to avoid having a large number of very short subroutines.

Supervisors

All supervisors will have the following organization:

1. Type a message of the form

   Innnn functionname FUNCTION REQUESTED (xxxSUP)

   'functionname' is the name of the processor without the asterisk (for example STATISTICS).
2. Read the control cards either in the supervisor or by calling a card reading subroutine.
3. Call an initiator if this processor uses one.

4. Perform the requested function either in the supervisor or by calls to major processing subroutines.
5. Type a message of the form

   Innnn functionname FUNCTION COMPLETED (xxxSUP)

6. Return to LARSMN

The following pages show an example of a supervisor (the supervisor for the STATISTICS FUNCTION.)

```
FILE. . .     STASUP   FORTRAN  B1
C         STASUP    LARS   320                                          STA00010
C                                                                       STA00020
C*****************************************************************STACC030
C                                                                       STA00040
C STASUP        SUPERVISOR FOR STATISTICS FUNCTION                      STA00050
C               REVISED 11/28/72 BY EARL RODD                           STA00060
C                                                                       STACC070
C*****************************************************************STAC0080
C                                                                       STA00090
      SUBROUTINE STASUP                                                 STA00100
      IMPLICIT INTEGER * 4 (A-Z)                                        STACC110
C                                                                       STA00120
      COMMON /GLOCOM/ BLANK, CARD(20), CHKOUT, COPFIL, CLASSR, CLASSX,  STA00130
     1    CLUSTA, CUNPUT, CPYOUT, CRDMCR, CRDSEQ, DATAPE,               STA00140
     2    DUPLIP, DUPRUN, ERRMSG, FBPNT,                                STACC150
     3    FILESV, FLDBND, HDATA, HEAD(88), ID(200), IMAGEX,             STA00160
     4    IMARK, KEYBD, MAPTAP, MAXCHA, MAXCLS,                         STA00170
     5    PAGSIZ, PNCH, POINT, PRESUX, PRNTR, READIN,                   STA00180
     6    RESTRT, RUNFIL, RUNTAB(10,3),                                 STA00190
     7    SDATA, SEPARX, SEPTPX, SPARE(10), TEMPAS(30),                 STA00200
     8    TPSTAT(6), TTELDX ,TYPEWR,                                    STA00210
     9    TOP, ARRAY(12500)                                             STA00220
C                                                                       STACC230
      REAL * 8 ARRAY                                                    STA00240
      REAL * 4 FRQCAL(5,30)                                             STA00250
      INTEGER * 4 COMENT(16), DATE(5), HED1(16), HED2(16), TIME(5)      STA00260
      INTEGER * 2 BLANK2                                                STACC270
      LOGICAL * 4 CHKOUT                                                STA00280
      LOGICAL * 1 BLANK1                                                STA00290
      EQUIVALENCE (DATSAV,ID(1)), (CURRUN,ID(3)), (FRQCAL(1),ID(51)),   STA00300
     1          (HED1(1),HEAD(8)), (DATE(1),HEAD(26)), (HED2(1),HEAD(39)),STACC310
     2          (TIME(1),HEAD(58)), (COMENT(1),HEAD(72)),               STA00320
     3          (MAPSAV,TPSTAT(1)),                                     STA00330
     4          (SEPSCR,TPSTAT(2)), (DUPIN,TPSTAT(3)), (DASIAT,TPSTAT(4)),STA00340
     5          (COPSER,TPSTAT(5)), (TRAOUT,TPSTAT(6)),                 STACC350
     6          (BLANK,BLANK2,BLANK1)                                   STA00360
C                                                                       STA00370
C*****************************************************************STA00380
C                                                                       STACC390
C                        VARIABLES USED IN STASUP                       STA00400
C                                                                       STA00410
C AVAR1  = BASE ADDRESS OF ARRAY AVAR                                   STA00420
C CLMEN1 = BASE ADDRESS OF THE ARRAY CLSMEN                             STA00430
C CLSID1 = BASE ADDRESS OF THE ARRAY CLSID                              STA00440
C CLVAR1 = BASE ADDRESS OF THE ARRAY CLSVAR                             STA00450
C CORE   = NUMBER OF BYTES OF ARRAY LEFT TO BE USED                     STA00460
C COVAR1 = BASE ADDRESS OF ARRAY COVAR                                  STA00470
C DATBAS = BASE ADDRESS OF THE IDATA ARRAY                              STA00480
C FLMEN1 = BASE ADDRESS OF THE ARRAY FLCMEN                             STA00490
C FLVAR1 = BASE ADDRESS OF THE ARRAY FLOVAR                             STA00500
C HCTAL1 = BASE ADDRESS OF THE ARRAY HCTALY                             STA00510
C HFTAL1 = BASE ADDRESS OF THE ARRAY HFTALY                             STA00520
C SPEC1  = BASE ADDRESS FOR ARRAY SPEC                                  STA00530
C*****************************************************************STA00540
C*****************************************************************STA00550
C                                                                       STA00560
C READ SUPERVISOR CONTROL CARDS                                         STA00570
C                                                                       STA00580
C*****************************************************************STA00590
      CALL TSTREC(CORE)                                                 STA00600
      WRITE (TYPEWR,9100)                                               STA00610
 9100 FORMAT (' I0198  STATISTICS FUNCTION REQUESTED  (STASUP)')        STA00620
      CALL STARDR(ARRAY(501))                                           STACC630
C*****************************************************************STA00640
C                                                                       STA00650
C SET UP DYNAMIC CORE ALLOCATION                                        STA00660
C                                                                       STACC670
C*****************************************************************STA00680
      CALL STAINT(ARRAY(501),SPEC1,COVAR1,AVAR1,CLSID1,FLMEN1,FLVAR1,   STA00690
     *            CLMEN1,CLVAR1,HFTAL1,HCTAL1,DATBAS)                   STA00700
      CORE = TOP-DATBAS*8                                               STACC710
      IF (CORE.GT.0) GO TO 150                                          STA00720
      CORE = -CORE                                                      STA00730
      CALL ERRPNT (250,'GO')                                            STA00740
      WRITE (PRNTR,9140) CORE                                           STAC0750
      WRITE (TYPEWR,9140) CORE                                          STA00760
 9140 FORMAT (10X,I7,' BYTES.')                                         STA00770
      CALL RTMAIN                                                       STA00780
```

```
FILE. . .     STASUP   FORTRAN  81

  150 IF (CHKOUT) GO TO 200                                                    STA00790
C*****************************************************************************STA00800
C                                                                              STA00810
C CALCULATE STATISTICS                                                         STA00820
C                                                                              STA00830
C*****************************************************************************STA00840
      CALL LEARN(ARRAY(SPEC1),ARRAY(COVAR1),ARRAY(AVAR1),ARRAY(CLSID1),        STA00850
     *           ARRAY(FLMEN1),ARRAY(FLVAR1),ARRAY(CLMEN1),ARRAY(CLVAR1)      STA00860
     *          ,ARRAY(HFTAL1),ARRAY(HCTAL1),ARRAY(DATBAS),                   STA00870
     *           DATBAS,ARRAY(1))                                             STA00880
  200 WRITE (ITYPEWR,9150)                                                     STA00890
 9150 FORMAT (' I0199  STATISTICS FUNCTION COMPLETED     (STASUP)')           STA00900
      RETURN                                                                   STA00910
      END                                                                      STA00920
```

Card Readers

Control card readers consist of three sections:

- The first section initializes flags and switches to default values, initializes arrays which have default values, and calls TSTREQ to clear the STOP/SUSPEND flag. This flag must be cleared before any calls are made to TSTREQ to check the flag.

- The second section is a loop which reads and interprets the control cards. Control passes out of the loop when a DATA or END card is read. This loop uses CTLWRD to read the card and determine the keyword on the card. Then a branch is made to code that completes the interpretation of that kind of control card.

- The third section checks for complete and consistent control information. In some cases, the user is asked to type in additional control cards, which are necessary for proper operation of the function.

The following section describes control card reader programming by illustrating the PRINTRESULTS card reader with an explanation of the logic used. Following the explanation is a listing of the complete subroutine. The subroutine shown is PRIRDR.

```
C                                                                       PRI01080
C       INITIALIZE                                                      PRI01090
C                                                                       PRI01100
      CALL TSTREQ(I)                                                    PRI01110
      RESTRT = C                                                        PRI01120
      DC 101 I=1,60                                                     PRI01130
  101 GRPSTK(I)=0                                                       PRI01140
      STOPFG = C                                                        PRI01150
      RESULT = MAPTAP                                                   PRI01160
      SERIAL = 0                                                        PRI01170
      THSCNT=0                                                          PRI01180
      RUNNUM=0                                                          PRI01190
      CALC=0                                                            PRI01200
      NCPAPS=1                                                          PRI01210
      CCPIES=1                                                          PRI01220
      STATKY=0                                                          PRI01230
      LISTKY=1                                                          PRI01240
      OTRKY=0                                                           PRI01250
      OTSKY=0                                                           PRI01260
      TRCLS=0                                                           PRI01270
      TRFLC=0                                                           PRI01280
      TSFLC=0                                                           PRI01290
      TSCLS=0                                                           PRI01300
      PCT=0                                                             PRI01310
      USE=0                                                             PRI01320
      TYPE=2                                                            PRI01330
      SYMCNT=0                                                          PRI01340
      BLKKY=0                                                           PRI01350
      BLOCK(1)=1                                                        PRI01360
      BLOCK(2)=30000                                                    PRI01370
      BLOCK(3)=1                                                        PRI01380
      BLOCK(4)=1                                                        PRI01390
      BLOCK(5)=30000                                                    PRI01400
      BLOCK(6)=1                                                        PRI01410
      DC 122 I=7,28                                                     PRI01420
  122 BLOCK(I)=C                                                        PRI01430
      LEAD=-100                                                         PRI01440
      STKPTR=0                                                          PRI01450
      NCFETS=0                                                          PRI01460
      NOCLSS=0                                                          PRI01470
      NOGRPS=0                                                          PRI01480
      RQTAPE = 0                                                        PRI01490
      RQFILE = 0                                                        PRI01500
      ERRCOR = C                                                        PRI01510
```

This is the first section of the card reader. The statement setting ERRCOR to 0 is required before the first call to CTLWRD is made. This tells CTLWRD that it is to read the card from the control card input device (rather than read a corrected card from the typewriter or an additional card).

```
      IF(CONPUT.EQ.2) WRITE(TYPEWR,1102)                                PRI01520
1102  FORMAT (' 10072 TYPE PRINTRESULTS CONTROL CARDS     (PRIRDR)')    PRI01530
```

If control cards are to be read from the typewriter, an informational message with the above format must be typed.

```
      ASSIGN 210 TO GO                                                  PRI01590
200   GO TO GO,(210,651,656)                                            PRI01600
```

This ASSIGN will stay in effect until section three of the card reader is executed. This statement starts the second section of the card reader. The GOTO will normally branch to statement 210 unless a branch has been made from section three into section two. In that case, the GOTO is used to transfer control back to section three.

```
210   CALL CTLWRD(CARD,COL,SUPWRD,8,CODE,READIN,ERRCOR)                 PRI01610
      IF (CODE .EQ. 8) SPARE(2) = 1                                     PRI01620
      IF (ERRCOR .EQ. 4) CALL RTMAIN                                    PRI01630
      IF (ERRCOR .EQ. 2) CALL ERPRNT (139,'STOP')                       PRI01640
      GO TO (300,420,450,500,550,600,650,650), CODE                     PRI01650
```

This call to CTLWRD reads the next control card and determines the keyword on the card. The vector SUPWRD is a list of possible keywords for the Printresults processor. If ERRCOR is returned from CTLWRD with a value of 2, EOF was read on the input data, so ERPRNT is called to write the appropriate message. If ERRCOR is returned from CTLWRD with a value of 4, 'KILL' was read and

the function should be terminated. The computed GOTO is used to branch to code complete interpretation of this particular control card. (CODE indicates which keyword is on the card.)

```
  220 WRITE (TYPEWR,9220) CARD                                    PRI01660
 9220 FORMAT (5X,20A4)                                            PRI01670
      CALL ERPRNT (ERRNUM,'GO')                                   PRI01680
  230 IF (ERRCUR .NE. 3) ERRCOR = 1                               PRI01690
      GO TO 210                                                   PRI01700
```

These five statements form an error handler. Statement 220 is branched to when an error is detected while interpreting a control card. Before branching to statement 220, the variable ERRNUM is set to the appropriate error number. The error handler types the erroneous card (note the FORMAT statement since this same format should be used when writing a card image to the printer or typewriter), calls ERPRNT to write the error message, then sets ERRCOR = 1 to inform CTLWRD that it is to read a corrected control card from the typewriter. If ERRCOR = 3, this means that control has come from section three (i.e., an additional input is necessary), and ERRCOR must remain 3 to retain that level of error correction.

The code from statements 300 to 650 is used to interpret the various control cards. The details on only two kinds of cards will be illustrated to show the general logic used in control card interpretation.

```
  420 IF (COL .EQ. 72) GO TO 200                                    PRI02690
      CALL CTLPRM (CARD,COL,RESCOD,3,CODE,&230)                     PRI02700
      GO TO (425,430,435), CODE                                     PRI02710
C                                                                   PRI02720
C         TAPE SPECIFICATION                                        PRI02730
C                                                                   PRI02740
  425 VECSZ = 1                                                     PRI02750
      CALL IVAL (CARD,COL,ROTAPE,VECSZ,&440)                        PRI02760
      IF (VECSZ .EQ. 0) GO TO 440                                   PRI02770
      GO TO 420                                                     PRI02780
C                                                                   PRI02790
C         FILE SPECIFICATION                                        PRI02800
  430 VECSZ = 1                                                     PRI02810
      CALL IVAL (CARD,COL,ROFILE,VECSZ,&445)                        PRI02820
      IF (VECSZ .EQ. 0) GO TO 445                                   PRI02830
      GO TO 420                                                     PRI02840
C                                                                   PRI02850
C         DISK SPECFICATION                                         PRI02860
C                                                                   PRI02870
  435 RESULT = CLASSR                                               PRI02880
      GO TO 420                                                     PRI02890
C                                                                   PRI02900
C         ERROR IN TAPE SPEC.                                       PRI02910
C                                                                   PRI02920
  440 ERRNUM = 454                                                  PRI02930
      GO TO 220                                                     PRI02940
C                                                                   PRI02950
C         ERROR IN FILE SPEC                                        PRI02960
C                                                                   PRI02970
  445 ERRNUM = 455                                                  PRI02980
      GO TO 220                                                     PRI02990
```

The code illustrated above is that used to interpret the RESULTS control card. Since the RESULTS card can have more than one control parameter, the illustration shows a loop going back to statement 420 after each control parameter is processed. Statement 420 asks if there is another control parameter on the card; if not, control returns to statement 200 to read the next card.

The call to CTLPRM has a non-standard return specified. This return will be executed if CTLPRM cannot recognize the control parameter or discovers a syntax error on the card. In either case, CTLPRM will have typed the erroneous card with the error message requesting a corrected card be entered. The non-standard return to statement 230 will set ERRCOR = 1 (to advise CTLWRD that it is reading a correction from the typewriter) and branch to call CTLWRD.

When IVAL makes a non-standard return, this means there is a syntax error in the specification. The non-standard return branches to set ERRNUM to the appropriate error numeric and then branches to the internal error handler at statement 220.

```
C*************************************************************************PRI03000
C                                                                         PRIC3010
C GROUP CARD                                                              PRI03020
C                                                                         PRI03030
C*************************************************************************PRI03040
  450 CALL GRPSCN(GRPNAM,GRPSTK,NOGRPS,COL,CARD,&460)                     PRIC3050
      GO TO 200                                                           PRI03060
  460 STOPFG = 1                                                          PRI03070
      STKPTR = 0                                                          PRI03080
      NOGRPS = 0                                                          PRIC3090
      CALL ERPRNT (446,'GOTO',&200)                                       PRI03100
```

This code interprets the GROUP card by a call to GRPSCN. The non-standard return indicates syntax error on the card. This is an error which cannot be recovered from by entering a corrected card; however, the program should continue to execute until all the control cards have been read. This is accomplished by setting a flag (STOPFG = 1.) STOPFG was initialized to 0. After all the cards have been read, STOPFG will be checked, and if it is 1, execution will terminate.

```
C*************************************************************************PRI04170
C                                                                         PRI04180
C END CARD                                                                PRI04190
C                                                                         PRI04200
C*************************************************************************PRIC4210
C                                                                         PRIC4220
C CHECK FOR CORRECT CONTROL CARD INPUTS                                   PRI04230
C                                                                         PRI04240
  650 ASSIGN 651 TO GO                                                    PRI04250
C                                                                         PRIC4260
C         CHECK FOR COMPLETE INPUT SPECS. I.E EITHER DISK OR              PRI04270
C         IF TAPE HAD TO GIVE  A FILE NUMBER. (TAPE NO. OF ZERO IS        PRI04280
C         OK SINCE THIS MAY MEAN USE SCRATCH TAPE FROM CLASSIFICATION)    PRI04290
C                                                                         PRIC4300
  651 IF (RESULT .EQ. CLASSR .OR. RQFILE .GT. 0) GO TO 655                PRI04310
      ERRNUM = 456                                                        PRI04320
      GO TO 800                                                           PRIC4330
```

Statement 650 begins section three of the card reader. Section three is reached after all control cards have been read in. It is used to check for valid control information given by the user and will give him the opportunity to type in an additional control card if needed (results card in this case). Statement 651 determines whether or not a file number was specified if classification results are to be recorded on tape. When statement 651 does discover an error (missing file number), ERRNUM is set to the appropriate error number and control passes to statement 800 (see code at 800 below) and then to statement 210 in section one to read in the additional results card. After the card is read, a branch will be made into the code in section two to interpret the revised results card. After the card has been interpreted, control will pass to statement 200.

