SENTRY

UTILITIES MANUAL

FAIRCHILD
SYSTEMS TECHNOLOGY
A DIVISION OF FAIRCHILD CAMERA AND INSTRUMENT CORPORATION

SENTRY UTILITIES MANUAL

SOFTWARE REVISION 11.0

Manual Part Number: 6709566 I
Released: June, 1977
Revision I

1725 Technology Drive, San Jose, California 95110

FAIRCHILD
SYSTEMS TECHNOLOGY
A DIVISION OF FAIRCHILD CAMERA AND INSTRUMENT CORPORATION

PREFACE

The SENTRY operating system provides utility programs to assist in organizing and maintaining data, and to aid in the development of FACTOR test programs. This publication discusses the functions provided by each utility program and gives instructions on how to perform these functions.

This manual is intended for use as a quick reference handbook, and as such does not provide an indepth discussion of all utilities described. Since most of the utilities are fully documented in either the Sentry II Operation Manual, part number 67095638, or as Application Notes, these documents may be referred to as desired. When applicable, the description of each utility in this manual includes a reference to additional documentation.

Each utility program described in this publication falls into one of five general classes of programs:

1. General purpose system utility programs
2. Analysis, characterization and datalogging utility programs
3. Debugging aid utility programs
4. FACTOR enhancement utility programs
5. Pattern generation utility programs

The program class into which a specific utility program falls is determined by the function that the utility program performs. (The manual is divided into five sections corresponding to the five program classes).

Within each class, programs are further categorized by the type of distribution which they receive. There are two distribution types:

1. System utilities included in standard revision level releases.
2. Special utility programs designed for a specific function available at extra cost.

(The distribution type for each program is listed in this manual in the table of contents.)

Of major importance to the user is how each utility program is called - that is, whether it is an "operator" or a "programmer" utility. The operator utilities are implemented by either a DOPSY or TOPSY command, as required, from the VKT keyboard. Programmer utilities are known as "assembly language linkage" routines which are called and executed from a user written FACTOR language program using the FACTOR "EXEC" statement.

The following table lists all the utilities by name, section, and page where explained in this manual, as well as how they are called - whether by a DOPSY or TOPSY command or by a FACTOR EXEC statement.

Utility Name	Callable From	Section	Page
FCOMP	DOPSY	1.1	1-1
LISTC	DOPSY	1.2	1-3
CRDTAP	DOPSY	1.3	1-4
TAPLP	DOPSY	1.4	1-5
FINDJOB	DOPSY	1.5	1-6
DELJOB	DOPSY	1.6	1-7
COPJOB	DOPSY	1.7	1-9
CHANGE	DOPSY	1.8	1-12
EDIT	DOPSY	1.9	1-15
PATCH	DOPSY	1.10	1-28
DBUP	DOPSY	1.11	1-30
TDX	DOPSY	1.12	1-34
BMT	DOPSY	1.13	1-41
INIT	DOPSY	1.14	1-43
INSERT	DOPSY	1.15	1-45
NOTE	DOPSY	1.16	1-46
LABEL	DOPSY	1.17	1-48
XMIT	DOPSYor TOPSY	1.18	1-49
XGRAPH	FACTOR	2.1	2-1
TTIME	FACTOR	2.2	2-8
SPLOT	TOPSY or FACTOR	2.3	2-11
PGLOG	FACTOR	2.4	2-18
PPLOG	TOPSY	2.5	2-29
DATAIO	TOPSY or FACTOR	2.6	2-42

Utility Name	Callable From	Section	Page
ACCESS	TOPSY	2.7	2-58
DEBUG	DOPSY or TOPSY	3.1	3-1
PSCAN	TOPSY or FACTOR	3.2	3-7
LMIO	TOPSY	3.3	3-12
CYCLE	TOPSY	3.4	3-18
LMMOD	FACTOR	4.1	4-1
LMTSF	FACTOR	4.2	4-3
LPLF	FACTOR	4.3	4-5
LMLOAD	FACTOR	4.4	4-6
LMSAVE	FACTOR	4.5	4-10
LOGREG	FACTOR	4.6	4-18
FMTAP	FACTOR	4.7	4-19
GLOBS	FACTOR	4.8	4-20
CSETF	FACTOR	5.1	5-1
ROMPAT	FACTOR	5.2	5-4
RAMPAT	FACTOR	5.3	5-8
ROMPONG	FACTOR	5.4	5-19

Any program that is optional or applies only to certain Sentry Systems will be so identified. All other utility programs are standard and apply to all Sentry Systems.

As new programs are developed they will be added to the respective sections of this manual.

NOTE: This manual is consistent with Software Revision 10.3E. For Revision 10.4 and higher, the records // for DOPSY and /. for TOPSY, are optional.

TABLE OF CONTENTS

				Page
PREFACE				iii
SECTION 1	**GENERAL PURPOSE SYSTEM UTILITY PROGRAMS**			
Section	**Name**	**Type***	**Descriptions**	**Page**
1.1	FCOMP	1	Compare two disc files	1-1
1.2	LISTC	1	List card images on line printer	1-2
1.3	CRDTAP	1	Write card images on magnetic tape	1-4
1.4	TAPLP	1	List tape records on line printer	1-5
1.5	FINDJOB	1	Display job numbers on VKT	1-6
1.6	DELJOB	1	Delete files by groups from disc storage	1-7
1.7	COPJOB	2	Copy groups of files from disc to magnetic tape	1-9
1.8	CHANGE	2	Multiple occurrence character string editor	1-12
1.9	EDIT	1	General purpose disc file string and character editor	1-15
1.10	PATCH	1	Modify words in disc files	1-28
1.11	DBUP	1	Disc to magnetic tape backup and bootload program	1-30
1.12	TDX	1	Tape to disc and disc to tape fast transfer	1-34
1.13	BMT	1	Blocked file transfer between tape and disc	1-41
1.14	INIT	1	Reconfigure test station assignments	1-43

* Refer to Preface for a description of "Type".

TABLE OF CONTENTS

Section	Name	Type	Description	Page
1.15	INSERT	1	Edits FACTOR source files and permanently inserts INSERT files in new composite file.	1-45
1.16	NOTE	1	Prints a message to an output device and allows operator control of disc command file execution	1-46
1.17	LABEL	1	Prints a message to an output device in large block letters.	1-47
1.18	XMIT	2	Dumps contents of VKT screen to line printer.	1-48

SECTION 2 ANALYSIS, CHARACTERIZATION AND DATALOGGING UTILITY PROGRAMS

Section	Name	Type	Description	Page
2.1	XGRAPH	1	X-Y and shmoo plotting on a specified output device	2-1
2.2	TTIME	1	High-speed time measurement algorithm	2-8
2.3	SPLOT	1	X-Y sensitivity plot routine	2-11
2.4	PGLOG	2	Plots RAM test pattern failmaps on specified output device. For use with Hardware Pattern Generator (HPG).	2-27
2.5	PPLOG	2	Plots RAM test pattern failmaps on specified output device. For use with Pattern Processor Module (PPM).	2-29
2.6	DATAIO	2	General purpose disc and magnetic tape I/O utility.	2-42
2.7	ACCESS	2	Determines number of disc access operations performed during the execution of a FACTOR program.	2-58

TABLE OF CONTENTS

SECTION 3 DEBUGGING AID UTILITY PROGRAMS

Section	Name	Type	Description	Page
3.1	DEBUG	1	General purpose debugging aid.	3-1
3.2	PSCAN	1	Analysis aid which displays status of programmed pins and power supplies at time of execution.	3-7
3.3	LMIO	1	Dumps Local Memory contents in various formats to specified output device. Loads Local Memory from a disc file.	3-12
3.4	CYCLE	2	Initiates a Local Memory continuous loop.	3-18

SECTION 4 FACTOR ENHANCEMENT UTILITY PROGRAMS

Section	Name	Type	Description	Page
4.1	LMMOD	1	Allows modification of local memory contents from FACTOR program.	4-1
4.2	LMTSF	1	Generates string files of functional data from local memory.	4-3
4.3	LPLF	1	Allows program control of line printer within a FACTOR program.	4-5
4.4	LMLOAD	1	Transfer functional test data between local memory and disc files in FACTOR program.	4-6
4.5	LMSAVE	2	Microprocessor test generation aid.	4-10
4.6	LOGREG	2	Long register reading and writing routine.	4-18
4.7	FMTAP	2	Allows magnetic tape unit control within a FACTOR program.	4-19
4.8	GLOBS	2	Extends number of global variables from 20 to 120.	4-20

TABLE OF CONTENTS

SECTION 5 PATTERN GENERATION UTILITY PROGRAMS

Section	Name	Type	Description	Page
5.1	CSETF	2	Algorithmetic pattern generator of SET F string files.	5-1
5.2	ROMPAT	2	ROM test pattern generator.	5-4
5.3	RAMPAT	2	RAM test pattern generator.	5-8
5.4	ROMPONG	2	ROM access time testing aid.	5-19

LIST OF ILLUSTRATIONS

Figure		Page
1-1	Sample LABEL Output Message	1-50
2-1	TTIME Binary Search Algorithm Flow Chart	2-10
2-2	Shmoo Graph Produced By a SPLOT Two-Variable Device Test	2-11
2-3	Shmoo Graph Produced By a SPLOT Three-Variable Device Test	2-16
2-4	Customized Shmoo Graph Produced by a SPLOT Multi-Variable Device Test	2-18
2-5	Format of a SPLOT-Generated Shmoo Graph	2-24
2-6	Sample PPLOG display of a Failing Checkerboard Pattern	2-34
2-7A	Sample PPLOG display of the First Position of a Failing Spiral Pattern (Walking Diagonal)	2-35
2-7B	Sample PPLOG display of the Fourth Iteration of a Failing Spiral Pattern (Walking Diagonal)	2-36
2-8	Sample PPLOG Display of a Spiral Pattern in Which the ALL Parameter was Specified	2-37
2-9	Sample PPLOG Display Illustrating 'Holes' in the Failmap Resulting from Incorrect Values of XMAX or YMAX	2-38
2-10	Sample PPLOG Display of a Diagonal Pattern In Which The Fail Parameter was Specified	2-39
2-11	Sample PPLOG TRACE Output Listing	2-40
2-12	A Sample PPLOG Display of Scrambled and Unscrambled Failmaps	2-41
2-13	Disc Record Format for Variable-Length Records	2-51
2-14	Disc Record Format for Fixed-Length Record Files.	2-52
2-15	Example of an ACCESS Printout	2-59
3-1	Sample PSCAN Display	3-11
3-2	Sample LMIO STRING Output	3-16
3-3	Sample LMIO DEBUG Output	3-17

LIST OF TABLES

Table		Page
1-1	Text Editor File Handling Directives	1-15
1-2	Text Editor Record Oriented Directives	1-16
1-3	Text Editor Backup Directives	1-18
1-4	Text Editor Non-Record-Oriented Directives	1-19
1-5	Text Editor Special Editor Control Directives	1-22
1-6	Text Editor Character Edit Mode Directives	1-24
1-7	TDX Command Elements	1-35
2-1	System Test Unit Numeric Identifiers	2-14
2-2	Contents of a RESULT Array	2-20
2-3	Recommended Load Addresses for ACCESS	2-60

SECTION 1

GENERAL PURPOSE SYSTEMS UTILITY PROGRAMS

1.1 FCOMP

1.1.1 Introduction

The File Compare routine, FCOMP, provides the user with the capability of comparing two disc files.

1.1.2 Program Usage

The command for invoking FCOMP is

// FCOMP 'file1' 'file2'

If 'file2' is omitted, 'file1' will be compared with working storage. Both files must meet the following conditions:

(1) They must appear in the file directory under the same job number.
(2) They must be of the same type.
(3) They must be the same length.

If a set of records fail the comparison, they are output to POD (the record from 'file1' is first), followed by a record containing:

(1) The word from 'file1' that failed (in TASCII)
(2) The word 'file2' that failed (in TASCII)
(3) The word number within the record of the failing word (1-20, Base 10)
(4) The record number within the file of the failing record (1-N, Base 10).

If console switch one is off, the program halts before continuing to check the file.

1.1.3 Error Messages

ERROR MESSAGE	DESCRIPTION
ERROR-- IN FILE SPECIFICATION	One or the other of the input files are not in the directory under the current job number.
ERROR-- IN FILE TYPE	One file does not match the other as to file type.
ERROR-- EOF ON X. where X = 1 or 2	If the two files are not the same length, this message will occur after the beginnings of the two files have been compared.
ERROR-- MISSING PARAMETER	Both file names were omitted.
ERROR-- INVALID FILE NAME	One or the other of the file names start with a blank or an illegal name ($DIRCT or $ARR) was specified.
ERROR-- SYSTEM-2	FCOMP was not able to output a "no-compare" record.
ERROR-- System 4	Working storage could not be opened.
ERROR-- SYSTEM 20	Phase 2 of FCOMP (FCOMP1) could not be loaded.

1.2 LISTC

1.2.1 Introduction

The List Cards routine, LISTC, provides the user with the capability of listing cards, including monitor commands (//) on the line printer.

1.2.2 Program Usage

The command for invoking 'LISTC' is

```
// LISTC
```

The card deck to be listed should be placed in the card reader. The card reader must be ready and the line printer on line. Cards are read until a card is sensed which contains dollar signs ($) in columns 1, 2, and 3 followed by two blank cards.

1.2.3 Error Messages

ERROR MESSAGE	DESCRIPTION
ILLEGAL PUNCH, RELOAD 2 CARDS	An illegal character is sensed in a card. After the message is output to the VKT, the CPU is halted. The illegal punch is in the card already read, one before the last. To continue, correct the bad card and reload the last tow cards and push the CPU START. To ignore bad cards set console switch 1 on. If an illegal character is sensed it is changed to a '%' sign on the list and the listing continues to the end.

1.3 CRDTAP

1.3.1 Introduction

The Card-to-Tape routine (CRDTAP) provides the capability of writing data from cards on magnetic tape, one card (20 words) per block. The resulting tape will be in a format acceptable to the FST-1 Assembler and DOPSY as standard input.

1.3.2 Program Usage

The card deck should be placed in the card reader followed by a card containing dollar signs in columns 1, 2, 3 and two blank cards. The card reader should be placed in the ready condition and the tape drive should be placed on line.

The program is invoked via the command:

```
// CRDTAP
```

Multiple files may be created on a single tape by first positioning the tape to the point at which writing is to begin.

At the end of the operation, CRDTAP writes a tape mark on the tape and leaves it positioned at that point so that further files may be written if desired.

1.4 TAPLP

1.4.1 Introduction

The Tape-to-Line printer routine (TAPLP) reads one record at a time from the magnetic tape unit and prints the records on the line printer.

1.4.2 Program Usage

The program is invoked via the command:

```
// TAPLP
```

The tape drive should be placed on line and the line printer should be in a ready condition.

The records on the tape are assumed to be formatted and coded such that they can be transmitted directly to the line printer without conversion. A tape produced either by CRDTAP, the FST-1 Assembler or DOPSY will be in the format acceptable to TAPLP. Any file on a multi-file tape may be listed by first positioning the tape to the proper file.

1.5 FINDJOB

1.5.1 Introduction

FINDJOB is an assembly language utility program which enables the user to search the directory for a specified file to find out which job it is stored under or if it is on the disc. When no file is specified it will return the current job number.

1.5.2 Program Usage

The program is invoked via the following command:

```
// FINDJOB 'file'
```

where:

'file' is the name of a disc file. If 'file' is missing, locate to the right and display the current job.

The program will respond:

FILE IN JOB YYYY	file on disc is in job YYYY
FILE NOT IN DIRECTORY	file was not found in the directory
CURRENT JOB IS 'XXXX'	the current job is XXXX

FINDJOB will search the entire directory so that if a file name is used in more than one job the user will be informed of each job name.

1.5.3 Error Messages

ERROR MESSAGE	DESCRIPTION
FORMAT ERROR	The file name was missing. The file name must be enclosed in single quotes.

1.6 DELJOB

1.6.1 Introduction

The delete job program, DELJOB, provides the capability of deleting all files under a specific job, except the system job (←←←←), at one time without having to use the system DELETE command directive for each file. It also provides the capability of deleting specific file types from all jobs or a specific job, always excepting the system job.

1.6.2 Program Usage

The program is invoked via the command:

// DELJOB

The delete job routine responds with:

JOB (NAME, "ALL", "END") =

The user answers with the job name to be deleted. Do not enclose the job name in quotes. Entering "ALL" causes all jobs except the system job to be deleted. Entering "END" causes exit from DELJOB.

The delete job routine then asks:

FOR FILES TO BE DELETED ENTER ONE OF THE FOLLOWING:

ALL FILES ENTER	1
ALL STRING FILES ENTER	2
ALL DATA FILES ENTER	3
ALL OBJECT FILES ENTER	4
ALL COREIMAGE FILES ENTER	5

When the specified files have been deleted, the system responds:

JOB NAME DELETED = XXXX

FILE TYPE DELETED = YYYY

and the JOB (NAME, "ALL", "END") question is repeated.

This procedure continues until the user responds with END at which time the following comment is displayed on the VKT:

END DELJOB

and a return is made to the DOPSY monitor.

1.6.3 Error Messages

ERROR MESSAGE	DESCRIPTION
DELJOB ERROR DSRD	An unrecoverable disc read error has occurred. A return is made to the DOPSY monitor.
DELJOB ERROR DSWT	An unrecoverable disc write error has occurred. A return is made to the DOPSY monitor.

1.7 COPJOB

1.7.1 Introduction

The copy job routine, COPJOB, provides the capability of writing string, data, and object files under a specific JOB to mag tape along with their associated // JOB, if requested, and // CREATE command directives. (Files are separated on the mag tape by EOF marks to allow the positioning of the tape using the utility tape handler MTAP.) The magnetic tape unit may then be assigned as the primary input device (// SET MTR) and the data created by COPJOB is used to create the files under a new system.

As the files are copied to mag tape, a log is maintained on the line printer detailing the name, type, and job. Any core image files which are encountered are not copied, but an indication is given on the log. The last record written on the mag tape is // SET TTK.

1.7.2 Program Usage

The program is invoked via the command:

```
// COPJOB
```

COPJOB responds with:

```
INCLUDE // JOB RECORDS ON OUTPUT TAPE? (Y
OR N)
```

If the user chooses to ignore the // JOB records on his mag tape, he responds with N, otherwise he responds with Y and the files' respective // JOB records will be included on the mag tape.

COPJOB then requests the JOB name of the files to be copied:

```
(1)          JOB (NAME,"ALL","END")=
```

If the user answers with a JOB name, COPJOB responds with a second question

```
(2) FOR FILES TO BE COPIED, ENTER ONE OF THE FOLLOWING:

        SPECIFIC FILE ENTER NAME OF FILE

        ALL FILES THIS JOB ENTER                1

        ALL STRING FILE THIS JOB ENTER          2

        ALL DATA FILES THIS JOB ENTER           3
```

ALL OBJECT FILES THIS JOB ENTER 4

ALL FILES "FROM-TO" ENTER 5

The user must respond with either a file name or one of the digits 1-5. In all cases, after the requested files have been copied to mag tape question number one (1) will be repeated again. This procedure continues until the user responds with END to question number one (1) at which time a // SET TTK record will be written on the mag tape; the mag tape will be rewound and a return is made to the DOPSY monitor.

If the response to question number two (2) is five (5), two more questions are displayed

NAME OF FILE FROM-

and

NAME OF FILE TO-

The user responds with the first and last file names within a group of files to be copied.

If the user responds to question number one with ALL, all of the files not under the system job number except core image files will be dumped to mag tape.

WARNING TO USER. The magnetic tape is not initially positioned. It is the user's responsibility to insure that the magnetic tape unit is on-line and positioned at the point where files are to be written.

If the user responds with a T, a return is made to the DOPSY monitor.

If the user responds with a C, files will continue to be copied starting at the beginning of the last interrupted file. (CAUTION - Do not forget to mount new tape, or your previous tape will be overwritten.)

1.7.3 Error Messages

ERROR MESSAGES	DESCRIPTION
NOT IN DIRECTORY	The job or file name entered is not in the disc directory. The question is repeated.
TYPE T TO TERMINATE, OR MOUNT NEW TAPE. TYPE C TO CONTINUE.	The end of tape mark has been reached while writing the mag tape. The tape is skipped back to the last EOF mark, // SET TTK record is written on the mag tape, the tape is rewound.

ERROR MESSAGE	DESCRIPTION
	If the user responds with a T, a return is made to the DOPSY monitor. If the user responds with a C, files will continue to be copied starting at the beginning of the last interrupted file. (CAUTION - Do not forget to mount net tape, or your previous tape will be overwritten).

1.8 CHANGE

1.8.1 Introduction

CHANGE is a DOPSY utility program which allows the user to edit any disk file and replace every occurrence of a given character string with a second given character string. The results of the edit are left in working storage from which the user may create a new file.

1.8.2 Program Usage

CHANGE is invoked by the following command:

```
// CHANGE
```

CHANGE will respond:

```
ENTER FILE NAME:
ENTER OLD STRING:
ENTER NEW STRING:
```

Note: Do not enclose file name in single quotes.

On completion CHANGE will respond:

```
XXX CHANGE(S) MADE
ANOTHER PASS (Y/N)?
```

Entering Y will cause CHANGE to re-execute with another request for the old string, then the new string. Entering N will terminate execution and produce the following message:

CREATE NEW FILE FROM WORKING STORAGE

If the line printer is the POD the above responses and those typed in by the user will be displayed on both the LP and TTP. Otherwise, they will appear only on the TTP.

If the request for the file name is answered with an immediate carriage return, the contents of working storage will be edited. The 'old' and 'new' character strings may optionally be enclosed in double quotes (") in order to include leading and/or trailing blanks. Imbedded blanks are also permitted.

An immediate carriage return entered as the new 'string' will replace the old 'string' with a single space. This will permit deletion of an entire record. Up to 72 characters may be entered as the new and old character string.

A double slash (//) may be entered as a response at any time and will cause CHANGE to abort and return control to DOPSY. The contents of working storage will remain intact and the new file may be created as usual.

1.8.2.1 ENTERING CHANGE COMMANDS FROM A DEVICE OTHER THAN THE TTK. CHANGE reads all user responses from the PID which may be set to the card reader, a DIF file or mag tape. To change a statement from WRITE (TTP) VALUE; to WRITE (LP) VALUE; from the card reader the following cards would be used:

```
// CHANGE
FNAME
(TTP)
(LP)
N
// CREATE 'FNAME' OVLY
// SET TTK
```

FNAME is the name of the original disc file to be changed. The user would place the card deck in the card reader then enter // SET CR at the TTK. The line printer could also be set to cause all dialogue between CHANGE and the user to be logged to the line printer.

1.8.2.2 CHANGING STRINGS WITH LEADING AND/OR TRAILING BLANKS. The 'old' and 'new' character strings may optionally be enclosed in double quotes (") to permit leading and/or trailing blanks to be included as shown in the following example.

```
given:      FORCE VOLTAGEXX.XX
change:     "AGE" to "AGE "
result:     FORCE VOLTAGE XX.XX
```

where XX.XX is any voltage value

1.8.2.3 USING PERCENT CHARACTER (%) AS A 'NO CHANGAE DESIRED' INDICATOR. The % character may be used to suppress the changing of characters which are bracketed by characters which are to be changed and should read as: "don't change the character in this position".

```
Examples:   given:    A ? C
                      A * C
                      A ! C
            change:   "A % C" to "X % Z"
            result:   X ? Z
                      X * Z
                      X ! Z
```

NOTE: % used in both old and new strings.

```
            given:    * A *
                      * B *
                      * C *
change:               "*% *" to "***"
result:               ***
                      ***
                      ***
```

NOTE: % used in old string only.

1.8.2.4 USING THE BACK ARROW (←) TO EDIT END OF RECORD (EOR) CODES. The back arrow (←) is equivalent to a TASCII 77B which is the code used to indicate the end of a record for string files. The back arrow may therefore be used with CHANGE to remove or add spaces between lines, to move the contents of several lines to one line or the contents of one line to more than one line. To make these changes the user must be aware that an EOR immediately precedes the first character in a line (i.e., card column 1).

Examples:

to insert a new record into a file

given:	CPMU PIN 3;
change:	"CPMU PIN 3;" to
	"REM VOH TESTS; ← CPMU PIN 3;"
results:	REM VOH TESTS;
	CPMU PIN 3;

to place the contents of two lines onto a single line

given:	CPMU PIN 3;
	MEASURE PIN;
change:	" MEASURE PIN;" to "MEASURE PIN;"
result:	CPMU PIN 3; MEASURE PIN;

1.8.3 Error Messages

ERROR MESSAGES	DESCRIPTION
FILE NOT FOUND	The disc file requested could not be found under the current job.
STRING NOT FOUND	The entire file was scanned and no changes were made because the old string could not be found.
INSUFFICIENT SPACE IN WORKING STORAGE	There is not enough room on disc to complete the job.
LINE PRINTER OFFLINE	The listing cannot be made to the line printer.
DISC WRITE DISABLED	A write to the disc cannot be done until the disc write is enabled.
CARD READER EOF OR MONITOR RECORD DETECTED	The input to CHANGE has been terminated.

1.9 EDIT

1.9.1 Introduction

The editor program allows editing of any string files in disc memory or the combining of two or more string files. Only string files may be edited. The editor program copies the requested string file into working storage, under control of operator entered editor directives. After editing is complete, the new file is in working storage, and may be added to the disc memory permanent file. The old file is still on disc and has not been changed.

1.9.2 Program Usage

The editor is envoked by one of the following commands:

```
// EDIT
```

This command calls the system editor program to edit a source file on disc which may be a FACTOR language source file consisting of tester statements, an assembly language source code file, or a DIF file containing DOPSY commands or system information.

During the edit process the old file is opened or made available to the editor as the input file and the new file, or output file is built under direction of the user entered directives and is placed in working storage on the disc. The old file is retained unaltered. When the editor is requested, working storage is initialized so that any previous file in working storage is lost.

```
// EDIT 'filename'
```

This statement calls the system editor and opens the file to be updated, i.e., the input file, The editor's prompting character is ' < ', signaling it is ready for directives.

1.9.2.1 DESCRIPTION OF EDITOR DIRECTIVES

TABLE 1-1 TEXT EDITOR FILE HANDLING DIRECTIVES

Editor Directives	Description
O 'filename'	Open the named file as the input file for editing purposes. This statement may be used to open a second file and edit its contents into the original.
I 'filename'	Open the named file as the temporary input file and insert the entire file into the output file. When the operation is complete the prompting character is returned. The original file is still open and editing will continue at the same position as before. To edit the inserted records either a second pass or a back save operation must be requested.

TABLE 1-2 TEXT EDITOR RECORD ORIENTED DIRECTIVES

Editor Directives	Description
C	Copy the next record of the input file to the output (working file to the output (working storage) so that it is retained in the new file. The record copied is displayed.
C (n) (label)	Copy all records up to and including the nth record or the labelled record to working storage. The label in the record to be copied must start in column 1. The last record copied is displayed.
C label + n	Copy to the label plus n records. N records past the next occurrence of the label are copied to working storage. A minus sign is ignored. The last record copied is displayed.
CEOF	Copies all records until the end of the input file. Therefore, it is not possible to copy to a label called 'EOF'.
D	Delete the next record of the input file, i.e., do not copy the record to working storage.
D (n) (label)	Delete all records up to and including the nth record or the record containing the label in column 1.
D label + n	Delete to and including the label plus n records. A minus sign is ignored. The last record deleted is displayed.
I	I for Insert places the editor in the insert mode. The editor responds with the equal sign (=) as the prompting character. The following records are entered in the output file until the insert mode is terminated by typing // in column 1 and 2 with no other data on the record. (It is possible to enter monitor records in the insert mode, so that a DIF file may be created or edited).
V	Verify causes the following record of the input file to be displayed without copying it to the output file. Successive verify requests automatically copy the last record verified and display the next record.

TABLE 1-2 TEXT EDITOR RECORD ORIENTED DIRECTIVES

Editor Directives	Description
V (n) (label)	Copy n-1 records or up to but not including the record containing the label. Display the nth record or the record containing the label in column 1 as the next record of input. When the verify command does not have a label request, successive verifies V (n) cause an automatic copy to take place.
V label + n	Copy to label + n - 1. Display the next record as the next record of input.
A An A label A label + n	The alter directive causes a delete of one record, n records, or to and including the label, or the label + n records. The last record deleted is displayed. The insert mode is then entered and all records entered are output to the new file until the insert mode is terminated by typing // in column 1 and 2 with no other data in the record.
Mn	M for Multiply places the editor in the insert-and-multiply mode. The editor responds with the equal sign as the prompting character. The following records are entered in the output file until the mode is terminated by typing // in column 1 and 2 with no other data on the record. When the mode is terminated the sequence of records entered is multiplied in the output file n times. Note that if n=1 this mode is identical to the insert mode. However, if no records have been entered when the terminating '//' is sensed, the next record of the input file is multiplied in the output file n times.

TABLE 1-3 TEXT EDITOR BACKUP DIRECTIVES

Editor Directives	Description
B Bn	Back up the input file and the output file 1 or n records. The last record remaining in the output file is displayed. When no records have been inserted or deleted in the section of the output file being backed up over, use this directive. For example, if it is desired to copy to a label minus three records, verify to the label and backup three records (B3). When the output file has been changed in the section to be backed up over either back up each file independently, erasing the changes, or back up the output file and save the changes. (See below.)
BO BOn	Back up the input file (Backup Old) 1 or n records. The last record remaining in the input file is displayed. If it is desired to duplicate a sequence of records of the input file, copy past the records, back up over them in the input file only (BOn) and copy them again (or alter the records in any way desired). If more records have been deleted from the output file than desired, back up the input file only. For example, to delete to but not including a label, delete to label (D label) and restore it again in the input file (BO). The next copy will copy the record to the output file.
BN BNn	Back up the output file (Backup New) 1 or n records. The last record remaining in the output file is displayed. If one or more records have been inserted or copied to the output file which are not wanted, a back up of the output file will delete the records. If backing up over a section where the number of records has been altered, back up the files separately with BOn and BNn so that the position of each file is displayed. The position of each can then be compared to avoid duplicate or missing records.

TABLE 1-3 TEXT EDITOR BACKUP DIRECTIVES

Editor Directives	Description
X, Xn	Same as BN, BNn
BS BSn	Back up the output file 1 or n records, saving the changes made (Backup Save). The last record remaining in the output file is displayed. The saved records may be edited in any way, for example they may be edited by the character editor, or records may be inserted or deleted. It is possible to back up saving the output, make changes or look at the position and back up and save again. However, there is a limit of 1000 words, or 50 full length records that may be saved. If it is desired to look back further a second pass must be requested, and the file repositioned with a copy request. While editing saved output the B and BO directives should not be used as duplicate and missing records will result.

TABLE 1-4 TEXT EDITOR NON-RECORD-ORIENTED DIRECTIVES

The following directives will scan for a 'string' of data anywhere on a line. Data may be altered by request without retyping the whole record.

Editor Directives	Description
V 'string'	Copy up to the record containing the 'string'. Display the record with the string as the next record of input. This command never copies a record automatically, so a verify to the same string will not advance past the current record. However, if this is followed by a V or Vn, the current record will be copied automatically since it has already been displayed.
V 'string' n	When a number is entered, the scan will search for the first occurrence of 'string' in the next n-1 lines. If it is not found, the nth record is displayed as the next record to be input, thus limiting the length of the scan.

TABLE 1-4 TEXT EDITOR NON-RECORD-ORIENTED DIRECTIVES

Editor Directives	Description
A 'string1' 'string2'	Copy to the record containing the first string and alter its contents to the second string. The changed line is output and is displayed. The next line of input is also displayed. If this directive is followed by a V or Vn directive, the second displayed line is automatically copied.
A 'string1' 'string2' n	When a number is entered, scan the next n records making the requested change on the records where 'string1' is found. All lines changed are displayed. The n + 1 record is also displayed as the next record of input. The number n therefore limits the scan to n records even if 'string1' is not found, or extends the scan to n records making the requested change as often as 'string1' is found. If this directive is followed by V or Vn, an automatic copy of the last displayed record occurs.

The '%' character may be used in these directives to suppress the comparison of a character in a certain position so that it may be changed regardless of what it is.

```
Example     Input:      SET F 00000   11011   00011;

                        SET F  00000   10111   00111;

            Directive:  A '1%%11'  '11111'    2

            Output:     SET F   00000  11111   00011;

                        SET F   00000  11111   00111;
```

The '2' causes the change to take place in the next 2 records. The third record will be displayed but not altered.

To suppress the comparison and the change of characters in a certain position, enter the '%' character in both strings.

```
Example     Input:      SET  F  00000  11011   00011;

                        SET F   00000  10111   00011;

            Directive:  A '1%%11'  '1%%00'    2

            Output:     SET F   00000  11000   00011;

                        SET F   00000  10100   00011;
```

The back arrow '←' is used to edit end of record (EOR) codes. The '←' precedes the first character in a line except at the beginning of the file. The '←' may be used to insert a new record or blank records, or to place the contents of two lines onto a single line.

Examples:

To insert a new record into a file

Input: CPMU PIN 3;

Directive: A 'CPMU PIN 3;' 'CPMU PIN 3;←REM VOH TESTS;'

Output: CPMU PIN 3;
REM VOH TESTS;

To place the contents of two lines onto a single line.

Input: CPMU PIN3;
MEASURE PIN:

Directive: A ' ← MEASURE PIN;' ' MEASURE PIN;'

Output: CPMU PIN 3; MEASURE PIN;

To place the contents of one line onto two lines.

Input: CPMU PIN 3; MEASURE PIN;

Directive: A 'MEASURE PIN;' ' ← MEASURE PIN;'

Output: CPMU PIN 3;
MEASURE PIN;

To delete a line.

Input: CPMU PIN 3;
REM VOH TESTS;

Directive: A ' ← REM VOH TESTS:' "

Output: CPMU PIN 3;

Note that the back arrow must precede the line. If the back arrow is missing a blank line will result.

