

Programming Support Library (PSL)
Maintenance Manual - Volume I

ADA107252

IBM

(9) Final rept.

REPORT DOCUMENTATION PAGE		1. REPORT NO. 19	2. Recipient's Accession No. AD-A107 252
4. Title and Subtitle		11. Report Date May 1978	12. 304
6. PROGRAMMING SUPPORT LIBRARY (PSL) Maintenance Manual, Volume I		15. F30602-77-C-0249	
7. Author(s)		16. Project/Task/Work Unit No.	
9. Performing Organization Name and Address Federal Systems Division International Business Machines Corporation Gaithersburg, Maryland		17. Contract/Grant/Order No.	
12. Sponsoring Organization Name and Address Rome Air Development Center RADC/COEE Griffiss Air Force Base, NY 13441		18. Type of Report & Period Covered	
13. Supplementary Notes (2) For magnetic tape, see AD-A107248. See also Volume 2, AD-A107 253.		The source agency has restricted sales of this item to Federal, state and local governments.	
14. Abstract (Limit: 200 words) The Programming Support Library (PSL) is a software system which provides the tools to organize, implement, and control computer program development. This involves the support of the actual programming process and also the support of the management process. The PSL is designed to support Top Down Design and Structured Programming (TDDSP). This Maintenance Manual provides detailed information on PSL program operations, data formats and special procedures to assist programmer personnel in the maintenance of PSL system programs.			
17. Document Analysis a. Descriptors Software Configuration Control Structured Programming Support Tool Software Development Support b. Identifier/Open-Ended Terms c. COSATI Field/Group 174950			
18. Availability Statement		19. Security Class (This Report) UNCLASSIFIED	20. No. of Pages
		21. Security Class (This Page) UNCLASSIFIED	22. Price

DTIC
ELECTED
NOV 17 1981

DMA
Programming Support Library (PSL)

Program
Maintenance Manual

(FINAL)

May 1978

Submitted to:

Rome Air Development Center
Griffis Air Force Base, New York

Under Contract F30602-77-C-0249

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
NTIS	
Distribution/	
Availability Codes	
Dist and/or	
Special	
A	21

Federal Systems Division
INTERNATIONAL BUSINESS MACHINES CORPORATION
Gaithersburg, Maryland

TABLE OF CONTENTS

<u>SECTION</u>		<u>PAGE</u>
1	GENERAL DESCRIPTION	1-1
1.1	Purpose of the Program Maintenance Manual	1-1
1.2	System Application	1-1
1.3	Equipment Environment	1-2
1.4	Programming Environment	1-2
1.5	Conventions	1-2
1.6	Conversion Considerations	1-4
2	SYSTEM DESCRIPTION	2-1
2.1	General Description	2-1
2.1.1	System Organization	2-1
2.1.2	System Operation	2-3
2.1.3	Program Organization	2-4
2.2	Detailed Description	2-17
2.2.1	BCTL - Batch Control	2-19
2.2.1.1	Function Processors	2-23
2.2.1.1.1	Library Maintenance	2-23
2.2.1.1.1.1	ITPJ - Initialize a Project	2-24
2.2.1.1.1.2	CRSC - Create a Section	2-27
2.2.1.1.1.3	BKLB - Backup Library	2-31
2.2.1.1.1.4	RSLB - Restore Library	2-33
2.2.1.1.1.5	PGSC - Purge a Section	2-38
2.2.1.1.2	Unit Maintenance	2-42
2.2.1.1.2.1	ADUN - Add a Unit	2-43
2.2.1.1.2.2	RPUN - Replace a Unit	2-48
2.2.1.1.2.3	CHUN - Change a Unit	2-51
2.2.1.1.2.3.1	CHRC - Edit Change Records	2-55
2.2.1.1.2.3.2	CHLN - Change Unit Lines	2-61
2.2.1.1.2.4	MVUN - Move a Unit	2-66
2.2.1.1.2.5	PGUN - Purge a Unit	2-70
2.2.1.1.3	Program Processing	2-73
2.2.1.1.3.1	CMML - Compile a Module	2-74
2.2.1.1.3.1.1	PPCB - Preprocess COBOL	2-77
2.2.1.1.3.1.2	PPFT - Preprocess FORTRAN	2-86
2.2.1.1.3.1.3	PPJV - Preprocess JOVIAL	2-88
2.2.1.1.3.2	LKPG - Link a Program	2-103
2.2.1.1.3.2.1	PPLK - Preprocess Link	2-107
2.2.1.1.3.3	EXPG - Execute a Program	2-110
2.2.1.1.3.4	PCML - Precompile Module	2-116
2.2.1.1.4	Management Data	2-118
2.2.1.1.4.1	FMMD - Format Management Data	2-119
2.2.1.1.4.1.1	FMED - Edit Format Records	2-124
2.2.1.1.4.1.2	FMV - Move Format Records	2-128
2.2.1.1.4.1.3	FMUP - Update Format Records	2-132

TABLE OF CONTENTS (continued)

<u>SECTION</u>		<u>PAGE</u>
2.2.1.1.4.2	UPMD - Update Management Data	2-135
2.2.1.1.4.2.1	UPMF - Edit Data Format	2-139
2.2.1.1.4.2.2	UPMR - Update Data Records	2-144
2.2.1.1.4.3	ITCL - Initiate Collection	2-147
2.2.1.1.4.3.1	CLMD - Collect Management Data	2-149
2.2.1.1.4.3.1.1	PPCL - Preprocess Collection	2-152
2.2.1.1.4.3.1.2	PCCL - Process Collection	2-156
2.2.1.1.4.3.1.3	UPCL - Update Collection	2-165
2.2.1.1.4.4	PRMR - Print Management Report	2-167
2.2.1.1.4.4.1	PRPS - Print Program Structure	2-170
2.2.1.1.4.4.2	PRMD - Print Management Data	2-173
2.2.1.1.5	Output Processing	2-176
2.2.1.1.5.1	PRXX - Print an Index	2-177
2.2.1.1.5.2	PRSD - Print Source Data	2-182
2.2.1.1.5.3	PRAU - Print Author Report	2-185
2.2.1.1.5.4	PRDC - Print Document	2-190
2.2.1.1.5.5	SCSG - Scan Character String	2-193
2.2.1.1.6	General Functions	2-197
2.2.1.1.6.1	ESPA - Establish Parameters	2-197
2.2.1.1.6.2	WRJB - Write Job Control Cards	2-198
2.2.1.1.6.3	PFJB - Perform Job Control Cards	2-201
2.2.1.2	Support Routines	2-212
2.2.1.2.1	BKSC - Backup a Section	2-213
2.2.1.2.2	Unit Processing	2-216
2.2.1.2.2.1	PCLU - Process Lower Units	2-216
2.2.1.2.2.2	DLUN - Delete a Unit	2-220
2.2.1.2.3	PRUN - Print a Unit	2-223
2.2.1.2.3.1	PRCB - Print COBOL	2-226
2.2.1.2.3.2	PRFT - Print FORTRAN	2-235
2.2.1.2.3.3	PRNI - Print with No Indentation	2-245
2.2.1.2.3.4	PRJV - Print JOVIAL	2-247
2.2.1.2.4	Top Down Read	2-254
2.2.1.2.4.1	RDCM - Read for Compile	2-255
2.2.1.2.4.2	RDPS - Read for Program Structure	2-259
2.2.1.2.5	Process Message	2-261
2.2.1.2.5.1	PRMS - Print a Message	2-262
2.2.1.2.5.1.1	PRM1 - Print Type 1 Message	2-265
2.2.1.2.5.1.2	PRM2 - Print Type 2 Message	2-267
2.2.1.2.5.1.3	PRM3 - Print Type 3 Message	2-269
2.2.1.2.5.1.4	PRM4 - Print Type 4 Message	2-271
2.2.1.2.5.2	PRER - Print Errors	2-274
2.2.1.2.5.3	PRMSI - Print Message in Spawned Job	2-276
2.2.1.2.6	Obtain Input Card	2-279
2.2.1.2.6.1	OBFN - Obtain a Function	2-280
2.2.1.2.6.2	OBKW - Obtain Keyword	2-283
2.2.1.2.6.3	OBSD - Obtain Source Data	2-287

TABLE OF CONTENTS (continued)

<u>SECTION</u>		<u>PAGE</u>
2.2.1.3	PSL Access Routines	2-291
2.2.1.3.1	Basic I/O	2-291
2.2.1.3.1.1	ITSF - Initialize a Standard File	2-292
2.2.1.3.1.2	ACFL - Access a File	2-297
2.2.1.3.1.2.1	ASFL - Assign a File	2-298
2.2.1.3.1.2.2	RLFL - Release a File	2-305
2.2.1.3.1.2.3	RLAF - Release All Files	2-307
2.2.1.3.1.3	ACBK - Access a Block	2-309
2.2.1.3.1.3.1	ASBK - Assign a Block	2-310
2.2.1.3.1.3.2	RDBK - Read a Block	2-312
2.2.1.3.1.3.3	WRBK - Write a Block	2-314
2.2.1.3.1.3.4	RLBK - Release a Block	2-317
2.2.1.3.2	Unit I/O	2-320
2.2.1.3.2.1	WRAC - Write Accounting Information	2-320
2.2.1.3.2.2	RDUN - Read a Unit	2-322
2.2.1.3.2.2.1	ITRD - Initialize to Read	2-323
2.2.1.3.2.2.2	RDLN - Read a Line	2-327
2.2.1.3.2.2.3	TMRD - Terminate Read	2-332
2.2.1.3.2.3	WRUN - Write a Unit	2-334
2.2.1.3.2.3.1	ITWR - Initialize to Write	2-335
2.2.1.3.2.3.2	WRLN - Write a Line	2-340
2.2.1.3.2.3.3	TMWR - Terminate Write	2-344
2.2.1.3.3	Index I/O	2-346
2.2.1.3.3.1	ADXE - Add an Index Entry	2-347
2.2.1.3.3.2	CHXE - Change an Index Entry	2-351
2.2.1.3.3.3	FDXE - Find an Index Entry	2-354
2.2.1.3.3.4	DLXE - Delete an Index Entry	2-357
2.2.1.3.3.5	RDXE - Read an Index Entry	2-360
2.2.1.4	Executive Operating System Interface Routines	
	Routines	2-364
2.2.1.4.1	Setup Parameter Routines	2-364
2.2.1.4.1.1	ALLOCATOR - Process Parameters	2-365
2.2.1.4.2	Executive Request Routines	2-377
3.	INPUT/OUTPUT DESCRIPTION	3-1
3.1	General Description	3-1
3.2	Characteristics and Description of System Data	3-2
3.2.1	PSL Function Cards	3-2
3.2.2	Types of Data Sections and Contained Data	3-2
3.2.3	Management Data	3-5
3.3	PSL Copy Library	3-12

TABLE OF CONTENTS (continued)

<u>SECTION</u>		<u>PAGE</u>
4	PROGRAM ASSEMBLING, LOADING, and MAINTENANCE PROCEDURES	4-1
APPENDIX A.	PROGRAM STRUCTURE MAPS	A-1
APPENDIX B.	LIBRARY PSLPRG LINKS	B-1
APPENDIX C.	PSLPRG ELEMENT TABLE	C-1

LIST OF TABLES

<u>TABLE</u>		<u>PAGE</u>
2-a	PSL Sections, Options and Default Values	2-29
2-b	Determination of Default Unit Type	2-47
2-1	Index Report Data Items and Source	2-179
2-2	Author Report Data Items	2-187
2-3	Data Items from the String Scan Report	2-195
3-1	Organization of a PSL Section	3-4
3-2	System Level Management Data Input	3-6
3-3	Subsystem Level Management Data Input	3-8
3-4	Module Level Management Data Input	3-9
3-5	Unit Level Management Data Input	3-10
3-6	Job Level Management Data Input	3-11
3-7	Management Data Base Structure	3-13

LIST OF FIGURES

<u>FIGURE</u>		<u>PAGE</u>
2-01	Hierarchical Structure of a Project	2-2
2-02	PSL System Flowchart	2-5
2-03	PSL Program Organization	2-6 thru 2-16
2-04	Code Generated by COBOL Precompiler PPCB	2-81
2-05	Code Generated by JOVIAL Precompiler PPJV	2-90
2-06	Code Generated by PPGL for COBOL Modules	2-99
2-07	Code Generated by PPGL for FORTRAN Modules	2-100
2-08	Code Generated by PPGL for ASM Modules	2-101
2-09	Code Generated by PPGL for JOVIAL Modules	2-102

ABSTRACT

This Program Maintenance Manual provides detailed information on PSL program operations, data formats and special procedures to assist programmer personnel in the maintenance of DMA PSL system programs.

SECTION 1. GENERAL DESCRIPTION

1.1 Purpose of the Program Maintenance Manual

The objective for writing this Program Maintenance Manual for the Defense Mapping Agency (DMA) Programming Support Library (PSL) is to provide the maintenance programmer personnel with the information necessary to effectively maintain the PSL software system.

1.2 System Application

The DMA PSL is a comprehensive system software package which supports the growth and maintenance of structured programming projects in a top-down development environment. The system provides:

- a. A framework for the organization of a project
- b. Simple functional statements to interface between the programmer and the machine
- c. Structural and statistical reports for control of the development process and for communication between programmers.

The DMA PSL system provides a special structured programming support for the Structured COBOL, Structured FORTRAN and Structured JOVIAL languages. However, unstructured programs may also be stored and maintained under the system.

The PSL system in the batch mode is invoked by an @ADD, @END, @ADD sequence and is directed to perform specific functions by PSL Function cards. The general functional capabilities available under the PSL are:

- a. Initialize a project
- b. Create sections in a library
- c. Add, change, move, replace, or purge a unit of code
- d. Print, punch, or write (to tape) a unit of code
- e. Print an index listing
- f. Print the top-down structure of a program
- g. Compile, link, execute

- h. Delete a section, a library, or a project
- i. Backup a project, library, or section
- j. Restore a project, library, section, or unit
- k. Collect and print management data
- l. Print textual material
- m. Print by author or character string.

Equipment Environment

PSL system operates on a Sperry Univac 1100 series computer with the Exec 8 Operating System using a standard DMA software hardware configuration. Libraries are maintained on direct access storage devices. The system utilizes the standard card reader and printer, one tape drive, and 40K words of core.

Programming Environment

PSL system utilizes the Univac Executive Operating System to dynamically create, allocate, deallocate and purge file storage space. Interface with the Executive system is made through Executive Request Control statements using the Univac assembler (FIELDATA) language.

Conventions

PSL system was developed using top-down, structured code techniques as described in Section 1 of the PSL User's Manual. The resulting PSL software is best described as an hierarchical organization of modules (i.e., compilable aggregations of code) which perform the many functions required in PSL operations. Each module, in turn, is composed of INCLUDED or PERFORMED units of code. Modules coded during Phase II of PSL development were designed to utilize the Phase I PSL software to maintain included units of code whereas Phase I PSL code was forced to use the LIMITED PERFORM capability for archiving in-line code performance. In either case, the included or performed units of code are designed to be generally 50 lines of code (i.e., page size) or less. Most of the subject code is printed using a PSL structured print routine which generates indented lines of COBOL code in accordance with its structure. The resulting listings are designed to read and comprehend once the general purpose of the units (paragraphs) of code are understood. In light of the above, the following rules, schemes and conventions are utilized in developing and describing the PSL system.

a. Module Identification

Module names are up to six characters in length and generally have mnemonic content that corresponds to the function performed. In most cases, modules written in structured COBOL (SCOBOL) or ASM have four character mnemonic names to which an "E" is appended, denoting "entry point". This convention was employed to cause the module name to be the same as the module entry print name. Module names of this composition utilize a pair of two-character mnemonics to denote their function. A consistent usage of these two-character mnemonics is maintained. For example, "DL" always is used to denote "delete". The mnemonic name of each module is paired with a descriptive name in Section 2 of this manual.

b. Unit Identification

Units (or paragraphs) of code are given up to thirty (30) character names which are descriptive of the functions performed by the code. A unit of code (as opposed to a paragraph) has its functional name prefixed by the mnemonic name of the module in which it is INCLUDED. The top-unit (or paragraph) of each module of code generally contains INCLUDEs or PERFORMs to reference the component functions of the module. A substantial understanding of module operations is usually gained at the top-level of code and immediately below. The Hierarchy plus Input-Process-Output (HIPO) diagrams given in Section 2 render a descriptive statement of the top-logic for each module combined with a visual picture of the input and output processing performed.

c. HIPO Diagram Conventions

Input and Output files, tables and parameters will be noted in the following manner:

- Subroutine USING and GIVING parameters will be enclosed in brackets: [Subroutine Parameter]
- Temporary file and table names will be preceded by a bullet: • Temporary File
- Permanent file names will appear in a boxed area: Permanent File

Functional steps in the Process section of the HIPO diagrams will be sequentially numbered for purposes of identification. When further detail on a process step is necessary, the HIPO diagram containing that detail is noted in the step. The functional description in each HIPO diagram are readily corresponded with the PSL program listings (provided as an appendix to this manual) when a more detailed knowledge of program operations is required.

1.6 Conversion Considerations

The DMA PSL system was created by modifying a version of the PSL which was installed on a non-Univac computer system. The conversion process necessitated changes to the PSL system as follows:

- a. Each entry point of multi-entry point modules required an interface routine to provide the correct parameter string. These entry points are:

OBFN	RDBK	ALFL
OBKW	WRBK	CHFL
OBSD	RLBK	CRFL
ASFL	ITRD	DAFL
RLFL	RDLN	PGFL
RLAF	ITWR	QYFL
ASBK	WRLN	SPJB

- b. PSL random blocks are numbered 0,1,2...n, while Univac random blocks are numbered 1,2,...n+1. This difference was reconciled by using an index key equal to the PSL block number plus one when I/O was required on PSL random blocks.
- c. Due to differences in parameter passing, certain temporary variables were needed to save parameters upon entry and restore them upon exit from a module. These variables are indicated by HOLD-parameter-name. Several complicated parameters were moved to the Working Storage section and temporary Linkage parameters were used. These temporary Linkage parameters are indicated by parameter-name A.

SECTION 2. SYSTEM DESCRIPTION

2.1 General Description

The PSL system is designed to support the program development and maintenance effort of an entire organization. In a large organization, the system must support many persons working on different programming projects which may be completely unrelated. The PSL provides means of maintaining control over all the data related to each project.

2.1.1 System Organization

One means of control is the convention used for identifying and organizing the data (source code, compiled modules, test data, etc.) stored under a project. This convention subdivides the data, beginning with the programming project level and proceeding down to single logical units of data. The hierarchical structure of a project is shown in Figure 2-01.

There are four levels of data identification under the PSL. The highest level of identification is a project. This corresponds to a major programming operation and consists of all of the data related to that operation.

The next level of identification is a library. Each project is composed of one or more libraries. Multiple libraries can be used effectively to provide version control over the programming process and to support top-down program development and integration. Any number of libraries may exist under a given project.

A library is composed of segments of programming data grouped according to type of data. Each type is stored in a specific section of a library. The names of a section is one of the following predefined section names:

a.	SOURCE	-	source code statements
b.	OBJECT	-	object module indexes and accounting records
c.	LOAD	-	load module indexes and accounting records
d.	LINK	-	collector control cards
e.	JOB	-	job control cards used during execution of user program
f.	TEST	-	test data for use by user program

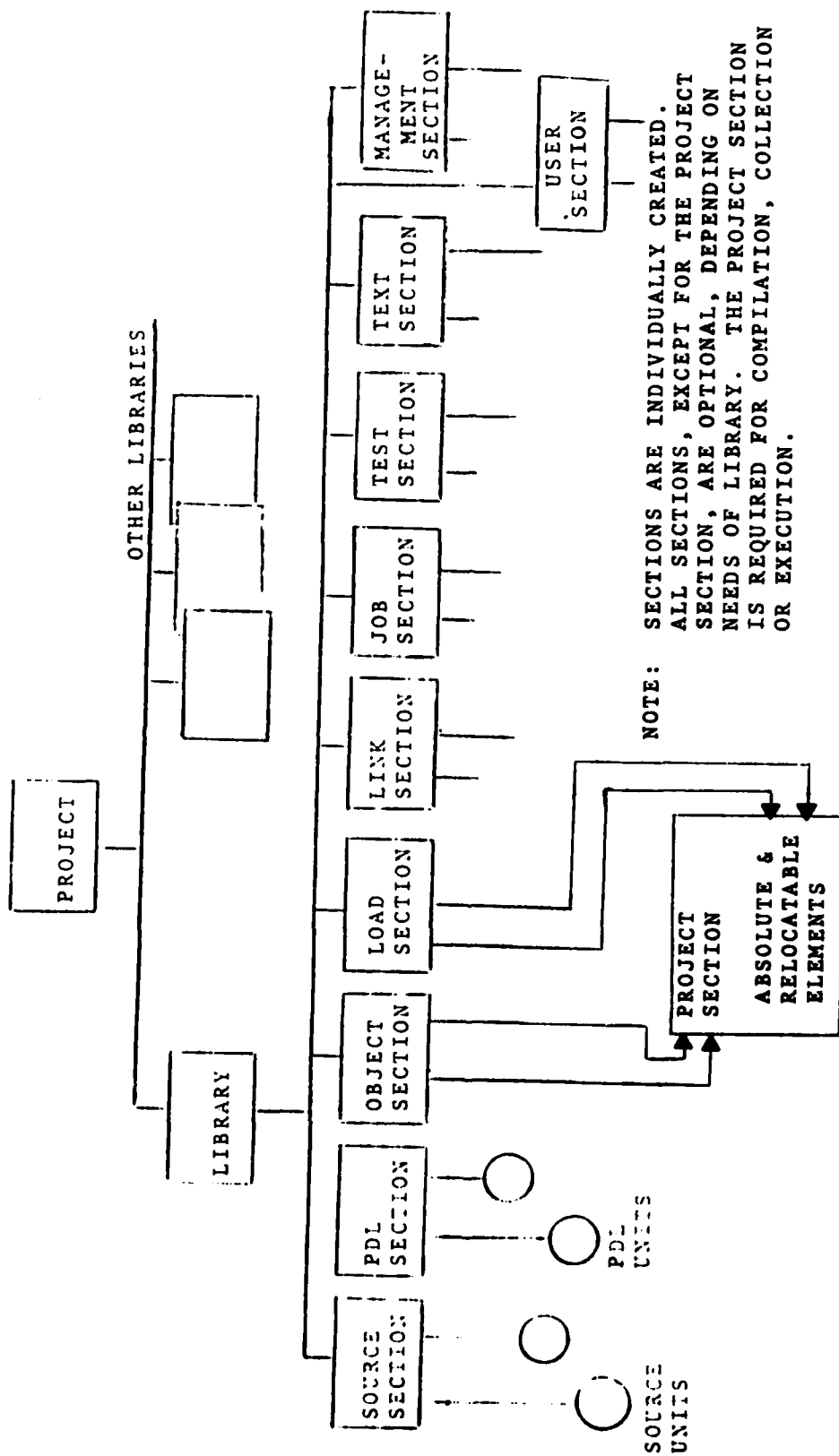


Figure 2-02. Hierarchical Structure of a Project

- g. PDL - Program Design Language statements
- h. TEXT - documentation
- i. MGMT - management data
- j. USER - PSL/NON-PSL generated data
- k. PROGRAM - relocatable and absolute elements

The PSL system manipulates the data for each section appropriately for that type of section.

A section is composed of logical segments of data called units. The unit is the lowest level of the naming convention used under the PSL system. A unit is therefore uniquely identified by the following set:

- a. Project name
- b. Library name
- c. Section name
- d. Unit name

The actual structure and content of a unit is related to the type of section to which the unit belongs. See Appendix A of the PSL User's Manual for a more detailed discussion of the structure of a section.

2.1.2 System Operation

The PSL system in the batch mode is invoked by a @ADD, @END, @ADD sequence and is directed to perform specific functions by PSL Functions cards. The general functional capabilities available under the PSL are:

- a. Initialize a project
- b. Create sections in a library
- c. Add, change, move, replace or purge a unit of code
- d. Print, punch, or write (to tape) a unit of code
- e. Print an index listing
- f. Print the top-down structure of a program
- g. Compile, link, execute
- h. Delete a section, a library, or a project
- i. Backup a project, library, section,
- j. Restore a project, library, section, or unit
- k. Collect and print management data

1. Print text material
- m. Print by author or character string.

The PSL Functions are described in detail in Section 3 of the PSL User's Manual. A general system flowchart is shown in Figure 2-02.

2.1.3 Program Organization

The PSL system organization is hierarchically configured as shown in Figure 2-03. All user jobs are initiated through the Batch Control (BCTL) processor under which four categories of program operations are grouped. The Function Processors group contains the program modules which are invoked to perform specific PSL Functions. The Support Routines group contains the modules which perform functional support operations that augment the performance of various PSL Functions. PSL Access Routines are logically grouped to exhibit the modules that are utilized to gain access to the projects, libraries, sections, and units contained in PSL storage. The fourth logical group contains those routines which are used in interfacing with the Exec 8 Operating System.

a. Function Processors

Function Processors are subdivided into the following categories:

- (1) Library Maintenance
- (2) Unit Maintenance
- (3) Program Processing
- (4) Output Processing
- (5) General Functions
- (6) Management Data Collection

Each of the above categories is hierarchically composed of PSL modules which are directly and indirectly invoked by the Batch Control (BCTL) processor to perform a specifically requested PSL Function. As will be noted, certain PSL Functions (i.e., COMPILE, LINK, EXECUTE, MDCOLLECT and MDPRIINT) "spawn" jobs which invoked the appropriate program required to complete the initial operation. All such operations (i.e., activities) are spawned under the control of the BCTL processor as an extension of the PSL run (using @ADD) for any given execution of the PSL system program.

b. Support Routines

Support Routines may generally be called upon to augment the operation of a functional processor routine; for example, as Backup Library (BKLB) under Library Maintenance calls upon Backup Section (BKSC) to provide

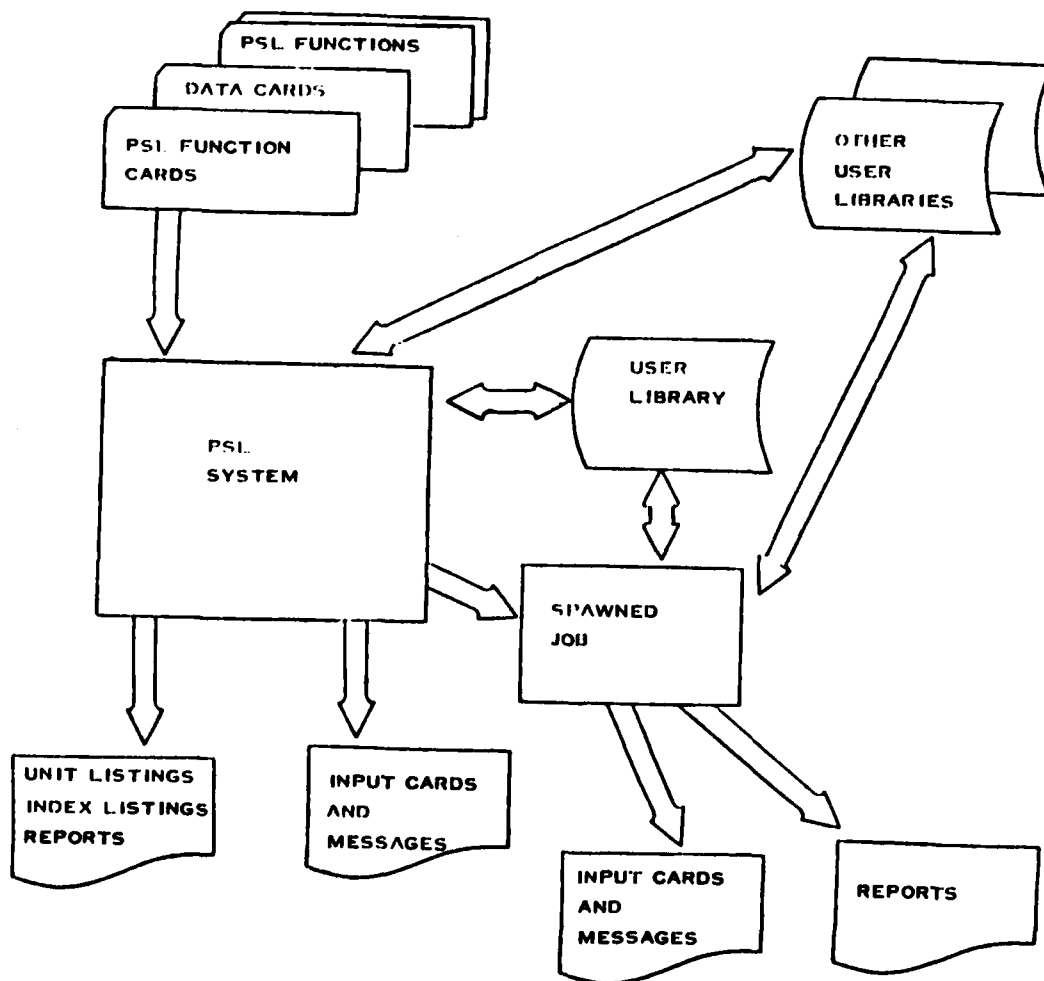


FIGURE 2-02. PSL SYSTEM FLOWCHART

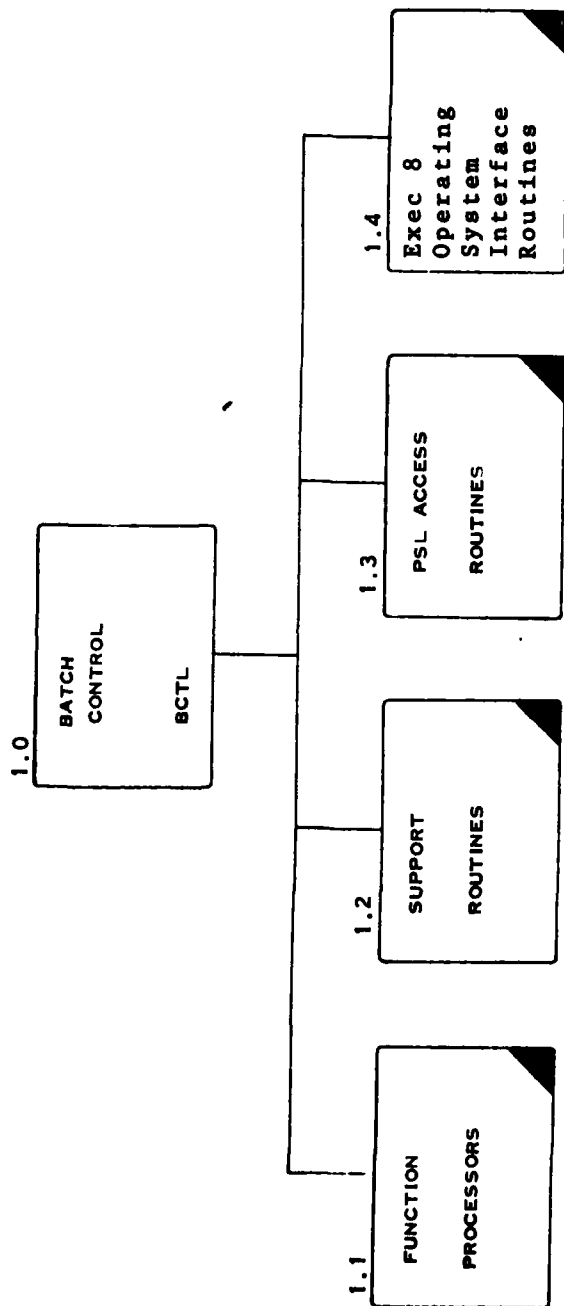


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 1 OF 11)

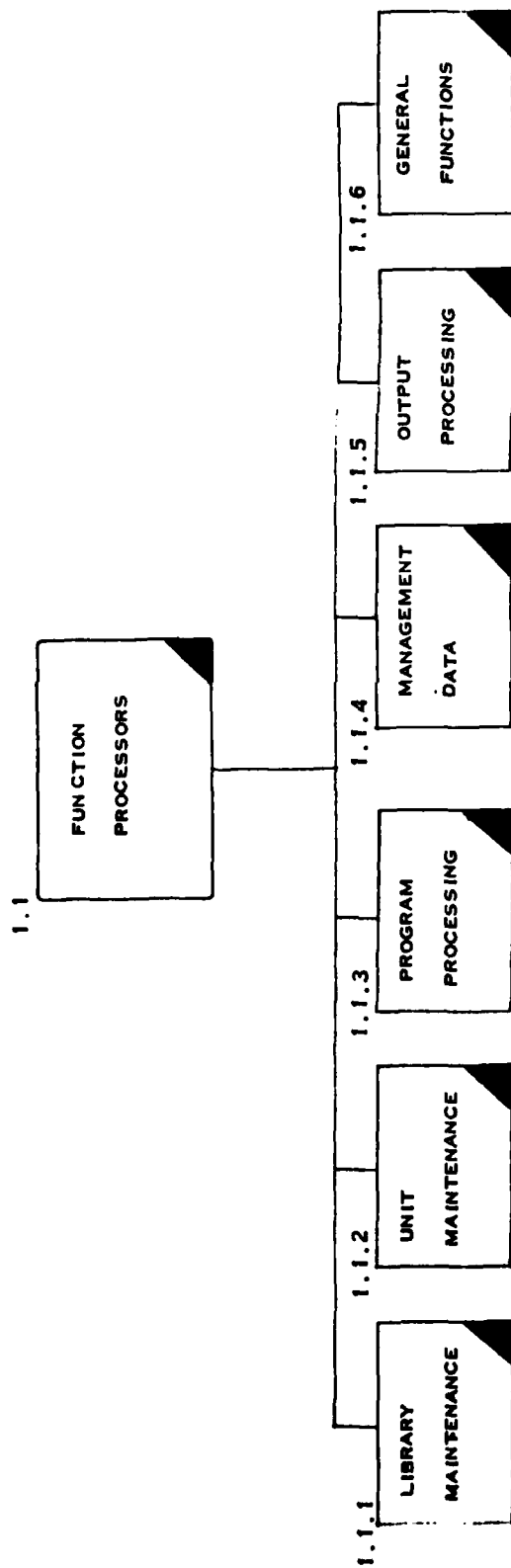


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 2 OF 11)

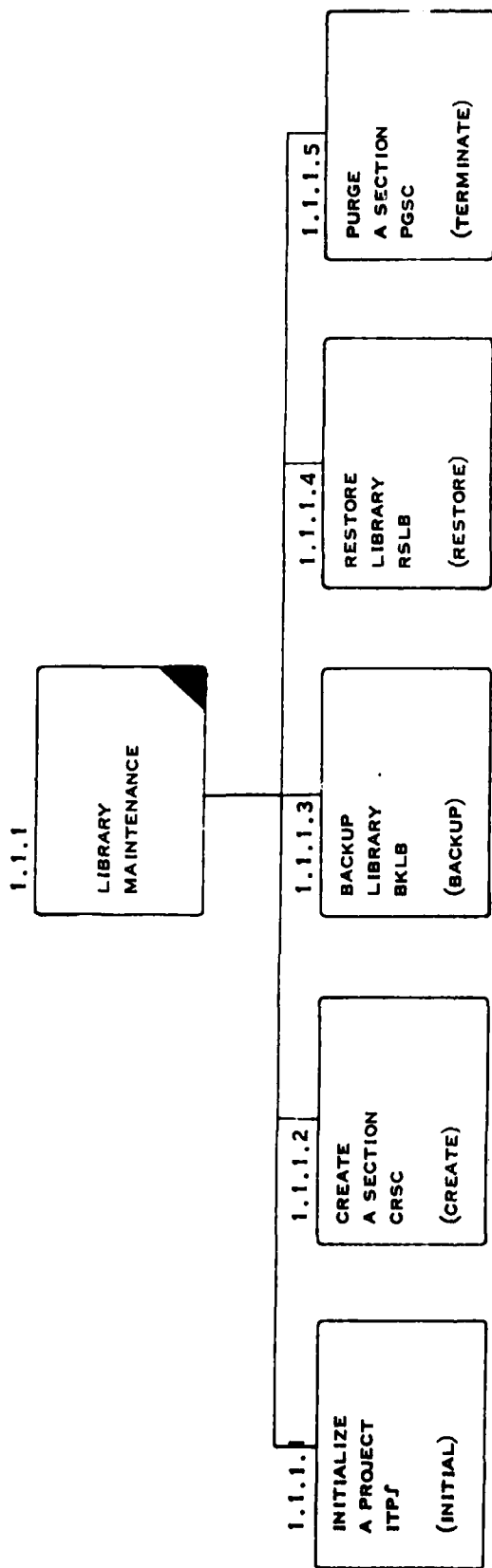


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 3 OF 11)

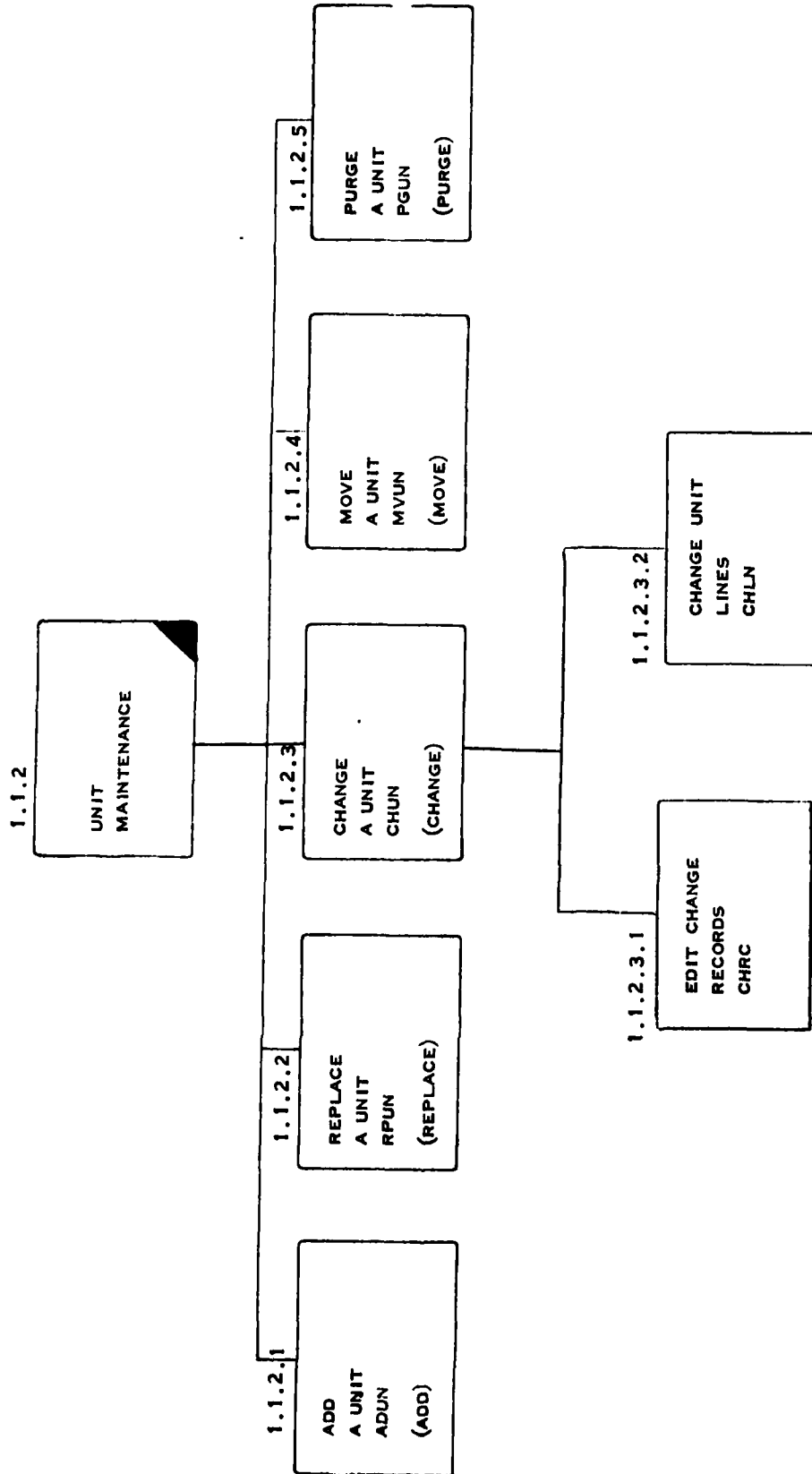


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 4 OF 11)