At this point, the purpose of the ASSIGN in statement 650 becomes clear: the program should not go to section one to read in another control card but should branch back into section 3 (statement 651). The code at statement 800 (shown on the next page) must be used in any card reader which reads in an additional control card after discovering missing data.

```
660 IF(NOGRPS.EQ.0)GO TO 850                                          PRIC4410
    DO 664 I=1,60                                                     PRI04420
    IF(GRPSTK(I) .GT. NOGRPS) CALL ERPRNT (458,'STOP')                PRI04430
664 CONTINUE                                                          PRI04440
    GO TO 850                                                         PRIC4450
```

This code illustrates a check for valid control information which does not result in a request for an additional control card. Statement 660 will skip the loop making the check on class grouping if no grouping was requested. The DO 664 loop checks for consecutive group numbers, and if an error is found, ERPRNT is called to write the error message and terminate execution. The branch to statement 850 means all control information checking has been completed. The code from statement 850 through the end of PRIRDR performs further initialization for the PRINTRESULTS function. It is peculiar to PRINTRESULTS and is not general programming which will normally be a part of a card reader.

```
800 ERRCOR = 3                                                        PRIC4490
    CALL ERPRNT (ERRNUM,'GOTO',&210)                                  PRIC4500
```

This code sets ERRCOR = 3 to use CTLWRD to read and interpret an additional control card. When ERRCOR = 1, CTLWRD will go to the input stream for the next card if a Carriage Return is the response to a request for a card. But when ERRCOR = 3, CTLWRD will respond to the Carriage Return with an error message stating that the additional card must be typed and will then read from

the typewriter again. Note that when ERRCOR = 3 on entry to CTLWRD, it remains 3. The call to ERPRNT writes the error message requesting additional control information and branches to the call to CTLWRD in section two.

The complete listing of the PRINTRESULTS card reader begins on the second page following this page.

Initiators

Initiators have very little in common. Normally, however, all card reading (both control and data cards) will be completed in the initiator. When all cards have been read, the following message should be typed:

I0034 ALL CONTROL AND DATA CARDS HAVE BEEN READ

Reading Data Cards

Data cards should be read using the same techniques as control cards. For reading data decks containing field description cards, use subroutine LAREAD which will read the card and interpret it as either a free form or fixed form of field description card. LAREAD will also identify TEST, CLAS, DATA and END cards. See the module description for more detail on the usage of LAREAD.

Card Reading Considerations

When reading a user input deck, use the 'END=' exit in the FORTRAN READ statement to point to a call to ERPRNT with error message 139. This indicates that end of file was reached in the middle of an input deck.

A READ to the typewriter will produce an EOF condition if the user enters a Carriage Return with nothing else on the line. Therefore, when the user enters only a Carriage Return, the 'END=' exit will be taken from the READ statement.

After a READ to the typewriter, a check must be made for the characters 'KILL' as the first four characters of the typewriter input. If KILL is entered, RTMAIN should be called to terminate execution of the function. This check supports the batch monitor.