TABLE 1-5 TEXT EDITOR SPECIAL EDITOR CONTROL DIRECTIVES

Directives	Description
Tn, n2 n3...n20	Enter up to 20 tab stops for use in the character edit mode. Tabs must be entered in ascending order. The default tab stops are 5, 10, 15, 20...
E	Enter the character edit mode and display the next line which is the current line available for character editing.
Ex	Enter the character edit mode and display the next line up to the first occurrence of x. The current line is available for character editing.
S [CR/TTK]	Change the input device used for reading edit commands to the card reader (SCR) or the keyboard (STTK).
L	List the input file to the VKT as it is processed.
U	Turn off the L request. (Unlist).
//	Exit the editor immediately, returning to DOP-SY. The output file is in working storage. To save the file it must be CREATEd.
Z 'string'	Replace the '//'as the exit request from the editor, multiply mode, and insert mode with 'string', i.e., any two character terminator.
// R	The repeat request copies the input file to working storage to the end of the file. A dummy file called '/$##$/' is created and the editor is reentered for a second pass. All capabilities and rules of the first pass apply to second and subsequent passes. There is no limit to the number of passes, however, there must be room on the disc for three copies of the file, the original file which remains unchanged, the last pass, and the current pass which is in working storage. (If the disc does overflow the message 'ERROR--SYSTEM-4' will result and the last pass remains on the disk as '/$##$/'.) When a repeat is requested the tab stops return to the default of 5, 10, 15, 20..., and the terminator becomes // again if it has been changed by a Z directive. The list request remains active, however. When the Editor is

Directives	Descriptions
	exited the dummy file '/$##$/' is deleted automatically, leaving two copies of the file on disc: the original and the result of the last pass. The user should not name any file '/$##$/' as this file will be automatically deleted by the editor on exit or if a second pass is requested.

// 'filename'	This request copies to the end of the input file and exits the editor. The 'filename' is deleted from the disc and the file in working storage is created with this name. This command takes the place of the sequence of commands:
	
	CEOF
	//
	// DELETE 'filename'
	// CREATE 'filename'
	// SET TTK TTP

CS1	Set on console switch 1 to suppress the display of the last record copied, or deleted as the file is processed. The L request overrides CS1.

TABLE 1-6 TEXT EDITOR CHARACTER EDIT MODE DIRECTIVES

When the input device is the keyboard the character edit mode may be used to change part of a line without reentering the whole line. Also records may be inserted in this mode or repeated with or without modifications. In this mode an uparrow, ' ', is always displayed when a record is output except when more than one record is scanned by CTRL-V.

There is no prompting character in the character edit mode (except for CTRL-V) because it would enter the character stream and displace the line. Note also that the control characters used as directives are not displayed as this also would displace the line. Therefore, after a line has been scanned and altered it appears on the screen as it will be in the output file.

Directives	Description
E	Enter the character edit mode and display the current line which is available for character editing.
Ex	Enter the character edit mode and display the current line up to the first occurrence of x. The current line is available for character editing.
CTRL-R x	Scan the current line and display to the next occurence of x. If x is a CTRL-T, display the line to the next tab stop.
CTRL-S	Scan and display the current line to the next occurrence of x, as defined by the last E or CTRL-R command. (Following a CTRL-P or CTRL-C directive, x is destroyed so that CTRL-S will display the entire current line.)
CTRL-N	Advance one character across the current line and display. Gives capability of walking across the line to the end of record. To insert blanks after the last character the space bar or CTRL-T should be used.
CTRL-P	Advance across the current line or the remainder of the current line and display. The line is still available for character editing.
Character other than Opcodes	Enter the character stream at the cursor location, extending the current line up to 80 characters.

Directives	Description
←	Truncate record here, i.e., insert an end of record. The record up to this point is still available for character editing.
CTRL-B	Back-up and delete one character from the current line. For example, to delete x from the line, scan up to x with CTRL-Rx, CTRL-S, or CTRL-N, back-up using CTRL-B until all characters not wanted are erased. New characters may be inserted, or the rest of the line may be kept with CTRL-P.
CTRL-L, RUBOUT	Restart line scan at the beginning, erasing the current changes.
CTRL-T	Insert spaces into the current line to the next tab stop. If tab stops are not entered by the user, the default is 5, 10, 15, 20...
(CR)	Copy the current line to working storage. The record is still available for character editing. If entered at column 1 the entire record is output, otherwise the record is truncated and output to the cursor location only.
CTRL-C	Copy the current line to working storage. The next input record is displayed and is available for character editing as the new current line. This directive may be entered anywhere on the line. If entered at the beginning of the line, the entire record is output, otherwise the line up to the cursor only is output. Successive CTRLC directives allow walking down the file. Note that to insert a blank line, hit the space bar and then carriage return or CTRL-C. The blank line then replaces the current line.
CTRL-E	Copy the current line to working storage and exit the character edit mode. If entered at column 1 the entire current record is output, otherwise the line up to the cursor only is output.
CTRL-X	Exit the character edit mode. The current line is lost.

Directives	Description
CTRL-V n	When CTRL-V is entered, the '=' prompting character is returned. Enter n and carriage return. The current line is automatically output to working storage. N-1 records are copied. The nth record is displayed and is available for character editing. Note that if n=1, the CTRL-V directive is identical to CTRL-C. To exit the mode without advancing enter '//' in column 1 and 2.
CTRL-V 'string1' 'string2' n	When the '=' prompting character is returned enter 'string' or n, and carriage return. The current line is automatically output to working storage. The input file is scanned for 'string' and the record containing 'string' is available for character editing. If a number is entered the scan will stop at the nth record if 'string' is not reached first. The nth record is then displayed and is available for character editing. (If a mistake is made while entering the line after CTRL-V is typed, CTRL-B will delete characters and CTRL-L or RUBOUT will delete the line and prompt again with the '='. If it is desired not to enter a CTRL-V request, typing // immediately following the '=' will terminate the mode and return control to the character edit mode.)

1.9.3 Error and Warning Messages

ERROR MESSAGES	DESCRIPTION
ERROR-- SYSTEM-2	The input or output file was lost during a backup operation.
ERROR-- SYSTEM-4	Working storage overflow.
ERROR-- IN FILE TYPE	Only source files may be edited.
ERROR-- INVALID FILE NAME	$DIRCT and $ARR may not be edited.
ERROR-- MISSING PARAMETER	The file name was not specified in the open command.
ERROR-- WS EMPTY	A second pass was requested with no data in the output file.

WARNING MESSAGES	DESCRIPTION
EOF INPUT	The end of the file was reached while processing the file.
INPUT FILE START	The beginning of the input file was reached during a backup operation (B, BO).
OUTPUT FILE START	The beginning of the output file was reached during a backup operation (B, X, or BS).
BACKUP LIMIT REACHED	During the backup-save operation (BS), the save buffer is exhausted. It is not possible to back up more. Request a second pass with '// R' and copy the file down to the desired position.

1.10 PATCH

1.10.1 Introduction

PATCH is used to examine or modify files on the disc. The user first makes a file available to PATCH opening it, and then may "read" or "write" words of the file by supplying addresses and, for write, the new contents. PATCH may also be used to add or subtract values.

1.10.2 Program Usage

The PATCH program is invoked via the command:

// PATCH ('filename')

This DOPSY command calls PATCH and enables the use of PATCH commands. If 'filename' is entered, the file specified is opened and may be examined or modified. A file must be opened before entering all commands except for calculation. If the file cannot be found under the current job, the error response '??' is output.

Input is from the system PID. All output is listed to the system POD. Numbers entered are assumed to be decimal unless octal is indicated by the user. All numbers output are octal. Spaces are allowed following the directive but are not required.

Patch Directives	Description
0 'filename'	Open the named file so it may be examined or modified. A file must be opened before entering all commands except for calculation. If the file cannot be found under the current job, the error response '??" is output.
C $n_1(\pm n_2 \ldots \pm n_x)$	Calculate the value of $n_1 \pm n_2 \ldots \pm n_x$. The result is displayed in octal. This feature may be used to do decimal to octal conversion since a single decimal number entered will be displayed in octal. The series of numbers input is truncated by carriage return or by (=) and carriage return.
D	Return control to the DOPSY monitor.

The following PATCH directives specify or infer an address or location in the file being patched.

For string files, object files, and data files, the addresses given to PATCH must start relative to the first word of the file. For core image files, PATCH will accept only those addresses corresponding to the absolute core location of the file to be patched. Addresses are assumed to be decimal unless otherwise specified.

Patch Directives	Description
R n (*comment) R n_1-n_2 (*comment) R n_1, n_2 (*comment)	Read the contents of the address specified or the contents of addresses nl through n2 and display to the POD. If either number is outside the file the error response '??' is output.
R (*comment) carriage return	Once an address is supplied to PATCH, the following address may be displayed by typing R or carriage return. The current location is bumped and the contents of next address location is displayed. No error checking is done on the address so if the address exceeds the file, the contents is specified as zero.
B (*comment)	Once an address is supplied to PATCH, the previous address may be displayed. The current location is decremented and the contents of the previous address location is displayed. No error checking is done on the address so if the address is below the file the contents is specified as zero.
W n:n_1 (*comment) W n:n_1,n_2...,n_x (*comment)	Write the value specified into the address location. If more than one value is entered, separated by commas, consecutive addresses are altered beginning with the address specified.
E (current address:) :n_1 (, n_2...,n_x) (*comment)	Alter the contents of the current address. When 'E' is specified, PATCH responds with the address of the current location followed by a colon and carriage return. The value entered becomes the contents of the current location. If more than one value is entered, separated by commas, consecutive addresses are altered beginning with the current location. The current location becomes the address corresponding to the last value entered.

1.11 DBUP

1.11.1 Introduction

The purpose of DBUP is to allow the user to create a magnetic tape copy of the system disc. This tape can then be read back to restore the system.

1.11.2 Program Usage

The program performs two functions: (1) creating a magnetic tape copy of the disc and (2) loading the created tape back to the disc.

1.11.2.1 DISC BACK-UP ONTO TAPE. The tape must be mounted on tape unit 0 at the BOT position, and the unit turned on. DBUP is called by entering the command:

```
// DBUP  (VERIFY)(ONLY)('Message up to 48 characters')
```

The contents of the disc is backed-up to the tape. The VERIFY option causes an automatic verification at the completion of the back-up procedure. The 48 character message may contain any valid character except , @, or a single quote. This message is output before writing to a tape and immediately after the load procedure is initiated. The ONLY option provides only the verification of the tape against the disc. No data transfer to tape or to disc occurs in this procedure.

1.11.2.2 BOOTLOAD FROM TAPE TO DISC. The tape must be mounted on tape unit 0 and the normal load operation is performed.

1.11.3 Program Description

1.11.3.1 DESCRIPTION OF THE DBUP PROCEDURE:

DBUP initially checks for the tape unit being ready, that the tape is at BOT, and that the write-ring is present. Appropriate error messages are displayed and the program aborted if these conditions are not met.

After the initial check , the first 2K words of memory (which includes the DBUP program) are written to the tape to serve as the bootload record for later reloading the disc.

After the boot record, the contents of the disc are transferred in 20 sector blocks (960 words) to the tape. The block is read from disc into memory where a checksum is derived. The block and checksum are then placed on the tape. The checksum of each block is also accumulated and is the total checksum displayed on the TTP at the end of the back-up procedure.

Should a tape-write error occur, four inches of tape will be skipped before an attempt is made to rewrite the record. DBUP will try writing tape up to ten times before aborting the operation.

At the end of the back-up procedure, the tape is rewound during which time the keyboard is locked out.

If the VERIFY Option is selected, DBUP skips past the boot record and performs a Verification operation similar to loading. There is no writing to the disc during Verification so that the contents of the disc are not altered. The memory size of the CPU is displayed.

The message "VERIFYING TAPE BACK-UP" is displayed on the TTP. If any error message is displayed during verification, the back-up procedure should be repeated.

At the end of Verification, the checksum is displayed again and the tape will be rewound.

1.11.3.2 DESCRIPTION OF THE BOOTLOAD PROCEDURE:

After the boot record is loaded, DBUP displays the user message if any, and the memory size of the CPU on which the system tape is being loaded. The maximum address of memory is stored in the loc 117B. DBUP then reads a record from the tape, checksums to the 20 sector block and compares it to the checksum obtained from the tape. If they are equal, the block is then written to the disc. It is then re-read from the disc and checksummed again to insure a proper transfer of data to the disc has occurred.

After the contents of the tape have been transferred to disc, the total checksum is displayed on the TTP.

1.11.4 Error Procedures

The following error messages are displayed during the disc back-up onto tape procedure.

ERROR MESSAGES	ACTION NEEDED, IF ANY
TAPE UNIT NOT READY PROGRAM ABORTED	Ready the tape unit and reenter the command.
TAPE IS NOT POSITIONED AT BOT PROGRAM ABORTED	Position the tape at BOT and re-enter the command.
WRITE-RING MISSING PROGRAM ABORTED	Place a write-ring on the tape and re-enter the command.
UNABLE TO WRITE TO TAPE PROGRAM ABORTED	Check the tape and tape unit.
DISC ERROR PROGRAM ABORTED	Check the disc.

The following error checks are performed during the Bootload procedure.

DBUP will attempt to re-read any record that is indicated as in error by the tape unit, e.g., parity errors. If, after ten attempts, the record is still in error, an appropriate error message is displayed and the procedure is normally aborted. However, if console switch 1 is on, DBUP will continue loading after displaying the error message and the disc sector address to which the record is transferred.

If a checksum error occurs during a tape read, i.e., the calculated checksum does not match with the checksum on the tape, the record is re-read up to five times in an attempt to obtain the same checksum. If the checksum still does not match, the record is written to the disc and the appropriate error message is displayed.

On the checksum errors from disc, the error message is displayed. But no attempt is made to re-read the data.

When the checksum error message is displayed, the disc sector address to which the error data is transferred is also displayed. Normally at this point, the procedure is aborted. However, if the console switch 1 is on, the DBUP will continue loading.

The following error messages are displayed during the disc load procedure.

ERROR MESSAGES	ACTION NEEDED
TAPE UNIT IS NOT READY PROGRAM ABORTED	Ready tape unit and reload the tape.
CHECKSUM ERROR OCCURRED DURING TAPE READ SECTOR # = nnnnnn ERROR IN xxxxx	Reload the tape.
CHECKSUM ERROR OCCURRED DURING DISC READ SECTOR # = nnnnnn IN xxxxx	Reload the tape.
DISC WRITE DISABLED PROGRAM ABORTED	Enable disc write switch and reload the tape.
DISC ERROR SECTOR # = nnnnnn IN xxxxx PROGRAM ABORTED	Check the disc.

ERROR MESSAGES	ACTION NEEDED
'ERROR IN xxxxx' is one of the following:	
ERROR IN ARR AREA	Error occurred in the sector where the Automatic Restart Routine is located; unrecoverable.
ERROR IN DIRECTORY AREA	Error occurred in the sector where the file directory is located; unrecoverable.
ERROR IN FILE xxxxxx JOB xxxx	Error occurred in the sector where the file xxxxxx of the job xxxx is located.

1.12 TDX

1.12.1 Introduction

TDX (Tape Disc Transfer) is a file management tool for transferring groups of files between Disc and Tape. It is a DOPSY Utility which functions by generating an appropriate set of DOPSY commands (CREATE, FDUMP, JOB, NOTE, SET, VERIFY, UTILITY 'BMT', etc.) on disc in a Disc Input File (DIF). When all the commands have been generated, the Primary Input Device (PID) is set to this DIF. This causes the group of files to be transferred using these standard DOPSY commands.

Because of the use of DOPSY commands, tapes made with TDX may be read independent of TDX if necessary, error messages are familiar to users, and any system routine improvements are incorporated automatically.

TDX contains many features which were previously unavailable in any single DOPSY command. These features are as follows:

A single file or groups of files may be written to or read from tape with a single command.

An individual file may be loaded from a TDX tape containing many files (i.e.: all files need not be loaded to get one or two files).

The validity of a file is automatically checked when loaded from tape.

Files are written to tape in a compact "blocked" format which yields a 20-fold savings in time and amount of tape used.

A TDX tape contains at least one file directory so that the tape contents may be readily ascertained or documented.

A given TDX directory may be used to determine whether or not a specific file on disc matches the same file on tape.

1.12.2 TDX Usage

TDX usage is first described in terms of a formal command description with a brief definition of command options. Secondly, TDX usage is described by various examples.

1.12.2.1 FORMAL TDX COMMAND DESCRIPTION. In the TDX command described below, the following conventions will be observed:

X/Y/Z	One of the listed options is required.
(X/Y/Z)	One or more of the listed options may be used but none is required.
Ø	Indicates that more than one of the listed options may be chosen.

An underlined item in a set indicates the default value if no other items are chosen.

Constant names are shown in upper-case. They must always be entered exactly as shown.

Variable names or quantities are shown in lower case. Their value changes with usage and must be supplied by the user.

The TDX command is:

// TDX [MAKE /LOAD/LIST/INIT/VALL/REV]
(DIRECT 1/n/L)(LP/TTP) (Notes) ((job#))

['filnam'/SOURCE/OBJECT/DATA/ALL/0]
(CTRL/NOVER/NOJOB/OVLY/CLEAR/HOLD/FDUMP/Ø)

The command elements are defined as follows:

TABLE 1-7 TDX Command Elements

Element	Definition
MAKE	Write all specified files to tape. If a (JOB#) has been specified, a job record will precede the files on tape unless NOJOB is also specified.
LOAD	Load the files from tape back to disc under the job corresponding to the job record which precedes the files on tape. If NOJOB is specified the files will be loaded to the current job.
LIST	List the directory specified in the DIRECT option on the specified output device. Follow with either LP or TTP.
INIT	Initialize a tape which is not currently a TDX tape. Before files can be dumped to a tape via TDX, the tape must be initialized.
VALL	Create a DIF file '.VRALL' which contains VERIFY records of all files under the specified job. No files are transferred.
REV	Display on the TTP the revision number and date of the version of TDX currently installed.
DIRECT 1/n/L	All additional commands reference the nth directory on the tape. n = 1 is the default value. This parameter, when used, must precede the (job #) and file identifying parameters. L denotes the LAST directory.

Element	Definition
LP/TTP	The output device destination for the LIST DIRECT function. The line printer (LP) is the default case.
Notes	Permits the insertion of comments or notes into the directory to improve readability. Notes must be enclosed by double quotes and may contain single quotes (i.e., "// SET DIF '.FIL' ").
(job#)	The job number which subsequent options reference. Must be enclosed in parentheses. Files dumped from a given job will be loaded back to disc under the same job unless NOJOB has been specified during the MAKE or LOAD operation. Must precede the file identifying parameters.
'filnam'	The name of the file(s) to be accessed.
SOURCE DATA OBJECT	Causes all source, object, or data files to be accessed. More than one may be specified.
ALL	When preceded by a (job#) specification, all files under that job are accessed. When a (job#) is not specified on a LOAD, all files under the current directory are accessed. ALL is the default specification when none is entered.
CTRL	Allows TDX command options to be continued on the next line. May be placed anywhere in the first record of the command following MAKE/LOAD. The end of the control records is indicated by '//' from the keyboard or $$ from within a DIF file.
NOVER	Defeats the automatic verification of files during a LOAD operation only.
NOJOB	Defeats the writing to tape of JOB records during a MAKE operation. During a LOAD permits files to be loaded into the current JOB, suppressing any JOB records written to the tape.
OVLY	Permits the file being LOADed to overlay any pre-existing file with the same name.

Element	Definition
CLEAR	Causes files to be deleted which have names matching those of files being loaded. Use on Rev. 10 Sentry software only.
HOLD	TDX execution is terminated following the creation of the DIF file '.TDX' and before any file transfers takes place so that the file '.TDX' may be examined.
FDUMP	Supresses the use of BMT on a MAKE only. FDUMP is used instead. Useful for making 'self loading' tapes to be loaded on systems which do not have TDX or BMT.
//	Terminates the entry of keyboard commands or control records which were accepted due to the CTRL specification.
$$	Equivalent to // but required for TDX commands being executed from within a DIF file.
??	Causes an abort to DOPSY due to an error in a control record.

The TDX command structure is essentially free format with the following exceptions:

The task specification (MAKE, LOAD, etc) must be the first item in the command string.

The DIRECT specification when used must immediately follow the task specification.

EXAMPLE: // TDX MAKE DIRECT....

Each (job#) must immediately precede the file specifications which relate to it.

EXAMPLE: (job#) 'files' (job#) 'files' (job#) 'files'

All other options may appear in any position in the TDX command.

1.12.2.2 TDX EXAMPLES - MAKE/LOAD. Under MAKE a directory of the specified files will first be written to the tape followed by the files themselves in BMT format. Under LOAD the directory is scanned to determine the file locations on tape then the requested files are loaded to disc.

(a) Write files 'TEST1" through 'TEST3' to tape then load them back to disc.

```
// TDX MAKE 'TEST1' 'TEST2' 'TEST3'
// TDX LOAD ALL
```

(b) Write the entire contents of job 'S6D' to tape then load back only the DATA files.

```
// TDX MAKE (S6D) ALL
// TDX LOAD DATA
```

NOTE: (1) The parenthesis are required around the job number exactly as shown.

(2) SOURCE or OBJECT may be substituted for DATA to load source or object files respectively.

(c) Write all source and object files from job 'DIAG' to tape then load back only the object files into the current job 'ABCD'.

```
// TDX MAKE (DIAG) SOURCE OBJECT
// JOB 'ABCD'
// TDX LOAD OBJECT NOJOB
```

(d) Write files from two jobs to tape without the files being preceded by job records on the tape.

```
// TDX MAKE (JOBA) SOURCE (JOBB) DATA NOJOB
```

1.12.2.3 TDX EXAMPLES - MULTIPLE DIRECTORIES. All TDX tapes have at least one directory. The directory is used by TDX to locate files on the tape and contains the largest group of files which may be transferred by a single TDX command. Since the directory may be listed on the line printer it also serves as documentation of a tape's contents. The total number of directories allowed is limited only by the amount of tape available. All the examples in 1.12.2.2 above address the first directory since DIRECT 1 is the default condition. The following examples will address directories 2 and greater.

(a) Write all the source files under job 'S6D' to a TDX tape under the second directory.

```
// TDX MAKE DIRECT 2 (S6D) SOURCE
```

(b) Append a new directory following the last directory of a TDX tape.

```
// TDX MAKE DIRECT L (S6D) OBJECT
```

(c) Load to disc the entire contents of the last directory of a TDX tape.

// TDX LOAD DIRECT L

1.12.2.4 TDX EXAMPLES - LIST DIRECTORY TO OUTPUT DEVICE

(a) List the first directory to the line printer.

// TDX LIST LP

NOTE: LP may be omitted since it is the default case.

(b) List the third directory to the video display unit.

// TDX LIST DIRECT 3 TTP

1.12.2.5 TDX EXAMPLES - TAPE INITIALIZATION

(a) Each new TDX tape must first be initialized as a TDX tape with:

// TDX INIT

1.12.2.6 TDX EXAMPLES - CREATING A DIF VERIFY FILE WITH VALL. The VALL function does not transfer files but rather creates a DIF filed named '.VRALL' which contains VERIFY records for verifying file validity of all files under the specified job. This file is the same as the directory that would have been created on tape on a MAKE. This function is useful to periodically test the validity of a set of files or the entire disc contents.

(a) Create a VERIFY file of the entire disc contents.

// TDX VALL

(b) Create a VERIFY file of job DIAG only.

// TDX VALL (DIAG)

The '.VRALL' file may be executed by:

// SET DIF '.VRALL'

Verify output statements may be directed to the line printer by:

// SET LP

1.12.2.7 TDX EXAMPLES - CHECK THE CONTENTS OF A GIVEN TDX DIRECTORY AGAINST THE CORRESPONDING FILES ON DISC.

(a) Determine which files in a given set have been changed since they were last written out under a given TDX directory.

// TDX CHECK DIRECT n

where n is the desired directory number.

1.12.3 General Comments and Restrictions

(a) Comments in the directory listing should not start with // S, // J, or // V, nor should they contain '←←←← '.

(b) TDX makes use of modified versions of JOB and NOTE which are standard only on SENTRY software REV 10. These programs must be supplied when installing TDX on earlier software releases. The new version of JOB will recognize 77777777B as an equivalent to '←←←←' since the ' ← ' is an illegal character within a DIF file. The new version of NOTE will accept notes written in double quotes as well as single quotes so that notes may contain file name specifications ('filnam').

(c) In using multiple directories, it must be understood that re-making an early directory will result in the loss of all subsequent directories and their files.

(d) When referencing files within the system job the specification (←←←←) may be used. However, when executing a TDX command from within a DIF file, since '←' is an illegal character, the specification (7777) may be used as equivalent to (←←←←).

// TDX MAKE (7777) '=TDX' '=BMT'

will make a TDX tape containing the specified files preceded by the job record: // JOB '←←←←

1.12.4 Additional Documentation

Application Note AD 1069

1.13 BMT

1.13.1 Introduction

BMT (Block Magnetic Tape) is a DOPSY Utility for moving files between disc and tape. It is approximately 20 times more efficient, both in time and in amount of tape used, than FDUMP and CREATE. This is because it moves files in large blocks, rather than 1 card image at a time, and does not fill out card images to make them all 20 words.

1.13.2 Program Usage

The two commands for BMT are:

```
// BMT WRITE 'filename'
// BMT READ 'filename' (OVLY) (STRING/OBJ/DATA(integer))
```

Users of software releases previous to Rev 10 must use:
// UTILITY 'BMT'....

Description:

1. The WRITE form writes a file from disc to tape.

   (a) Parameters must appear in the order given.
   (b) String, object, and data files may be written.
   (c) Coreimage files may not be written.

2. The READ form reads a file from tape and writes it on disc.

   (a) Parameters must appear in the order given.
   (b) The options after // BMT READ are identical to the parameters expected by // CREATE when creating a file from working storage.

3. When executing a READ command, BMT issues the message:

   FILE ON TAPE WAS filename (filetype)

   This allows the user to recover a file from tape under a dummy name and then rename it under its correct name.

4. When executing a READ command, BMT moves a file from tape into working storage and invokes CREATE to create it (see Note 2(b) above). If the CREATE commands are incorrect (e.g., if the user does not specify the correct filetype), CREATE will issue an error message. The user may still be able to create the file from working storage by issuing the correct // CREATE command.

1.13.3 Error Messages

ERROR MESSAGE	DESCRIPTION
FILE ON TAPE NOT IN BLOCKED FORM	The file being read is not correctly formatted for use with BMT. Check that the proper tape is being used and that the tape is properly positioned.

1.14 INIT

1.14.1 Introduction

The purpose of INIT is to optimize TOPSY core space usage by allowing the user to define the test station/test head configuration.

1.14.2 Program Usage

The program is invoked via the command:

```
// INIT
```

This command is used to check or change the stations which may be used in TOPSY. When a customer receives his system TOPSY has already been initialized for the physical test stations at the customer's installation so it may never be necessary to use this command.

On entry to the initializer utility, the stations which may currently be used are listed. This is called system configuration. It then asks if a change is desired. If 'N' for 'no' is entered, the initializer is terminated and control returns to DOPSY. This allows the user to check the configuration without making any change to TOPSY.

If the configuration is to be changed, the '=' prompting character is displayed and the user enters the identifiers for those test stations which he wants to use. The series of entries is terminated when a carriage return is entered on a line without a station identifier.

The initializer then displays the new configuration on the VKT and again asks if any change is desired to allow the user to correct any mistakes. When no further changes are to be made, the initializer asks if TOPSY is to be updated with this configuration. If the user replies "NO", the initializer is terminated and control returns to DOPSY. TOPSY is still configured as it was before. If the reply is affirmative, TOPSY is initialized with the new configuration and control is passed to the TOPSY monitor.

Example:

A system currently configured for stations 1A and 1B is to be changed to 1A, 1B, and 3C.

Example:

```
STAT 1A 1B
CHANGE? (Y/N) Y
STAT
=1A
=1B
=3C
= (Carriage return)
```

```
STATIONS CURRENTLY CONFIGURED:
STAT 1A 1B 3C
CHANGE? (Y/N) N
UPDATE TOPSY ON DISC? (Y/N) Y
TOPSY UPDATED
GOING TO TOPSY
```

In the above example, the underlines indicate the user's entries. The rest of the messages are output by the program.

An identifier for a test station is indicated as "nx" where n is a numeric from 1 to 4 indicating the test station number and x is a letter from A to D indicating the test head selection. Since there is a maximum of four for either the test station number or the test head selection, the identifier may be 1A through 4D.

1.15 INSERT

1.15.1 Introduction

INSERT reads a FACTOR string file and replaces each insert statement with the corresponding string file. On completion the new string file remains in working storage.

1.15.2 Program Usage

The program is invoked via the command:

```
// INSERT
INSERT responds: ENTER FACTOR FILE NAME:
```

The user responds with the file name.

The following message is printed on completion:

ANOTHER PASS ? (Y/ N)

Typing Y will re-execute the program permitting the insertion of nested insert statements. One pass is required for each level of nesting.

Typing N terminates the program with the following message:

XX FILE(S) INSERTED. XXXX CARDS GENERATED. CREATE NEW FILE FROM WS.

The original FACTOR program must be free of any syntax errors with the exception that the final END statement is not required. During execution the names of the files being inserted are printed on the TTP.

1.15.3 Error Messages

ERROR MESSAGE	DESCRIPTION
ERROR--- xxxxx NOT FOUND	The FACTOR file requested or the insert file was not found in the directory. Return to the DOPSY monitor takes place.
ERROR---INSUFFICIENT SPACE IN WS	There is not enough space in working storage on the disc for the new file. Return to the DOPSY monitor takes place.

1.16 NOTE

1.16.1 Introduction

NOTE allows messages to be printed on an output device and allows operator control of the execution of DIF files.

1.16.2 Program Usage

The program is invoked via the command:

```
// NOTE  'string'  (output)  (HALT)  (ANSW)
// NOTE  "string"  (output)  (HALT)  (ANSW)
```

The NOTE command outputs the string text to the specified device. If both the input and output unit are the VKT, the message is not output. If the string is delimited by double quotes, single quotes may be used inside the string as part of the text and if single quotes delimit the message, double quotes may be part of the text.

Specifying HALT causes the CPU to stop. This feature allows time for the operator to do a task requested by a note, such as place a new magnetic tape on the tape drive. Press the CPU START button to continue.

ANSW allows the operator to direct the execution of an input command file. This makes use of the accept-command feature of command entry. When ANSW is specified, the system sends the '>' prompting character to the VKT and waits for a one character input. The character becomes the accept command character and only records with // or /. in column one and two and the accept-command character or a blank in column three will be accepted. Commands with a non-blank character in column three not equal to the accept-command character and all records not beginning with// following a command which is skipped will be ignored. (This allows control records following a CREATE or TDX command to be skipped if the command record is skipped.)

The HALT and ANSW options are available from DOPSY only.

Example 1:

Entering the command

```
// NOTE 'THIS IS A "SPECIAL" COMMAND' LP
```

will cause the line

```
THIS IS A "SPECIAL" COMMAND
```

to be output to the line printer.

Example 2

The coreimage file of $TOPSY may be created with MTIO for standard usage or with DEBUG for a debugging tool. It is possible to create two disc input files and set control to the one which creates the desired system configuration, however one file may be created which allows the operator to select the desired configuration. Its format could be as follows:

Record

```
1     // NOTE 'M = CREATE $TOPSY WITH MTIO' TTP
2     // NOTE 'D = CREATE $TOPSY WITH DEBUG' ANSW TTP
3     //M CREATE '$TOPSY' OVLY COREIM 'MPRO' 214ØB CTRL
4     LOAD BLOBAL FFLTS   )
5     NOLOAD NDEBUG       }  Control records
6     LOAD MTIO           )
7     $$
8     //D CREATE '$TOPSY' OVLY COREIM 'MPRO' 214ØB CTRL
9     LOAD GLOBAL FFLTS   )
10    LOAD NDEBUG         }  Control records
11    NOLOAD MTIO         )
12    $$
13    // SET TTK TTP
```

When this DIF file is set, record 1 and 2 causes the message:

```
M = CREATE $TOPSY with MTIO
D = CREATE $TOPSY with DEBUG
```

to be displayed to the VKT.

Record 2 contains ANSW so following the second line, the system prompts with the ' ' character and waits for input.

If the operator responds with 'M', record 3 is accepted and also the control records 4,5,6, and 7. The command is executed and $TOPSY is created with MTIO and without NDEBUG as the control records specify. Record 8 contains a D in the special character position so its is ignored and all records following it are ignored until the next // record. Record 13 contains a // record with a blank in column 3 so this command is accepted and executed and control is returned to the keyboard as requested.

If following record 2 the operator responds with 'D', records 3,4,5,6, and 7 are bypassed. Record 8 is accepted so the control records are also accepted and $TOPSY is created with NDEBUG (the $TOPSY version of DEBUG). Record 13 is accepted because of the blank character in column 3 and control is returned to the keyboard as requested.

The accept-command character is still the one entered from the DIF file and only records with the accept-command character or a blank in column 3 or without // will be accepted.

1.17 XMIT

1.17.1 Introduction

XMIT is a Utility Program designed to dump the contents of the VKT screen to the line printer. The information printed includes all the characters on the VKT screen regardless of how they got there. This includes prompting characters, TOPSY or DOPSY messages and information entered by the user. XMIT may be executed from either TOPSY or DOPSY and each output is preceded by a Top of Form.

1.17.2 Program Usage

XMIT is executed by either of the two following commands:

From DOPSY:

// XMIT

From TOPSY:

/. XMIT

There are two adjustments which must be made to the Video Keyboard Terminal prior to executing XMIT. These adjustments are the usual defaults case, however, and may be left in effect indefinitely without adversely affecting other VKT usage.

These adjustments are:

(1) Inside the terminal panel control box on the front of the VKT, the switch CR/EOT must be placed into the CR position.

(2) The VKT display made must be set to the foreground mode. This is accomplished by typing a combination of (CONTROL + SHIFT + PERIOD) followed by the combination (CONTROL + SHIFT + O). This is the mode which results in brighter images on the screen and should be the normal operating mode.

1.18 LABEL

1.18.1 Introduction

LABEL is a DOPSY utility program which permits user generated messages to be printed on the line printer or video display unit in a large block character type format. Applications include the prefixing of a program source listing with the program name or other pertinent data, or prefixing datalogged test results with pertinent identifying information. Messages may be up to 9 characters wide and may be continued on subsequent lines.