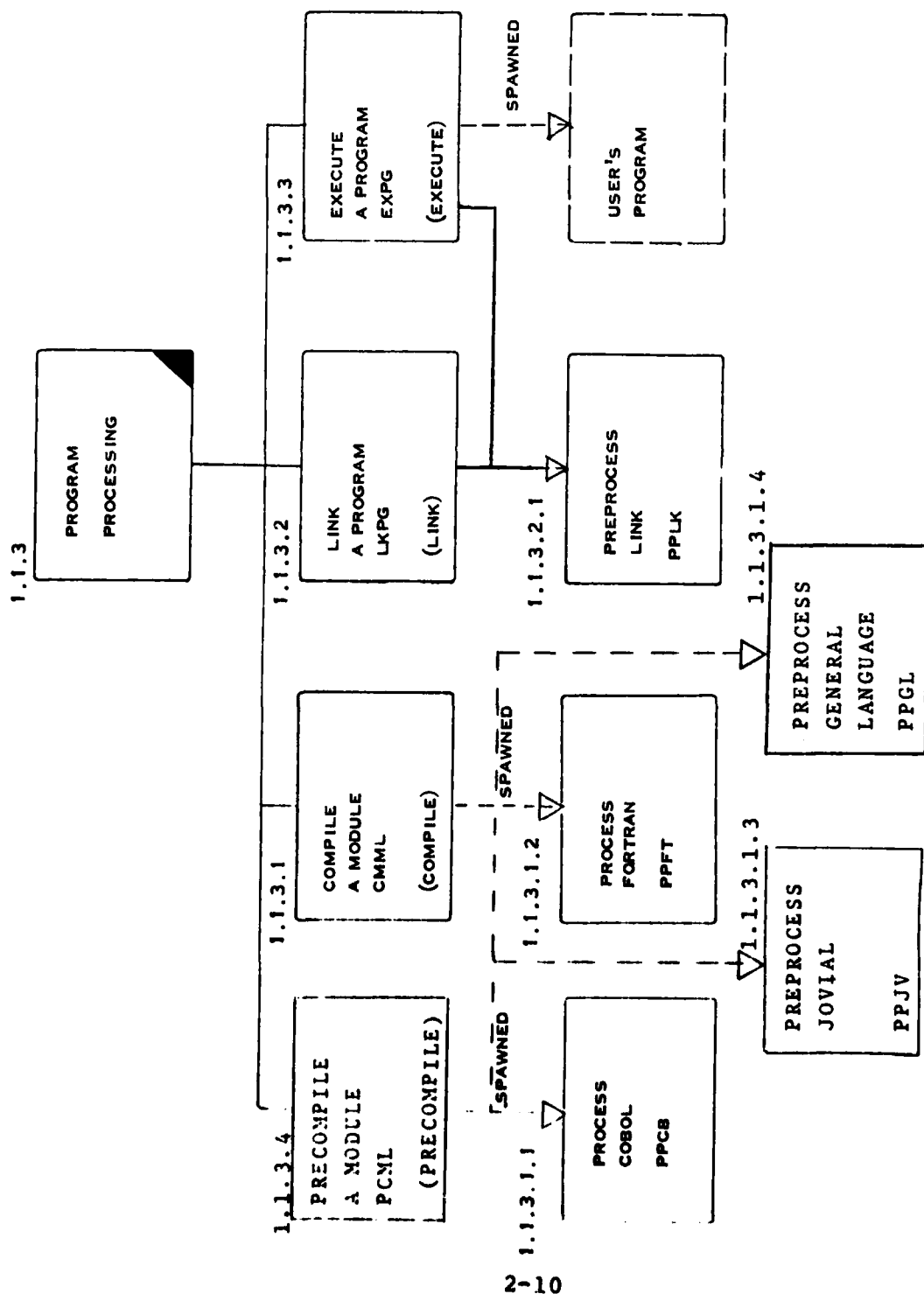


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 5 OF 11)

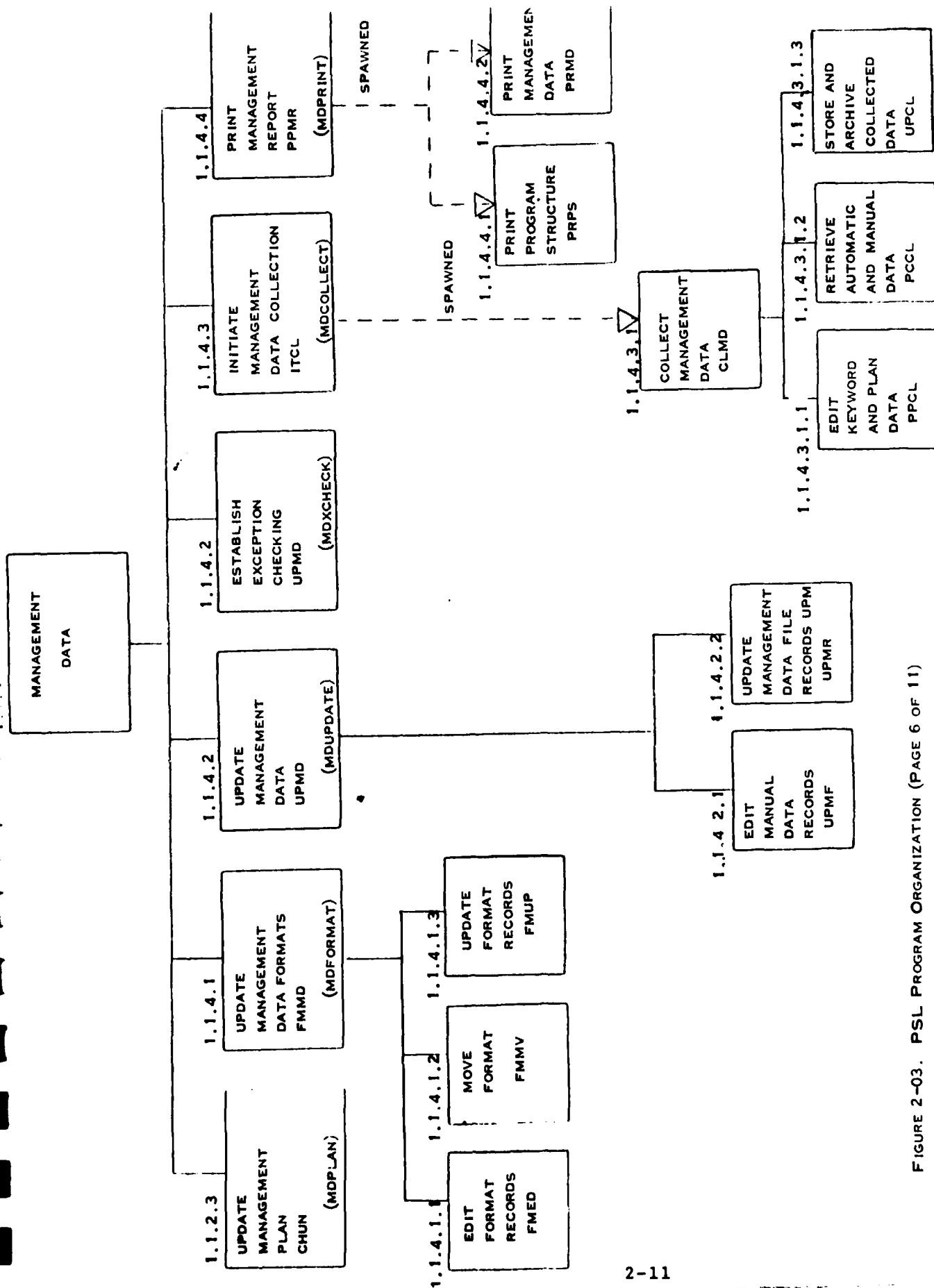


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 6 OF 11)

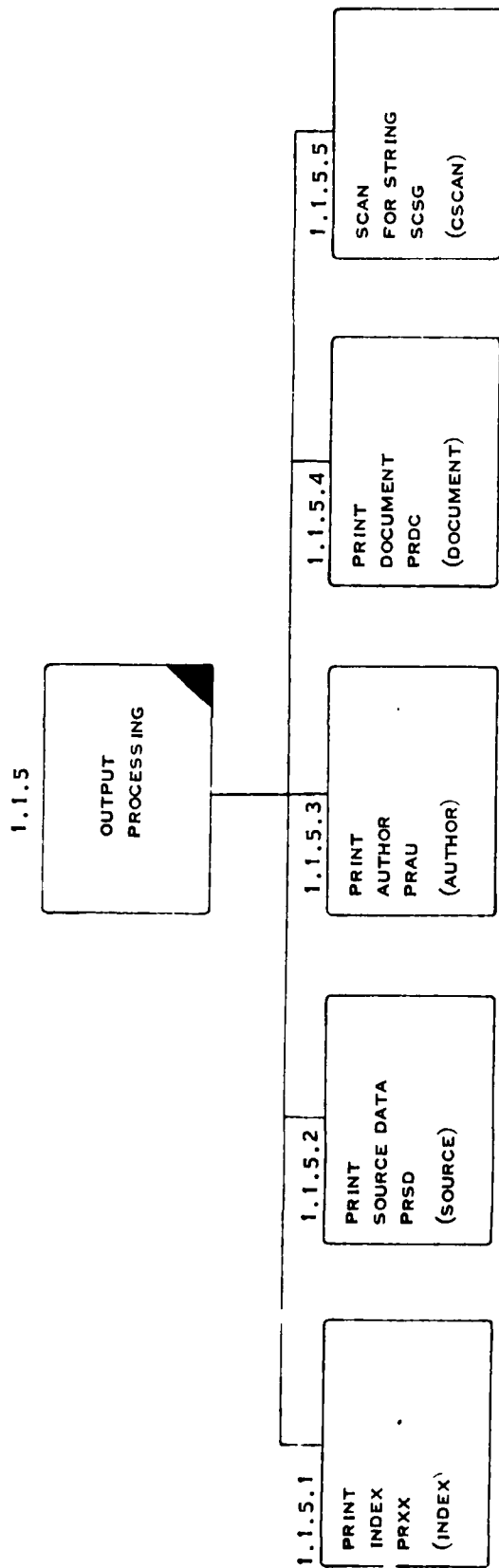


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 7 OF 11)

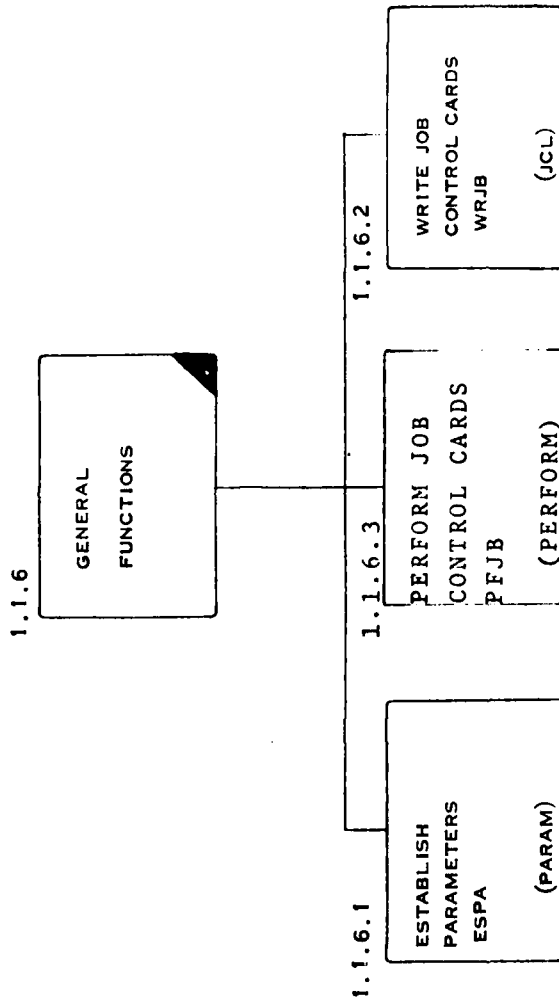


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 8 OF 11)

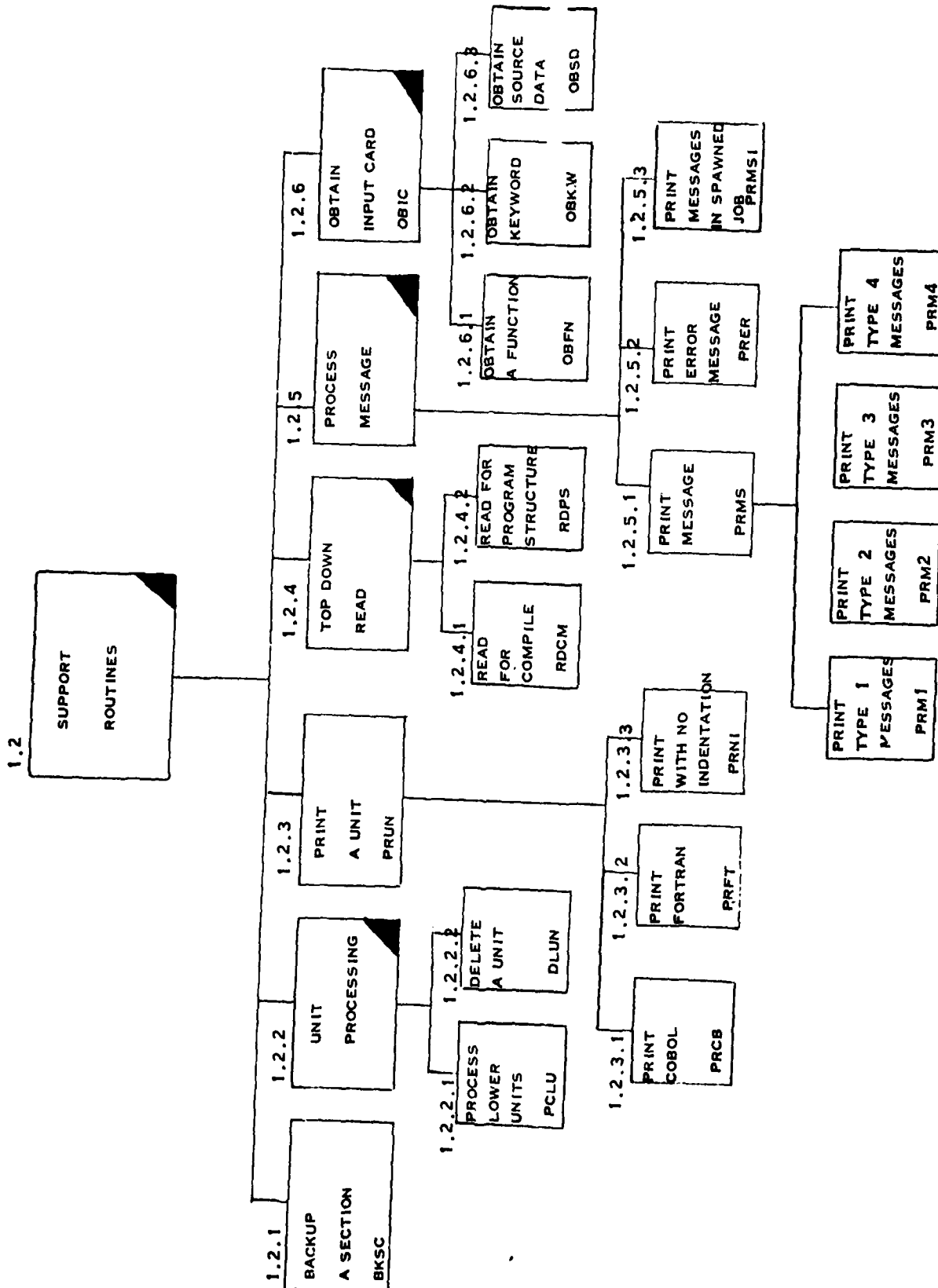
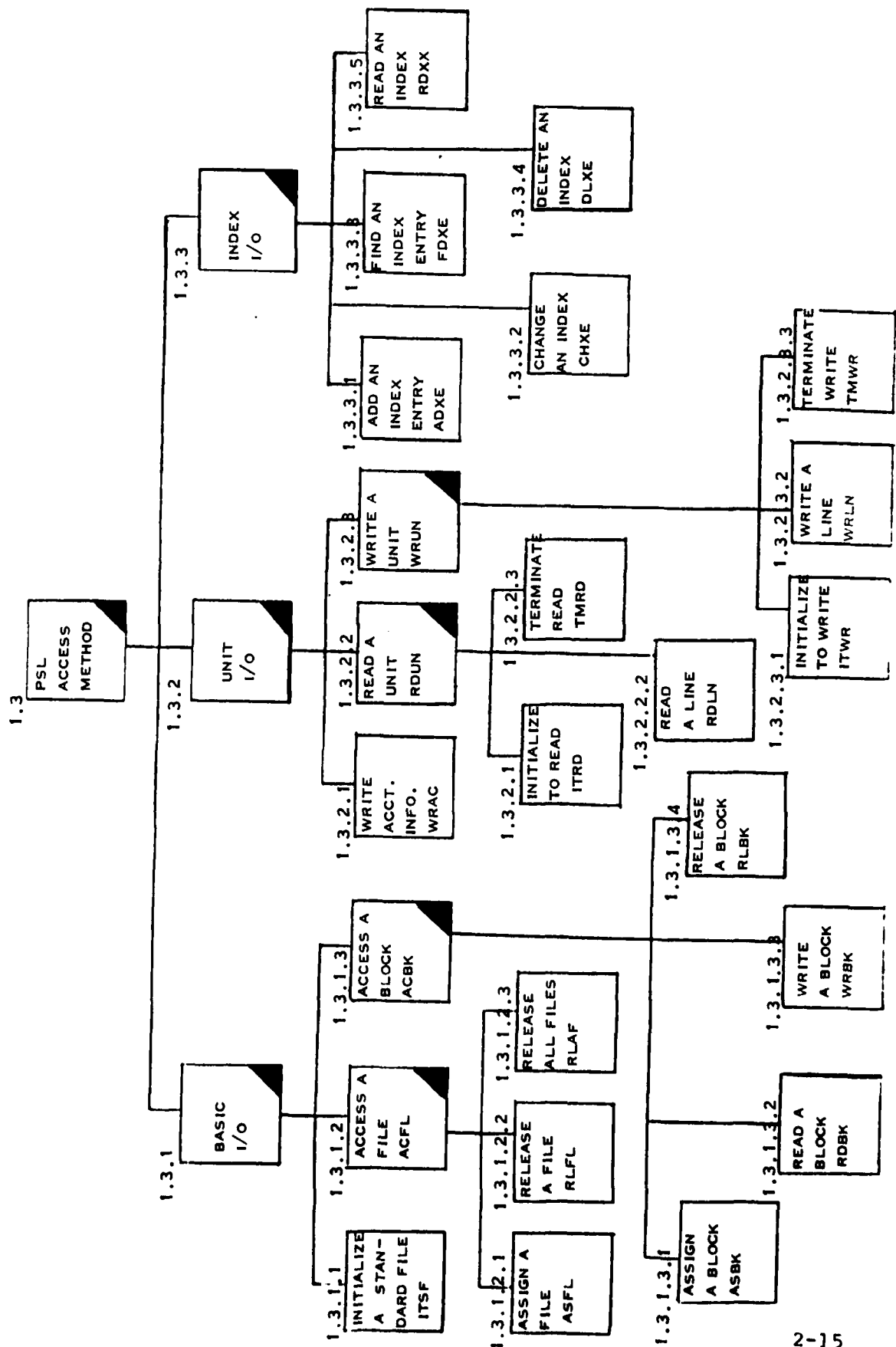


FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 9 OF 11)

FIGURE 2-03. PSL PROGRAM ORGANIZATION (PAGE 10 OF 11)



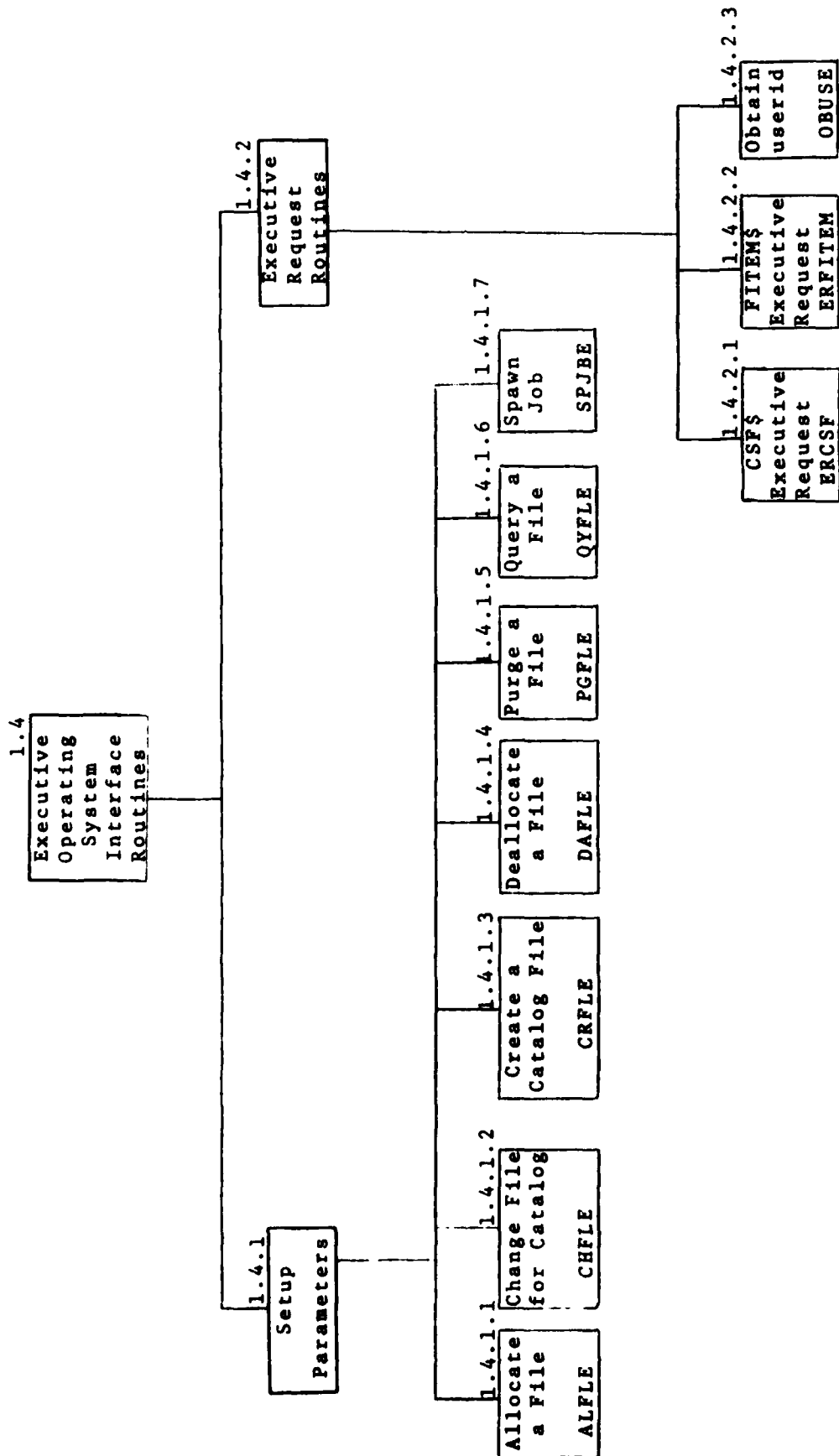


Figure 2-03. PSL Program Organization (Page 11 of 11)

backup data output. The PSL Function TERMINATE may optionally require that the purged library section be backed up prior to release of space so that the Purge Section (PGSC) module will also call upon the BKSC module to provide backup data output. This is the general nature of routines in the Support category as will be further evidenced in the detailed description given in subsection 2.2.

c. PSL Access Routines

PSL Access Routines are divided into three categories:

- (1) Basic I/O
- (2) Unit I/O
- (3) Index I/O

Basic I/O is employed to create, access and release file space maintained under the Executive system control. Once access has been gained, Unit I/O and Index I/O is utilized to read and write that file space as required.

d. Executive Operating System Interface Routines

The Executive Operating System Interface routines are divided into two categories:

- (1) Setup Parameter Routines
- (2) Executive Request Routines

Routines in the first category are written in COBOL. The second category is written in assembler (Fieldata) language.

2.2 Detailed Description

Discussion of the PSL system program modules is given according to the hierarchical identification numbers prescribed in Figure 2-03 so that a correspondence between paragraphs of this subsection and the identified program modules is established. Each program module is described under the following subject categories:

- a. Program Operations
- b. Data File and Table Descriptions
- c. Branching and Error Conditions

A HIPO diagram of each program module is provided to delineate the operations performed and the inputs and outputs processed. The subject of program operations is discussed to clarify and amplify the HIPO diagram presentation. Data files and tables are next described to provide the necessary detail for relating file and table formats to program operations. Lastly, the noteable "condition codes" which may be set during module operation are given for applicable process steps identified in the referenced HIPO diagram and if a PSL message is generated, the message type (e.g., ERR, ADV, etc.) is given. A complete list of PSL messages is given in Appendix C of the PSL User's Manual. Given the message type and message number (i.e., condition code) the reader may determine the message text which identifies the nature of a particular branching and error condition. The program action taken relative to the given condition is specified and explanatory notes are added when needed.

2.2.1 BCTL - Batch Control

The BCTL module initiates and controls all PSL processing. It obtains the function name from the user's control cards, determines which function processing module should be used and invokes the appropriate module. In addition, it opens and closes files used by more than one function processor. At the completion of all function processing, if any function processor has been invoked which requires a spawned activity to complete its processing, BCTL spawns a job using the JCL generated by the function processor.

a. Program Operations

HIPO diagram 1.0 describes operations performed. Files MESSAGE-FILE, SPAWN-JOB-FILE, UNIT-CARD-FILE, UNIT-LISTING-FILE, and UNIT-TAPE-FILE are opened for output at the start of processing and closed at the end of processing because various function processing modules append information to these files as processing proceeds (steps 3 and 5). The INPUT-CARDS file is opened for input because the function processors will read cards from this file to obtain their keyword-values and data cards as required. In step 7, each function word, as it is obtained by module OBFN from the INPUT-CARD file is compared against a table of legal values. If a match is found, the link name from the table is used to load the program which processes that function (step 8). The appropriate function processing module is called. The calling sequence for each function processor uses input argument PARAMETER-TABLE and output arguments PARAMETER-TABLE and PROCESSING-STATUS. If any of the function processors invoked has returned a code indicating that it has written JCL cards on the SPAWNED-JOB-FILE, the cards from the SPAWN-JOB-FILE are copied onto the FINAL-JOB-SPAWN-FILE (step 8). In front of the JCL written by the function processors, is inserted an Exec 8 @ADD card. The file name (FJ) of the FINAL-JOB-SPAWN-FILE is passed to the spawn job module (SPJB) and its contents are to be submitted to the operating system as a new job.

Diagram ID: 1.0

Name: BCTL - Batch Control

INPUT

Slave Service Area (SSA)
Slave Program Prefix (SPP)

INPUT-CARD-FILE

User's PSL Libraries

PROCESS

1. Obtain USERID and IDENT card information (OBUS).
2. Save USERID as USERID and as PCMR.
3. Open files for card input and message output.
4. Read input card.
5. IF input present
6. Open files for spawned-job JCL and unit input media.
7. Obtain first PSL Function (OBFN).
8. DO WHILE not end of input cards
9. Verify Function against table.
10. Call Function processor.
11. Obtain next Function (OBFN).
12. ENDDO
13. Close file for spawned-job JCL and unit output media.
14. IF spawned-job JCL was generated
15. Spawn job (SPJB).
16. ENDIF
17. Release all files (RLAF).
18. Close files for card input and message output.
19. Stop run.

OUTPUT

Parameter table

UNIT-LISTING-FILE

UNIT-CARD-FILE

UNIT-TAPE-FILE

User's PSL Libraries

Spawned job file

FINAL-SPAWN-JOB-FILE

Spawned job

Message File

In step 12, the module, Release-All-File (RLAF) is called to close and deallocate any PSL library files which may have been allocated by the function processors but not yet deallocated.

b. Data File and Table Descriptions

1. FINAL-JOB-SPAWN-FILE

This file is used to write a copy of the JCL cards generated by the function processors. The JCL on this file is added to the runstream via the executive request @ADD.

2. PSL-FUNCTION-TABLE

This table contains a list of functions currently supported by the PSL system. A function name may be up to twelve characters long, but only the first four characters are checked to identify the function. Items contained in the table are:

05 PSL-FUNCTIONS

OCCURS 28 TIMES

ASCENDING KEY

PSL-FUNCTION INDEXED

BY PF-INDEX.

10 PSL-FUNCTION

PIC X(4).

- first four characters of function name.

10 FILLER

PIC X(8).

10 FILLER

PIC X(8).

- last 8 characters of function name.

c. Branching and Error Conditions

Function Reference	Condition Code	Process Category	Program Action	Note
4	15	ERR	Normal termination	
7,9	14	INF	Normal termination	
8	1	ERR	Bypass keywords and data cards; continue with next function.	
9	28	N/A	Continue processing with next function	
8	16	ADV	Spawn a job	
11	6	ADV	Print message 52; cannot spawn a job	1
11	52	ERR	Cannot spawn a job	

Note:

- (1) This happens if the project index file from which the password for the spawned job is to be obtained cannot be assigned. See ASFL description for conditions which cause file assignment to fail.

2.2.1.1 Function Processors

The modules in this subdivision of the PSL system are uniquely called into operation in response to PSL Function requests. Function processors are further divided into the six operational categories depicted in Figure 2-03 (1.1).

2.2.1.1.1 Library Maintenance

Library maintenance functions perform the operations to initialize a project (INITIAL), create a library section (CREATE), backup library sections (BACKUP), restore library sections (RESTORE) and purge a library section or project (TERMINATE). The PSL system modules that correspond with each function are described below.

2.2.1.1.1.1 ITPJ - Initialize a Project

The module ITPJ, which corresponds to the PSL function ** INITIAL, establishes a project as a PSL project. This involves the creation and initialization of a random file to serve as the index for library-sections to be created under the project and storing the password associated with the project. There is also an option to change the password.

a. Program Operations

HIPO diagram 1.1.1.1 describes operations performed. If the user provides parameters to be used for file creation, they are stored (step 1) on the FMS-PARAMETER-FILE. The module Create File (CRFL) is then called (step 2) to create the project index file via the Executive Request @CAT.

The module Initialize a Standard File (ITSF) is then called (step 3) to initialize the newly created file.

b. Data File and Table Descriptions

No unique files or significant tables are used.

c. Branching and Error Conditions

Function Reference	Condition Code	Process Category	Program Action	Note
1	2	ERR	Subsequent processing bypassed	
1	19	ERR	Subsequent processing bypassed	

Diagram ID: 1.1.1.1

INPUT

[Parameter-table]

[Input-card-file]

Name: ITPJ - Initialize a Project

PROCESS

1. Verify keywords and store values.
2. Create new random project-index file which will also contain project directory (CRFL).
3. Initialize file as standard PSL file (ITSP).

OUTPUT

[Parameter-table]
[Processing-status]

[PMS-PARAMETER-FILE]

Project index file
PSL-CONTROL-BLOCKS
PSL-INDEX-BLOCK
PSL-DIRECTORY-BLOCK

[Message file]

c. Branching and Error Conditions (Cont'd).

1	32	ERR	Subsequent processing bypassed	
2	200	FMS	Index file not created; subsequent processing bypassed	
3	23	ERR	Subsequent processing bypassed	1
4,6	6	ADV	Subsequent processing bypassed	1,2
5,8	24	PSL	Subsequent processing bypassed	1
8	51	PSL	Subsequent processing bypassed	
8	31	PSL	Subsequent processing bypassed	

tes:

- (1) If the project index file has just been created, it will be purged before returning to the calling program.
- (2) The project index file could not be assigned. See description of ASFL for conditions which cause file assignment to fail.

2.2.1.1.1.2 CRSC - Create a Section

The CRSC module, which corresponds to the PSL ** CREATE function, builds a section in a user's library after the project has been initialized. A random file is created and initialized to contain the section index, control information, selected section options, and unit data. For a MGMT section, a management plan unit is entered in the newly created section. The name of the section is entered in the Project index.

a. Program Operations

HIPO diagram 1.1.1.2 describes operations performed. Specific section options are appropriate only for specific sections. If the user specifies an option for a section for which it is not appropriate, it is ignored. If a required option is not specified, a default value is used. Table 2-a shows the options, the sections for which they are appropriate, and the default values. The same sequence of module calls (CRFL, ITSF) described in section 2.2.1.1.1.1 for the ITPJ module is used to create the random section file. The section options are passed to ITSF to be stored in the PSL-CONTROL-BLOCK of the random section file.

b. Data File and Table Descriptions

1. SECTION-OPTION-TABLE

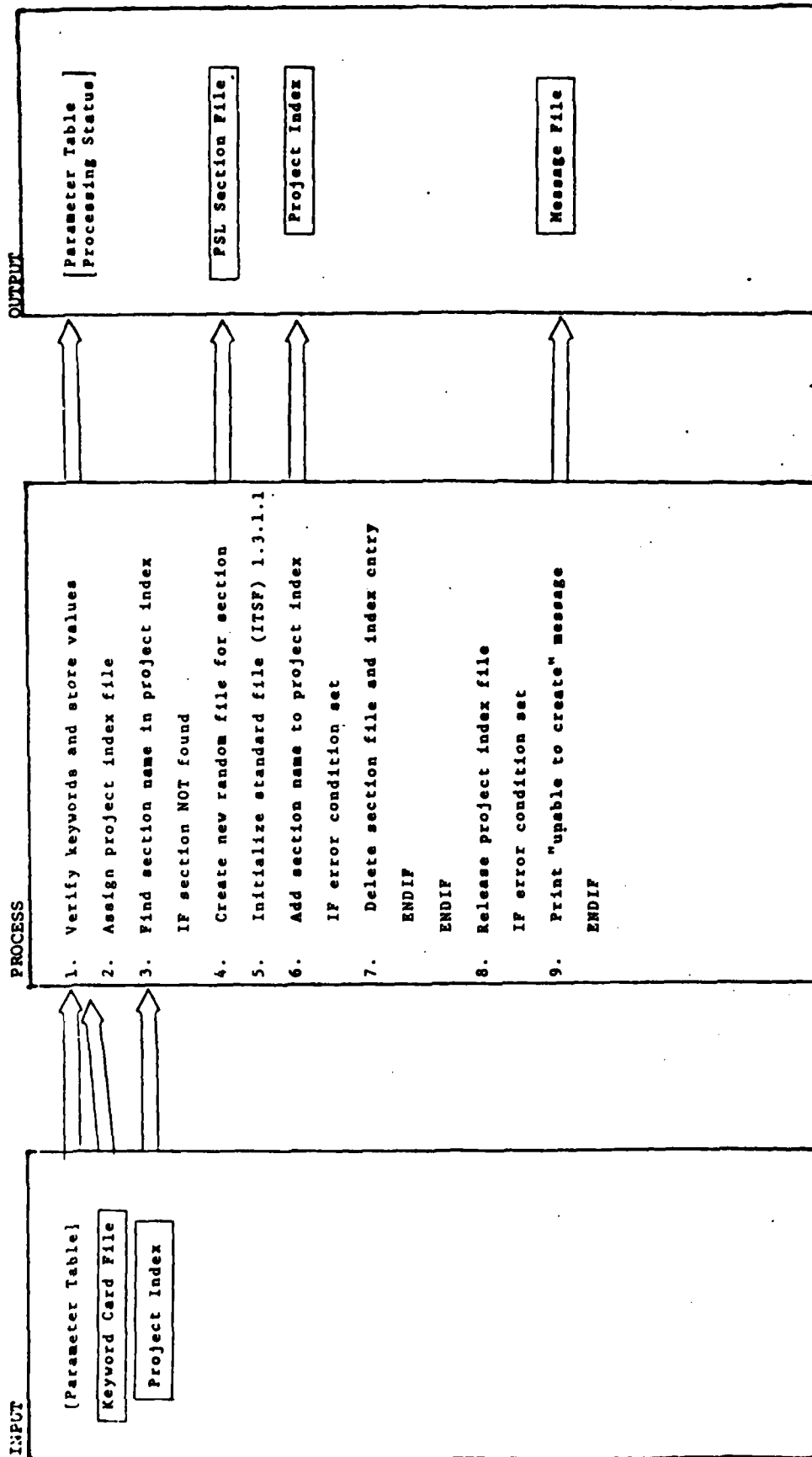
The SECTION-OPTION-TABLE is a two dimensional table whose rows represent the nine possible PSL sections and whose columns represent the six possible section options. For each section/option pair the table entry contains a one if the option is appropriate for that section or the table entry contains a zero if the option is not appropriate for the section. Table 2-a shows section/option correspondence represented by this table. The items contained in the table are:

05	SECTION-OPTION-TABLE-ENTRY	OCCURS 9 TIMES.
10	SECTION-OPTION	OCCURS 6 TIMES PIC X. - contains a 0 or 1 as described above
88	OPTION-PERMITTED	VALUE 1.

Diagram ID: 1.1.1.2

Name: CRSC - Create a Section

Description: Function Processor (CREATE)



Section Options

<u>Section</u>	<u>COMPRESS</u>	<u>FMS (1)</u>	<u>MGMTDATA</u>	<u>STANDARD</u>	<u>SPCHECK</u>	<u>SLENGTH</u>
JOB	ok	ok	ok	ok		
LINK		ok	ok	ok	ok	
LOAD		ok		ok		
MGMT	ok	ok				
OBJECT		ok	ok	ok		
PDL	ok	ok	ok	ok	ok	ok
SOURCE	ok	ok	ok	ok	ok	ok
TEST		ok	ok	ok		
TEXT		ok	ok	ok		
Default values	YES	NO	NO	(2)	NO	(3)

Notes:

- (1) FMS option indicates whether or not the user has provided FMS parameters to override the default values.
- (2) The default value for STANDARD option is YES except for the MGMT section. For the MGMT section, the default value is NO.
- (3) SLENGTH option is not used unless the user specifies SPCHECK=YES. Then the default value is 50 lines.