When reading input decks, every card must be checked for an END card. When an END card is found, the appropriate cell in GLOCOM must be set = 1. (This is currently SPARE(2) though it is subject to change so check the variable definitions in GLOCOM). This is necessary because when LARSMN receives control after the completion of a function, it flushes any unused cards for that function unless the END card has already been read. It uses this flag to determine if the END card has been read.

```
FILE. . .     PRIRDR   FORTRAN  B1
C         PRIRDR     LARS   248                                                PRI00010
C                                                                               PRI00020
C*****************************************************************************PRICC030
C                                                                               PRI00040
C PRIRDR        FUNCTION CONTROL CARD READER FOR PRINTRESULTS                   PRI00050
C               REVISED 9/22/72 BY EARL RODD                                    PRI00060
C                                                                               PRICC070
C*****************************************************************************PRI00080
C                                                                               PRI00090
      SUBROUTINE PRIRDR                                                         PRI001C0
      IMPLICIT INTEGER * 4 (A-Z)                                                PRICC110
C*****************************************************************************PRI00120
C                                                                               PRI00130
C DEFINE PROGRAM VARIABLES                                                      PRICC140
C                                                                               PRI00150
C*****************************************************************************PRI00160
      COMMON /GLOCOM/ BLANK, CARD(20), CHKOUT, COPFIL, CLASSR, CLASSX,          PRI00170
     1    CLLSTX, CONPUT, CPYOUT, CRDRDR, CRDSEQ, DATAPE,                       PRICC180
     2    DUPLTP, DUPRUN, ERRMSG, FBPNT,                                        PRI00190
     3    FILESW, FLDBND, HDATA, HEAD(68), ID(200), IMAGEX,                     PRI00200
     4    IMARK, KEYBD, MAPTAP, MAXCHA, MAXCLS,                                 PRI00210
     5    PAGSIZ, PNCH, POINT, PRESUX, PRNTR, READIN,                           PRICC220
     6    RESTRT, RUNFIL, RUNTAB(10,3),                                         PRI00230
     7    SDATA, SEPARX, SEPTPX, SPARE(10), TEMPAS(30),                         PRI00240
     8    TPSTAT(6), TTFLDX ,TYPEWR,                                            PRI00250
     9    TOP, ARRAY(12500)                                                     PRICC260
C                                                                               PRICC270
      REAL * 8 ARRAY                                                            PRI00280
      REAL * 4 FROCAL(5,30)                                                     PRICC290
      INTEGER * 4 COMENT(16), DATE(5), HED1(16), HED2(16), TIME(5)              PRI00300
      INTEGER * 2 BLANK2                                                        PRI00310
      LOGICAL * 4 CHKOUT                                                        PRI00320
      LOGICAL * 1 BLANK1                                                        PRICC330
      EQUIVALENCE (DATSAV,ID(1)), (CURRUN,ID(3)), (FROCAL(1),ID(51)),           PRI00340
     1          (HED1(1),HEAD(8)), (DATE(1),HEAD(26)), (HED2(1),HEAD(39)),     PRI00350
     2          (TIME(1),HEAD(58)), (COMENT(1),HEAD(72)),                      PRI00360
     3          (MAPSAV,TPSTAT(1)),                                            PRICC370
     4          (SEPSCR,TPSTAT(2)), (DUPIN,TPSTAT(3)), (DASTAT,TPSTAT(4)),     PRI00380
     5          (COPSER,TPSTAT(5)), (TRADUT,TPSTAT(6)),                        PRI00390
     6          (BLANK,BLANK2,BLANK1)                                          PRI004C0
C                                                                               PRI00410
      COMMON /PRICOM/ CLSNAM(60), GRPNAM(61),                                   PRI00420
     1          CALC, CLSDAT(5), CSET(3,30), COPIES, INFO(17),                 PRI00430
     2          ITHRES(60), LINE, LISTKY, NOCLS, NOCLS5, NOFET5,               PRI00440
     3          NOFLD, NOGRPS, NOMAPS, NOTST, OTRKY, OTSKY,                    PRICC450
     4          PCT, RESULT, RUNNUM, SAVLIN, SERIAL, STATKY,                   PRI00460
     5          STCPCD, SYMCNT, THSCNT, THRES(60), TRCLS,                      PRI00470
     6          TRFLD, TSCLS, TSFLD,                                           PRI00480
     7          BLOCK(28), CSEL(30), FETVEC(30), GRPSTK(60),                   PRI00490
     8          HWCRK(128), NUMRAL(10), SYMMTX(64)                             PRI00500
C                                                                               PRI00510
      REAL * 8 CLSNAM, DWORK1(14), DWORK2(15), GRPNAM                           PRI00520
      REAL * 4 CSET, RWORK(30), THRES                                           PRICC530
      INTEGER * 4 IWORK(30)                                                     PRICC540
      LOGICAL * 4 SAVLIN                                                        PRI00550
      INTEGER * 2 BLOCK, CALTRN, CALTST, CSEL, DILIN, DISAM,                    PRI00560
     1          DLINT, DLLIN, DLSAM, DSINT, FETVEC, GRPSTK,                    PRICC570
     2          HWCRK(60), IPT, LAG, LEAD, LPT, NDW, NPD, PINT,                PRI00580
     3          PISAM, PLSAM, PPINT, PSINT, PTS                                PRI00590
      LOGICAL * 1 LWORK, NUMRAL, SYMMTX                                         PRI00600
C                                                                               PRICC610
      EQUIVALENCE (INFO(1),CRUN), (INFO(4),CILIN), (INFO(5),CLLIN),             PRI00620
     1          (INFO(6),CLINT), (INFO(7),CISAM), (INFO(8),CLSAM),             PRI00630
     2          (INFO(9),CSINT),                                               PRI00640
     3          (BLOCK(7),IPT), (BLOCK(8),LPT), (BLOCK(9),PINT),               PRICC650
     4          (BLOCK(10),PISAM), (BLOCK(11),PLSAM), (BLOCK(12),PSINT),       PRI00660
     5          (BLOCK(13),LEAD), (BLOCK(14),LAG), (BLOCK(15),NPD),            PRI00670
     6          (BLOCK(16),NDW), (BLOCK(17),PPINT), (BLOCK(18),PTS),           PRI00680
     7          (BLOCK(19),DILIN), (BLOCK(20),DLLIN), (BLOCK(21),CLINT),       PRICC690
     8          (BLOCK(22),DISAM), (BLOCK(23),PLSAM), (BLOCK(24),DSINT),       PRI00700
     9          (BLOCK(25),CALTRN), (BLOCK(26),CALTST),                        PRI00710
     A          (LWORK(1),DWORK1(1),DWORK2(1),RWORK(1),HWORK(1),               PRI00720
     B          HWCRK(1))                                                      PRI00730
C                                                                               PRI00740
      REAL*4          RWORK1(60)                                                PRI00750
      INTEGER * 4 BLKCOD(4), DUM(10), OTLCOD(6), PRICOD(7),                     PRI00760
     1          RESCOD(3), SUPWRD(8), WORK(60)                                 PRICC770
      EQUIVALENCE (WORK(1), RWORK1(1), LWCRK(1))                                PRI00780
```

```
FILE. . .     PRIRDR   FORTRAN  81

      LOGICAL*1       LWORK1(240)                                          PR1CC790
      DATA IMSICP /'KILL'/                                                 PR100800
      DATA OTLCOD /'R', 'A', 'E', 'S', ',', ')' /                          PR100810
      DATA PRICOD / 'STAT', 'MAPS', 'OUTL', 'NOLI', 'TRAI',                PR10C820
     1     'TEST', 'TABL' /                                                PR1CC830
      DATA BLKCOD / 'RUN', 'CALC', 'LINE', 'COL' /                         PR100840
      DATA RESCOD / 'TAPE', 'FILE', 'DISK' /                               PR100850
      DATA SUPWRD / 'PRIN', 'RESL', 'GROU', 'SYMB', 'THRE',                PR100860
     1     'ELOC', 'DATA', 'END ' /                                        PR1CC870
      DATA          FBCD/'F'/,CBCD/'C'/,PBCD/'P'/                          PR100880
C*********************************************************************** PR100890
C                                                                          PR100900
C                    VARIABLES USED IN PRIRDR                              PR1CC910
C                                                                          PR1CC920
C ANSWER = BCD ANSWER--YES NO                                              PR100930
C BLANK  = 4 BLANKS (FULL WORD)                                            PR100940
C BLKCOD = ARRAY CONTAINING BLOCK CONTROL PARAMETERS                       PR1CC950
C CARD   = 20 FULL WORD ARRAY FOR READING CONTROL CARDS                    PR1C0960
C CHECK  = FIRST WORD READ FROM MAPTAP                                     PR100970
C CODE   = POINTER TO PARAMETER FOUND IN CARD                              PR100980
C COL    = COLUMN POINTER IN CARD                                          PR1CC990
C IEND   = FLAG FOR END OF DISPLAY PROCESSING                              PR101000
C NFILE  = FILE NUMBER READ FROM THE MOUNTED MAPTAP                        PR101010
C NWTP   = TAPE NUMBER READ FROM THE MOUNTED MAPTAP                        PR101020
C PRICOD = VECTOR CONTAINING PRINT CONTROL PARMS.                          PR1C1030
C SUPWRD = ARRAY CONTAINING FUNCTION CONTROL PARAMETERS                    PR101040
C RQTAPE = NUMBER OF MAPTAP TO BE MOUNTED                                  PR101050
C                                                                          PR101060
C*********************************************************************** PR101070
C                                                                          PR101080
C          INITIALIZE                                                      PR101090
C                                                                          PR101100
      CALL TSTREQ(I)                                                       PR101110
      RESTRT = C                                                           PR101120
      DO 101 I=1,60                                                        PR101130
  101 GRPSTK(I)=0                                                          PR101140
      STCPFG = 0                                                           PR1C1150
      RESULT = MAPTAP                                                      PR101160
      SERIAL = 0                                                           PR101170
      THSCNT=0                                                             PR101180
      RUNNUM=0                                                             PR10119C
      CALC=0                                                               PR101200
      NOMAPS=1                                                             PR101210
      COPIES=1                                                             PR10122C
      STATKY=0                                                             PR101230
      LISTKY=1                                                             PR101240
      OTRKY=0                                                              PR101250
      OTSKY=0                                                              PR101260
      TRCLS=0                                                              PR101270
      TRFLC=0                                                              PR101280
      TSFLC=0                                                              PR101290
      TSCLS=0                                                              PR101300
      PCT=0                                                                PR101310
      USE=0                                                                PR101320
      TYPE=2                                                               PR101330
      SYMCNT=0                                                             PR101340
      BLKKY=0                                                              PR101350
      BLCCK(1)=1                                                           PR101360
      BLOCK(2)=30000                                                       PR101370
      BLCCK(3)=1                                                           PR101380
      BLCCK(4)=1                                                           PR101390
      BLCCK(5)=30000                                                       PR101400
      BLOCK(6)=1                                                           PR101410
      DO 122 I=7,28                                                        PR101420
  122 BLCCK(I)=0                                                           PR101430
      LEAD=-10C                                                            PR101440
      STKPTR=0                                                             PR101450
      NOFETS=0                                                             PR101460
      NCCLS5=0                                                             PR101470
      NOGRPS=0                                                             PR101480
      RQTAPE = C                                                           PR101490
      RQFILE = C                                                           PR101500
      ERRCOR = 0                                                           PR101510
      IF(CONPUT.EQ.2) WRITE(TYPEWR,1102)                                   PR101520
 1102 FORMAT (' [0072 TYPE PRINTRESULTS CONTROL CARDS    (PRIRDR)')       PR101530
C*********************************************************************** PR101540
C                                                                          PR101550
C DECODE CONTROL CARDS                                                     PR1C1560
```

```
FILE. . .    PRIRDR   FORTRAN  B1
C                                                                       PRI01570
C***********************************************************************PRI01580
      ASSIGN 210 TO GO                                                  PRI01590
  200 GO TO GO,(210,651,656)                                            PRI01600
  210 CALL CTLWRD(CARD,COL,SUPWRD,8,CODE,READIN,ERRCOR)                 PRI01610
      IF (CODE .EQ. 8) SPARE(2) = 1                                     PRI01620
      IF (ERRCOR .EQ. 4) CALL RTMAIN                                    PRI01630
      IF (ERRCOR .EQ. 2) CALL ERPRNT (139,'STOP')                       PRI01640
      GO TO (300,420,450,500,550,600,650,650), CODE                     PRI01650
  220 WRITE (TYPEWR,9220) CARD                                          PRI01660
 9220 FORMAT (5X,20A4)                                                  PRI01670
      CALL ERPRNT (ERRNUM,'GO')                                         PRI01680
  230 IF (ERRCOR .NE. 3) ERRCOR = 1                                     PRI01690
      GO TO 210                                                         PRI01700
C***********************************************************************PRI01710
C                                                                       PRI01720
C PRINT CARD                                                            PRI01730
C                                                                       PRI01740
C***********************************************************************PRI01750
  300 IF(CCL.GE.72) GO TO 200                                           PRI01760
      CALL CTLPRM(CARD,CCL,PRICOD,7,CODE,&230)                          PRI01770
      GO TO (301,302,303,325,402,405,411),CODE                          PRI01780
C                                                                       PRI01790
C STATS                                                                 PRI01800
C                                                                       PRI01810
  301 STATKY=1                                                          PRI01820
      GO TO 300                                                         PRI01830
C                                                                       PRI01840
C MAPS                                                                  PRI01850
C                                                                       PRI01860
  302 I=1                                                               PRI01870
      CALL IVAL(CARD,COL,NOMAPS,I,&341)                                 PRI01880
      GO TO 300                                                         PRI01890
C                                                                       PRI01900
C OUTLINE                                                               PRI01910
C                                                                       PRI01920
  303 J=CCL                                                             PRI01930
  304 CALL LOCATE(CARD,COL,OTLCOD,6,CODE,&342)                          PRI01940
      IF(CODE.GT.2) GO TO 306                                           PRI01950
      CALL LOCATE(CARD,COL,OTLCOD,6,CODE,&342)                          PRI01960
      IF(CODE.NE.2) GO TO 342                                           PRI01970
      OTRKY=1                                                           PRI01980
      GO TO 308                                                         PRI01990
  306 IF(CODE.GT.4) GO TO 342                                           PRI02000
      CALL LOCATE(CARD,COL,OTLCOD(3),4,CODE,&342)                       PRI02010
      IF(CODE.NE.2) GO TO 342                                           PRI02020
      OTSKY=1                                                           PRI02030
  308 CALL LOCATE(CARD,COL,OTLCOD(5),2,CODE,&342)                       PRI02040
      IF(CODE-1) 304,304,320                                            PRI02050
  320 I=10                                                              PRI02060
      COL=J                                                             PRI02070
      CALL BCDVAL(CARD,COL,DUM,I,&342)                                  PRI02080
      GO TO 300                                                         PRI02090
  325 LISTKY=0                                                          PRI02100
      GO TO 300                                                         PRI02110
C                                                                       PRI02120
C     ERROR IN MAPS SPEC                                                PRI02130
C                                                                       PRI02140
  341 ERRNUM = 441                                                      PRI02150
      GO TO 220                                                         PRI02160
C                                                                       PRI02170
C         ERROR IN OUTLINE SPEC                                         PRI02180
C                                                                       PRI02190
  342 ERRNUM = 442                                                      PRI02200
      GO TO 220                                                         PRI02210
C                                                                       PRI02220
C TRAIN                                                                 PRI02230
C                                                                       PRI02240
  402 LETSZ=2                                                           PRI02250
      CALL BCDVAL(CARD,COL,DUM,LETSZ,&415)                              PRI02260
      IF (LETSZ .EQ. 0) GO TO 415                                       PRI02270
      DO 403 I=1,LETSZ                                                  PRI02280
      IF (DUM(I) .EQ. FBCD) TRFLD=1                                     PRI02290
  403 IF (DUM(I) .EQ. CBCD) TRCLS=1                                     PRI02300
      GO TO 300                                                         PRI02310
C                                                                       PRI02320
C TEST                                                                  PRI02330
C                                                                       PRI02340
```

```
FILE. . .     PRIRDR   FORTRAN  B1

  405 LETSZ=3                                                          PRI02350
      CALL BCDVAL(CARD,COL,DUM,LETSZ,&416)                             PRI02360
      IF (LETSZ .EQ. 0) GO TO 416                                      PRI02370
      DO 407 I=1,LETSZ                                                 PRI02380
      IF (DUM(I) .EQ. FBCD ) TSFLD=1                                   PRI02390
      IF (DUM(I) .EQ. CBCD ) ISCLS=1                                   PRI02400
  407 IF (DUM(I) .EQ. PBCD ) PCT=1                                     PRI02410
      GO TO 300                                                        PRI02420
C                                                                      PRI02430
C COPIES                                                               PRI02440
C                                                                      PRI02450
  411 I=1                                                              PRI02460
      CALL IVAL(CARD,COL,COPIES,1,&417)                                PRI02470
      GO TO 300                                                        PRI02480
C                                                                      PRI02490
C         ERROR IN TRAINING SPEC                                       PRI02500
C                                                                      PRI02510
  415 ERRNUM = 443                                                     PRI02520
      GO TO 220                                                        PRI02530
C                                                                      PRI02540
C         ERROR IN TEST SPEC                                           PRI02550
C                                                                      PRI02560
  416 ERRNUM = 444                                                     PRI02570
      GO TO 220                                                        PRI02580
C                                                                      PRI02590
C         ERROR IN TABLES SPEC.                                        PRI02600
C                                                                      PRI02610
  417 ERRNUM = 445                                                     PRI02620
      GO TO 220                                                        PRI02630
C*********************************************************************PRI02640
C                                                                      PRI02650
C         PROCESS THE RESULTS CARD                                     PRI02660
C                                                                      PRI02670
C*********************************************************************PRI02680
  420 IF (COL .EQ. 72) GO TO 200                                       PRI02690
      CALL CTLPRM (CARD,COL,RESCCD,3,CODE,&230)                        PRI02700
      GO TO (425,430,435), CODE                                        PRI02710
C                                                                      PRI02720
C         TAPE SPECIFICATION                                           PRI02730
C                                                                      PRI02740
  425 VECSZ = 1                                                        PRI02750
      CALL IVAL (CARD,COL,ROTAPE,VECSZ,&440)                           PRI02760
      IF (VECSZ .EQ. 0) GO TO 440                                      PRI02770
      GO TO 420                                                        PRI02780
C                                                                      PRI02790
C         FILE SPECIFICATION                                           PRI02800
  430 VECSZ = 1                                                        PRI02810
      CALL IVAL (CARD,COL,ROFILE,VECSZ,&445)                           PRI02820
      IF (VECSZ .EQ. 0) GO TO 445                                      PRI02830
      GO TO 420                                                        PRI02840
C                                                                      PRI02850
C         DISK SPECFICATION                                            PRI02860
C                                                                      PRI02870
  435 RESULT = CLASSR                                                  PRI02880
      GO TO 420                                                        PRI02890
C                                                                      PRI02900
C         ERROR IN TAPE SPEC.                                          PRI02910
C                                                                      PRI02920
  440 ERRNUM = 454                                                     PRI02930
      GO TO 220                                                        PRI02940
C                                                                      PRI02950
C         ERROR IN FILE SPEC                                           PRI02960
C                                                                      PRI02970
  445 ERRNUM = 455                                                     PRI02980
      GO TO 220                                                        PRI02990
C*********************************************************************PRI03000
C                                                                      PRI03010
C GROUP CARD                                                           PRI03020
C                                                                      PRI03030
C*********************************************************************PRI03040
  450 CALL GRPSCN(GRPNAM,GRPSTK,NOGRPS,COL,CARD,&460)                  PRI03050
      GO TO 200                                                        PRI03060
  460 STOPFG = 1                                                       PRI03070
      STKPTR = 0                                                       PRI03080
      NOGRPS = 0                                                       PRI03090
      CALL ERPRNT (446,'GOTO',&200)                                    PRI03100
C*********************************************************************PRI03110
C                                                                      PRI03120
```

```
FILE. . .    PRIRDR   FORTRAN  81
C SYMBOLS CARD                                                          PRIC3130
C                                                                       PRI03140
C*************************************************************************PRI03150
  500 I=34                                                              PRI03160
      CALL BCDVAL (CARD,COL, WORK,I,&540)                               PRI03170
      K=SYMCNT+I                                                        PRI03180
      IF (K .GT. 60) GO TO 543                                          PRI03190
  502 DO 505 J=1,I                                                      PRI03200
  505 SYMMTX(SYMCNT+J)=LWORK(4*J-3)                                     PRI03210
      SYMCNT=K                                                          PRI03220
      GO TO 200                                                         PRI03230
  540 ERRNUM = 447                                                      PRI03240
      GO TO 220                                                         PRI03250
C                                                                       PRI03260
C          TOO MANY SYMBOLS SPECIFIED. REDUCE NUMBER                    PRI03270
C                                                                       PRI03280
  543 WRITE (TYPEWR,9543) K,MAXCLS                                      PRI03290
      WRITE (PRNTR,9543) K,MAXCLS                                       PRI03300
 9543 FORMAT (' I0073  YOU HAVE ENTERED ',I3,' SYMBOLS. THE MAXIMUM',  PRI03310
     1        ' ALLOWED IS ',I3,'.'/9X,'EXCESS SYMBOLS WILL NOT ',      PRI03320
     2        'BE USED    (PRIRDR)')                                    PRIC3330
      I = I -(K-MAXCLS)                                                 PRI03340
      GO TO 502                                                         PRI03350
C*************************************************************************PRI03360
C                                                                       PRIC3370
C THRESHOLD CARD                                                        PRIC3380
C                                                                       PRI03390
C*************************************************************************PRI03400
  550 I=60-THSCNT                                                       PRIC3410
      IF (I .NE. 0) GO TO 551                                           PRI03420
      WRITE (TYPEWR,5510)  CARD                                         PRI03430
      WRITE (PRNTR,5510)  CARD                                          PRI03440
 5510 FORMAT (' I0074  MAXIMUM OF 60 THRESHOLDS ALREADY STORED.',       PRIC3450
     1        ' ABOVE CARD IGNORED    (PRIRDR)')                       PRI03460
      GO TO 200                                                         PRI03470
  551 CALL FVAL(CARD,COL,RWORK1,I,&590)                                 PRI03480
      DO 552 J=1,I                                                      PRIC3490
      IF (RWORK1(J) .GT. 99.0001) GO TO 591                             PRI03500
  552 THRES(THSCNT+J)=RWORK1(J)                                         PRI03510
      THSCNT=THSCNT+I                                                   PRI03520
      GO TO 200                                                         PRIC3530
C                                                                       PRI03540
C          SYNTAX ERROR IN THRESHOLD SPECS.                             PRI03550
C                                                                       PRI03560
  590 ERRNUM = 448                                                      PRIC3570
      GO TO 220                                                         PRIC3580
  591 ERRNUM = 263                                                      PRI03590
      GO TO 220                                                         PRI03600
C*************************************************************************PRIC3610
C                                                                       PRIC3620
C BLOCK CARD                                                            PRI03630
C                                                                       PRI03640
C*************************************************************************PRI03650
  600 BLKKY=1                                                           PRIC3660
  601 IF (COL .GE. 72) GO TO 200                                        PRI03670
      CALL CTLPRM(CARD,COL,BLKCOD,4,CODE,&230)                          PRI03680
      GO TO (605,610,615,625),CODE                                      PRI03690
C                                                                       PRI03700
C RUN                                                                   PRI03710
C                                                                       PRI03720
  605 I=1                                                               PRI03730
      CALL IVAL(CARD,COL,RUNNUM,I,&642)                                 PRIC3740
      IF (RUNNUM .GE. 10000000 .AND. RUNNUM .LT. 100000000) GO TO 601   PRI03750
      ERRNUM = 460                                                      PRI03760
      GO TO 220                                                         PRI03770
C                                                                       PRIC3780
C CALC                                                                  PRI03790
C                                                                       PRI03800
  610 CALC=1                                                            PRI03810
      GO TO 601                                                         PRIC3820
C                                                                       PRI03830
C LINES                                                                 PRI03840
C                                                                       PRI03850
  615 I=3                                                               PRIC3860
      CALL IVAL(CARD,COL,WORK,I,&643)                                   PRI03870
      DO 616 J=1,I                                                      PRI03880
      BLOCK(J)=WORK(J)                                                  PRI03890
  616 IF (BLOCK(J) .LE. 0) BLOCK(J)=1                                   PRIC3900
```

```
FILE. . .     PRTRDR   FORTRAN  B1

      IF (BLOCK(1) .GT. BLOCK(2)) GO TO 644                              PRI03910
      GO TO 601                                                          PRI03920
C                                                                        PRI03930
C COL                                                                    PRI03940
C                                                                        PRI03950
  625 I=3                                                                PRI03960
      CALL IVAL(CARD,COL,WORK,I,&645)                                    PRI03970
      DO 626 J=1,I                                                       PRI03980
      K=J+3                                                              PRI03990
      BLOCK(K)=WORK(J)                                                   PRI04000
  626 IF (BLOCK(K) .LE. 0) BLOCK(K)=1                                    PRI04010
      IF (BLOCK(4) .GT. BLOCK(5)) GO TO 646                              PRI04020
      GO TO 601                                                          PRI04030
C                                                                        PRI04040
C          ERRORS IN BLOCK CARD                                          PRI04050
C                                                                        PRI04060
  642 ERRNUM  = 449                                                      PRI04070
      GO TO 220                                                          PRI04080
  643 ERRNUM = 450                                                       PRI04090
      GO TO 220                                                          PRI04100
  644 ERRNUM = 451                                                       PRI04110
      GO TO 220                                                          PRI04120
  645 ERRNUM = 452                                                       PRI04130
      GO TO 220                                                          PRI04140
  646 ERRNUM = 453                                                       PRI04150
      GO TO 220                                                          PRI04160
C*************************************************************************PRI04170
C                                                                        PRI04180
C END CARD                                                               PRI04190
C                                                                        PRI04200
C*************************************************************************PRI04210
C                                                                        PRI04220
C CHECK FOR CORRECT CONTROL CARD INPUTS                                  PRI04230
C                                                                        PRI04240
  650 ASSIGN 651 TO GO                                                   PRI04250
C                                                                        PRI04260
C          CHECK FOR COMPLETE INPUT SPECS. I.E EITHER DISK OR            PRI04270
C          IF TAPE HAS TO GIVE  A FILE NUMBER. (TAPE NO. OF ZERO IS      PRI04280
C          OK SINCE THIS MAY MEAN USE SCRATCH TAPE FROM CLASSIFICATION)  PRI04290
C                                                                        PRI04300
  651 IF (RESULT .EQ. CLASSR .OR. RQFILE .GT. 0) GO TO 655               PRI04310
      ERRNUM = 456                                                       PRI04320
      GO TO 800                                                          PRI04330
C                                                                        PRI04340
C          CHECK FOR EXISTENCE OF SYMBOLS IF MAPS REQUESTED              PRI04350
C                                                                        PRI04360
  655 ASSIGN 656 TO GO                                                   PRI04370
  656 IF (NOMAPS .EQ. 0 .OR. SYMCNT .GT. 0) GO TO 660                    PRI04380
      ERRNUM = 457                                                       PRI04390
      GO TO 800                                                          PRI04400
  660 IF(NCGRPS.EQ.0)GO TO 850                                           PRI04410
      DO 664 I=1,60                                                      PRI04420
      IF(GRPSIK(I) .GT. NOGRPS) CALL ERPRNT (458,'STOP')                 PRI04430
  664 CONTINUE                                                           PRI04440
      GO TO 850                                                          PRI04450
C                                                                        PRI04460
C          GET AN EXTRA INPUT CARD FOR MISSING DATA                      PRI04470
C                                                                        PRI04480
  800 ERRCOR = 3                                                         PRI04490
      CALL ERPRNT (ERRNUM,'GOTO',&210)                                   PRI04500
C                                                                        PRI04510
C          IF RESULTS ARE ON TAPE, MOUNT IT                              PRI04520
C                                                                        PRI04530
  850 IF (CHKOUT) GO TO 900                                              PRI04540
      IF (RESULT .EQ. CLASSR) GO TO 870                                  PRI04550
      CALL MNTAPE (RQTAPE, RQFILE, 0)                                    PRI04560
      GO TO 900                                                          PRI04570
C                                                                        PRI04580
C          DO A READ OF THE RESULTS FILE TO BE SURE IT EXITS ON THE DISK PRI04590
C                                                                        PRI04600
  870 REWIND CLASSR                                                      PRI04610
      READ (CLASSR,END=880,ERR=880) I                                    PRI04620
      REWIND CLASSR                                                      PRI04630
      GO TO 900                                                          PRI04640
  880 CALL ERPRNT (459,'STOP')                                           PRI04650
C*************************************************************************PRI04660
C                                                                        PRI04670
C PRINT OUT OPTIONS                                                      PRI04680
```

```
FILE. . .    PRIRCR   FORTRAN  E1

C                                                                       PRI04690
C***********************************************************************PRI04700
  900 WRITE(PRNTR,9000)                                                 PRI04710
      IF (BLKKY   .EQ. 1) WRITE(PRNTR,9001)                             PRI04720
      IF (CALC    .EQ. 1) WRITE(PRNTR,9002)                             PRI04730
      IF (STATKY  .EQ. 1) WRITE(PRNTR,9003)                             PRI04740
      IF (CTRKY   .EQ. 1) WRITE(PRNTR,9004)                             PRI04750
      IF (CTSKY   .EQ. 1) WRITE(PRNTR,9005)                             PRI04760
      IF (TRCLS   .EQ. 1) WRITE(PRNTR,9006)                             PRI04770
      IF (TRFLD   .EQ. 1) WRITE(PRNTR,9007)                             PRI04780
      IF (TSFLD   .EQ. 1) WRITE(PRNTR,9008)                             PRI04790
      IF (TSCLS   .EQ. 1) WRITE(PRNTR,9009)                             PRI04800
      IF (PCT     .EQ. 1) WRITE(PRNTR,9010)                             PRI04810
      IF (THSCNT  .GT. 0) WRITE(PRNTR,9011)                             PRI04820
      IF (LISTKY  .EQ. 0) WRITE(PRNTR,9012)                             PRI04830
      IF (RESULT  .EQ. CLASSR) WRITE (PRNTR,9013)                       PRI04840
 9000 FORMAT (//' YOU HAVE SELECTED THE FOLLOWING DISPLAY OPTIONS'/)    PRI04850
 9001 FORMAT (T6,'CONSIDER A SUBSET OF CLASSIFIED AREA FOR MAP')        PRI04860
 9002 FORMAT (T6,'CONSIDER A SUBSET OF CLASSIFIED AREA FOR ',           PRI04870
     1        'CALCULATIONS')                                           PRI04880
 9003 FORMAT (T6,'PRINT REDUCED STATISTICS')                            PRI04890
 9004 FORMAT (T6,'OUTLINE TRAINING FIELDS')                             PRI04900
 9005 FORMAT (T6,'OUTLINE TEST FIELDS')                                 PRI04910
 9006 FORMAT (T6,'PRINT TRAINING CLASS PERFORMANCE')                    PRI04920
 9007 FORMAT (T6,'PRINT TRAINING FIELD PERFORMANCE')                    PRI04930
 9008 FORMAT (T6,'PRINT TEST FIELD PERFORMANCE')                        PRI04940
 9009 FORMAT (T6,'PRINT TEST CLASS PERFORMANCE')                        PRI04950
 9010 FORMAT (T6,'PRINT TEST FIELD PERCENTAGES')                        PRI04960
 9011 FORMAT (T6,'APPLY THRESHOLDS')                                    PRI04970
 9012 FORMAT (T6,'DO NOT LIST TRAIN AND TEST FIELDS')                   PRI04980
 9013 FORMAT (T6,'USE CLASSIFICATION RESULTS FROM DISK')                PRI04990
C                                                                       PRI05000
C         RETURN                                                        PRI05010
C                                                                       PRI05020
  991 IF (STOPFG .EQ. 1 .AND. .NOT. CPKOUT) CALL RTMAIN                 PRI05030
      RETURN                                                            PRI05040
      END                                                               PRI05050
```

4.4 GENERATING FUNCTIONAL LOAD MODULES

This section first defines the functional load modules. It then describes the flow of control and physical arrangement of programs in main storage. Finally, the programming of EXEC routines to generate functional load modules is described in detail. Figure 4-1 showing the main storage arrangement under various conditions is shown on the following page and is referred to throughout the discussion.

The functional load modules are CMS module files. They are created by loading the appropriate TEXT files and using the CMS GENMOD command to create the MODULE file. The name that is assigned to CMS MODULE file is the name of the first program or routine in the module. Thus, the name of the root load module is LARSMN since the FORTRAN program LARSMN is the first program (physically) in the module; likewise, the name of a functional MODULE file will be xxxSUP (where xxx is the first three letters of the function name) since the supervisor will always be loaded at the beginning of the functional load module.