1.18.2 Program Usage

The command:

```
// LABEL 'message' (LP/TTP)
```

will direct the 'message' to the specified output device.

```
// LABEL 'msg1' 'msg2' 'msg3' (LP/TTP)
```

will cause the three messages to be printed on three individual lines of output.

See the example in Figure 1-1.

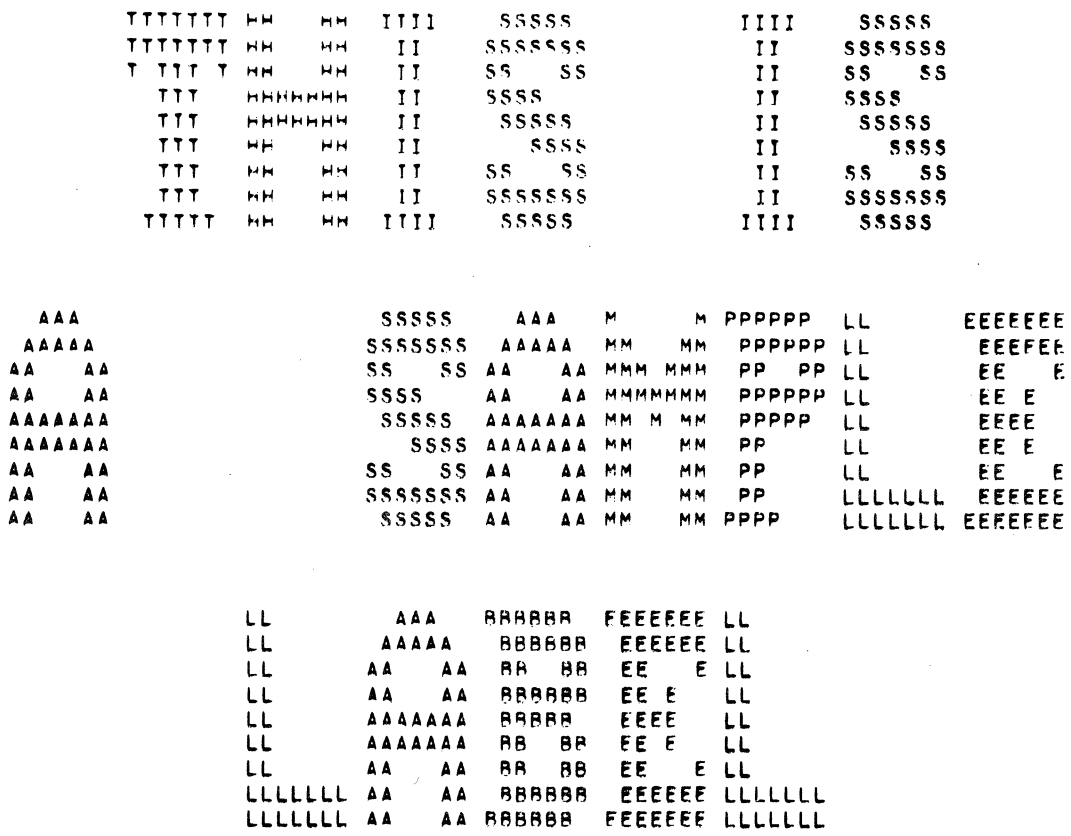


Figure 1-1 Sample LABEL Output Message

SECTION 2

ANALYSIS, CHARACTERIZATION, AND DATALOGGING UTILITY PROGRAMS

2.1 XGRAPH

2.1.1 Introduction

XGRAPH is an Assembly Language Utility program which allows test results and other data to be plotted graphically on either the line printer or the video display unit. XGRAPH is executed by an EXEC statement call from within the user's FACTOR test program. Each individual call to XGRAPH performs a specific function such as output a single line of the graph on the output device, clear and/or open the disc file required for the Composite Shmoo plot, or composite one set of test results upon the previous set. Among the functions currently supported by XGRAPH are:

Shmoo Plots
Composite Shmoo Plots
X-Y Plots
Bar Graphs

2.1.2 Program Usage

XGRAPH usage is described in four sections as follows: Shmoo Plots, Composite Shmoo Plots, X-Y plots and Bar Graphs, and miscellaneous functions.

2.1.2.1 SHMOO PLOT. The calling sequence for the basic shmoo plot is:

EXEC XGRAPH (OP, X, TPAS, IND);

Where OP = 1Ø, for conventional Shmoo plots with X's in Shmoo field.

= 11, for plot where Shmoo field contains alpha-numeric characters corresponding to TASCII codes which user placed in TPAS array. See note 4 below.

X = The numerical value to be printed to the left of the Shmoo line on the vertical (ordinate) axis. A constant, variable, or expression.

TPAS = The name of the array containing the pass (1)/fail(Ø) pattern for the current Shmoo line to be plotted. Typically 51 words in length but may be declared within range of 1 to 67 (see note 2 below).

IND = 0, for standard plot with every fifth location on the horizontal axis delineated by a column of periods, ('.'). The default case if IND is omitted is the same as 0.

= 3, To suppress the printing of any X value (even ∅) on the vertical axis and the '*' which follows it.

= 4, To suppress the printing of the columns of periods which delineate every fifth location on the horizontal axis.

= 5, Produces the combined result of both 3 and 4 above.

SHMOO PLOT PROGRAMMING NOTES

(1) The basic algorithm for producing a Shmoo plot is to place one FOR loop within another FOR loop where the inner FOR loop count equals the declared size of the array minus one. The EXEC XGRAPH statement is placed within the outer FOR loop which has a loop count equal to the number of horizontal Shmoo lines desired; Each EXEC XGRAPH prints one line the length of which is equal to the declared size of TPAS (1-67). Non zero TPAS words print as 'X' and zero words print as blanks.

(2) TPAS must be declared one word larger than the apparent maximum required by the FOR loop count to allow for possible round off error if floating point numbers are used as the FOR loop limits. 51 is the typical value if plots are to be directed to the TTP as well as the LP. TPAS length may be increased up to 67 if only the LP is to be used since the TTP truncates at column 73 and the LP allows all 80 columns. Any number in the range 1 to 67 is legal.

(3) Data to enhance the appearance of the plot such as X or Y axis scaling information must be printed by standard FACTOR statements.

(4) A variation of the Shmoo plot function (OP=11) allows the user to print any character in the Shmoo line simply by placing the correct TASCII code into the desired position in TPAS. The character specified will print instead of the standard 'X'.

2.1.2.2 COMPOSITE SHMOO PLOT. The calling sequence for the Composite Shmoo plot is as follows:

EXEC XGRAPH (12, X, TPAS, IND, TPASCT, XROWS);

Where X = The numerical value to be printed to the left of the Shmoo line of the vertical (ordinate) axis. A constant, variable, or expression.

TPAS = The name of the array containing the pass (1)/fail(0) pattern to be composited upon the previously collected data. Must be declared as 51 words in length.

IND = 1, overlay the current pass/fail pattern in TPAS upon the area of previously collected data specified by TPASCT. See note 5 below.

= 2, output a line of composited Shmoo data according to the area of data specified by TPASCT. See note 5 below.

= 10, open the disc file specified by TPASCT and load its entire contents into the internal core buffer. See note 4 below.

= 20, clear the contents of the internal core buffer and the disc file specified by TPASCT. See note 3 below.

TPASCT = When opening or clearing the disc file, TPASCT is a 2 digit number between 00 and 99 to be used as the last 2 characters of the file name. See note 2 below.

When compositing and printing composited results TPASCT is a number in the range of 50 to 2500 in increments of 50.

XROWS = A number in the range 1 through 50 which determines the number of rows or lines of Shmoo data which are composited and printed along the X (vertical) axis. May be omitted in which case 40 is the default value.

COMPOSITE SHMOO PLOT PROGRAMMING NOTES

(1) The basic algorithm for producing a Composite Shmoo Plot is to place one FOR loop within another FOR loop where the inner FOR loop count equals 50; one less than the required size of the TPAS array which is 51. The EXEC XGRAPH statement is placed within the outer FOR loop which has a loop count equal to the number of horizontal Composite Shmoo lines desired; XROWS.

(2) Before compositing may procede, a disc file must be assigned as follows:

// ASSIGN '%SHMxx' 1250 WORDS DATA

Where xx is a two digit number between 00 and 99. TPASCT takes on the value of this number during the file open and clear operation.

(3) Files just assigned and old files with obsolete data which are to be reused must first be cleared by the IND=20 function.

(4) On the first pass through the program the file must be opened and loaded to core with the IND=1- function, unless the file has just been cleared. An exception to this rule exists when another test station is in use and running a different test program. In this case the file must be opened on every pass through the program.

(5) The disc file into which composited data is accumulated may be visualized as a rectangular matrix 50 units wide x 'XROWS' units high in the image of the plot to be produced. The parameter TPASCT, which is initially set to 50 and then is incremented by 50 for each subsequent XGRAPH call, first references plot 'cells' 1 through 50, then 51 through 100, 101 through 150 and so on; TPASCT 'points to' the last 'cell' of the 50 word file block currently being composited or plotted.

(6) Composite Shmoo plot rules are summarized as follows:

(a) The file '%SHMxx' must be assigned 1250 words.

(b) The array TPAS must be declared as 51 words in length.

(c) TPASCT must start at 50 and be incremented by 50 with each subsequent XGRAPH call.

(d) The disc file must be opened at least once before compositing and before printing the plot.

(e) New files just assigned must first be cleared before compositing begins.

(f) The loop count of the FOR loop containing the EXEC XGRAPH statement must equal XROWS or 40 if XROWS is omitted.

2.1.2.3 X-Y PLOTS and BAR GRAPHS. The calling sequence for X-Y plots and Bar Graphs is as follows:

EXEC XGRAPH (OP,X,Y);

Where OP = 1, FOR X=Y plots where an "X" is printed on the plot at location X,Y.

= 2, FOR Block Graphs where a row of "X's" of length Y is printed at location X.

X = The X axis coordinate which will be printed on the left hand margin of the plot. A constant, variable, or expression. See note 1 below.

Y = The Y axis coordinate which must be normalized to a value between YMIN and YMAX. A constant, variable, or expression. See note 2 below.

X-Y PLOTS AND BAR GRAPH-PROGRAMMING NOTES

(1) The procedure for producing an X-Y plot or Bar Graph is to place the EXEC XGRAPH statement within a single FOR loop which ranges from XMIN to XMAX in increments of X where X = (XMAX-XMIN)/ no. of increments of horizontal rows in the plot; typically 40.

(2) The value Y must be normalized to 50 points between YMIN and YMAX as follows:

(a) Given a value Y to be plotted between YMIN and YMAX.

(b) ΔY = YMAX-YMIN

(c) Y normalized = $Y/\Delta Y$

This is the value to be used as the parameter Y in the calling sequence above.

2.1.2.4 MISCELLANEOUS XGRAPH FUNCTIONS

Line Printer Top of Form

The following statement will force a top of form to the LP if it has been set as the principal output device.

EXEC XGRAPH (0);

Using XGRAPH's Internal Buffer for Auxiliary Data Storage by the FACTOR Program.

The following statement will transfer a given word of data between the FACTOR program and the internal 1250 word buffer within XGRAPH.

EXEC XGRAPH (3, LOC, VAL, IND);

Where

LOC = Buffer address between 0 and 1250 to be accessed.

VAL = Data fetched from or to be stored to buffer location LOC.

IND = 0, places VAL in buffer location LOC.

= 1, Returns data from buffer location LOC to VAL.

2.1.3 Miscellaneous XGRAPH Usage Notes

Directing Output to the Line Printer

All plots may be directed from the TTP to the LP by:

```
// SET LP  , or
/. DATALOG LP STATXX
```

Output may be directed back to the TTP by:

```
// SET TTP      , or
/. DATALOG OFF STATXX , or
/. DATALOG TTP STATXX
```

2.1.4 XGRAPH Error Messages

When a user error occurs XGRAPH will produce the appropriate error message as defined below. The error message will be followed by a conventional TOPSY terminal error 100 messages according to the following example:

```
FILE %SHMXX NOT ON DISK
INST    #   TERMINAL      ERROR
       25       100
```

ERROR MESSAGE	DESCRIPTION
PARAMETER ERROR	Insufficient parameters supplied in the XGRAPH calling statement.
FILE %SHMXX NOT ON DISC	File not assigned on disc, or if assigned name is incorrect.
FILE %SHMXX NOT OPENED	File was not opened prior to compositing or plotting results to output device.
COMPOSITE LIMIT OF 4096 EXCEEDED	Due to buffer size constraints a limit of 4096 composite passes has been imposed.
ARRAY SIZE EXCEEDED	A value for LOC has been used which exceeds 1250, the internal buffer size.
%SHMXX FILE TOO SMALL	File not assigned 1250 words as required.

XROWS PARAMETER ERROR	Value specified for XROWS not within range of 1 to 50.
TPASCT PARAMETER ERROR	TPASCT has value out of range 50 to 2500 or was not incremented by 50.

2.1.5 Additional Documentation

Application Noted AD1037 and AD1087

2.2 TTIME

2.2.1 Introduction

TTIME is a utility program which may be called from FACTOR, which performs a binary search with a timing generator to make a time measurement.

2.2.2 Program Usage

The calling sequence for TTIME is:

EXEC TTIME (TSTART, TSTOP, START, STOP, TG, RESULT);

TSTART = = The starting time delay of the selected timing generator. Must be greater than TSTOP and less than 10 microseconds.

TSTOP = The minimum time delay of a selected timing generator during the binary search. Must be less than TSTART and greater than or equal to 10 nanoseconds.

NOTE: = The rules for usage of TSTART require that the specified function tests must pass at a time delay TSTART and a function fail must occur somewhere in the region between TSTOP and TSTART. Regardless of the polarity of signal being sampled, these conditions can be met by the appropriate use of positive or negative logic and/or control of the mask enable in the local memory.

START = The local memory staring address of the function tests exercised during the binary search .

STOP = The local memory stop address of the function tests during the binary search.

TG = The timing generator to be manipulated by the binary search algorithm. Must be an integer number 1 through 8.

RESULT = A floating point number return to TOPSY from the time algorithm indicating the final time delay of the timing generator being used.

2.2.2.1 PROGRAM DESCRIPTION. After entering the assembly language program, the instruction number and status register values are saved in locations INDSAVE and STATSAVE for restoration when returning to TOPSY. Five variables must be transferred from TOPSY to the assembly language program. The first two variables are the starting and stopping times. These two variables will be divided by FLSB and converted to integer by FFIX. FLSB will make the least significant bit of the times equal to .08 nanoseconds. The other three variables, local memory start and stop and timing generator number are converted to integers. The timing generator is shifted left 21 bits and right by 9 bits to be in the proper location for the long register instructions.

The timing start value is compared with 125,000 which is an equivalent to 10 microseconds, and if it is greater than this number, error return is used. If TSTART passes this test, it is compared with TSTOP and if TSTART≤TSTOP, a second error return is used. TSTOP is then compared with 125 which is 10 nanoseconds, and if it is less than this, a third error return is used. The local memory stop location is written into the L register and the local memory start location is stored with a SET START op. code at the beginning of a DMA buffer. The binary search procedure is shown in the flow chart below (figure 2-1).

If no errors occurred, the exit procedure reads the timing generator being used, converts the result to floating point and transfers it to TOPSY. The routine that programs the timing generator delay (SDELAY) is similar to that used by the TOPSY interpreter. In this case however, it is assumed that the width of the specified timing generator is to remain constant. The delay T is divided by 125 which forms 2 words. The quotient in the E register will have a least significant bit of 10 nanoseconds and is written onto the digital delay of the timing generator. The remainder of the division in the A register has a least significant bit of 80 pico-seconds and 1 is added to this to account for round-off errors. It is then multiplied by 2 and is written into the delay and verbier registers. A time delay of 525 microsecond is allowed for the verniers to settle.

2.2.2.2 CONSOLE SWITCH OPTIONS. Index register 5 is pointing to the beginning of the DMA buffer, and this value is written into the MAR register. The DMA buffer contains a SET START and an ENABLE TEST instruction. Setting Bit 9 of the MODE register starts DMA. After waiting for the DMA test to be completed, the test subroutine exits. If console switch 1 is up, the test subroutine will put the local memory in the continuous loop state. In addition, if console switch 2 is up, the CPU will halt with the A register displaying the coarse or digital timing and the E register displaying the vernier timing.

2.2.3 Error Messages

There are 3 error conditions that may occur when using TTIME. The value of RESULT will indicate errors as follows:

RESULT = 1 if TSTART ≥ 10 microseconds, TSTOP ≤ 10 nanoseconds, or TSTART ≤ TSTOP.

RESULT = 2 if a functional fail occurs on the first try of the binary search (function fail at TSTART).

RESULT = 3 if no function fail occurs throughout the region from TSTOP to TSTART.

2.2.4 Additional Documentation

Application Note AD 1078

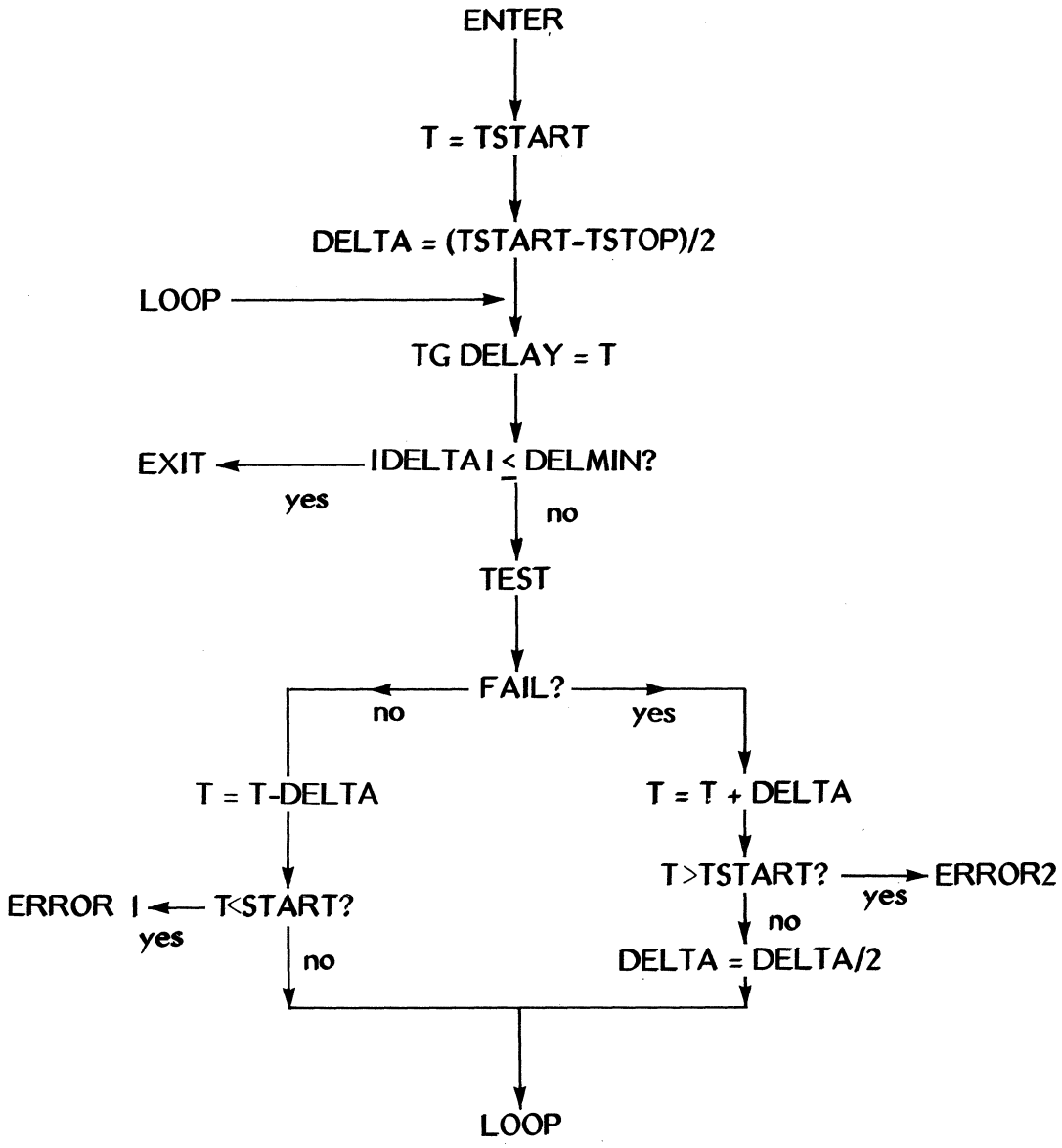


Figure 2-1 Flow Chart of TTIME Binary Search Algorithim

2.3 SPLOT

2.3.1 Introduction

The program SPLOT (sentivity plot) is a diagnostic and device testing tool for use in SENTRY test systems. SPLOT is used in conjunction with an executing device test program to aid in the analysis of the results of device go/no/go functional or dc parametric tests. When called, SPLOT:

1. performs a SPLOT-controlled device test during which two or more of the current device test conditions are selected and varied according to user-specified parameters,
2. initiates periodic DUT pass/fail tests throughout the device test and maintains a record of the results,
3. generates an X, Y shmoo-type graph which displays the pass/fail performance of the device under test when the device is subjected to a user-specified range of test conditions.

The shmoo graph generated as the result of a SPLOT-controlled device test is displayed for analysis by the user either on the system video keyboard terminal (VKT) or, when specified, on a line printer. An example of the type of shmoo graph produced by SPLOT is shown in Figure 2-2. As shown in Figure 2-2, the X, Y shmoo graph produced by SPLOT is plotted as a 21 by 51 element matrix in which the Y-axis component is divided into 50 equal steps. The value for each display step is calculated by SPLOT from required input information.

A detailed description of the shmoo graph format used in SPLOT display operations is given in paragraph 2.3.10.

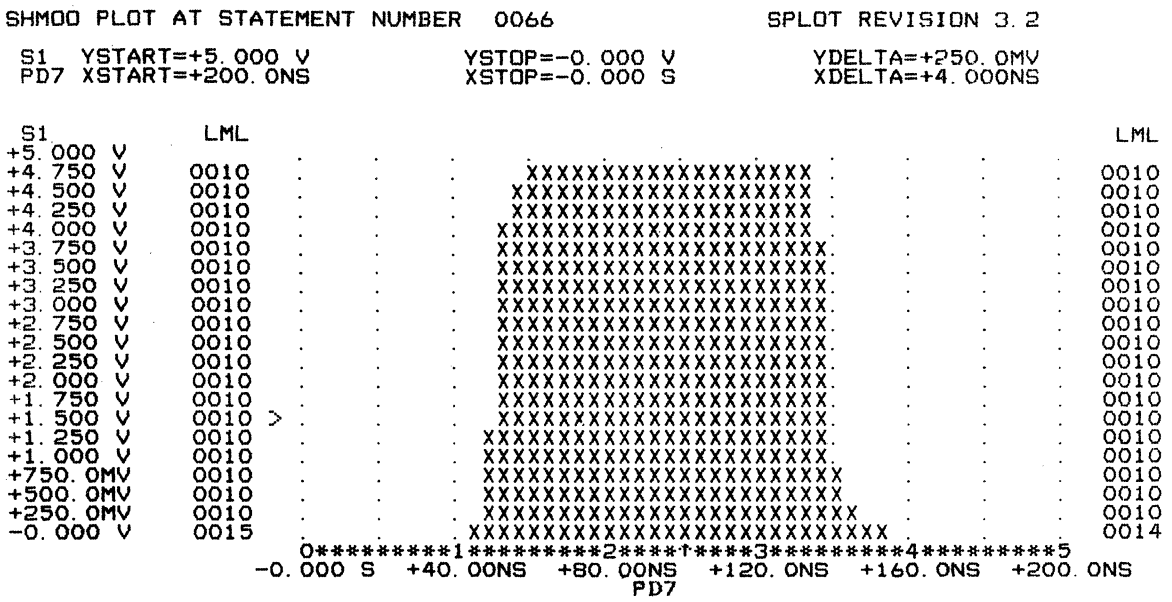


Figure 2-2 Shmoo Graph Produced by a SPLOT Two-Variable Device

To employ SPLOT, the user must call the program and specify:

1. the test conditions to be plotted on the X-axis and the Y-axis of the shmoo graph to be produced by SPLOT,
2. the range over which each selected test condition is to be varied during the SPLOT-controlled test,
3. any permitted graph formatting or handling of the test results.

During a SPLOT-controlled device test, both the specified X-axis and the Y-axis test conditions are set to their range values. The X-axis test condition is then stepped through its 50-step range with an ENABLE TEST or MEASURE PIN statement executed at each step. (MEASURE PIN statement is used only when the DCT register is used to supply one of the SPLOT-controlled test conditions.) The pass/fail result of each test is stored in a line buffer until the X-axis test condition has been stepped through its 50 steps. The contents of the line buffer are then displayed (or printed) as the first (top) line of the test shmoo graph plot. The Y-axis test condition is then set to its next condition and the above process is repeated until the Y-axis test condition has been stepped through its 20-step test range. On completion of its test and display operations, SPLOT restores the selected test conditions to their original states and returns control to TOPSY.

2.3.2 Loading And Executing SPLOT

The program SPLOT may be called and initiated either automatically by the addition of a FACTOR EXECUTE statement to the user program or manually by a call entered at the system VKT during a TOPSY ANALYSIS mode of operation.

The characteristics of the FACTOR statement required to add the SPLOT function to a FACTOR program are described in subsection 2.3.3.

The VKT/user interactive procedure required to initiate the SPLOT function is described in subsection 2.3.8.

2.3.3 FACTOR Call Statement

A FACTOR EXECUTE statement may be included in any device test program to load SPLOT and to initiate the execution of any of the following:

1. a two-variable test in which only one Y-axis and one X-axis test condition (i.e., tester test unit) is selected for the SPLOT-controlled device test (refer to paragraph 2.3.4),
2. a multi-variable test in which one or more Y-axis test unit(s) and one or more X-axis test unit(s) are selected for use in a SPLOT-controlled device test (refer to paragraph 2.3.5),
3. a "return test results" operation in which the test results obtained during a SPLOT-controlled device test are returned to the calling program (refer to paragraph 2.3.7),

4. a shmoo graph "customization" operation in which user-specified names and values are displayed (or printed) on the shmoo graph produced by a SPLOT-controlled device test (refer to paragraph 2.3.6).

The general form the FACTOR statement required to load and initiate SPLOT is:

EXEC SPLOT (Yunit,Ystart,Ystop,Xunit,Xstart,Xstop);

NOTE

Each SPLOT call statement must contain a value for each of the parameters shown in the above format statement.

Parameter	**Function**
Yunit	Identify the test unit the outputs of which are to be plotted as the Y-axis components of the test shmoo graph.
Ystart and Ystop	Define the start and stop values of the range through which the output of the selected Yunit is to be stepped during the SPLOT-controlled device test.
Xunit	Identify the test unit the outputs of which are to be plotted as the X-axis components of the test shmoo graph.
Xstart and Xstop	Define the start and stop values of the range through which the output of the selected Xunit is to be stepped during the SPLOT-controlled device test.

The values which may be assigned to the SPLOT call statement parameters are described in paragraphs 2.3.3.1 and 2.3.3.2.

2.3.3.1 TEST UNIT PARAMETERS - The Yunit and Xunit call statement parameters may be constants, variables, array elements or array names. With one exception, the Yunit and Xunit must either specify or represent one of the 32 possible identifiers for the system test units (refer to Table 2-1). The exception is when a parameter is used to specify a special name or value which is to appear on the test shmoo graph; this use is described in paragraph 2.3.6.

TABLE 2-1 SYSTEM TEST UNIT NUMERIC IDENTIFIERS

Identifier	Unit	Identifier	Unit
1	E0	17	PD1
2	E1	18	PD2
3	EA0	19	PD3
4	EA1	20	PD4
5	EB0	21	PD5
6	EB1	22	PD6
7	EC0	23	PD7
8	EC1	24	PD8
9	S0	25	PW1
10	S1	26	PW2
11	DPS1	27	PW3
12	DPS2	28	PW4
13	DPS3	29	PW5
14	PMU	30	PW6
15	DCT	31	PW7
16	TR	32	PW8

NOTE

Only the test units which are initialized for use in the device test program in which a SPLOT call is to be included may be specified as an X or Y test unit in that call.

2.3.3.2 TEST RANGE PARAMETERS - The Ystart, Ystop and Xstart and Xstop call statement parameters may be constants, variables, array elements or array names. With one exception, each start/stop parameter pair must define the actual range through which the output of the corresponding test unit is to be varied during the SPLOT-controlled device test. The exception is when a start/stop parameter pair is used to define a range of values which are to appear on the test shmoo graph but do not necessarily represent the output range of a selected test unit. The foregoing use of a start/stop parameter pair is described in detail in paragraph 2.3.6.

The test range defined by a start/stop parameter must be either equal to or be within the range established for the corresponding test unit by the program in which the SPLOT call statement is located. If any voltage, current or timing value specified in a SPLOT call statement exceeds the program specified range or if an RVS or DPS voltage was specified which was greater than +6VDC or was less than -30VDC, terminal error 103 is issued.

With regard to the shmoo graph, Ystart defines the value assigned to the top end of the Y-axis, Ystop defines the value assigned to the bottom of the Y-axis. In a similar manner, Xstart defines the value assigned to the start (rightmost end) of the X-axis and Xstop defines the value assigned to the end of the X-axis. SPLOT uses the start and stop values assigned to each axis to calibrate the steps indicated along each axis.

SPLOT will recognize the use of a SET VOFFSET statement in the user program from which it is called and will produce corresponding X and/or Y plot parameters which are relative to the given offset value.

2.3.4 Two-Variable Test Operation

In a two-variable test operation, only one X-axis test unit and one Y-axis test unit is selected to produce the variable X-Y test conditions for the SPLOT-controlled device test. The Xunit and Yunit SPLOT call statement parameters for operations of this type may be constants, variables or array elements - not array names.

An example of the type shmoo graph produced by a two-variable SPLOT test operation is shown in Figure 2-2. The FACTOR SPLOT call statement added to a device test program to produce the shmoo graph shown in Figure 2-2 is:

```
EXEC SPLOT (10, 5, 0, 23, 200E-9, 0);
```

2.3.5 Multiple-Variable Test Operation

The program SPLOT permits the user to specify up to ten Xunit and/or Yunit test units and their associated range definition parameters for a SPLOT-controlled device test. The specified test units which will determine the name and values to be printed along the X and Y axis of the resulting shmoo graph are referred to as "primary" test units. Specified test units the outputs of which are to track (i.e., maintain a constant relationship to) a primary test output are referred to as "secondary" test units. One primary test unit and up to nine secondary test units may be assigned to the SPLOT call statement Xunit and/or Yunit parameters.

Since the SPLOT call statement is limited to six test unit and range parameters (i.e., Yunit,Ystart,Ystop,Xunit,Xstart,Xstop), it is necessary to assign the names of previously declared arrays to the statement parameters which are to represent more than one test unit identifiers and set of test range start and stop parameters. The actual values of the desired identifiers and range parameters are then assigned to elements of the declared arrays. When loaded and initiated, SPLOT obtains all assigned test unit identifiers and their respective range values from the previously established arrays.

The following rules must be observed in preparing a FACTOR program for a multiple-variable, SPLOT-controlled device test:

1. The call statement Xunit or Yunit parameter which is to be assigned more than one test identifier must be the name of a previously declared array.

2. The number of elements in an Xunit or Yunit array must be equal to or greater than the number of test units to be assigned to that parameter.

3. Either or both the Xunit and Yunit parameters of the same SPLOT call statement may be arrays. If both parameters are arrays they need not have the same number of elements.

4. The first element of a test unit array must be assigned the identifier of the primary test unit.

5. The range definition parameters associated with a test unit array must also be arrays which have the same number of elements as the test unit array. For example, the statement DCL Xunit 3 ; must be followed by the statements DCL Xstart 3 ; and DCL Xstop 3 ;

EXAMPLE

The following statement sequence illustrates the FACTOR program statements required to specify two test units for the Y-axis test functions of a SPLOT-controlled device test:

```
DCL YUNIT [2], YSTART [2], YSTOP [2]; REM Y ARRAYS;
YUNIT [1] = 10; REM MAKE S1 PRIMARY UNIT:
YUNIT [2] = 25; REM MAKE PW1 SECONDARY UNIT:
YSTART [1] = 5; YSTOP [1] = 0; REM VARY S1 FROM 5 TO 0;
YSTART [2] = 125E-9; YSTOP [2] = 75E-9; REM PW1 RANGE;
EXEC SPLOT (YUNIT,YSTART,YSTOP,23,200E-9,0);
```

The shmoo graph produced by the addition of the sbove program sequence to a FACTOR test program is hown in Figure 2-3.