Table 2-a. PSL Sections, Options and Default Values

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Subsequent processing is bypassed	
1	32	ERR	Subsequent processing is bypassed	
1	19	ERR	Subsequent processing is bypassed	
2,7	7	ADV	Subsequent processing is bypassed	1,2
3	36	ERR	Subsequent processing is bypassed	
4,6	200	FMS	Random section file not created; subsequent processing bypassed	1
5	23	ERR	Subsequent processing is bypassed	1
7	46	ERR	Subsequent processing is bypassed	1
7,8	24	ERR	Subsequent processing is bypassed	1

Notes:

- (1) If the random section has already been created, it will be purged and the index entry deleted from the project index file.
- (2) The project index file or the new random section file could not be assigned. See ASFL description for conditions which cause file assignment to fail.

2.2.1.1.1.3 BKLB - Backup Library

Library sections of a PSL project are saved on a backup file. This backup file may be used as input to the RESTORE Function in succeeding PSL jobs.

a. Program Operations

HIPO diagram 1.1.1.3 depicts the top-level operations of the BKLB module. The input keywords are first validated. If more than one section is designated for backup (i.e., LIBRARY=ALL or SECTION=ALL), the project index file is read to determine the library section names corresponding with the user's backup request. These names are saved on a temporary file and immediately read back to order the assignment of the PSL sections which are to be written to the Backup Tape file. The Backup Section (BKSC) module is called to perform the detailed backup procedure (refer to paragraph 2.2.1.2.1).

b. Data File and Table Descriptions

No special files or tables are required.

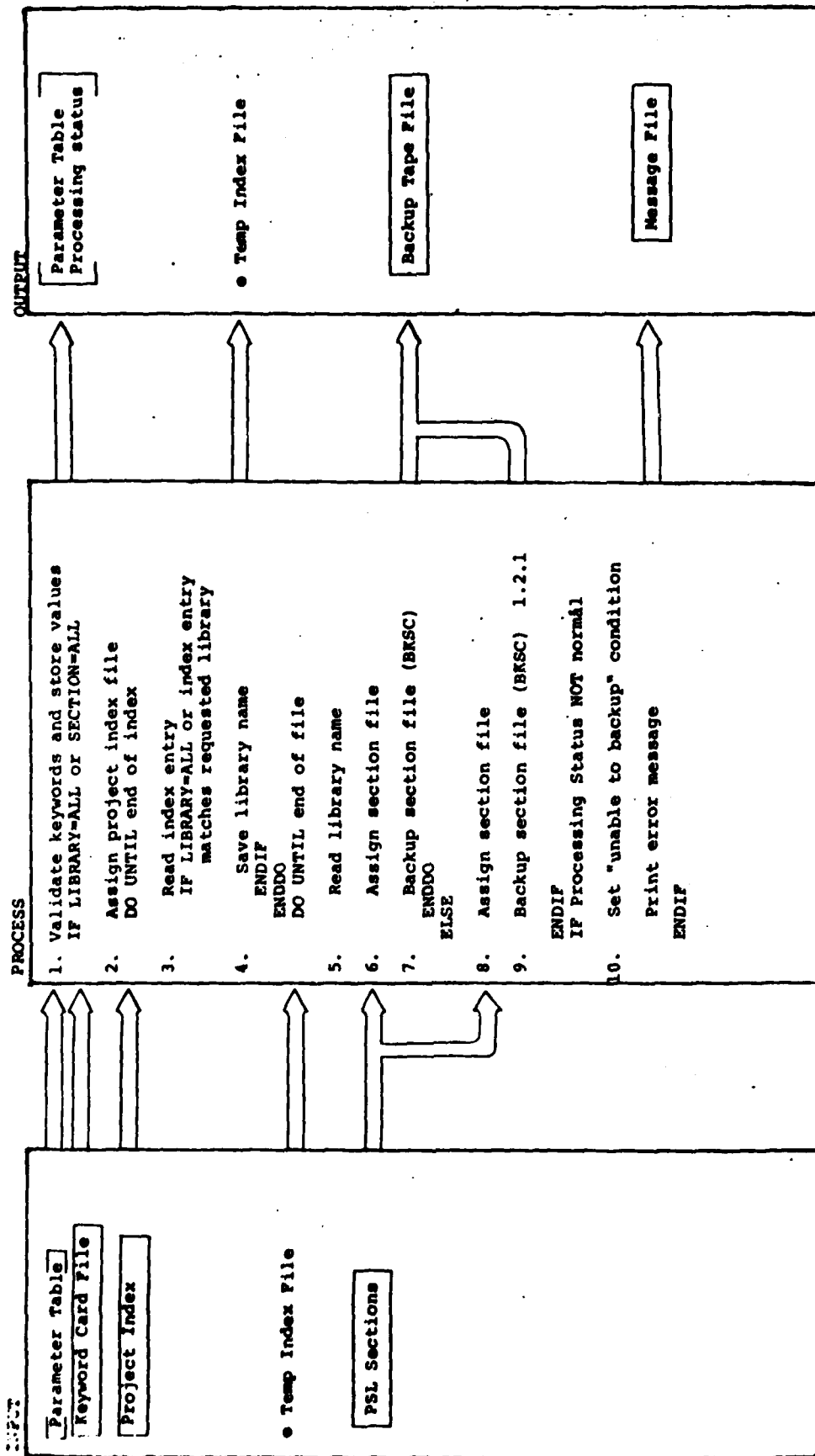
c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Perform Process #10	
	32	ERR	Perform Process #10	
	98	ERR	Perform Process #10	
2	6	ADV	Perform Process #10	
3	69		Bypass Process #4	1
10	7	ERR	Error Exit	

Note:

(1) End-of-index condition also terminates DO loop.

Diagram ID: 1.1.1.3 Name: BKLB - Backup Library Description: Function Processor



2.2.1.1.1.4 RSLB - Restore Library

A PSL Backup File is read and designated user-project library sections and units are restored.

a. Program Operations

HIPO diagram 1.1.1.4 depicts the top-level operations of the RSLB module. Each record input from the Backup File contains a "Record Sequence Nbr" and a "Record Type Nbr". The section header record has a sequence number of one and each succeeding record for that section (through the end-of-section record) is checked for an incremental increase of one in its sequence number value. The record sequence number is verified to detect omitted records in the event that tape input errors are processed to skip unreadable records. A case figure is used in response to the record type number which will be seen to functionally correspond with record types generated by the Backup Section (BKSC) module described under paragraph 2.2.1.2.1. Once a section header record has been verified under the CASE 1 entry as to the requirement to restore it and the suitability of the section contents (i.e., the section must be newly created, having no units if the entire section is to be restored), then account is kept of the record sequence until the end-of-section record is received or another section header record is received before that event occurs. An out-of-sequence record will force an end-of-unit condition if a unit is in the process of being restored and normal processing will be resumed when the next unit header record for that section is received. An independent file is dynamically allocated and then created using Exec 8 Operating System Interface routines. If the FMS parameters record (i.e., record type 4) on the backup file contains zeroes, FMS default parameters are utilized except as may be overridden by FMS parameters specified through the RESTORE Function FMS keyword value input. Independent files are written in response to record type 5 inputs provided that the prior step to allocate and create the file is completed normally. A record type 6 input causes the independent file to be closed and deallocated. The handling of standard PSL units is somewhat simpler since record type 3 inputs are read directly after the unit is initialized into PSL block storage. These inputs are written to PSL block storage and writing is terminated when record type 6 is received. Receipt of record type 7 signals the end of a section and causes the assigned section file to be released and the section verification switch setting to change from a "section-verified" value to a "section-completed" value. When a specific unit is designated to be restored, then only that unit will be restored

Diagram ID: 1.1.1.4

Name: RSLB - Restore Library

Description: Function Processor (RESTORE)

I:PUT

Parameter Table

Keyword Card File

Backup File

PROCESS

1. Validate keywords and store values
DO UNTIL end of backup file
2. Read backup record
3. Verify record sequence nbr
CASENTY Record Type Nbr
CASE 1.
4. Windup incomplete section restore
5. Verify section restore requirement
6. Verify section content
CASE 2.
7. IF section verified
Initiate PSL unit
ENDIF
CASE 3.
8. Restore unit data line
CASE 4.
9. Create independent file
CASE 5.
10. Restore independent file record
CASE 6.
11. Windup unit restore
CASE 7.
12. Windup section restore
ENDCASE
13. Resolve format error
ENDDO
14. Conclude restore procedure

OUTPUT

Parameter Table
Processing Status

Section Index

Unit Data Block
Accounting Record
Data Lines

FMS Catalog/File

File Records

Message File

in a given RSLB operation. If that unit already exists in the designated library section, the matching backup file unit must be a corresponding unit type in order for it to replace that existing unit. Replacement is performed by first deleting the existing unit and then adding the backup file unit.

b. Data File and Table Descriptions

1. BACKUP FILE

The backup file consists of various record formats. The following File Description (FD) is utilized for this file:

```

FD  BACKUP-FILE
    LABEL RECORDS ARE STANDARD.
01  BACKUP-RECORD.
    05  RECORD-CONTROL-NUMBERS.
        10  RECORD-SEQUENCE-NBR          PIC S9(9) COMP
        10  RECORD-TYPE-NBR              PIC S9(9) COMP
    05  BACKUP-RECORD-DATA                PIC X(384).

01  SECTION-HEADER-RECORD.
    05  FILLER                            PIC S9(9) COMP
    05  FILLER                            PIC S9(9) COMP.
    05  SECTION-HEADER-DATA.
        10  BACKUP-SECTION-REFERENCE.
            15  BACKUP-PROJECT-NAME      PIC X(12).
            15  BACKUP-LIB-SEC-NAME.
            20  BACKUP-SECTION-CODE
                PIC X(1).
            20  BACKUP-LIBRARY-NAME
                PIC X(7).
                15  FILLER                PIC X(4).
        10  BACKUP-SECTION-DATA          PIC X(28).
    05  FILLER                            PIC X(8).

01  UNIT-HEADER-RECORD.
    05  FILLER                            PIC S9(9) COMP.
    05  FILLER                            PIC S9(9) COMP.
    05  UNIT-HEADER-DATA.
        10  BACKUP-UNIT-NAME             PIC X(30).
        10  ACCOUNTING-INFO              PIC X(194).
    05  FILLER                            PIC X(4).

01  STANDARD-UNIT-RECORD.
    05  FILLER                            PIC S9(9) COMP.
    05  FILLER                            PIC S9(9) COMP.
    05  STANDARD-UNIT-DATA.
        10  UNIT-DATA-LINE               PIC X(80).
    05  FILLER                            PIC X(4).
  
```

```

01 BACK-UP-FMS-PARAMETER-RECORD.
05 FILLER PIC S9(9) COMP.
05 FILLER PIC S9(9) COMP.
05 FMS-PARAMETERS-DATA.
10 FMS-PERMISSIONS-DATA PIC X(6).
10 FMS-OPTIONS-DATA PIC X(192).
05 FILLER PIC X(6).

01 LOAD-UNIT-RECORD.
05 FILLER PIC S9(9) COMP.
05 FILLER PIC S9(9) COMP.
05 LOAD-UNIT-DATA PIC X(384).

01 OBJECT-UNIT-RECORD.
05 FILLER PIC S9(9) COMP.
05 FILLER PIC S9(9) COMP.
05 OBJECT-UNIT-DATA.
10 FIRST-DATA-CHARACTER PIC X(1).
10 FILLER PIC X(161).
05 FILLER PIC X(6).

```

End-of-unit (record type 6) and end-of-section (record type 7) records do not contain any backup record data content.

c. Branching and Error Conditions

The following branching and error conditions are noteable:

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Go to Error Exit	
	19	ERR	Go to Error Exit	
	32	ERR	Go to Error Exit	
	98	ERR	Go to Error Exit	
	135	ERR	Go to Error Exit	
3	144	ERR	Go to Error Exit	
	139	ERR	Recover from error	1
6	136	ERR	Go to Error Exit	
	138	ERR	Recover from error	2
7	4	ERR	Recover from error	3

c. Branching and Error Conditions (continued)

13	144	ERR	Go to Error Exit	
	145	ERR	Recover from error	4
	146	ERR	Go to Error Exit	
14	134	ERR	Go to Error Exit	
	7	ERR	Error Exit	

Notes:

- (1) Windup unit restore, as required, and resume processing at next unit header record (type 2).
- (2) Bypass section input records (i.e., section not verified).
- (3) Existing unit not replaced with backup file unit; backup file search continues.
- (4) Backup file search continues (as long as valid record types are input) for a section header record (type 1)).

2.2.1.1.1.5 PGSC - Terminate a Library (Purge Sections)

The PGSC module, which processes the PSL ** TERMINATE function, deletes a section, a library, or an entire project. On option, a backup copy of the section(s) terminated is written to tape.

a. Program Operations

HIPO diagrams 1.1.1.5 and 1.1.1.5.1 describe operations performed. In diagram 1.1.1.5 step 8, module Release All Files (RLAF) is called even though each PSL file which was assigned has already been released with a call to Release a File (RLFL). This is done here, but not in other PSL function processor modules, because RLFL simply marks a released file as being "not in use" in its own file table. Then, if a subsequent function processor requires the use of the same file, it needs only to re-mark the file "in use" and the steps close, deallocate, allocate, and open are saved. However, in the case of terminate, since the section files have been purged, these files should not be available to subsequent function processors.

b. Data File and Table Descriptions

1. TEMP-INDEX-FILE

The TEMP-INDEX-FILE is used to store the names of those sections which are to be terminated. The items contained in each record are:

• 01 TEMP-INDEX FILE PIC X(8).

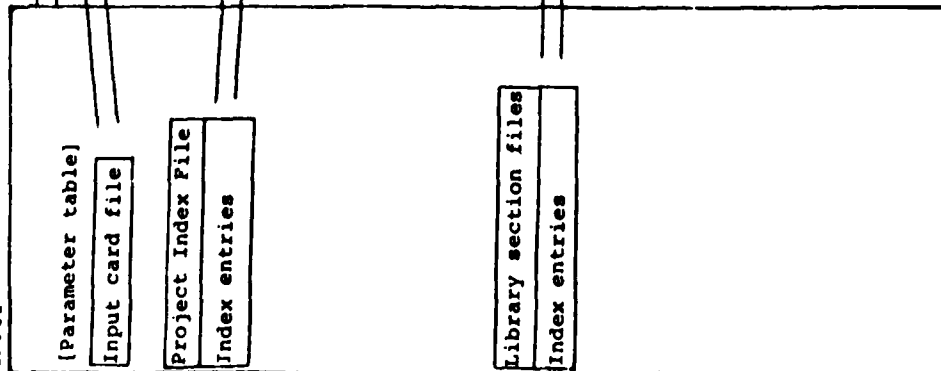
- name of library section to be terminated.

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	98	ERR	Subsequent processing is bypassed	
1	2	ERR	Subsequent processing is bypassed	

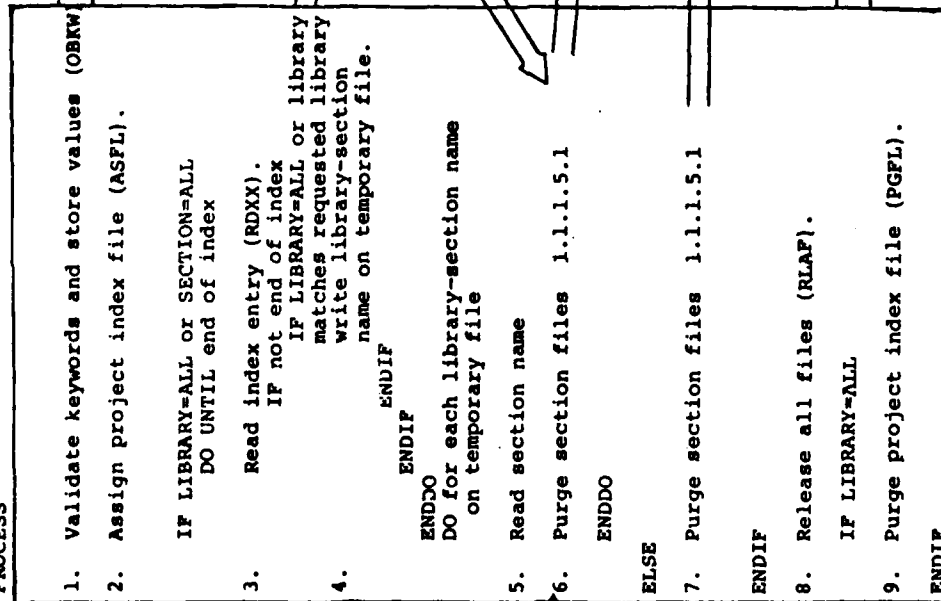
Diagram ID: 1.1.1.5

INPUT

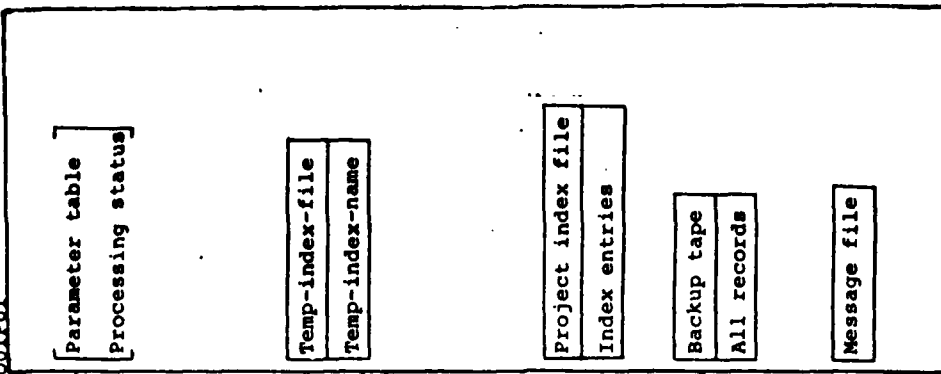


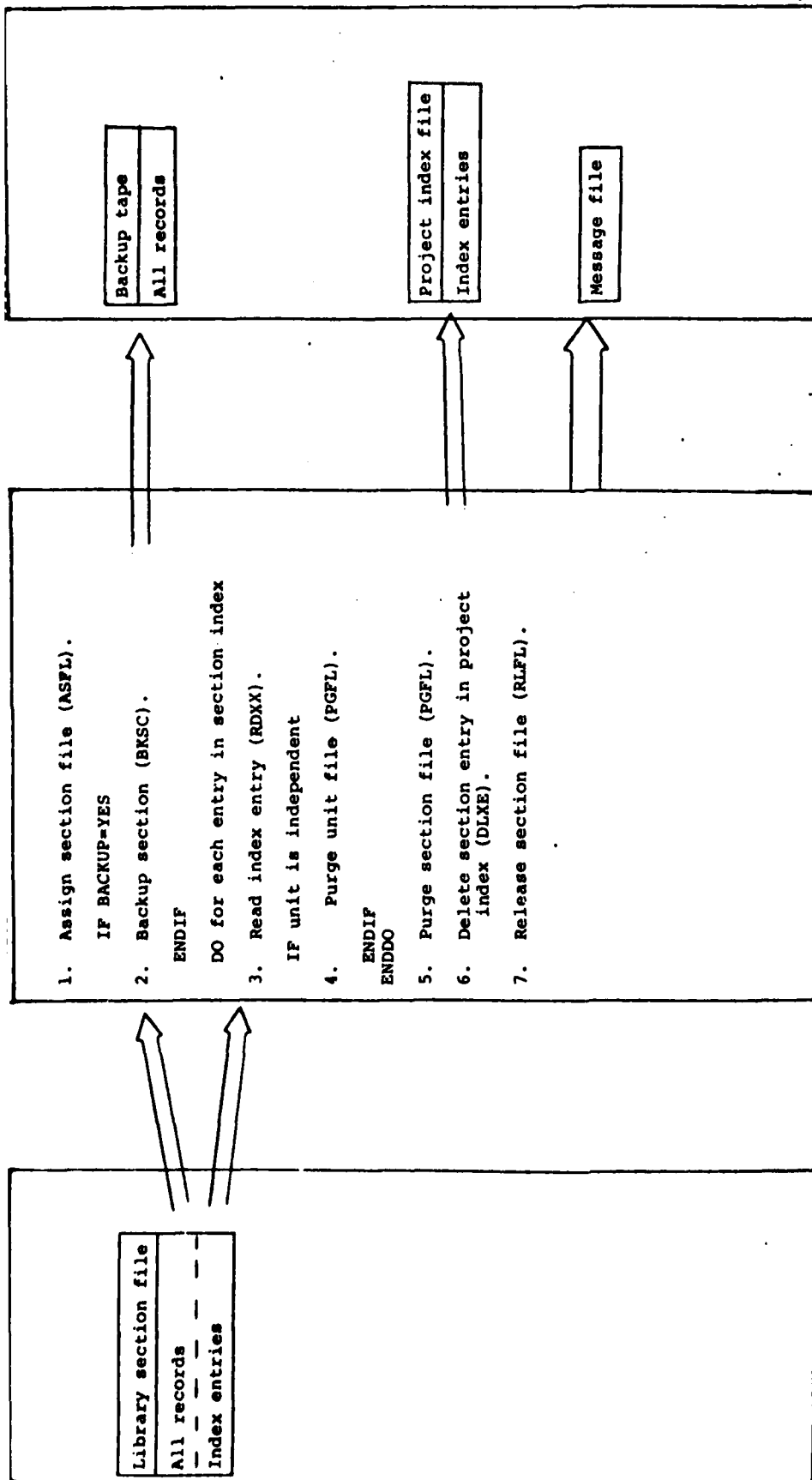
Name: PGSC - Terminate Library

PROCESS



OUTPUT





c. Branching and Error Conditions (Cont'd.)

Function Reference	Condition Code	Message Category	Program Action	Note
1	19	ERR	Subsequent processing is bypassed	
1	32	ERR	Subsequent processing is bypassed	
7,1.1	7 from BKSC	ADV	Subsequent processing is bypassed	
1.2	131	ERR	Subsequent processing is bypassed	
1.4	205	FMS	Normal processing continues	
9,1.4	other 200's	FMS	Subsequent processing is bypassed	1

Note:

- (1) An FMS error which occurs when trying to purge the project index may not be significant since all designated sections have already been terminated.

2.2.1.1.2 Unit Maintenance

Unit maintenance functions are performed to add a unit (ADD), replace a unit (REPLACE), change a unit (CHANGE), move a unit (MOVE) and purge a unit (PURGE). The program modules whose operations are especially invoked by these functions are described below.

2.2.1.1.2.1 ADUN - Add a Unit

The module ADUN, which corresponds to the PSL ** ADD function, enters a unit of data into the SOURCE, LINK, JOB, TEST, TEXT or PDL section of a PSL library. The unit accounting information is initialized. For unit in the SOURCE and PDL sections the lines of data (source code) are scanned for "INCLUDE" statements and pointers are created in the "included" unit to the including unit. If the "included" unit does not exist in the section, a stub unit is created to hold the required pointers.

a. Program Operations

HIPO diagram 1.1.2.1 describes the operations performed. In step 3, if a stub unit already exists in the section, the only user specified unit type which will be accepted is INCLUDED. This is because the presence of the stub unit indicates that there exists in the section an "INCLUDE" statement for that unit in the section which implies a unit type of INCLUDED. Any other definition would introduce a contradiction in the section. If the user does not specify unit type, the default value is determined as shown in Table 2-b. In step 4, the unit name is checked to insure that it conforms to the rules listed in the PSL Users Manual Figure 3-11 Name Lengths and Restrictions. Also in step 4, if the user has not provided a language for the new unit, a check is made to determine whether the language is required. The language keyword is required only in SOURCE section for a top unit (MAIN, CALLED, INDEPENDENT) and for any included unit for which a stub does not already exist. In step 8, ADUN scans each statement for an "INCLUDE". If the "INCLUDE" is found, module PCLU is called to update the including unit pointers and create a stub if required.

b. Data File and Table Descriptions

1. SUPPORTED-LANGUAGE-TABLE

The SUPPORTED-LANGUAGE-TABLE is a list of language names for which PSL provides structured programming support. For those language which appear in this table, the default unit type is MAIN. The data items in this table are:

05 SUPPORTED-LANGUAGE

OCCURS 7 TIMES
INDEXED BY SP-INDEX
PIC X(8).

- name of language supported. The entries are SPFORT, SCOBOL, SJOVIAL, and the General Preprocessor indicators ASMG, COBOLG, FORTRANG, JOVIALG.

Diagram ID: 1.1.2.1

INPUT

[Parameter table]

Input card file

Key words and values

Source data cards

PSL library section file

Index entries

Unit accounting info

As updated in steps 5-9

PSL library section file

Unit accounting info

Unit source data

Unit higher unit block

Name: ADUN - Add a Unit

PROCESS

1. Verify keywords and store values. (OBKW)
2. Assign section file (ASFL).
3. Determine unit-type for new unit.
4. Edit unit-name and unit-language.
5. Set up accounting information.
6. Initialize to write unit.
- DO UNTIL end of source data
7. Obtain input data and write to unit. (OBSD)
- IF Section is SOURCE or PDL and unit-type not INDEP
8. Process INCLUDES (PCLU)
- ENDIF
- ENDDO
9. Write final information to accounting record (WRAC).
- IF unit was stub
10. Change index entry (CHXE).
- ELSE
11. Add new index entry (ADX).
- ENDIF
12. Print unit (PRUN).
13. Release section file (RLFL).

OUTPUT

[Parameter table]
[Processing status]

PSL library section file
Unit accounting info
Unit source data
Unit higher unit list
Index entries

Unit listing file

Message file

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Subsequent processing is bypassed.	
1	19	ERR	Subsequent processing is bypassed	
1,4	32	ERR	Subsequent processing is bypassed	
1	10	ERR	Subsequent processing is bypassed	
2	7	ADV	Subsequent processing is bypassed	1
3	4	ERR	Subsequent processing is bypassed	
3	3	ERR	Subsequent processing is bypassed	
4	56	ERR	Subsequent processing is bypassed	
6	43	PSL	Subsequent processing is bypassed	
7	44	PSL	Unit is truncated at point where error occurred	2
7	55	ERR	Erroneous "INCLUDE" statement has "%" inserted in front of word INCLUDE and is not processed	
12	400	N/A	Code minus 400 is stored in the unit accounting record as the SP-ERROR-SW if extended accounting data is present	
6	200	FMS	Subsequent processing is bypassed	3

Notes:

- (1) See description of module ASFL for specific conditions which cause the "unable to assign file" condition.
- (2) See description of module ITWR for specific conditions which cause the "unable to initialize write" condition.
- (3) See Sperry Univac 1100 series Executive System Volume 2 Exec Appendix C section C.2 Facility Request Status Codes.

DEFAULT UNIT TYPE

<u>Section</u>	<u>Unit-Type</u>
SOURCE	
Structured language	
(SCOBOL, SPFORT, SJOVIAL)	
Stub already exists	INCLUDED
No unit previously existed	MAIN
Unstructured language	INDEPENDENT
(ASM, COBOL, FORTRAN, JOVIAL)	
General preprocessor indication languages	MAIN
(ASMG, COBOLG, FORTRANG, JOVIALG)	
PDL	MAIN
JOB	MAIN
LINK	MAIN
TEST	MAIN
TEXT	MAIN
USER	MAIN
USER	MAIN

Table 2-b. Determination of Default Unit Type

2.2.1.1.2.2 RPUN - Replace a Unit

The module RPUN, which corresponds to the ** REPLACE function, replaces all of the data lines in a unit of the SOURCE, LINK, PDL, JOB, TEST, or TEXT section of a PSL library. The unit accounting information is not re-initialized, but is updated appropriately. For units in the SOURCE and PDL sections, the lines of data (source code) are scanned for "INCLUDE" statements and pointers are created and deleted in the Included unit to the including unit accordingly. If the included unit does not exist in the section, a stub unit is created to hold the required pointers.

a. Program Operations

HIPO diagram 1.1.2.2 describes the operations performed. In step 4, an independent unit lines of data are not scanned for "INCLUDE" statements. If a stub unit already exists in the section, the unit type that will be stored is INCLUDED. This is because the presence of the stub unit indicates that there exists in the section an "INCLUDE" statement for that unit which has no lines of data. In step 6, if the unit is a stub unit or independent unit, the unit will not be deleted via DELETE A UNIT (DLUN). Instead, an independent unit will be written over to preserve the FMS file description. A stub unit's index will be changed and point to lines of data. In step 10, if the unit index cannot be changed and the unit had previously been deleted, then the unit index will reflect that a stub unit now exists. Consequently, all reference to the stored lines of data are lost.

b. Data File and Table Descriptions

There is no significant files or tables in this module.

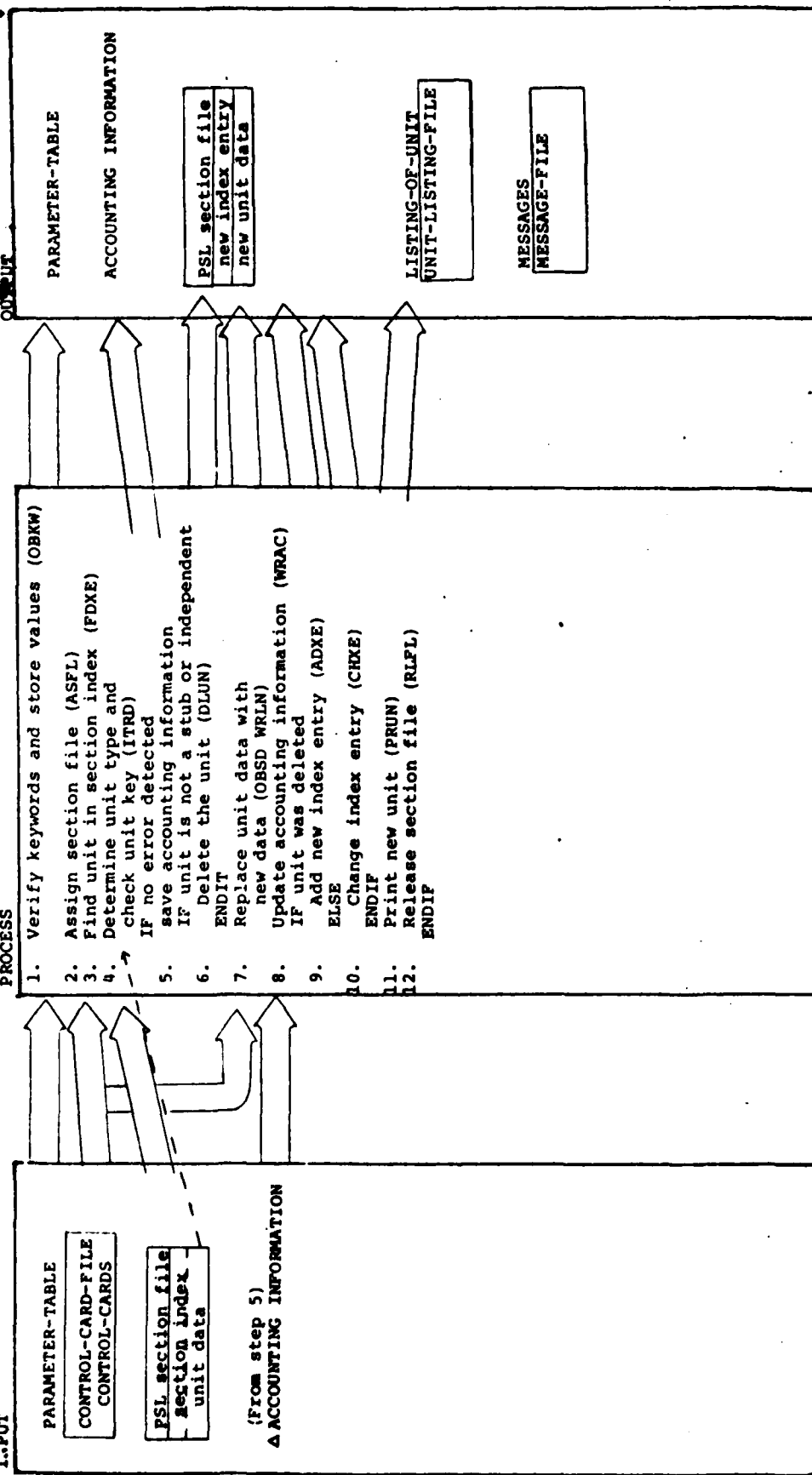
c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Subsequent processing is bypassed	
1	19	ERR	Subsequent processing is bypassed	
1	32	ERR	Subsequent processing is bypassed	
2	6	ERR	Subsequent processing is bypassed	1
3	26	ERR	Subsequent processing is bypassed	

Diagram ID: 1.1.2.2

Name: RPUN - Replace a Unit

1:PUT



c. Branching and Error Conditions (Cont'd.)

3	26	ERR	Subsequent processing is bypassed
6	96	ERR	Subsequent processing is 2,3 bypassed
7	43	ERR	Subsequent processing is bypassed
7	44	ERR	Unit is truncated at 3 point where error occurs
9	24	ERR	Unit is not stored in section file
10	29	ERR	Unit is not stored in section file
11	66	ERR	Unit is not printed
12	37	ERR	Section file is not released

Notes:

- (1) See description of module ASFL for specific conditions which cause the "unable to assign file" condition.
- (2) See description of module DLUN for specific conditions which cause the condition.
- (3) See description of module ITWR for specific conditions which cause the "unable to initialize write" condition.

2.2.1.1.2.3 CHUN - Change a Unit

The module CHUN, which corresponds to the ** CHANGE function, modifies the contents of a unit of data in the SOURCE, LINK, JOB, TEST, TEXT or PDL section of a PSL library. The module CHUN, also corresponds to the ** MDPLAN function which modifies the contents of the PLAN unit in the MANAGEMENT section of a PSL library. Modification details are provided on the Subfunction cards (COPY, DELETE, INSERT, MODIFY, and SHIFT) which follow the CHANGE or MDPLAN card. New source statements follow the INSERT and MODIFY Subfunction cards. The unit accounting information is not re-initialized, but is updated, appropriately. For units in the SOURCE and PDL sections, the lines of data are scanned for "INCLUDE" statements and pointers are created and/or deleted in the included unit to the including unit. If the included unit does not exist in the section, a stub unit is created to hold the required pointers. The module CHUN calls two subroutines, the module CHRC 1.1.2.3.1 and module CHLN 1.1.2.3.2, that are responsible for the detailed update processing. The module CHRC validates and edits all Subfunction cards, formats the specified modification details and stores the formatted details in a temporary file. The temporary file is sorted, then used by the module CHLN. The module CHLN is responsible for modifying FMS file descriptions for independent units, changing the language of a unit, and updating the unit's lines of data as specified by the modification details stored on the temporary file.

a. Program Operations

HIPO diagram 1.1.2.3 describes the operations performed. In step 1, the subroutine module CHRC 1.1.2.3.1 is called to generate the CHANGE-CARD-FILE file and the FMS-PARAMETER-FILE file. The module CHRC also collects important function values necessary for further processing. In step 2, if ** MDPLAN function is specified, the MANAGEMENT section file is accessed. In step 3, the Plan unit is located in the MANAGEMENT section. In step 6, the sort key is comprised of a start line number, an internal sequence number, an end line number, and a subfunction type code. Each record of the CHANGE-CARD-FILE starts with the sort key elements. In step 7, the subroutine module CHLN is called to update the existing unit by using the FMS-PARAMETER-FILE to change Independent unit file description and the CHANGE-CARD-FILE to update unit lines of data. The stored language name of a unit can be changed independent of line updates.

Name - CHUN - Change a Unit

PROCESS



b. Data File and Table Descriptions

1. CHANGE-SORT-FILE

This file is used to store modification details. The file is then used to update a unit of data.

o 01 CHANGE-SORT-RECORD

- 05 FILLER PIC S9(9) COMP.
 - corresponds to the first line number from the Subfunction card
- 05 FILLER PIC S9(9) COMP.
 - internally generated for new lines of data
- 05 FILLER PIC S9(9) COMP.
 - corresponds to the last line number from the Subfunction card
- 05 FILLER PIC X(136)
 - remaining modification details

2. SORT-FILE

This file is used as the collation file for the 1100 Series executive Sort/Merge Program. The CHANGE-CARD-FILE records are identical to the sort records. The sort key is comprised of the first four data items of the record.

o 01 SORT-RECORD

- 05 SORT-FLDS
 - 10 SORT-START-FLD PIC S9(9) COMP.
 - 10 SORT-SEQ-FLD PIC S9(9) COMP.
 - 10 SORT-BND-FLD PIC S9(9) COMP.
 - 10 SORT-TYPE-FLD PIC X(1).
 - internal code denoting the subfunction¹
 - 10 FILLER PIC X(135).
 - remainder of the modification detail¹

¹Refer to DATA FILE description of module CHRC 1.1.2.3.1 for CHANGE-RECORD.

3. FMS-PARAMETER-FILE

Refer to the DATA FILE description of the module
HRC 1.1.2.3.1.

c. Branching and Error Conditions

unction eference	Condition Code	Message Category	Program Action	Note
1	10,2,32,19	ERR	Subsequent processing is bypassed	1
2	6	ERR	Subsequent processing is bypassed	
3	26	ERR	Subsequent processing is bypassed	
4	6	ERR	Subsequent processing is bypassed	
5	26	ERR	Subsequent processing is bypassed	
6	N/A	N/A	N/A	
7	9,100,69	ERR	Subsequent processing is bypassed	2
7	44,45,48,49	PSL	Unit is truncated at point where error occurred	2
7	43,46,47,96	PSL	Subsequent processing is bypassed	2
8	37	PSL	Section file is not released	
9	7	ERR	Continue processing	

otes:

- (1) See description of the module CHRC for specific conditions which cause the various conditions.
- (2) See description of the module CHLN for specific conditions which cause the various conditions.

2.2.1.1.2.3.1 CHRC - Edit Change Records

The module CHRC initiates all processing for the ** CHANGE and ** MDPLAN processing. It obtains the function name from the user's control cards to determine which function has been specified. The module CHRC reads through all user's control cards that apply to the function, validating and storing required values and building temporary files to be used in update processing by the modules CHUN and CHLN.

a. Program Operations

HIPO diagram 1.1.2.3.1 describes the operation performed. The files CHANGE-CARD-FILE and FMS-PARAMETER-FILE are opened at the start of processing and closed at the end of processing because various user supplied control cards and data cards are translated into formatted records describing modification details. The file FMS-PARAMETER-FILE is written only if the user supplies FMS file changes for independent units. In step 7, two editing processes are available, one edit procedure covers the ** CHANGE/ ** MDPLAN cards and their associated keywords and values. The other edit procedure covers the Subfunction cards and their associated keywords and values. In step 8, CHANGE-RECORD is initialized according to Subfunction card and related data cards.

The CHANGE-SORT-RECORD is then written from the CHANGE-RECORD. This is done for all Subfunction cards and data cards. In the case of the Subfunction COPY, file is accessed then the unit, a section to be copied, is located and read. The CHANGE-RECORD is built from the subfunction keyword values and from the lines of data in the copied unit. The unit is read then terminated and the section file released. In step 9, the next Subfunction card is obtained from user's control cards.

b. Data File and Table Descriptions

1. CHANGE-CARD-FILE

Refer to module CHUN for file description.

2. FMS-PARAMETER-FILE

This file stores the user FMS directives for creating or modifying independent files.