In the following text, the word 'load' in lower case refers to the general process of loading from disk to main storage. 'LOAD' in upper case means loading by the CMS LOAD or INCLUDE command, 'LOADMOD' means loading by the CMS LOADMOD command. For further information on these commands refer to CMS Command and Macro Reference GC20-1818

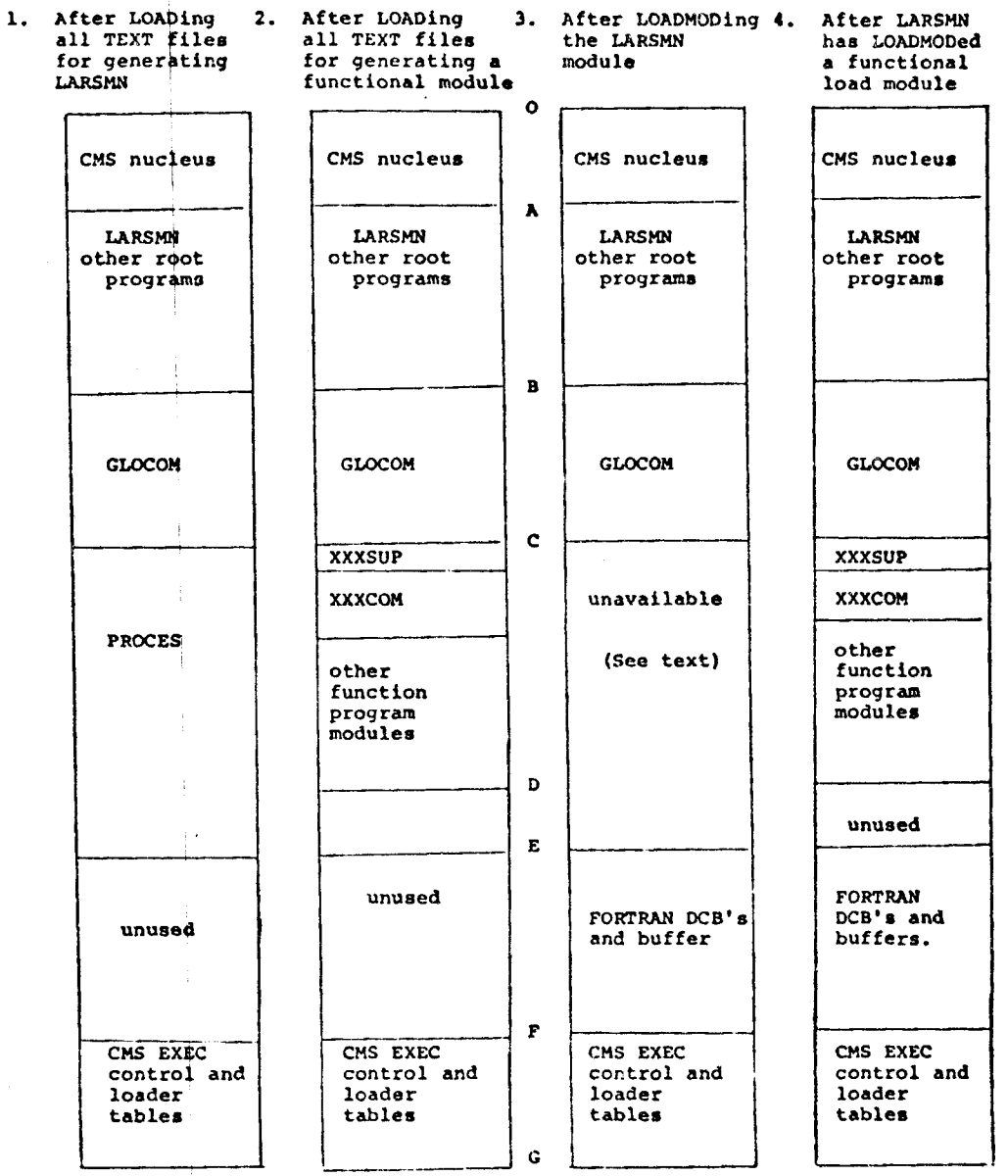


Figure 4-1. Main Storage Allocation

Flow of Control and Physical Arrangement

When the LARSFRIS control command RUN LARSYS is issued, the RUNLS EXEC is invoked. RUNLS uses the CMS LOADMOD command to load the root load module (LARSMN). The contents of main storage at this point are shown in the figure in column 3. When LARSMN reads a Function Selector card, it calls BLOAD which uses the LOADMOD command to load the requested functional module. The contents of main storage at this point are shown in the figure in column 4. After the functional module is loaded, LARSMN passes control to the first byte of the module (address C in the figure).

The figure also shows the contents of main storage at various times. Each of the significant addresses shown on the figure are represented by the following letters:

O - Address zero, start of main storage of the virtual machine.

A - End of CMS nucleus and start of the storage that is available for user programs. The FORTRAN program LARSMN always starts at this point. The remainder of the programs in the root follow LARSMN. These include LARSFRIS support subroutines such as TAPOP and CTLWRD and FORTRAN library subroutines. These programs end at address B.

B - GLOCOM is loaded at the end of the other programs in the root. B is the beginning of GLOCOM.

C - The end of GLOCOM. All functional load modules are loaded at this point, with the address 'C' being the beginning address of the load module supervisor. LARSMN always loads a functional load module at this address and branches to it through a CALL to a dummy subroutine named PROCES. This dummy subroutine is used to create a symbolic address for address 'C' in the following way:

- When the LARSMN load module is loaded during the generation procedure, the dummy subroutine PROCES is loaded at address 'C'. This resolves a CALL to PROCES contained in LARSMN, which becomes a branch to address 'C'.
- When the GENMOD command is issued to actually generate the LARSMN load module, it ensures that PROCES is not included in the module file.
- When the generated LARSMN load module subsequently loads a functional load module at address 'C' and CALLs PROCES, the call results in a branch to address 'C', which is now the beginning address of a functional load module supervisor.

D - Highest address used by the functional module. All functional programs including the supervisor, common, programs unique to this module and support programs lie between C and D.

E - Start of free storage. Free storage is used by FORTRAN for storing its data control blocks and for I/O buffers for FORTRAN data sets. The address of 'E' must be greater than the ending address of the largest LARSFRIS functional load module. If any LARSFRIS functional load module has an ending address greater than 'E', its loading by LARSMN will overlay data control blocks and I/O buffers established by LARSMN. This will result in strange I/O errors. The address is fixed during the generation of the LARSMN load module, based on the size of the PROCES subroutine. PROCES contains a large array to make it larger than the largest LARSFRIS load module. When LARSMN is GENMODed, CMS stores the address E in the MODULE file (even though as previously stated, PROCES itself is not in the MODULE file). When LARSMN is subsequently loaded, CMS uses the address 'E' from the MODULE file to set the start of free storage.

F - Start of an area of high memory used by CMS. The very top of memory is used by CMS for loader tables. The storage just below the loader tables is used by CMS for EXEC routine control.

G - End of main storage of the virtual machine.

Programming Module Generation EXEC Routines

The first part of this section describes the GLARSMN EXEC routine used to generate the root load module; the second part describes the creation of EXEC routine to generate functional load modules. The GLARSMN and GCLASUP routines are shown (as examples) in Figures 4-2 and 4-3 at the end of the section.

GLARSMN begins by executing an EXEC routine called LLARSMN which LOADs the programs in the root. It then uses the CMS USE command to load PROCES. The LARSMN module is then generated by a GENMOD command which specifies that the new MODULE file contains the contents of main storage only from the beginning of the program LARSMN to the beginning of the program PROCES. The GENMOD command creates the file with a filemode of A2. This must also be done in EXEC routines that create functional load modules. At the time GENMOD is issued main storage appears as shown in column 1, of Figure 4-1.

After generating the module, GLARSMN types out an address on the typewriter terminal. This is done by using the CP function DISPLAY to display the contents of storage cell 574 (hex). 574 is a word called LOCCNT in the CMS NUCON area, which contains the address of the next available free storage. This address may change in subsequent releases of CMS.

GLARSMN finishes with a procedure common to all load module generation EXEC routines. The procedure prints the load map after first RENAMing it to have a filename the same as that of the EXEC generating the module. If NOMAP was specified as a parameter to the EXEC routine, the map is not printed.

The EXEC routines (GXXXSUP) that generate functional load modules being the same as GLARSMN, i.e., by executing LLARSMN to load the programs in the root. This is necessary to enable the loader to resolve the addresses of calls from the functional programs to subroutines in the root. CMS INCLUDE commands are then issued to load the functional programs. STACK HT is in effect during all but the last of these INCLUDE commands, in order to delete multiple repetitions of the message

THE FOLLOWING NAMES ARE UNDEFINED: (etc.)

Before the last INCLUDE command, STACK RT is issued so that the message will type for the final INCLUDE and the programmer will be informed of any unresolved references. The message will always state that PROCES is undefined. This is not an error.

The GENMOD command is then issued to create a MODULE file containing the contents of main storage from the beginning of the supervisor (address C) to the end of all loaded programs (address D). The address D is then displayed on the typewriter by using the CP DISPLAY function to display address 574 (hex). The programmer should always be certain that this value is less than the address E that is typed out by GLARSMN. The EXEC ends just like GLARSMN by printing the load map if one was requested.

```
FILE: GLARSMN  EXEC      Y   PURDUE UNIVERSITY / LARS

&COMMENT GLARSMN   LARS  0069
         &GOTO -START
         GLARSMN GENERATES THE ROOT LOAD MODULE
         WRITTEN 11/20/72 BY EARL RODD
         REVISED 2/15/79 BY LOUIS LANG
 IF &1 = NOMAP, LOAD MAP WILL NOT BE PRINTED
-START   &CONTROL OFF
         EXEC LLARSMN
         INCLUDE PROCES (NOAUTO)
         GENMOD LARSMN MODULE A2
         &TYPE FREE STORAGE STARTS AT:
         CP DISPLAY 574
         ERASE &0 MAP
         RENAME LOAD MAP A1 &0 MAP A1
         LISTF &0 MAP A1 (E D)
&BEGSTACK LIFO
FILE
CHANGE /:/./
FILE
OVERLAY &STACK LIFO I THIS LOAD MAP CREATED
OVERLAY  ------------------------------------------------
NEXT
&END
&STACK HT
         EDIT CMS EXEC
         EXEC CMS
&STACK HT
         EDIT &0 MAP
&STACK RT
         ERASE CMS EXEC
         &IF &$ EQ NOMAP &EXIT
         PRINT &0 MAP
```

Figure 4-2. The LARSMN Load Module Generation EXEC

```
FILE: GCLASUP  EXEC     F   PURDUE UNIVERSITY / LARS

&COMMENT GCLASUP   LARS  0071
         &GOTO -START
  GCLASUP GENERATES THE CLASUP LOAD MODULE.
  WRITTEN 11/20/72 BY EARL RODD
  REVISED 13 FEBRUARY 1979 BY LOUIS  LANG
 IF ANY ARGUMENT = NOMAP, THE LOAD MAP WILL NOT BE PRINTED
-START   &CONTROL OFF
         EXEC LLARSMN
&STACK HT
 INCLUDE CLASUP CLACOM CLARDR CLAINT MCONTX CONTEX CLSFY1 CCVIN  (NOAUTO
         INCLUDE CLSFY2 CLASS THRESC GRPSCN (NOAUTO
   INCLUDE  MMTAPE STAT REDSTA CLSCHK REDSAV LAREAD TSPACE  (NOAUTO
         INCLUDE GTSERL MINV (NOAUTO
&STACK RT
         INCLUDE WRTTRN WRTMTX DECCLS HEADER (NOAUTO
         GENMOD CLASUP MODULE A2
         &TYPE HIGHEST STORAGE USED:
         CP DISPLAY 574
         ERASE &0 MAP
         RENAME LOAD MAP A1 &0 MAP A1
         LISTF &0 MAP A1 (E D)
&BEGSTACK LIFO
FILE
CHANGE /:/./
FILE
OVERLAY &STACK LIFO I THIS LOAD MAP CREATED
OVERLAY  ------------------------------------------------
NEXT
&END
&STACK HT
         EDIT CMS EXEC
         EXEC CMS
&STACK HT
         EDIT &0 MAP
&STACK RT
         ERASE CMS EXEC
         &IF &$ EQ NOMAP &EXIT
         PRINT &0 MAP
```

Figure 4-3. The CLASUP Load Module Generation EXEC

4.5 LARSFRIS ERROR HANDLING

This section describes how LARSFRIS handles three kinds of errors; application errors detected in LARSFRIS application programs, FORTRAN library program errors, and program interrupts.

Application Program Errors

Each of these errors has a unique error message number assigned to it. When the error message is printed, the number is prefixed with an 'E'. Most of the error messages are printed and/or typed by a CALL from the functional routine to the subroutine ERPRNT. (See the module documentation for further details on this routine). The CALL passes ERPRNT the error number, which it uses to retrieve the text of the error message from a file called ERROR MESSAGE (located on the Y-disk and assigned DSRN 8). Some programs do not use this facility. Instead they print and/or type the error messages directly. This is true in EXCOMD and in support programs such as CTLWRD which are used by programs other than those in LARSFRIS.

In addition to the error number, ERPRNT is passed a disposition code which tells it whether to terminate a function or to continue. ERPRNT can either print and type the message or only type it. Errors which are mainly informational and need not terminate a function are only typed so that the message will not interrupt formatted printed output.

When an error occurs which necessitates termination of a function, the function either calls ERPRNT with a disposition of 'STOP' or terminates directly. Termination is accomplished via a call to RTMAIN (whether in a functional program or in ERPRNT). A FORTRAN STOP statement is not used since this would terminate the entire run rather than just the function. RTMAIN will pass control to LARSMN which will read any remaining cards for this function and go on to the next function.

Error messages may require a variable numeric data, such as "iii" in

```
Ennn A POOL IS MISSING FROM THE DATA.  POOL NUMBER IS
                                       iii
```

Such an error message is implemented by a call to ERPRNT to write the first line, and the numeric information is written from the program (typed and printed) on a second line. The numeric information should be indented at least ten spaces.

FORTRAN Library Errors

FORTRAN library routines can detect errors such as I/O errors and improper CALLS (e.g., an attempt to take a square of a negative number). These errors cause IHCnnn

errors to appear on the printed output. The errors are documented in the OS FORTRAN G Programmers Guide. Error number 208 (for exponent underflow) is suppressed by use of the FORTRAN ERRSET routine called from LARSMN. The error will not print; however, in the FORTRAN summary of errors at the end of the execution, the number of occurrences will appear. Other calls to ERRSET are made to terminate execution after one occurrence of all other FORTRAN errors (thus bypassing attempted "fix-up's" that cause more trouble than they prevent).

Program Interrupts

LARSFRIS is designed to intercept all program interrupts except 8 (fixed-point overflow), 10 (decimal overflow) and 14 (significance), which are marked off by the FORTRAN Library routine IHCFCOMH, and must be left disabled. (See ERRINT module documentation for more details). This is done by use of the SPIE macro (See IBM publication GC28-6646-6, S/360 Operating System Supervisor Services) executed from a subroutine called ERRINT. ERRINT is called by LARSMN at the beginning of execution to initialize the error interception capability. (See module documentation of ERRINT for detail of its operation). When a program interrupt occurs, control is passed by CMS to an entry in ERRINT called ERRINT1. If the program interrupt is code 13 (exponent underflow) it is passed to the FORTRAN error handling routine; if it is any other code, the ERPRT1 entry in ERPRNT is called (see module documentation). The latter will print a LARSFRIS error message and terminate execution.

4.6 USE OF THE LARSFRIS SYSTEM FOR TEST RUNS

In program development (modification and debugging of existing functions and writing of new functions), it is necessary to make test runs. The disk hierarchy of CMS makes it very convenient to use most of the standard program modules when testing new modules. The discussion below first lists the steps necessary to make a test run. These are followed by supplemental information which is useful in certain situations.

Steps to make a test run:

1. The standard system programs and load modules will be available on disk. The system programmer is responsible for configuring a programmer's virtual machine to give him read-only access to all standard program modules and load modules or informing him of how to LINK to such disks.

2. All subroutines which are to be modified should be copied to the A-disk. It is strongly recommended that EXEC routines be written to perform this task since the CMS COPYFILE command requires considerable typing. If the programmer is writing a new function rather than modifying an existing one, then this step is not needed, since all new programs will be created on the A-disk.

3. Use the EDIT command to make the required modifications for the test run.

4. Compile (and/or assemble) the modified programs.

5. Use the load module generation EXEC routine to create a test version of the load module. The MODULE created will be on the A-disk. It will include the modified programs from the A-disk instead of the standard programs since the A-disk is searched before any other disks to locate the programs. (When writing a new function, time will be saved if a load module generation EXEC routine is written at the beginning rather than later.)

6. Begin LARSFRIS execution by executing the EXCOMD routine (typing EXCOMD). Then enter CCINPUT and any other needed control commands and finally issue the RUN command to start the program running. Use of CCINPUT is recommended since the control card deck can be easily modified using EDIT and can be used as many times as needed without the need to read the deck in each time.

7. After completing the test run, the programmer can return to CMS by entering the control command 'CMS'. (This command is not documented in the User's Manual).

Supplemental Information

1. Intermediate printouts can be directed to the typewriter (unit 16) or printer (unit 6). Also, the programmer is free to use disk files not used by the function he is running. When adding new data set reference numbers, first consult the system programmer to insure their validity. Be sure when writing to symbolic unit numbers (e.g., TYPEWR or PRNTR) that the subroutine has defined these variables (i.e., the subroutine has GLOCOM or has defined the variables in a DATA statement).

2. A function can be killed anytime the keyboard is unlocked for a control card correction by typing in 'KILL'. This will cause the function to terminate. All disk files will be intact.

3. If a COMMON block needs to be changed, it is necessary to copy all program modules containing the COMMON block onto the A-disk (see the section on COMMON block usage).

4. If a module in the Root Load Module must be modified to make the test run, then it will be necessary to generate a copy of LARSMN on the A-disk. This is done as described in Steps to make a test run above. If a new root load module is created, it is also

necessary to create a new version on the P-disk of any functional load modules that are used. This is necessary because the length of test version of LARSMN is different from the old one, and consequently, the address of the beginning of the functional load module will be different (as well as the addresses in the root of subroutines called by the functional load module).

5. The CMS DEBUG facility provides a powerful debugging tool. (Note that this facility is different from the FORTRAN G facility and requires a knowledge of assembler language.) Two features of LARSFRIS place small restrictions on the use of the facility:

   - The ERRINT subroutine must not be executed. This is due to the fact that DEBUG sets breakpoints by replacing the actual instruction operation code with an invalid operation code. Normally CMS interprets the resulting error interrupt as a breakpoint. However, when ERRINT is executed, LARSFRIS intercepts and handles these interrupts and will produce an E901 error message instead.

- LARSFRIS uses an overlay structure. When LARSMN is loaded in RUNLS EXEC, it is not possible to set breakpoints in a functional load module, since the load module has not been loaded into main storage. Any breakpoint that is set will be overlaid when the subroutine BLOAD actually loads the load module.