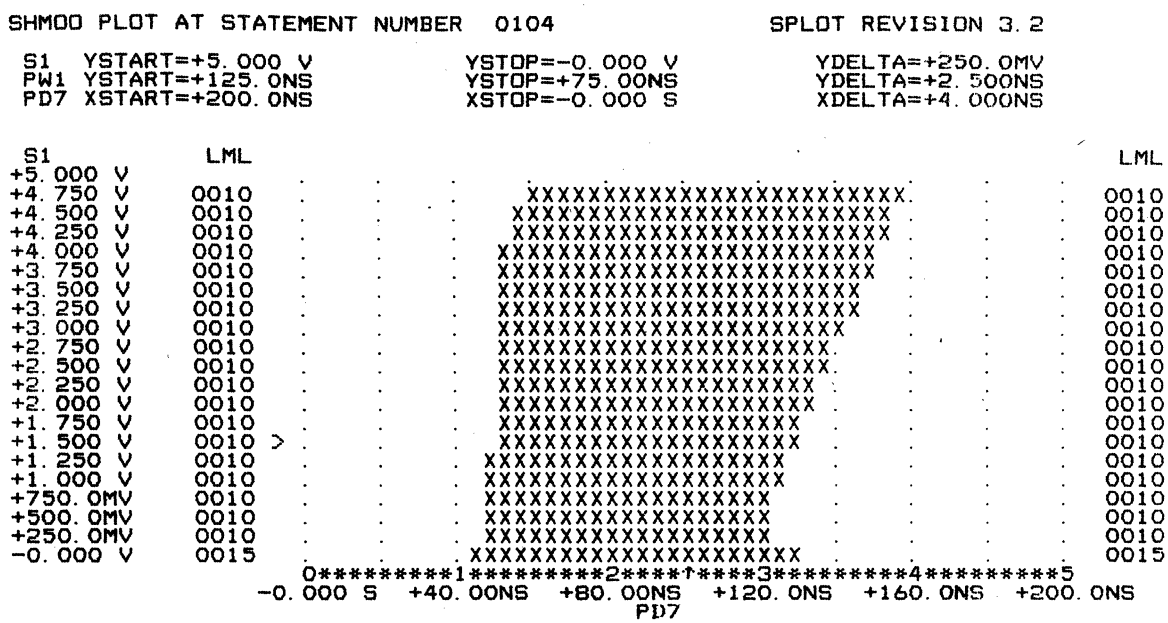


Figure 2-3 Shmoo Graph
Produced By A SPLOT Three-Variable Device Test

2.3.6 Specifying Names For SPLOT Shmoo Graph X And Y Functions

SPLOT permits the user to customize the shmoo graph produced by a SPLOT-controlled device test by specifying literals and selected values for either or both the call statement X and Y test unit and range parameters. The names (literals) specified will be displayed along the corresponding axis of the test shmoo graph. The programming technique required to use this feature of SPLOT, is similar to that required to specify multiple test units and test ranges.

The following rules must be observed in preparing for a SPLOT device test in which user-specified names and range values are to be displayed in the resulting shmoo graph:

1. The SPLOT call statement parameters which are to be assigned a specified literal or value must be declared arrays.
2. The first element of a Xunit or Yunit array must be assigned the desired name (literal). The name specified must consist of from one to four characters with the first character an alphabetic character (A to Z).
3. The values assigned to the first element of an Xunit or Yunit test range start and stop arrays will determine the values which are to be printed along the corresponding axis of the test shmoo graph. Any range of values may be defined, however, it should be meaningful for the test function being represented by the axis of the shmoo graph along which the step values of the range are to appear.
4. When the name feature is used, the engineering units displayed along the X or Y axis of the generated shmoo graph are a function of the first character of the name (literal) given for that axis. The type of engineering unit descriptors used are determined:

The First Character	**Produces the Descriptor**
V	V
I	I
T	S
(other than above)	(none displayed)

EXAMPLE

The following command sequence illustrates the program statement sequence required when both the X and Y axis of the resulting test shmoo graph are to be named.

```
DCL YUN 3 /'VOH',10,25/; REM YUINT NAME AND IDENTIFIERS;
DCL Y2[3], Y1[3]; REM YUNIT RANGE ARRAYS;
DCL XUN [3]/'TPD',23,21/; REM XUNIT NAME AND IDENTIFIERS;
DCL X2[3], X2[3]; REM XUNIT RANGE ARRAYS;
Y2[2] = 5; Y1[2] = ø; REM S1 RANGE;
Y2[3] = 125E-9; Y1[3] = 75E-9; REM PW1 RANGE;
Y2[1] = Y2 2 -10; Y1[1] = Y1 2 -10; REM 10V OFFSET FOR VOH;
X2[2] = 200E-9; X1[2] = 0; REM PD7 RANGE;
X2[3] = 5; X1[3] = 0; REM E1 RANGE;
X2[1] = X2[2]/2; X1[1] = 0; REM HALF SCALE PD7 RANGE FOR TPD;
EXEC SPLOT (YUN,Y2,Y1,XUN,X2,X1); REM SPLOT CALL;
```

Figure 2-4 illustrates a shmoo graph produced by a FACTOR test program which contained the above statement sequence.

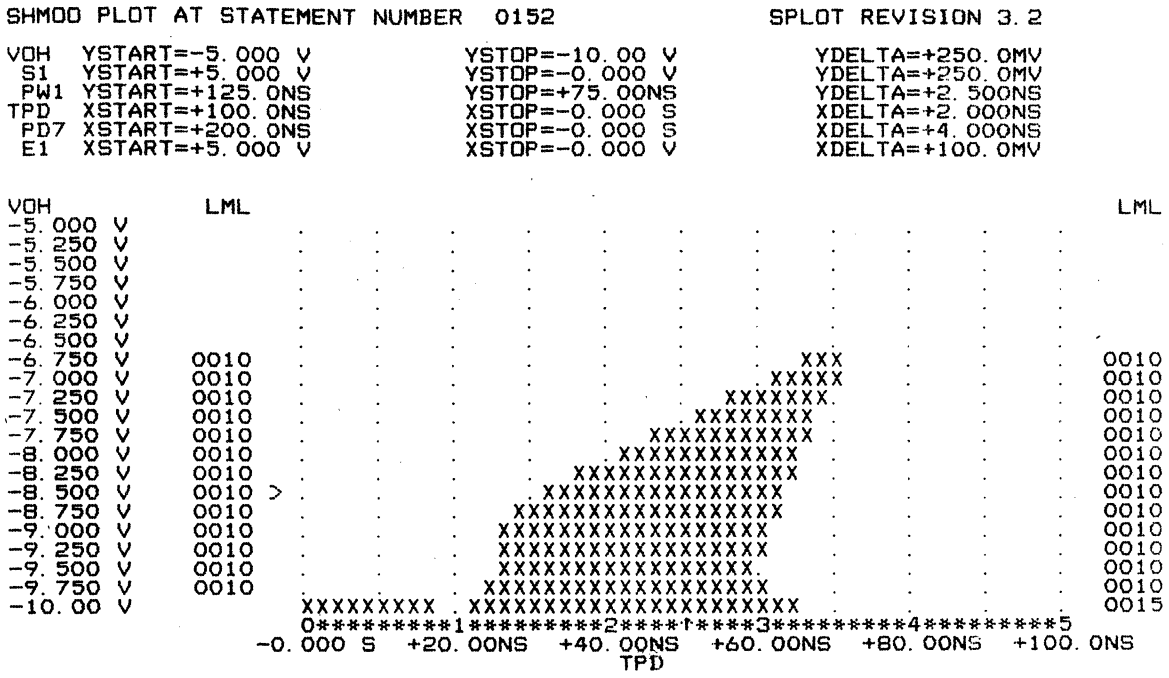


Figure 2-4 Customized Shmoo Graph Produced By A SPLOT Multi-Variable Device Test

2.3.7 Passing Test Results To Factor Test Program

When called from a FACTOR test program, SPLOT can pass the information obtained during the SPLOT-controlled device test back to the calling program. This feature is specified to SPLOT by the addition of the parameter "result" to the FACTOR call statement. The format for this call is:

```
EXEC SPLOT (Yunit,Ystart,Ystop,Xunit,Xstart,Xstop,result);
```

where the parameter "result" is the name of a 96-element array located in the calling test program before the call to SPLOT.

To use the SPLOT "data return" feature, the calling test program must contain:

1. A declaration statement which establishes a 96-element array the name of which is to be used as the "result" operand of the SPLOT call statement.
2. A statement immediately before each SPLOT call which assigns a plot ID number to the first element of the array RESULT,
3. A WRITE statement located after each SPLOT call to cause the information in the array RESULT to be written onto the disc.

The statement sequence needed to set up and use the SPLOT "data return" feature is:

```
DCL RESULT [96];
   .
   .
   .
RESULT[1] = 123; REM PLOT IS NUMBER;
EXEC SPLOT (YUNIT,YSTART,YSTOP,XUNIT,XSTART,
   XSTOP,RESULT);
WRITE (FDOF) RESULT;
   .
   .
   .
RESULT [1] = 321;
EXEC SPLOT (10,4,0,23,100E-9,0,RESULT);
WRITE (FDOF) RESULT;
   .
   .
   .
```

The information contained by each element of a FACTOR test program "data return" array (i.e., RESULT) is described in Table 2-2.

TABLE 2-2 CONTENTS OF A RESULT ARRAY

ELEMENT	MEANING
1	PLOT number ID, defined in FACTOR program
2	Statement number at which SPLOT was called
3	Y unit name in TASCII
4	X unit name in TASCII
5	Y engineering units in TASCII
6	X engineering units in TASCII
7	YSTART in floating point format
8	XSTART in floating point format
9	YSTOP in floating point format
10	XSTOP in floating point format
11	Y current value in floating point format
12	X current value in floating point format
13	First 16 bits of pass/fail information corresponding to first 16 points of the top Y line of the plot. This number is in floating point format.
14	Second 16 bits (points) of first Y line
15	Third 16 bits (points) of first Y line
16	Last 3 bits (points) making a total of 51 points per line.
17-20	
⋮	
93-96	21st and last Y line

2.3.8 Calling SPLOT From the VKT

The program SPLOT may be loaded into CPU memory and initiated via commands entered at the VKT. For this type of operation the user must:

1. Establish PAUSE condition to halt the execution of the current device test at the desired program statement.
2. Load and start SPLOT using a TOPSY ANALYSIS command.
3. Enter via the VKT all desired test parameters in response to SPLOT requests.

Once the input requests of SPLOT are satisfied, the SPLOT-controlled device test is performed and the resulting shmoo graph is displayed.

The procedure required to load and execute SPLOT is described in detail in the following steps:

NOTE

It is assumed that a program PAUSE condition has been established and that the system is in a TOPSY ANALYSIS command mode.

1. The formats of the commands which may be used to load and initiate SPLOT and to specify the device on which the resulting shmoo graph is to be displayed are:

Command	Output Device Specified
/. SPLOT STATxx	Output device is a function of a DATALOG LP/TTP or LP/TTP command
/. SPLOT TTP STATxx	VKT
/. SPLOT LP STATxx	Line printer

2. Once loaded, SPLOT displays, at the VKT, a list of all possible test units and their corresponding numeric identifiers (32).

3. The user must select and enter the identifiers of the test unit which is to control the "Y" test functions.

   The entry must:

   a. be a numeric with the range 1 to 32, inclusive,
   b. represent a test unit specified for use by the interrupted test program,
   c. specify one and only one test unit identifier.

   If the entry does not comply with the above items, an error is indicated at the VKT and the user is requested to enter a new test unit identifier. This error/request cycle is continued until an acceptable entry is made.

4. On completion of item 3, SPLOT displays, at the VKT, the value of the current test output of the selected Y test unit in the format:

   CURRENT test unit name current value

   (e.g., CURRENT S1 +4.900 V)

5. The user is then requested to define the desired test and output range for the SPLOT-controlled test by entering:

   a. the start value of the range (Ystart),
   b. the stop value of the range (Ystop),

The values entered must define a test range which is equal to or within the range selected for the Y test unit in the interrupted device test program. The value given Ystart may be greater than or less than the value of Ystop.

An unacceptable entry causes an error to be indicated and the request for range values to be repeated. This error/repeat cycle is continued until acceptable entries are made.

6. When acceptable Ystart and Ystop entries are made, SPLOT displays, at the VKT:

   a. the name of the selected Y test unit (Yunit),
   b. the specified test range start value (Ystart),
   c. the specified test range stop value (Ystop),
   d. The size of the Y-axis test steps to be generated and displayed on the resulting shmoo graph (YDELTA).

7. On completion of item 6, SPLOT requests the user to enter the identifier for the test unit which is to control the "X" test functions. The entry must:

   a. specify only one test unit,
   b. not be the same as that entered for the Y test unit,
   c. be a number within the range 1 to 32, inclusive,
   d. represent a test unit which is specified for use by the interrupted test program.

   An unacceptable entry causes an error indication and a request for a new entry to be displayed at the VKT. This error/repeat cycle is continued until an acceptable entry is made.

8. On completion of item 7, SPLOT displays, at the VKT, the value of the current output of the selected X test unit. The format of the display is:

   CURRENT test unit name value

   (e.g., CURRENT PW1 +50.00ns).

9. On completion of the foregoing display, the user is requested to define the test unit output range required for the SPLOT-controlled device test by entering:

   a. the start value of the range (Xstart),
   b. the stop value of the range (Xstop).

   The values entered for Xstart and Xstop must follow the same requirement described in step 5 for the Ystart and Ystop parameters.

10. When acceptable test range start and stop values have been entered, SPLOT executes a device test and displays a shmoo graph of the device's pass/fail test results on the selected (see step 1) output device.

The above procedure may be repeated as many times as desired during a program pause to obtain shmoo graphs for any desired combinations of test units and test output ranges.

2.3.8.1 OPERATOR CONTROL INPUTS - The VKT and test station entries which affect the SPLOT load and test operations are:

1. A station RESET entry terminates SPLOT operation and returns control to TOPSY,
2. A CR VKT entry during SPLOT interactive operations terminates SPLOT operation and returns control to TOPSY,
3. A VKT keyboard START entry terminates SPLOT operations and continues the execution of the interrupted device test program.

2.3.9 Using the Pattern Processor For SPLOT Tests

SPLOT permits the user to select the Pattern Processor (PPM) as the functional pattern source for a device test instead of local memory. This feature of SPLOT may be used in either a manual mode (SPLOT called from VKT) or an automatic mode (SPLOT called by the execution of a FACTOR EXEC statement).

When SPLOT is to be loaded and initiated in an automatic mode, the FACTOR call statement must be preceded by:

```
ENABLE PPM;
```

to enter the pattern processor.

When SPLOT is to be loaded and initiated via the VKT, the system must be in a TOPSY ANALYSIS command mode, the current device test program must be in a pause condition and the command:

```
/. WRITE SAMD 101
```

must precede the SPLOT call (i.e., /. SPLOT STATxx).

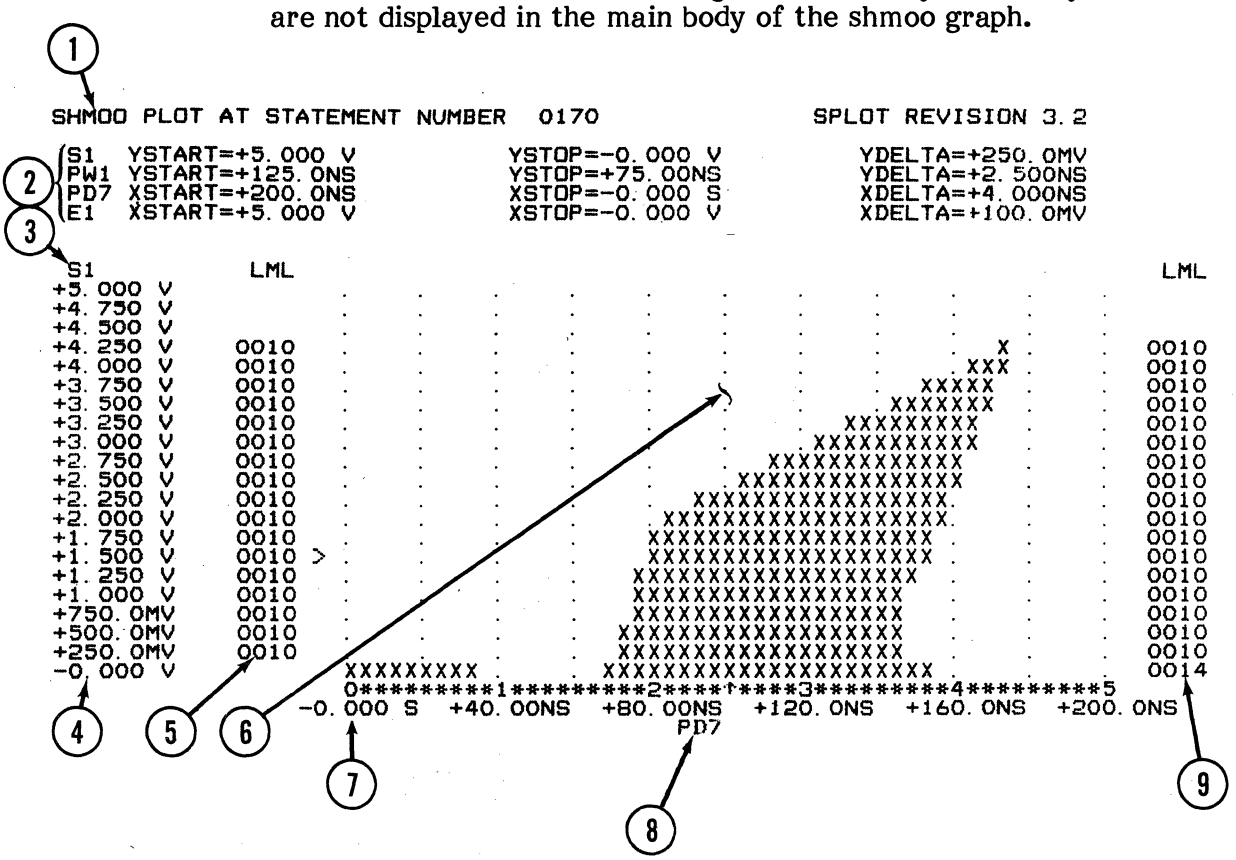
2.3.10 Description of SPLOT-Generated Shmoo Graph

The display of a SPLOT-generated shmoo graph is formatted in the following manner (refer to Figure 2-5):

1. The first (top) horizontal line specifies the number of the user device test program statement which was current when the SPLOT controlled device test was performed and the revision number of the called SPLOT overlay.
2. The next two or more lines specify:
   a. the primary and, when included, the secondary test units the outputs of which were varied to produce the display X-Y plot,
   b. the start and stop values which define the test output range for each of the specified test units,
   c. the value of the change to be made in each test output range for each step as the test is advanced through the 20-step Y-axis test range and the 50 step X-axis test range.

3. The main body of the display which consists, from left to right, of a data column, an address column, a 20 x 50 element plot region, and a second address column.

   a. DATA COLUMN - This column is headed by the designation of the primary Y test unit and contains 20 Yunit test output values which are to be used as the Y-axis components of the plot. The start and stop values which define the test range of the primary Yunit are displayed as the first (top) and last (bottom) values in the column. The names and range values of any secondary Yunits are not displayed in the main body of the shmoo graph.



① Identification of test program statement and SPLOT revision.

② Test unit names, test range parameter values and display step values calculated by SPLOT.

③ Y-axis column heads, Primary test unit name (or a literal if given) and LML column heads.

④ Y-axis step values calculated by SPLOT.

⑤ LML left margin local memory addresses.

⑥ Plot Region (21 x 51 points)

⑦ X-axis scale and step values (one per 10 steps)

⑧ X-axis Primary Test Unit (or literal if given)

⑨ LML right margin local memory addresses.

Figure 2-5 Format of a SPLOT-Generated Shmoo Graph

The symbol is displayed immediately to the right of the value of the data column at which Yunit was set when SPLOT was called.

b. LML (Local Memory Location) COLUMNS - The purpose of these two columns is to specify the point(s), that is the local memory location, at which the plotted test points either entered or left a pass zone. For example, a plot may have, from left to right, a fail region, a pass region, an another fail region. In such a case, the leftmost LML column would contain the address (at each Y-axis step) of the last test fail point to occur before the pass zone was entered; the rightmost column would contain the address (for each Y-axis step) of the first test fail point to occur after leaving the pass zone.

If the plot region begins at the left margin with a pass zone, no LML column is displayed to the left of the plot region.

If the plot region began at the left margin with a fail zone and enters a pass zone which continues through the last X-axis step, no LML column is displayed to the right of the plot region.

NOTE

LML data is not displayed for plots of device tests in which the DC strobe (unit 15) is used for PMU measurements.

LML data displayed for plots of device tests in which the Pattern Processor (PPM) is used as the functional pattern source is invalid.

c. PLOT REGION - This region of the graph consists of a 21 x 51 element matrix which defines the DUT pass and fail points for specific combinations of the X and Y test variables.

An X displayed in the plot region specifies a DUP pass condition for the values of the X and Y variable tester outputs applied to the DUT at that point in the device test. The lack of a displayed X specifies a DUT fail condition.

Vertical lines of dots originating from the X-axis are displayed throughout the plot region at 5-step (X-axis) intervals. These vertical lines are provided to aid in reading the displayed pass/fail information.

A line consisting of asterisks and the numbers 0 through 5 is displayed along the bottom of the plot reqion to provide a scale for the X-axis. The plot position of each of the 50 values through which the X-variable is stepped during the device test is indicated by either an asterisk or, at 10-step intervals, a number. This line also contains an up-arrow symbol ( ) which is displayed immediately under the value at which the primary Xunit was set when SPLOT was called.

The value of the Xunit test output at each of the numbered steps is displayed immediately under the step number. The designation of the primary Xunit selected for the plotted device test is displayed on the last line of the display; it is postioned at the approximate midpoint of the plot region.

2.4 PGLOG

2.4.1 Introduction

PGLOG is a specialized datalogger which prints a 'fail-matrix' map of the RAM test pattern being executed by the Hardware Pattern Generator (HPG) on the POD.

2.4.2 Program Usage

The FACTOR calling sequence is:

EXEC PGLOG;

This statement can be used in place of the ENABLE TEST statement for executing a pattern generator test. A typical program sequence would be to load the pattern, initialize the pattern generator start address and call PGLOG as shown below.

```
PGEN LOAD Ø
WR DIAG;
RD DIAG;
BRANCH RESET Ø;
PGEN START;
ENABLE PGEN Ø, ON;
EXEC PGLOG;
.
.
.
```

If a function failure occurs, PGLOG will set the system fail flag in FAILMM so at end of test the fail lamp will be on. Also, ON FCT, BRANCH will execute if a fail occurred.

To enable datalogging, the TOPSY command

/. DATALOG FCT STATxx N

may be used. (DCT, TRIP and MEASURE can also be included in the command.) If N = Ø, then only the first fail will be datalogged. If N=Ø, then for Solid, Checkerboard or Diagonal type N patterns, the RAM Fail Matrix indicating all failures will appear. For first fail information, PGLOG can presently handle 5 pattern types: Solid, Checkerboard, Diagonal, Ping-Pong and Walking patterns.

2.4.2.1 FIRST FAIL RESULTS. For the Solid pattern, the row and column address and expected data are displayed. For the Checkerboard or Diagonal pattern, the pattern name and failing row and column address are displayed. For a Ping-Pong fail, the failing address, and expected data plus the previously read address and background is displayed. For a Walking One or Zero fail, the failing address plus the location of the one (zero) and background are displayed. For an indeterminate pattern, the failing address (plus or minus two rows) and expected output are displayed.

Since the row size cannot be determined by PGLOG, it is assumed that NROWS* NCOLS = SIZE and NROWS = NCOLS or NROWS = NCOLS/2 where SIZE $=2^i$, i = 2, 3, 4, ..., 12.

For a 4096 address RAM that is organized 32 rows by 128 columns, the failing address information can be converted by subtracting 32 from the row address and doubling the column address.

2.4.2.2 RAM FAIL MATRIX. The RAM Fail Matrix allows datalogging of multiple failures of simple type N patterns for analysis of possible decoder faults. Since the hardware pattern generator does not have an equivalent of the local memory ignore fail function, the following procedure is used:

N is the number of RAM addresses defined in the SET PGEN1 statement and N-1 is loaded into the size register ($0 \leq N-1 \leq 4095$). When the pattern generator starts, counters A and B are automatically initialized with the complement of the size register. For example, with A 1K RAM, the size register has 1777B and $A_o = B_o$ = 6000B. Since only 10 address pins correspond to 1024 addresses, the upper two bits of A_o and B_o have no meaning; hence, the effective starting address is Ø. If the size register is loaded with N-B-1,then $A_o=B_o=B$. Thus, the starting address count can be increased by decreasing the size. When a failure occurs, the original size can be reduced by the failing count plus one. This causes the restart initial count to be one past the previous fail. In this manner, all of the failures of the pattern may be found.

2.4.2.3 PATTERN LIMITATIONS. Because of the pipeline in the pattern generator logic, three cycles will have been executed past a fail. This is accounted for in the PGLOG datalogger. However, this limits the datalogger to certain types of previously defined and predictable patterns. For complex patterns, with several loops, it would be impossible to work backwards three cycles to determine the failing instruction.

2.4.3 Additional Documentation

PGLOG Application Note AD 1048

2.5 PPLOG

2.5.1 Introduction

PPLOG is a specialized data logger which produces, on the specified output device, a 'failmatrix' map of the RAM test pattern being executed by the Pattern Processor Module (PPM). Execution is accomplished within the TOPSY environment and requires no modification of the FACTOR test program. PPLOG will datalog both passing and failing devices. Patterns currently supported are basic type N and type N walking patterns, without refresh. Also supported is a program 'trace' option to aid in the development and debugging of PPM programs, and to datalog test results in a tabular format.

2.5.2 Program Usage

The complete command for executing PPLOG is given below. Although there are many parameters, they are all optional for most patterns. Their application is discussed in detail below.

PPLOG is executed as a TOPSY or Manual Analysis command by pausing at the FACTOR program REXEC statement which corresponds to the pattern to be plotted.

```
/. PAUSE ON FAIL  or,
/. PAUSE ON XXXX  (statement no.)
press START, then

/. PPLOG [ROW n/COL n/STEP n/WALK n/ALL/FAIL n/TOPO/c]
         [TTP/LP/MTW]
```

Where:

ROW n
COL n — Defines maximum rows and columns. n 64 for columns, n 256 for rows. Optional, as PPLOG will compute rows and columns but with an increase of 30% to 50% in processing time.

STEP n — For walking type N patterns only. n defines a PPM test step (count) encompassed within the 'read' portion of the iteration to be plotted. n must be calculated properly or results will be unpredictable. The chart below illustrates the range of the PPM test step for two sequences of a walking diagonal run on a 1024 bit RAM.

Test Function	PPM Test Step-Range
DGEN statement for all zeroes pattern	1
Write zeroes sequence	2-1025

Test Function	PPM Test Step-Range
DGEN statement for diagonal pattern	1026
Write diagonal sequence Ø	1027-2050
Read diagonal sequence Ø	2051-3074
Write diagonal sequence 1	3075-4098
Read diagonal sequence 1	4099-5122

If a fail map of walking test sequence 1 is desired, a PPM step in the range 4099 to 5122 must be chosen for n. Note that n must be within the 'read' portion of the walking sequence. For programs utilizing the classic structure illustrated above in which the device is initially cleared to zeroes, and the entire RAM written once and read once during each iteration of the walking pattern, the following expression may be used to calculate the PPM step count n corresponding to the first cell read during walking iteration W. W=0 corresponds to the very first walking test sequence, W=1 corresponds to the second and so on. SIZE is the size of the RAM under test.

n = 2 * SIZE (W+1)+3

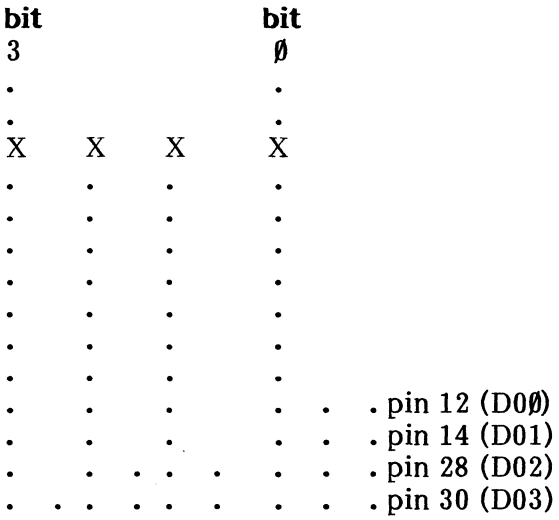
For example, to calculate n for the walking diagonal sequence 1, as illustrated above, in which the PPM step range was 4099 to 5122, we use W=1:

n = (2 * 1024 (1+1))+3 = 4099

See Figures 2-7A and 2-7B for sample plots of a walking diagonal.

WALK n — For walking type N patterns only. Similar to STEP, above, except that PPLOG computes the step count according to the above equation. n is the walk position desired. n = 1,2,etc. Will produce valid results only when used on programs with the classic program structure described under STEP above. See Figures 2-7A and 2-7B for a sample plot of a walking pattern.

ALL — Causes the pass/fail status of each of the four output pins to be included in the failmap. This is accomplished by overprinting a hexadecimal character 1 thru F over the failing '.' or '-' designation. Applies only when LP is specified as the output device. The hexadecimal character is to be interpreted as a four bit mask field in which the bits set indicate which output pins failed, as follows:



The ALL feature uses working storage on disk as a repository for fail data. Therefore, an increase in processing time should be expected, especially with large RAMS with a high percentage of failing cells. See figure 2-8 for an example of the ALL option.

FAIL n — Suppresses the printing of the background pattern in the failmap and allow the status of only the failing device cells through step n to be logged. This reduces processing time by about 60%. If n is omitted, all fails in the program will be logged.

TOPO — For use when the topological scrambler has been enabled within the PPM program. Results in the failmap displaying an unscrambled pattern (i.e. a diagonal looks like a diagonal). The row and col addresses appearing on the failmap are scrambled, however, to maintain an accurate representation of the generated pattern. Without the TOPO parameter, the pattern will appear scrambled as directed by the PPM TOPO statements. See Figure 2-12 for examples of scrambled and unscrambled fail maps.

c — Represents an integer, typically 2,3,4, etc., which is the cycle count of the number of read/write cycles per cell for the test iteration being plotted. Required if the cycle count is greater than 1, as 1 is the default value. In this example:

W-R-R/W-R-R/W-R-R. . . . . .

there are 3 cycles per cell.

TTP
LP
MTW

Determines the destination of the printed output. The TTP is the default case if no other specifications are made. With MTW output, a tape mark follows each individual failmap. These parameters will override the TOPSY and DOPSY POD commands which may also be used to determine the output device as follows:

```
// SET LP
/. DATALOG FCT LP STATXX
```

The above parameters may appear in any order in the PPLOG command. Parameters which include numbers must be entered as shown with the number immediately following the word: i.e., STEP 4. It is acceptable, however, to abbreviate parameters down to their first two letters, to reduce the number of key strokes: 'ROW 32' may be entered 'RO32'. I/O device parameters must appear as shown.

2.5.3 Trace Option

A PPM trace option is provided as an aid in the development and debugging of PPM programs, and as a means of datalogging test results in a tabular format. The trace output contains all available, varying PPM data pertaining to the program currently running. The trace start and stop step counts may optionally be specified in the command. Additionally, the user may specify whether all cycles or only failing cycles are to be included in the output. Unless specified otherwise, the printout will begin with the first read instruction and run continuously to the end of the program, or until RESET is pressed. Only read cycles will be included. The sample output in figure 2-11 describes the data format and content more fully. The 'fail matrix' map is not printed.

The trace is enabled by:

```
/. PPLOG TRACE [start/stop/XRn/FAIL] [TTP/LP]
```

where:

start
stop

Denotes the beginning and ending step count to be included in the trace output. Both or stop may be omitted in which case start defaults to Ø and stop defaults to the end of the program.

XR n

Defines the number (1-3) of the PPM index register to be included in the trace output. Defaults to 1 if omitted.

FAIL

Causes only failing tests to be included in trace output. All read cycles within 'start/stop window' will be traced if omitted.

2.5.4 Error Messages

The following error messages are displayed on the TTP when appropriate. Some error conditions are terminal and result in an immediate error message before processing is terminated. Other error conditions produce an error message immediately following the printing of the failmap. In these cases, the failmap may or may not accurately depict the pattern executed.

NON TERMINAL ERROR MESSAGES	DESCRIPTION
XMASK/YMASK ERROR	Incorrect mask values specified in PPM or FACTOR program (PPLOG reads back incorrect X/Y address from hardware). Also may result from a read of RAM rather than a write.
FLOATING OUTPUT, STEP, OR XMAX/YMAX ERROR	Open socket or load board not installed; output pin improperly terminated. Incorrect value of STEP specified causing 'overlapping patterns'. Or, incorrect value of XMAX/YMAX specified in PPM or FACTOR program.
ADDR NOT FOUND IN TOPO RAM	The TOPO parameter was specified but the topological scrambler was not properly programmed. Some device addresses not specified.

TERMINAL ERROR MESSAGES	DESCRIPTION
CRAM EMPTY	PPLOG executed at statement other than REXEC statement. The PPM control RAM contains no program.
STEP TOO LARGE	User specified a value of STEP, or PPLOG calculated a value of STEP based on a user specified value of WALK, which exceeds the scope of the PPM program. The PPM step count generated during program execution never reached the user specified value.
C TOO LARGE OR XMAX/YMAX ERROR	User specified an incorrect value of 'c' (cycles per cell per pass over RAM). Or, incorrect value of XMAX/YMAX specified in PPM or FACTOR program.
INSUFFICIENT SPACE IN W/S	There is insufficient space in working storage on disc for use of the ALL option. ALL uses this area to save fail data for each failing device location.

2.5.5 Additional Documentation

Application Note AD 1089

2.5.6 PPLOG Output Examples

The following pages contain examples of typical PPLOG fail maps and a sample program trace listing.