• 01 FMS-PARAMETER-RECORD PIC X(96).

- contains FMS file directives and permission for independent files.
- record format is same as INPUT-CARD-KEYWORD-VALUE.

Diagram ID: 1.1.2.3.1

INPUT

[PARAMETER-TABLE]

control-card-file
control-cards

CHANGE-CARD-FILE
CHANGE-CARD-RECORD

PSI-section-file
index entry
unit data

Name - CHRC - Edit Change Records

PROCESS

1. Set all switches off
2. Set valid-function sw on
3. Open output change-card-file, FMS-PARAMETER-FILE
4. Check function-name for MDPLAN
IF MDPLAN function
5. Set mdplan-specified-sw on
ENDIF
- DO while valid function
(MODIFY, INSERT, SHIFT, COPY, DELETE)
- DO until last keyword
6. Obtain keyword (OBKMW)
7. Store and edit keywords
FNDDO
8. Build change-records 1.1.2.3.1.1
(OBSD, ASFL, FDXE)
9. Obtain new function
IF no error detected
10. Set switches off according to
function
ENDIF
ENDDO
11. Close change-card-file,
FMS-PARAMETER-FILE

OUTPUT

PARAMETER-TABLE
CH-UNIT-NAME
CH-UNIT-KEY
CHANGE-LANGUAGE
MOD-NBR
FMS-FILE-MODIFICATION-SW
SORT-OPTION-SW
MOD-SPECIFIED-SW
MDPLAN-SPECIFIED-SW
PROCESSING-STATUS

CHANGE-CARD-FILE
CHANGE-CARD-RECORDS

FMS-PARAMETER-FILE
FMS-PARAMETER-RECORD

MESSAGE-FILE

Diagram ID: 1.1.2.3.1.1

Name: CHRC - Build Change Records

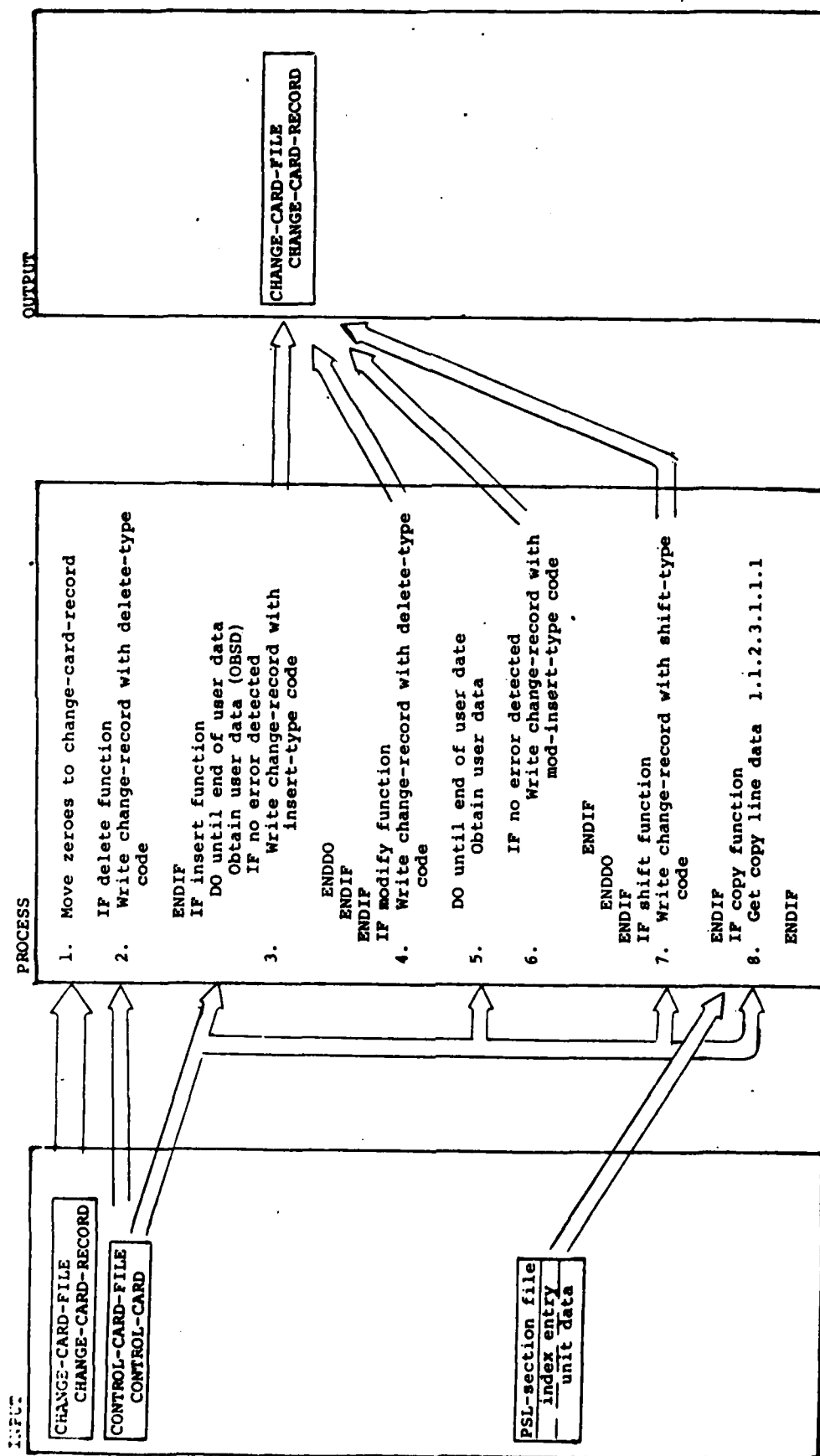


Diagram ID: 1.1.2.3.1.1.1

INPUT

PSL section file
index entry
unit data

CONTROL-CARD-FILE
CONTROL-CARD-FILE

Name: CIIRC - Get Copy Line Data

PROCESS

1. Assign section file according to library and section specified for copy (ASFL)
2. Find unit to be copied (FDXE)
3. Initialize to read unit (ITRD)
DO while copy start line not reached
and no error detected
4. Read unit line (RDLN)
ENDDO
DO UNTIL copy end line or error
detected
5. Write change record with copy type
code
ENDDO
6. Read next unit line (RDLN)
ENDDO
7. Terminate read of unit (TMRD)
8. Release section file (RLFL)

OUTPUT

CHANGE-CARD-FILE
CHANGE-CARD-RECORD

• 01 CHANGE-RECORD

- 05 START-FLD PIC S9(9) COMP.
- corresponds to line-nbr or first-line-nbr 1 in the case of ALL.
- 05 SEQ-FLD PIC S9(9) COMP.
- internally generated number for new lines of data.
- 05 END-FLD PIC S9(9) COMP.
- corresponds to line-nbr or last-line-nbr or high value in the case of ALL.
- 05 CHANGE-TYPE PIC X(1).
- code to denote subfunction
- 05 FILLER PIC X(4).
- unused area
- 05 SHIFT-INDICATOR PIC 9(1).
- indicates direction of shifting 1 denotes left, 0 denotes right
- 05 COLUMN-NBR PIC 9(6).
- column at which to start character string modification
- 05 CHANGE-DATA
- 10 NEW-STRING PIC X(40).
- new character string
- 10 OLD-STRING PIC X(40).
- existing character string
- 05 FILLER PIC X(4).
- unused area
- 05 NEW-STRING-LENGTH PIC S9(9) COMP.
- number of characters in new string

05 OLD-STRING-LENGTH PIC S9(9) COMP.

- number of characters in existing string

05 FILLER PIC X(28).

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
6	19,2,28	ERR	Subsequent processing continues	
7	19,2,32	ERR	Subsequent processing may be bypassed	
8	28,26	ERR	Subsequent processing is bypassed	1
8	14	INF	Subsequent processing continues	
8	6,47,48	PSL	Copy processing bypassed	1
9	14	INF	NORMAL-TERMINATION	

Notes:

- (1) If the COPY Subfunction is specified, see description of module ASFL, module ITRD, and module RDLN for the specific condition which caused error.

2.2.1.1.2.3.2 CHLN - Change Unit Line

The module CHLN modifies the contents of a unit of data and prints a listing of the updated unit. The module CHLN uses the CHANGE-CARD-FILE when updating lines of data in the unit and the FMS-PARAMETER-FILE when an independent unit FMS file description is being updated. The accounting information of the unit to be changed is stored and updated to reflect changes to the unit.

a. Program Operations

HIPO diagram 1.1.2.3.2 describes the operation performed. In step 1, if the user specifies that the FMS file description of an independent unit is to be changed, the FMS parameters are stored on the FMS-PARAMETER-FILE. In step 2, the module CHFL performs the file modification. In step 6, updated independent unit will be stored on a temporary file (TEMP-FILE). The lines stored on the temporary file will be later used to overwrite the existing unit. In step 8, the CHANGE-SORT-RECORDS are read into the CHANGE-TABLE. All update processing is controlled by the table. The CHANGE-TABLE can hold a maximum of six records. Only one insert data record type may be in the table at a time. After a line of data has been processed through the table, any CHANGE-TABLE entry that has executed is removed from the table. The table is compressed and new CHANGE-SORT-RECORDS are read in.

In the case of independent units, the updated data line is written to TEMP-FILE. In step 11, the independent unit is written over with the data lines stored in TEMP-FILE.

b. Data File and Table Descriptions

1. CHANGE-CARD-FILE

Refer to DATA FILE description of module
CHUN 1.1.2.3.

2. FMS-PARAMETER-FILE

Refer to DATA FILE description of module
CHRC 1.1.2.3.1.

3. TEMP-FILE

This temporary file is used to store the updated data lines of an independent unit.

Diagram ID: 1.1.2.3.2

INPUT

REQUESTED-INDEX-ENTRY
FMS-CATALOG-FILE-STRING
UNIT-FILE-NBR
UNIT-INDEX-BLOCK-NBR
SECTION-OPTIONS
CH-UNIT-KEY
PARAMETER-TABLE
CHANGE-LANGUAGE
PGMR-NAME
MOD-SPECIFIED-SW
FMS-FILE-MODIFICATION-SW
FMS-PARAMETER-FILE
FMS-PARAMETER-RECORD
PSL-section-file
PSL-index entry
unit data
TEMP-FILE
TEMP-RECORD
Change-card-file
change-card-record

Name: CHLN - Change Unit Line

PROCESS

- IF unit is independent
1. Setup to modify file
 2. Change file or catalog (CHFC)
 3. ENDIF
 4. ENDIF
 5. IF no errors detected
 6. Initialize to read unit (ITRD)
 7. Initialize update accounting info
 8. Initialize write for new unit (ITWR)
 9. ENDIF
 10. IF no errors detected
 11. Apply changes to unit 1.1.2.3.2.1
 12. ENDIF
 13. Change card-file
 14. Close change-card-file
 15. IF unit is independent
 16. Close temp-file
 17. Rewrite unit using temp-file (WRPLN)
 18. ELSE
 19. Change unit index (CHKE)
 20. ENDIF
 21. Update accounting info (WRAC)
 22. Print new unit (PRUN)

OUTPUT

[PROCESSING-STATUS]

PCL section-file
index entry
unit-date

TEMP-FILE
TEMP-RECORD

PSL section-file
new-index
new-unit

UNIT LISTING FILE

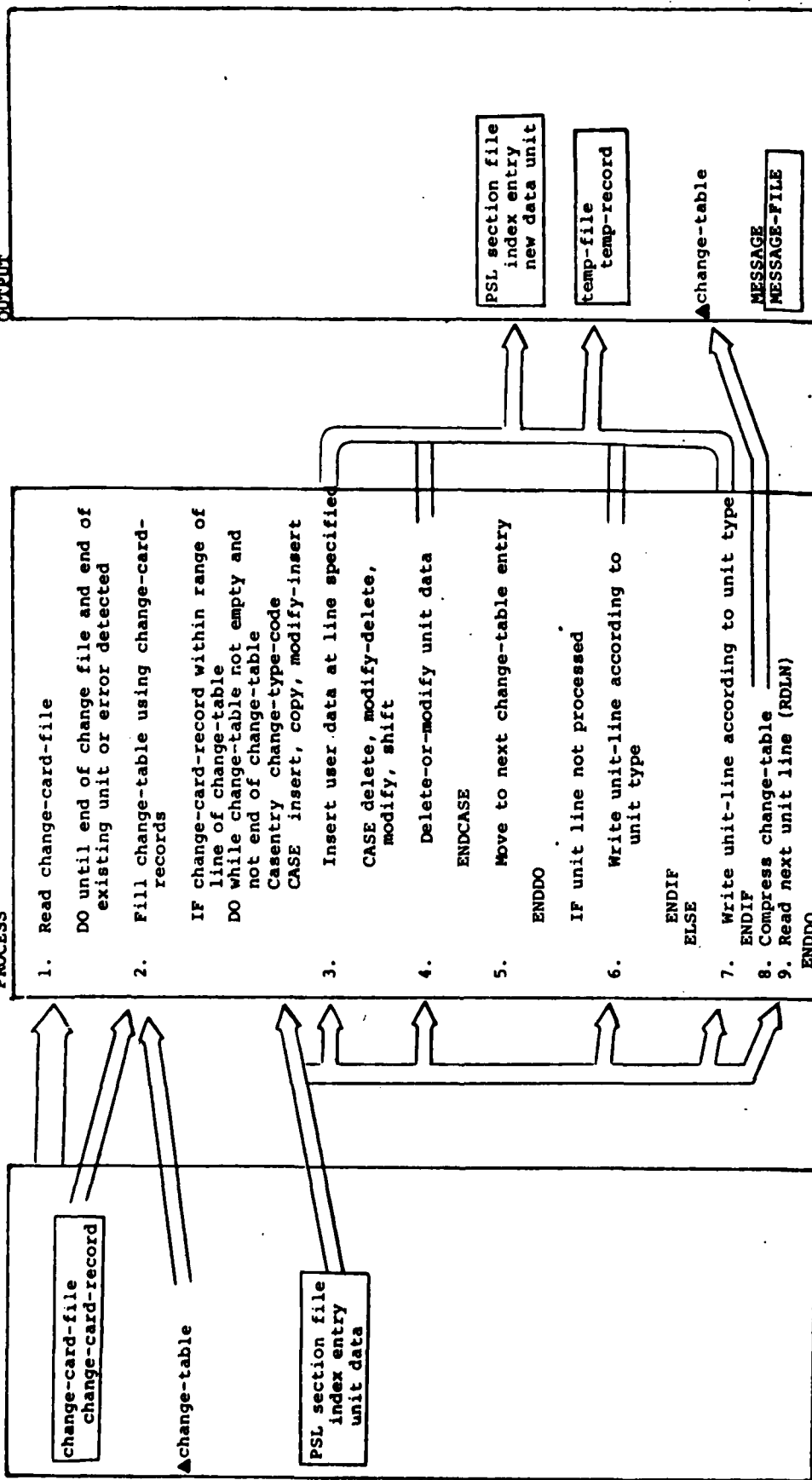
MESSAGE-FILE

Diagram ID: 1.1.2.3.2.1

I:PUT

Name: CHLN - Apply Changes to Unit

PROCESS



• 01 TEMP-RECORD

PIC X(80).

- records contain updated lines of data from CHANGE processing of independent unit.

4. CHANGE-TABLE

This table is used to control the modification of a unit of data. The table allows for a maximum of six entries. The only source of input for the CHANGE-TABLE is the CHANGE-CARD-FILE.

• 05 CHANGE-TABLE-ENTRY OCCURS 6 TIMES.

10 CHANGE-SEQ-NBR

15 CHANGE-START-LINE PIC S9(9) COMP.*

- corresponds to the START-FLD of the CHANGE-RECORD.

15 CHANGE-INPUT-SEQ PIC S9(9) COMP.*

- corresponds to the SEQ-FLD of the CHANGE-RECORD.

10 CHANGE-END-LINE PIC S9(9) COMP.*

- corresponds to the END-FLD of the CHANGE-RECORD.

10 CHANGE-TYPE-CODE PIC 9(10).*

- corresponds to the CHRN

10 FILLER PIC X(4).*

10 CHANGE-SHIFT-DIRECTION PIC 9(1).*

10 CHANGE-COLUMN-NBR PIC 9(6).*

10 CHANGE-INSERT-LINE

15 CHANGE-NEW-STRING PIC X(40).*

15 CHANGE-OLD-STRING PIC X(40).*

10 FILLER PIC X(4).*

10 NEW-STR-LENGTH PIC S9(9) COMP.*

10 OLD-STR-LENGTH PIC S9(9) COMP.*

10 CHANGE-REPLACE-COUNTER PIC S9(9) COMP.*

- number of times CHANGE-OLD-STRING is modified.

10 FILLER PIC X(22).

* Corresponds to data items of CHANGE-RECORD.

c. Branching and Error Conditions

Functions Reference	Condition Code	Message Category	Program Action	Note
1	61,69	ERR	Subsequent processing is bypassed	
2	200	FMS	Subsequent processing is bypassed	1
3	47	PSL	Subsequent processing is bypassed	2
7	43	PSL	Subsequent processing is bypassed	3
8	44,48,45,49	PSL	Unit is truncated at point where error occurs	4,5
11	43,44,45	PSL	Unit is truncated at point where error occurs	5
12	29	ERR	Processing continues	
13	45	PSL	Processing continues	
14	66	ERR	Processing continues	

Notes:

- (1) See Sperry Univac 1100 series Executive System Volume 2 EXEC, Appendix C section C.2 Facility Request Status Codes for specific condition which cause the unable to modify a file condition.
- (2) See description of module ITRD for specific conditions which cause the "unable to initialize read" condition.
- (3) See description of module ITWR for specific conditions which cause the "unable to initialize write condition.
- (4) See description of module RDLN for specific conditions which cause the "unable to read line" conditions.
- (5) See description of module WRLN for specific conditions which cause the "unable to write line" condition.

2.2.1.1.2.4 MVUN - Move a Unit

The module MVUN, which corresponds to the ** MOVE function, moves a unit of data from one project to another, from one library to another, or within a library. The section must be JOB, LINK, PDL, SOURCE, TEST, or TEXT. The module MVUN can move a single unit or an entire section of a PSL library. For units being moved to the SOURCE or PDL section, the lines of data are scanned for "INCLUDE" statements and pointers are created in the INCLUDED unit to the including unit. If the included unit does not exist in the section, a stub unit is created to hold the required pointers.

For a MOVE involving an independent unit, if the space assignment for the receiving unit is insufficient to hold all source data from the old unit, the PSL will abort. In this case, the user should use the PURGE Function to purge the receiving unit from the section and rerun the job using the FMS keyword on the MOVE Function to assign a larger size to the independent unit file. The unit type is subject to change if the unit to be moved does not exist in the receiving section.

a. Program Operations

HIO diagram 1.1.2.4 describes the operations performed. In step 3, all the units of a section are to be moved.

The MOVE processing reads the index of the sending section. Every unit in the sending index is validated. If the unit can be moved, it is located in the receiving section. In step 4, the sending unit cannot be a stub unit. If the user has specified "REPLACE=NO", and the receiving unit is located, the unit cannot be moved. If the receiving unit has a password assigned to it then that unit cannot be moved. If the sending unit is not located in the receiving section, the unit will be added to that section and accounting information updated (unit type).

In step 5, if the receiving unit is independent the MOVE will write over the existing unit. Otherwise, the receiving unit will be written in available spaces within the section file. If the sending unit is independent and the receiving unit is not present, then MOVE processing creates an independent receiving unit with either default or user defined FMS specifications before moving the sending unit. In step 6, the accounting information of the receiving unit is updated to reflect changes in unit type, higher unit name, lines in unit, version, and modification number. In step 9, a single unit is located in the sending and receiving section index.

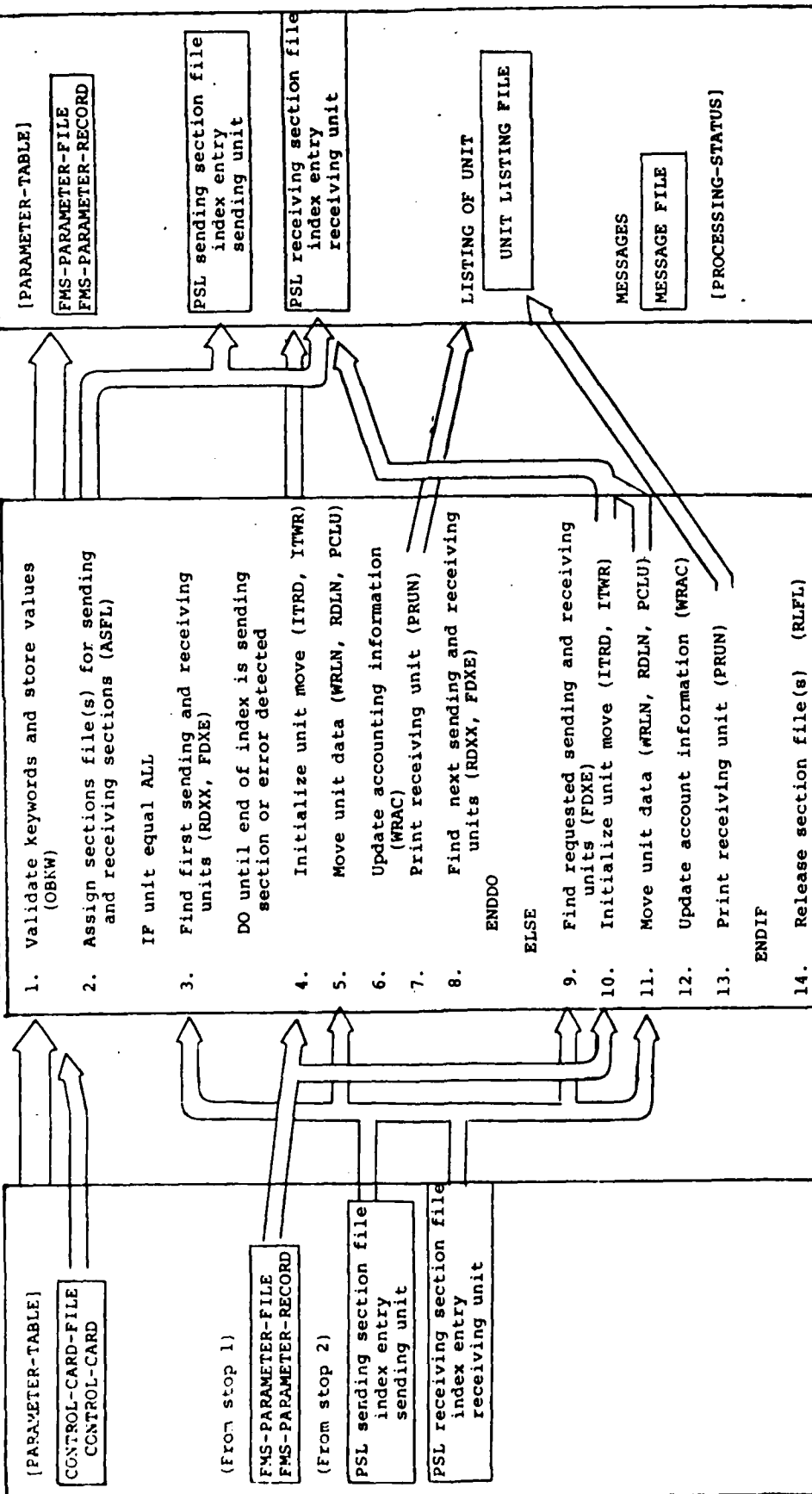
Diagram ID: 1.1.2.4

Name: MVUN - Move a Unit

INPUT

PROCESS

OUTPUT



b. Data File and Table Descriptions

1. FMS-PARAMETER-FILE

Refer to DATA FILE description in module
CHRC 1.1.2.3.1.

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2,10,19,32, 103,104	ERR	Subsequent processing is bypassed	
2	6	PSL	Subsequent processing is bypassed	
3	26	ERR	Processing continues	
3	7	ERR	Subsequent processing is bypassed	
3	106	ADV	Unit processing is bypassed	
4,10	47,43,49,45	PSL	Unit processing is bypassed	1,2
4,10	69,102,105	ERR	Unit processing is bypassed	
4,10	200	FMS	Unit processing is bypassed	3
4,10	133	ADV	Unit processing is bypassed	
5,11	44,48,49,45	PSL	Unit is truncated at point where error occurs	
5,11	38,39,40,55	ERR	Processing continues	4
5,11	96,29,24	PSL	Moved unit is not address in the receiving section	
5,11	46	PSL	Accounting information is updated	
6,12	66	ERR	Unit is not listed (Printed)	

c. Branching and Error Conditions (Cont'd.)

7,13	7	ERR	Unit is not moved; Processing continues
14	37		Section files not released

Notes:

- (1) See description of the module ITRD for specific conditions which cause the "unable to initialize unit" condition.
- (2) See description of the module ITWR for specific condition which cause the "unable to initialize write to unit".
- (3) See Sperry Univac 1100 series Executive System Volume 2 EXEC, Appendix C section C.2 Facility Request Status Codes for specific condition which cause the unable to modify file condition.
- (4) See description of the module PCLU for specific conditions which cause the rejected include statement for unit condition.

2.2.1.1.2.5 PGUN - Purge a Unit

The module PGUN, which corresponds to the PSL function
** PURGE, removes an individual unit from a library.

a. Program Operations

HIPO diagram 1.1.2.5 describes operations performed. In step 4, a unit to be purged is checked for unit type if the MGMT section has been specified. This is because only the COLLECTION units can be purged with the purge function in the MGMT section. If a unit is not found in the section index, step 8 attempts to purge an independent unit file anyway. This is because it is possible for the independent unit file to have been created by a previous PSL function but the index entry not to have been entered due to an interruption in the processing. The most common situation is that an INDEPENDENT unit is being added by module ADUN (or MVUN OR RSLB) but the file space allocated by the user is exhausted, and a program abort occurs before the index entry has been entered.

b. Data File and Table Descriptions

No unique data files or significant tables are used.

c. Branching and Error Conditions

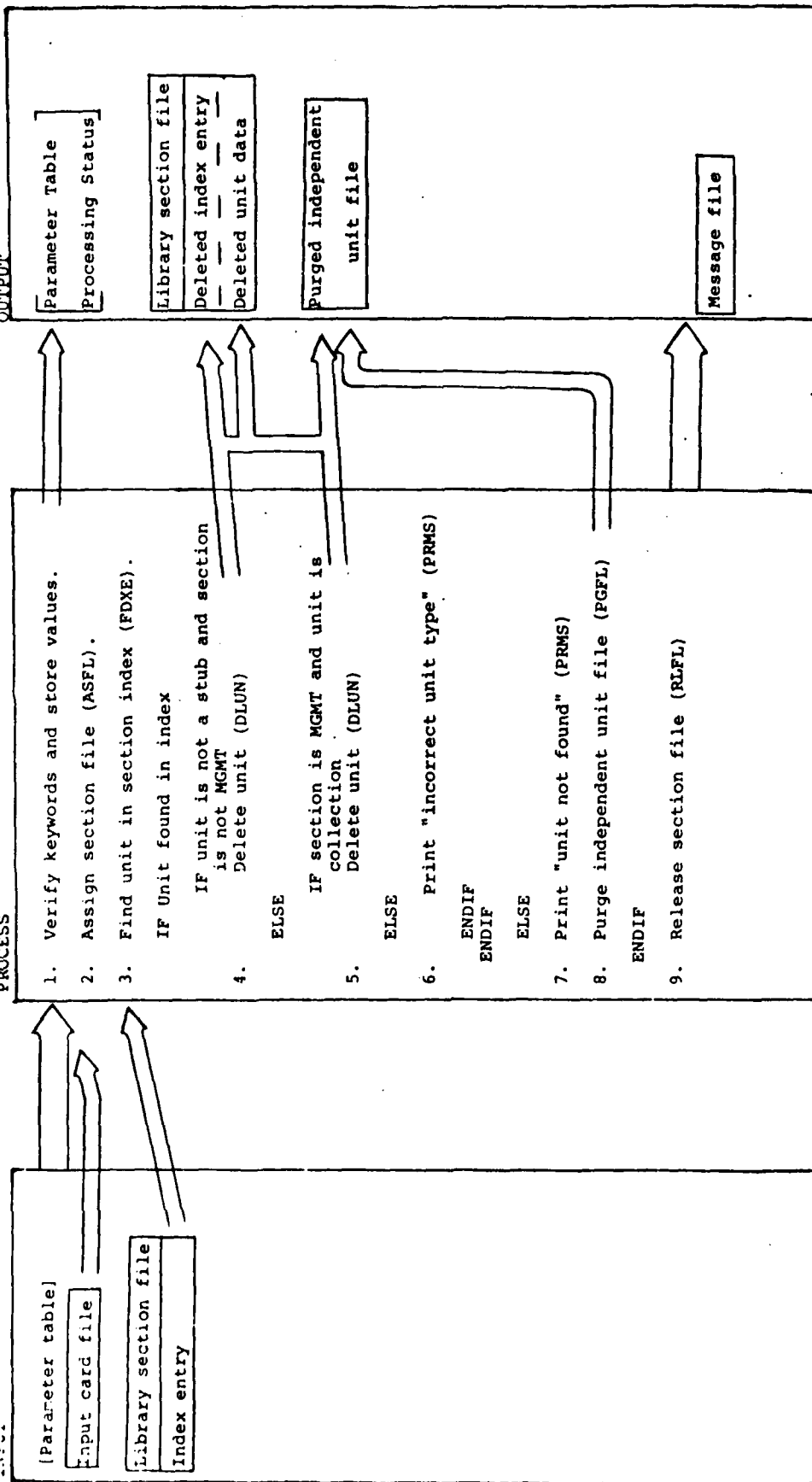
Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Subsequent processing is bypassed	
1	19	ERR	Subsequent processing is bypassed	
1	32	ERR	Subsequent processing is bypassed	
2	7	ADV	Subsequent processing is bypassed	1
3	26	ERR	Attempt to purge independent unit file	
4,5			Error return from DLUN. Unit not purged	2
6	154	N/A	Wrong unit type for MGMT section. Unit not purged.	

Diagram ID: 1.1.2.5

PGUW

PGUW - Purge a Unit

PROCESS



Notes:

- (1) See description of module ASFL for detailed list of causes of "unable to assign a file" condition.
- (2) See description of module DLUN for detailed list of causes of "unable to delete unit" condition.

2.2.1.1.3 Program Processing

Program processing functions are performed to compile a module (COMPILE), link a program (LINK) and execute a program (EXECUTE). The program modules whose operations are especially invoked by these functions are described below.

2.2.1.1.3.1 CMML - Compile a Module

The user-designated SOURCE section unit is located and a compilation procedure is identified by the SOURCE unit language or a user-designated procedure name. The identified procedure is located in the System project PSL library JOB section and additional file reference information is added to the job control cards stored there. A job is spawned using these control cards to direct the required compilation activities including precompiler operation, when applicable.

a. Program Operations

HIPO diagram 1.1.3.1 depicts CMML program operations. If the user-designated "top" (i.e., MAIN, CALLED, or INDEPENDENT) unit is found, the accounting record of that unit is updated to indicate the compilation attempt (provided that the section option MGMTDATA=YES was chosen when the SOURCE section was created). If compiler object code output is to be provided (from the spawned job procedure) then an OBJECT section must exist in the project and library specified by the Parameter Table entries to receive the object index which is to be stored as a relocatable element unit in the PROGRAM section under the same unit index name as the top SOURCE code unit identified by the user. If that unit is not entered in the OBJECT section, it is added to that section. A unit accounting record is written to PSL OBJECT section storage to contain the required management data statistics. If, on the other hand, the unit is already stored in the OBJECT section, the unit accounting record is modified to update the management data statistics. The applicable compilation procedure is then retrieved and the job control cards are processed and modified to insert the additionally required file reference information before being written to the Spawned Job file.

b. Data File and Table Descriptions

No special files or tables are utilized.

c. Branching and Error Conditions

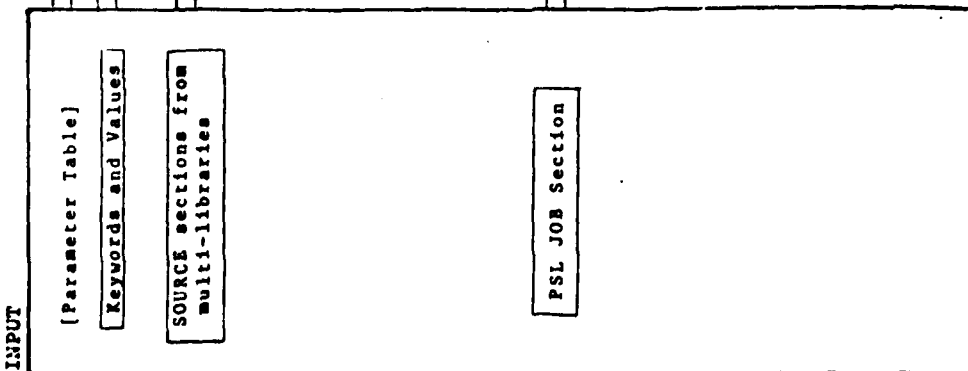
The following branching and error conditions apply to HIPO diagram 1.1.3.1:

Diagram ID: 1.1.3.1

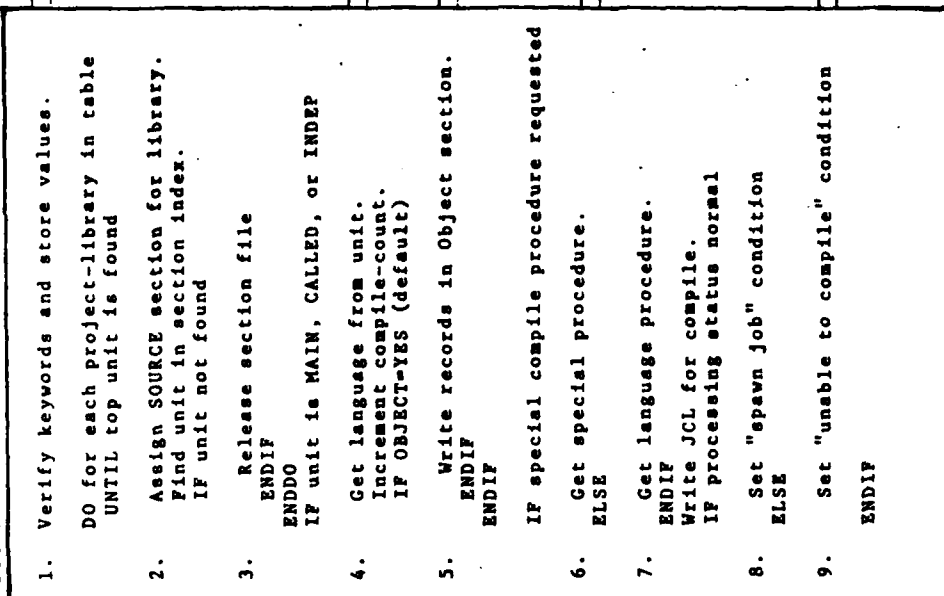
Name: CMNL - Compile a Module

Description: Function Processor (COMPILE)

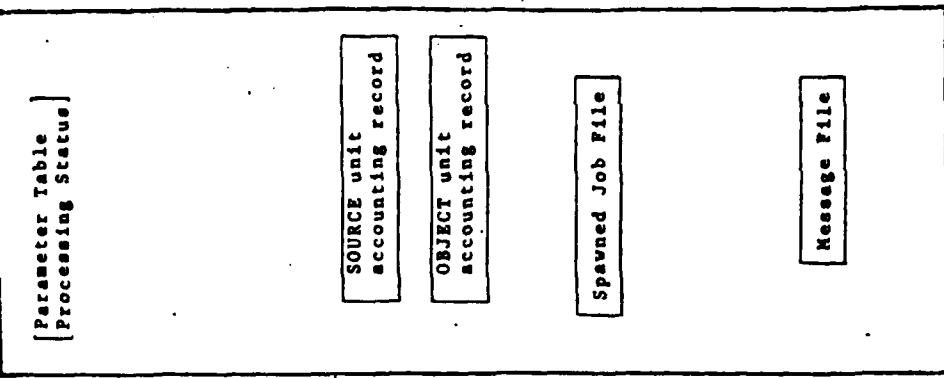
INPUT



PROCESS



OUTPUT



<u>Function Reference</u>	<u>Condition Code</u>	<u>Message Category</u>	<u>Program Action</u>	<u>Note</u>
1	2	ERR	Perform Process #10	
	19	ERR	Perform Process #10	
	32	ERR	Perform Process #10	
2	6	ADV	Perform Process #10	
	26	ERR	Perform Process #10	
	92	ERR	Perform Process #10	
5	6	ADV	Perform Process #10	
7	81	ERR	Perform Process #10	
8	81	ERR	Perform Process #10	
9	16	ADV	Normal Exit	
10	7	ERR	Error Exit	

2.2.1.1.3.1.1 PPCB - Pre-process COBOL

The module PPCB obtains source code from one or more Source sections of the user's PSL libraries and translates structured COBOL code into standard ANS COBOL. The special structuring control statements which are translated are: IF, ELSE, ENDIF, DO, ENDDO, CASEENTRY, CASE, ELSECASE, ENDCASE and INCLUDE. All other statements which appear in the input source code are passed to the compiler unchanged (except that any statement appearing before a structuring statement will have a period added to it if none already exists).

a. Program Operations

HIPO diagrams 1.1.3.1.1 and 1.1.3.1.1.1 describe operations performed. More detailed HIPO diagrams can be found in the Structured Programming Series, Volume II Precompiler Specifications, May 1975 Section 5. HIPO diagram 1.1.3.1.1.1 describes operations which have been added to the original COBOL precompiler, described in the reference document, to handle the INCLUDE statement.

b. Data File and Table Descriptions

1. PRECOMPILER-OUTPUT File

The PRECOMPILER-OUTPUT file contains the source code generated by the precompiler from the user's input code. Only the special structuring control statements cause translated code to be generated on this file. All other input statements are written to the PRECOMPILER-OUTPUT file unchanged, with the exception that a period is added to the end of those statements which appear immediately preceding one of the structuring statements, if one did not previously exist. The translations for the structuring statements IF, ELSE, ENDIF, DO, ENDDO, CASEENTRY, CASE, ELSECASE and ENDCASE are shown in the PSL User's Guide, June 1977, Section G.5. Additional code is also generated by the precompiler on the PRECOMPILER-OUTPUT file under the following conditions:

- Preceding the first statement of the module.
- In place of each INCLUDE statement for which a real unit of code is found in the input PSL libraries.
- Following the last statement of each unit of code processed.

Diagram ID: 1.1.3.1.1

INPUT

PSL libraries SOURCE sections
Unit data lines

Name: PPCB - Preprocess COBOL

PROCESS

1. Open output file.
2. Get source data line until procedure division found. 1.1.3.1.1.1
3. Get a source line. 1.1.3.1.1.1
DO UNTIL end of input
4. Find first word on line.
5. Process according to word:

```
CASE
CASENTY
DO
ELSE
ELSECASE
ENDCASE
ENDDO
ENDIF
ENTER
IF
Not a structure figure
```
6. Get a source line. 1.1.3.1.1.1
ENDDO
7. Close output file.