The following explanation of the use of DEBUG contains the action required to use DEBUG in LARSFRIS. Following the explanation is a sample terminal session showing the steps. First, two breakpoints must be set in the root. To do this, a copy of RUNLS EXEC must be created on the A-disk and modified to contain a call to DEBUG after LARSMN is LOADMODed and before it is STARTed (step 1). When DEBUG is entered, a breakpoint must be set at the entry to ERRINT. This address is on the load map. This will permit ERRINT to be bypassed (step 2). A second breakpoint (step 3) is set at location LOADED (also obtained from the load map).

When ERRINT is entered, (step 4) the contents of register 14 are displayed to find the address to which ERRINT is to return. Then, using the DEBUG "GO"

command, control is passed directly to that address (thus bypassing execution of ERRINT). When location LOADED is reached (step 5), this means that the overlay module has been LOADMODed by subroutine BLOAD and BLOAD is about to return control to LARSMN to start execution of the function. At this point, a breakpoint is set in the functional load module. Step 6 shows that the breakpoint in the load module was reached. In this example, this breakpoint was the entry point to a subroutine in the overlay module. The programmer then set the origin to this address and set a breakpoint in this subroutine. It turned out that this breakpoint was never reached. A caution should be noted here. When breakpoints have been set and not executed, CMS should be re-ipl'ed before using DEBUG again.

If this procedure is to be used several times, steps 2-5 can all be set up as stacked lines before the DEBUG command in the RUNLS EXEC.

6. Some very important cautions should be noted. Whenever the programmer creates a new load module, any TEXT files that are on the A-disk will be used in preference to the standard program TEXT files. This is a useful feature in making test runs. However, it means that once a

```
.copy runls exec n = = a
R; T=0.09/0.21 15:45:05

.edit runls exec                          Programmer makes his
EDIT:
.1 /loadmod/                              own copy of RUN EXEC      (1)
        LOADMOD LARSMN
.1          debug                         and adds line "debug"
.file
R; T=0.07/0.19 15:45:37

.excomd

T=0.18/0.46 15:45:50
.cdinput flmclas cc
T=0.05/0.08 15:45:58
.statdeck use fleming
T=0.18/0.34 15:46:07
.run lasersys
DMSDBG728I DEBUG ENTERED.

.break 1 2bb18            location of ERRINT (from load map)        (2)

.break 2 2bf54            location of LOADED (from load map)        (3)

.return
DMSL10740I EXECUTION BEGINS...
DMSDBG728I DEBUG ENTERED.
BREAKPOINT 01 AT 02BB18

.gpr 14
4202092C                  location to return to from ERRINT         (4)

.go 2092c

DMSDBG728I DEBUG ENTERED.
BREAKPOINT 02 AT 02BF54

.break 3 51678            breakpoint set in overlay module          (5)

.go
 I0112  CLASSIFYPOINTS FUNCTION REQUESTED    (CLASUP)
 I0002 TAPE 9999 HAS BEEN REQUESTED ON UNIT 181    (TAPMOUNT)
TAPE 181 ATTACHED
 I0003 TAPE READY...    EXECUTION CONTINUING    (TAPMOUNT)
 I0032  REDUCED STATISTICS COMPUTED.    (REDSAV)
 I0034  ALL CONTROL AND DATA CARDS HAVE BEEN READ    (CLAINT)
DMSDBG728I DEBUG ENTERED.
BREAKPOINT 03 AT 051678

.origin 51678             set at breakpoint in subroutine           (6)

.x 160
000543dC

.go
 I0002 TAPE 2652 HAS BEEN REQUESTED ON UNIT 182    (TAPMOUNT)
TAPE 182 ATTACHED
 I0003 TAPE READY...    EXECUTION CONTINUING    (TAPMOUNT)
 INNNN  DATA IS IN LARSYS FORMAT   (GADRUN)
 I0036  DESIRED RUN FOUND ... 77010200    (GADRUN)
 I0049    100 OUT OF    176 LINES ARE CLASSIFIED    (CLSFY2)
 I0040  CLASSIFYPOINTS FUNCTION COMPLETED    (CLASUP)
 I0103  CPU TIME USED WAS   49.817 SECONDS.    (LARSMN)

 I0004  END OF INPUT DECK - RUN COMPLETED    (LARSMN)
 I0050  TOTAL CPU TIME FOR THIS RUN WAS   49.851 SECONDS.    (LARSMN)
TAPE 181 DETACHED
TAPE 182 DETACHED
T=43.82/50.57 16:16:55
```

test version is no longer needed it should be erased. Also, any MODULE files left on the A-disk will continue to be used rather than the standard system modules. If the system programmer has changed the root load module since the programmer last created a functional module on his A-disk, that functional module will no longer work. For this reason, it is recommended that load modules for test versions not be kept but rather that they be regenerated whenever needed. In light of the above environment the first step to be taken when unforseen events take place is to check for unwanted TEXT or MODULE files on the A-disk. It is also important that when a program source is modified, it must be compiled or assembled before it will appear in the newly generated load module.

4.7 ATTACHING AND DETACHING OF TAPE DRIVES

The attaching and detaching of tape drives is handled automatically by LARSFRIS and the computer operator and requires no user action. When a tape is needed by a processing function, it calls the subroutine MOUNT, passing to it the tape number, DSRN, and the characters 'RI' or 'RO' to indicate either ring in or ring out. MOUNT assumes that DSRN 11 is TAP1, DSRN 12 is TAP2, and DSRN 13 is TAP3. It checks the NUCON device table, however, to find the assigned unit address for each virtual tape drive.

MOUNT then sends the CP operator a message requesting that the tape with the number specified by the user be mounted on the appropriate virtual tape drive. If the required physical drive is not attached to the user's virtual machine, the operator is notified and must first attach one and then mount and ready the tape. If the drive is already attached, the operator need only mount and ready the tape. The LARSFRIS programs are designed so that when MOUNT is called, the tape drive that is required is either not attached or not ready.

MOUNT goes into a wait state via the CMS WAIT macro after it has requested the tape. When the tape becomes ready, CP will pass the resultant interrupt to CMS and the program will automatically resume processing. The interrupt will not be passed

to CMS if the operator readies the tape drive before attaching the drive. In addition, the operator cannot recover from this condition by unloading and then readying the tape. It is necessary to detach and then properly re-attach the tape drive, or else have the user ready the unit via the CP 'READY' command.

At the processing level (i.e., during the execution of a single input deck), tapes are attached only while actually needed. This is implemented by detaching all attached drives when a processing function is requested which does not need them. The detaching is done by calling UNMNT immediately after a Function Selector Card has been detected. UNMNT has a table of the tape drives that each function requires and it detaches any that are not needed. The status of tapes remain unchanged during execution of initialization functions, since these precede the Function Selector Card. The Cluster function reads a Multispectral Image Storage Tape at the beginning of the functional processing and then has no further use for it while the computing is taking place. Therefore, this drive is detached immediately after all of the data has been read from the tape. Since it is possible for the user to re-ipl as a method of terminating a run, thus bypassing the code at the end of RUNLS which detaches tapes, EXCOMD detaches any tape drives before executing any commands. It determines which drives are attached by issuing a TAPE RUN command and checking the return code.

4.8 IMPLEMENTATION OF THE CONTROL CARD CHECKOUT OPTION

The control card checkout option permits a user to verify the accuracy of the control and data cards in an input deck without actually executing the deck. This feature has been implemented by using the normal LARSFRIS programs and bypassing coding which actually executes the function. Only coding which reads and checks control and data cards is executed.

The user invokes control card checkout by including the -CHECKOUT Initialization Function control card in his input deck. When this card is read, LARSMN sets the logical flag 'CHKOUT' in GLOCOM to ".TRUE." rather than its normal value of ".FALSE.". The "-RESET" Initialization function resets the flag to '.FALSE.'.

The following are the general specifications for the control card checkout option.

1. All control and data cards are read.
2. All control and data cards are checked within the limitations that are mentioned below.
3. No tapes are mounted. This means that some data checking cannot be performed.
4. Any disk data sets created by a previous function are not read since this function may not have been executed

(and probably will not have been, since it was probably run with the control card checkout option).

5. No data is written to disk data sets.
6. No cards are punched.
7. The messages indicating 'FUNCTION REQUESTED' and 'FUNCTION COMPLETED' are typed. The message 'ALL CONTROL AND DATA CARDS HAVE BEEN READ' may or may not be typed, depending upon the structure of the functional programs.

Below are listed special cases and exceptions to the above specifications.

- Those functions that assume that the run number equals the current run when no run is given in the control cards (HISTOGRAM, COLUMNGRAPH, and LINEGRAPH) do not make any check for the presence of a run number since there is no way to know if a previous function would have left a run mounted.

- The check normally made for an invalid group number on a TEST card (made in RDFLDS) cannot be made at all in PRINTRESULTS and cannot be made in SAMPLECLASSIFY unless the Statistics File is available (see below). This check requires the statistics to determine the number of groups with just one class. In both of the

above cases, the number of groups is set to 60 before the call to RDFLDS. This causes any group number in the possible range of 1-60 to be accepted.

- The Statistics File is an exception to the specification that no disk data sets are read or written. The following rules apply to it.

    a. If a Statistics File is present in the input deck, it is read and the disk file is created as usual. This allows all normal data checking concerning the statistics to be performed.
    b. If the file is not present in the input deck (indicating that statistics are expected on disk), a check is made to see if the Statistics File exists on disk. This check is made by determining whether the first record is 'EOS'. If this is the case, the file does not exist and data checking which requires it cannot be performed. If the file does exist, all normal data checking is performed.

These rules affect the Separability, Classifypoints and Sample-classify functions. Since PRINTRESULTS reads statistics from the Results File, the statistics are never read when control card checkout is in effect.

SECTION 5

LARSFRIS DATA ORGANIZATION

SECTION 5

LARSFRIS DATA ORGANIZATION

This section describes the data files that are used in LARSFRIS and gives the detailed formats and the contents of each file. The section is divided into three subsections:

5.1 LARSFRIS Data Set Reference Numbers

5.2 LARSFRIS Processing Level Files

5.3 LARSFRIS System Information Files

5.4 Other LARSFRIS Files

5.1 LARSFRIS DATA SET REFERENCE NUMBERS

The table that follows gives the symbolic DSRN, the DSRN, the file description and the CMS FILEDEF command for each of the data sets in the LARSFRIS system. The files are arranged in numeric order by DSRN. The descriptions of the processing level files that follow in subsection 5.2 are presented in the same order. Note that DSRN 14 is not used at this time. It may be used in the future for an additional tape assignment.

Symbolic	DSRN	Description	Filedef
SDATA	1	Statistics File	FILEDEF 1 DISK STATS DATA D1 (NOCHANGE)
HDATA	2	Histogram File	FILEDEF 2 DISK HISTO DATA D1 (RECFM F LRECL 360 BLKSIZE 360 XTENT 61 NOCHANGE)
CLUSTX	3	Cluster Scratch File	FILEDEF 3 DISK CLUSTER SCRATCH D4 (RECFM VS BLKSIZE 800
PRESUX	4	Printresults Scratch File	FILEDEF 4 DISK PRESULT SCRATCH D1 (LRECL 133 BLKSIZE 133 NOCHANGE)
CRDRDR	5	Virtual card reader	FILEDEF 5 READER (RECFM F NOCHANGE)
PRNTR	6	Virtual printer	FILEDEF 6 PRINTER (RECFM FA NOCHANGE)
PNCH	7	Virtual punch	FILEDEF 7 PUNCH (NOCHANGE)
ERRMSG	8	Error Message File	FILEDEF 8 DISK ERROR MESSAGE M1 (LRECL 80 BLKSIZE 80
RUNFIL	9	System Runtable	FILEDEF 9 DISK RUNTABLE FILE O4 (RECFM VS BLKSIZE 808
TTFLDX	10	Training and Test Fields File	FILEDEF 10 DISK TRNTEST FIELDS D4 (RECFM VS LRECL 800 BLKSIZE 800)
MAPTAP	11	Classification Results Tape	FILEDEF 11 TAP1 (RECFM VS LRECL 1500 BLKSIZE 1500)
SEPTPX	11	Separability Scratch Tape	FILEDEF 11 TAP1 (RECFM VS LRECL 1500 BLKSIZE 1500)

Symbolic	DSRN	Description	Filedef
DUPRUN	11	CHANNELTRANSFORM	FILEDEF 11 TAP1 (RECFM VS LRECL 1500 BLKSIZE 1500)
DATAPE	12	Multispectral Image Storage Tape	None required (Only TAPOP is used)
CPYOUT	12	Copyresults Output Tape	FILEDEF 12 TAP2 (RECFM VS LRECL 1500 BLKSIZE 1500)
DUPLTP	13	Transferdata Output Tape	FILEDEF 13 TAP3 (RECFM F LRECL 80 BLKSIZE 80 NOCHANGE)
SCNDTP	14	Classification Results Tape	FILEDEF 14 TAP4 (RECFM VS BLKSIZE 1500 LRECL 1500)
KEYBD	15	Terminal keyboard (input)	FILEDEF 15 TERM (LRECL 120 BLKSIZE 120 NOCHANGE)
TYPEWR	16	Terminal typewriter (output)	FILEDEF 16 TERM (LRECL 120 BLKSIZE 120 NOCHANGE)
CLASSR	17	Classification Results File	FILEDEF 17 DISK CLASSIFY RESULTS D4 (RECFM VS BLKSIZE 800 LRECL 800)
SEPARX	18	Separability Scratch File	FILEDEF 18 DISK SEPAR SCRATCH D4 (RECFM VS BLKSIZE 800
FLDBND	19	Field Boundaries File	FILEDEF 19 DISK FIELD BNDRIES D4 (RECFM VS BLKSIZE 800)
CLASSX	20	Classifypoints Scratch File	FILEDEF 20 DISK CLASSIFY SCRATCH D4 (RECFM VS BLKSIZE 800 LRECL 800)

5.2 PROCESSING LEVEL FILES

This subsection contains the descriptions of all files that are used at the LARSFRIS processing level. This includes not only the remote sensing application files, such as the Statistics File, but also the support files, such as the Runtable, that are used at this level.

The functional processing disk files in this group (all of the disk files except ERROR MESSAGE and RUNTABLE FILE which are system files stored on the M-disk) are stored on the user's temporary disk (D-disk) during functional processing. When the user initially logs into the system and issues the 'i larsys' command, all of these files are erased. They will also be erased if he later re-issues the 'i larsys' command and when he logs off the system via the 'quit' command. During the time between the first 'i larsys' command and any subsequent 'i larsys' or 'quit' command, the files are handled as follows:

- Scratch files (filetype = SCRATCH) are always erased as soon as the function that is using them is completed.
- All of the other disk files except CLASSIFY RESULTS remain intact on the D-disk.
- CLASSIFY SCRATCH is erased each time the Classifypoints function is executed, immediately before it is executed. Except for these erasures caused by re-executing CLASSIFY-POINTS, the file remains intact on the D-disk.

The files are described in the following sequence:

STATISTICS FILE (DSRN 1)

The Statistics file is created by the Statistics function or the Cluster function on disk (as STATS DATA) and may also be punched on cards. Both forms of the file are in the identical card image format (RECFM = F, and record length of 80). The card images are sequenced in columns 73-80 of each record. The sequence field is in I8 format. The first record is sequence number 1 and each record is incremented by 1.

File Usage:

The file (from either storage medium) is a primary input for the Separability, Classifypoints, BIPLOT, RATIOMEANS, MERGESTATISTICS, SECHO and Sampleclassify functions. Also, the Classification Results File contains the Statistics File, though the record format and record size are different.

There are eight types of records on the file. In the Statistics function the first and second type records (the ID and training field portions of the file) are written by STAINT. The remainder of the file is written by PCHSTA which also punches the deck if a card version is requested in STATISTICS. In Cluster the entire file is written by CLUPRO which also punches the deck if it is requested. The file is read, wherever it is used in other functions, by subroutine STAT which transfers it to disk. REDSTA reads the disk file and actually reads the statistics into memory. CLAINT reads the Statistics File to count the records and to transfer the file to the results file.

WRTTRN reads the first record and then reads the training fields via LAREAD and then flushes the rest of the file. In the Statistics function, LEARN reads the training fields from the STATS DATA file when they are the only information on the file. If the Statistics file is written by the Cluster function, there are no actual training fields and these records are replaced by one dummy field card for every class.

File Format:

The Statistics File contains 8 types of records. The content of these records and the number of each record in the file is described below.

RECORD TYPE 1

There is one record type 1.

Columns	Format	Description
1-32	Alpha	'LARSYS VERSION 3 STATISTICS FILE'
39-43		blank
40-44	I1	1 = hexadecimal format or 0 = character format (see record type 6 & 7)
41-45		blank
73-80	I8	Sequence number

RECORD TYPE 2

The type 2 records are copy of the training field deck including the 'CLAS' cards. When written from the Statisitcs function, the only differences between these records and the training field data deck are that any 'CLAS' cards which had no class names on them have 'NONE' in columns 17-20 and all records have the sequencing in columns 73-80. When written by the Cluster function, the 'CLAS' cards have 'CLASS NS-' in columns 1-12, the class number in column 13, and the total number of classes in column 17. The dummy field cards have the run number in columns 1-8 and class number information in columns 51-54. All other data fields contain 9's. All records are sequenced in columns 73-80 in the normal manner.

RECORD TYPE 3

There is one record type 3.

Columns	Format	Description
1-5	I5	Number of classes in the statistics
6-11	Alpha	' CLASS'
12-16	I5	Total number of training fields
17-22	Alpha	' FIELD'
23-27	I5	Number of channels used in the statistics
28-36	Alpha	' CHANNELS'
37-72		blank
73-80	I8	Sequence number

RECORD TYPE 4

There is one record type 4 for each channel used in the statistics run.

Columns	Format	Description
1-4	Alpha	'CHAN'
5-7	I3	Channel number
8-18	Alpha	' WAVELENGTH'
19-23	F5.2	Low end of spectral band (in micrometers)
24	Alpha	'-'

25-29	F5.2	Upper end of spectral band (in micrometers)
30-34	Alpha	' CODE'
35-37	I3	Calibration code
38-40	Alpha	' CO'
41-47	F7.2	Value of CO used
48-50	Alpha	' C1'
51-57	F7.2	Value of C1 used
58-60	Alpha	' C2'
61-67	F7.2	Value of C2 used
68-72		blank
73-80	I8	Sequence number

RECORD TYPE 5

There are (NC-1)/9+1 record type 5's. where NC = Number of classes.