The fail map header contains information intended to enable location of the specific PPM instruction at which the first fail occurred. However, due to a hardware pipeline condition, the CRAM INST # may be offset by a certain amount. The direction and magnitude of offset may vary depending upon whether the failing instruction was executed as a result of a BRU, a subroutine call, or straight line code. See figure 2-11 for additional information on resolving the PPM offset.

```
STATIC              TEST PLAN  93415              SN     12

FIRST FAIL AT:    FACT    CRAM    STEP    ROW    COL
                  INST#   INST#           (X)    (Y)
                  0610     010     2051     0      0

      RAM FAIL MATRIX  . = BAD ZERO, - = BAD ONE

COL             1111111111222222222233333333334444444444555555555566668
     0123456789012345678901234567890123456789012345678901234567890123456
ROW
  0  -.-.-.-.-.-.-.-.-.1010101010101010101
  1  0101010101010101.-.-.-.-.-.-.-.-.-.-
  2  -.-.-.-.-.-.-.-.-.1010101010101010101
  3  0101010101010101.-.-.-.-.-.-.-.-.-.-
  4  -.-.-.-.-.-.-.-.-.1010101010101010101
  5  0101010101010101.-.-.-.-.-.-.-.-.-.-
  6  -.-.-.-.-.-.-.-.-.1010101010101010101
  7  0101010101010101.-.-.-.-.-.-.-.-.-.-
  8  -.-.-.-.-.-.-.-.-.1010101010101010101
  9  0101010101010101.-.-.-.-.-.-.-.-.-.-
 10  -.-.-.-.-.-.-.-.-.1010101010101010101
 11  0101010101010101.-.-.-.-.-.-.-.-.-.-
 12  -.-.-.-.-.-.-.-.-.1010101010101010101
 13  0101010101010101.-.-.-.-.-.-.-.-.-.-
 14  -.-.-.-.-.-.-.-.-.1010101010101010101
 15  0101010101010101.-.-.-.-.-.-.-.-.-.-
 16  -.-.-.-.-.-.-.-.-.1010101010101010101
 17  0101010101010101.-.-.-.-.-.-.-.-.-.-
 18  -.-.-.-.-.-.-.-.-.1010101010101010101
 19  0101010101010101.-.-.-.-.-.-.-.-.-.-
 20  -.-.-.-.-.-.-.-.-.1010101010101010101
 21  0101010101010101.-.-.-.-.-.-.-.-.-.-
 22  -.-.-.-.-.-.-.-.-.1010101010101010101
 23  0101010101010101.-.-.-.-.-.-.-.-.-.-
 24  -.-.-.-.-.-.-.-.-.1010101010101010101
 25  0101010101010101.-.-.-.-.-.-.-.-.-.-
 26  -.-.-.-.-.-.-.-.-.1010101010101010101
 27  0101010101010101.-.-.-.-.-.-.-.-.-.-
 28  -.-.-.-.-.-.-.-.-.1010101010101010101
 29  0101010101010101.-.-.-.-.-.-.-.-.-.-
 30  -.-.-.-.-.-.-.-.-.1010101010101010101
 31  0101010101010101.-.-.-.-.-.-.-.-.-.-
```

Figure 2-6 Sample PPLOG Display of a Failing Checkerboard Pattern. The Header Includes Complete First Fail Information.

```
STATIC              TEST PLAN  93415              SN     14

FIRST FAIL AT:    FACT    CRAM    STEP    ROW     COL
                  INST#   INST#           (X)     (Y)
                  0633     013     2051     0       0


      RAM FAIL MATRIX   . = BAD ZERO, - = BAD ONE

COL                1111111111222222222233333333334444444444555555555566666
      01234567890123456789012345678901234567890123456789012345678901234567890123
ROW
   0  -.
   1  01
   2  00-.
   3  000100
   4  0000-.0
   5  000001
   6  000000-.
   7  0000000100
   8  00000000-.
   9  000000000100
  10  0000000000-.
  11  00000000000100
  12  000000000000-.
  13  0000000000000100
  14  00000000000000-.
  15  000000000000000100
  16  0000000000000001000
  17  00000000000000000.-
  18  000000000000000001000
  19  0000000000000000000.-
  20  00000000000000000001000
  21  000000000000000000000.-
  22  0000000000000000000001000
  23  00000000000000000000000.-
  24  000000000000000000000001000
  25  0000000000000000000000000.-
  26  00000000000000000000000001000
  27  000000000000000000000000000.-
  28  0000000000000000000000000001000
  29  00000000000000000000000000000.-
  30  000000000000000000000000000001000
  31  0000000000000000000000000000000.-
```

Figure 2-7A Sample PPLOG Display of The First Position of a Failing Spiral Pattern (Walking Diagonal)

```
STATIC              TEST PLAN  93415              SN     14

 FACT    CRAM    STEP
 INST#   INST#
 0633     013    10243


      RAM FAIL MATRIX   . = BAD ZERO, - = BAD ONE

COL                 1111111111222222222233333333334444444444555555555566666
     0123456789012345678901234567890123456789012345678901234567890123456789012345
ROW
  0  ..0.0...0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1...0
  1  ...0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.-.0.0
  2  .0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0
  3  ...0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.-.-
  4  -.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
  5  0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
  6  .0.0-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
  7  .0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
  8  .0.0.0-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
  9  .0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 10  .0.0.0.0-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 11  .0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 12  .0.0.0.0.0-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 13  .0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 14  .0.0.0.0.0.0-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 15  .0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 16  .0.0.0.0.0.0.0-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 17  .0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 18  .0.0.0.0.0.0.0.0-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 19  .0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 20  .0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 21  .0.0.0.0.0.0.0.0.0.0.-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 22  .0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 23  .0.0.0.0.0.0.0.0.0.0.0.0.-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 24  .0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 25  .0.0.0.0.0.0.0.0.0.0.0.0.0.0.-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 26  .0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 27  .0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 28  .0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 29  .0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 30  0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
 31  .0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.-.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
```

Figure 2-7B Sample PPLOG Display of the Fourth Iteration of a Failing Spiral Pattern (Walking Diagonal). This Map Illustrates the use of the Commands: /. PPLOG STEP 10243 or /. PPLOG WALK 4.

```
STATIC             TEST PLAN  93415           SN     15

FIRST FAIL AT:     FACT    CRAM    STEP    ROW    COL
                   INST#   INST#           (X)    (Y)
                   0633     013    2051     0      0


     RAM FAIL MATRIX   . = BAD ZERO,  - = BAD ONE

COL             1111111111222222222233333333334444444444555555555566666
      0123456789012345678901234567890123456789012345678901234567890123
ROW
  0   66000000000000000000000000000000
  1   01000000000000000000000000000000
  2   00660000000000000000000000000000
  3   00010000000000000000000000000000
  4   00006600000000000000000000000000
  5   00000100000000000000000000000000
  6   00000066000000000000000000000000
  7   00000001000000000000000000000000
  8   00000000660000000000000000000000
  9   00000000010000000000000000000000
 10   00000000006600000000000000000000
 11   00000000000100000000000000000000
 12   00000000000066000000000000000000
 13   00000000000001000000000000000000
 14   00000000000000660000000000000000
 15   00000000000000010000000000000000
 16   00000000000000001000000000000000
 17   00000000000000006600000000000000
 18   00000000000000000010000000000000
 19   00000000000000000066000000000000
 20   00000000000000000000100000000000
 21   00000000000000000000660000000000
 22   00000000000000000000001000000000
 23   00000000000000000000006600000000
 24   00000000000000000000000010000000
 25   00000000000000000000000066000000
 26   00000000000000000000000000100000
 27   00000000000000000000000000660000
 28   00000000000000000000000000001000
 29   00000000000000000000000000006600
 30   00000000000000000000000000000010
 31   00000000000000000000000000000066
```

Figure 2-8 Sample PPLOG Display of a Spiral Pattern in Which the ALL Parameter was Specified, Resulting in the Printing of a Hexadecimal 6 over the Failing Characters. This Indicates That Pins 28 and 14 Failed.

```
STATIC              TEST PLAN  93415              SN     35

 FACT    CRAM    STEP
 INST#   INST#
 0633     014     1923


      RAM FAIL MATRIX   . = BAD ZERO, - = BAD ONE

COL                1111111111222222222233333333334444444444555555555566666
     0123456789012345678901234567890123456789012345678901234567890123456789012345
ROW
  0   1.....................
  1   11.....................
  2   ...1.....................
  3   ...1.....................
  4   ...1.....................
  5   ...1.....................
  6   ...1.....................
  7   ...1.....................
  8   ...1.....................
  9   ...1.....................
 10   ...1.....................
 11   ...1.....................
 12   ...1.....................
 13   ...1.....................
 14   ...1.....................
 15   ...1.....................
 16   ...1.....................
 17   ...1.....................
 18   ...1.....................
 19   ...1.....................
 20   ...1.....................
 21   ...1.....................
 22   ...1.....................
 23   ...1.....................
 24   ...1.....................
 25   ...1.....................
 26   ...1.....................
 27   ...1.....................
 28   ...1.....................
 29   ...1.....................
 30   ...1.....................
 31   ...1.....................
```

Figure 2-9 Sample PPLOG Display Illustrating 'Holes' in the Failmap Resulting From Incorrect Values of XMAX or YMAX, or Other Programming Deficiencies. In This Case the Pattern Representation May Also Be Inaccurate.

```
STATIC              TEST PLAN  93415              SN     31

FIRST FAIL AT:    FACT    CHAM    STEP    ROW     COL
                  INST#   INST#           (X)     (Y)
                  0707     036    1025     0       0

     RAM FAIL MATRIX   . = GAD ZERO, - = BAD ONE

COL              1111111111222222222233333333334444444444555555555566666
     01234567890123456789012345678901234567890123456789012345678901234567890123
ROW
  0  -.
  1
  2    -.
  3
  4      -.
  5
  6        -.
  7
  8          -.
  9
 10            -.
 11
 12              -.
 13
 14                -.
 15
 16                  -.
 17
 18                    -.
 19
 20                      -.
 21
 22                        -.
 23
 24                          -.
 25
 26                            -.
 27
 28                              -.
 29
 30                                -.
 31
```

Figure 2-10 Sample PPLOG Display of a Diagonal Pattern in Which the FAIL Parameter was Specified, Resulting in a 'Fails Only' Map. Maps of Large RAMs may be Produced More Quickly by the use of this Parameter.

STAT1A TEST PLAN 93415 SN 58

	CRA	STEP	X	Y	DATA	XR X1	XR Y1	IF1/2
1	8	2051	0	0	1	3	0	2051
2	9	2052	31	31	1	3	0	1
3	8	2053	1	0	0	4	0	1
4	9	2054	30	31	0	4	0	1
5	8	2055	2	0	1	5	0	1
6	9	2056	29	31	1	5	0	1
7	8	2057	3	0	0	6	0	1
8	9	2058	28	31	0	6	0	1
9	8	2059	4	0	1	7	0	1
10	9	2060	27	31	1	7	0	1
11	8	2061	5	0	0	8	0	1
12	9	2062	26	31	0	8	0	1
13	8	2063	6	0	1	9	0	1
14	9	2064	25	31	1	9	0	1
15	8	2065	7	0	0	10	0	1
16	9	2066	24	31	0	10	0	1

CRA — The control RAM address. This value is obtained from register #126 (alternate bank) and is <u>not</u> pipeline compensated. See note below.

STEP — Total PPM cycle count since beginning of test. This value is obtained from register #1707 in the main bank and #1707 in the alternate bank, and is pipeline compensated.

X,Y — Actual coordinates of cell accessed. Obtained from register #060 and #062 in the alternate bank which are pipeline compensated.

DATA — Actual data generated by PPM according to specified data equation. Obtained from registers #060 and #062 in the alternate bank which are pipeline compensated.

XR Xn
XR Yn — Contents of requested index register (1-3). Obtained from registers #102, #104, and #106 in the alternate bank which are <u>not</u> pipeline compensated. See Note below.

IF1/2 — Contents of requested index register (1-3). Obtained from registers #102, #104, and #106 in the alternate bank. Begins with total step count at initial fail and continues with the number of intervening steps between each read or fail if FAIL was specified. This data is pipeline compensated.

NOTE: As indicated above, the CRA and XR trace data is <u>not</u> pipeline compensated and is, therefore, out of sync with the other data. To correct for this a convenient point should be selected in the X/Y column which corresponds to a particular transition in the XR column, and a line drawn through both points as illustrated above The same principle may be applied to alligning CRA with the rest of the data.

Figure 2-11 Sample PPLOG TRACE Output Listing

```
STATIC              TEST PLAN  P93410              SN      4

FIRST FAIL AT:    FACT    CRAM    STEP    ROW     COL
                  INST#   INST#           (X)     (Y)
                  0244     004     257     6      14


      RAM FAIL MATRIX   . = BAD ZERO, - = BAD ONE

COL               1111111111222222222233333333334444444444555555555566666
     01234567890123456789012345678901234567890123456789012345678901234567890123
ROW
   0  00000000-0000000                          TOPO statements          TOPO 6,14
   1  0000000000-000000                          from PPM program:       TOPO 7,15
   2  00000000000-00000                                                  TOPO 4,12
   3  000000000000-0000                                                  TOPO 5,13
   4  0000000000000-000    Scrambled fail map                            TOPO 2,10
   5  00000000000000-00    produced by basic                             TOPO 3,11
   6  000000000000000-0    command:                                      TOPO 0,8
   7  0000000000000000-                                                  TOPO 1,9
   8  -000000000000000      /. PPLOG                                     TOPO 9,1
   9  0-00000000000000                                                   TOPO 8,0
  10  00-0000000000000                                                   TOPO 11,3
  11  000-000000000000                                                   TOPO 10,2
  12  0000-00000000000                                                   TOPO 13,5
  13  00000-0000000000                                                   TOPO 12,4
  14  000000-000000000                                                   TOPO 15,7
  15  0000000-00000000                                                   TOPO 14,6


STATIC              TEST PLAN  P93410              SN      4

FIRST FAIL AT:    FACT    CRAM    STEP    ROW     COL
                  INST#   INST#           (X)     (Y)
                  0244     004     257     6      14


      RAM FAIL MATRIX   . = BAD ZERO, - = BAD ONE

COL   1111110000000000111122222222233333333334444444444555555555566666
     45230189103254766789012345678901234567890123456789012345678901234567890123
ROW
   6  -0000000000000000
   7  0-000000000000000      Unscrambled fail map
   4  00-00000000000000      produced by TOPO
   5  000-0000000000000      parameter:
   2  0000-000000000000
   3  00000-00000000000      /. PPLOG TOPO
   0  000000-0000000000
   1  0000000-000000000
   9  00000000-00000000
   8  000000000-0000000
  11  0000000000-000000
  10  00000000000-00000      Note Row and Col addresses
  13  000000000000-0000      are scrambled according to
  12  0000000000000-000      TOPO statements above.
  15  00000000000000-00
  14  000000000000000-0
```

Figure 2-12 A Sample PPLOG Display of Scrambled and Unscrambled Failmaps

2.6 DATAIO

2.6.1 Introduction

DATAIO permits file-oriented I/O functions to be performed from FACTOR language programs. Data input/output is permitted to both disc and magnetic tape units using either variable or fixed-length records. Disc I/O may be carried out with files located in working storage, files opened by the user via the VKT, or files opened by the FACTOR program using DATAIO implemented statements.

2.6.1 FACTOR DATAIO Call Statement

The FACTOR statement required to load and initiate DATAIO has the format:

EXEC DATAIO (opcode, IO flag, buffer, device);

Each DATAIO call statement parameter list must contain values for all four parameters.

Parameter	Description
opcode	A constant, variable, or array element the value of which specifies the operation to be performed. The following opcode values are permitted:

Value	Description
0	open file (tape rewind)
1	Read one record from device into buffer.
2	Write one record from buffer to device.
3	Close file (write tape end-of-file mark).
4	Skip N records (+N means go forward, -N means go backward).
5	Position to record N.
6	Skip N filemarks then position after file mark (tape only).
7	Position after filemark N (tapeonly).

IOflag	DATAIO operation status/error return location An integer value is placed into this location by DATAIO to specify the status of the last operation requested by the calling program.

The following IO flag values are permitted:

Value	Condition Specified
0 or positive value	Operation successfully completed.
-1	a. Read operation - end of file reached. b. Write operation - end of file space or end of tape detected. c. File can't be accessed.
-2	Error - needs operator attention or maintenance.
-3	Error incalling statement.
-4	Data overflow.

buffer — A constant variable, or array element which may specify:
1. The name of an array which contains data to be read or written.
2. Record length when opening fixed record length file.
3. The value of N for the operations specified by opcodes 0, and 4 through 7.

device — A constant, variable, array element or array name which specifies the device to be used for the DATAIO operation. The value or array name which may be assigned to "device" is:

Value	Unit Specified
-1	Tape unit 0, the standard magnetic tape unit.
-2	Tape unit 1, the optional magnetic tape unit.
0	File in working storage (Use DOF PMF header).
1	Disc file (DIF PMF opened and closed via TOPSY commands).
2	Disc file (DOF PMF opened and closed via TOPSY commands).
PMF	Name of an array which contains 70 elements or more.

NOTE

The use of the PMF array and of DOF and DIF headers are described in detail in paragraph 2.6.3.

2.6.2 DATAIO Magnetic Tape Operations

Records written to or read from Magnetic Tape are always considered variable-length records. When a record is written, it's size is determined by the buffer size used. The minimum record size is six words. For example, the statement sequence:

```
DCL ARAY [15];
READ (CR) ARAY;
EXEC DATAIO (2, IOFLAG,ARAY,-1);
```

would read a 15-word block from the Card Reader and would write a 15-word record to Tape Unit 0 (Standard Unit) starting at the current tape position. In a similar manner, the statement sequence:

```
DCL ARAY1[20];
EXEC DATAIO (1,IOFLAG,ARAY1,-2);
```

would attempt to read a 20-word record from Tape Unit 1 starting from the current tape position.

A group of records on a tape which are separated from another group by a tape mark is referred to as a "file". Operation code 3 of DATAIO controls the writing of a Tape Mark. Tape Marks are also referred to as "File Marks"; these terms are used interchangeably in the following descriptions.

The magnetic tape operations which may be initiated and controlled using DATAIO are described, individually, in paragraphs 2.6.2.1 through 2.6.2.8. Each description contains:

1. the general form of the FACTOR statement required,
2. a brief description of the operation initiated,
3. a list of the specific error and other conditions which may occur during the specified operation and the IOflag values which may be returned to the calling program.

NOTE

The parameter "device" in the FACTOR DATAIO call statement presented in paragraphs 2.6.2.1 through 2.6.2.8 must be scalar variable or constant which has either the value -1 (standard tape unit) or -2 (optional tape unit).

2.6.2.1 REWIND

General Form:

```
EXEC DATAIO (0,IOflag, buffer, device);
```

Parameter "buffer is not used; it may be assigned either a scalar value or an array name.

Operation Performed:

The magnetic tape unit specified by "device" is issued a Rewind command. DATAIO does not wait for completion of rewind before returning control to the calling program; this, rewind and program operations are overlapped.

IOflag Returns:

VALUE	CONDITION
-2	Selected unit is unavailable.

2.6.2.2 WRITE

General Form

EXEC DATAIO (2, IOflag,buffer,device);

Parameter "buffer" must be the name of a previously declared array.

Operation Performed:

The contents of the array specified by "buffer" are written, as a record, onto the magnetic tape unit identified by "device".

IOflag Returns:

VALUE	CONDITION
0	Successful operation
-1	End-of-Tape (EOT) encountered
-2	Tape unit unavailable or no write ring installed on tape reel.
-3	Error in calling sequence (e.g., buffer is not an array or is less than 6 words in length).

2.6.2.3 READ

General Form:

EXEC DATAIO (1,IOflag,buffer,device);

Parameter "buffer" must be the name of a previously declared array.

Operation Performed:

A record is read from the specified magnetic tape unit into the array identified by "buffer". Differences between the record and array lengths result in specific IOflag returns and DATAIO actions.

IOflag Returns:

VALUE	CONDITION/ACTION
0	a. Record<Array, underflow is not an error condition. All of record is read into the specified array and unused array elements are set to zero. b. Record = Array, normal condition.
4	Record > Array, overflow error condition, the array is filled but the remaining record input words are lost.
-1	The Tape Mark was read, the contents of the array specified by "buffer" ramain unchanged.

VALUE	CONDITION/ACTION
-2	Specified magnetic tape unit was not available.
-3	Error detected in calling statement.

2.6.2.4 WRITE TAPE MARK

General Form:

EXEC DATAIO (3,IOflag,buffer,device);

Parameter "buffer" is not used; it may be assigned either a scalar value or an array name.

Operation Performed:

A tape mark is written onto the tape contained by the specified magnetic tape unit. The Tape Mark is written into the current tape position.

IOflag Returns:

VALUE	CONDITION
-2	Specified magnetic tape unit is unavailable.

2.6.2.5 SKIP N RECORDS

General Form:

EXEC DATAIO (4,IOflag,N,device);

Parameter normally specified as "buffer" is used to specify the number "N" of skips which are to be performed. The parameter "N" must be a scalar variable or constant; it is positive for forward skip operations and negative for backward skip operations.

Operation Performed:

The tape contained by the specified magnetic tape unit is moved from its current position either forwards or backwards over "N" records.

NOTE

The skip command should only be used while reading a previously written tape.

Tape marks are treated as records during skip operation, therefore, it is possible to skip into a different file inadvertantly. The use of a position command avoids this problem for backwards skip operations but requires more time.

IOflag Returns:

VALUE	CONDITION
-1	An End-of-Tape (EOT) mark was encountered during a forward skip operation.

NOTE

A Beginning-of-Tape (BOT) mark encountered during a backward skip operation halts the operation but does not cause an error indication to be returned to the calling program.

-2	Specified magnetic tape unit is unavailable.
-3	An error was detected in the calling statement.

2.6.2.6 POSISION TAPE TO RECORD N

General Form:

EXEC DATAIO (5,IOflag,N,device);

The parameter normally specified as "buffer is used to specify the number "N" of records from the tape BOT that the desired position is located. The parameter "N" must be a scalar variable or constant.

Operation Performed:

The tape contained by the specified magnetic tape unit is rewound to its BOT then it's moved forward through "N" tape marks.

A value of zero assigned to the "N" parameter causes the tape to be positioned to its BOT and to the beginning of record 0 in file 0.

IOflag Returns:

VALUE	CONDITION
-1	EOT encountered during forward skip.
-2	Tape unit unavailable.
-3	Either a non-scalar or a negative value was assigned to "N".

2.6.2.7 SKIP N FILE MARKS

General Form:

EXEC DATAIO (6,IOflag,N,device);

The parameter normally as "buffer" is used to specify the number "N" of file mark which are to be skipped over during the specified operation. The parameter "N" must be a scalar variable or constant; it is made positive for forward operations and negative for backward skip operations.

Operation Performed:

The tape mounted on the specified magnetic tape unit is moved forward or backwards (depending on sign of N) over "N" file marks or until BOT or EOT are encountered.

During backward skip operations the tape is stopped after skipping the "Nth" file mark and is then moved forward back over the file mark until it is in position just before the first record in the Nth file.

IOflag Returns:

VALUE	CONDITION
-1	EOT encountered during forward skip.
-2	Tape unit unavailable.
-3	A non-scalar value was assigned to "N".

2.6.2.8 POSITION AFTER FILE MARK N

General Form:

EXEC DATAIO (7,IOflag,N,device);

The parameter normally specified as "buffer" is used to specify the number "N" of files from the tape BOT that the desired position is located. The parameter "N" must be a scalar variable or constant.

Operation Performed:

The tape contained by the specified magnetic tape unit is rewound to its BOT and then it is moved forward over "N" file marks.

A value of zero assigned to the "N" parameter causes the tape to be positioned to its BOT and to the beginning of file 0 the first file on the tape.

IOflag Returns:

VALUE	CONDITION
-1	EOT encountered during forward skips.
-2	Tape unit unavailable.
-3	Either a non-scalar or a negative value was assigned to "N".

2.6.3 DATAIO Disc File Operations

The following paragraphs describe the disc storage and access requirements that effect DATAIO operations, disc file formats, and the individual disc I/O operations which may be implemented by DATAIO.

2.6.3.1 DISC STORAGE AND ACCESS REQUIREMENTS - The storage area on the disc is referred to as "Peripheral Memory"; files stored in this area are referred to as Peripheral Memory Files (PMF).

Disc PMF access operations require the establishment of file management areas, referred to as "PMF headers", and buffers in CPU memory. The system maintains separate DIF PMF and DOF PMF headers and their associated buffers in CPU memory for use in disc file access operations. Normally, the systems DIF PMF and DOF PMF headers and buffers are used in all disc file I/O operations, DATAIO, however, provides users with the ability to establish their own PMF header and buffer area in CPU memory for any desired input or output operation.

2.6.3.2 USE OF DIF AND DOF PMF HEADERS - Disc file operations in which the DIF and/or DOF PMF headers are to be used require that the file(s) involved be opened using the TOPSY command:

```
/. OPEN DIF 'filename'
```

or

```
/. OPEN DOF 'filename'
```

Since the system maintains separate PMF headers, both DIF and DOF files may be open at the same time.

Once opened by TOPSY commands, DIF and DOF files may be accessed by DATAIO. In such cases, the DATAIO call statement "device" parameter is used to specify which PMF header is to be used:

1. "device" = 1 specifies a DIF PMF header,
2. "device" = 2 specifies a DOF PMF header.

The DOF PMF header is also used by DATAIO for I/O operations with files located in the disc working storage area. Insuch cases the DATAIO call statement "device" parameter is assigned the value 0.

2.6.3.3 ESTABLISHING AND USING A USER-DEFINED PMF HEADER - To define a PMF header, the user must establish an array with a minimum of 70 elements prior to the call for the desired DATAIO operation. The array, referred to as "PMF" for this description, must have the following structure:

Element(s)	Content/Use
PMF[1] and [2]	Filename
PMF[3]	Job number (0 for current job)
PMF[4] through[22]	Reserved for DATAIO use (bookkeeping)
PMF[23] through [70]	Disc buffer (DATAIO use only)

The size of the array (i.e., the buffer area) may be increased in increments of 48 words each. The time required for an overall data transfer is decreased as the size of the array is enlarged since fewer disc access operations are required if more data is transferred during the same transfer period.

The use of a defined PMF header is specified to DATAIO by using the array name of the PMF header as the DATAIO call statement's "device" parameter. For example, to establish a PMF header array named PMF1 for DATAIO operations with the file named PREDAT, the user must include the following FACTOR statements in his/her program:

```
DCL PMF1 [70]/'PRED','AT',0/; REM DEFINE ARRAY;
EXEC DATAIO (0,FLAG,N,PMF1); REM CALL DATAIO;
```

DATAIO uses the defined PMF array as both a bookkeeping area and a disc buffer. The bookkeeping are provided by the PMF array enables DATAIO to perform functions which may not be carried out during magnetic tape operations and DATAIO operations in which DIF PMF or DOF PMF headers were specified. These functions are:

1. Files may be opened by DATAIO.

2. Fixed-record files may be written, read, and positioned with far lower disk and software overhead than variable record files. The record length of a fixed-record file which is to be used for write and/or read operations is set by specifying a scalar value N for the third calling parameter (i.e., "buffer") when opening a file. Files that are to be read-only have their type and record size by DATAIO.

3. File names are not restricted. DOF files opened by TOPSY must have names starting with a period (.). This does not apply to files opened by DATAIO.

4. Files in other Jobs may be accessed. For example, if the current job is 'TEST' and it is required that a file named PREDAT in Job 'DATA' be used, the PMF array would be declared as

   ```
   DCL PMF1[70]/'PRED','AT','DATA'/;
   ```

   Failure to specify a Job causes default to the current Job. It is not possible to access files in System Job.

5. Filenames may be input from the keyboard or other peripherals. The following FACTOR coding will read the filename and Job from either the VKT or PID;

   ```
   DCL PMF1[70]; REM SETUP PMF HEADER ARRAY;
   DCL NAME[2], JOB; REM SETUP JOB NAME ARRAY;
   READ &NAME;
   READ &JOB;
   PMF1[1] = MA,E[1] ;PMF1[2] = NAME[2]; REM NAME TO PMF;
   PMF1[3] = JOB; REM JOB NUMBER TO PMF;
   ```

CAUTION

All words with the PMF Array except the first three are for the use of DATAIO only. Violation of this rule causes unpredictable results which may include the destruction of the disk contents.

2.6.3.4 DATAIO DISC RECORD AND FILE FORMATS - Files written to disc may be of either variable or fixed record length. The disc format for variable-length records is shown in Figure 2-13 of this subsection.

As shown in Figure 2-13, the first word of a variable-length record file is always set to zero to indicate that the file consists of variable-length records. The first word in each subsequent record will specify the size of the record. The size of a variable-length record is specified when the file is opened and the records are written. Variable-length files are not written in a random fashion, they must be written in a serial fashion.

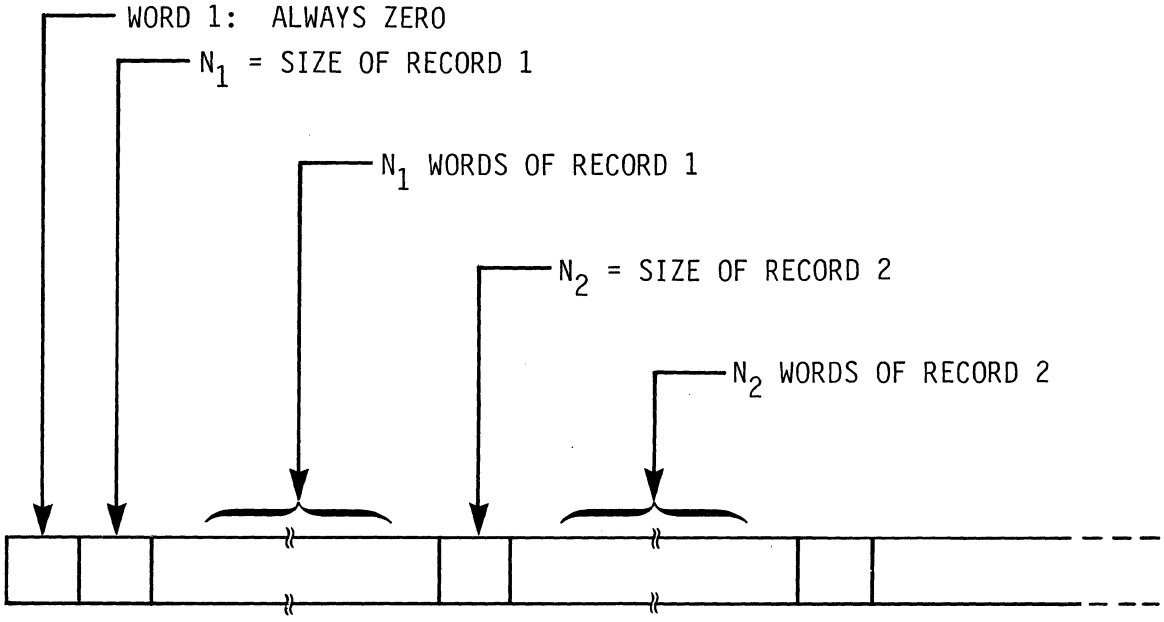


Figure 2-13 Disc Record Format for Variable-Length Records

NOTE

Attempting to write a variable-length file in a random fashion may result in the destruction of all following record length indicators, thus, making the file unreadable. Write operations may be started at a specific record position as long as all subsequent writes are performed sequentially.

The disc format for fixed-length records is shown in Figure 2-14.

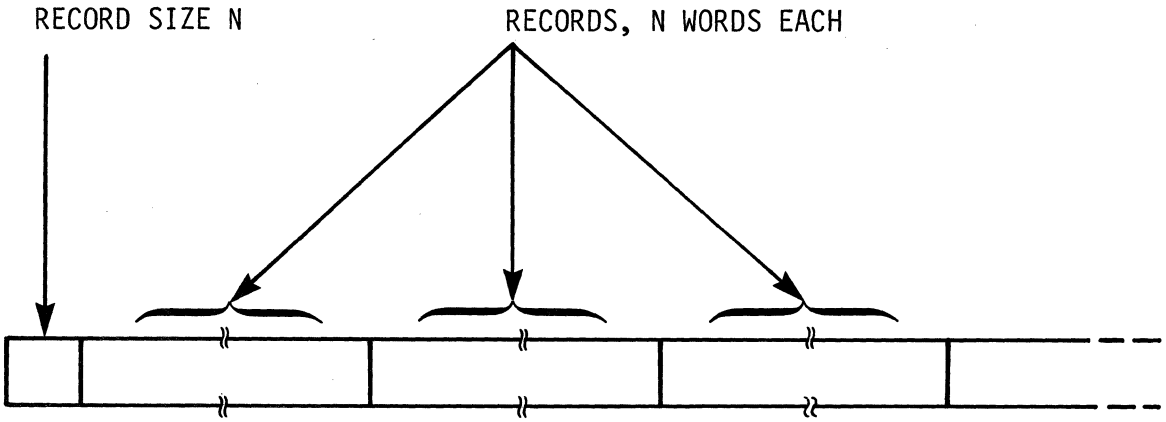


Figure 2-14 Disc Format for Fixed-Length Records

The first word in a fixed-length record file is set to the desired record length by DATAIO. The DATAIO call statement "buffer" parameter is set to the record length desired when a fixed-length record file is opened for its first write operation. Once a file has been written into, subsequent read and write operations may be done in a random fashion by using the positioning DATAIO command.

The mechanism for handling fixed-length files requires a bookkeeping area which must remain available to DATAIO after the file is opened. For files opened using the PMF array option, the PMF array is used for this puspose. Similar space does not exist for siles using the DIF and DOF PMF headers, thus, such files must be written as variable-length files. Fixed-length files may be read using the DIF PMF header but may not be written using the DOF PMF header.

For either type file, the effective length of the file depends on the operation being performed. For a read operation, the length of the file is the highest numbered location previously written. For a write operation, the length of the file is the total assigned file length. When positioning within a file, the file length depends on the previous read or write operation.

2.6.4 DATAIO Disc File FACTOR Statements

The individual operations which may be initiated and controlled using DATAIO are described in paragraphs 2.6.4.1 through 2.6.4.6. Each description contains:

1. the general form of the FACTOR statement required,
2. a brief description of the operation initiated,
3. a list of error and other conditions which may occur during the specified operation together with the IOflag value for each.

NOTE

The parameter "device" in the FACTOR DATAIO call statements presented in paragraphs 2.6.4.1 through 2.6.4.6 must be either a scalar variable or constant, or the name of a user-defined PMF header array. If "device" is a scalar variable or constant, it must have the value-1 (standard tape unit) or -2 (optional tape unit).

2.6.4.1 OPEN FILE

General Form:

EXEC DATAIO (0,IOflag,N,device);

The parameter normally specified as "buffer" is used to specify the length "N" of the record(s) (i.e., number of words) to be contained by the opened file.

Operation Performed:

For "device" = 0 or 1 (DIF or DOF PMF headers), DATAIO checks to ensure that the file had been opened by a previous TOPSY command.

For "device" = 2 (working storage file), DATAIO opens working storage using the DOF PMF header for the required management functions.

For "device" = PMF array name, DATAIO opens the file specified in the array temporarily switching jobs, if necessary.

In all of the preceding operations, the opened file is set to start operations at the first record of the file.

If a record length value (i.e., "N") is given during a PMF array operation, it is saved by DATAIO as a tentative fixed record length. If a value is not given, variable-length records are assumed.

IOflag Returns:

VALUE	CONDITION
0	Operations completed sucessfully
-1	File either not found or can't be opened
-3	PMF array too small.