OUTPUT

COBOL source file for
compiler

Message listing

Diagram ID: 1.1.3.1.1.1

INPUT

PSL libraries	SOURCE
section	
Unit data lines	
Unit accounting info	
Index entries	

o Save-line-stack

Name: PPCB - Get a Source Line

PROCESS

IF no generated statement saved in stack
IF not end-of-module

1. Read source line (RDCM).

CASENTRY Return-from-RDCM

CASE Line-of-unit

2. Save line-of-unit

CASE INCLUDE statement -
start of unit: start-of-module
Make comment of INCLUDE statement

3. Generate start-of-unit (module),
PROJ/LIBR comments

4. Add period to end of previous
statement

5. Increment Level number

CASE End-of-unit (module)

6. Decrement level number

7. Generate end of unit (module)
comment

CASE INCLUDE statement-stub unit,
INCLUDE error

8. Make comment of INCLUDE statement

IF PROCEDURE DIVISION started

9. Generate DISPLAY statement

ENDIP

10. Generate PROJ/LIBR comment

11. ENDCASE

ENDIP

ELSE

Get statement from top of save-stack

ENDIP

OUTPUT

o Line-of-unit

o Save-line-stack

- In place of each INCLUDE statement for which a stub unit is found in the input PSL libraries.
- In place of each INCLUDE statement for which an error is detected.
- Following the last line of the module.

Figure 2-04 shows the code which is generated in these cases.

2. Data Storage Push Down Stacks

Data required by the precompiler is stored in one dimensional arrays called push down stacks. All stacks except the OUTPUT-STACK are accessed on a last in first out (LIFO) basis. The program uses six such stacks. In all stacks the index always points to the next available storage entry in the stack.

- LABEL-STACK indexed by LABEL-NEXT-OPEN-ENTRY

Each entry is 13 characters in length. The entries in this stack hold the internally generated paragraph-names which the precompiler produces.

- NEST-STACK-ENTRY indexed by NEST-NEXT-OPEN-ENTRY

Each time the start of a new figure is detected, a new entry is placed in this stack. It is 7 characters in length and contains 2 alphabetic characters to identify the type of figure started and an integer in the range of 1 through 9999. This integer is left justified and padded on the right with dashes and is used to start all paragraph-names generated for that figure.

- CONDITIONAL-STACK indexed by COND-NEXT-OPEN-ENTRY

This stack holds the COBOL conditional expression which follows the WHILE/UNTIL keywords on the structured DO statements. It is saved until the corresponding ENDDO is detected at which point the stored conditional expression and its associated IF or IF NOT are moved from this stack to the output stack. The entries in this stack are 86 characters in length. In addition to the 80-column record holding the condition (which may extend over more than one record) the entry contains a 5-digit source input sequence number and a special

- Start of module comments


```

      ***1*** START OF MODULE      //// top-unit-name      ////
      *****PROJ=project-name LIBR=library V/M vvv/mod UPDATED mo/dd/yy hh:mn
      
```
- Code generated for INCLUDE statement with real unit


```

      * INCLUDE included-unit-name
      ***1*** START OF UNIT      //// included-unit-name      ////
      *****PROJ=project-name LIBR=library V/M vvv/mod UPDATED mo/dd/yy hh:mn
      
```
- Code generated for INCLUDE STATEMENT WITH STUB UNIT


```

      * INCLUDE included-stub-name
      DISPLAY "INCLUDED STUB      /included-stub-name      /".
      ***** INCLUDED STUB      //// included-stub-name      ////
      *****PROJ=project-name LIBR=library V/M vvv/mod UPDATED mo/dd/yy hh:mn
      
```
- End of unit comment


```

      ***1*** END OF UNIT      //// included-unit-name      ////
      
```
- End of module comment


```

      ***1*** END OF MODULE      //// top-unit-name      ////
      
```
- Code generated for INCLUDE statement error


```

      * Erroneous INCLUDE statement
      DISPLAY "INCLUDE ERROR      /included-unit-name      /".
      ***** INCLUDE ERROR      //// included-unit-name      ////
      *****PROJ=project-name LIBR=library V/M vvv/mod UPDATED mo/dd/yy hh:mn
      
```

Figure 2-04. Code Generated by COBOL Precompiler PPCB

flag to delimit the start of the conditional from the one below it, a situation which occurs when DO figures are nested.

- CASE-STACK indexed by CASE-NEXT-OPEN-ENTRY

This stack holds the identifier which appears on the CASEENTRY structure verb. It is saved until the ELSECASE (or ENDCASE if no ELSECASE is present) corresponding to the appropriate CASEENTRY is detected. The identifier is then removed from the stack and used as part of the GO TO ... DEPENDING ON statement which is generated at the end of the CASE Statement. Each entry is 30 characters long. Also associated with each identifier is a CASE-LABEL-POINTER index which contains the starting value of the index of the CASE-LABEL-STACK at which the paragraph-names for this CASE starts.

- CASE-LABEL-STACK indexed by CASE-LABEL-NEXT-OPEN-ENTRY

This stack holds the paragraph-names associated with the CASE verbs that are part of a CASE statement. The format of these names are as indicated in Section 4 with the numbers on the CASE verb being used to pad the last three characters of the paragraph-name. These paragraph-names are not only generated as entry points to the CASE code block when a CASE verb is encountered, but are also saved in the stack until the end of the CASE figure is detected for use in the generated GO TO ... DEPENDING ON statement.

- SOURCE-OUTPUT-STACK indexed by
SO-NEXT-OPEN-ENTRY
SO-CURRENT-NON-BLANK
SO-PREVIOUS-NON-BLANK
SO-CURRENT-KEYWORD
SO-PREVIOUS-STATEMENT
SO-NEXT-KEYWORD

This stack is the one in which all precompiler output, including generated code is placed, prior to being written to the output file. It contains six different indices which must be manipulated as output records are placed in the stack and removed. Its manipulation is complicated by the fact that until the current keyword is identified and processed, it is necessary to

hold the previously processed statement in the stack. This is because if the keyword currently being processed generates code, it is necessary to add a period to the end of the statement preceding it (if one is not already there). Therefore it is not possible to route a statement to the output file as soon as it is processed. A second complication arises when scanning for the end of a character string associated with a keyword such as a conditional on an IF or DO. The end of a conditional is signaled by the detection of an ANS COBOL or structured programming verb. The records containing these data are all kept in the output stack and therefore indices are required to indicate their positions. A description of the function of the six indices follows:

a. **SO-NEXT-OPEN-ENTRY**

This index points to the first entry in the output stack which does not contain a program statement. This is the position into which the next card image to be processed will be placed. (Note: SO stands for SOURCE-OUTPUT.)

b. **SO-CURRENT-NON-BLANK**

This index points to the last entry in the output stack which contains a non-blank program statement. When looking for a keyword, the statement pointed to by SO-CURRENT-NON-BLANK is examined.

c. **SO-PREVIOUS-NON-BLANK**

This index points to the next to the last entry in the output stack which contains a non-blank program statement. When looking for the next COBOL or structuring verb, the end of the last COBOL statement (SO-LAST-STATEMENT) preceding that verb is set from SO-PREVIOUS-NON-BLANK.

d. **SO-CURRENT-KEYWORD**

This index points to the entry in the output stack which contains the first character of the keyword being processed. When looking for a keyword, SO-CURRENT-KEYWORD is set from SO-CURRENT-NON-BLANK.

e. SO-PREVIOUS-STATEMENT

This index points to the entry in the output stack which contains the last line of the COBOL or structuring statement previous to the statement being processed. When looking for the next COBOL or structuring verb for processing, SO-PREVIOUS-STATEMENT is set from SO-PREVIOUS-NON-BLANK.

f. SO-NEXT-KEYWORD

This index points to the entry in the output stack which contains the first character of the word following the keyword currently being processed. When looking for the COBOL or structuring verb which starts the next statement to be processed, it must first be determined if the statement containing that verb has already been read from the input file. If the SO-NEXT-KEYWORD index points to the same statement as the SO-CURRENT-KEYWORD index, the next statement has not yet been read. This is true because of the rule that each new statement must start on a new line. Thus, if the next keyword occurs on the same line as the current keyword, it cannot be the beginning of a new statement.

c. Branching and Error Conditions

Errors which affect processing and the resulting action are described in the PSL User's Manual, June 1977, Section G.7.

2.2.1.1.3.1.2 PPFT - Preprocess FORTRAN

The PPFT module obtains source code from one or more source sections of the user's PSL libraries and translates structured FORTRAN code into standard FORTRAN. The special structuring control statements which are translated are: IF, ELSE, ENDIF, DO WHILE, DO UNTIL, ENDWHILE, ENDUNTIL, ENDDO, CASE OF, CASENTRY, CASE, CASEELSE, ELSECASE, ENDCASE, INVOKE, BLOCK, ENDBLOCK and INCLUDE. All other statements which appear in the input source code are passed to the compiler unchanged.

a. Program Operations

HIPO diagram 1.1.3.1.2 describes operations performed by the PPFT module. Additional information regarding the capability as it was developed by the General Research Corporation for RADC may be obtained from the Final Technical Report, Structured Programming Translators, RADC-TR-76-253, Volumes I to V.

Diagram ID: 1.1.3.1.2

Name: PPFT - Pre-process FORTRAN

Description: FORTRAN Precompiler

INPUT

Control card file

SOURCE section files

PROCESS

1. DO for each FORTRAN statement
DO UNTIL all lines of statement are read
2. Get line of code (RDCM)
ENDDO
3. Determine statement type
4. IF statement type is:
IF, ELSE, ENDIF, DO WHILE,
DO UNTIL, ENDO, CASE OF, CASEENTRY,
CASE, CASEELSE, ELSECASE,
ENDCASE, INVOKE, BLOCK,
ENDBLOCK, INCLUDE
5. Generate appropriate FORTRAN
statements
ELSE
Pass input statement to output
ENDIF
6. Determine indentation for output code
7. Indent output statement(s)
8. Write statement(s) on FORTRAN
output file
IF end-of-module
9. Clear structure figure
nesting stacks
10. Print module errors
ENDIF
ENDDO

Message File

Generated
source code file

OUTPUT

2.2.1.1.3.1.3 PPJV Preprocess JOVIAL

The module PPJV obtains source code from one or more Source sections of the user's PSL libraries and translate structured JOVIAL code into standard JOVIAL, (JOVIAL-J-3) as defined in the Air Forces document AFM 100-24 with modifications, as indicated in the JOCIT Compiler User's Manual. The special structuring control statements which are translated are: IF, ELSE, ENDIF, DO, ENDDO, CASENTRY, CASE, ELSECASE, ENDCASE and INCLUDE. All other statements which appear in the input source code are passed to the compiler unchanged.

a. Program Operations

HIPO diagrams 1.1.3.1.3 and 1.1.3.1.3.1 describe operations performed. The following HIPO diagrams depicts the processing of the Structured Programming JOCIT/JOVIAL precompiler as produced under U. S. Air Force (RADC) contract #F30602-76-C-0166. In step 2, the OPTION-CARDS file is read to obtain the preprocessor options (MAP and LIST). If no card is present the default options which are defined in the Data Division are retained. In step 4, HIPO diagram 1.1.3.1.3.1 describe operations which have been added to the original JOVIAL precompiler to handle the INCLUDE statements. Each input record is processed, if a listing has been requested in the option card the record is printed. In step 6, the JOVIAL precompiler generates standard JOVIAL statements to replace the structured verb statements. The precompiler also checks the nesting of structured figures and the syntax associated with structured code.

b. Data File and Table Descriptions

1. Option-CARDS File

The Option-Cards file contains a single 80 character input record which contains keywords to indicate whether a listing of the precompiler JOVIAL input is to be generated and whether the sequence numbers on the optional precompiler source listing are to be placed in columns 73 through 80 of the precompiler output.

2. JOVIAL-OUTPUT File

The JOVIAL-OUTPUT file contains the source code generated by the precompiler from the user's input code. Only the special structuring control statements cause translated code to be generated on this file. All other input statements are written to the JOVIAL-OUTPUT

file unchanged. The translations for the structuring statements IF, ELSE, ENDIF, DO, ENDDO, CASENTRY, CASE, ELSECASE, and ENDCASE are shown in the PSL User's Guide, Section H.5. Additional code is also generated by the precompiler on the JOVIAL-OUTPUT file under the following conditions.

- Preceding the first statement of the module.
- In place of each INCLUDE statement for which a real unit of code is found in the input PSL libraries.
- Following the last statement of each unit of code processed.
- In place of each INCLUDE statement for which a stub unit is found in the input PSL libraries.
- In place of each INCLUDE statement for which an error is detected.
- Following the last line of the module. Figure 2-05 shows the code which is generated in these cases.

```

• Start of module comments

"***1*** START OF MODULE      //// top-unit-name      ////
"*****PROJ=project-name LIBR=library V/M vvv/mod UPDATED mo/dd/yy hh:mn'

• Code generated for INCLUDE statement with real unit

"      * INCLUDE included-unit-name      "
"***1*** START OF UNIT      //// included-unit-name      ////
"*****PROJ=project-name LIBR=library V/M vvv/mod UPDATED mo/dd/yy hh:mn'

• Code generated for INCLUDE STATEMENT WITH STUB UNIT

"      * INCLUDE included-stub-name      "
      DISPLAY "INCLUDED STUB      "
"***** INCLUDED STUB      //// included-stub-name      ////
"*****PROJ=project-name LIBR=library V/M vvv/mod UPDATED mo/dd/yy hh:mn"

• End of unit comment

"***1*** END OF UNIT      //// included-unit-name      ////

• End of module comment

"***1*** END OF MODULE      //// top-unit-name      ////

• Code generated for INCLUDE statement error

"      * Erroneous INCLUDE statement      "
      DISPLAY "INCLUDE ERROR /included-unit-name      "
"***** INCLUDE ERROR      //// included-unit-name      ////
"*****PROJ=project-name LIBR=library V/M vvv/mod UPDATED mo/dd/yy/hh:mn"

```

Figure 2-05. Code Generated by JOVIAL Precompiler PPJV

3. PRINT-OUT File

The PRINT-OUT file generates the listing of the source lines to be input into the JOVIAL precompiler.

- 01 JOVIAL-PHRASES-AND-LABELS

This data group contains all the standard JOVIAL formats which are used by the precompiler in translating structured JOVIAL statements.

- 01 LINE-HOLD-STACK¹

This stack contains the additional code which is generated by the precompiler on the JOVIAL-OUTPUT file as indicated in the description of JOVIAL-OUTPUT.

- 01 CASE-NUMB-STACK¹

This stack may hold a maximum of 200 case numbers. The stack contains the structured figure number and the interger associated with the case statement. Once a CASE statement is terminated with an ENDCASE, the CASE numbers associated with it are removed from the stack.

- 01 NEST-STACK

This stack contains the starting and intermediate structure words of each structure set for which no terminating word has yet been encountered. The maximum depth to which figures may be stacked is set at 300.

- 01 NEST-NUMBER-STACK¹

This stack contains the nest numbers of each structured figure which at any time in the program has not yet been terminated. The maximum depth to which figures may be nested is set to 50.

- 01 CONDITION-STACK¹

The conditions on DO statements are saved in the stack and removed when the corresponding ENDDO is detected. The stack is set to hold 50 cards maximum.

- 01 CONDITION-COUNT-STACK¹

The CONDITION-COUNT-STACK is used with DO statements to keep track of the number of cards in each of the conditions stacked in the CONDITION-STACK.

¹Descriptions from Structured Programming JOCIT/JOVIAL Precompiler User's Manual - April 1977.

c. **Branching and Error Conditions**

Errors which affect processing and the resulting action are described in the PSL User's Manual, Section H.7.

Diagram ID: 1.1.3.1.3
INPUT

PSL libraries SOURCE
section
Unit-data-lines

Name: PPJV - Preprocess JOVIAL
PROCESS

1. Open output file.
2. Read and process precompiler options.
DO until end of input
3. Initialize processing
4. Get a source line 1.1.3.1.3.1
IF not end of file
DO until end of module or
JOVIAL exit statement
5. Find first word on line
6. Process according to word

```
CASE  
CASENTRY  
DIRECT  
DO  
ELSE  
ELSECASE  
ENDDO  
ENDIF  
IF  
TERM  
not a structure figure
```

7. Get a source line 1.1.3.1.3.1
ENDDO
8. Set program end flag
ENDIF
ENDDO
9. Close output files

Description: Program Processing
OUTPUT

JOVIAL source file for
compiler

Message listing

Diagram ID: 1.1.3.1.3.1
I:PUT

PSL libraries SOURCE
section
Unit data lines
Unit accounting info
Index entries

(from steps 4,6,8)
• Line-hold-stack

Name: Get a Source Line
PROCESS

```

IF no generated-statement saved
  IF not end-of-module
    Read source line (RDCM)
    CASENTRY return from RDCM
  CASE line-of-unit
    Move line-of-unit to
      output-area
  CASE start-of-unit (module),
    Increment level number
    Generate start-of-unit (module),
      PROJ/LIBR comments
  CASE end-of-unit (module),
    Decrement level number
    Generate end-of-unit (module)
      comment
  CASE INCLUDE statement-stub unit,
    INCLUDE error
  Make comment of INCLUDE
    statement
  Generate PROJ/LIBR comment
    ENDCASE
  ENDIF
ELSE
  Get statement from top of line-hold-
    stack
  ENDIF
  IF not end of program
    IF list option specified
      Write source line
    ENDIF
  ENDIF

```

Description: Program Processing

- Line-of-unit
- Line-hold-stack

2.2.1.3.1.4 PPGL - General Preprocessor

The General Preprocessor provides an alternative method for processing unstructured source code. The General Preprocessor allows unstructured code the features, data compression and INCLUDE processing. Data compression is obtained by storing unstructured code in the SOURCE section file in random blocks. Formerly, unstructured code was only stored in independent files as sequential card images. INCLUDE processing is handled at compilation time. When initiated by the COMPILE function, the General Preprocessor scans the unstructured source code for INCLUDE statements and inserts the included units. At present the General Preprocessor accomodates four languages: ASM, COBOL, FORTRAN, and JOVIAL.

a. Program Operations

HIPO diagram 1.1.3.1.4 describes operations performed by the PPGL module. PPGL calls READ-FOR-COMPILE (RDCM) until the end-of-module condition is encountered. Upon return from RDCM the LINE-STATUS is set to indicate one of the following conditions: START OF MODULE (HIPO process steps 5,6,7,9,10), START OF UNIT (process steps 5,8,9,10), END OF MODULE (process steps 11,12,13), END OF UNIT (process steps 11,14,15), INCLUDED STUB (process steps 16,17,18), INCLUDE ERROR (process steps 16,17,18) or LINE OF DATA, i.e., any source line which is not one of the preceding conditions, (process steps 3 and 4). After the first call to RDCM (LINE STATUS = START OF MODULE), PPGL determines the UNIT LANGUAGE and prepares the comment areas for the given language. The generated comments are discussed below.

b. Data File and Table Descriptions

PRECOMPILER-OUTPUT FILE

The PRECOMPILER-OUTPUT File contains the source code generated by the General Preprocessor from the user's input source code. The INCLUDE statements are processed and the included units are copied in-line into the PRECOMPILER-OUTPUT File. All other input statements are written to the PRECOMPILER-OUTPUT File unchanged. Additional comments are generated by PPGL under the following conditions:

- Preceding the first statement of the module.
- In place of each INCLUDE statement for which a real unit of code is found in the input PSL libraries.

- Following the last statement of each unit of code processed.
- In place of each INCLUDE statement for which a stub unit is found in the input PSL libraries.
- In place of each INCLUDE statement for which an error is detected.
- Following the last line of a COBOL module.
- Preceding the last line of an ASM, FORTRAN, or JOVIAL module.

Figures 2-06, 2-07, 2-08 and 2-09 show the code which is generated in these cases.

c. Branching and Error Conditions

Errors are detected and handled by RDCM. See Section 2.2.1.2.4.1 RDCM (page 2-255) for error conditions and messages.

Diagram ID: 1.1.3.1.4

Name: PPCL

Page 1 of 2

INPUT

Source Section
Files

PROCESS

1. DO UNTIL END-OF-MODULE
2. Read source line (RDCM)
CASEENTRY return from RDCM
(LINE-STATUS +1)
3. CASE Line of Source
4. Output any previous hold-line
Create new hold-line from
source line with type
determined by unit-language
5. CASE Start-of-Module/Start-of-Unit
Increment level-number
IF Start-of-Module
6. Determine unit language*
7. Prepare comment fields
depending on unit-language
8. ELSE (Start-of-Unit)
Output any previous hold-line
ENDIF
9. Output Start-of-Module/
Start-of-Unit comment
Output PROJ/LIBR comment
10. CASE End-of-Module/End-of-Unit
Decrement level-number
IF End-of-Module
12. for COBOL: Output End-of-
Module comment after output-
ting last source line.
For ASM, FORTRAN, JOVIAL:
Output End-of-Module comment
before outputting last
source line.
ELSE (End-of-Unit)
- 13.

* If language not COBOL, FORTRAN, CHAP or
JOVIAL then no comments will be generated
by PPCL.

OUTPUT

Message File

Precompiler-Output
File

INPUT

PROCESS

```
14.      Output any previous hold-line
15.      Output End-of-Unit comment
      ENDIF
      CASE Include Statement
16.      Stub-Unit/Include-Error
17.      Output any previous hold-line
      Create and output comment from
18.      Include Statement
      Output Stub-Unit/Include-Error
      comment
      ENDCASE
      ENDDO
```

OUTPUT

```

• Start of module comments

****1** START OF MODULE      //// top-unit-name      ////
*****PROJ-project-name LIBR-library V/M vvv/mod UPDATED mo/dd/yy hh:mn

• Code generated for INCLUDE statement with real unit

1* INCLUDE included-unit-name
****1** START OF UNIT      //// included-unit-name      ////
*****PROJ-project-name LIBR-library V/M vvv/mod UPDATED mo/dd/yy hh:mn

• Code generated for INCLUDE STATEMENT WITH STUB UNIT

2* INCLUDE included-stub-name
*****PROJ-project-name LIBR-library V/M vvv/mod UPDATED mo/dd/yy hh:mn
***** STUB OF UNIT      //// included-stub-name

• End of unit comment

****1** END OF UNIT      //// included-unit-name      ////

• End of module comment

****1** END OF MODULE      //// top-unit-name      ////

• Code generated for INCLUDE statement error

3* Erroneous INCLUDE statement
***** INCLUDE ERROR      //// included-unit-name      ////

```

Figure 2-06. Code Generated by PPGL for COBOL Modules

```

• Start of module comments

C**1** START OF MODULE      //// top-unit-name      ////
C*****PROJ=project-name LIBR=library V/M vvV/mod UPDATED mo/dd/yy hh:mn

• Code generated for INCLUDE statement with real unit

C      INCLUDE included-unit-name
C*****PROJ=project-name LIBR=library V/M vvV/mod UPDATED mo/dd/yy hh:mn
C***** STUB OF UNIT      //// included-stub-name

• End of unit comment

C**1** END OF UNIT      //// included-unit-name      ////

• End of module comment

C**1** END OF MODULE      //// top-unit-name      ////

• Code generated for INCLUDE statement error

C      Erroneous INCLUDE statement
C***** INCLUDE ERROR      //// included-unit-name      ////

```

Figure 2-07. Code Generated by PPGL for FORTRAN Modules

```

• Start of module comments
  • *1** START OF MODULE      //// top-unit-name      ////
  • ****PROJ=project-name LIBR=library V/M vv/mod UPDATED mo/dd/yy hh:mn

• Code generated for INCLUDE statement with real unit
  • INCLUDE included-unit-name
  • *1** START OF UNIT      //// included-unit-name      ////
  • ****PROJ=project-name LIBR=library V/M vv/mod UPDATED mo/dd/yy hh:mn

• Code generated for INCLUDE STATEMENT WITH STUB UNIT
  • INCLUDE included-stub-name
  • ****PROJ=project-name LIBR=library V/M vv/mod UPDATED mo/dd/yy hh:mn
  • **** STUB OF UNIT      //// included-stub-name      ////

• End of unit comment
  • *1** END OF UNIT      //// included-unit-name      ////

• End of module comment
  • *1** END OF MODULE      //// top-unit-name      ////

• Code generated for INCLUDE statement error
  • Erroneous INCLUDE statement
  • **** INCLUDE ERROR      //// included-unit-name      ////

```

Figure 2-08. Code Generated by PPGL for ASM Modules

```

• Start of module comments
  ''*1** START OF MODULE // top-unit-name
  ''*****PROJ=project-name LIBR=library V/M vv vv/mod UPDATED mo/dd/yy hh:mn''

• Code generated for INCLUDE statement with real unit
  ''
  ''      INCLUDE included-unit-name
  ''*1** START OF UNIT // included-unit-name
  ''*****PROJ=project-name LIBR=library V/M vv vv/mod UPDATED mo/dd/yy hh:mn''

• Code generated for INCLUDE STATEMENT WITH STUB UNIT
  ''
  ''      INCLUDE included-stub-name
  ''*****PROJ=project-name LIBR=library V/M vv vv/mod UPDATED mo/dd/yy hh:mn''
  ''***** STUB OF UNIT // included-stub-name

• End of unit comment
  ''*1** END OF UNIT // included-unit-name

• End of module comment
  ''*1** END OF MODULE // top-unit-name

• Code generated for INCLUDE statement error
  ''
  ''      Erroneous INCLUDE statement
  ''***** INCLUDE ERROR // included-unit-name

```

Figure 2-09. Code Generated by PPGL for JOVIAL Modules

2.2.1.1.3.2 LKPG - Link a Program

A user-program is identified through a LINK section unit input as one or more object modules whose names appear on LINK unit INCLUDE cards. For each object name which is a stub, a stub unit is compiled. The INCLUDE cards are changed to Collector IN cards and all of the Collector control cards in the LINK unit are output to a spawned job which produces and stores the user-program load module.

a. Program Operations

HIPO diagrams 1.1.3.2 and 1.1.3.2.1 depict the operation of the LKPG module. After successfully verifying and storing the keyword value input, the LINK unit specified is searched for in up to nine project libraries. If no error condition exists, the LINK unit is processed by PPLK to create object stubs and IN cards. LKPG then produces an @MAP card directing the absolute element to be stored in the proper PROGRAM section. Next LKPG invokes the Collector with the Map File from PPLK as the Collector control card input. Finally the LOAD section is accessed to store accounting information for the newly created LOAD unit.

b. Data File and Table Descriptions

1. PROJECT LIBRARY TABLE

This table is generated from project and library keyword inputs. Up to nine project library combinations may be specified. The following table format applied:

• 01 PROJECT-LIBRARY-TABLE.

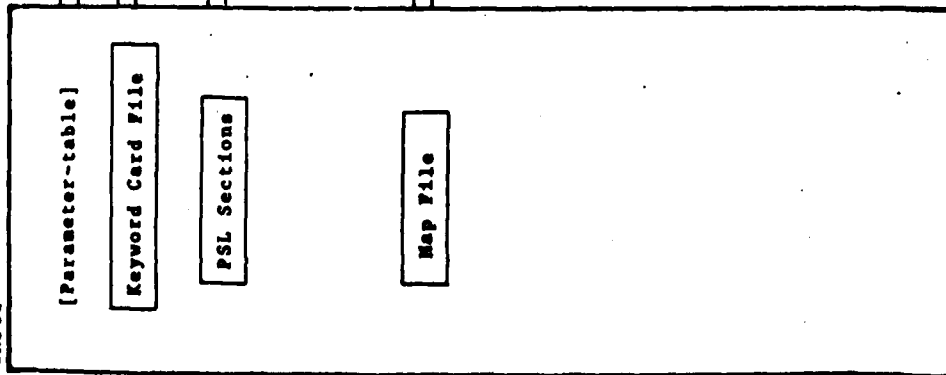
05	PLT-ENTRY	OCCURS 9 TIMES.
10	PLT-PROJECT	PIC X(12).
10	PLT-LIBRARY	PIC X(7).

c. Branching and Error Conditions

The branching and error conditions which apply to HIPO diagram 1.1.3.2 are given in the following table:

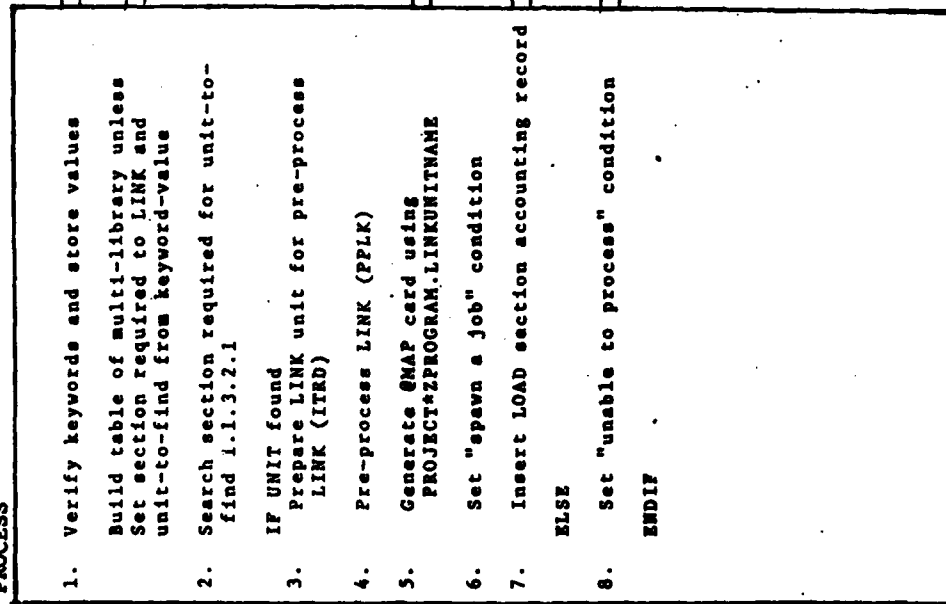
Diagram ID: 1.1.3.2

INPUT



Name: LKPG - Link a Program

PROCESS



Description: Function Processor (LINK)

OUTPUT

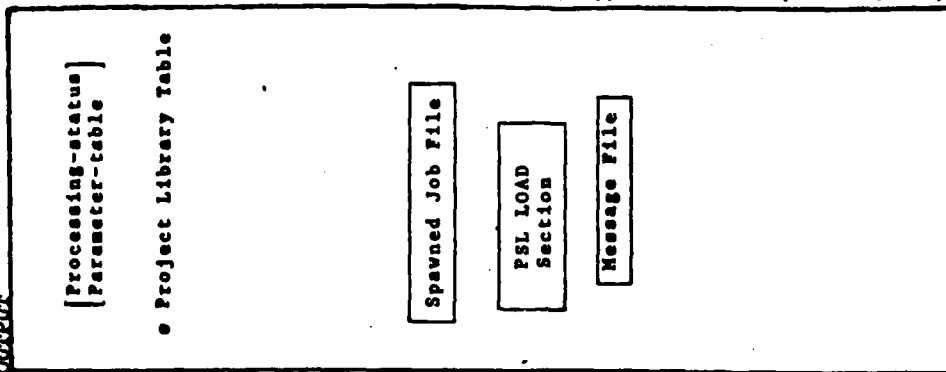
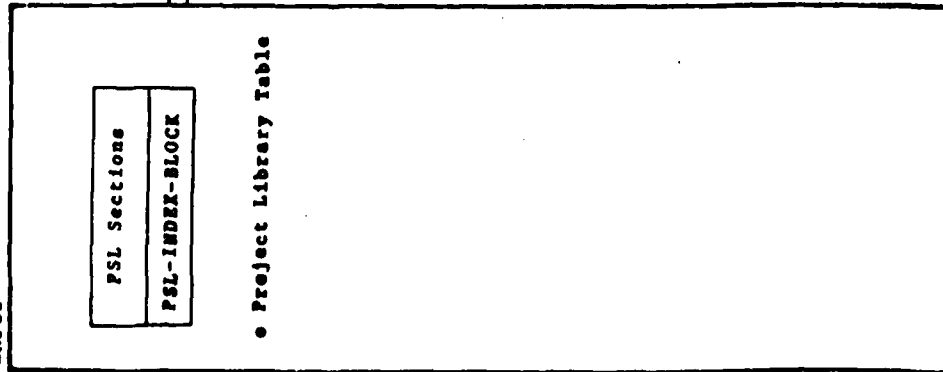


Diagram ID: 1.1.3.2.1

INPUT



Name: LKPC - Link a Program

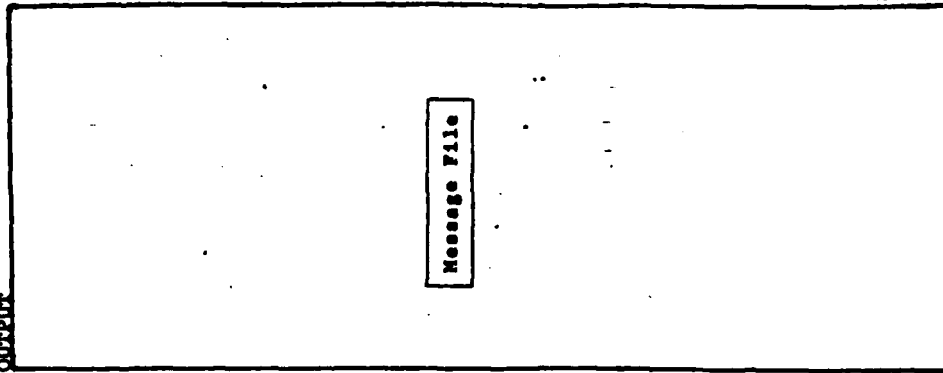
PROCESS

DO for each library specified by user
UNTIL unit-to-find is located

1. Assign file for section required (ASPL)
2. Search for unit-to-find (FDXE)
IF unit not found
ENDIF
3. Release section file (RLFL)
ENDDO
IF unit not found
4. Print message (PMNS)
ENDIF

Description: Search for required unit

OUTPUT



<u>Function Reference</u>	<u>Condition Code</u>	<u>Message Category</u>	<u>Program Action</u>	<u>Note</u>
1	2	ERR	Perform Process #8	
	19	ERR	Perform Process #8	
	32	ERR	Perform Process #8	
2	6	ADV	Bypass Process #3	
4	26	ERR	Perform Process #8	1
5	6	ADV	Perform Process #8	
6	81	ERR	Perform Process #8	
7	16	ADV	Normal Exit	
8	7	ERR	Error Exit	

Note:

- (1) When an OBJECT unit is not found, the error condition is reset to zero and processing continues normally since an OBJECT stub unit will be provided in "pre-process link" operations.

2.2.1.1.3.2.1 PPLK - Pre-process Link

PPLK processes LINK units attempting to locate the OBJECT elements listed on INCLUDE cards. If no object module exists PPLK spawns a compile to create a relocatable element for the stub.

a. Program Operations

HIPO diagrams 1.1.3.2.2 and 1.1.3.2.2.1 show the operation of the PPLK module. PPLK is called by either LKPG or EXPG with the LINK unit initialized for reading. PPLK reads the LINK unit and locates OBJECT units. If an OBJECT unit cannot be located, a stub is generated by modifying a stored source stub and compiling the stub before the Collector is invoked. Only the INCLUDE cards are modified (to IN cards) by PPLK. All other Collector control cards are written to the MAP FILE unchanged.

b. Data File and Table Descriptions

No special files or tables are required.

c. Branching and Error Conditions

The branching and error conditions that apply to HIPO diagram 1.1.3.2.1 are as follows:

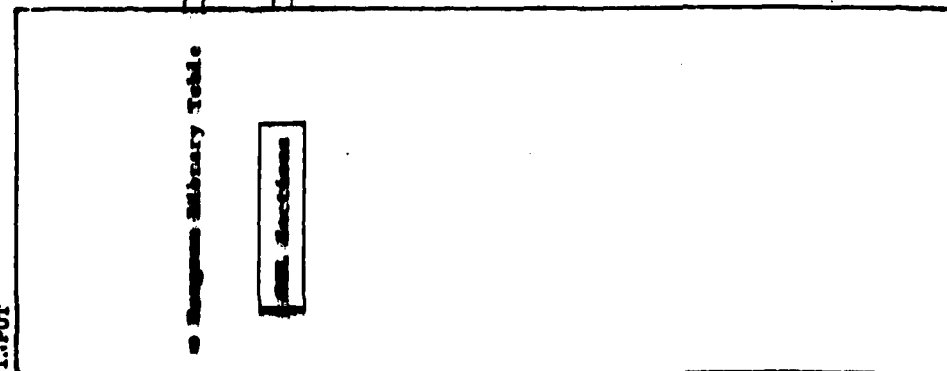
Function Reference	Condition Code	Message Category	Program Action	Note
5	84	ERR	Conclude procedure	
6	84	ERR	Conclude procedure	
	85	ERR	Conclude procedure	
7	6	ADV	Terminate program	
8	83	ERR	Terminate program	
9	132	ERR	Go to ENDDO	
12	84	ERR	Terminate program	
	85	ERR	Terminate program	

Diagram ID: 1.1.3.2.2

Name: PPLK - Pre-process LINK

Description: Generate Collector Cards

INPUT



PROCESS

1. Initialize STUB-TYPE-FLAG to NO-STUBS
2. DO for each line of LINK unit
3. Read line of unit
IF INCLUDE
4. Locate OBJECT unit
IF OBJECT unit not found
IF first INCLUDE
5. Set STUB-TYPE-FLAG to MAIN
ENDIF
6. Compile an OBJECT stub 1.1.3.2.2.1
7. Set STUB-TYPE-FLAG to CALLED
ENDIF
8. Replace "INCLUDE" with "IN"
ENDIF
9. Write line of unit
ENDDO
- IF STUB-TYPE-FLAG NOT-NO-STUBS
10. Write LIB 9AC0B*3R1. card
ENDIF

OUTPUT

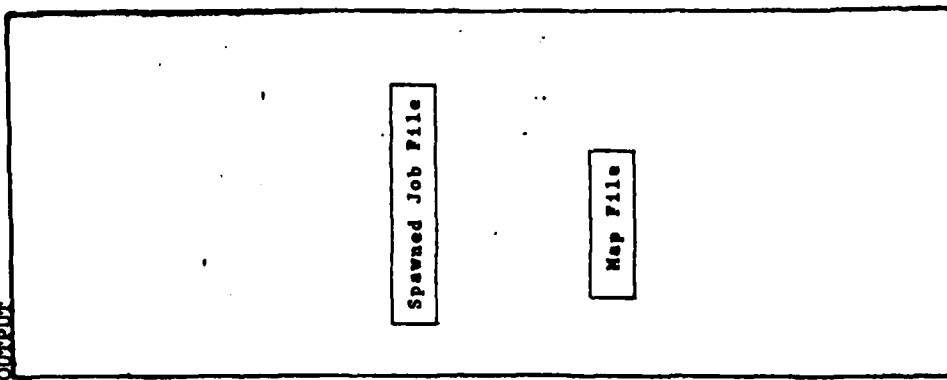


Diagram ID: 1.1.3.2.2.1

INPUT

[STUB-TYPE-FLAG
OBJECT-UNIT-NAME]

STUB SOURCE
UNIT

Name: PPLK - Pre-process LINK

PROCESS

IF STUB-TYPE-FLAG = MAIN

1. Write compile card for a MAIN stub

ELSE

2. Write compile card for a called stub

ENDIF

3. Read STUB SOURCE UNIT

4. Set PROGRAM-ID name to OBJECT-UNIT-NAME

5. Write STUB-SOURCE-UNIT

Description: Compile an OBJECT
Stub

OUTPUT

Spawned Job File

2.2.1.1.3.3 EXPG - Execute a Program

A user program is retrieved from either a load module in the LOAD section or a series of one or more object modules selected by collector control cards in the LINK section. The user's job control cards for execution of his program are stored in a unit in the JOB section. Special PSL flags are placed in these cards to direct the system to insert additional file reference information into the spawned job control cards.

a. Program Operations

HIPO diagram 1.1.3.3 depicts the top unit of code for the EXPG module. The procedure for validating keywords and keyword values culminates in the building of a Project Library Table listing all the user-designated libraries to be searched. A search is first made for the LOAD or LINK section unit designated by the user, followed (when successful) by a search for the JOB section unit containing the user's job control cards according to the procedure shown in HIPO diagram 1.1.3.3.1. If the user's program is contained in a LOAD section unit, it is already in a loadable format, otherwise the object module(s) specified will be directed to the collector by control cards spawned by PPLK. This procedure is essentially the same as that used by the LKPG module described in paragraph 2.2.1.1.3.2. The job control cards (designated by the user) to execute the user program are then written to the Spawned Job file so as to contain the additional file references, as shown in HIPO diagram 1.1.3.3.2.

b. Data File and Table Descriptions

1. PROJECT LIBRARY TABLE

This table is generated from project and library keyword inputs. Up to nine project library combinations may be specified. The following table format applies:

• 01 PROJECT-LIBRARY-TABLE.