Columns	Format	Description
1-7	Alpha	'NO. PTS'
8-70	7I9	7 entries each of which is the number of points in a class. Entry i on the jth record type 5 is the number of points in class ((j-1) *9+i)

RECORD TYPE 6

There is one set of record type 6's for each training class. Each set contains the means of the data values for each channel used in the statistics. The beginning of each class is on a new record. Each record is of the form 'MN' in columns 1 and 2 and data in columns 3-72. Columns 73-80 contain the sequence number. The data can be in either of two formats, and is indicated by the flag on record 1. If the flag = 1, the format of the data is 17A4 (Note that A4 format records the hexadecimal contents of memory with no format conversion. On cards, the EBCDIC punched card code for the hexadecimal is punched with one byte represented by each column). There is a maximum of two cards per class for this format. If the flag = 0, the format of the data is 5E14.7. With this format, there will be more record type 6's since there will be only five data values per record rather than 17. There is a maximum of six cards per class with this format.

The data elements for a set of record type 6's for one class are arranged so that element (i) is the mean of data values for the i'th channel used.

RECORD TYPE 7

There is one set of record type 7's for each training class. Each set contains the covariance matrix for one class. The beginning of each class is on a new record. Each record has the form 'CV' in columns 1 and 2 and data in columns 3-72. Columns 73-80 contain the sequence number. These records can have two formats for the data, identical to those for record type 6. The data for the set of records for one class is arranged so that the i'th data element is the i'th element in the covariance matrix for the class. The lower traingular portion only of the covariance matrix is recorded (the matrix being symmetric) and the elements are in the order c_{11}, c_{21}, c_{22}, c_{31}, c_{32}, etc.

RECORD TYPE 8

There is one record type 8. This record is fixed in format.

Column	Content
1-3	'EOS'
4-15	blank
16-59	'***** LAST CARD OF STATISTICS DECK *****
60-72	blank
73-80	Sequence number

HISTOGRAM FILE (DSRN 2)

The Histogram File is produced by the Histogram and Pictureprint functions and may be input to those same three functions and the Graphhistogram function. The file is always produced on disk and may, optionally, be punched onto cards. While the data contained on the two media are similar, the formats are not the same. GRAPHHISTOGRAM and HISTOGRAM use only the disk version whereas the other two functions will accept either format.

Histogram File on Cards

The Histogram Deck contains 11 different types of records (card images). A given Histogram Deck has its total number of cards determined by how many channels were histogrammed. There are 10 cards (1 of each record type 1 through 10) for each channel histogrammed plus one type 11 record at the end signifying the end of the deck. Therefore

Number of cards = (NCR*10) +1

where NCR is the number of channels requested.

The Histogram Deck is punched by modules HISTD and PIC1. PIC1 also reads the deck, checking fore sequence errors. Each card has a sequence number in columns 73-80. The sequence number = 10*channel number + (record type -1). The last card of the deck has a sequence number one greater than the one preceding it. The format size and content of each record type follows:

RECORD TYPE 1

This record identifies the data that was histogrammed.

Columns	Format	Description
1-16	Alpha	'LARS HISTO RUN('
17-24	I8	Run number for data histogrammed
25-31	Alpha	'),LINE('
32-36	I5	First line number for data histogrammed
37	Alpha	','
38-42	I5	Last line number for data histogrammed
43	Alpha	','
44-48	I5	Line interval for data histogrammed
49-54	Alpha	'),COL('
55-58	I4	First column number for data histogrammed
59	Alpha	','
60-63	I4	Last column number for data histogrammed
64	Alpha	','
65-67	I3	Column interval for data histogrammed
68	Alpha	')'
73-80	I8	Sequence Number

RECORD TYPE 2

All entries are fixed point except those indicated otherwise in the description field.

Columns	Format	Description
1-4	A4	Lower limit of spectral band in micrometers (floating point)
5-8	A4	Upper limit of spectral band in micrometers (floating point)
9-10	A2	Number of histograms accumulated (maximum is 20)
11-14	A4	Number of samples histogrammed
15-18	A4	Lower limit of histogram (floating point)
19-22	A4	Size of each histogram bin (floating point)
23-26	A4	Number of lowest bin used
27-30	A4	Number of highest bin used
31-70	20A2	Calibration code for each block histogrammed (maximum of 20 codes)
73-80	I8	Sequence number

RECORD TYPE 3

Columns	Format	Description
1-4	A4	Mean value of histogram (floating point)
5-8	A4	Standard deviation (floating point)
9-72	A2	First 32 elements (out of 100) of histogram results array HISTA
73-80	I8	Sequence number

RECORD TYPE 4

Columns	Format	Description
1-66	A2	Next 34 elements (out of 100) of histogram results array HISTA
67-72		blank
73-80	I8	Sequence number

RECORD TYPE 5

Columns	Format	Description
1-66	A2	Last 34 elements (out of 100) of histogram results array HISTA
67-72		blank
73-80	I8	Sequence number

RECORD TYPE 6

Columns	Format	Description
1-72	A4	First 18 elements (out of 20) of an array for storing run number of accumulated histograms
73-80	I8	Sequence number

RECORD TYPE 7

Columns	Format	Description
1-60	A2	First 10 segments of a 20 segment array where each segment contains 3 elements: starting and ending line numbers and the line interval for accumulated histograms
61-68	A4	Last 2 elements of the array for storing run numbers of accumulated histograms
69-72		blank
73-80	I8	Sequence number

RECORD TYPE 8

Columns	Format	Description
1-60	A2	Last 10 segments of a 20 segment array where each segment contains 3 elements: starting and ending line number and the line interval for accumulated histograms
61-72		blank
73-80	I8	Sequence number

RECORD TYPE 9

Columns	Format	Description
1-60	A2	First 10 segments of a 20 segment array where each segment contains 3 elements: starting and ending column numbers and the column interval for accumulated histograms.
61-72		blank
73-80	I8	Sequence number

RECORD TYPE 10

Columns	Format	Description
1-60	A2	Last 10 segments of a 20 segment array where each segment contains 3 elements: starting and ending column numbers and the column interval for accumulated histograms
61-72		blank
73-80	I8	Sequence number

RECORD TYPE 11

Columns	Format	Description
1-3	Alpha	'EOH'
16-63	Alpha	'***** LAST DATA CARD OF HISTOGRAM DECK ***** '
64-72		blank
73-80	I8	Sequence number

Histogram Disk File

The direct access disk file HISTO DATA is used to store histogram information on the temporary disk (D-DISK). There are 61 records of 360 bytes each. LARSMN contains the following statements:

```
DEFINE FILE 2(61,360,L,POINT)
```

Global COMMON(GLOCOM) contains the symbolic DSRN, HDATA, which is set to 2, and the record pointer POINT.

There are three types of records. The first 60 records alternate between a Type 1 and a Type 2. The 61st record is a Type 3 record (a dummy record) which when written will force FORTRAN to format the file for direct access if it has not been formatted previously in the current terminal session. This prevents errors if a module tries to read the file when it does not exist. A Type 1 with a Type 2 record contains the data pertinent to the histogram for a single channel; thus up to 30 channels may be histogrammed. Records 1 and 2 pertain to channel 1, records 3 and 4 pertain to channel 2, etc.

HISTO DATA is initialized (the dummy record 61 written) by PICRDR, and HISTD. The file is written and read by HISTD and PIC1 and read by GRHIST.

The format, size, and content of each of the three record types is described below.

RECORD TYPE 1

Bytes	Format	Size	Description
1-4	R*4	1 word	Lower limit of spectral band in micrometers
5-8	R*4	1 word	Upper limit of spectral band in micrometers.
9-10	I*2	1 halfword	Number of histograms accumulated
11-14	I*4	1 word	Number of samples histogrammed
15-18	R*4	1 word	Lower limit of histogram
19-22	R*4	1 word	Size of the histogram bin
23-26	I*4	1 word	Number of lowest bin used
27-30	I*4	1 word	Number of highest bin used
31-34	R*4	1 word	Mean value of histogram
35-38	R*4	1 word	Standard deviation
39-238	I*2	100 half words	Histogram results array (HISTA)

RECORD TYPE 2

Bytes	Format	Size	Description
1-80	I*4	20 words	Array for storing run numbers for up to 20 accumulated histograms
81-200	I*2	60 half words	Array for storing the beginning and ending line numbers and the line interval for up to 20 accumulated histograms
201-320	I*2	60 half words	Array for storing the beginning and ending column numbers and the column interval for up to 20 accumulated histograms
321-360	I*2	20 half words	Array for storing the calibration codes used for up to 20 accumulated histograms

RECORD TYPE 3

Bytes	Format	Size	Description
1-8	I*4	2 words	Zeroes(Null record)

CLUSTER SCRATCH FILE (DSRN 3)

The Cluster Scratch File is used to store the definition of the areas to be clustered. The format of all records on the file is the same. The file is written and read using FORTRAN unformatted I/O statements. The record format (RECFM) is variable spanned, unblocked, and the blocksize is 800 bytes. Eight bytes of control information is placed at the beginning of each of these records. (This record format is described in detail in IBM reference manual GC28-6817-2, S/360 Operating System Fortran IV (G&H) Programmer's Guide, pp69-70 "Unformatted Control".) The file is used only by the Cluster function.

The format of the records is described below. Each record contains 19 fullwords. Seventeen of these are used and the last two contain unpredictable data. The reason for the existence of the last two words is to use a generalized subroutine (RDFLDS) to write the file, and it writes 19 word records. Each record has the following form:

Word	Format	Content
1	I*4	Run number
2-3	Alpha	A description taken from columns 11-18 of a fixed form Field Description Card.

4	I*4	First line number for the area
5	I*4	Last line number for the area
6	I*4	Line interval
7	I*4	First column number for the area
8	I*4	Last column number for the area
9	I*4	Column interval
10-17	Alpha	Other information taken from columns 51-80 of a fixed form Field Description Card.
18-19	Alpha	Unpredictable

The file is created with one record for each area to be clustered. At the time of creation, the records are in the order in which the field description cards describing them were input. The records are written by RDFLDS in a call from CLURDR. FIXFLD then rewinds the file and reads all records into main storage. The file is rewound again and the records are written onto the file in order of increasing run and line number. At this time, certain field description values may also be changed to reflect changes required in order to fit all data into main storage. FIXFLD then rewinds and reads the file. CLUPRO rewinds and reads the file several times.

PRINTRESULTS SCRATCH FILE (DSRN 4)

The Printresults scratch file is a disk resident file (PRESULT SCRATCH) used exclusively by the Printresults function. The file is used to generate multiple copies of Printresults printed output. The format of the file is printer image (RECFM = F and record length of 133) with the first byte of each record containing carriage control information. The records are duplicates of those written to the printer except that a record of 'EOF' is used to separate sections of the file.

The file is first used to save the Printresults map for printing multiple copies. After the multiple copies of the map have been printed, the file is used to save the performance tables for printing multiple copies of them. The file is read by DISPLY when multiple copies of the map are printed and by DISPY2 when multiple copies of the performance tables are printed.

The description below indicates the contents of the file both when it contains the map and when it contains the performance tables. Detailed record formats are not given since these are identical to the printed output. In parenthesis after each item is the name of the subroutine writing it. Contents of file when it contains the map:

1. Heading including standard LARSFRIS heading plus the classification ID heading and list of channels and classes. Also lines indicating the outline symbols to be used. (DISPY1)
2. A record with the characters 'EOF'. (DISPY1)
3. The column number header. (DISPY1)
4. A record with the characters 'EOF'. (DISPY1)
5. The map. (DISPLY)
6. A record with the characters 'EOF'. (DISPY1)
7. The number of points displayed. (DISPY1)
8. A record with the caracters 'EOF'. (DISPY1)

Contents of the file when it contains the performance tables:

1. All performance tables including headings. (DISPY2)
2. A record with the characters 'EOF'. (DISPY2)

ERROR MESSAGE FILE (DSRN 8)

The error message file is resident on the system M-disk, as file ERROR MESSAGE. The file contains text for error messages, one message text per record. The records are card image form (RECFM = F and record length of 80). The format of each record is:

Columns	Format	Description
1-5	I5	Error number
6-77	Alpha	Text of error message
78-79		blank
80	Alpha	Contains 'N' if the message is to be typed only and not printed. Otherwise blank which means the message should be typed and printed.

This file is maintained by the system programmer. It is used by the subroutine ERPRNT which searches the file for the appropriate error number and then uses the text.

RUNTABLE FILE (DSRN 9)

The Runtable is a permanent disk file called RUNTABLE FILE residing on the O-disk. It is maintained by the system programmer. This file contains the ID records for all Multispectral Image Storage Tape runs that are "cataloged" by LARSFRIS. The first part of the file contains a directory (by run number) for accessing the ID records. The Runtable is a FORTRAN direct access data set. This means that it has fixed length records (RECFM = F). The record length is 800 bytes. The Runtable is created by the system programmer and is read by GADRUN and RUNSUP.

Each of the ID records is an exact duplicate of the ID record on the Multispectral Image Storage Tape (for its format, see the description of this file in this section of the System Manual). The ten directory records are each divided into 133 six byte entries (leaving the last 2 bytes unused), each of which identifies a single run. The last valid entry is followed by an entry with a run number of zero.

Bytes	Format	Description
1-4	I*4	Run number
5-6	I*2	Record number in Runtable that contains the ID record for the run. This record number can be used in a FORTRAN direct access read statement to read the requested ID record.

TRAINING AND TEST FIELDS SCRATCH FILE (DSRN 10)

This file is used by PRINTRESULTS and SAMPLECLASSIFY to store field description information. In PRINTRESULTS the file may contain descriptions of both the training fields and the test fields. The training fields are those obtained from the Statistics File, and the test fields are input by the user in his input deck on Field Description Cards. The training field information is written on the file first, followed by a single record containing only "EOF", followed by the test fields and another "EOF".

In SAMPLECLASSIFY the file contains only the test field descriptions that were input by the user on Field Description Cards. These are followed on the file by a single record containing only "EOF".

The file is created by RDFLDS (with the portion containing training fields created by the entry point RDTRN). In PRINTRESULTS, DISPY2 reads the file and in SAMPLECLASSIFY, SMCLS2 reads it.

The file is accessed via FORTRAN unformatted I/O and the record format is RECFM = VS with a block size of 800 bytes. Eight bytes of control information are placed at the beginning of each of

these records. The format of each record is given below:

Record Format:

Each record is 19 fullwords long and contains the following:

Word	Format	Contents
1	I*4	Multispectral Image Storage Tape run number for the field.
2-3	Alpha	The field designation furnished by the user.
4	I*4	The beginning line number for the field.
5	I*4	The last line number for the field.
6	I*4	The line interval.
7	I*4	The beginning column number for the field.
8	I*4	The last column number for the field.
9	I*4	The column interval.
10-17	Alpha	Other information taken from columns 51-80 of a fixed form Field Description Card.

18	I*4	Contains the group number for the field. For training fields this is the group number that was assigned in Printresults (see the function description for more information). For test fields it is the number on the 'TEST' card in the input deck.
19	I*4	Contains valid information only for training fields, in which case it contains the pooled class number for the field.

CLASSIFICATION RESULTS FILE (DSRN 11,12,14,17)

The Classification Results File is produced by the Classify-points function and is the primary input to the Printresults, Copyresults, Listresults, Compareresults, Smoothresults and Punchstatistics functions. The file may be output on either tape or on disk, however, the latter two functions will accept it for input only if it is on tape.

When the file is produced on tape, special support is provided to allow more than one results file to be placed on a single tape reel. The last file on the tape is followed by a special "marker" file, which contains only a single type 1 record (see the record types below). A special file type code is then used to indicate that this is the last file on the tape and that it is not a true data file.

The first two parts of this description describe the file usage and the format of the file under normal conditions. The last part describes the temporary records that are created when the SUSPEND command is issued. These temporary records are always destroyed when processing of the "suspended" file is restarted.

File Usage:

The Classification Results File is created by the Classifypoints function. Copies of the file can be created with the Copyresults

function. Below is a list of program modules that create records on the results file. These are all modules in Classifypoints. All records of COPYRESULTS output file are created by the module COPY.

Record Type	Module writing
1	MMTAPE writes this record if the file is on tape and is being initialized. Otherwise, CLAINT writes it. The form of this record with filetype = 1 is written by CLSFY2 (or by CLSFY1 in the case of an error termination).
2	CLAINT
3	CLAINT
4	CLSFY1
5	CLSFY2
6	CONTEX
7	CONTEX
8	CLSFY2

The Listresults, Punchstatistics, Copyresults, Compareresults, Smoothresults and Printresults functions all read the results file on tape. Only COPYRESULTS, COMPARERESULTS, SMOOTHRESULTS and PRINTRESULTS can read the results file on disk. All the results functions leave the results tape positioned at the beginning of the next file after the one they processed. The results file on disk is left positioned at the end.

All functions that read the results file use MMTAPE to mount and position it if the file is on tape. MMTAPE reads the first record of the file and then repositions it back at the beginning of the file.

For Listresults, Punchstatistics and Copyresults, all records are read by the module COPY and RESCOP initially reads the first two records.

In Printresults, record types 1 and 2 are read by PRIINT. Record type 3's are read via a CALL to RDTRN. Record type 4 is read by STATS and by DISPY1. DISPY1 may also read over record types 6 and 7 in order to get to the next record type 5. Record types 6 and 7 are read by DISPLY. Record type 8 is read by DISPY1.

The File Format

The Classification Results File contains eight different types of records. The format, size, and content of each entry in each of these records, and the number of records of each type that occur in a single file, is described below. All records are written using Fortran unformatted I/O, a variable spanned (RECFM = VS) record form and a block size of 1500. This record format is described in detail in IBM reference manual GC28-6817-3, Fortran IV (G and H) Programming Guide, pp 69-70 ("Unformatted Control"). The records are unblocked. Briefly,

if the logical record is 1492 bytes or less, the physical record consists of 8 bytes of control information plus the logical record. If the logical record is 1493 bytes or more, it is spanned onto additional physical records each beginning with 8 bytes of control information. The final physical record written may be less than 1,500 bytes.