2.6.4.2 READ

General Form:

EXEC DATAIO (1,IOflag,buffer,device);

Operation Performed:

For read operations with a file specified by a DIF PMF header (i.e., "device" = 1), DATAIO first checks to ensure that the file has been opened; it then reads the file type and record length specifiers from the first word of the file. DATAIO then reads the next consecutive file record into the device buffer identified by the "buffer" parameter.

For read operations in which a PMF header array name was specified by the "device" parameter, DATAIO opens the file named in the array, reads the file type and record length specifiers from the first word in the file, and then reads the next consecutive file record into the device buffer identified by the "buffer" parameter. The file type and record length values read from the file will override any values assumed when the file was opened and will be used in any subsequent write or position operations.

The effective (actually used) length of a file depends on the type of PMF header being used. A DIF PMF header file is opened as an input file; its effective length is the length specified when the file was lat closed at the end of a write operation. A DOF PMF file (including a working storage file) has the entire assigned file space as its effective length. Files using a PMF header array are dynamically allocated. An EOF error condition is returned, via parameter "IOflag", for both PMF header and DIF PMF header files during read operations.

IOflag Returns:

VALUE	CONDITION
0	a. Record < Buffer, underflow condition which is not considered an error. The input characters are read into the buffer and any unused locations are set to zero. b. Record = Buffer, normal condition.
-4	Record > Buffer, overflow error condition. When the buffer is full, excessive input characters are ignored.
-1	End of File reached.
-3	Error inDATAIO call statement.

2.6.4.3 WRITE

General Form:

EXEC DATAIO (2,IOflag,buffer,device);

Operations Performed:

DATAIO checks the file (specified in the header identified by "device") to ensure that it is open. If the file is open and this is the first write to the file, the file type and record length specifications stored in the header being used by the previously executed OPEN statement or TOPSY command are written into the first word of the file. After writing the first file word, if needed, DATAIO transfers the contents of the device buffer areas identified by the parameter "buffer" into the next available disc record.

If a user-defined PMF header array name is passed to DATAIO as the "device" parameter, DATAIO will use the length and type specifications assigned when the file was opened.

If a DOF PMF header indicator (i.e., 0 or 2) is apssed to DATAIO as the "device" parameter, DATAIO always assumes that the file to be written into is a variable-length record file.

Underflow or overflow conditions may occur when writing into a fixed-length record file.

IOflag Returns:

VALUE	CONDITION
-4	Buffer > Record, overflow error condition. When record is filled, excess input buffer characters are ignored.
0	a. Buffer = Record, normal condition. b. Buffer < Record, underflow condition which is not taken as an error. The input buffer characters are written into the record; unused character locations are set to zero.

2.6.4.4 CLOSE FILE

General Form:

EXEC DATAIO (3,IOflag,buffer,device);

Operations Performed:

During close operations, the effective (i.e., used area) size of the file specified by the "device" parameter is written back into the disc directory entry for that file. In closing a working storage file, the file type directory entry is set to "DATA" to permit any subsequent DOPSY CREATE operation to be correctly performed.

The value of the effective file size written into the disc directory is determined by the last performed read or write operation since it consists of the number of the last record read or written.

A file may be inadvertantly truncated if it is closed after being partially read since the new effective size of the file which is returned to the disc directory is the number of the last record read. To avoid truncating a file after a read operation the user may:

1. position the file to its last record before closing it, or
2. not close the file.

A working storage file which is to be referenced by a subsequent DOPSY CREATE command must be closed. A file must also be closed if any data had been written into the file beyond the file's previous last record.

IOflag Returns:

VALUE	CONDITION
-2	File not found, system error.
-3	File was not open.

2.6.4.5 SKIP N RECORDS

General Form:

EXEC DATAIO (4,IOflag,N,device);

The parameter normally specified as "buffer" is used to specify the number "N" of records which are to be skipped over from the current record. The parameter "N" must be a scalar variable or constant; it must be a negative value for backward skip operations or a positive value for forward skip operations.

Operations Performed:

NOTE

This function may be used only for those files which use a PMF header array, since the current record number is known only for files of this type.

In skip operations the pointer used to identify the next file record which may be written into or read may be skipped either backwards or forwards over a specified number of records. The number of records to be skipped and the direction the pointer is moved in are determined by the value and sign of the parameter "N".

IOflag Returns:

VALUE	CONDITION
-1	The EOF was detected, the pointer is positioned past the last file record.
-3	Too large a negative value was assigned to "N". The pointer is positioned before record zero of the specified file.

2.6.4.6 POSITION TO RECORD N

General Form:

```
EXEC DATAIO (5,IOflag,N,device);
```

The parameter normally specified as "buffer" is used to specify the number "N" of the record to which the file pointer is to be positioned. The parameter "N" must be a positive scalar variable or constant which is greater than or equal to zero.

Operations Performed:

The record pointer of the file identified by the PMF header used (i.e., value or name assigned to "device"), is moved to the start of record 0 and then is moved forward until a record number equal to parameter "N" is detected. The pointer is left positioned before the first word of record "N".

Care must be taken not attempt to position to a negative record number or to a record located past the extent of the file.

The extent (amount of area reserved) of a file depends on the type of PMF header used by the file and on the last previous operation performed on the file.

For files which use the DOF PMF header (DOF and working storage files), the entire file is the file extent.

For files which use the DIF PMF header,the used space (i.e., area written into or read from) is the file extent.

For files which use the PMF header array, the file extent is the entire file if the last previous operation was a write. If the previous operation was a read, the file extent is the effective size (record 0 to last record read).

If a positioning operation is performed immediately after opening a file using a PMF header array, the used file space is the file extent.

IOflag Returns:

VALUE	CONDITION
-1	An EOF error is indicated. The value of "N" was too large and the file pointer is positioned beyond the file extent.
-3	An incorrect parameter error is indicated. A negative value was assigned to "N" and the file pointer is positioned before record zero.

2.7 ACCESS

2.7.1 Introduction

The program ACCESS is a FACTOR program analysis tool; it is used to:

1. monitor the execution of a FACTOR program,
2. detect each disc access operation performed during the execution of the monitored program,
3. generate a display which identifies each disc access operation and indicates the amount of data transferred during each access.

The information provided by an ACCESS display enables the user to analyze the monitored program's disc I/O requirements for any possible means of minimizing disc I/O time.

ACCESS is loaded and initiated by the entry of a TOPSY command at the system's video keyboard terminal (VKT). The display generated by ACCESS is always presented at the VKT and, at the option of the user, is printed by the line printer.

2.7.2 Loading and Initiating ACCESS

ACCESS is called from TOPSY by the command:

/. ACCESS(ON/OFF) (LP);

Parameter	Function
ON	Enable ACCESS display to VKT.
OFF	Disable ACCESS display to VKT.
LP	Enable ACCESS display to line printer.

ACCESS is a self-starting core image overlay; once loaded it starts executing immediately.

2.7.3 ACCESS Display Format

Anexample of an ACCESS generated display is shown in Figure 2-15. Each line of an ACCESS display presents information for one of the disc access operations performed in the monitored program. The items listed in each line are:

Item	Meaning
IND	Specifies the octal address of the FACTOR program statement that caused the disc access to be performed.
ADDR	Specifies the octal address of the computer instruction which called DISCIO.
SIZE	Number of words (in octal) which are read from the disc during the access operation.

2.7.4 Computing Access Time

The amount of time required by each disc access reported by ACCESS can be computed using the formula:

$$T=(N \times 9.11 \times 10^{-6}) + (17.5 \times 10^{-3})$$

where:

1. T = time expressed in seconds
2. N = number of words read (i.e., value of SIZE converted to decimal)
3. 9.11×10^{-6} = transfer time per word
4. 17.5×10^{-3} = disc latency time

2.7.5 Installing ACCESS

The program ACCESS may be installed in any system which has any version of TOPSY from revision 9.6 through revision 10.4.

In order to not affect the amount of CPU memory available to the monitored FACTOR program, the ACCESS program, when called, overlays part of (92 words) the resident operating system. The location at which ACCESS is loaded into memory is determined by the user during the installation procedure. Care must be taken in the selection of the ACCESS load point to ensure that the part of the operation system overlayed by ACCESS is not required for the execution of the program to be monitored.

```
IND=  000001    ADDR=  021760    SIZE=  001034
IND=  000002    ADDR=  021760    SIZE=  000520
IND=  000004    ADDR=  021760    SIZE=  000660
IND=  000135    ADDR=  021760    SIZE=  000660
IND=  000164    ADDR=  021760    SIZE=  000660
IND=  000213    ADDR=  021760    SIZE=  000660
IND=  000241    ADDR=  021760    SIZE=  000660
IND=  000270    ADDR=  021760    SIZE=  000660
IND=  000317    ADDR=  021760    SIZE=  000660
IND=  000346    ADDR=  021760    SIZE=  000660
IND=  000374    ADDR=  021760    SIZE=  000660
IND=  000423    ADDR=  021760    SIZE=  000662
IND=  000452    ADDR=  021760    SIZE=  000545
```

Figure 2-15 Example of an ACCESS Printout

2.7.5.1 CREATING ACCESS FROM AN OBJECT FILE - The required procedure is:

1. Load the object file '=ACCESS' into the system JOB ' ←←←← '
2. Determine what area of the operating system is to be overlayed by ACCESS. Table 2-3 presents a list of recommended start addresses for loading the core image form of ACCESS.

TABLE 2-3 RECOMMENDED LOAD ADDRESSES FOR ACCESS

Function	Global	TOPSY Revision 9.6	10.2	10.3	10.4
MTIO	552B	17066B	6646B	6604B	6512B
TAPIO	551B	6263B	4751B	4707B	4615B

NOTE

Global represents the address of the transfer vector to either MTIO or TAPIO. PATCH may be used to interrogate this location in '$TOPSY' to either verify the address or to use any other version of TOPSY.

3. Create a core image of ACCESS which has the user-selected load address using the command:

```
// CREATE ':ACCESS' CORIMAGE '=ACCESS' nnnB
```

where nnnB is the selected load address.

2.7.6 Program Analysis Considerations

A disc read operation is initiated from TOPSY each time that a "page" of a referenced disc data file is to be loaded into memory.

A new "page" load is required whenever the stacks or an ALLINK program causes part of the monitored program to be overlayed. This type of operation takes place after the execution of a DCL, EXEC, or REXEC statement. The analysis of a program using an ACCESS generated display enables the user to analyze the effect that an ALLINK program or an array has on the run time required by a FACTOR program.

A disc read operation is also initiated when TOPSY;

1. reaches the end of the current "page",
2. executes a GOTO or CALL statement which references part of the program which is not in CPU memory.

The reorganization of the FACTOR program to ensure that often referenced loops and subroutines are maintained in CPU memory would minimize disc access time requirements.

The address of the computer instruction (display item ADDR) which calls DISCIO for access is of value when evaluating the effect that an ALLINK program which performs DISC ACCESS in addition to the TOPSY DISC ACCESS (e.g., the program LMLOAD) has on the run time requirements of a FACTOR program. The address of an ALLINK program is normally greater than 30000B.

SECTION 3

DEBUGGING AID UTILITY PROGRAMS

3.1 DEBUG

3.1.1 Introduction

DEBUG is a debugging aid that can be used in a variety of ways to aid a user in testing and debugging. DEBUG allows the user to display or change memory locations or registers, and to halt execution of an assembly language program at specific locations so that memory or registers may be examined or changed. DEBUG can be called directly when running under the DOPSY or TOPSY monitors or it can be loaded with a user's object program as part of the coreimage environment.

3.1.2 Program Usage

DEBUG is invoked by the user in one of the following ways:

(1) It can be loaded with a user's object program to form part of the coreimage version of the user's program. The user creates the coreimage file with this command:

// CREATE 'coreimage filename' COREIMAGE 'object filename' DEBUG

in order to form a coreimage file for later execution, or DEBUG can be included in the coreimage file at execution time by inputting

// EXECUTE 'object filename' DEBUG

The rule of not loading system word 200B must be followed. After loading, control is passed to DEBUG, which echoes with the debug prompting character, which is the number sign (#).

(2) It can be called directly when running under the DOPSY monitor. // DEBUG

(3) It can also be called directly when running under TOPSY if DEBUG was included in $TOPSY creation.
/. DEBUG

(4) When DEBUG has been used to request an address halt and the specified address is executed, DEBUG is entered.

(5) When DEBUG is in memory and has been entered at least once a branch to address 100B causes entry into DEBUG.

Prompting character for DEBUG is '#', signaling it is ready for directives.

3.1.2.1 DESCRIPTION OF DEBUG DIRECTIVE - After entry into DEBUG the DEBUG directives may be entered in order to examine or change memory locations, the A, E, and index registers, or the hardware registers.

The input and output device is always the VKT. A directive is accepted and executed as soon as complete. A carriage return is an illegal entry except in the 'Enter' mode. Any number entered becomes the current address or current location, even if the line is terminated by CTRL-L or RUBOUT. All numbers must be octal, however the B is assumed.

DEBUG DIRECTIVES	DESCRIPTION
L nL n, mL ,mL	List the contents of the memory address indicated. If no number is entered, the contents of the current address is listed. If one number is entered, that address is listed and becomes the current address. If two numbers are entered, the addresses between n and m inclusive are listed. The current address is then m. If ',mL' is entered, the contents of addresses from the current address to m are listed and m becomes the new current address.
line-feed	The current address is incremented and the contents of the new current address is displayed. This allows stepping down a series of memory locations without reentering addresses.
↑	The current address is decremented and the contents of the new current address is displayed. This allows stepping backward through a series of memory locations without reentering addresses.

DEBUG DIRECTIVES	DESCRIPTION
E nE	Directs DEBUG to begin the 'Enter' mode and receive data to change the memory address indicated. If no number is entered the current address is changed. If a number is entered, the contents of that address will be altered and n will be the new current address. The address to be changed is listed and the new data is input. If the data is terminated by carriage return, the 'Enter' mode is ended and new directives may be entered. If the data is terminated by a line-feed, DEBUG stays in the 'Enter' mode, the current address is incremented, and the data entered becomes the contents of the new current address. If CTRL-L is entered, the contents of the address is not changed and DEBUG exits the 'Enter' mode.
0R	Display the contents of the A and E registers. The A register is on the right and the E register is on the left.
X	Display the contents of the right index registers.
nMA	Modify A. The number entered becomes the new contents of the A register.
nME	Modify E. The number entered becomes the new contents of the E register.
n,m MX	Modify index register n. The number m entered becomes the new contents of index register n.
A nA n,mA ,mA	Address halt request. Assembly language program execution stops after executing the instruction at the address indicated and control is given to DEBUG. If n is not specified, the address halt occurs at the current address. If n is specified, the address halt occurs at n and n becomes the new current address. If m is entered, the address halt occurs after the indicated memory address is executed m times, otherwise the address halt occurs at the first execution of n or the current address.

DEBUG DIRECTIVES	DESCRIPTION
	After the entry into DEBUG from an address halt, any of the DEBUG directives may be entered to examine or change the contents of the registers or memory. For example, it may be discovered by displaying the A and E registers that a program is not working because the A register is expected to be zero and is not. The A register can then be changed to the correct contents and program execution continued to verify that that is the solution to the bug. The instruction at the location where the address halt is registered must not be a branch instruction because the instruction is executed first and therefore would transfer control back out of DEBUG. These restricted instructions are: BRU, BOI, BG, BGE, BE, BLE, BL, BNE, BBC, BOS, BSM, BAT, BP, BPZ, BNZ, BN, BNEZ, BO. Only one address halt location may be set at a time.
C	Continue. This directive should be entered only when the entry into DEBUG was caused by an address halt request. This request program execution to continue at the address following the location of the last address halt request. To cause reentry into DEBUG another address halt may be set first.
G nG	Go to the address indicated and execute the assembly language code, i.e., a BRU to the address occurs to transfer control to the address. If n is missing control transfers to the current address, otherwise control transfers to n and n is the current address if DEBUG is reentered.

DEBUG DIRECTIVES	DESCRIPTION
	This directive is generally used when DEBUG is entered because an object file is executed with DEBUG or a corrimage file which has been created with DEBUG is executed. When the execute command is entered the user program and DEBUG are loaded into memory but control passes to DEBUG rather than to the program's PROC statement. DEBUG may then be used to examine the program in memory or to set an address halt. (If an address halt is not set DEBUG can not be reentered except by a branch to address 100B.) When the user is ready to enter the program the address one past the PROC entry point would be branched to via the G directive. The directive could also be used when in DEBUG due to an address halt. To continue the program execution at a location other than the location following the address halt, use the G directive rather than C.
D	Return control to the DOPSY Monitor directly from DEBUG. This directive causes a branch to location 125B. If this directive is used while in TOPSY, the normal TOPSY exit procedure is not followed so that on reentry into TOPSY the system environment is the same as at the last entry into TOPSY rather than that of the last exit.
T	Return control to the TOPSY monitor from DEBUG if currently in TOPSY. If in DOPSY control goes to the DOPSY monitor, i.e., control is passed to the address in the entry point of DEBUG. This directive should only be used when entry to DEBUG has occurred due to operator command.

DEBUG DIRECTIVES	DESCRIPTION
nR	The contents of the tester register n is read and displayed. If $1 \leq n \leq 77$, n specifies a short register. If n is ≥ 200 a long register is read. (Octal is always assumed.) The execute bit (bit 22 of the register) must be in the address of long registers. If console switch 1 is down the read is continuous until carriage return or CTRL-L is entered. If console switch 1 is up, only one read takes place and control returns to DEBUG.
n,m W	Write the tester register n with the data m. If $1 \leq n \leq 77$, n specifies a short register. If $n > 200$ a long register is written into. (Octal is always assumed.) The execute bit (bit 22 of the register) must be on in the address of long registers. If console switch 1 is down the write is continuous until carriage return or CTRL-L is entered. If console switch 1 is up, only one write takes place and control returns to DEBUG.
n,mS	A special SPU is written to tester register n. If m is entered the data is also written at the time of the special SPU. If $1 \leq n \leq 77$, the register is assumed to be a short register, otherwise it is a long register. The execute bit must be set on in the long register's address. If switch 1 is down the special is continuous until carriage return or CTRL-L is entered. If console switch 1 is up, only one read takes place and control returns to DEBUG.
N	Allows a user to type a message. All printing characters can be typed. CTRL-L terminates the command.

3.2 PSCAN

3.2.1 Introduction

PSCAN (Pin SCAN) is a program diagnostic and analysis aid which simplifies the development and documentation of FACTOR programs. PSCAN "scans" all programmed tester pins to establish the complete pin by pin programmed status of the test system. The results are displayed in a tabular format on either the line printer or teletype printer (see Figure 3-1.) PSCAN may be executed either from the keyboard as a TOPSY/MANUAL ANALYSIS command or from an EXEC statement placed in the FACTOR source program.

Following PSCAN execution, tester status etc. and all applicable registers are saved and restored; thus the programmed status of the test system is unaffected by the use of PSCAN.

3.2.2 Program Usage

PSCAN is executed by either of the following two methods:

As a TOPSY or MANUAL ANALYSIS command at a program pause:

/. PSCAN n LP/TTP

Or, as a FACTOR EXEC statement:

EXEC PSCAN (n);

where n is the highest tester pin number programmed. ($n \leq 120$)

PSCAN first decodes all tester registers which define the tester elements (i.e., PMU, DPS1, E0, etc.) which are programmed to pins 1 through n, where n is defined by the user.

The registers for the individual elements are then decoded to determine their current programmed status. For pins which are tied to function test drivers (i.e., E0, E1, etc.) a MEASURE VALUE is performed and the result is printed, in parentheses, directly under the programmed value for the given pin. If a DPS is programmed to a pin (S600 only) a MEASURE VALUE is performed for the voltage reading and a MEASURE NODE for the current. These results are printed, in parentheses, immediately under the programmed status for the DPS (see Figure 3-1). After the nth pin has been processed and printed, the programmed and measured status of all power supplies which are not tied to tester pins are printed at the bottom of the table.

Either of the following commands will direct PSCAN output to the line printer:

```
// SET LP
/. DATALOG LP STATX
/. PSCAN n LP
```

The TTP will receive the output by default if LP is not specified. Once printing begins on the TTP, 26 lines will be printed then the display will halt for inspection. Pressing the LF key will display one additional line at a time. Pressing CR will cause another 26 lines to be printed.

Pressing STATION RESET will abort printing and reset the system in the normal fashion.

3.2.2.1 PMU/DPS/RVS STATUS REPORT - A printout listing the complete programmed and measured status of ALL used power supplies without the "pin by pin" part of the table may be produced with the following command.

```
/. PSCAN 1 LP/TTP
```

3.2.2.2 DESCRIPTION OF PSCAN DISPLAY - Most of the information at the top of the PSCAN display is self-explanatory with the possible exception of LOCAL MEM ADR which refers to the last local memory address executed (see Figure 3-1).

The following describes the contents of the PSCAN table column headings (see Figure 3-1).

PIN	Refers to tester pin numbers 1 through n, where n is defined by the user ($n \leq 120$)
TIED TO	Tester elements which are "tied" to the corresponding pin. PSCAN will present the actual status of the system exactly as programmed whether meaningful or not.
M-D REG	Represents the status of the M and D registers respectively for the corresponding pin.
PROGRAMMED VALUE (MEASURED VALUE)	The programmed status of the corresponding DPS, RVS, TGEN or PMU is expressed. For RVS's the E0 value is printed on the left and the E1 value on the right. The voltage on the corresponding pin is then measured with the PMU. This value is printed under the programmed value for the RVS which is currently tied to that pin based on the F data for that pin in the last SET F executed. For DPS's, the voltage measurement from the node are printed under the programmed voltage and current measurements respectively.

UN-COMMITTED SUPPLIES — The programmed and measured status of the PMU or any RVS or DPS which has not been programmed to a pin will be printed. All measurements are made at nodes.

EXAMPLE: — The sample printout in Figure 3-1 shows that pin 3 is tied to a utility relay and that the M register is set to zero and the D register to one thus pin 3 is also an input pin and is tied to the EB0, EB1 functional drivers which are programmed to 700mV and 1.7V respectively. The voltage on pin 3 was measured and found to be 716mV. EB0 is currently driving the pin so the measured value is printed, in parentheses, under the 700mV programmed value. TGEN2 is also driving the pin and its corresponding delay and width is printed.

The variation between the programmed and measured voltage values will typically be between 0 and ± 40mV and varies with the range and magnitude of the values. The exact variation to be expected is equal to the sum of the RVS accuracy and the PMU voltage sense accuracy and may be determined from the following table:

RANGE	RVS ACCURACY
2	± .1% + 10mV
3	± .1% ± 40mV

MEASURING RANGE	PMU ACCURACY
0 to + 1.023V	+ .2% + 3m
0 to ± 10.23V	± .1% ± 10mV
0 to ± 40.92V	± .1% ± 40mV
0 to -102.30V	± .1% ±100mV

3.2.3 Cautions to User

Although PSCAN usage does not perturb the programmed status of the system in any way, there are certain precautions that should be observed if one is interested in preserving the original PASS/FAIL status thst existed prior to PSCAN usage. Since the PMU is used by PSCAN for measuring the voltage present on all input pins and therefore must be disconnected from its previous programmed connection, PSCAN usage should be avoided in test sequences where the PMU connection must be maintained. The PMU is always returned to its original programmed status on exit from the program. PSCAN usage should also be avoided during dynamic functional test sequences for which critical timing must be maintained. If PSCAN is used in a program which is using RAMPAT (RAM software PATtern generator), the RAMPAT address tables will be destroyed thus negating all RAMPAT calls of the form: EXEC RAMPAT (Ø, . . .) through EXEC RAMPAT (8, . . .) and a functional fail will result. Again these restrictions may be ignored if the PASS/FAIL status of the system is of no interest following PSCAN usage.

PSCAN assumes that pins connected to input drivers are in the NRZ mode and are not referenced in a SET INVERT statement. RZ mode may cause the measured pin voltage to disagree with the programmed value, and inverted data will cause the measured value to appear on the wrong side of the printout (i.e., the measured E0 value will print under the programmed E1 value).

```
STATION NO.     2D
PROGRAM NAME    TEST6
STATEMENT NO.   000073

DC DELAY        +1.015E-02 S                                        DCT   FAIL
PERIOD          +6.000E-04 S                                        FCT   FAIL
PMU CLAMPED   +-1.050E+01 V                                         POS   LOGIC
LOCAL MEM ADR 0000
MA,DA ENABLED

DCT      - - - - - -       DCTO  LT +1.000E-01 V     DCT1  GT +2.000E-01 V
```

PIN	TIED TO	M-D REG	PROGRAMMED VALUE (MEASURED VALUE)			
1	PMU	0 1	FORCE	+8.500E-03 I	SENSE	+1.308E+01 V
	RELAY					
2	RELAY	0 1				
	IN,EO/1			+5.000E-01 V		-1.500E-00 V
						(-1.490E-00 V)
	TGEN1		DELAY	+1.000E-05 S	WIDTH	+1.501E-05 S
3	RELAY	0 1				
	IN,EBO/1			+7.000E-01 V		+1.700E-00 V
				(+7.160E-01 V)		
	TGEN2		DELAY	+2.000E-05 S	WIDTH	+2.500E-05 S
4	CLK	0 1				
	IN,EAO/1			+5.000E-01 V		-1.500E-00 V
				(+4.900E-01 V)		
5	CLK	0 1				
	IN,ECO/1			+4.000E-01 V		+1.400E-00 V
				(+4.130E-01 V)		
6	OUT,SO/1	1 0		+2.000E-00 V		+4.000E-00 V
	TGEN8		DELAY	+8.000E-05 S	WIDTH	+8.500E-05 S
7	OUT,SAO/1	1 0		-0.000E-00 V		-0.000E-00 V
	TGEN7		DELAY	+7.000E-05 S	WIDTH	+7.500E-05 S
	TGEN8		DELAY	+8.000E-05 S	WIDTH	+8.500E-05 S
8	IN,EO/1	0 1		+5.000E-01 V		-1.500E-00 V
				(+5.030E-01 V)		
	TGEN3		DELAY	+3.000E-05 S	WIDTH	+3.500E-05 S
9	IN,EO/1	0 1		+5.000E-01 V		-1.500E-00 V
				(+5.040E-01 V)		
	TGEN1		DELAY	+1.000E-05 S	WIDTH	+1.501E-05 S
	TGEN2		DELAY	+2.000E-05 S	WIDTH	+2.500E-05 S
10	TCOM	0 0				
11	DPS1	0 1	FORCE	+4.000E-00 V	TRIP GT	+1.000E-01 I
				(+4.000E-00 V		+1.023E-01 I)
12	DPS2	0 1	FORCE	+1.000E-01 I	TRIP LT	+5.000E-00 V
				(+1.023E-01 I		+7.760E-00 V)
UN-COMMITTED SUPPLIES						
	DPS3		FORCE	+3.000E-02 I	TRIP GT	+3.000E-00 V
				(+1.023E-01 I		+4.240E-00 V)

Figure 3-1 PSCAN Display

3.3 LMIO

3.3.1 Introduction

LMIO is an Application Utility Program designed to speed test program development and debugging by allowing functional test data to be transferred between Local Memory and a variety of output devices. Execution is accomplished in an interactive mode via the video keyboard and allows the user to repetitively access any section of Local Memory.

LMIO will dump Local Memory contents in one of three different formats to any output device including a disc file. The three formats provided are string, data, and debug (a concise form of string format for program debugging). LMIO also allows the contents of a specified disc file which contains functional test data in 'data' format, to be loaded to Local Memory at any specified address. A function also exists which clears Local memory to zero.

3.3.2 Program Usage

LMIO is executed from the TOPSY environment and typically within Manual Analysis at a program pause in the FACTOR test program. The pause being established by either command as follows:

```
/. PAUSE ON FAIL
/. PAUSE ON XXXX (statement no.)
```

However, it is not required that a test program be loaded or a test station be on line to execute LMIO. An example of this mode of operation would be the alternate loading of Local Memory from disc and then saving in different disc files in order to create larger files which are composites of other files.

Since LMIO performs the two separate functions of dumping and loading Local Memory, each function will be described as a separate command.

3.3.2.1 DUMPING LOCAL MEMORY CONTENTS. To dump Local Memory contents in a specified format to a specified output device, the command is:

```
/. LMIO [DATA/STRING/DEBUG] [DISC/LP/TTP/MTW]
```

LMIO will respond:

```
ENTER:
START STOP RANKS FILE (// TO QUIT)
=
```

The user next enters the first and last Local Memory address to be dumped, the number of ranks desired (1 through 8), and if DISC was the specified output device, the I.D. of the desired disc file (see 'filnam', in section 3.3.2.2 for file naming conventions). Values are accepted as octal when followed by a 'B'.

The format and output device parameters are defined as follows:

DATA	Function test data in tester executable format as output by the compiler. DISC is the default and the only allowed output device for this format.
STRING	Original source format in which test data appeared prior to compilation. Will include all SPM instructions, if any, and any Local Memory labels referenced by the '-LGOTO' statement. This format is compatible with the compiler for recompilation. See example in Figure 3-2.
DEBUG	Similar to STRING format but condensed such that each line of output contains all programmed data relating to the current Local Memory location (except that pin data is printed on subsequent lines when more than two ranks are specified). Provides a more convenient means of checking Local Memory contents during program debugging. This is the default format. See example in Figure 3-3.
DISC	Causes output to be directed to the disc file specified in interactive mode.
LP TTP MTW	Causes output to be directed to the corresponding devices. MTW must not be used if TOPSY has been reconfigured with DEBUG as MTIO will not have been loaded.

The commands DATA, STRING, DEBUG, and DISC may be abbreviated using just the first two letters, if desired.

3.3.2.1.1 Assigning Disc Files. Prior to dumping Local Memory contents to disc, a file must be pre-assigned within DOPSY. The size of the files will be different for the DATA format than for the STRING and DEBUG formats.

DATA	1 file word must be assigned for each rank of data to be dumped.
Example:	Dump 4096 Local Memory locations by 4 ranks. 4096 x 4 = 16,384
	// ASSIGN 'LMIØØ1' 16348 WORDS DATA
NOTE:	For DATA format, files must be assigned type DATA.

STRING OR DEBUG — The file size required is difficult to predict precisely since it varies with the type of instructions in Local Memory. However, the following formula will provide an approximate value:

SIZE = ((20 + pins) /4) *L/M LOCATIONS

Example: Assume 4 ranks (60 pins) by 1024 Local Memory locations.
SIZE = ((20+60)/4) *1024 = 20,480 words
// ASSIGN 'LMSF01' 20480 WORDS

When in doubt about the required file size, simply ASSIGN a very large file; then after LMIO execution, reduce the size to the minimum required with:

// ASSIGN 'filnam'

3.3.2.2 LOADING LOCAL MEMORY FROM DISC. The following command may be used to load a specific disc file in DATA format to Local Memory, or to clear the contents of Local Memory to zero.

/. LMIO [[LOAD PAGE/'filnam'] /ADDR] /CLEAR]

Where:

LOAD — Causes local Memory to be loaded with the contents of the file defined by PAGE or 'filnam'. Data is loaded starting at the address specified by ADDR. If ADDR is omitted, loading occurs at the current value in the MCS register. MCS may be set from Manual Analysis or by the 'AT' statement within the FACTOR program.

PAGE — Is a number in the range 000 to 999 which, when appended to a prefix like 'LMI', yields the file name 'LMIXXX'. 'LMI' is the required prefix for DATA type files and 'LMSF' is for STRING or DEBUG files. This method of denoting the file name is an alternative to 'filnam'.

'filnam' — Is a 6 character or less alpha-numeric file name in which the first character must be alphabetic. Working storage on disc may be specified by specifying WS.

ADDR — Is the Local Memory address at which the data file on disc is to be loaded. May be entered as a decimal or octal number if followed by a 'B'. If omitted, the current contents of the MCS register is used.

CLEAR — Clears Local Memory contents to zero.
Clears 8 ranks by 4096 locations.

3.3.3 Error Messages

The following error messages are displayed on the TTP when appropriate.