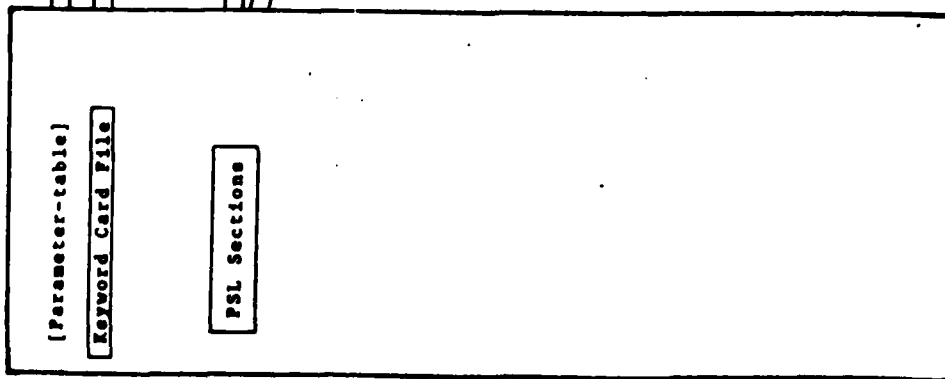
05	PLT-ENTRY	OCCURS 9 TIMES.
10	PLT-PROJECT	PIC X(12).
10	PLT-LIBRARY	PIC X(7).

Diagram ID: 1.1.3.3

Name: EXPG - Execute a Program

Description: Function Processor
(EXECUTE)

INPUT



PROCESS

1. Verify keywords and store values
Build table of multi-library values
Set section required to LINK or LOAD
and unit-to-find from keyword-value
find 1.1.3.3.1
2. Search section required for unit-to-
find 1.1.3.3.1
3. Search JOB section for user's job
unit 1.1.3.3.1
IF LINK option
IF LINK unit found
4. Prepare LINK unit for pre-process
LINK (ITRD)
5. Pre-process LINK (PPLK)
ELSE
6. Error Condition
ENDIF

OUTPUT

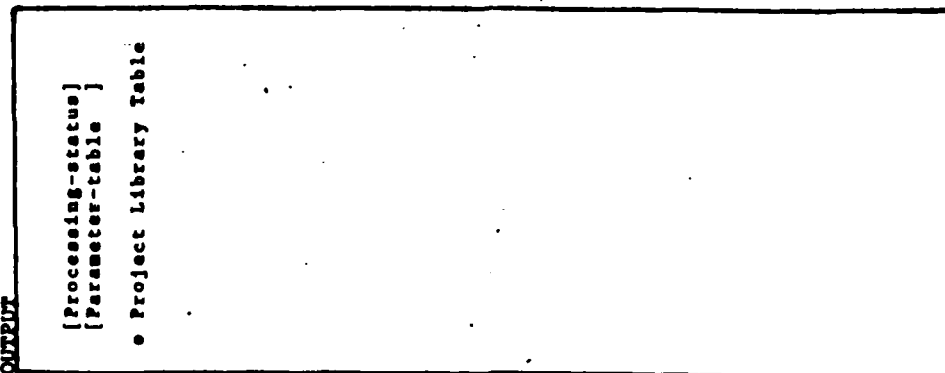


Diagram ID: 1.1.3.3 (continued)

Name: EXPG - Execute a Program

Description: Function Processor (EXECUTE)

INPUT

MAP FILE

PROCESS

```
7.  Generate QMAP card using
    TPFS.LINKUNITNAME
    ENDIF

8.  Write JCL for execute 1.1.3.3.2
    IF successful completion
9.  Set "spawn a job" condition
    ELSE
10. Set "unable to process" condition
    ENDIF
```

OUTPUT

Spawned Job File

Message File

Diagram ID: 1.1.3.3.1

Name: ZPGC - Execute a Program

Description: Search for required unit

INPUT

PSL Sections
PSL-INDEX-BLOCK

o Project Library Table

PROCESS

DO for each library specified by user
UNTIL unit-to-find is located

1. Assign file for section required (ASFL)
2. Search for unit-to-find (PDXE)
IF unit not found
3. Release section file (RLFL)
ENDIF
ENDDO
IF unit not found
4. Print message (PRMS)
ENDIF

OUTPUT

Message File

Diagram ID: 1.1.3.3.2

INPUT

User's PSL Library
JOB Section
PSL-DATA-BLOCK

Name: XPG - Execute a Program

Description: Write JCL for
EXECUTE

PROCESS

1. Initialize to read user's job unit
(ITRD)
2. DO for each line of job unit
3. Read line of unit
IF line-type = EXECUTE
IF LOAD option
4. Write QXQT card using
PROJECT=PROGRAM.LOADUNITNAME
ELSE
5. Write QXQT card using
TPP%.LINKUNITNAME
ENDIF
ELSE
6. Write line of unit
ENDIF
ENDDO

Spawned Job File
JCL Card

OUTPUT

c. Branching and Error Conditions

The following branching and error conditions apply to HIPO diagram 1.1.3.3.

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Perform Process #10	
	7	ERR	Perform Process #10	
	32	ERR	Perform Process #10	
2	26	ERR	Perform Process #10	1
4	26	ERR	Perform Process #10	
9	16	ADV	Normal Exit	
10	7	ERR	Error Exit	

Notes:

- (1) If OBJECT section unit cannot be found, error message is output but condition code is reset to zero allowing processing to continue normally since an object STUB unit will be supplied via the PPLK module operation (refer to paragraph 2.2.1.1.3.2.1).

2.2.1.1.3.4 PCML - Precompile Module

The user-designated unit is located in the SOURCE section and the indicated precompilation procedure is located in the PSL System project library JOB section. PSL file reference information is inserted into the job control cards found in the located job unit before writing them out to Spawn-Job file from which the precompilation activity is spawned.

a. Program Operations

HIPO diagram 1.1.3.4 depicts PCML program operations. If the user-designated "top" (i.e., MAIN, CALLED or INDEPENDENT) unit is found, the applicable precompilation procedure is determined from the source unit language or from the special precompilation procedure name provided in the PROCEDURE keyword input. The precompilation procedure is then retrieved and the job control cards are processed to modify and insert the additionally required PSL file references before writing them to the Spawn Job File. Processing of the job control cards is completed when the flagword "COMPILE" is detected in a job control card or the last job control card is processed.

b. Data File and Table Description

No special files or tables are utilized.

c. Branching and Error Conditions

The following branching and error conditions apply to HIPO diagram 1.1.3.4:

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Skip to error exit	
1	19	ERR	Skip to error exit	
1	32	ERR	Skip to error exit	
4	92	ERR	Skip to error exit	
5	26	ERR	Skip to error exit	
9	81	ERR	Skip to error exit	
10	7	ERR	Error Exit	
10	10	ADV	Normal Exit	

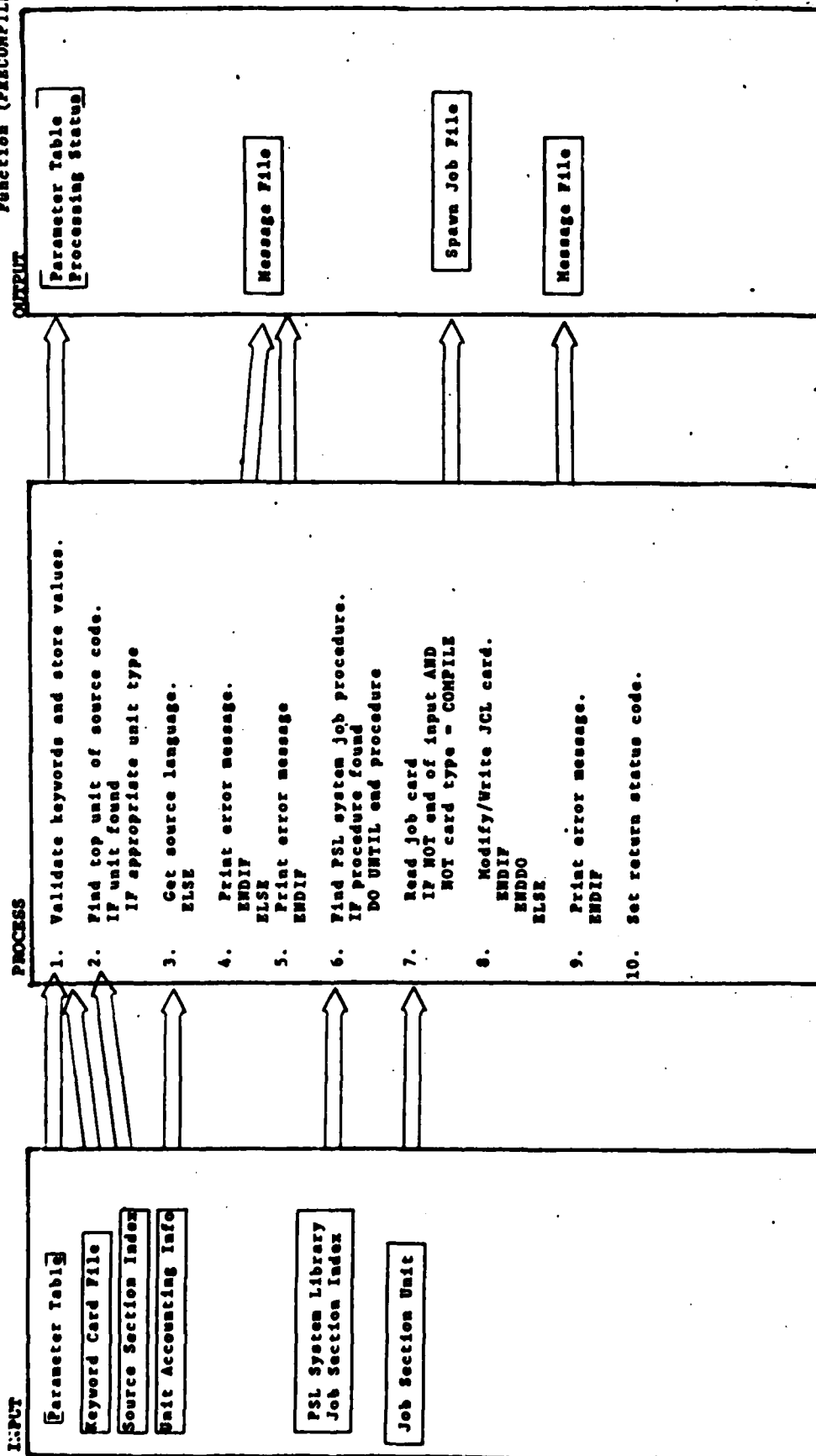
Diagram ID: 1.1.3.4

INPUT

Name: PCHL - Precompile Module

Description: Program Processing Function (PRECOMPILE)

PROCESS



2.2.1.1.4 Management Data

Management data functions are performed to update a management plan (MDPLAN), update a management data format (MDFORMAT), update manual input data (MDUPDATE), establish exception checking (MDXCHECK), initiate management data collection (MDCOLLECT), and print management reports (MDPRINT). The program modules whose operations are especially invoked by these functions are described below.

2.2.1.1.4.1 FMMD - Format Management Data

The FMMD module controls the process to edit, move and update format input/output records by which management data format units are established and maintained.

a. Program Objectives

HIPO diagram 1.1.4.1 depicts the program operations of the FMMD module. In step 1, editing the keywords indicate what operation is to be performed (MOVE, ADD, CHANGE, DELETE). In step 3, the management section index is searched for the format unit name. If the name is not present and the operation is "MOVE", the subroutine FMMVE is called. In step 4, the subroutine FMEDE is called to read and edit either new data or updates to existing data. Consequently, the subroutine FMEDE builds the TRANS-FILE, which is used by the subroutine FMUPE to update the format units. In step 5, the TRANS-FILE is sorted on ascending values of the DATA-RECORD-ID. In step 7, the subroutine FMUPE will write a new unit in the Management section. The new unit will be recorded in the Management section index only after it has been successfully stored.

b. Data File and Table Descriptions

• TRANS-FILE

The TRANS-FILE is used to store new and update format records generated by the editing of format records. The file is sorted for the updating process, which regards the record-id as unique.

• SORT-FILE

The SORT-FILE is used as the collation file for the 1100 series Executive Sort/Merge Program. The TRANS-RECORD is identical to the SORT-RECORDS.

01 FMMD-KEY DATA

The data group FMMD-KEY DATA stores values for the ** MDFORMAT function card and from the accounting information of an existing format unit. The values are used to indicate specific processing and to restrict access.

05 FMMD-KEY

PIC X(12).

- unit key assigned when unit was added.

Diagram ID: 1.1.4.1

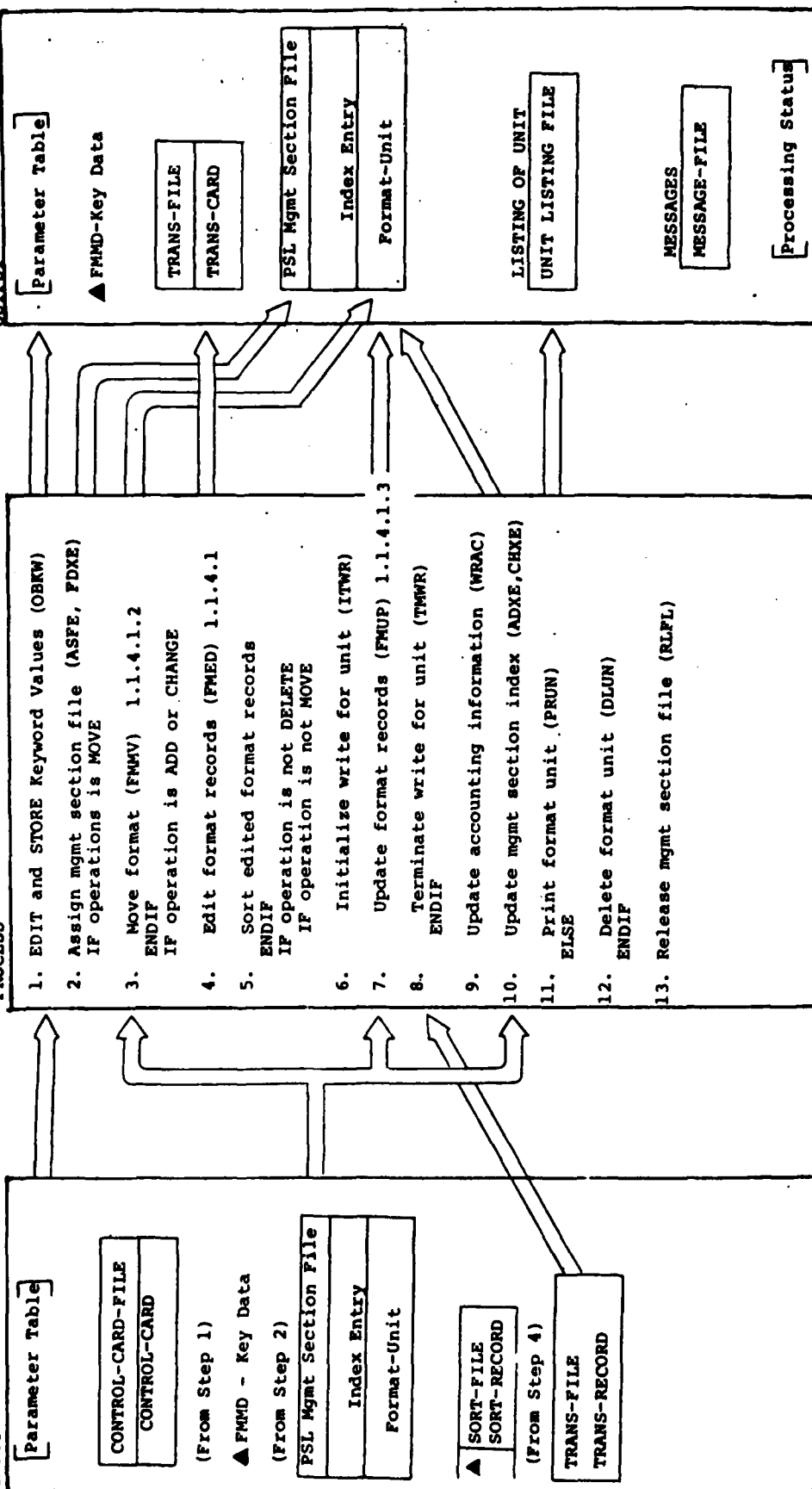
Name: FMMD - Update Management Data Formats

Description: Management Data

INPUT

PROCESS

OUTPUT



05 FMMD-MOD PIC X(3).
 - modification number specified by user and checked against accounting information.

05 FMMD-OPCODE PIC X(6).
 - update operation to be performed (ADD, CHANGE, MOVE, DELETE).

05 FMMD-LEVEL.
 - level name either given by user or obtained from accounting record.

10 FMMD-LEV-NBR PIC 9(1).
 - corresponds to occurrence in FMMD-MDCR-UNIT-NAME data group.

10 FMMD-LEV-NAME PIC 9(6).
 - values are "SYSTEM, UNIT, MODULE, SUBSYS, JOB".

05 FMMD-CYCLE PIC 9(2).
 - value from accounting information calculated when archiving.

05 FMMD-OLDPROJ PIC X(12).
 - project of sending format unit in MOVE operation.

05 FMMD-OLDLIB PIC X(7).
 - library of sending format unit in MOVE operation.

05 FMMD-ARCHIVE PIC X(1).
 - indicate collection (MDCOLLECT) processing will archive data.

05 SEQ-CTR PIC S9(9) COMP.
 - number of edited format records to be sorted.

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2,19,32	ERR	Subsequent processing bypassed	
2	6	ADV	File is not assigned Subsequent processing bypassed	1
	31	PSL	Subsequent processing bypassed	
	3	ERR	If operation is ADD or MOVE, subsequent processing bypassed	
3	See Branching and Error Conditions for module FMMVE 1.1.4.1.2.			
4	See Branching and Error Conditions for module FMEDE 1.1.4.1.1.			
6	43	PSL	Subsequent processing bypassed	
7	See Branching and Error Conditions for module FMUPE 1.1.4.1.3.			
8	45	PSL	Processing continues	2
9	46	PSL	Processing continues	3
10	24,29	PSL	Index entry is not updated. Processing continues	
11	66	ERR	Unit not printed Processing continues	4
12	9	ERR	Unit not DELETED Processing continues	
13	37	PSL	File is not released. Error return from DAFL may have unpredictable result.	

NOTES:

- (1) Unable to assign a file. See description of ASFL for detailed list of causes for this condition.
- (2) Unable to terminate write for unit. See description of WRUN for detailed list of causes for this condition.
- (3) Unable to write accounting information for unit. See description of WRAC for detailed list of causes for this condition.
- (4) Error while printing unit. See description of PRUN for detailed list of causes for this condition.

2.2.1.1.4.1.1 FMED - Edit Format Records

The FMED module edits user-input format records prerequisite to updating the designated format unit.

a. Program Operations

HIPO diagram 1.1.4.1.1 depicts the program operations of the FMED module. In step 2, the user data immediately follows the ** MDFORMAT card of the "ADD" operation or the appropriate subfunction cards when the operation is "CHANGE". In step 3, the user data which is stored in TEMP-DATA-FILE is edited field by field. If an error is detected in any field, a flag is set in the data group "TRANSACTION-ERROR-FLAGS". In step 4, edited fields of the TEMP-DATA-FILE record is moved to the corresponding fields of the TRANS-DATA record.

b. Data File and Table Descriptions

01 SECONDARY-CP-CODE-EDIT-TABLE.

The SECONDARY-OP-CODE-EDIT-TABLE is used to evaluate the SECONDARY-OP-CODE of the TEMP-DATA-FILE record. The table associates the SECONDARY-OP-CODE with the appropriate level(s) which may be referenced or processed.

05	FILLER	PIC X(6) VALUE " 11110".
05	FILLER	PIC X(6) VALUE "*11110".
05	FILLER	PIC X(6) VALUE "S10000".
05	FILLER	PIC X(6) VALUE "M11000".
05	FILLER	PIC X(6) VALUE "J11000".
05	FILLER	PIC X(6) VALUE "A00101".
05	FILLER	PIC X(6) VALUE "\$11110".

01 FILLER REDEFINES SECONDARY-OP-CODE-EDIT-TABLE.

05	SEC-OP-CODE-ENTRIES	OCCURS 7 TIMES
		INDEXED BY SOC-INDEX.
10	SEC-OP-CODE	PIC X(1).
		OCCURS 5 TIMES
		INDEXED BY LV-INDEX.
15	LEVEL-PERMISSIONS	PIC X(1).
38	SEC-OP-CODE-VALID	VALUE 1.

01 TEMP-DATA-FILE.

The TEMP-DATA-FILE record is the format of the user input to the ** MDFORMAT function. Each field is edited and/or validated before being moved to the corresponding field in the TRANS-DATA record.

05 PRIMARY-OP-CODE PIC X(1).
 - indicates the operation to be performed on input
 (I-INSERT, R-REPLACE, D-DELETE).

05 PRIM-ITEM-NBR PIC X(3).
 - item number indicating a management data collection
 item.

05 SECONDARY-OP-CODE PIC X(1).
 - indicates a reference to another management data
 collection item.

05 SECONDARY-ITEM-NBR PIC X(3).
 - item number referenced by PRIM-ITEM-NBR or
 indicator for repeated items.

05 SPECIAL-OP-ID.
 - indicates data value edit specification or collection
 function processing (TTL-TOTAL, MAX-MAXIMUM, AVG-
 AVERAGE, CYC-CYCLE).

10 EDIT-CODE PIC X(1).
 - indicates edit for data value (N-NUMBER,
 A-ALPHABETIC, B-ALPHANUMERIC).

10 EDIT-OPERAND PIC X(2).
 - indicates length of data value.

05 PRIM-ITEM-NAME PIC X(12).
 - corresponds to PRIM-ITEM-NBR.

05 FILLER PIC X(1).

05 PRIM-ITEM-LABEL PIC X(48).
 - stores text used as output descriptive label.

05 FILLER PIC X(8).

01 TRANSACTION-ERROR-FLAGS.

The TRANSACTION-ERROR-FLAGS data group stores flags which indicate whether or not specific fields in the TEMP-DATA-FILE record were edited successfully.

05	POC-FLAG	PIC X(1).
	- indicate result of edit on the PRIMARY-OP-CODE.	
05	PIN-FLAG	PIC X(1).
	- indicate result of edit on the PRIM-ITEM-NBR.	
05	FILLER	PIC X(2).
05	SOC-FLAG	PIC X(1).
	- indicate result of edit and table search on the SECONDARY-ITEM-NBR.	
05	FILLER	PIC X(2).
05	SOP-FLAG	PIC X(1).
	- indicate result of edit on SPECIAL-OP-ID.	
05	EDO-FLAG	PIC X(1).
	- indicate result of edits on related fields EDIT-CODE and EDIT-OPERAND.	
05	FILLER	PIC X(1).

NOTE: See module FMMDE for description of input/output arguments.

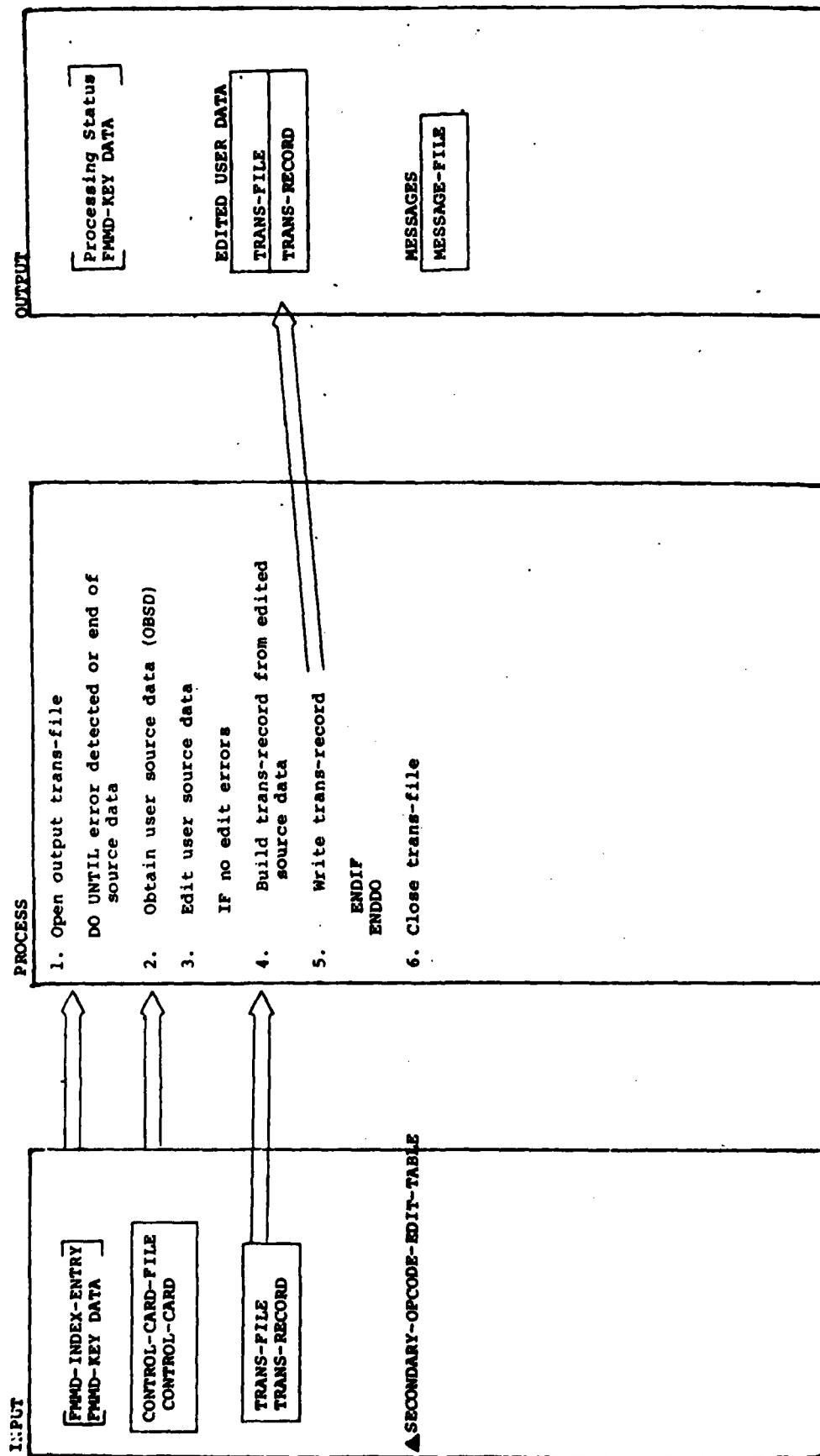
c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
2	14	INF	End of user data Processing returns control to calling routine.	
3	190	ERR	Source data rejected Processing continues	

Diagram ID: 1.1.4.1.1

Name: FMED - Edit Format Records

Description: Management Data



2.2.1.1.4.1.2 FMMV - Move Format Records

The FMMV module moves format unit records from one designated project library to another.

a. Program Operations

HIPO diagram 1.1.4.1.2 depicts the program operations of the FMMV module. In step 1, the specified Management section file is assigned. In step 2, if the sending unit is not located, the move processing will terminate. In step 9, the sending Management section file is released.

b. Data File and Table Descriptions

- TRANS-FILE see description of file in module FMMD.
- RECEIVING-OPTIONS.
 - section options of receiving management section.
 see description of "SECTION-OPTIONS" in Section 2.
- FMMD-KEYDATA
 - see description of module FMMD for description of item.
- NEW-STRING PIC X(80).
 - stores the catalog file string for the receiving format unit.
- NBR-LINES PIC S9(9) COMP.
 - stores the number of lines written to the new format unit.
 - used to update accounting information of receiving unit.

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	6	PSL	Subsequent processing bypassed	1
2	31	PSL	Subsequent processing bypassed	
	26	ERR	Subsequent processing bypassed	

Diagram ID: 1.1.4.1.2

Name: FMV - Move Format

Description: Management Data

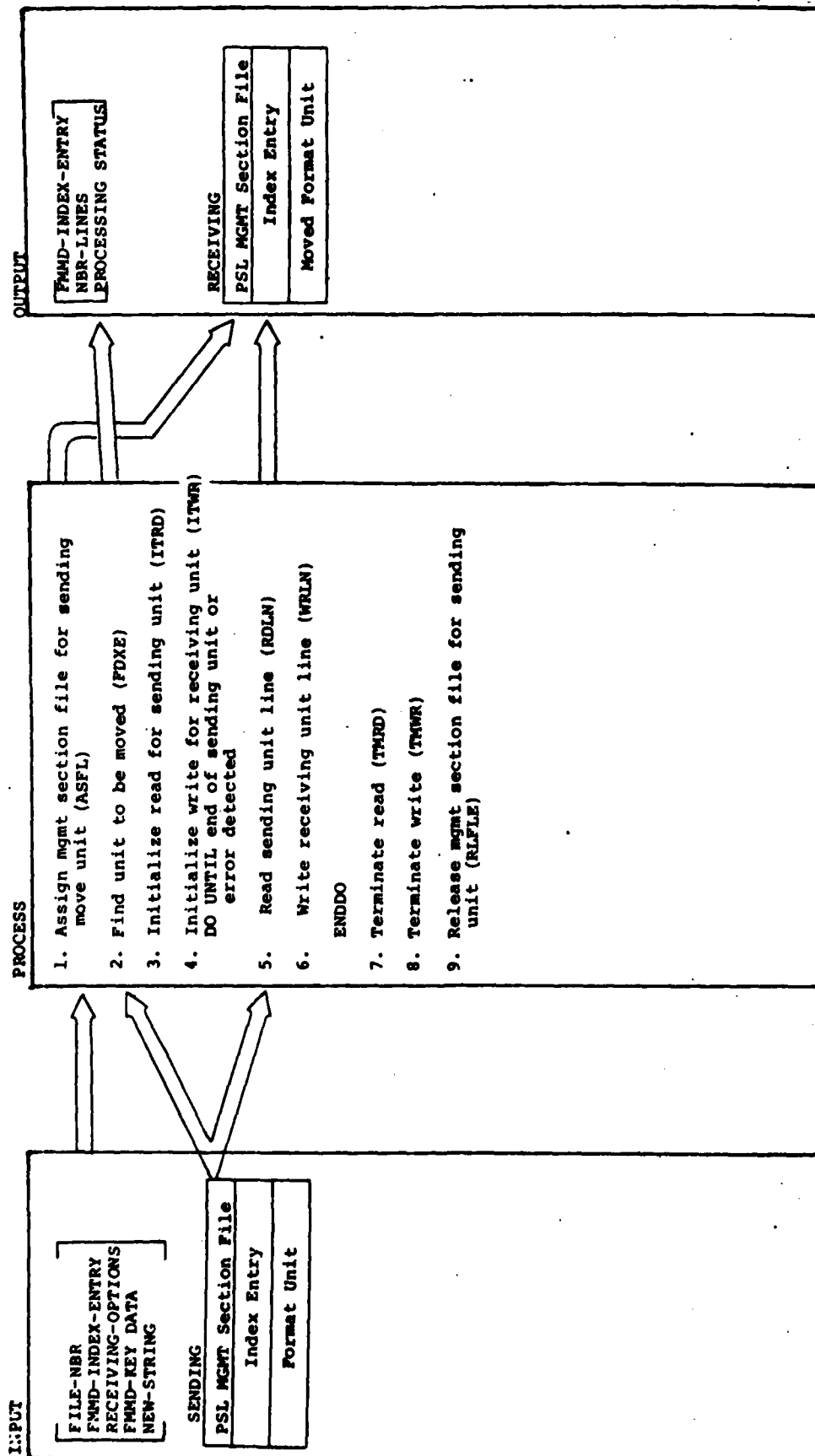


Diagram ID: 1.1.4.1.3

Name: FMUP - Update Format Records

Description: Management Data

INPUT

FMS-CATALOG-STRING
FMD-INDEX-ENTRY
SECTION-OPTIONS
FILE-NBP
OLD-BLOCK-NBR
COUNTING-INFORMATION
FMD-KEY DATA

TRANS-FILE
TRANS-RECORD

PSL MGMT Section File
Index Entry
Format Unit

PROCESS

1. Open input trans-file
IF operation is ADD
DO UNTIL end of trans-file or
error detected
2. Read trans-file
3. Write new unit line using
trans-record (WRLN)
ENDDO
ELSE
4. Initialize read for existing unit
5. Read trans-file
6. Read unit line (RDLN)
DO while no error detected
7. Update unit lines 1.1.4.1.3.1
ENDDO
IF end of unit
DO while not end of trans-file
and insert records
Write new unit line (WRLN)
using trans-record
Read next trans-record
ENDDO
ENDIF
ENDIF
10. Close trans-file

OUTPUT

NBR-LINES
ACCOUNTING-INFORMATION
PROCESSING STATUS

PSL MGMT Section File
Index Entry
New Format Unit

MESSAGES
MESSAGE-FILE

c. Branching and Error Conditions (continued)

Function Reference	Condition Code	Message Category	Program Action	Note
3,5,7	47,48,49	PSL	Processing continues	2
4,6,8	43,44,45	PSL	Unit may be truncated Processing continues	3
9	37	PSL	File is not released. Error return from DAFL have unpredictable result.	

NOTES:

- (1) Unable to assign file. See description of ASFL for detailed list of causes for this condition.
- (2) Unable to perform read processing. See description of RDUN for detailed list of causes for these conditions.
- (3) Unable to perform write processing. See description of WRUN for detailed list of causes for these conditions.

2.2.1.1.4.1.3 FMUP - Update Format Records

The FMUP module updates a designated format unit with edited input format record transactions.

a. Program Operations

HIPO diagrams 1.1.4.1.3 and 1.1.4.1.3.1 depict the program operations of the FMUP module. In step 3, each record of the TRANS-FILE is written to the space assigned for the new format unit being added. In step 4, a check is made of the UNIT-KEY value and the MODIFICATION number before update begins. In step 7, the HIPO diagram 1.1.4.1.3.1 details the processing of updating an existing format unit. The updating process is mainly dependent on two factors:

- (1) the correlation between the PRIMARY-ITEM-NBR of the existing unit and the TRANS-ITEM-NBR of the TRANS-FILE, and
- (2) the value of TRANS-OP-CODE of the TRANS-FILE.

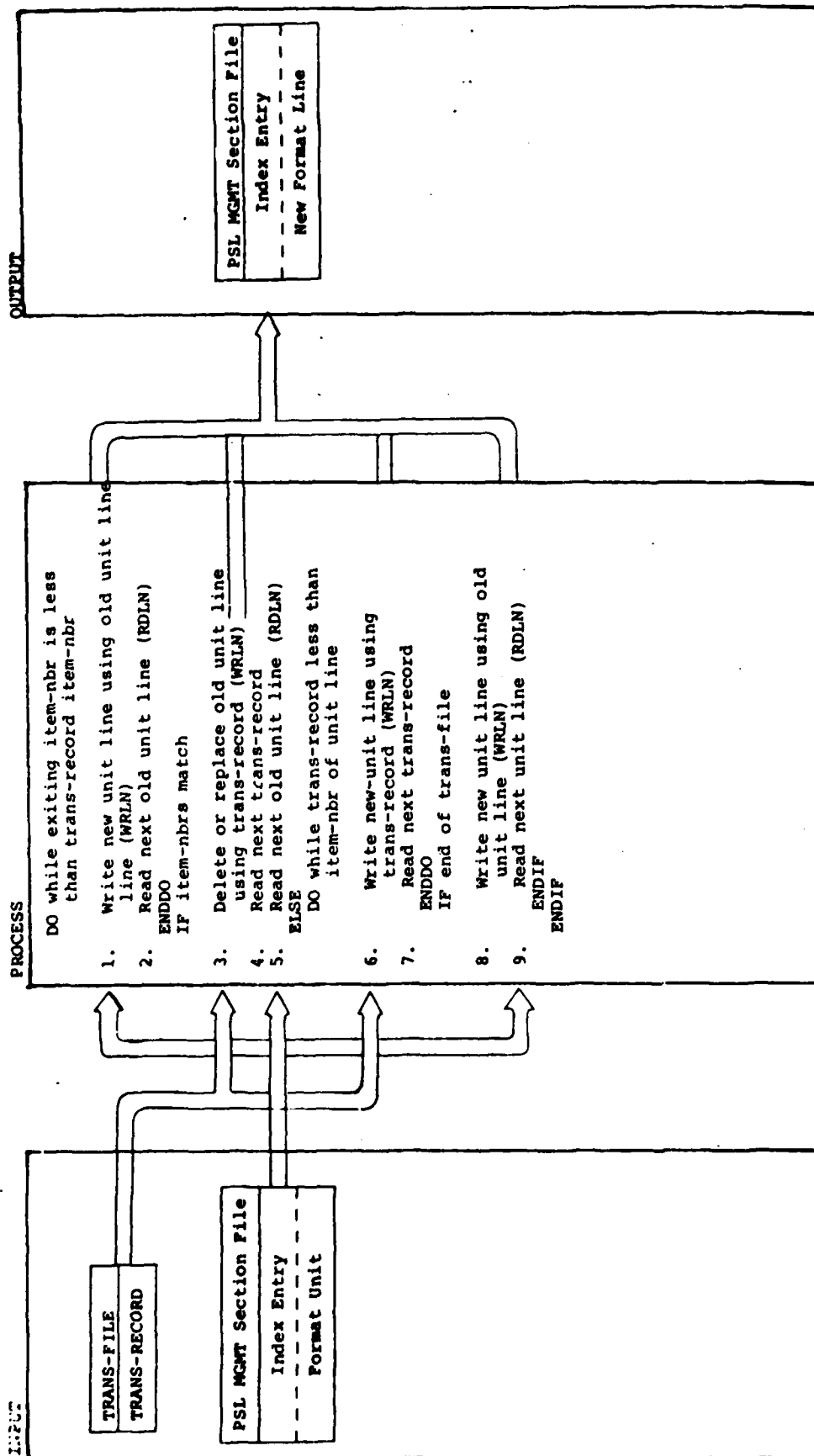
b. Data File and Table Description

- FMS-CATALOG-STRING
 - see Section 3 for description of "FMS-CATALOG-FILE-STRING".
- FMMD-INDEX-ENTRY
 - see data description of module FMMD.
- OLD-BLOCK-NBR PIC S9(9) COMP.
 - block number assigned to existing format unit used in initializing the format unit to be read.
- ACCOUNTING-INFORMATION
 - the accounting information of the existing format will be modified to reflex updating of the unit.
 - see Section 3 for description of "ACCOUNTING-INFORMATION".
- NBR-LINES PIC S9(9) COMP.
 - number of lines written to new/updated unit.