Each record begins with a two word prefix which precedes the data described below. The first word contains the record type as a fullword integer. The second word is set to zero for record types 1-4 and record type 8. For record types 5-7, it contains the area number as a fullword integer. The first area classified in the file is area 1, the second area classified is 2 and so forth. The record sizes shown below do not reflect this additional 8-byte prefix, which should be added to determine record size.

RECORD TYPE 1

This record contains identification information for the file. Each results file has one record of type 1 and it is always 12 fullwords (48 bytes) long.

Format	Size	Description
I*4	1 fullword	Tape Number (zero if file is on disk) A scratch tape is number 0.

I*4	1 fullword	File Number (zero if file is on disk)
I*4	1 fullword	LARSYS Version Number (currently the value is 3)
I*4	1 fullword	Filetype (0 for a results file, 1 for a restart file, and -1 for a file containing only record type 1)
I*4	1 fullword	Serial number in the form "ydddssssss", where y = the last digit of the year, ddd = the day of the year, and sssss = the number of seconds since midnight.
I*4	1 fullword	Level flag (0 for LARSYS Version 3, 1 for modified classification results file)
I*4	6 fullwords	Not currently defined though may be in the future. All six words are now zeroes.

RECORD TYPE 2

Each file has one variable length record of type 2. It contains a number of entries relating to the channels, classes and pools that were used in the classification and varies in size with the number of each of these that were used.

Format	Size	Description
I*4	1 fullword	Number of classes used in calculating statistics before grouping into pools.
I*4	1 fullword	Number of channels used in the classification.
I*4	1 fullword	Number of training fields used in the classification
I*4	1 fullword	Number of classification pools

The size of the next segment varies with the number of channels (represented by a ch in the size column).

I*2	(1 x ch) halfwords	A vector of the channel numbers for all channels used in the classification. Channel numbers are in ascending sequence, and each is stored as a half-word integer.
I*2	(1 x ch) halfwords	A vector containing the calibration codes for the channels used in the classification. (See LARS Information Note 071069). Same format as above.
R*4	(1 x ch) fullwords	A vector containing the upper wavelength band limits of all channels (in micrometers) used in the classification.
R*4	(1 x ch) fullwords	A vector containing the lower wavelength band limits of all channels (in micrometers) used in the classification.

The size of the next segment varies with the number of pools and the number of classes (represented by po and cl in the size column).

Alpha	(8 x po) bytes	A list of the eight byte names for all of the pools, in ascending order by pool number.
I*2	(2 x po) halfwords	Pool Pointer Matrix (POLPTR). A 2 by j matrix where j = the number of pools. POLPTR (1,j) = the number of classes in pool j, and POLPTR (2,j) = the location of the first class for pool j in the Pool Stack Vector (POLSTK) below.

I*2	(1 x c1) halfwords	Pool Stack Vector (POLSTK). This is a vector containing the class numbers of all classes used in the statistics deck grouped by classification pool.
R*4	(1*po) fullwords	Weight vector (PROB). This vector contains the weights assigned to each pool in the classification.
Alpha	20 bytes	The date the classification was performed in EBCDIC. For example Sept. 11, 1972 would be represented as "SEPT,11b1972bbbbbbbb".

RECORD TYPE 3

These records provide a copy of the Statistics File that was used in the classification run. There is one logical (and physical) record of type 3 created for each "card" in the Statistics File. The record consists of the exact 80-column image of the card. See the description of the Statistics File in this section for more details on its organization and content.

RECORD TYPE 4

This record contains the covariance matrices and the mean vectors for the reflectance values of the channels for each of the classification pools. The record is of variable length, varying with the number of pools and number of channels used in the classification. (Represented by po and ch in the size column).

Format	Size	Description
The covariance matrices for all of the pools are written first:		
R*4	$\frac{(ch^2+ch)}{2}$ po fullwords	The Covariance Matrix for each classification pool is placed in the record in ascending pool number sequence. Only the lower triangular "half" of the symmetrical matrix is stored and the individual elements (values) are stored as floating point numbers in "column by row" sequence. Hence each element C_{ij} of each matrix is stored according to the sequence; C_{11}, C_{21}, C_{22}, C_{31}, C_{32}, C_{33}, C_{41}, etc.
The mean vectors for all of the pools follow the covariance matrices:		
R*4	(1 x ch)po fullwords	The Mean Vector for each classification pool is then placed in the record in ascending pool number sequence. Each individual vector has the mean value for each channel used in the classification ordered in ascending channel number sequence.

RECORD TYPE 5

This is the area identification record. There is one such record at the beginning of the results records for each area considered in the classification run. It is followed by the series of results records (record type 6) which contain the classification results of each line in the area; and by a single "end-of-the-area"

record (record type 7) following the last results record. Record Type 5 is fixed length and is always 309 fullwords long.

Format	Size	Description
I*4	1 fullword	The number of points classified in each line of record type 6.
I*4	1 fullword	The number of lines classified in the area.
I*4	17 fullwords	The INFO array used in Classify-points (in CLACOM) INFO(1) = run number INFO(2) = field designation (first four characters) INFO(3) = field designation (last four characters) INFO(4) = line number of first line of field INFO(5) = line number of last line of field INFO(6) = line interval INFO(7) = column number of first column INFO(8) = column number of last column INFO(9) = column interval INFO(10-11) = class name - eight characters INFO(12-17) = user information that was contained in card columns 59-80 of the Field Description Card (fixed form) that defined this area. The last two characters are blank.

I*4	200 fullwords	The Multispectral Image Storage Tape Identification Record for this area.
R*4	90 fullwords	The Calibration Set Array. This is the same as the array CSET3 from CLACOM. The array is dimensioned three by thirty, providing an element for each possible calibration value for each possible channel. Entries that are not specified by the user are set to the values contained on the Multispectral Image Storage Tape ID record. The elements are ordered such that: CSET3(1,J)=the value of C0 for channel J CSET3(2,J)=the value of C1 for channel J CSET3(3,J)=the value of C2 for channel J The array is stored on the file row by column, i.e., CSET3(1,1), CSET3(2,1), CSET3(3,1), CSET3(1,2)...etc.

RECORD TYPE 6

There is one record type 6 for each line in each area that was classified. Its length will vary with the number of points that were classified in the line. (represented by pt in the size column).

Format	Size	Description
I*4	1 fullword	The line number from the Multispectral Image Storage Tape.

Following the line number there is a series of entries, one for each point in the line.

I*2	(1 x pt) halfwords	The entry contains two items of data. The first byte of the halfword contains a likelihood code which represents the probability of the point belonging to the class in which it was classified. The larger the code, the greater the probability of correct classification. The code is an integer ranging from 1 to 234. See the Classify-points algorithm description in the LARSFRIS User's Manual for information on how the code is calculated. The second byte of the halfword contains an integer identifying the number of the pooled class in which the point was classified.

RECORD TYPE 7

At the end of each area that was classified is a single type 7 record to identify the end of the area. It is the same length as the preceeding type 6 record and is distinguished from it by the fact that the line number (first fullword) is set to zero. Only the first fullword of the record is currently used.

RECORD TYPE 8

At the end of the results for the last area that was classified in the run (following its type 7 record) is a single type 8 record. This record is the same length as the type 5 record and is distinguished from it by the fact that the "number of points" field (first fullword) is set to zero. Only the first fullword of the record is currently used.

The file is then terminated by a file mark.

Implementation of the Suspend Command

When the Suspend command is issued, CLASSIFYPOINTS terminates processing in such a way that the classification may later be restarted at the point at which it was suspended. To enable the function to do this, special restart information must be appended to the Classification Results File. If the Results File is on tape, CLASSIFYPOINTS simply writes a special "restart file" (containing four types of records) directly after the suspended classification results. If the results file has been initially written to disk, it is first copied to a tape designated by the user and then the same restart file is written behind it. Prior to this disk-to-tape copy some vital information from main storage is temporarily saved on the disk file.

There is no difference in the final suspended Classification Results File, regardless of whether it was initially written to disk or to tape. When the classification is later restarted, however, it can only use the tape for output results.

The Restart File

When CLSFY2 detects the Suspend Command (or upon copying the file to tape if it was on disk), it completes the results file on tape by writing the normal record types 7 and 8 followed by a tape mark. It then writes the special restart file, which contains one record each of the first three types described below and a

variable number of the fourth type. When a classification is later restarted, this restart file is read in by CLASUP and CLSFY2.

Only record type 1 of the restart file contains the two word prefix that is common to all of the normal Classification Results File records.

Restart Record Type 1

This record is identical to the normal Classification Results File record type 1 except that the filetype is set to one rather than zero. The record is written by CLSFY2 and read by CLASUP.

Restart Record Type 2

There is one such record. It is written by CLSFY2 and read by CLASUP. This record contains a core image dump of the common area CLACOM (Refer to the program module documentation) in the Classifypoints function plus three additonal words containing (as fullword integers):

1. The first element of the array ARRAY in GLOCOM to be used for storing covariance matrices (base address of array COVMTX in CLSFY2).
2. The first element of the array ARRAY to be used for storing mean vectors (base address of array AVEMTX in CLSFY2).

3. The first element of the array ARRAY that is free for use for buffers (base address of array RDATA in CLSFY2).

Restart Record Type 3

There is one restart record type 3. It is written and read by CLSFY2. (Refer to the program module documentation for GLOCOM and CLSFY2 for a precise definition of variables.) It contains the following data:

1. The array HEAD from GLOCOM.
2. The array RUNTAB from GLOCOM.
3. The variable IMARK from GLOCOM.
4. The array AVEMTX from CLSFY2 (the array of mean vectors for the classes).
5. The array COVMTX from CLSFY2 (the array of inverse covariance matrices for the classes).
6. The variable BLOCK from CLSFY2. This describes the current position in the area being classified.
7. The variable GADLIN from CLSFY2 which contains the number of lines in the area being classified which could not be classified because data from the Multispectral Image Storage Tape was not available.
8. The variable JLINER from CLSFY2 which contains the number of lines actually classified in the area being classified at the time of suspension. This is used in restarting to position the tape.

9. The variable IAREA from CLSFY2 which contains the area number (the second word of the prefix of record type 6) of the area being classified at the time of suspension.

Restart Record Type 4

There is one such record for each area remaining to be classified. Each record contains the INFO array for the area to be classified. Thus the form of these records is the same as the file CLASSIFY SCRATCH. The first of these records describes the unclassified part of the area which was being classified at the time of suspension. The remaining type 4 records are generated by reading the records from the CLASSIFY SCRATCH file and writing them to the restart file.

Special Disk Record

When SUSPEND is detected and the results are on disk, they are first transferred to a tape and then the normal restart file is written on the tape. This transfer requires reading the data into main storage and then writing it onto tape. Doing so destroys the variables COVMTX and INFO in CLSFY2, which must be saved on the tape as part of the restart file. To save this information until after the results are copied, they are written on the disk immediately after the last results record. The disk file is the 'rewound' and copied to the tape. The extra record is then read back into main storage for incorporation into the restart file. It contains:

1. The current value of the prefix which indicates record type 7 and the area number.
2. The array COVMTX from CLSFY2. (Inverse covariance matrices)
3. The vector INFO from CLACOM which describes the area currently being classified.

SEPARABILITY SCRATCH FILE (DSRN's 11,18)

The Separability scratch file can reside on disk or tape. If the D-disk has sufficient space for the file, disk is used; otherwise, tape is used. The disk file is SEPAR SCRATCH. The file is used only by the Separability function, where it is created and read once for each COMBINATIONS request to the Separability function and reread for each new set of options typed in by the user. The file is created by subroutine DIVRG2 and is read by subroutine GETDAT.

All records are written using FORTRAN unformatted I/O with a record format of variable spanned (RECFM = VS), thus each record also contains 8 bytes of control information. If the file is on disk the blocksize is 800; if on tape, the blocksize is 1500. With variable spanned records, this blocksize has no effect on the source program I/O statements. (The record format is described in detail in IBM reference manual GC28-6817-2, S/360 Operating System, Fortran IV (G&H) Programmer's Guide, pp 69-70 "Unformatted Control").

As noted above the file is created once for each COMBINATIONS request. The file is rewound before each new request. Every record in the file has the same format and length. The number of records is $\frac{k!}{r!\,(k-r)!} + 1$ where k is the total number of

channels available and r is the number of channels in the COMBINATIONS request. (i.e., the user has requested a separability study to find the best r out of k channels). This provides one record for each combination of r channels out of the possible k channels plus one trailer record. The format of an individual record is described below.

Format	Size	Description
R*4	1 fullword	The sum of the divergences for all class combinations for this channel combination. On the trailer record, this value is -1.0 which serves as a flag indicating the trailer record.
I*2	r halfwords	Channel combination used for this record. The ith element in this vector is the channel number of the i'th channel used for divergence calculations recorded in this record. On the trailer record, this vector is the same as for the last normal record.
R*4	$\frac{(po^2-po)}{2}$ fullwords (po=number of pools (classes) considered)	The vector of divergences for all class combinations computed using the set of channels listed earlier in this record. The order of these divergences is: divergence for class pairs 1-2,1-3,1-4,...1-po,2-3, 2-4,...2-po,3-4...3-po,... (po-1) -po. On the trailer record, this vector is the same as for the last normal record.

MULTISPECTRAL IMAGE STORAGE TAPE (DSRN 11,12)

The format of this file requires that each data run be completely identified, and that each data sample be stored as an eight-bit integer. This permits the sample data value stored on the tape to range between 0 and 255. The data samples and calibration measurements from all channels for each scan line, along with the line number, is stored in a data record. The linear calibration procedure used in LARSFRIS can be used to restore the tape data values back to the irradiance. (Refer to Appendix IV of the LARSFRIS User's Manual for details on calibration.)

The file is often positioned for use through the TAPOP module. GETRUN (which uses TAPOP) positions the tape to the correct file and always reads the identification information.

GADLIN, also using TAPOP, is then used to read the data record for each line and to calibrate the data. GETRUN and GADLIN are used by 11 of the 23 LARSFRIS Processing Functions.

There are three types of physical records on the file. They are:

1. ID record - 200 fullwords, fixed length
2. Data record - variable length
3. End-of-Tape record - 200 fullwords, fixed length

A single tape may contain one or more data runs, each of which consists of an ID record, multiple data records and a Tape Mark (a special record written by the I/O control software, which indicates end of file). After the last data run on the tape, a single End-of-Tape record and two Tape Marks are written.

Figure 5-1, on the following page, graphically illustrates the format of the file. At the bottom of the figure the overall tape containing several runs (files) is depicted. The format for the data of an entire run, a single line, and a single channel of a line are shown in three successive "exploded" views as the reader proceeds up the figure. Each "exploded" area is shaded on the figure. This figure should be referenced in conjunction with the descriptions of all of the individual fields. Note that the notation ID(n) is used on the figure to reference the nth word in the ID record.

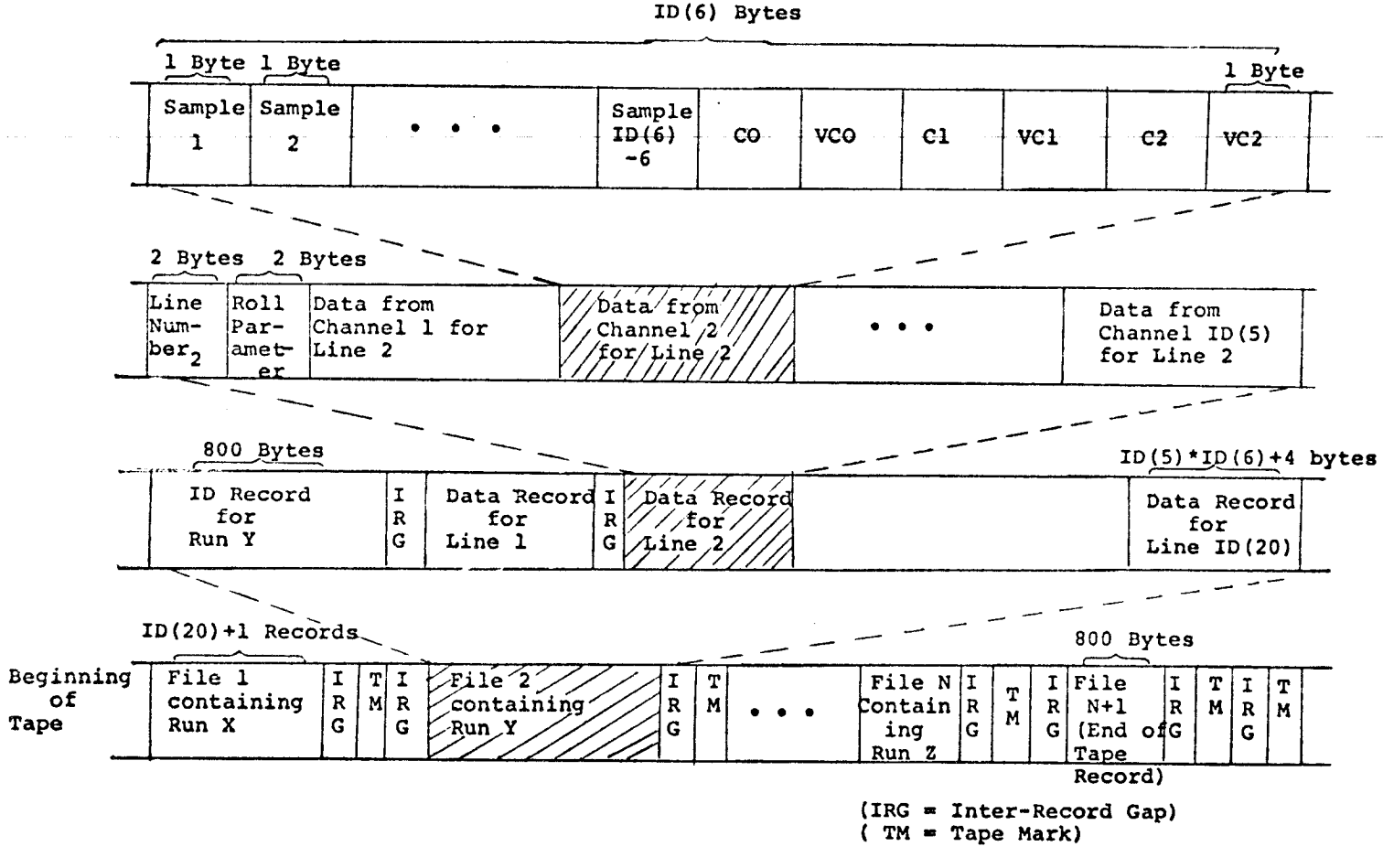


Figure 5.1 Multispectral Image Storage Tape Format

1. ID record (200 fullwords, fixed length)

Word	Format	Description
1	I*4	LARS Tape Number (e.g., 102, etc.)
2	I*4	Number of the file on this tape
3	I*4	Run number (8 digits aabbbbcc) aa - last 2 digits of the year bbbb - serial number for the year data was taken cc - uniqueness digits for runs which would otherwise have the same number
4	I*4	Continuation Code A value of 0 means the first line of data follows this ID record. A value of X means that the data following this ID record is a continuation of a flightline started on tape X.
5	I*4	Number of Data Channels (Spectral Bands) on tape (30 maximum)
6	I*4	Number of Data Samples per channel per line
7-10	Alpha	Flightline Identification (16 characters)
11	I*4	Month Data was Taken
12	I*4	Day Data was Taken
13	I*4	Year Data was Taken
14	Alpha	Time Data was Taken
15	I*4	Altitude of Aircraft
16	I*4	Ground Heading of Aircraft
17-19	Alpha	Date Data Run was Generated on this Tape (12 characters)