ERROR MESSAGE	DESCRIPTION
FILE NOT ASSIGNED	The specified file was not found on disc. Possible error in file naming procedure.
FILE TOO SMALL	The assigned size of the file is too small for the amount of data being dumped.
SYMBOL TABLE OVERFLOW	The limit of 500 SPM Local Memory labels has been exceeded.
FILE EMPTY	User attempted to load Local Memory data from an empty disc file.

```
ENABLE DA,MA;
SP030
LSET IX  1110000000 0000010100 0000000000;
LSET STROBE
         1110000000 0000010100 0000000000;
LSET RZ  1110000000 0000010100 0000000000;
LSET XOR 1110000000 0000010100 0000000000;
LSET I   1110000000 0000010100 0000000000;
LCGEN TG1 4,6,8,9;
LCGEN TG2 5,7,10;
LCGEN TG12 1,2,3;
LSET DA  1110000000 0000010100 0000000000;
LSET DB  1110000000 0000010100 0000000000;
LSET MA  1110000000 0000010100 0000000000;
LSET MB  1110000000 0000010100 0000000000;
ENABLE DB,MB;
SET F    1110000000 0000010100 0000000000
         =LCALL SP001;
LSET IX  1110000000 0000010100 0000000000
         =LCALL SP002;
SET F    1110000000 0000010100 0000000000
         =LGOTO SP003;
SET FC   NORMAL    35
         1110000000 0000010100 0000000000;
SET FC   MATCH    250
         1110000000 0000010100 0000000000;
SET FC   CONTIN
         1110000000 0000010100 0000000000;
LSUBR SP001 MATCH   455;
SET F    1110000000 0000010100 0000000000;
LCGEN TG5 11,22,30,44,55;
SET F    1110000000 0000010100 0000000000;
SET F    1110000000 0000010100 0000000000
         =LEND;
LSUBR SP002 CONTIN    1;
SET F    1010101010 0000010100 0000000000;
SET F    1010101010 0000010100 0000000000
         =LEND;
```

Figure 3-2 A sample of LMIO STRING output produced by the command:

/. LMIO STRING LP

Note that all user labels are replaced by labels SPXXX where XXX is a three digit number 000 to 999.

```
0000   LSET IX                          DA MA 1110000000 0000010100 0000000000
0001   LSET STROBE                      DA MA 1110000000 0000010100 0000000000
0002   LSET RZ                          DA MA 1110000000 0000010100 0000000000
0003   LSET XOR                         DA MA 1110000000 0000010100 0000000000
0004   LSET I                           DA MA 1110000000 0000010100 0000000000
0005   LCGEN TG1 4,8,8,9
       LCGEN TG2 5,7,10
       LCGEN TG12 1,2,3
0010   LSET DA                          DA MA 1110000000 0000010100 0000000000
0011   LSET DB                          DA MA 1110000000 0000010100 0000000000
0012   LSET MA                          DA MA 1110000000 0000010100 0000000000
0013   LSET MB                          DA MA 1110000000 0000010100 0000000000
0014   SET F    =LCALL   0022           DB MB 1110000000 0000010100 0000000000
0015   LSET IX  =LCALL   0030           DB MB 1110000000 0000010100 0000000000
0016   SET F    =LGOTO   0000           DB MB 1110000000 0000010100 0000000000
0017   SET FC   NORMAL     35           DB MB 1110000000 0000010100 0000000000
0020   SET FC   MATCH      250          DB MB 1110000000 0000010100 0000000000
0021   SET FC   CONTIN                  DB MB 1110000000 0000010100 0000000000
0022   SET F    LSUBR    MATCH    455  DB MB 1110000000 0000010100 0000000000
0023   LCGEN TG5 11,22,30,44,55
0026   SET F                            DB MB 1110000000 0000010100 0000000000
0027   SET F    =LEND                   DB MB 1110000000 0000010100 0000000000
0030   SET F    LSUBR    CONTIN     1  DB MB 1010101010 0000010100 0000000000
0031   SET F    =LEND                   DB MB 1010101010 0000010100 0000000000
```

Figure 3-3 A sample of LMIO DEBUG output produced by the command:

/. LMIO DEBUG LP

Note the three LCGENs corresponding the location ØØØ5. LMIO will always print the results of any CGENs which appeared in the FACTOR program previous to the Local Memory load along with the first LCGEN encountered. Since each individual LCGEN in Local Memory causes the address printed to jump by 3, it may be observed above that two of the LCGENs actually represent previous CGENs, since the address only jumps from 5 to 8. The user must examine his program to determine which is the genuine LCGEN. Subsequent LCGENs print exactly as programmed (note address ØØ23).

3.4 CYCLE

3.4.1 Introduction

The program CYCLE is a debug aid which, during the execution of a device test program, enables the user to establish a continuous loop between two specified addresses in local memory. CYCLE is loaded and initiated in an interactive user/VKT mode during a pre-established PAUSE condition in the program in which the loop is to be established. The user is required to supply the loop start and end addresses in the call entered for CYCLE. The user may also, as an option, specify the generation of a synch pulse at a tester pin during the operation of the CYCLE established loop.

3.4.2 Operating Procedure

The procedure required to use CYCLE is:

1. Establish a TOPSY mode of operation (i.e., /. ANALYSIS STATn).
2. Enter a PAUSE command to halt the execution of the test program at the point at which the CYCLE loop is to be established. Use either the command:
   a. /. PAUSE statement number [ON/OFF] to pause at the specified statement, or
   b. /. PAUSE FAIL [ON/OFF] to pause at a fail point.
3. Load and run the test program in which the loop is to be established.
4. When the test program enters the desired PAUSE condition, load and initiate CYCLE using a command of the form:

   ./ CYCLE [J,K] (SADDR,SPIN);

Parameter	Function
J	Defines the loop start address.
K	Defines the loop end address.
SADDR	Defines an address in local memory at which a synch pulse is to be generated. Whenever SADDR is used, the parameter SPIN must be used.
SPIN	Specifies the tester pin on which the synch pulse generated at location SADDR is to appear. The pin SPIN is set to a logic 1 value for each synch pulse.

5. Terminate the CYCLE loop and restore any affected registers to their original (pre-loop) state(s) by entering the command:

   /.CYCLE OFF

NOTE

The command /. WRITE MCS 101* may also be used to terminate a CYCLE loop, however, registers which are affected by the loop operation are not restored to their original (pre-loop) state(s).

3.4.2.1 OPERATING CONSIDERATIONS - The following items should be considered when using CYCLE:

1. A second or subsequent loop may be established in a program without affecting the operation of a previously established loop.
2. All fails are suppressed during looping.
3. The following registers are affected by a CYCLE loop operation and must be restored to their original state(s) by terminating CYCLE operations with the command /. CYCLE OFF
4. a. Major loop end #1706 (L)
   b. Test start #1700 (SA)
   c. Minor loop start #1704 (J)
   d. Minor loop end #1705 (K)
4. On either a SENTRY 610 or SENTRY II test system without SPO enabled, looping occurs between minor loop limits (J) and (K).
5. On a SENTRY II test system with SPO enabled, looping occurs between test start (SA) and Test end (L).

SECTION 4

FACTOR ENHANCEMENT UTILITY PROGRAMS

4.1 LMMOD

4.1.1 Introduction

LMMOD is an assembly language program designed to assist the FACTOR programmer in modifying locations in local memory using the assembly language linkage statement EXEC. The LMMOD program only modifies local memory; if a string file of the functional pattern is required, the user may create his string file using the assembly language program LMTSF.

4.1.2 Program Usage

The following functions are implemented:

A. Clear local memory

EXEC LMMOD (Ø)

B. Complement the designated pins in all locations.

EXEC LMMOD (1, P_1, ..., P_n);

P = pin number

If P = Ø or is missing, complement all pins.

Examples: EXEC LMMOD (1, 3, 7, 14);
EXEC LMMOD (1);

C. Swap the condition of the first designated pin for the condition of the second designated pin in all locations. Do not change condition of second pin.

EXEC LMMOD (2, P_1, P_2);

P_1 takes on the value of P_2.P_2 remains unchanged.

D. Insert ENABLE DA/DB, MA/MB statements.

EXEC LMMOD (FUNCT, LINE, REPT, AB_1, NO_1, AB_2, NO_2, . . ., AB_n, NO_n);

FUNCT = 3 - insert ENABLE DA/DB
4 - insert ENABLE MA/MB

LINE = Ø - 4096 - local memory loc. to tsart first ENABLE
REPT = 1 - 4096 - number of times to repeat the following AB-NO pattern.

AB = 0 - A
1 - B

NO = 1 - 4096 - number of times to set preceding A/B

There can be a total of 30 AB - NO combinations.

E. Shift designated pins in channel by displacement factor.

EXEC LMMOD (5, DISP, FROM, TO, P_1, P_2,...,P_2);

DISP = Number of channels to shift designated pins. If DISP is negative, get value from lesser local memory address. If DISP is positive, get value from greater local memory address.

FROM = First local memory address to change

TO = Last local memory address to change

P = Pin numbers (channels) whose values are to be shifted.

F. Clear designated pin in all local memory locations.

EXEC LMMOD (6, P_1,P_2,...,P_n);

P = Pin number (channel) to clear.

4.1.3 Error Message

ERROR MESSAGE	DESCRIPTION
LMMOD ERROR STATEMENT NNNNN	A syntax error is encountered in the EXEC LMMOD statement. The program is terminated and a return is made to the DOPSY monitor.

4.2 LMTSF

4.2.1 Introduction

The purpose of LMTSF is to create, from local memory, a TASCII file of SET F and SPM instructions on disc. ENABLE MA/B, DA/B instructions are also produced as required.

LMTSF is written in assembly language and is called by the FACTOR programmer using the assembly language linkage statement EXEC.

4.2.2 Program Usage

FACTOR Calling Sequence:

EXEC LMTSF (FILE, TPINS, LMWORDS, START, TAG);

where

FILE	=	a one or two digit number nn to be appended to the character string "LMSF" to create a unique DOPSY file name 'LMSFnn' for storing the generated data. Where nn=00-99.
TPINS	=	total number of bits in binary pin pattern (1-120)
LMWORDS	=	total number of local memory words to use, i.e., total number of SET F statements to generate.
START	=	first local memory address to dump. If missing, local memory address zero is assumed.
TAG	=	a one digit number to be used to create a unique local memory label on the last SET F statement. The label generated is LMSFn@. n=1 - 9. If no tag is defined, no local memory label will be created.

Examples:

1. EXEC LMTSF (13, 42, 512);

The generated data will be stored in string file 'LMSF13'.

There are 42 bits in the pin pattern.

The first 512 words of local memory will be used to create the file.

2. EXEC LMTSF (3, 15, 200, 768, 7);

200 SET F statements, 15 pins wide, starting with local memory location 768 will be written to disc file LMSF03. The last statement will be tagged with local memory label LMSF7@.

4.2.2.1 ASSIGNING DISC FILES. Prior to executing the program which calls LMTSF the user must reserve space in working storage to store the generated data using the DOPSY ASSIGN command. To calculate the approximate number of words to be saved, the following formula can be used:

WORDS = ((20+PINS)/4) * L/M LOCATIONS

For the first example above the following ASSIGN command would first be executed:

// ASSIGN 'LMSF13' 8200 WORDS

4.2.3 Error Messages

ERROR MESSAGE	DESCRIPTION
PARAMETER ERROR	Incorrect parameter(s) in FACTOR EXEC statement
FILE NOT ASSIGNED	User forget to assign the LMSFXX file prior to LMTSF execution.
FILE TOO SMALL	The LMSFXX file was assigned too few words for the amount of data in Local Memory.
CAN'T CLOSE FILE	A disc or other hardware error occurred during attempt to close LMSFXX file.
SYMBOL TABLE OVERFLOW	The maximum of 100 Local Memory labels referenced by SPM LGOTO and LCALL statements was exceeded. (For Sentry II only).

4.3 LPLF

4.3.1 Introduction

LPLF is an assembly language program which permits the user to position the printer paper within a FACTOR language program.

4.3.2 Program Usage

LPLF is invoked by placing the following FACTOR assembly language linkage statement in the source program:

```
EXEC LPLF (NUMSP, NUMT);
```

where

NUMSP = number of lines to space

If NUMSP is missing or equal to zero (Ø), the paper is positioned at top of form.

when NUMSP = Ø

NUMT = number of top of forms to issue.

EXAMPLES:

1. Issue top of form (page eject)

```
EXEC LPLF;
```

or

```
EXEC LPLF (Ø);
```

2. Issue 5 page ejects

```
EXEC LPLF (Ø,5);
```

3. Space 6 lines

```
EXEC LPLF (6);
```

4.4 LMLOAD

4.4.1 Introduction

LMLOAD is an Assembly Language Utility program which allows functional test data to be transferred between local memory and files on disc at run time. Large bodies of SET F data may be executed from within a given test program without actually being present in the program source file, thus reducing compile time significantly.

4.4.2 Program Usage

There are 4 facets to LMLOAD usage:

Assigning space to the required disc file(s) into which local memory contents will be stored.

The basic LMLOAD calling sequence which saves local memory contents on disc or loads the disc file contents to local memory.

Defining within LMLOAD the first 3 characters of the names of the disc files previously assigned (available on Rev 10 software only).

An optional LMLOAD procedure which will save the disc addresses of previously referenced files in a special fashion which may result in decreased execution time in certain instances as described below.

4.4.2.1 ASSIGNING DISC FILES. Each execution of LMLOAD will transfer the contents of local memory to one of up to 140 individual disc files. Each file may, therefore, be thought of as a local memory image on disc. The files used must be individually assigned with the DOPSY ASSIGN command under any job desired. The amount of disc space in words assigned to a given file must be equal to or greater than the "size" of local memory being saved on disc. For example, if 1024 local memory locations of 2 ranks each are to be saved the "size" required would be:

1024 x 2 = 2048 words

All files must be assigned a name of the form:

LMIxxx (Local Memory Image)

Where xxx is a right justified 3 digit number between 000 and 139.

Example:

Save on disc 3 separate local memory loads of 1024 words by 2 ranks each. 3 disc files are required.

// ASSIGN 'LMIØØØ' 2Ø48 WORDS DATA

// ASSIGN 'LMIØØ1' 2Ø48 WORDS DATA

// ASSIGN 'LMIØØ2' 2Ø48 WORDS DATA

NOTE:

(1) LMLOAD "optimizes" the data being saved on disc by saving only changed ranks. Therefore, the actual file space required may be less than that calculated above. The file may, therefore, be compacted to the minimum required size.

(2) An optional procedure is available with Rev 10 SENTRY software which allows a user defined file name prefix to replace the prefix 'LMI' (see section 4.4.2.3.2 below).

4.4.2.2 BASIC LMLOAD CALLING SEQUENCE. Functional test data will be transferred between Local Memory and disc with a FACTOR statement of the form:

EXEC LMLOAD (PAGE, N, STOP);

Where PAGE = a 3 digit file name suffix between 000 and 139 for use in name 'LMIxxx' as described in 4.4.2.1 above.

N = 1, save Local Memory contents in the disc file defined by PAGE.

= 2, load the contents of the disc file defined by PAGE into Local Memory.

STOP = Highest Local Memory location to be saved on disc. Applicable for N=1 only. A number in the range 0 to 4095. The default value is 1023 if STOP is not specified.

The STOP parameter is not applicable when loading Local Memory from disc; N=2. All the data previously written to the disc file will be loaded back to Local Memory.

4.4.2.3 OPTIONAL LMLOAD PROCEDURES. The following optional LMLOAD procedures are available:

4.4.2.3.1 Saving File Disc Addresses On Disc. The disc address of each previously referenced file is automatically saved by LMLOAD in order to minimize the time spent searching the disc directory should a given file be referenced more than once. These addresses are, however, saved only if LMLOAD remains in core throughout the entire testing session. If another Assembly Language Utility is called, or if another test station comes on line, LMLOAD will be removed from core and the disc directory search will have to be repeated on the next call to LMLOAD.

Therefore, an optional LMLOAD procedure is available which will save these file disc address on disc within the coreimage LMLOAD file so that they will be available on the next call to LMLOAD following its removal from core. All addresses saved during a previous testing session are always cleared forcing a fresh directory search following initial TOPSY entry from DOPSY. The calling sequence is as follows:

```
EXEC LMLOAD (0,3);
```

This statement should be either the last call to LMLOAD in the program or it should follow the last call to LMLOAD prior to an EXEC statement which calls another Assembly Language Utility.

4.4.2.3.2 Allowing User Defined Names to Replace "LMI" in the "LMIXXX" File Names

NOTE: This feature is available only on SENTRY software REV 10 and above.

As described in section 4.4.2.1 above, all disc files to be referenced by LMLOAD must be ASSIGN'ed on disc and given names of the form:

LMIxxx where xxx is a 3 digit number between 000 and 139.

The characters "LMI" are actually a default value and may be replaced by any 3 user defined alpha-numeric characters.

The calling sequence is as follows:

```
DCL NAME /'FST'/;
EXEC LMLOAD (NAME,4);
```

Where NAME is any FACTOR variable name defined in the above DCL statement

FST represents any 3 user defined alpha-numeric characters

These statements must precede all other calls to LMLOAD in the program.

"LMI" files and files with "user defined" names may not both be used by the same FACTOR program.

WARNING

Problems may result if this function is used in conjunction with that described in Section 4.4.2.3.1 and a second station comes on line with a different test program. The program on the second station must either reference files with different numbers (000-139) or must use the same three character "user defined" file name prefix as the program on the first station.

4.4.3 Error Messages

When a user error occurs LMLOAD will produce the appropriate error messages as defined below. The error message will by followed by a conventional TOPSY terminal error 100 message including the statement number of the EXEC LMLOAD.

ERROR MESSAGE	DESCRIPTION
FILE NOT FOUND	The disc file LMIxxx was not ASSIGNED on disc as described in Section 4.4.2.1.
FILE EMPTY	An attempt was made to load an empty file to Local Memory. Data was not previously saved in the file.
FILE TOO SMALL	File size too small compared to actual amount of Local Memory data being saved.
PAGE OUT OF RANGE	PAGE exceeds legal range as defined in section 4.4.2.2.
N OUT OF RANGE	N exceeds legal range as defined in section 4.4.2.2.
STOP OUT OF RANGE	STOP exceeds legal range as defined in section 4.4.2.2.
STOP EXCEEDS LM SIZE	STOP exceeds the actual size of Local Memory.
DISK I/O ERROR	A hardware disc error occurred in reading or writing disc.
INVALID NAME	The file name prefix specified in a DCL statement to replace "LMI" was either all blank or contained more than 3 characters.

4.5 LMSAVE

4.5.1 Introduction

LMSAVE is an Assembly Language Utility program designed to be used in conjunction with microprocessor test generation programs. LMSAVE currently supports 6 separate functions as outlined below:

Transfers functional test data between Local Memory and files on disc. Data is transferred at run time thus obviating need for large quantities of SET F's in the FACTOR source file. This function is identical to that of utility program LMLOAD (see 4.5.2.1).

Transfers single data items between FACTOR program and "virtual RAM memory" file on disc. Data is "read" or "written" in true random access fashion (see 4.5.2.3.1).

Allows input of user generated microprocessor diagnostic program from cards or mag tape into the "virtual RAM memory" (see 4.5.2.3.2).

Converts floating point numbers to Octal or Hexdecimal and returns them to the calling FACTOR program in TASCII format for printing on an output device (see 4.5.2.4.1).

Converts microprocessor op-codes to their corresponding mnemonics and returns them to the calling FACTOR program in TASCII format for printing on an output device (see 4.5.2.4.2).

Accepts data from an input device in either Octal or Hexidecimal and returns it to the calling FACTOR program (see 4.5.2.4.3).

4.5.2 Program Usage

There are 3 facets to LMSAVE usage:

The basic LMSAVE calling sequence for the particular function desired.

Assigning space to the required disc file(s) into which local memory contents will be stored.

Defining within LMSAVE the first 3 characters of the names of the disc files previously assigned (available on Rev 10 software only).

Each of these items will be discussed in detail below where applicable.

4.5.2.1 TRANSFERRING FUNCTIONAL TEST DATA BETWEEN LOCAL MEMORY AND DISC. Functional test data will be transferred between Local Memory and disc with a FACTOR statement of the form:

EXEC LMSAVE (PAGE, N, STOP);

Where PAGE = a 3 digit file name suffix between ØØ1 and 139 for use in name 'LMIxxx' as described in 4.5.3 below:

N = 1, save Local Memory contents in the disc file defined by PAGE.

= 2, load the contents of the disc file defined by PAGE into Local Memory.

STOP = Highest Local Memory location to be saved on disc. Applicable for N=1 only. A number in the range Ø to 4095. The default value is 1Ø23 if STOP is not specified.

The STOP parameter is not applicable when loading Local Memory from disc; N=2. All the data previously written to the disc file will be loaded back to Local Memory.

4.5.2.2 ALLOWING USER DEFINED NAMES TO REPLACE "LMI" IN THE "LMIXXX" FILE NAMES.

NOTE: This feature is available only on SENTRY software REV 10 and above.

As described in Section 4.5.3 below all disc files to be referenced by LMSAVE must be ASSIGN'ed on disc and given names of the form:

LMIxxx where xxx is a 3 digit number between ØØØ and 139.

The characters "LMI" are actually a default value and may be replaced by any 3 user defined alpha-numeric characters.

The calling sequence is as follows:

DCL NAME /'FST'/;

EXEC LMSAVE (NAME,4);

Where NAME is any FACTOR variable name defined in the above DCL statement

FST represents any 3 user defined alpha-numeric characters

These statements must preceed all other calls to LMSAVE in which disc files are being used.

"LMI" files and files with "user defined" names may not both be used by the same FACTOR program.

WARNING

> Problems may result if this function is used in conjunction with that described in Section 4.4.2.3.1 and a second station comes on line with a different test program. The program on the second station must either reference files with different numbers (000-139) or must use the same three character "user defined" file name prefix as the program on the first station.

4.5.2.3 TRANSFERRING DATA TO OR FROM THE "VIRTUAL RAM MEMORY" FILE LMIØØØ. The following two functions are designed to initialize or update a file on disc referred to as the "Virtual RAM memory" file. This file, which must be named LMIØØØ, will contain the user generated diagnostic program which is intended to exercise a known good microprocessor and elicit response which will be saved in files LMIØØ1 thru LMInnn for future testing of subsequent devices. See Technical Bulletin 4 for more details.

Before executing LMSAVE in this mode, the file LMIØØØ must be assigned 66528 words on disc with the following DOPSY command:

// ASSIGN "LMIØØØ" 66528 WORDS DATA

If the file re-naming feature described in 4.5.2.2 above is to be used in the FACTOR program, the file name prefix LMI may be changed accordingly.

4.5.2.3.1 Transferring Single Data Items Between the FACTOR Program and File LMIØØØ. Data items may be randomly fetched from and returned to the "Virtual RAM Memory" file LMIØØØ with a FACTOR statement of the form:

EXEC LMSAVE (Ø, N2, N3, N4, N5, N6);

where N2 = 1 to place data into LMIØØØ
2 to fetch data from LMIØØØ

N3 = Random word address within LMIØØØ of data being accessed; $Ø \leq N3 \leq 65535$

N4 = Data, in floating point format, to be put to or fetched from LMIØØØ. Accesses bits Ø thru 15 of word addressed by N3. Must be a positive integer <32767.

N5 = Ø for normal use of N3 as defined above.
= 1 for LMIØØØ use as a repository for peripheral data. N3 is redefined: $Ø \leq N3 < 999$ where N3 = peripheral ID. This function utilizes LMIØØØ addresses beyond 65535.

N6 = Data, in floating point format, to be put to or fetched from LMIØØØ. Accesses bits 16 thru 23 of word addressed by N3. Must be a positive integer < 255.

Parameter N6 is optional and may be omitted if desired.

4.5.2.3.2 Loading File LMIØØØ With User Generated Data From an Input Device. The "Virtual RAM Memory" file, LMIØØØ, may be initially loaded with the user generated diagnostic program with the following FACTOR statement:

EXEC LMSAVE (Ø, N2, N3);

where N2 = 3 for data input from cards
7 for data input from mag tape

N3 = Ø for data in Octal
1 for data in Hexadecimal

The user generated program data will be in the form of Assembled object code for the particular microprocessor under test. This data as it appears on the input media, cards or mag tape, must conform to the following format:

1. The first item in a record, beginning in column 1, is the address in memory that the following data will be stored.

2. A blank space separates address and data items in the record.

3. Up to three data items may be in a single record, each item separated by one space.

4. The data items will be stored at adjacent addresses starting with the address specified by the first item (column 1).

5. Two consecutive blank spaces indicate the end of data items. Comments may appear in remaining columns.

6. If the first column on a record is left blank, the address for data on that record will be where the previous item left off.

7. Data items may be in Octal or Hexadecimal. The call to LMSAVE from the generation program will define which form is being used.

8. The User Program data is terminated by a // record. This stops the loading of user data onto the disc file LMIØØØ.

4.5.2.4 MISCELLANEOUS LMSAVE DATA CONVERSION AND DATA INPUT FUNCTIONS. The following functions involve interaction between LMSAVE and the FACTOR program for purposes of data conversion and data input into the FACTOR program.

4.5.2.4.1 Convert Numbers From Floating Point to Octal or Hexadecimal Format. A number in floating point format will be obtained from the FACTOR program, converted as directed, and returned to the FACTOR program with the following statement:

EXEC LMSAVE (Ø, 4, N3, N4, N5, N6);

where N3 = Ø for conversion to Octal
1 for conversion to Hexadecimal

N4 = The floating point number to be converted.

N5 = Upper 2 characters of result in TSACII

N6 = Lower 4 characters of result in TSACII

4.5.2.4.2 Converting Microprocessor Op-Codes Into Their Corresponding Mnemonics

EXEC LMSAVE (Ø, 5, N3, N4, N5);

where N3 = Op-code in floating point format

N4 = Upper for characters of mnemonic

N5 = Lower four characters of mnemonic

A user generated file, which must be named MNEMON, contains the mnemonics for the microprocessor under test listed in numerical order according to the op-codes corresponding to each mnemonic. The string file should be named '*MNEMON.' The object file, 'MNEMON', should be created as follows:

```
// ASM '*MNEMON' OBJ
// CREATE 'MNEMON' OBJ
```

This file must reside under the same job as LMSAVE prior to the creation of the coreimage file 'LMSAVE'.

EXAMPLE:

```
MNEMON  PROC    Ø
        TEXT    'NOP'   Ø
        TEXT    'LXI'   1
        TEXT    'RLC'   2
          .
          .
          .
          .
        TEXT    'STA'   n
        END
```

4.5.2.4.3 Transfer Data From Input Device to FACTOR Program. Data will be accepted in Octal or Hexidecimal and passed to the FACTOR program in floating point format with the following statement:

EXEC LMSAVE (Ø, 6, N3, N4, N5, N6);

where N3 = Ø for data input in Octal
1 for data input in Hexadecimal

N4 = First data item in floating point format received from input device.

N5 = Second data item in floating point format received from input device.

N6 = 0 for data input from keyboard
1 for data input from card reader
2 for data input from mag tape

A second data item in the input record (N5) is optional and need not be supplied when using this LMSAVE function. If one is entered, it must be in the same record as the first and separated by at least one space.

4.5.2.4.4 Perform a Line Printer Top of Form

EXEC LMSAVE (0);

4.5.3 Assigning Disc Files

The execution of LMSAVE as described in section 4.5.2.1 to transfer data between disc and Local Memory requires that disc files be preassigned. Each execution of LMSAVE in this mode will transfer the contents of local memory to one of up to 139 individual disc files. Each file may, therefore, be thought of as a local memory image on disc. The files used must be individually assigned with the DOPSY ASSIGN command under any job desired. The amount of disc space in words assigned to a given file must be equal to or greater than the "size" of local memory being saved on disc. For example, if 1Ø24 Local Memory locations of two ranks each are to be saved the "size" required would be:

1024 x 2 = 2048 words

All files must be assigned a name of the form:

LMIxxx (Local Memory Image)

where xxx is a right justified 3 digit number between ØØ1 and 139.

Example:

Save on disc 3 separate local memory loads of 1024 words by 2 ranks each. Three disc files are required.

```
// ASSIGN 'LMI001'  2048 WORDS DATA
// ASSIGN 'LMI002'  2048 WORDS DATA
// ASSIGN 'LMI003'  2048 WORDS DATA
```

NOTE:

(1) LMSAVE "optimizes" the data being saved on disc by saving only changed ranks. Therefore, the actual file space required may be less than that calculated above. The file, may, therefore, be compacted to the minimum required size by the command: // ASSIGN 'LMIxxx'

(2) An optional procedure is available with Rev 10 SENTRY software which allows a user defined file name prefix to replace the prefix 'LMI' (see section 4.5.2.2 above).

4.5.4 Error Messages

When a user error occurs LMSAVE will produce the appropriate error message as defined below. The error messgage will be followed by a conventional TOPSY terminal error 100 message including the statement number of the EXEC LMSAVE.

ERROR MESSAGE	DESCRIPTION
FILE NOT FOUND	The disc file LMIxxx was not ASSIGNED on disc as described in section 4.5.3.
FILE EMPTY	An attempt was made to load an empty file to Local Memory. Data was not previously saved in the file.
FILE TOO SMALL	File too small compared to amount of Local Memory data being saved or LMI000 assigned less than 66528 words.
PAGE OUT OF RANGE	PAGE exceeds legal range as defined in section 4.5.2.1.
N OUT OF RANGE	N exceeds legal range as defined in section 4.5.2.1.
STOP OUT OF RANGE	STOP exceeds legal range as defined in section 4.5.2.1.
STOP EXCEEDS LM SIZE	STOP exceeds the actual size of Local Memory.

ERROR MESSAGE	DESCRIPTION
DISK I/O ERROR	A hardware disc error occurred in reading or writing disc.
INVALID NAME	The file name prefix specified in a DCL statement to replace "LMI" was either all blank or contained more than 3 characters.

4.6 LOGREG

4.6.1 Introduction

LOGREG is an Assembly Language utility program which, when called from a FACTOR program, permits reading and writing of all tester long registers. The register contents may optionally be logged, in binary, on the POD.

LOGREG will allow reading and writing of registers with four digit register numbers in the 1701 thru 1707 range in addition to registers 1734 and 1735.

4.6.2 Program Usage

LOGREG is invoked by placing the following FACTOR statement in the source program:

```
EXEC LOGREG (NUM, VAL, OP);
```

where NUM = Long register read or write code as found in Appendix B of the appropriate FACTOR manual.

VAL = Value to be written to desired register or contents of register just read. The most significant bit (left most digit) corresponds to pin 1. Must not exceed 5 octal digits.

OP = Ø, Read or write a register with no logging to POD.

1, Read or write a register and log VAL to the POD in binary.

2, Supress reading or writing and log VAL to the POD.

Reading or writing to a register is determined by the first digit of the register read/write code. For example, to write to rank 1 of the M register:

```
EXEC LOGRAG (240B, VAL, Ø);
```

To read rank 1 of the M register:

```
EXEC LOGREG (440B, VAL, 1);
```

VAL may be logged to the line printer by:

```
// SET LP or
/. DATALOG LP STATXX
```

Otherwise, logging will take place on the TTP.

4.6.3 Additional Documentation

Application Note AD 1042.

4.7 FMTAP

4.7.1 Introduction

FMTAP is a FACTOR callable utility mag tape routine which permits multiple mag tape files to be read or written during a single program execution.

4.7.2 Program Usage

The following functions are performed by placing the associated FACTOR statement in the source program.

```
EXEC FMTAP (0); rewind tape
EXEC FMTAP (1, N); skip forward N end of file marks
EXEC FMTAP (2, N); skip backward N end of file marks
EXEC FMTAP (3); write an end of file mark
EXEC FMTAP (4, ARR); read one record into array ARR.
EXEC FMTAP (5, ARR); write one record from array ARR.
```

The number of words read/written is determined by the number of elements assigned by the DCL statement. Only one array name may be defined, but the array size may exceed 512 words.

4.8 GLOBS

4.8.1 Introduction

The program GLOBS adds 100 GLOBS system global variables to the existing 20 global variables provided by the system software. The use of GLOBS then provides the user with a total of 120 system global variables.

GLOBS is loaded and its functions controlled by the insertion of FACTOR call statements into the program in which it is to be used.

4.8.2 FACTOR Call Statement

The FACTOR statement required to load and initiate DATAIO has the format:

EXEC GLOBS (num,opcode,val);

PARAMETER	DESCRIPTION
num	A constant, variable, or array element the value of which specifies the number of the global variable (i.e., 1 to 120) which is to be used for the operation by the parameter "opcode".
opcode	A constant, variable, or array element the value of which specifies the operation to be performed. The following "opcode" values are permitted:

VALUE	OPERATION SPECIFIED
0	Reset the specified global variable to zero.
1	Increment the global (i.e., value of "num") by 1.
2	Decrement the global (i.e., value of "num") by 1.
3	Copy the contents of the memory locations specified by "val" into the global variable specified by "num".
4	Copy the contents of the global variable specified by "num" into the memory location specified by "val".

val	A constant, variable, array element or scalar value. The global variable or location specified by "val" is to be used for the operations specified by "opcodes" 3 and 4.

4.8.3 Program Operation

GLOBS is an FST-2 assembly language overlay which, when loaded, occupies the upper 180 word locations of CPU memory.

A GLOBS program must be configured for the specific test system on which it is to be used. The required configuration procedure is described in detail in the program's assembly language source listing.

Since GLOBS is an overlay, a small amount of system overhead time is required for its use. For example the incrementation of a standard systems global variable (i.e., 1 to 120) requires approximately 1.6 milliseconds (e.g., GLOB4 = GLOB4+1;); the incrementation of a GLOBS global variable (i.e., 21 to 120), however, requires approximately 2 milliseconds (e.g., EXEC GLOBS (50,1);).

4.8.4 Resetting GLOBS Global Variables

The TOPSY keyboard commands:

```
/. CLEAR STATxx and
/. LOAD "name" STATxx
```

which are used to reset the standard systems global variables to zero do NOT effect GLOBS global variables.

To reset a GLOBS global variable, it is necessary to include a statement in the FACTOR program which explicitly resets the specified GLOBS global variable. For example to reset GLOBS global variable 100 to zero the FACTOR statement:

```
EXEC GLOBS (100, 0);
```

must be included in the program using GLOBS.

All or any selected sequence of GLOBS global variables may be reset by including a reset loop routine in the program. For example the FACTOR statement:

```
FOR I = 1 THRU 120 DO EXEC GLOBS (I, O);
```

would reset all of the global variables to zero.

4.8.5 Examples, Use of GLOBS

a. Reset all GLOBS global variables to zero when a specific condition occurs:

```
IF SWITCH EQ 0 THEN FOR I = 1 THRU 120 DO EXEC GLOBS
(I, O);
```

The value of SWITCH must be other than 0 for all subsequent test executions.

b. Place a given floating point number into GLOB 47:

```
AA = 4.624;
EXEC GLOBS (47, 3, AA);
```

or

```
EXEC GLOBS (47, 3, 4.624);
```

c. Measure the voltage on pin 6 and place the result in GLOB 47:

```
CPMU PIN 6;
MEASURE VALUE;
EXEC GLOBS (47, 3, VALUE);
```

d. Display the contents of GLOB 47 on the TTP:

```
EXEC GLOBS (47, 4, VAL);
WRITE (TTP) VAL;
```

SECTION 5

PATTERN GENERATION UTILITY PROGRAMS

5.1 CSETF

5.1.1 Introduction

CSETF is an assembly language program designed to assist the FACTOR programmer in generating Sentry compatible string files of SET F data when the input pin patterns are of a repetitive nature.

5.1.2 Program Usage

The user creates his test program in two steps.

During step 1, the CSETF program will generate SET F commands and store them in a pre-ASSIGNed string file.

During step 2, the user's FACTOR program is recompiled and the string file generated in step 1 is INSERTed into the user's program ready for execution.

5.1.2.1 CALLING SEQUENCES. In step 1 CSETF is called using the FACTOR assembly language linkage statement EXEC. The form is

A. Initialize program and designate output file.

EXEC CSETF (Ø, N);

N = a one or two digit number to be appended to the character string "SETF" to create a unique DOPSY file name 'SETFN' for storing the generated data.