Diagram ID: 1.1.4.1.3.1

Name: FMUP - Update Unit Lines

Description: Management Data



c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
3	44	PSL	Subsequent processing bypassed	1
4	47	PSL	Subsequent processing bypassed	2
6	48	PSL	Subsequent processing bypassed	2
7	44,48	PSL	Unit is truncated Subsequent processing bypassed	1,2
	58,161	ERR	Record not updated Processing continues	

NOTES:

- (1) Unable to perform write processing. See description of WRUN for detailed list of causes for these conditions.
- (2) Unable to perform read processing. See description of RDUN for detailed list of causes for these conditions.

2.2.1.1.4.2 UPMD - Update Manual Management Data

The module UPMD, which corresponds to the ** MDUPDATE function, originally introduces data into a management data unit, modify the contents of a management data unit, or delete a management data unit. The module UPMD also corresponds to the ** MDXCHECK function which is used to add, change or delete exception checking specifications within an existing management input unit. Modification details are provided by the operation-code and the Subfunction cards (DELETE, MODIFY, and INSERT) which follow either the MDUPDATE or MDXCHECK card. New management data follows either the MDUPDATE card for the ADD operation or the INSERT and MODIFY Subfunction cards for the CHANGE operation. New and revised exception checking specifications follow the INSERT and MODIFY Subfunction cards, respectively. Identification of either exception checking specification or management data items to be deleted follow the DELETE Subfunction card. The module UPMD calls two subroutines, the module UPMF and the module UPMR, that are responsible for the detailed MDUPDATE/MDXCHECK processing. The module UPMF validates and edits all Subfunction cards, verifies appropriate formats units, builds formatted input records and stores them on a temporary file. The temporary file is sorted, then, used by the module UPMR to add, update, or delete, either management input data or exception checking data.

a. Program Operations

HIPO diagram 1.1.4.2 describes the operation performed. In step 1, the subroutine UPMF 1.1.4.2.1 is called to generate the UPMD-FILE file. The module UPMF also verified management data format units and items contained therein with input data being edited. The module UPMF returned important function values necessary for further processing. In step 3, the subroutine module UPMR is called to update the management data input unit. The sorted temporary file UPMD-FILE is used as input to the module UPMR.

b. Data File and Table Description

1. INPUT ARGUMENTS

Parameter-Table see Section 3 for description

2. OUTPUT ARGUMENTS

Processing-Status	PIC S9(9) COMP.
Return-code	- zero for normal status.
	- error code indicating cause of failure of file assignment. See section 2.2.1.1.4.2.c for complete list of codes.
Parameter-Table	see Section 3 for description

3. SORT-FILE

This file is used as the collation file for the 1100 Series Executive Sort/Merge Program. The CLIB-MDCR-DATA-RECORD-FORMAT is identical to the sort record and the UPMD-RECORD. The sort key is comprised of the DATA-TYPE-CODE, DATA-RECORD-ID, DATA-INPUT-SEC and the EDIT-CHECK-TYPE.

01 SORT-RECORD.

05 FILLER	PIC X(5).
	- corresponds to the first five characters of the UPMD-RECORD.
05 SORT-FLDS	PIC X(12).
	- represents the sort key described above.
05 FILLER	PIC X(63).
	- remainder of the UPMD-RECORD.

4. UPMD-FILE

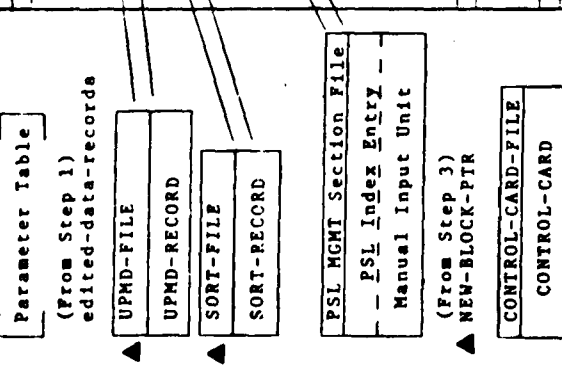
This file is used to store the modification detail along with the line date specified for processing by the user.

01 UPMD-RECORD	PIC X(80).
01 CLIB-MDCR-DATA-RECORD-FORMAT	see Section 3 for description
	- data grouping of UPMD-RECORD.

Diagram ID: 1.1.4.2

Name: UPMDE - Update Manual Management Data Description: Management Data

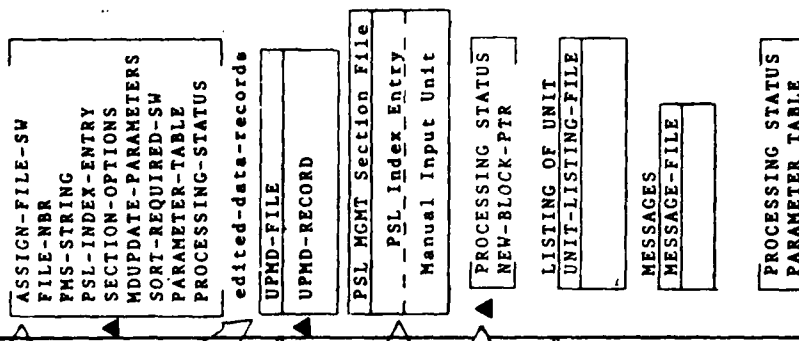
INPUT



PROCESS

1. Edit manual data records UPMFE 1.1.4.2.1
IF user supplied data
2. Sort edited data records by record ID
ENDIF
3. Update management data file records
UPMFE 1.1.4.2.2
IF error detected
4. Release used space (RLBK)
5. Read through remaining data cards (OBSD)
ELSE
6. Print manual input unit (PRUN)
ENDIF
7. Release PSL mgmt section file (RLPL)

OUTPUT



c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	19,32,3,100, 3,161,4,192, 194,185,196,9	ERR	Subsequent processing continues	1
1	47,49,6	PSL	Subsequent processing continues	1
3	47,49,43,44, 41,48	PSL	Subsequent processing continues	2
4	42	PSL	Used blocks not released	
6	66	ERR	Management unit not listed	
7	37	PSL	section file is not released	

Notes:

- (1) See description of module UPMF for specific conditions which cause error conditions.
- (2) See description of module UPMR for specific conditions which cause error conditions.

2.2.1.1.4.2.1 UPMF - Edit Data Format

The UPMF module edits the user input transactions for compliance with format requirements.

a. Program Operations

HIPO diagrams 1.1.4.2.1 and 1.1.4.2.1.1 depicts the program operations of the UPMF module. In step 3, the index block of the management section is searched for the management unit. If the unit is not in the management index and the operation specified for the update is "ADD", later processing will attempt to add the entry to management index. In step 4, the unit level name is obtained from the accounting information of the stored management unit. In step 5, either the level value specified on the add operation or the unit level name obtained from step 4 is used to determine the appropriate format. If the appropriate format is not located, processing is terminated. In step 8, the input records are edited and reformatted according to the PSL function being processed (MDUPDATE/MDXCHECK).

b. Data File and Table Descriptions

• UPMD-FILE

The UPMD-FILE is used to store the formatted update records to be read and processed by the UPMRE module.

01 DATA-RECORD

The DATA-RECORD group defines the format of updating records. See Section 3, DATA group "CLIB-MDCR-DATA-RECORD-FORMAT" definition.

1. INPUT ARGUMENTS

• PARAMETER-TABLE

See Section 3 for description of the data items of the Parameter Table.

2. OUTPUT ARGUMENTS

• ASSIGN-FILE-SW

PIC S9(9) COMP.

- switch to indicate whether section file was assigned.

- FILE-NBR PIC S9(9) COMP.
 - number of file code, with the allocation table to which file has been assigned.
- FMS-STRING
 - see Section 3, DATA group "FMS-CATALOG-FILE-STRING" definition.
- UPMD-INDEX-ENTRY
 - see Section 3, DATA group "REQUESTED-INDEX-ENTRY" definition.
- SECTION-OPTIONS
 - see Section 3 for description of data items.
- 01 MDUPDATE-PARAMETERS

The MDUPDATE-PARAMETER data group stores data values used in determining processing by the module UPMRE.

 - 05 MD-LEVEL PIC X(6).
 - level of management unit (SYSTEM, SUBSYSTEM, MODULE, ETC.)
 - 05 MD-UNIT-KEY PIC X(12).
 - key assigned when unit was added to PSL storage.
 - 05 OPERATION-NAME PIC X(6).
 - operation to be performed by functions (ADD, CHANGE, DELETE)
- 77 SORT-REQUIRED-SW PIC S9(9) COMP.
 - switch to indicate whether 1100 Series Executive Sort/Merge Program will be invoked.
- 01 PROCESSING-STATUS PIC S9(9) COMP.

Return Code - zero for normal status
 - error code indicating cause of failure

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2,19,32	ERR	Subsequent processing bypassed	
2	41	PSL	Mgmt section file not assigned Subsequent processing bypassed	1
	6	ADV		1
3	3,9,26,100	ERR	Subsequent processing bypassed	
	31,43,47	PSL	Subsequent processing bypassed	
4,5	23,26	ERR	Subsequent processing bypassed	
	31,41,47	PSL	Subsequent processing bypassed	1
6	11 48	ERR PSL	Processing continues	2
8,10	1	ERR	Skip to next function Processing continues	
9,11	19,64,65 193,195,196	ERR	Processing continues	

NOTES:

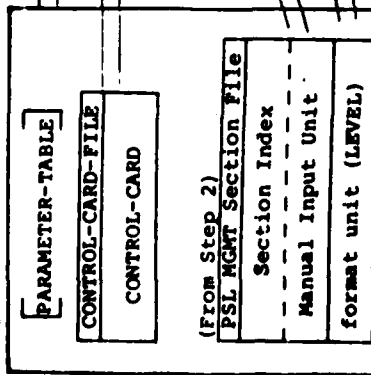
- (1) Unable to assign required library-section file. See description of ASFL for conditions which cause file assignment to fail.
- (2) Unable to initialize to read line or to read line. See description of RDUN for conditions which cause read or read initialization to fail.

Diagram ID: 1.1.4.2.1

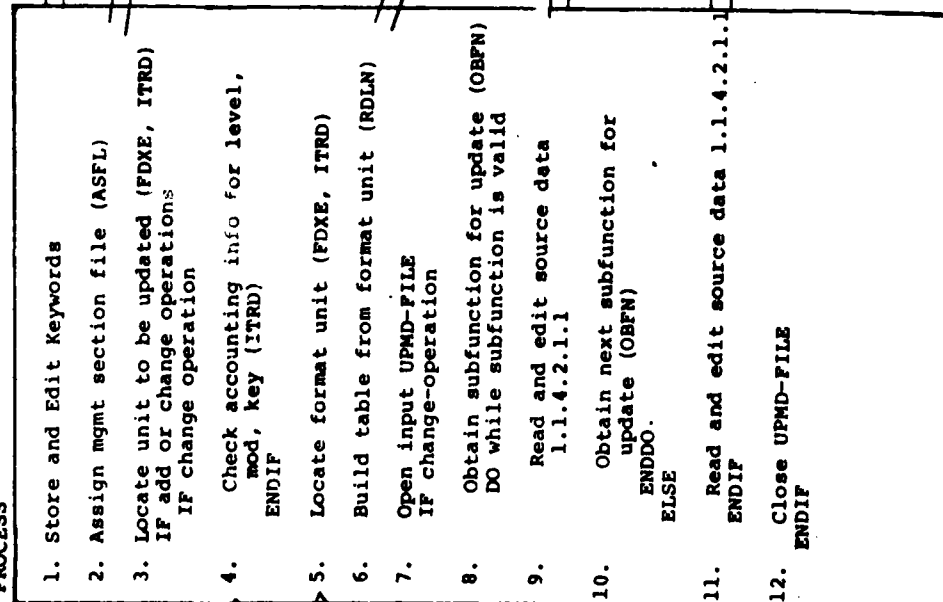
Name: UPMFE - Edit Manual Data Records

Description: Management Data

INPUT



PROCESS



OUTPUT

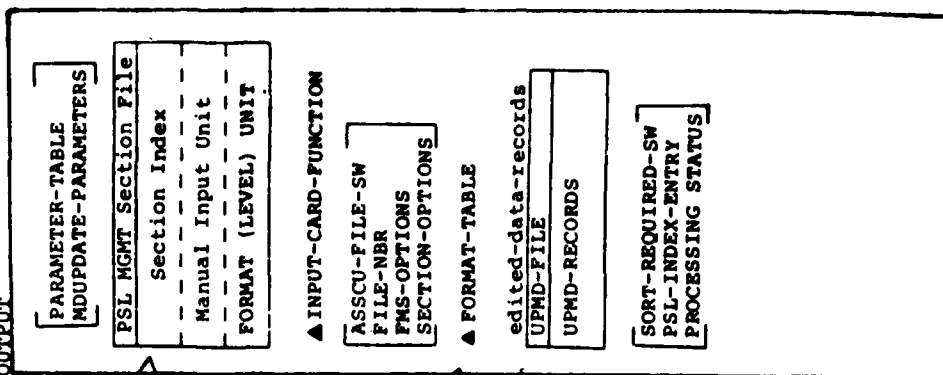
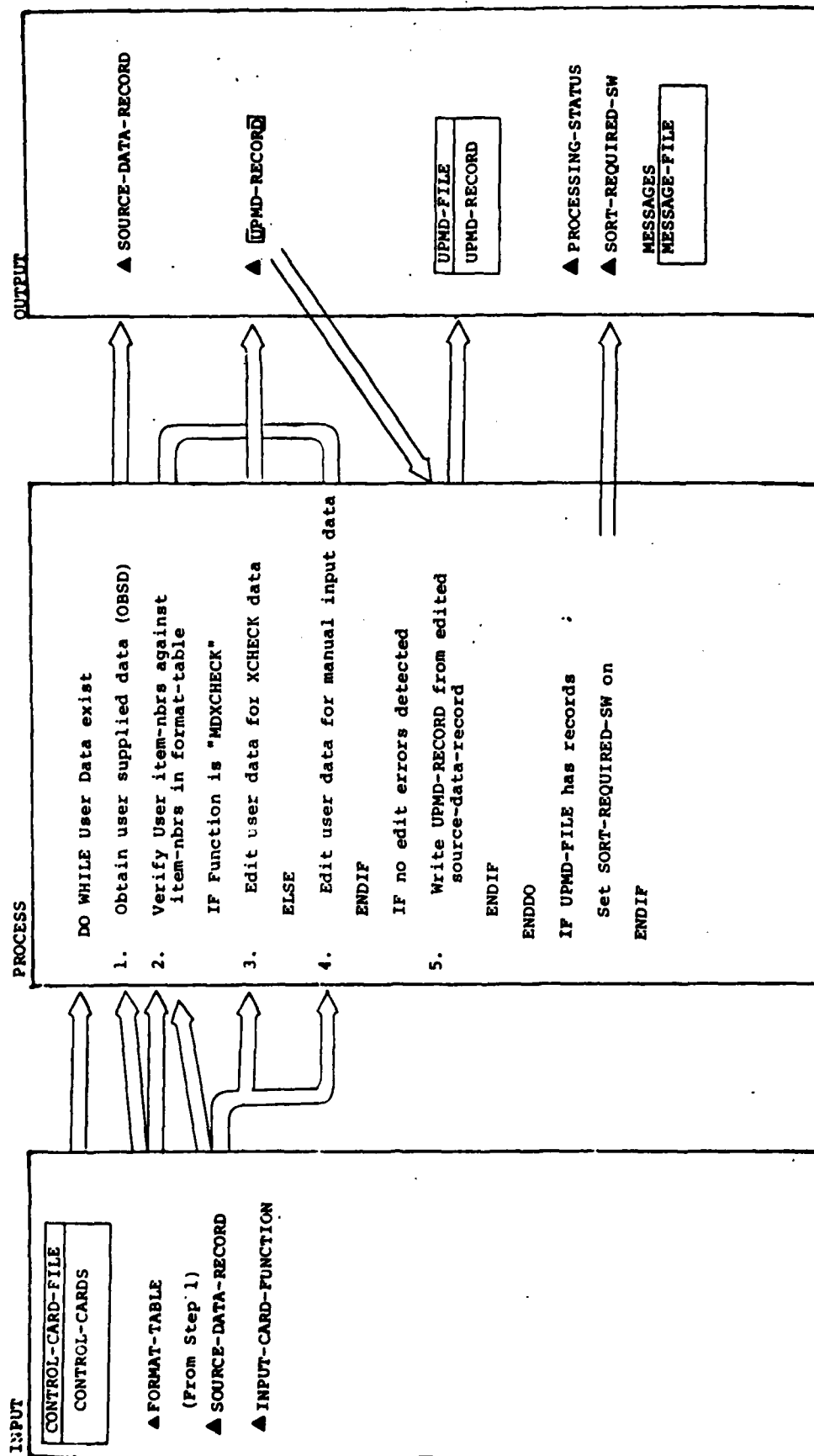


Diagram ID: 1.1.4.2.1.1

Name: Read and Edit Source Data

Description: Management Data



2.2.1.1.4.2.2 UPMR - Update Management Data File Records

The UPMR module updates a management input unit with edited user input transactions.

a. Program Operations

HIPO diagram 1.1.4.2.2 depicts the program operations of the UPMR module. In step 3, each new UPMD-RECORD is read from the UPMD-FILE and edited for repeated items or XCHECK data. In step 6, record-ids of the UPMD-RECORD and existing management record (DATA-RECORD) are compared. If the record-id of the DATA-RECORD is lower than that of the UPMD-RECORD, the DATA-RECORD is written to the new unit. If the record-id of the DATA-RECORD is equal to that of the UPMD-RECORD, the DATA-RECORD is either deleted or updated and written to the new unit. If the record-id of the DATA-RECORD is greater than that of the UPMD-RECORD, then the UPMD-RECORD is inserted and written to the new unit.

b. Data File and Table Descriptions¹

• UPMD-FILE

The UPMD file is used to store the formatted update records needed for updating a manual input unit. See Section 3 DATA group "CLIB-MDCR-DATA-RECORD-FORMAT" definition.

• NEW-DATA-BLOCK

PIC S9(9) COMP.

The NEW-DATA-BLOCK is the address of the first block for the new or updated unit.

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	43	PSL	Subsequent processing bypassed	1
3	44	PSL	Unable to write management records	1
4	24	PSL	Unit not added to index	2

¹ See Data File and Table Description for module UPMDE 1.1.4.2 and UPMFE 1.1.4.2.1.

c. Branching and Error Conditions (continued)

Function Reference	Condition Code	Message Category	Program Action	Note
5	47,48 44,49,45 29	PSL	Processing terminates passing control to UPMD module	1,3,4
	161	ERR	Specific data rejected Processing continues	

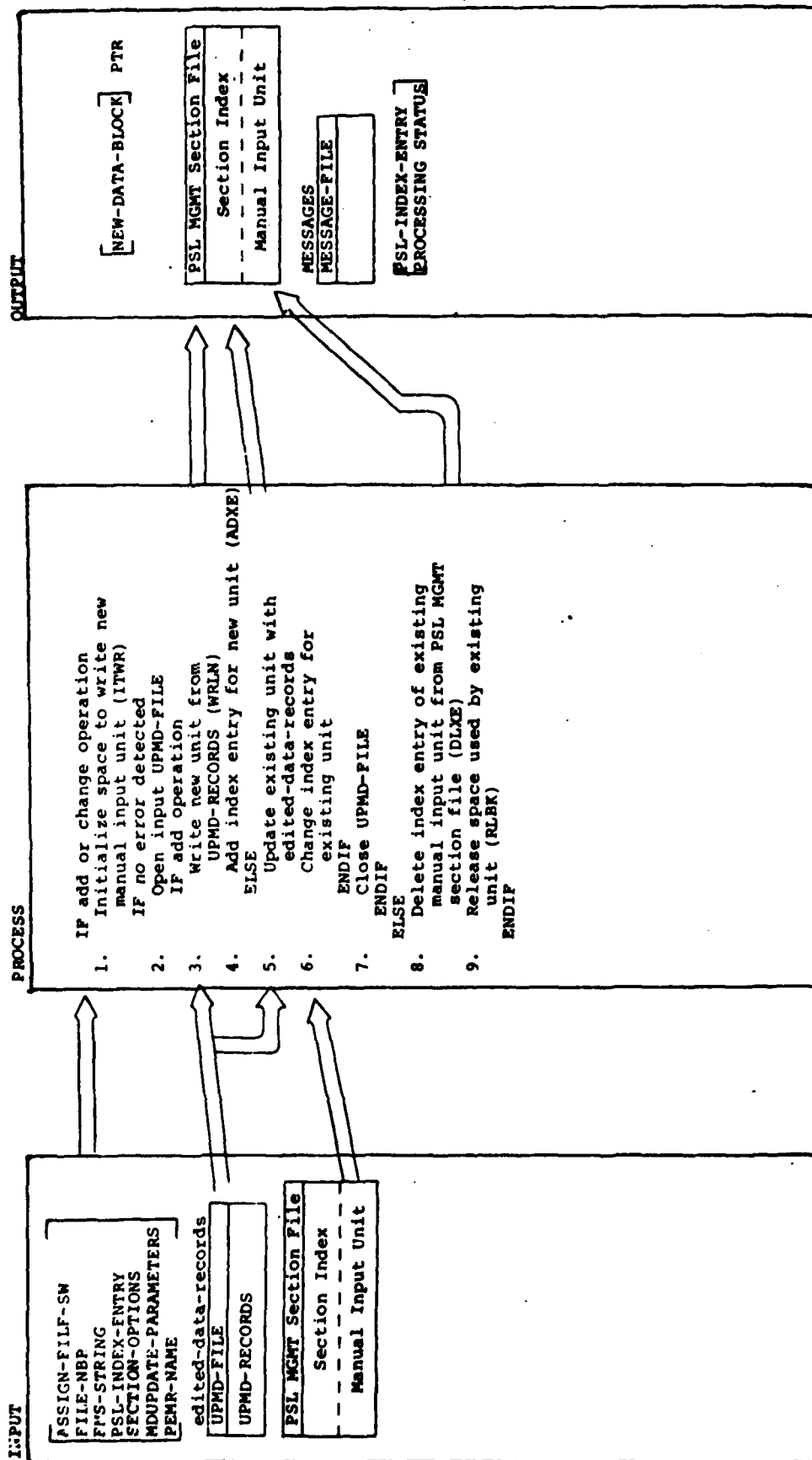
NOTES:

- (1) Unable to initialize write or write line. See description of WRUN for conditions which cause failure.
- (2) Unable to add an entry to the PSL index block. See description of ADXE for conditions which cause failure.
- (3) Unable to initialize read or read line. See description of RDUN for conditions which cause failure.
- (4) Unable to change index entry in the PSL index block. See description of CHXE for conditions which cause failure.

Diagram In: 1.1.4.2.2

Name: UPMDF - Update Management Data File Records

Description: Management Data



2.2.1.1.4.3 ITCL - Initiate Collection

This program module initiates the collection and storage of management data in the management section of a specified PSL project and library in response to the use of the MDCOLLECT Function.

a. Program Operations

HIPO diagram 1.1.4.3 describes the operations required to initiate management data collection. Existence of the specified management section is verified and the collection unit key is checked when a collection unit exists and a key has been previously assigned. The spawned job procedure (e.g., default PROC=CLMD) is obtained from the job section of the system library and the spawned job file is written as indicated.

b. Data File and Table Descriptions

No special files or tables are utilized.

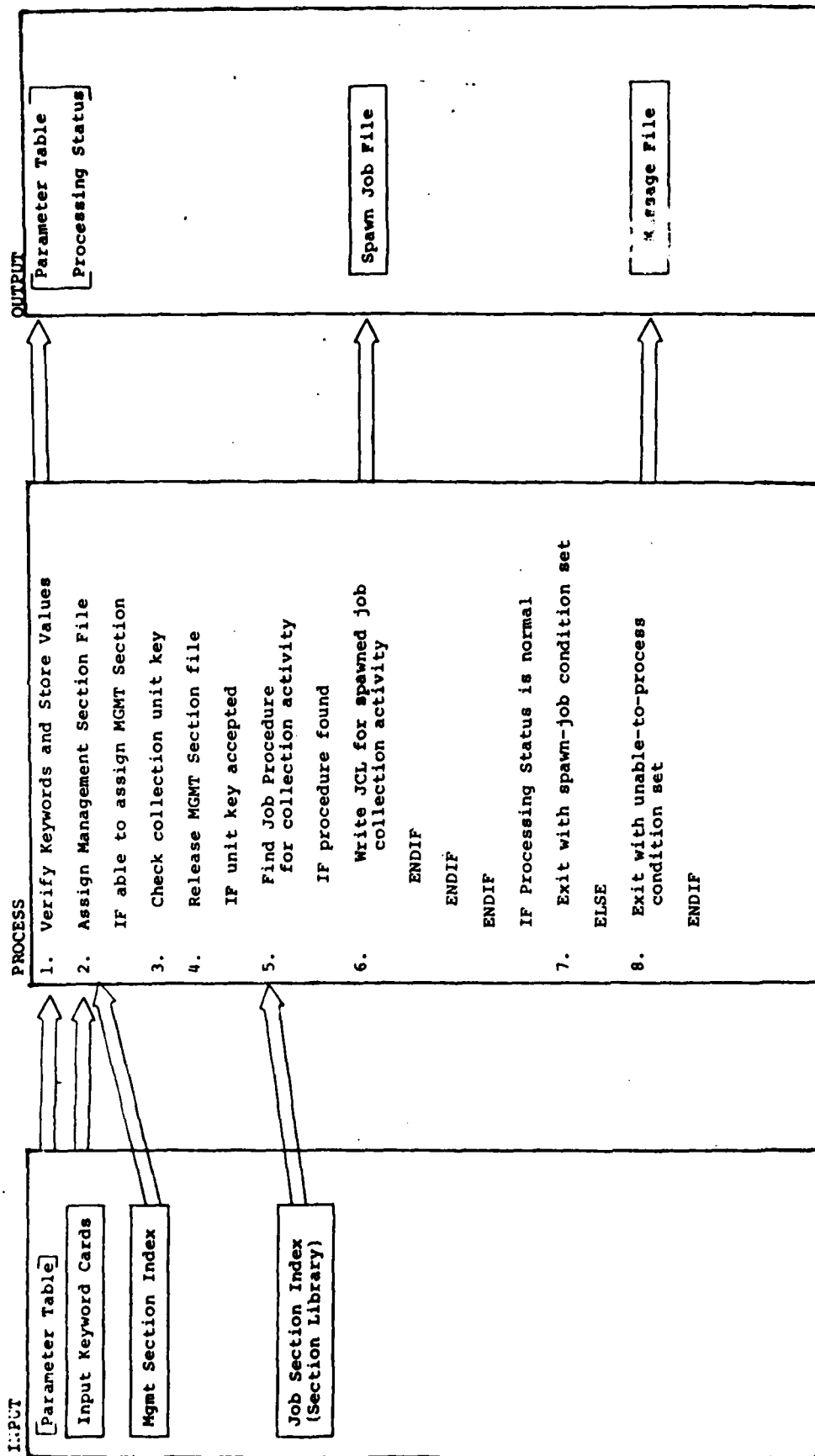
c. Branching and Error Conditions

The following table describes the branching and error conditions relative to HIPO diagram 1.1.4.3.

Function Reference	Condition Code	Message Category	Program Action	Note
1	2,19,32	ERR	Perform Process #8	
2	6	ERR	Perform Process #8	
3	9	ERR	Perform Process #8	
5	81,47	ERR	Perform Process #8	
6	18		Perform Process #7	
7	16		Normal exit	
8	7		Error exit	

Diagram ID: 1.1.4.3

Name: ITCL - Initialize Collection



2.2.1.1.4.3.1 CLMD - Collect Management Data

Management data is collected and stored under the control of the CLMD module which is executed via the spawned job procedure initiated by the ITCL module described in the preceding paragraph.

a. Program Operations

HIPO diagram 1.1.4.3.1 depicts the top-level operations that constitute the collection activity. Three subroutines are used to perform the detailed operations as indicated. The first subroutine (PPCL) edits keyword inputs and pre-processes the MDCR Plan unit to validate its syntax. The next subroutine (PCCL) performs the major work of collecting the data designated by the management plan and format units. The third routine (UPCL) performs any required archiving and stores the newly collected data.

b. Data File and Table Descriptions

1. Link Names Table

The following data is utilized to provide the link names of the three subroutines called by the CLMD module.

• 01 LINK-NAMES-FOR-CLMD.

05 LINK-PPCL	PIC X(6) VALUE "LKPPCL".
05 LINK-PCCL	PIC X(6) VALUE "LKPCCL".
05 LINK-UPCL	PIC X(6) VALUE "LKUPCL".

2. Collection Parameters

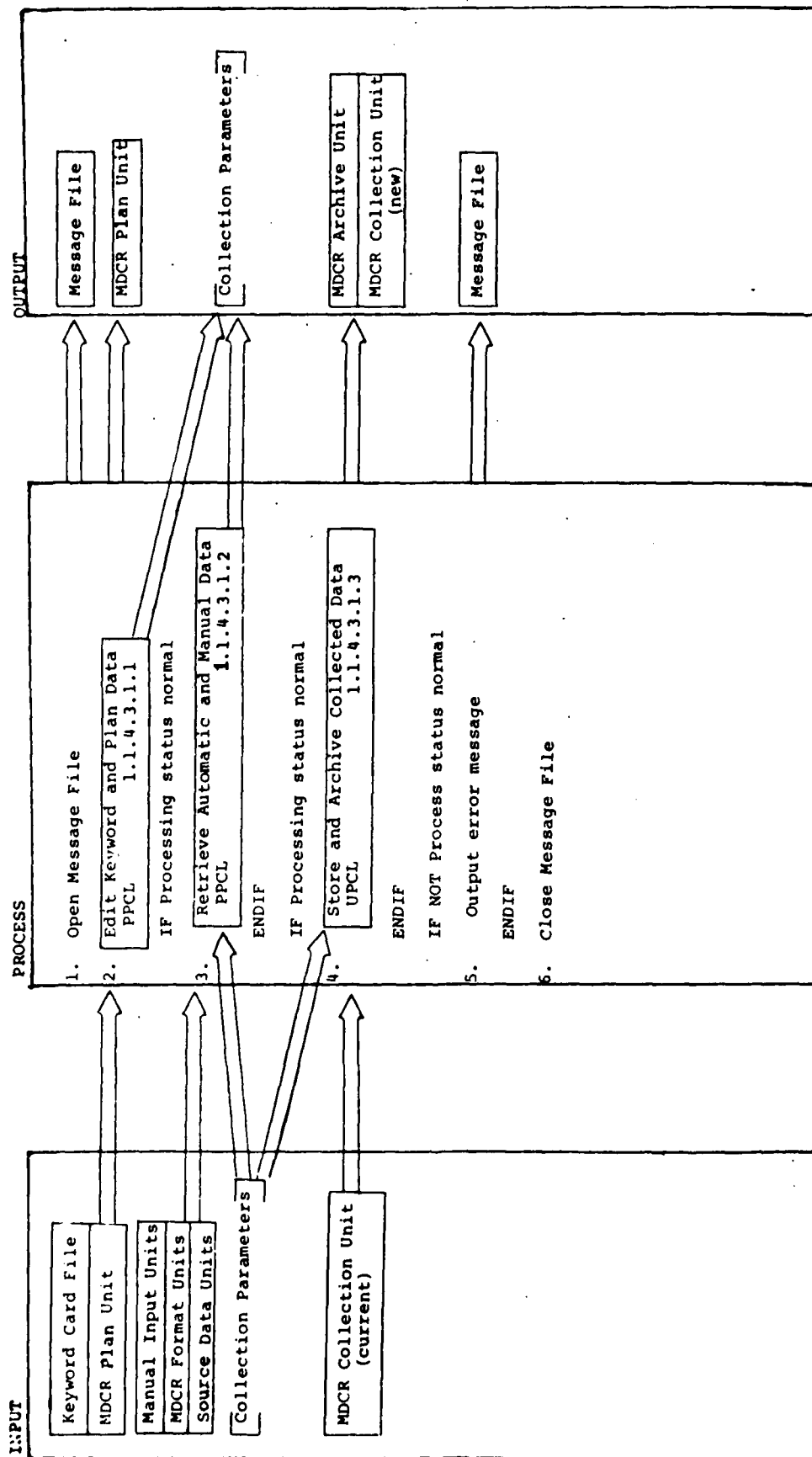
The following parameters are used to communicate information between the subroutines called by the main CLMD module:

• 01 COLLECTION-CONTROL-TABLE. (to be described)

77 ARCHIVE-VALUE	PIC X(3).
77 RECYCLE-VALUE	PIC X(3).
77 PROCESSING-STATUS	PIC S9(9) COMP.

Diagram ID: 1.1.4.3.1

Name: CLMD - Collect Management Data



The initial values for these parameters are determined by the PPCL subroutine. The ARCHIVE-VALUE and RECYCLE-VALUE entries are taken from the corresponding MDCOLLECT Function keyword input values (i.e., YES or NO). The PROCESSING-STATUS parameter is initially set to zero; it may subsequently be set to non-zero value indicating a condition for which CLMD processing is unable to be completed.

The Collection Control Table parameter is initially used to the following extent:

- 01 COLLECTION-CONTROL-TABLE.
 - 05 PROGRAMMER-NAME PIC X(12).
 - 05 MGMT-FMS-CATALOG-ENTRIES.
 - 10 MGMT-PROJECT-NAME PIC X(12).
 - 10 FILLER PIC X(12).
 - 10 MGMT-LIB-SEC-NAME.
 - 15 MGMT-LIB-SEC-CODE PIC X(1).
 - 15 MGMT-LIBRARY-NAME PIC X(7).
 - 10 FILLER PIC X(40).
 - 05 (remaining entries detailed under paragraph 2.2.1.1.4.3.1.2).

The PROGRAMMER-NAME and MGMT-FMS-CATALOG-ENTRIES are determined from PROGRAMMER, PROJECT and LIBRARY keyword input values. Assignment of the management section file is made using this data.

c. Branching and Error Conditions

Normal Exit: Processing status equals zero (0).
Error Exit: Processing Status equals seven (7).

2.2.1.1.4.3.1.1 PPCL - Pre-process Collection

The PPCL module performs operations which are preparatory to collecting management data.

a. Program Operations

HIPO diagram 1.1.4.3.1.1 depicts the processing performed by the PPCL subroutine. The keyword values originally verified by the ITCL module and output on the spawn job file are processed and stored in step 1. After assigning the indicated management section file, the MDCR Plan unit is processed to verify its syntax and to reformat and direct the plan keyword value input data to a temporary sequential file (i.e., Plan Keyword File). If no serious error is detected in the plan data, the MDCR Collection unit is read (provided that it exists) and where a correspondence is found between a collected element (i.e., system, subsystem, module or job name) and a plan element (a list of which elements is retained in a management data plan table generated in step 5), the numeric data items contained within that collected element are stored on a random file with a record key computed from the table entry number of the plan element which corresponds. This same entry number will later be used (in PCCL module operations) to retrieve data from the Random Collect File in computing cyclic data statistics. If, as an alternative to the above, the MDCR Plan unit contains improper keyword data (i.e., contains syntax errors), the plan input will be rejected and the Verification Status indicator in that unit's accounting record will be updated to indicate that rejection has been made. Unless the MDCR Plan unit is subsequently updated (using the MDPLAN Function), it remains in "rejected" status and is immediately rejected in subsequent collection operations prior to performing step 3.

b. Data File and Table Descriptions

1. Management Data Plan Table

This table is utilized as described above and has the following working storage definition:

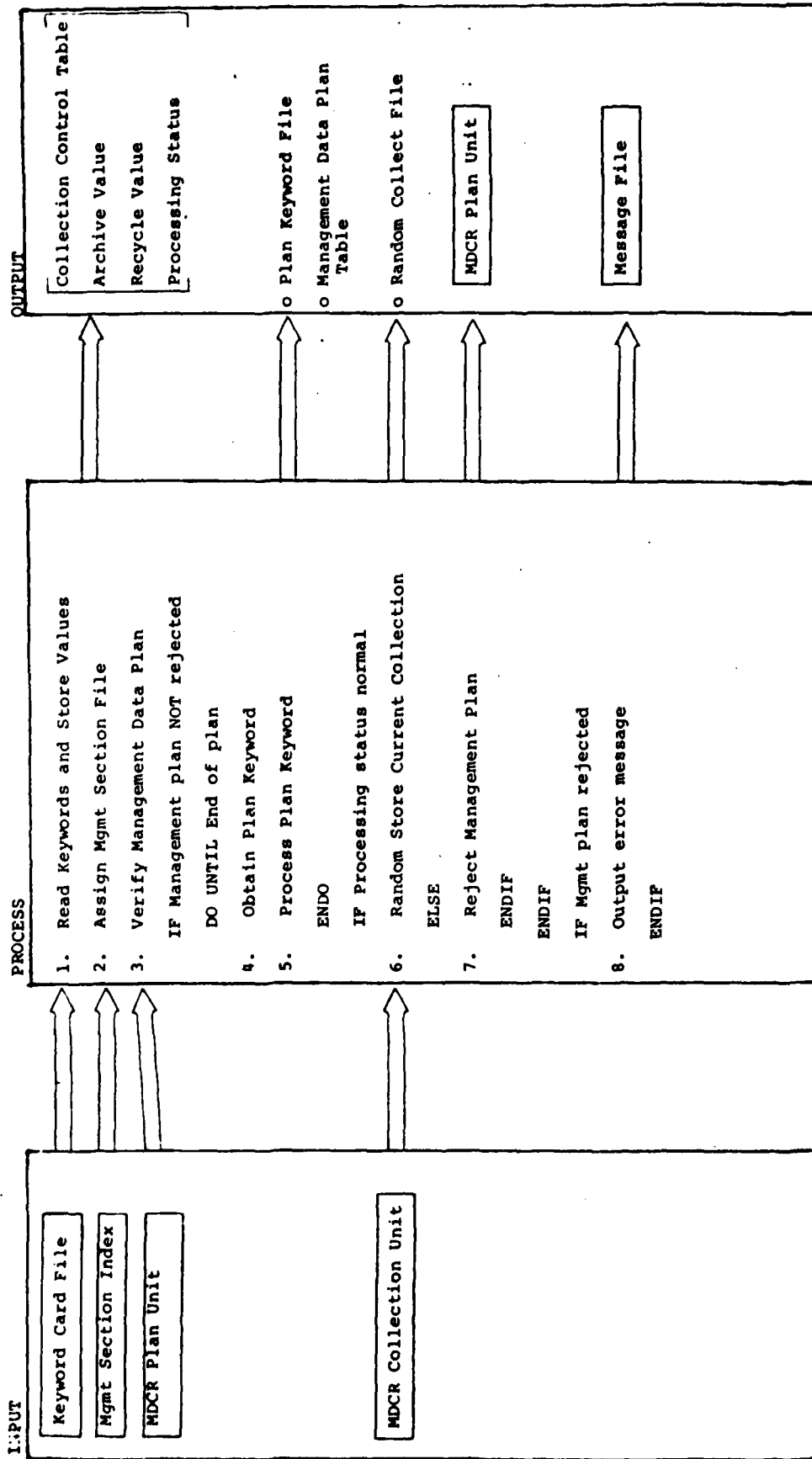
• 01 MD-PLAN-TABLE.