20	I*4	Number of Lines in this Run
21-50	I*4	All zero (to be defined later)
51	R*4	Lower Limit in Micrometers of the first Spectral Band on Tape
52	R*4	Upper Limit in Micrometers of the first Spectral Band on Tape
53	R*4	The suggested Value of "C0" calibration pulse for the first spectral band.
54	R*4	The suggested value of "C1" calibration pulse for the first spectral band
55	R*4	The suggested Value of "C2" calibration pulse for the first spectral band
56-200	R*4	Repeat of words 51-55 for the number of channels shown in word 5, in order of appearance in Data Records. NOTE: words 51-200 = 0.0 if Data Channels do not exist

2. Data Record

Each data record will contain one scan line of data from ID(5) (see ID Record) channels. The first halfword (2 bytes) of the record will be the line number. The second halfword (2 bytes) will be the roll parameter which is a number indicating relative position of the roll of the aircraft for this line of data. If the roll parameter is -32,767, the data for the given line does not exist. If

the roll parameter has not been calculated, it will be set to 32,767. The fifth byte will be the first sample from the first channel. The sixth byte will be the second sample from the first channel, and so on through ID(6) samples and ID(5) channels. A data record will be ID(5)* ID(6)+4 bytes long.

The data from each channel will be from the field of view of the scanner except the last six bytes. The last six are calibration data in the order of appearance.

1. C_0 "0" or dark level
2. VC_0 Variance of C_0
3. C_1 Calibration source C_1
4. VC_1 Variance of C_1
5. C_2 Calibration source C_2
6. VC_2 Variance of C_2

where C_i = Calibration value i and VC_i = calculated variance of calibration value i

During the reformatting process a record may be bad due to tape or other errors. When this happens, the data roll parameter and calibration points will all be set to zero. On good data records all data and calibration values will be in the range of 0 to 255 (bit form) with no sign included in the eight bits. A data value of 0 to 255 means that the

data point was cut off during the digitization process. Data values then range between 0 and 255 with 0 indicating low relative irradiance and 255 indicating high relative irradiance.

3. End-of-Tape Record

The End-of-Tape Record is very similar to the ID Record with 200 fullwords in the following format:

Word	Format	Description
1	I*4	LARS tape number
2	I*4	File number on this tape
3	I*4	Set equal to 0
4	I*4	Continuation Code A value of 0 means the end of data. A value of X means the data in the previous file is continued on tape X.
5-50	I*4	All zero (may be defined later)
51-200	R*4	0.0 (may be defined later)

TRANSFERDATA OUTPUT FILE(DSRN13)

The Transferdata function converts Multispectral Image Storage Tape data to a more easily read FORTRAN format in order to simplify the use of this data in other than the LARSFRIS programs. It can both punch the data into cards and write it on a magnetic tape. In either case, however, the record format is identical. The records are formatted in "card image", i.e., in 80 byte records. The three types of records for each module of data are described below.

Field Identification Record

One of these records is produced as the first record for each module of data that is copied. The record identifies and describes the field.

Card Columns	Format	Description
1-8	I8	Run number from the Multispectral Image Storage Tape
11-18	Alpha	Name of the data module.
21-25	I5	Line number for the first line of the module.
26-30	I5	Line number for the last line of the module.
31-35	I5	Line interval for the module.
36-40	I5	Sample number for the first sample on each line.

41-45	I5	Sample number for the last sample on each line
46-50	I5	Sample interval for the module
51-58	Alpha	Class name of the class to which the module has been assigned
59-80		Blank

Calibration Records

There is one calibration record for each channel that was requested. They immediately follow the Field Identification Record. The information in columns 7 through 54 is taken from the ID record on the Multispectral Image Storage Tape.

Card Columns	Format	Description
1-3	I3	Channel number
4-6	I3	Calibration code used to calibrate data
7-12	F6.2	Lower spectral band limit for this channel
13-18	F6.2	Upper spectral band limit for this channel
19-30	2PE12.2	Calibration value for C0
31-42	2PE12.2	Calibration value for C1
43-54	2PE12.2	Calibration value for C2
55-72		Blank
73-76	I4	Module number
77-80	I4	Card sequence number

Data Records

These records contain the calibrated data values in integer form. They follow the channel records. All the values requested for a field are punched consecutively, 24 to a record, in order by line, column, and channel in the following manner:

$$L_1C_1K_1, L_1C_1K_2, \ldots, L_1C_1K_n, L_1C_2K_1,$$
$$\ldots, L_1C_mK_n, L_2C_1K_1, 111, L_kC_mK_n$$

where L = line, C = column, K = channel, n = number of channels requested, m = samples per line and k = number of requested lines. Note that k·m is the number of samples per channel in the module (columns 64-69 on first card).

Card Columns	Format	Description
1-72	I3	24 calibrated data values
73-76	I4	Module number
77-80	I4	Card sequence number in module after Field Identification Record.

FIELD BOUNDARIES FILE (DSRN 19)

The Field Boundaries file is created and used by the Pictureprint function. It has a record format of VS (variable spanned), a blocksize of 800 bytes, and is stored on the D-disk as file FIELD BNDRIES. Pictureprint calls BONDSU to add records to the file when the BOUNDARY STORE option is used. If BOUNDARY DELETE is specified, the BONDSU calls the ERASE FORTRAN library subroutine to erase the file. PIC1 reads the file if BOUNDARY OUTLINE is specified. Before attempting to read they call STATE to be certain that the file exists. There is one record written onto the file for each field stored. Each record is 7 words in length (28 bytes). Note that when using variable spanned format, FORTRAN places eight bytes of control information in front of each record.

Each record has the form:

Word	Format	Description
1	I*4	Run number
2	I*4	First line number of field.
3	I*4	Last line number of field.
4	I*4	First sample number (column) in field.

5	I*4	Last sample number in field.
6	I*4	Flag indicating whether the field is a training or test field. A value of 0 indicates a training field and a value of 1 indicates a test field.
7	I*4	A sequence number starting at 1 and incremented by 1 (thus it is the record number).

The last record of the file contains 'EOF' in bytes 1-4 and a sequence number in the seventh word. Words 2-6 are undefined for this record.

CLASSIFYPOINTS SCRATCH FILE (DSRN 20)

The Classifypoints Scratch File is used to store field description information defining areas to be classified. The file is written and read by the Classifypoints function. The file is stored on disk as CLASSIFY SCRATCH. It is created by CLAINT and is read by CLSFY2.

The record format is variable spanned (RECFM = VS) with a block size of 800. This record format is described in detail in IBM reference manual GC28-6817-2, S/360 Operating System, Fortran IV(G&H), Programmers Guide, pp 69-70, "Unformatted Control". Each record has 8 bytes of control information followed by 17 fullwords describing the field. There is one record for each area to be classified. The layout of a record is given below:

Word	Format	Description
1	I*4	Run number
2-3	Alpha	Name of area taken from a fixed form Field Description Card. (Col. 11-18)
4	I*4	First line number of field
5	I*4	Last Line number of field
6	I*4	Line Interval
7	I*4	First column number of field

8	I*4	Last column number of field
9	I*4	Column interval
10-11	Alpha	A class name taken from columns 51-58 of the fixed form Field Description Card.
12-17	Alpha	Other information (comments and identification) taken from columns 59-80 of the fixed form field description card. The last two bytes of the record are always blanks since columns 59-80 can supply only 22 bytes.

5.3 LARSFRIS SYSTEM INFORMATION

System Documentation files are files that are created and/or maintained by the system programmer as part of the process of updating the LARSFRIS system. They constitute "data" for the operation of the LARSFRIS 'news', 'reference' and 'list' commands. See the program documentation for these routines for more information on the use of these files. The files are maintained on the LARSFRIS 19C system disk as CMS files having various filenames and filetypes of NEWS, REFERENC or INDEX. Each of them consists of 80 character card images (with the exception of RUNTABLE REFERENC, which consists of 120 character lines).

NEWS Files

The first card in any NEWS file is required to have 'REVISED' appearing in columns 2-8, followed by the revision date in the format 'MM/DD/YY' in columns 10-17. The remainder of the first card may be blank or may contain any other desired information. This revision date is the one that appears in the file LARSFRIS INDEX and is printed as a result of entering the 'list' command.

The remainder of any NEWS file content is arbitrary, and the file may be of variable length.

REFERENC Files

The first card of any REFERENC file is required to have a '1' in column 1 and the remainder of the card must have the same format as the first card of a NEWS file. In REFERENC files, the first column of each card is a carriage control position that does not appear on the printed output from the REFERENC command. Columns 2-80 contain arbitrary information, and the file may be of variable length.

The file ALL REFERENC is a suitably ordered concatenation of all the other REFERENC files (excluding RUNTABLE). Its first line has blanks in place of 'MM/DD/YY' in columns 10-17 and has the legend 'LARSYS VERSION 3 REFERENCE FILES' appearing centered on the remainder of the card. ALL REFERENC is created by the system maintenance routine CREFALL as a standard part of the LARSFRIS update procedure.

The file RUNTABLE REFERENC is created as a standard part of the LARSFRIS update procedure by the RTUPDT system maintenance routine. Each of the lines containing a run number must have the run number appear in columns 2-9 to enable the 'reference runtable nnnnnnnnn' command to work.

The File LARSFRIS INDEX

The file LARSFRIS INDEX is automatically created by the system maintenance routine CRINDEX as a standard part of the LARSFRIS

update procedure. Like the NEWS and REFERENC files, it carries the 'REVISED MM/DD/YY' legend in columns 2-17 of the first line. The lines in the file which carry information about the specific NEWS and REFERENCE files must have the filename in columns 50-57, the number of lines in the file in columns 61-64, and the revision date in columns 69-76. Other lines (headings, etc.) may be arbitrarily interspersed.

PRIORITY NEWS

This file is used to advise the user of important news. It only exists when there is news of importance to pass to the user. When it does exist, it is resident on the N-disk and has 80-byte card image records. When EXCOMD is entered, it will print the file automatically.

5.4 OTHER LARSFRIS FILES

ACCT BATCH

This file contains one 80-byte record. It is created by BATRD on the A-disk to store the userid, account number, and starting time of the current job. BATEND reads the file when the job is completed. The format of the record is:

Byte	Contents
1-12	Starting time for the job in EBCDIC characters using the format MMDDYY.
13-16	Milliseconds of CPU time used by the batch virtual machine at the time the current job was started. Format is a fullword integer.
17-24	The userid.
25-32	The user's account number.
33-40	The time limit being used for this job. This is in character format and is in seconds. The number is right justified and passed with zeroes.

BATCH STOP

This file exists on the A-disk of the batch machine. It has 80-byte card image records. All records contain the characters 'KILL'. The batch machine normally uses the FILEDEF command to FILEDEF unit 15 (usually the keyboard) to BATCH STOP. This means that any program attempting to read from the terminal on the batch machine will read the characters 'KILL'. This allows LARSFRIS to terminate the function rather than log out the

batch machine (which is disconnected). The file has 10 of these records in case one function reads a record and then another read is made without rewinding the file.

EOSTAT INIT

This file is on the M-disk. It contains one record containing the characters 'EOS'. When EXCOMD is entered (and when BPROFILE begins), just after the D-disk is cleared, this file is copied onto the D-disk and into the file STATS DATA. Thus, if LARSFRIS attempts to read statistics from disk and no such file has been created during the terminal session, LARSFRIS will read 'EOS' and can give the user an appropriate error message.

SAVED HISTDECK

This file is on the user's A-disk. Its content and form are identical to the file HISTO DATA (DSRN 2). It is created by the HISTDECK SAVE command and read by the HISTDECK USE command.

SAVED STATDECK

This file is on the user's A-disk. It's content and form are identical to the Statistics file (DSRN 1). It is created by the STATDECK SAVE command and it is read by the STATDECK USE command.

BATCH DATA

This file is a temporary file on the D-disk. The BATCH command reads a data deck from the virtual reader into this file. The file is then placed in the reader of the batch machine and offline printed. The BATCH command finishes by erasing this file.

APPENDIX I

LARSFRIS SYSTEM PROGRAM MODULES

Module Name	LARS Program Abstract Number	Module Name	LARS Program Abstract Number
ARBASE	301	COSPEC	334
BATRD	055	COVIN	158
BCDVAL	001	CPFUNC	003
BIPCLA	404	CTLWRD	004
BIPDIV	406	DECCLS	124
BIPELL	403	DISPLY	241
BIPLTR	402	DISPY1	242
BIPMEN	405	DISPY2	243
BIPRDR	401	DIVERG	303
BIPSUP	400	DIVPRT	304
BISPLT	335	DIVRG1	305
BLOAD	100	DIVRG2	306
BONDSU	122	ERMNAM	101
BPROFILE	056	ERPRNT	102
BSTCHK	302	ERRINT	103
CCHIS	181	EXCOMD	057
CGROUP	179	FILBUF	366
CHANEL	002	FIXFLD	176
CHANGE	384	FLDBN	244
CHASUP	271	FLDBOR	231
CHTAPE	385	FLDCOV	323
CLACOM	151	GADLIN	005
CLAINT	152	GADRUN	006
CLARDR	153	GATHER	371
CLASS	154	GBIPSUP	083
CLASUP	150	GCLASUP	071
CLSCHK	123	GCLUSUP	072
CLSDEC	142	GCOL	191
CLSFY1	155	GCOMSUP	084
CLSFY2	156	GENALL	054
CLSHIS	321	GENALLDV	089
CLSSPC	322	GETDAT	307
CLUCOM	171	GETINV	308
CLUMP	173	GETSHW	309
CLUMP1	174	GGRHSUP	073
CLUPRO	175	GHISSUP	074
CLURDR	172	GLARSMN	069
CLUSUP	170	GLIN	192
CMGRAY	218	GLOCOM	104
CMSFNC	035	GMERSUP	085
COMCOM	381	GPICSUP	076
COMPAR	383	GPRISUP	077
COMRDR	382	GRESSUP	078
COMSUP	380	GRATSUP	086
CONTEX	157	GRHIST	125
COPY	261	GRHRDR	193

Module Name	LARS Program Abstract Number	Module Name	LARS Program Abstract Number
GRHSUP	190	PICRDR	233
GRPSCN	126	PICSUP	230
GRUNSUP	079	PIC1	234
GSAMSUP	080	POLMER	340
GSECSUP	087	POLSET	341
GSEPSUP	081	PRACRE	252
GSMOSUP	088	PRICOM	246
GSTASUP	082	PRIINT	247
GTDATE	007	PRIRDR	248
GTSERL	159	PRISUP	240
HEADER	127	PROCES	106
HISRDR	201	PRTHED	249
HISSUP	200	PRTPCT	250
HISTD	128	RATCOM	391
IBCD	025	RATIO	394
IDNAME	008	RATRDR	392
INVPNT	180	RATSUP	390
LAREAD	129	RATTAP	393
LARSMN	105	RDFLDS	131
LARS12	009	READMX	354
LEARN	324	REDFLD	337
LINRDR	194	REDSAV	133
LIST	058	REDSTA	134
LLARSMN	070	REFERENC	061
LOGICOPS	014	RESCOM	262
LOGLIK	368	RESCOP	264
MCONTX	161	RESRDR	263
MDIST	177	RESSUP	260
MERCOM	331	RTBSUP	107
MERINT	332	RTMAIN	028
MERRDR	333	RUNCOM	272
MERSTT	338	RUNERR	120
MERSUP	330	RUNLS	062
MEXPAN	339	RUNSUP	270
MMTAPE	130	SAMCLS	367
MOUNT	010	SAMCOM	281
MOVBYT	015	SAMINT	282
NEWS	059	SAMINV	283
PACCT	029	SAMRDR	284
PBID	060	SAMSUP	280
PCHFLD	178	SECCOM	361
PCHSTA	325	SECHO1	370
PCTTAL	245	SECINT	363
PICCOM	232	SECPRT	365
PICINT	235	SECRD	364

Module Name	LARS Program Abstract Number
SECRDR	362
SECSUP	360
SEPCOM	310
SEPINT	311
SEPRDR	312
SEPSUP	300
SMCLS1	285
SMCLS2	286
SMMULT	287
SMOCOM	351
SMOINT	352
SMOOTH	355
SMORDR	353
SMOSUP	350
SMOTAB	356
SPACE	063
STACOM	326
STAINT	327
STARDR	328
STASUP	320
STAT	135
STATE	032
STATRD	336
STATS	251
STATS2	369
SUBGEN	275
SYMSET	313
TAPOP	011
TERMTEST	068
THRESC	160
TIMER	016
TRARDR	273
TRAXEQ	274
TSPACE	136
TSTREQ	012
TYPEITEM	064
UNIDAT	108
UNMNT	121
URADST	013
USER	314
WRTMTX	017
WRTTRN	137