B. Define Pins

EXEC CSETF (PIN, LINE, REPT, BIT_1,NO_1,...,BIT_n,NO_n);

PIN = pin number ≤1≤ PIN 60
LINE = line number on which the subsequent pattern will begin.
REPT = the number of times the subsequent BIT-NO combinations will be repeated.
BIT = Ø or 1.
NO = number of times the previous bit will be repeated.

C. Terminate input and create output file.

EXEC CSETF (61);

Upon encountering the termination card, the SETF file is created and the following comment is displayed on the TTP

END CSETF

5.1.2.2 ASSIGNING DISC FILES. Prior to executing the FACTOR program which calls CSETF the user must reserve space in working storage to store his generated data using the DOPSY ASSIGN command. The number of words to reserve is approximately equal to (the number of SET F statements generated) times ((the highest pin number divided by four) plus two).

// ASSIGN 'SETFn' x Words

If the user is storing his data in a previously used file, the following questions will be displayed on the TTP:

DATA EXISTS IN SETF FILE. UPDATE? (Y/N)

If the response is Y, the old data will be destroyed and the new data will be stored in the selected file. Any other response will cause a return to the DOPSY monitor maintaining the original data in the selected file.

5.1.3 Error Message

If any errors are encountered while executing CSETF, the following message is displayed on the TTP:

CSETF ERROR XXXX

The program is terminated and a return is made to the DOPSY monitor.

ERROR CODE	DESCRIPTION
OPSV	Unable to open SETFNN file (probably not assigned).
WRSV	EOF encountered on SETFNN file (not enough space assigned to file).
CLSV	Unable to close SETFNN file (probably will never occur, system problem).
EXEC	Syntax error in EXEC CSETF statement. The following comment will also be displayed to designate which card is in error: STATEMENT NNNNN

The three following error messages are also generated. They probably will never occur, but if they do, you are either out of disc storage or there is a system problem.

RDWK WRWK OPWK	Unable to READ, WRITE, or OPEN working storage.

5.2 ROMPAT

5.2.1 Introduction

ROMPAT is an assembly language program designed to assist the FACTOR programmer in developing test programs for read only memories. Given a ROM with known inputs the program will generate the output pattern which can then be used to test subsequent ROMs. A device with up to sixty pins having any combinations of input, output, and constant pins can be tested.

5.2.2 Program Usage

The user creates his test program in two steps.

During step 1, the user's FACTOR program will EXECute ROMPAT to generate SET F statements and store them as a string file on disc storage.

During step 2, the EXEC ROMPAT statement is removed from the user's FACTOR program; the program is recompiled and the string file generated in step 1 is INSERTed into the user's program ready for execution.

During step 1, the user may preload local memory with his own address sequence or he may let the ROMPAT program generate all possible address combinations. In either case, after all the commands are loaded into the tester's local memory, the tester is enabled. The SET F commands are executed, and when a fail occurs, the C Register results are ORed in with the original data to create a new SET F command. The new command is stored back into local memory, and the tester is again enabled restarting at local memory start address zero. If any location fails five consecutive times, the test is terminated and a return is made to the DOPSY monitor. Upon reaching the local memory Test End address without a failure the contents of local memory are then converted to TRASCII format and stored as a string file in working storage.

5.2.2.1 ADDRESS GENERATION. In step 1, before calling ROMPAT the user must first set up the conditions defining voltages, timing, mask register, etc. ROMPAT is then called using the FACTOR assembly language linkage statement EXEC. The form is

EXEC ROMPAT (F, FILE, TPINS, IPINS, IP1,...,IPN, CPINs, CP1,...CPn);

where

F = Function

= 1 generate SET F statements subsequently starting at lowest order address going to highest address.

= 2 generate SET F statements in a complementary pattern starting with lowest address then highest address, etc.

```
Example:  SET F  1 Ø Ø Ø. . . . . .
          SET F  Ø 1 1 1. . . . . .
          SET F  Ø 1 Ø Ø. . . . . .
          SET F  1 Ø 1 1. . . . . .
```

= 4 generate SET F statements in a random order.

Any combination of patterns may be generated and stored in the same string file by summing the F parameters for the functions desired; i.e., to generate both sequential and random patterns in the same file F would be set equal to 5 (1 and 4). To generate all patterns in one file, F would be set to 7 (1 and 2 and 4).

FILE = a one or two digit number nn to be appended to the character string "SETF" to create a unique DOPSY file name- 'SETFnn' for storing the data generated in step 1. (See ASSIGN below).

TPINS	= total number of pins
IPINS	= total number of address pins.
IP1 - IPn	= address pin numbers.
CPINS	= total number of pins remaining a constant one.
CP1 - CPn	= constant pin numbers.

Example: EXEC ROMPAT (1, 11, 8, 4, 1, 3, 6, 7, 1, 5);

The data will be generated sequentially and stored in string file 'SETF11'. The device has a total of eight pins. The device has four address pins. The address pins are one, three, six and seven. The device has one pin which will remain a constant one. The constant pin is five.

5.2.2.2 NO ADDRESS GENERATION. The user may preload local memory with his own address sequence and execute ROMPAT to generate the output pattern. The FACTOR statement is

EXEC ROMPAT (F, FILE, TPINS, LMWORDS);

F = Ø No address generated.
FILE = same as above.

TPINS = same as above.
LMWORDS = total number of local memory words to use, i.e., total number of SET F statements to generate in string file (1-1Ø24).

Example: EXEC ROMPAT (Ø, 21, 15, 512);

The data will be generated using previously stored addresses in local memory and stored in string file 'SETF21'. The device has a total of 15 pins. A total of 512 SET F statements will be created.

5.2.2.3 LOCAL MEMORY START ADDRESS. Unless otherwise specified, the functional patterns will be stored in local memory starting at address zero (Ø). However, if the user wishes to start loading his patterns at an address other than zero he can specify the local memory start address with the following FACTOR statement:

EXEC ROMPAT (1ØØ, LMADD);

LMADD = The local memory address to begin storing the functional patterns. In all cases, the string file generated on disc will begin with the contents of local memory zero.

5.2.2.4 ASSIGNING DISC FILES. Prior to executing the FACTOR program which calls ROMPAT, the user must reserve space in working storage to store his generated data using the DOPSY ASSIGN command. To calculate the approximate number of words to be saved, the following formula can be used:

WORDS = (2** IPINS) * (2 + (TPINS/3)) * NUMBER OF FUNCTIONS

For the first EXEC example in 5.2.2.1 above, the following ASSIGN command would first be executed.

//ASSIGN 'SETF11' 8Ø WORDS

5.2.2.5 INSERTING GENERATED DATA. After completing step 1, the user can now recompile his FACTOR program replacing the EXEC statement with an INSERT statement.

Example:

INSERT SETF11;

5.2.3 Error Messages

During step 1 if any errors are encountered, the following message is printed on the output device:

ROMPAT ERROR N

The user program is terminated and a return is made to the DOPSY monitor.

ERROR NUMBER N	DESCRIPTION
1	Error in parameters list in EXEC ROMPAT statement.
2	Unable to open file to store data. ASSIGN statement probably not executed.
3	EOF encountered while writing data to working storage. Not enough storage has been saved with ASSIGN statement.
4	Unable to CLOSE output file.
5	Unable to successfully verify generated SET F statements.
6	Error encountered in generating random addresses.

Errors 1, 2, and 3 are user caused errors and corrective action should be taken by the user and the program rerun.

Errors 4 and 6 indicate errors not within the user's power to correct. There is a problem with the system.

Error 5 can be overcome by using a different device of the same type. This error indicates a defective device.

5.2.4 Summary of events of ROMPAT usage:

1 ASSIGN output file.
2 Compile FACTOR program with EXEC ROMPAT statement.
3 Execute program under TOPSY creating output file.
4 Recompile FACTOR program with INSERT output file statement.
5 Test.

5.2.5 Additional Documentation

SENTRY 600 ROMPAT
Application Note AD 1023

SENTRY 500 ROMPAT
Application Note AD 1052

SENTRY 200 ROMPAT
Application Note AD 1033

5.3 RAMPAT

5.3.1 Introduction

RAMPAT is an assembly language program designed to assist the FACTOR programmer in developing test programs for random access memories. It augments the standard FACTOR instruction set to allow generation of complex function test patterns with only a few user language instructions. This software pattern generator uses standard Sentry 500/600 hardware for test execution. Also RAMPAT supplies special failure analysis datalogging and a plotting routine for failure analysis. The following program description applies to RAMPAT versions Rev 4L and Rev 4S. The "L" and "S" denote long and short versions of the same revision level. The short version does not include the N^2 type patterns, resulting in a savings in computer memory space.

5.3.2 Program Usage

RAMPAT is called using the FACTOR statement EXEC. The general form is:

EXEC RAMPAT (N1, parameter 1, parameter 2, ... parameter n);

5.3.2.1 TESTER PIN DEFINITION. To define tester pins the following values for N1 are used:

N1	
0	Define Device Size
1	Define Row Address Pins
2	Define Column Address Pins
3	Define Data Input Pins*
4	Define Inverse Data Input*
5	Define Data Output Pins
6	Define Inverse Output Pins
7	Define Write Clock Pin
8	Define Pins of Constant 1 Data
9	Define Chip Select Pins
10	Topologically Scramble Rows
11	Topologically Scramble Columns
12	Define Oscilloscope Sync Pin

* For devices where the same pin is used for input and output, specify only the output pin (N1 = 5 or 6) and the input data will automatically be generated.

5.3.2.1.1 For N1 = Ø, define RAM size by
EXEC RAMPAT (Ø, COLUMNS, ROWS, BITS)
where
COLUMNS = number of column address $1 \leq COL \leq 64$
ROWS = number of row addresses $2 \leq ROWS \leq 64$
BITS = number of bits per word $1 \leq BITS \leq 9$

The product of rows and columns must not exceed 2048. RAMPAT must execute this instruction before those listed below.

5.3.2.1.2 For N 1 = 1, define row address pins by
EXEC RAMPAT (1, RØ, R1, R2, . . .)

where RØ = tester pin number of least significant row bit
R1= tester pin number of row bit I,
$I \leq 6$ (2^{I+1} = ROWS)

5.3.2.1.3 For N1 = 2, define column address pins by
EXEC RAMPAT (2, CØ, C1, C2,...)
where CØ = tester pin number of least significant column bit
C1 = tester pin number of column bit I, $I \leq 6$
(2^{I+1} = COLUMNS)

5.3.2.1.4 For N1 = 3, define data input pins by
EXEC RAMPAT (3, DIØ, DI1, . . .)
where DIØ = tester pin number of first bit of memory word
DI1 = tester pin number of bit I of memory word, I<BITS

5.3.2.1.5 For N1 = 4, define complement data input pins by
EXEC RAMPAT (4, IDIØ, IDI1, . . .)
where IDI0 = tester pin number of first complement data in bit of memory word.
IDI1 = tester pin number of bit I or complement data in word, I < BITS

5.3.2.1.6 For N1 = 5, define data output pins by
EXEC RAMPAT (5, DOØ, DO1, . . .)
where DOØ = tester pin number of first output bit of memory word
DO1= tester pin number of bit I of memory word, I < BITS

5.3.2.1.7 For N1 = 6, define complement data output pins by
EXEC RAMPAT (6, IDOØ, IDO1, . . .)
where IDOØ = tester pin number of first complement data out bit of memory word.
IDO1 = tester pin number of bit I of complement data out word, I < BITS

5.3.2.1.8 For N1 = 7, define read/write pin by
EXEC RAMPAT (7, WRITE)
where WRITE = tester pin number of read/write clock

5.3.2.1.9 For N1 = 8, define chip selects or clock pins that require constant ones in local memory by
EXEC RAMPAT (8, P1, P2, . . .)
where P1 = tester pin number requiring a constant one in Local Memory
PI = tester pin I requiring a constant one in Local Memory, I≤6.

5.3.2.1.10 For N1 = 9, define chip select pin and number of addresses for refresh during ping-pong or walking tests for devices requiring write with the chip disabled for refresh
EXEC RAMPAT (9, CENABLE, NADDR)

Where CENABLE = tester pin number of primary chip enable pin.
NADDR = Number of consecutive address required for refresh (typically the number of rows)

5.3.2.1.11 For N1 = 10, define row scramble by
EXEC RAMPAT (10, R0, R1, . . . RI)
Where the sequence of numbers R0, R1, . . . RI is the true geometric order of rows on the chip relative to address definition. (I<ROWS)

5.3.2.1.12 For N1 = 11, define column scramble by
EXEC RAMPAT (11, C0, C1, . . . CI)
Where the sequence of numbers C0, C1, . . . CI is the true geometric order of rows on the chip relative to address definition.

5.3.2.1.13 For N1 = 12, define oscilloscope sync pin by
EXEC RAMPAT (12, SYNC)
Where SYNC is the desired tester pin for synchronizing the scope to the test pattern.

5.3.2.2 PIN DEFINITION EXAMPLE. An example of defining the 1103 - 1024 bit RAM is as follows:

```
EXEC RAMPAT (0, 32, 32, 1); REM 32 ROWS, 32 COLS, 1 BIT;
EXEC RAMPAT (1, 7, 5, 3, 1,  24); REM PRIMARY ROW PINS;
EXEC RAMPAT (2, 14, 12, 16,  21, 10); REM PRIMARY COLUMN
   PINS;
EXEC RAMPAT (3, 20); REM DATA INPUT;
EXEC RAMPAT (6, 23); REM INVERSE DATA OUT;
EXEC RAMPAT (7, 28); REM WRITE CLOCK PIN;
EXEC RAMPAT (8, 9, 26, 29); REM PRECHARGE, CENABLE
  AND READ;
```

Execution of the above statements completely defines the functional pins of the device. The number of row or column pins defined should be consistent with the number of row or column addresses. Note that in the above example, EXEC RAMPAT (4, . . .) or EXEC RAMPAT (5, . . .) were not executed because no inverse data input or true data output pin was used.

Reassignment of pin definition may be made by adding 20 to the first parameter in EXEC. For example, to reverse the order of row addressing

EXEC RAMPAT (21, 24, 1, 3, 5, 7);

or to inhibit the CENABLE pin,

EXEC RAMPAT (28, 9, Ø, 29);

5.3.2.3 SET UP RULES. RAMPAT requires the following rules in the set-up portion of the test program.

1. Set DB to all 1's for input pins, including I/O pins, power, clocks addresses, etc.
2. Set DA equal to DB except a 0 for I/O pins.
3. Set S to 1 for pins using alternate references. This may be to invert the address sequence to reverse the address sequence. Since clocks are RZ, the alternate reference may be necessary to produce the correct polarity on them.
4. Force EB1 equal to EØ.
5. Force EBØ equal to E1.
6. Set the write pin to RZ mode.
7. Set the MA register to all zeroes.
8. SET F Ø; This indicates to the compiler that the local memory was loaded. Actually, RAMPAT will load the local memory.

For example, an 1103 type device would be defined as follows:

```
SET DB 10101010110101010111110011111111;
SET DA 10101010110101010111110010111101;
SET S [9] 1  [26] 1 [28] 1; REM PRECHARGE, CENABLE, WRITE;
SET RZ [9] 1  [26] 1 [28] 1;
FORCE EØ VINLO; FORCE E1VINHI;
FORCE EB1 VINLO; FORCE EBØ VINHI;
SET MA (30: Ø);
SET F Ø;
Other timing, power and reference set up.
```

Here, pin 28, the write pin, is tied to pin 29 on the load board. During a write, pin 28 is a 1 in DA. During a read, Set DA 28 01 lets pin 29 supply a constant bias on that pin. DB was set for both pins 28 and 29, so no relays change state when changing from write to read.

5.3.2.4 PATTERN SELECTION. The next step is to define a pattern to be exercised. RAMPAT has a library of ten patterns. The current patterns are (1) checkerboard, (2) column bars, (3) row bars, (4) diagonal, (5) solid, (6) parity, (7) walking one (zero), (8) ping-pong, (9) shift and (10) surround disturb.

There are two basic types of patterns; Type N, where the total number of tests is directly proportional to the RAM size N, and Type N^2, the walking patterns.

5.3.2.4.1 Type N Pattern Selection. The general form of Type N pattern selection is:

```
For N1 from 100 through 199,
EXEC RAMPAT (N1, N2, N3, N4, N5)
WHERE          N1   PATTERN
               100  ALL ZEROES
               101  CHECKERBOARDS
               102  COLUMN BARS
               103  ROW BARS
               104  DIAGONAL
               105  PARITY
               106  SHIFT
N2 = COLUMN START, THE FIRST COLUMN
     ADDRESSED WITH THE BEGINNING OF EACH NEW
     ROW

N-3            ADDRESS SEQUENCE

Ø              MARCH (Ø, 1, 2, 3,. . .)
1              COMPLEMENT (Ø, Ø/, 1, 1/, 2, 2/)
N4, N5 = DATA FOR SHIFT PATTERN
```

where N4 and N5 represent the data in the first row, to be shifted right by one for each consecutive row. The maximum number of bits per row is 32. Data is right justified and each data word (N4, N5) represents 16 bits of data. For example, if the data pattern for the first row is, starting with column 31,

00111110000011110000111000110010

then N4 = 0, 011, 111, 000, 001, 111 = 037017B

and N5 = 0, 000, 111, 000, 110, 010 = 007062B

For these patterns, the EXEC statement only loads local memory with data. Actual execution of write and read, etc., is done with normal FACTOR statements.

N2 (COLSTR) defines the first column addressed as each new row is entered. This is for many dynamic devices that are refreshed by rows so that the worst case cell (column) in a row may be tested first during refresh testing. Also, COLSTR defines the starting column of row 0 for the diagonal pattern. COLSTR should be less than COLUMNS (starting from zero).

The above statements could be put in a subroutine and called for various patterns or column starts. For example, to do the first six patterns

```
FOR PATT = 100 THROUGH 105 CALL XTEST;
```

or for a walking diagonal

```
PATT = 104;

FOR COLSTR = Ø THROUGH 31 CALL XTEST;
```

or for critical refresh column address tests

```
PATT = 101;

FOR COLSTR = Ø THROUGH 31 CALL XTEST;
```

Other pattern variations can easily be done. For example, to reverse the addressing sequence, switch to the alternate address line by

```
SET S [1] 1 [3] 1. . .; or

SET INV . . . . ; with 1 ns. option
```

Or, the least significant address bit may be held constant so that only odd or even rows are addressed during a refresh disturb test.

```
FORCE EB1 VINHI; FORCE EBØ VINHI;
SET S [7] 1; REM ALT REF ON LSB ADDRESS;
SET MINOR 1ØØ; REM LOOP FOR 54 MSEC;
ENABLE TEST; REM DISTURB ODD ROWS;
SET S [7] 0;
SET MINOR 1;
ENABLE TEST; REM READ ALL ROWS;
```

5.3.2.4.2 Example of Type N Pattern Execution. The sequence of events is given in the example below. Here pin 29 provides bias on the write line during a read.

```
SET MAJOR 1, SIZE - 1;
SET MINOR 1, 0, SIZE - 1;
EXEC RAMPAT (PATT, COLSTR, SEQUENCE);
ENABLE TEST; REM WRITE PATTERN;
SET DA [28] Ø 1; REM DISABLE WRITE CLOCK;
SET MA [23] 1; REM ENABLE STROBE;
ENABLE TEST; REM READ OUT PATTERN;
```

To execute the complement of the above test, reverse the S1, S0 sense levels, reverse data in by selecting the alternate reference supplies (EB1/EBØ) with the S register and define the negative logic mode.

```
SET S1 VOL; SET SØ VOH;
SET LOGIC NEG;
SET S [20] 1; REM INVERT DATA IN;
SET DA [28] 1Ø; REM ENABLE WRITE;
SET MA [23] Ø; REM DISABLE STROBE;
ENABLE TEST; REM WRITE COMPLEMENT PATTERN;
SET DA [28] Ø1;
SET MA [23] 1;
ENABLE TEST; REM READ COMPLEMENT PATTERN;
```

5.3.2.5 TEST EXECUTION WITHIN RAMPAT. RAMPAT also has a set of special instructions where testing is actually executed by RAMPAT. That is, in the previously described patterns for N1 from 100 to 199, RAMPAT only generated the data and deposited the results in the Local Memory. Actual test execution was initiated by the user's FACTOR program instructions. However, some tests require repetitive execution, so those cases are handled automatically by RAMPAT.

These functions are

N1	FUNCTION
200	Ping-Pong Walking One (Zero)
202	RAM Fail Matrix Datalog
203	Walking One (Zero)
204	Surround Disturb

5 3.2.5.1 Ping-Pong Test For N1 = 200, Execute Ping-Pong Walking Pattern By
EXEC RAMPAT (2ØØ, LOG, N3);
where LOG = 1 Enables Ping-Pong Datalogger
LOG = Ø Disables Ping-Pong Datalogger
N3 = Ø for no special refresh or I/O
= 1 for special refresh mode where the chip is disabled and DUT written to refresh. Must define CENABLE with EXEC RAMPAT (9, . . .)

The ping-pong test executes a one walking through a field of zeroes with all possible address transitions. Initially, all cells are written to 0. Then a one is written into the first. The flow chart shows the sequence of events of a ping-pong test. The number of tests is approximately $2N^2$. Counting both true and complementary cases, there are $4N^2$ tests. Execution is as follows:

```
LOG = Ø;
EXEC RAMPAT (200, LOG, Ø);
```

ALTER may be used to modify LOG to a 1 to enable the ping-pong datalogger. The reason for a special datalog here is due to the nature of ping-pong testing. Here, all possible address transitions are being tested. So, a failure of one cell may be a function of the previous cell tested, i.e., the transition of bits in the address decoders may have caused the fail. For example,

> PING PONG TEST FAILED TO READ A 1 IN LOCATION COL 19 ROW 1 FOLLOWING A READ Ø IN LOCATION COL 2 ROW 5
> BACKGROUND WAS A Ø

5.3.2.5.2 RAM Fail Matrix Datalog Of Type N Patterns. The second case of test execution generated by RAMPAT is a special datalog feature for displaying up to 1024 bits of data by row and column coordinates. This is called the RAM Fail Matrix Datalog. For multiple bit RAMs, enable only one output pin at a time for use with this display.

> FOR N1 - 2Ø2, EXECUTE DYNAMIC MEMORY FAIL MAP BY EXEC RAMPAT (2Ø2);
> For output on the line printer, set LP in DOPSY or datalog LP in TOPSY. For multibit word RAMs, enable only one output pin with SET MA for the fail map on that pin.

The Fail Matrix is currently limited to RAMS with rows and columns less than or equal to 32.

5.3.2.5.3 Walking One (Zero) Test

> FOR N1 = 2Ø3, EXECUTE THE WALKING ONE (ZERO) PATTERN BY EXEC RAMPAT (203,LOG,N3);
>
> Where
>
> N3 = 0 for no special refresh or I/O
> = 1 for special refresh mode where the chip is disabled and the DUT written to refresh. Must define CENABLE with EXEC RAMPAT (9, . . .)
> = 2 for I/O devices where the input drivers must be disabled during the read cycle.
> LOG = 1 enables fail matrix datalogger
> 0 disables datalog for this test

The walking test writes a 0 in all cells. Then a one is written in a cell and all cells from 0 to SIZE - 1 are read. Then the cell with the one is restored to zero and a one written into the next cell, etc. Counting both the true (walking one) and complement (walking zero), a total of about 2N(N + 2) tests is executed. If LOG is 1, the fail matrix datalogger is automatically called if a fail is detected. This is the same datalogger as number 202 and will show the location of the one in the field of zeroes and the failing cell.

5.3.2.5.4 Surround Disturb Pattern. For N1 = 204 execute the surround disturb test by

> EXEC RAMPAT (204, LOG);

Since this instruction is in the 200 category, this means that the local memory pattern generation and execution are done within RAMPAT (no FACTOR Enable Test required).

The pattern loaded in local memory will be as follows:

Ø Write a Ø in Row M, Column N - 1
1 Write a Ø in Row M, Column N
2 Write a Ø in Row M, Column N + 1
3 Write a 1 in Row M - 1, Column N
4 Read a Ø in Row M, Column N + 1
5 Write a 1 in Row M + 1, Column N
6 Read a Ø in Row M, Column N - 1
7 Read a Ø in Row M, Column N

The user must define minor and major loop constants before executing the surround disturb pattern by

```
SET MINOR C, 3, 6;
SET MAJOR 1, 7;
EXEC RAMPAT (204, LOG);
```

RAMPAT will start with M = 0, N = 0. In this case, M - 1 or N - 1 will wrap around to be the maximum row or column address. The pattern will be stored in local memory and executed. The minor loop count, C, should equal the number of times the surround disturb of cell M, N will be exectued. For example, if the cycle time is 900 nanoseconds, and the desired disturb time is 1.8 milliseconds, then C should be 500.

During execution of the first pattern, the software computes the local memory data for the next value of M or N. When the first pattern is complete and if no failure occurred, the next pattern is loaded and executed in DMA (Direct Memory Access) mode. All values of M and N are done. When M or N equals the maximum row or column address M + 1 or N + 1 will be zero (wrap around). A failure in any cell will terminate the test. If LOG is one in the EXEC statement, the RAM Fail Matrix Datalogger will be called (the same as EXEC RAMPAT (202).

The complement surround disturb may be executed by inverting data in and out as previously discussed.

5.3.2.6 SHMOO PLOTTING. Two other routines are contained within RAMPAT for special datalogging features.

```
EXEC RAMPAT (40, ARRAY);
```

where ARRAY is a FACTOR array of 61 elements. RAMPAT prints a line of 61 characters with a space or blank where the ARRAY element is Ø and an X otherwise.

5.3.2.6.1 Line Printer Top of Form. Another RAMPAT statement does a top of form on the line printer for plotting preparation

```
EXEC RAMPAT (41);
```

5.3.2.6.2 Shmoo Plot Example. A complete plot of device performance for all values of two parameters may be executed as follows:

```
EXEC RAMPAT (41); REM TOP OF FORM;
WRITE 'header informaiton';
FOR PAR1 = P1MAX THRU P1MIN BY DELP1
DO BEGIN
.
.
.
define values that are a function of PAR1
.
.
I = 0
FOR PAR2 = P2MIN THRU P2MAX BY DELP2
DO BEGIN
.
.
define values that are a function of PAR2
.
.
I = I + 1; REM TPAS ARRAY ELEMENT;
ON FCT, FAIL1; REM BRANCH TO FAIL1 IF FAILURE;
CALL XTEST; REM TEST SUBROUTINE;
TPAS  I  =  1; REM SET ELEMENT TO 1 IF PASS;
GOTO PASS1;
FAIL 1:  TPAS  1 = Ø
PASS1:   END;
EXEC RAMPAT (40, TPAS); REM PRINT DATA;
END;
WRITE 'bottom header information';
```

5.3.2.7 MULTIPLE BIT WORDS. For testing RAMs with multiple bit words, the word bit relations are defined by selecting combinations of data in and inverse data in. For example,

```
EXEC RAMPAT (3, BIØ, BI1, BI2, BI3)
EXEC RAMPAT (5, BOØ, BO1, BØ2, BO3)
```

will make all four bits of the word equal. To define alternate 1, 0, 1, 0 for the word,

```
EXEC RAMPAT (3, BIØ, Ø, BI2, Ø);
EXEC RAMPAT (4, Ø, BI1, Ø, BI3);
EXEC RAMPAT (5, BOØ, O, BO2, Ø);
EXEC RAMPAT (6, Ø, BO1, Ø, BO3);
```

This makes BIØ = BI2 = BI1 = BI3

Other combinations can be defined by this method.

5.3.3 RAMPAT Terminal Errors

The following terminal errors will be produced by RAMPAT in the event of a user error in the calling procedure.

TERMINAL ERROR	DESCRIPTION
101	Illegal value used for N1 in EXEC statement
102	Illegal value used for N2 and/ or N3 in EXEC statement (ROWS x COLUMNS exceeds size of Local Memory).
103	Too many parameters used in EXEC statement (too many pins specified for data in, out, etc.).
104	RAM size improperly defined by N1 =Ø procedure. RAMPAT may have been removed from computer memory by execution of another Assembly Language Overlay.

5.3.4 Additional Documentation

SENTRY 500/600 RAMPAT Application Note AD 1013
SENTRY 200 RAMPAT Application Note AD 1002

5.4 ROMPONG

5.4.1 Introduction

ROMPONG is an assembly language program designed to assist the FACTOR programmer in testing access times of read only memories. The program performs a ping-pong type test on a functional pattern testing the access time from every device address to every other address. Also, ROMPONG supplies optional failure analysis datalogging.

5.4.2 Program Usage

5.4.2.1 CALLING SEQUENCE. The FACTOR calling sequence is:

EXEC ROMPONG (SIZE, LOG);

where

SIZE = the number of functional patterns loaded into local memory (less than or equal to one half the local memory size)

LOG = 0 - do not log device failure data

1 - log failure data on system primary output device

5.4.2.2 OPERATING PROCEDURE. The user first loads his functional test pattern into local memory starting at location zero. The test pattern cannot exceed one half the system's local memory size. ROMPONG is then invoked using the assembly language linkage statement EXEC ROMPONG described above.

The first task performed by ROMPONG is to expand the truth table of the ROM into the even numbered locations of local memory.

The pattern for address Ø remains at location Ø.
The pattern for address 1 is stored at location 2.
The pattern for address 2 is stored at location 4.
.
.
.
The pattern for address N is stored at location 2N.

Upon completion of the relocation of local memory, the accessing test is begun. The contents of local memory location Ø are stored into all odd locations from 1 to 2N + 1 and the test is "ENABLED". The contents of location 2 are then written into all odd locations and again the test is "ENABLED". This procedure is repeated until the acccess time from every device address to every other address is tested or until a failure occurs.

In either case, upon termination, local memory is restored to its original state and a return is made to the user's FACTOR program.

5.4.2.3 FAILURE ANALYSIS. If the user selects to log his failure results, the following data is output

```
    IND= 000452      ADDR= 021760      SIZE= 000545
INSTRUCTION   000467          LOC. MEM.    0012    ITERATION     0350
INSTRUCTION   000467          LOC. MEM.    0012    ITERATION     0350
FAILED SET F      RANKS 1-2   01010 00110 10000   01110 00000 00000
                  RANKS 3-4   00000 00000 00000   00000 00000 00000
COMPARISON REG.   RANKS 1-2   01000 00100 00000   00000 00000 00000
                  RANKS 3-4   00000 00000 00000   00000 00000 00000
PREVIOUS SET F    RANKS 1-2   00001 10000 01011   11110 00000 00000
                  RANKS 3-4   00000 00000 00000   00000 00000 00000
INSTRUCTION   000467          LOC. MEM.    0754    ITERATION     0317
FAILED SET F      RANKS 1-2   00000 00001 10111   11110 00000 00000
                  RANKS 3-4   00000 00000 00000   00000 00000 00000
COMPARISON REG.   RANKS 1-2   01000 00000 00000   00000 00000 00000
                  RANKS 3-4   00000 00000 00000   00000 00000 00000
PREVIOUS SET F    RANKS 1-2   01000 00011 11001   11110 00000 00000
                  RANKS 3-4   00000 00000 00000   00000 00000 00000
```

INSTRUCTION = the FACTOR statement number of EXEC ROMPONG statement which failed.

LOC. MEM. = the local memory location of the failed test pattern.

ITERATION = the number of times the odd locations have been changed. This is an octal number and also indicates the local memory location of the original functional pattern as loaded into local memory by the user.

5.4.3 Error Messages

ERROR MESSAGE	DESCRIPTION
ROMPONG ERR EXEC	An error is encountered in the EXEC ROMPONG statement. The user program is terminated and a return is made to the DOPSY monitor.

INDEX

ACCESS, 2.7, 2-58

BMT, 1.13, 1-41

CHANGE, 1.8, 1-12
COPJOB, 1.7, 1-9
CRDTAP, 1.3, 1-4
CSETF, 5.1, 5-1
CYCLE, 3.4, 3-18

DATAIO, 2.7, 2-42
DBUP, 1.11, 1-30
DEBUG, 3.1, 3-1
DELJOB, 1.6, 1-7

EDIT, 1.9, 1-15

FCOMP, 1.1, 1-1
FINDJOB, 1.5, 1-6
FMTAP, 4.7, 4-19

GLOBS, 4.8, 4-20

INIT, 1.14, 1-43
INSERT, 1.15, 1-45

LABEL, 1.17, 1-49
LISTC, 1.2, 1-3
LMIO, 3.3, 3-12
LMLOAD, 4.4, 4-6
LMMOD, 4.1, 4-1
LMSAVE, 4.5, 4-10
LMTSF, 4.2, 4-3
LOGREG, 4.6, 4-18
LPLF, 4.3, 4-5

NOTE, 1.16, 1-46

PATCH, 1.10, 1-28
PGLOG, 2.4, 2-27
PPLOG, 2.5, 2-29
PSCAN, 3.2, 3-7

RAMPAT, 5.3, 5-8
ROMPAT, 5.2, 5-4
ROMPONG, 5.4, 5-19

SPLOT, 2.3, 2-11

TAPLP, 1.4, 1-5
TDX, 1.12, 1-34
TTIME, 2.2, 2-8

XGRAPH, 2.1, 2-1
XMIT, 1.18, 1-48

MANUAL REGISTRATION FORM
AND COMMENT SHEET

FILL OUT AND MAIL TO RECEIVE UPDATES AND SUPPLEMENTS AUTOMATICALLY AS THEY ARE PRINTED.

FROM:

NAME _____

TITLE _____

BUSINESS ADDRESS _____

_____

_____

BUSINESS PHONE _____

MANUAL PART NO. 67095661

REVISION NO. 1

DATE OF PUBLICATION June 1977

COMMENTS:

(Please describe errors, suggested additions or deletions, and reference part number, page number, paragraph number, or drawing number.

ERRORS, OR CHANGE SUGGESTED, ON PAGE(S)

_____

_____

_____

_____

_____

_____

_____

_____

_____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

Fold Along Dotted Line

FIRST CLASS
PERMIT NO. 5699
San Jose,
California

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

Postage Will Be Paid By

FAIRCHILD SYSTEMS TECHNOLOGY
1725 TECHNOLOGY DRIVE
SAN JOSE, CA 95110

ATTN: SYSTEMS TECHNICAL PUBLICATION DEPT.

Fold Along Dotted Line