05	PLAN-UNIT-ENTRY	OCCURS 200 TIMES.
10	PLAN-UNIT-NAME	PIC X(30).
10	PLAN-UNIT-LEVEL	PIC 9(1).

Diagram ID: 1.1.4.3.1.1

Name: PPCL - Pre-process Collection

Description: Edit Keyword and Plan Data



A maximum of 200 unique plan unit (element) entries are permitted. Each plan element entry consists of the element name and level (i.e., system, subsystem, module or job) number (correspondingly being 1, 2, 3 or 4).

2. Plan Keyword File

The following data record description applies:

• 01 PLAN-KEYWORD-RECORD.

05	PRIME-KEYWORD	PIC X(12).
05	FILLER	PIC X(12).
05	UNIT-LEVEL-NBR	PIC S9(9) COMP.
05	KEYWORD-VALUE	PIC X(48).
05	UNIT-INPUT-NBR	PIC S9(9) COMP.
05	FILLER	PIC S9(9) COMP.
05	KEYWORD-CASE-NBR	PIC S9(9) COMP.

The PRIME-KEYWORD and KEYWORD-VALUE fields contain corresponding data as obtained from the MDCR Plan unit. The KEYWORD-CASE-NBR is determined by the contents of PRIME-KEYWORD and is later utilized (in PCCL module operations) to designate the applicable case figure code to be performed. The UNIT-LEVEL-NBR field contains the plan element level number as determined from the prime keyword (e.g., if prime keyword equals "system", the unit level number is set to 1). The UNIT-INPUT-NBR field contains the MD-PLAN-TABLE entry number corresponding to the plan element name contained in the KEYWORD-VALUE field.

3. Random Collect File

The following data record description applies:

• 01 RANDOM-COLLECT-RECORD.

05	LAST-BLOCK-INDICATOR	PIC X(6).
88	LAST-BLOCK	VALUE HIGH-VALUE.
05	STORED-ITEM-NBR	OCCURS 42 TIMES PIC X(3).

88 LAST-ITEM

VALUE HIGH-VALUE.

05 STORED-ITEM-VALUE

OCCURS 42 TIMES
PIC S9(9) COMP.

The above record description constitutes a 384 character block in random file storage. The record key is computed such that up to three record block (i.e, 126 items) may be stored in the random file for each element of the MDCR Collection unit that corresponds to a current plan element input. The number of record blocks may be increased (when required) by modifying the following data description:

77 BLOCK-RANGE

PIC S9(9) COMP VALUE 3.

The last block used for any given element is indicated by the LAST-BLOCK-INDICATOR. The element item number whose value is stored in STORED-ITEM-VALUE is given in STORED-ITEM-NBR. If all 42 item storage spaces are not used in the last block, "high-values" are placed in the entry following the last-stored item to terminate retrieval operations.

c. Branching and Error Conditions

The following table describes the branching and error conditions relative to HIPO diagram 1.1.4.3.1.1.

Function Reference	Condition Code	Message Category	Program Action	Note
3	30	ERR	Perform Process #8	
4	17	ERR	Bypass Process #5	
5	2,19	ERR	Exit Process #5	

2.2.1.1.4.3.1.2 PCCL - Process Collection

Automatically generated source code statistics and manually input management data is collected in accordance with management plan and report level format specifications.

a. Program Operations

MDCR Format units, corresponding to report levels (i.e., system, subsystem, etc.) designated in the management plan, are read and output to a temporary sequential file (i.e., the MD Formats File). Working storage tables are also generated to contain an entry for each statistical (i.e., numeric) data item referenced in the format specifications. The Plan Keyword File is then read and a table of Source Section library names is maintained based on "project" and "library" keyword inputs from that file. The occurrence of a "module" level plan element causes all designated Source section libraries to be assigned and a search to be made for a section index entry corresponding to the designated module name. After the "top unit" code is found, it is scanned for "included-unit" references. As included-unit references are found, a search is made to find the included unit name among the Source section indexes. The search continues through the included-unit code to look for further included-unit references, and so on. Required statistical data is extracted from the unit accounting record of each such unit for summarization in module level format table storage. A procedure is next performed to collect manual data statistics (i.e., numeric data inputs) that may have been supplied for the system, subsystem, module or job plan element being processed. At this point all numeric data inputs (both automatic and manual) have been collected. If there are any cyclic data references present in the report level being processed, retrieval is made against the Random Collect File to obtain data previously collected relative to the items referenced by the cyclic data format specification. This retrieved data is processed to produce the required cyclic data values as noted in step 9 of HIPO diagram 1.1.4.3.1.2). Step 10 positions the MD Formats File to obtain a list of all data items for the report level being processed and utilizes that input to "drive" the collection of numeric and non-numeric data items. The output of this process is written to a temporary sequential file (i.e., the New Collect File) awaiting input to the final stage of processing performed by the UPCL subroutine.

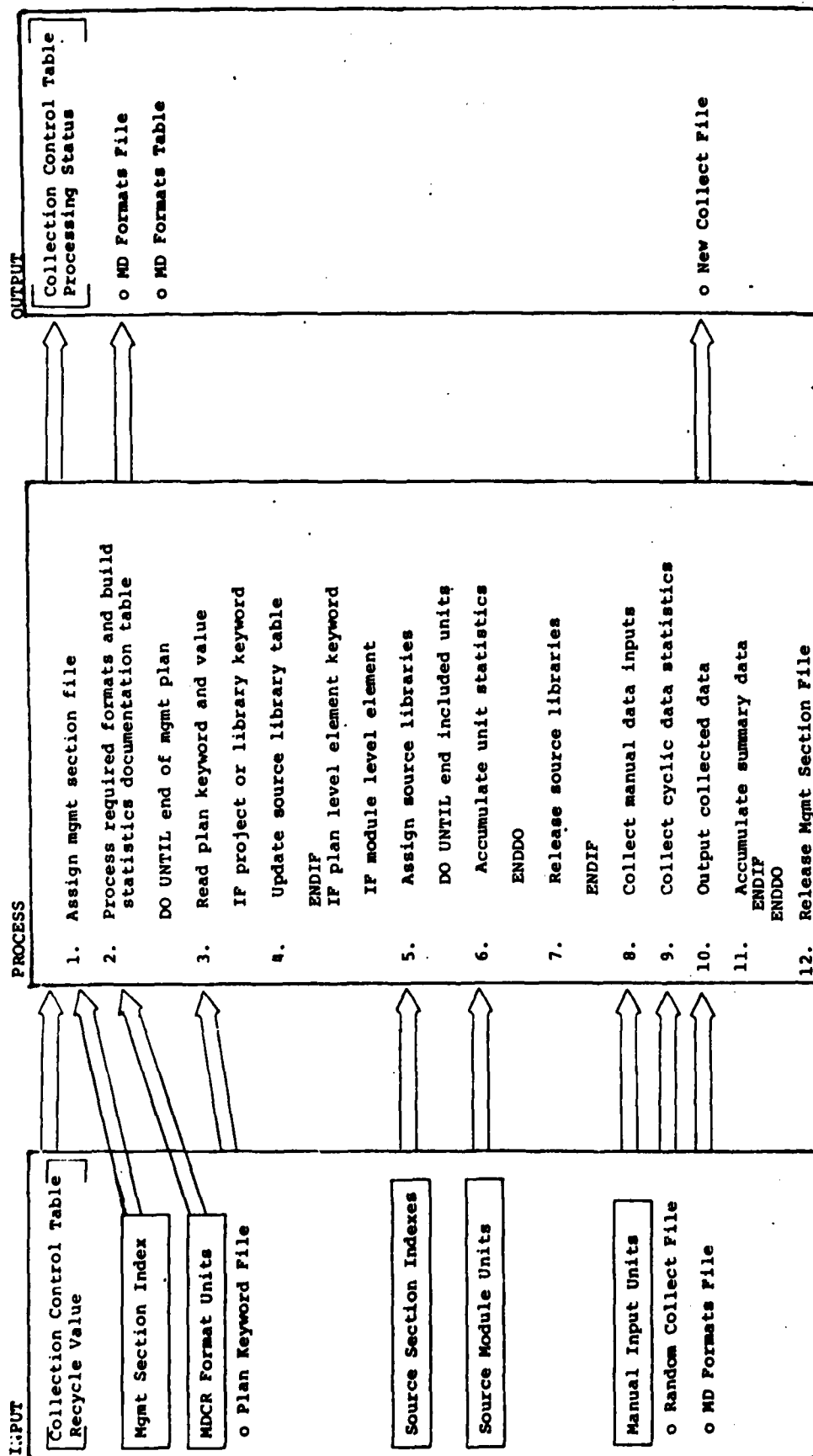
b. Data File and Table Descriptions

1. MD Formats File

This sequential file contains records read from the MDCR Format units in PSL random storage blocks. If present,

Diagram ID: 1.1.4.3.1.2

Name: PCCL - Process Collection Description: Retrieve Automatic and Manual Data



the unit level format is first output to this file succeeded by module, job, subsystem and system formats, as required, with intervening "header" records placed to identify the succeeding format levels. The following description applies to such records:

• 01 MD-FORMATS-RECORD.

05 DATA-RECORD-CODES

10 FILLER PIC X(5).
10 DATA-TYPE-CODE PIC X(1).
88 HEADER-RECORD VALUE "H".

05 FORMAT-DATA.

10 DATA-RECORD-ID.

15 PRIMARY-ITEM-NBR PIC 9(3).
15 DATA-SEQUENCE-NBR PIC 9(3).

10 DATA-INPUT-SPEC.

10 DATA-SOURCE-CODE PIC X(1).
88 MANUAL VALUE SPACE.
88 SPECIAL VALUE "\$".
88 INTRA-FORMAT VALUE "*".
88 ACCOUNT-RECORD VALUE "A".
15 REFERENCED-ITEM-NBR PIC 9(3).

10 DATA-EDIT-SPEC.

15 EDIT-CHECK-TYPE PIC X(1).
88 NUMERIC-EDIT VALUE "N".
15 EDIT-CHECK-VALUE PIC 9(2).

10 FILLER REDEFINES DATA-EDIT-SPEC.

15 SUMMARY-FUNCTION PIC X(3).

	88	AVERAGE	VALUE "AVG".
	88	TOTAL	VALUE "TTL".
	88	MAXIMUM	VALUE "MAX".
	88	CYCLE	VALUE "CYC".
10		ITEM-NAME	PIC X(12).
10		FILLER	PIC X(1).
10		ITEM-LABEL	PIC X(48).
05		HEADER-DATA	REDEFINES FORMAT-DATA.
10		FILLER	PIC X(6).
10		FORMAT-LEVEL-NBR	PIC 9(1).
	88	SYSTEM-LEVEL	VALUE 1.
	88	SUBSYS-LEVEL	VALUE 2.
	88	MODULE-LEVEL	VALUE 3.
	88	JOB-LEVEL	VALUE 4.
	88	UNIT-LEVEL	VALUE 5.
10		FILLER	PIC X(6).
10		HEADER-LEVEL-NAME	PIC X(6).
10		FILLER	PIC X(7).
10		HEADER-UNIT-NAME	PIC X(30).
10		FILLER	PIC X(18).

The above description also applies to the MDCR Format unit data lines exception that no header data is present.

2. Collection Control Table

A section of the control table is used to contain data relative to each report format level that may be in use. The following description applies to this section:

05	FORMAT-ENTRIES	OCCURS 5 TIMES.
10	START-INDEX	PIC S9(9) COMP.
10	END-INDEX	PIC S9(9) COMP.
10	FORMAT-REQUIRED-SW	PIC S9(9) COMP.
88	FORMAT-REQUIRED	VALUE 1.
10	RECYCLE-REQUIRED-SW	PIC S9(9) COMP.
88	RECYCLE-REQUIRED	VALUE 1.
10	FORMAT-LEVEL-NAME	PIC X(6).
10	START-CYCLE-DATE	PIC X(6).
10	CYCLE-DURATION	PIC S9(9) COMP.

The first four entries are reserved for system, subsystem, module and job level formats which utilize START-INDEX and END-INDEX to denote the beginning and end of data item entries made to the MD Format Table for each format level. The fifth table entry, utilized by the unit level format, does not reference the MD Formats Table since none of its data items require accumulation, however the remaining data elements apply to this and the other four entries. The FORMAT-REQUIRED-SW is used to communicate information from the PPCL subroutine (which mutually processes the management plan) to the PCCL subroutine which indicates whether reference to a given format level is made in the management plan. Only format levels to which reference is made are required. The unit level format is not required but will be utilized when module level data is processed to control the collection of unit level data associated with each module referenced by the plan. The RECYCLE-REQUIRED-SW is set by the PCCL subroutine based upon calculations made using the MDCR Format unit accounting record indication of "Start date of cycle" and "Cycle period" in conjunction with the current (i.e., system) date which determine if the specified (i.e., non-zero) cycle period has reached an end. If so, the "End of cycle switch" and "End date of cycle" is set in the MDCR Format unit accounting record which will cause cyclic data accumulations at that level to be recycled the next time the MDCOLLECT Function is used. The RECYCLE-REQUIRED-SW element is set and made available for data collection to indicate that the end of cycle for that format level has been determined. The START-CYCLE-DATE and CYCLE-DURATION elements are also set and made available for output. The FORMAT-LEVEL-NAME entry is taken from the contents of "Unit level name" in the MDCR Format unit accounting record.

Another section of the Collection Control Table is used to contain data relative to each plan level element currently being processed; that is, for which statistical data is being accumulated. The following description applies to this section:

05	MD-PLAN-ENTRIES	OCCURS 4 TIMES.
10	MD-UNIT-NAME	PIC X(30).
10	MD-UNIT-NBR	PIC S9(9) COMP.
88	NO-PLAN-INPUT	VALUE ZERO.
10	MD-BLOCK-NBR	PIC S9(9) COMP.
88	NO-MANUAL-INPUT	VALUE ZERO.
10	SUB-UNIT-COUNT	PIC S9(9) COMP.

As plan level elements are read from the Plan Keyword File they are entered in this section of the Collection Control Table. The MD-UNIT-NAME is the name of the plan level element; MD-UNIT-NBR provides a value from which a record key to Random Collect File can be computed for the purpose of retrieving previously collected cyclic data statistics. The MD-BLOCK-NBR is determined as the PSL storage block number of manual input data that is user-provided for the named plan level element. The SUB-UNIT-COUNT entry is determined from cumulative count of elements (or units) which are subordinate to the referenced plan level element. For example, if the referenced level is "MODULE" (i.e., the third table entry), a count of included units (both real and stub) would be determined; if the referenced level is "SYSTEM" (i.e., the first table entry), a count of plan level elements immediately subordinate to the system level would be computed. The value thus determined is made available for collection at the referenced level of output.

3. MD Formats Table

This table is referenced by the Collection Control Table as previously described. An entry is made in the MD Formats Table for each numeric data item to be collected. The data description of these entries is as follows:

• 01 MD-FORMATS-TABLE.

05	FORMAT-DATA-ENTRY	OCCURS 400 TIMES.
10	PRIME-ITEM-NBR	PIC X(3).

10	REF-LEVEL-CODE	PIC X(1).
10	REF-ITEM-NBR	PIC X(3).
10	EDIT-FUNCTION	PIC X(3).
05	COLLECTED-ITEM-VALUES	OCCURS 400 TIMES.
10	TBL-ITEM-VALUE	PIC S9(9) COMP.

The FORMAT-DATA-ENTRY items are reflective of the data taken from the MD Formats File. The COLLECTED-ITEM-VALUES entries are dynamically updated to contain the accumulated data statistics for each plan level report element currently in process. These values are reset, accumulated, and output in accordance with the management data plan and the dictates of the FORMAT-DATA-ENTRY specifications.

4. New Collect File

The New Collect File is a sequential file having the same description as the MD Formats File with the exception of the following redefinitions:

10	FILLER	REDEFINES ITEM-LABEL.
15	ITEM-VALUE	PIC X(48).
10	FILLER	REDEFINES ITEM-LABEL.
15	COMP-ITEM-VALUE	PIC S9(8).
15	FILLER	PIC X(40).
10	FILLER	REDEFINES ITEM-LABEL.
15	XCHECK-REMARK	PIC X(48).

This redefined data description also applies to the data lines of the MDCR Collection and Archive units which include header data lines as described for the MD Formats File to identify the collected report level data elements. Also, for purposes of data collection the DATA-RECORD-CODES entry is given the following expanded definition:

05	DATA-RECORD-CODES.	
10	FILLER	PIC X(1).
10	XCHECK-VARIANCE	PIC 9(2).

```

10 FILLER                                PIC X(1).

      10 ACTION-CODE                      PIC X(1).

          88 NUMERIC-ITEM                 VALUE "N".

          88 INPUT-HEADER                 VALUE "I".

          88 OUTPUT-HEADER               VALUE "O".

      10 DATA-TYPE-CODE                 PIC X(1).

          88 ITEM-DATA                   VALUE SPACE.

          88 XCHECK-DATA                 VALUE "X".

          88 HEADER-RECORD               VALUE "H".

```

Input header records are placed in advance of collected data records so as to dynamically indicate the report level hierarchy for each plan level element whose data appears following an output header record. The DATA-TYPE-CODE entry distinguishes the three types of collected data as shown while the ACTION-CODE enables numeric (i.e., computational) item data to be distinguished from non-computational item data and to distinguish the two types of header records previously noted. When XCHECK data is noted the XCHECK-VARIANCE entry will contain the computed percent variance and the redefined XCHECK-REMARK entry may contain a user-provided remark.

c. Branching and Error Conditions

The following table describes the branching and error conditions relative to HIPO diagram 1.1.4.3.1.2.

Function Reference	Condition Code	Message Category	Program Action	Note
1	11	ERR	Return to CLMD	
2	26	ERR	Return to CLMD	
2	30	ERR	Return to CLMD	
2	64	ERR	Continue processing	1
5	6	ADV	Continue processing	
6	26	ERR	Continue processing	2
10	65	ERR	Continue processing	

Notes:

- (1) No data is collected for rejected format unit item.
- (2) No data is collected for non-existent source module.

2.2.1.1.4.3.1.3 UPCL - Update Collection

The current MDCR Collection unit is archived and/or replaced by the newly collected management data.

a. Program Operations

HIPO diagram 1.1.4.3.1.3 delineates the processing functions performed by the UPCL subroutine. The New Collect File is read into PSL storage to establish an updated (i.e., new) MDCR Collection unit. The current (i.e., previously stored) MDCR Collection unit data is archived when required or else the PSL storage which it occupies is released. The MDCR Plan unit accounting record is read, when archiving is performed, to obtain the current (i.e., last used) archive unit serial number (ARCHIVE-COUNT). This number is incremented by one and used in conjunction with the date on which the archived data was collected to provide a unique suffix for the new MDCR Archive unit name. The accounting record of the MDCR Plan unit is written to reflect the incremented ARCHIVE-COUNT value.

b. Data File and Table Descriptions

The New Collect File is as previously described.

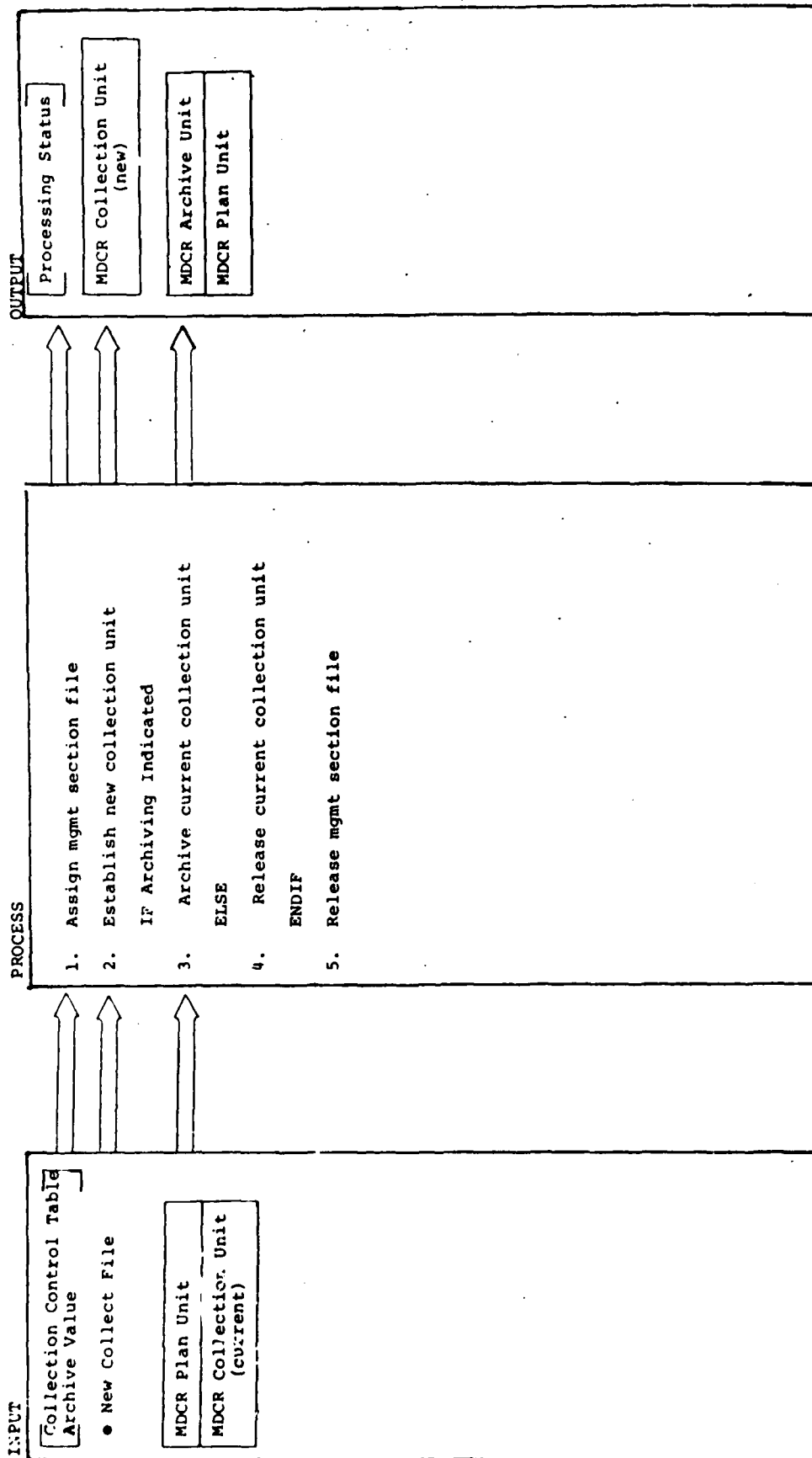
c. Branching and Error Conditions

The one error condition which may be expected is that relating to a lack of sufficient storage space in the designated Management Section. Error condition ERR005 may occur when adding an index or when writing a data line, a PSL024 or PSL044 condition may occur which will cause the UPCL subroutine to exit with the PROCESSING-STATUS parameter set accordingly.

Diagram ID: 1.1.4.3.1.3

Name: UPCL - Update Collection

Description: Store and Archive Collected Data



2.2.1.1.4.4 PRMR - Print Management Report

A job to produce a Program Structure or Management Data report is spawned.

a. Program Operations

HIPO diagram 1.1.4.4 depicts the program operations of the PRMR module. Keyword names and input values are obtained from the Keyword Card file. If the keyword name is found in the keyword table containing a list of general keywords, the keyword value is validated, as required, and stored, else the named keyword is presumed to be utilized by the spawned program. To accomodate this latter purpose, non-general keyword inputs are temporarily stored on the Keyword Hold file awaiting further processing. When all keyword inputs have been processed, the general keyword input is verified to determine if any required keyword input is missing. If no errors have occurred to this point, the designated report procedure is located in the JOB section of the system project library. The job control cards which constitute the located procedure are read and examined for an INPUT flagword (in columns 73 through 80). When such is found, the file code on that job card is inserted into a \$ DATA card and written to the spawned job file after which the stored (i.e., general) keywords and values, followed by the held keywords and values are written to the Spawned Job file. Job control cards which do not contain the INPUT flagword are written directly to the Spawned Job file without modification. The nature of the spawned job activity is determined by the contents of the job procedure stored in the system project library which is not restricted to a specific set of procedures, although the MDPRT Function currently supports the spawning of the PS and MD report procedures and programs as an integral part of the PSL system facility.

b. Data Files and Table Descriptions

1. Keyword Hold File

The following File Description (FD) applies as defined in the PSL Copy Library (CPYLIB).

01 KEYWORD-CARD.

05	KEYWORD-FIELD	PIC X(12).
05	SEQUENCE-FIELD	PIC 9(3).
05	VALUE-FIELD.	
10	PROJECT-FIELD	PIC X(12).
10	SEQUENCE-FIELD	PIC 9(3).
10	LIB-SEC-FIELD	PIC X(8).

```

10 PASSWORD-FIELD      PIC X(12).
10 NAME-FIELD          PIC X(12).
10 FILLER              PIC X(4).
05 UNIT-VALUE-FIELD    REDEFINES VALUE-FIELD.
10 UNIT-NAME-FIELD     PIC X(30).
10 UNIT-KEY-FIELD      PIC X(12).
10 UNIT-TYPE-FIELD     PIC X.
10 FILLER              PIC X(5).
05 FILLER              PIC X(17).

```

The keyword name is stored in KEYWORD-FIELD and the keyword value is stored in VALUE-FIELD according to the nature of the information being forwarded. LIBSEC keyword information is stored in PROJECT-FIELD and LIB-SEC-FIELD to denote a project library section. Non-general keyword values are stored in VALUE-FIELD as received from the Obtain Keyword (OBKW) subroutine described in subsection 2.2.1.2.6.2. The keyword data passed to the Spawned Job file is subsequently in the above format and so utilized by the Print Program Structure (PRPS) and Print Management Data (PRMD) programs.

c. Branching and Error Conditions

The following branching and error conditions apply to HIPO diagram 1.1.4.4:

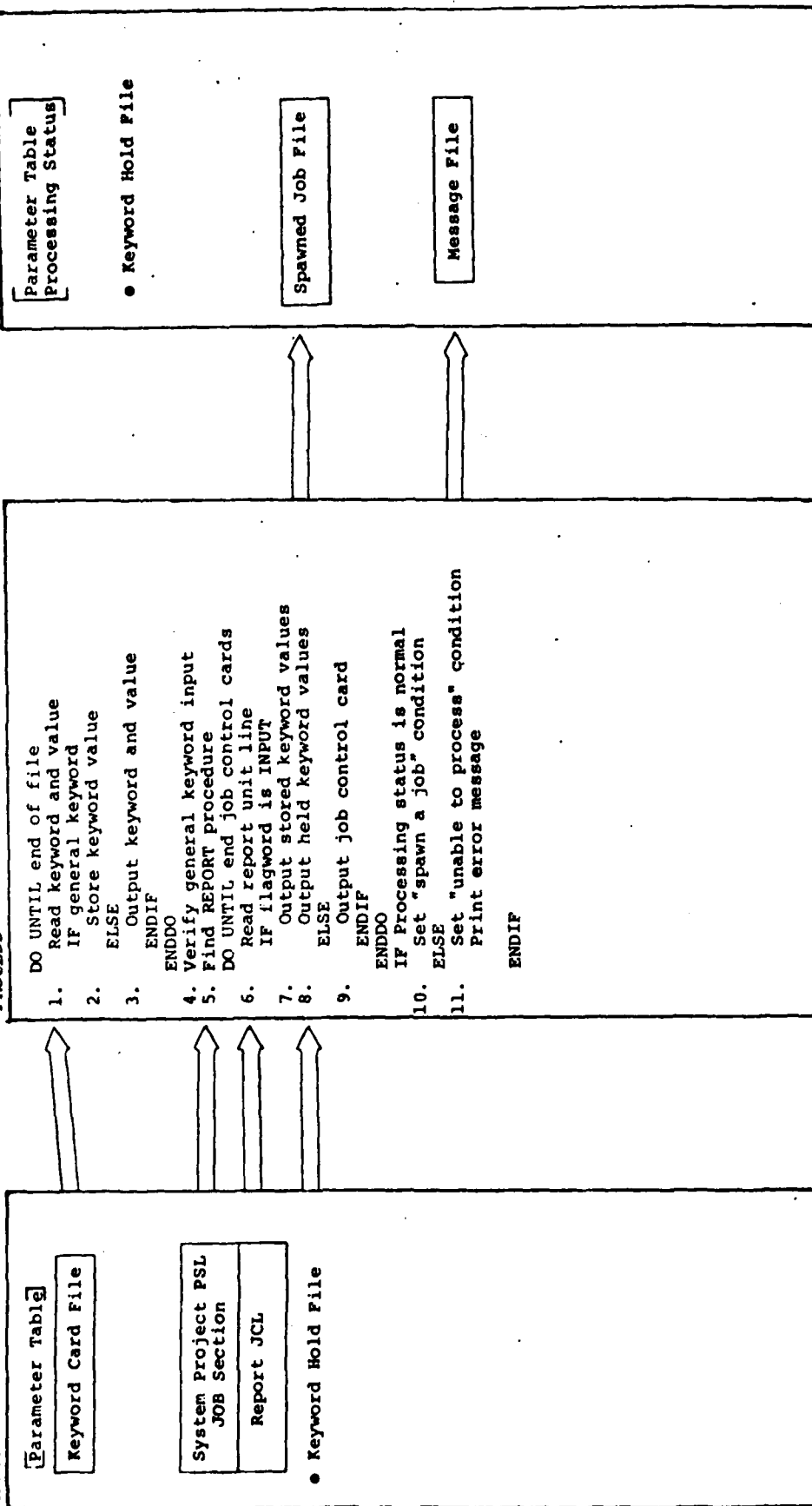
Function Reference	Condition Code	Message Category	Program Action
2	19	ERR	Skip process #5 thru #10
4	32	ERR	Skip process #5 thru #10
5	81	ERR	Skip process #6 thru #10
10	16	ADV	Normal exit
11	7	ERR	Error exit

Diagram ID: 1.1.4.4

Name: PRMR - Print Management Report

Description: Management Data Routine

I:PUT



2.2.1.1.4.4.1 PRPS - Print Program Structure

The PRPS module is called to produce a program structure map for a designated top unit and the included units which are referenced by INCLUDE statements in the top unit and included unit lines of code. CALL statements are optionally processed to delineate the occurrence of called unit references in the program structure.

a. Program Operations

HIPO diagram 1.1.4.4.1 depicts the program operations of the PRPS module. The keywords UNIT, CALL, NEST and associated keyword values are processed from the Keyword Card file. Only the UNIT keyword and value are specifically required since default values are determined for the CALL and NEST keyword inputs. The Read for Program Structure (RDPS) module is called to obtain the start-of-module line for the designated top unit name and the report header lines are initially generated. The INCLUDE and CALL statements contained in the top unit are sequentially processed in the DO loop operation that includes steps 4 through 7. The level number is initialized to 1 at the start of the module and thereafter incremented by 1 for each start-of-unit and decremented by 1 for each end-of-unit processed until the end-of-module (i.e., top unit) is reached. The updated level number is equivalent to the current level of INCLUDED unit nesting and so determines the indentation of the printed output. Units whose level numbers exceed the maximum specified by the NEST keyword input or default value will not be printed in the Program Structure report. If the CALL=YES option is selected, the names of CALLED programs will be non-redundantly added to the CALL-TABLE so that when the end of module is reached, a complete list of all the units called by the top-unit program is stored. The DO loop operation involving steps 2 through 10 is active in obtaining each successive called program name from the CALL-TABLE and treating it as the next top unit name to be processed and reported. SP flag line remarks are printed at the bottom of each page of the program structure report and a Cross Reference list is printed as adjunct which alphabetically orders the INCLUDED unit names and alphabetically lists the INCLUDING units that contain the given INCLUDED unit.

b. Data File and Table Descriptions

The following data files and tables are especially important in the preceding program operations:

FD XREF-FILE.

01 XREF-RECORD.

05	XREF-UNIT-NAME	PIC X(30).
05	XREF-HIGHER-NAME	PIC X(30).
05	XREF-PROJECT-NAME	PIC X(12).
05	XREF-LIBRARY-NAME	PIC X(7).
05	XREF-UNIT-TYPE	PIC X(16).

SD SORT-FILE.

01 SORT-RECORD.

05	SORT-NAMES	PIC X(60).
05	FILLER	PIC X(35).

The XREF-RECORD is written for each included unit as well as the top unit (for which XREF-HIGHER-NAME is set equal to "TOP OF TREE"). The SORT-FILE is used in sorting the XREF-FILE on ASCENDING KEY SORT-NAMES to produce the Cross Reference listing.

c. Branching and Error Conditions

The following branching and error conditions apply to HIPO diagram 1.1.4.4.1:

Function Reference	Condition Code	Message Category	Program Action	Note
1	19	ERR	Exit program	1
	7	ERR	Exit program	1

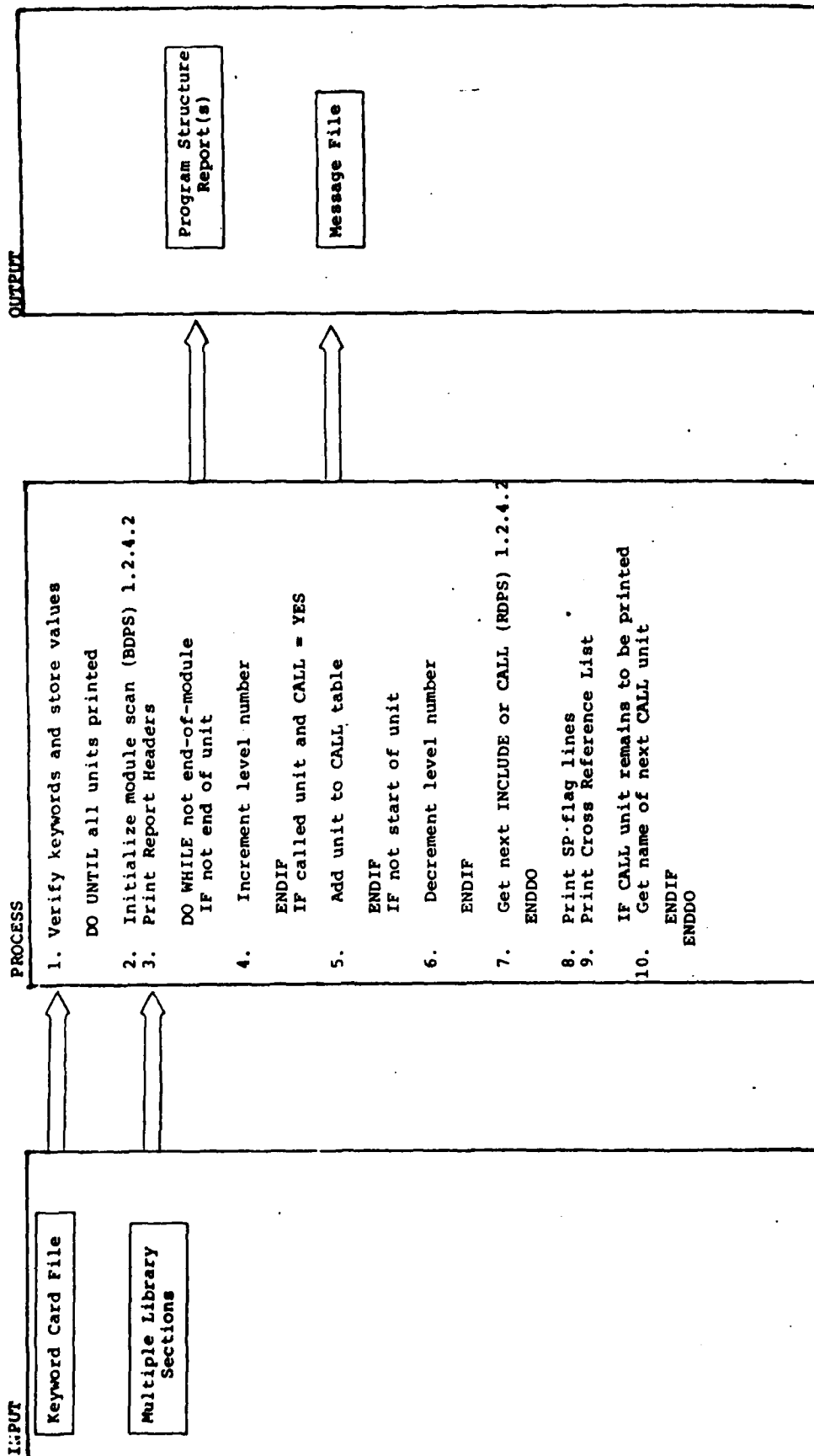
NOTES:

- (1) Error messages are printed using the Print Error (PRER) module. Exit is made from the program to conclude the current spawned job activity and the next spawned job activity (if any) is executed.

Diagram ID: 1.1.4.4.1

Name: PRPS - Print Program Structure

Description: Print Management Report



2.2.1.1.4.4.2 PRMD - Print Management Data

A report is produced from the contents of the most recent management data collection or from a designated set of archived data collections.

a. Program Operations

HIPO diagram 1.1.4.4.2 depicts the processing performed by the PRMD module in a job activity spawned via the MDPRIOT Function using the REPORT=MD option. The Keyword Card File contains the keywords and keyword values that direct PRMD to the appropriate management data library, select the MDCR Collection unit or MDCR Archive unit(s) to provide input data, and specify the report level elements to be retrieved and printed. The MDCR Format units are read to obtain descriptive labels for each of the data items that may appear in the reported output. A Format Label Table is built to contain these output labels for ready use when a report data line is generated. MDCR Collection or Archive unit input is initiated (based upon keyword inputs) and each data line in the initiated unit read to determine its data type. Header data lines precede report data lines in the collected data line sequence to identify the report element hierarchy (e.g., SYSTEM and SUBSYSTEM element names) with which a collected report level element (e.g., MODULE level element) is associated. Report data lines are qualified for output by comparing preceding header line information with the keyword value inputs that are given to suppress/enable designated report levels and/or restrict output to a specific report element hierarchy. The OKAY-TO-PRINT-SW (included in the PRMD-WORKING-STORAGE-77 unit) is set to indicate whether succeeding report data lines are qualified for output. The COBOL Report Writer facility is utilized to generate the required management report. Report level elements and data items are printed in the same sequence as collected and stored so that the management data plan (as prescribed in the MDCR Plan unit) and management data formats (as prescribed in the MDCR Format units) determine the order and content of management data reports which can be produced.

b. Data File and Table Descriptions

The Format Label Table is as follows:

- 01 FORMAT-LABEL-TABLE.

- 05 FILLER

- OCCURS 400 TIMES.

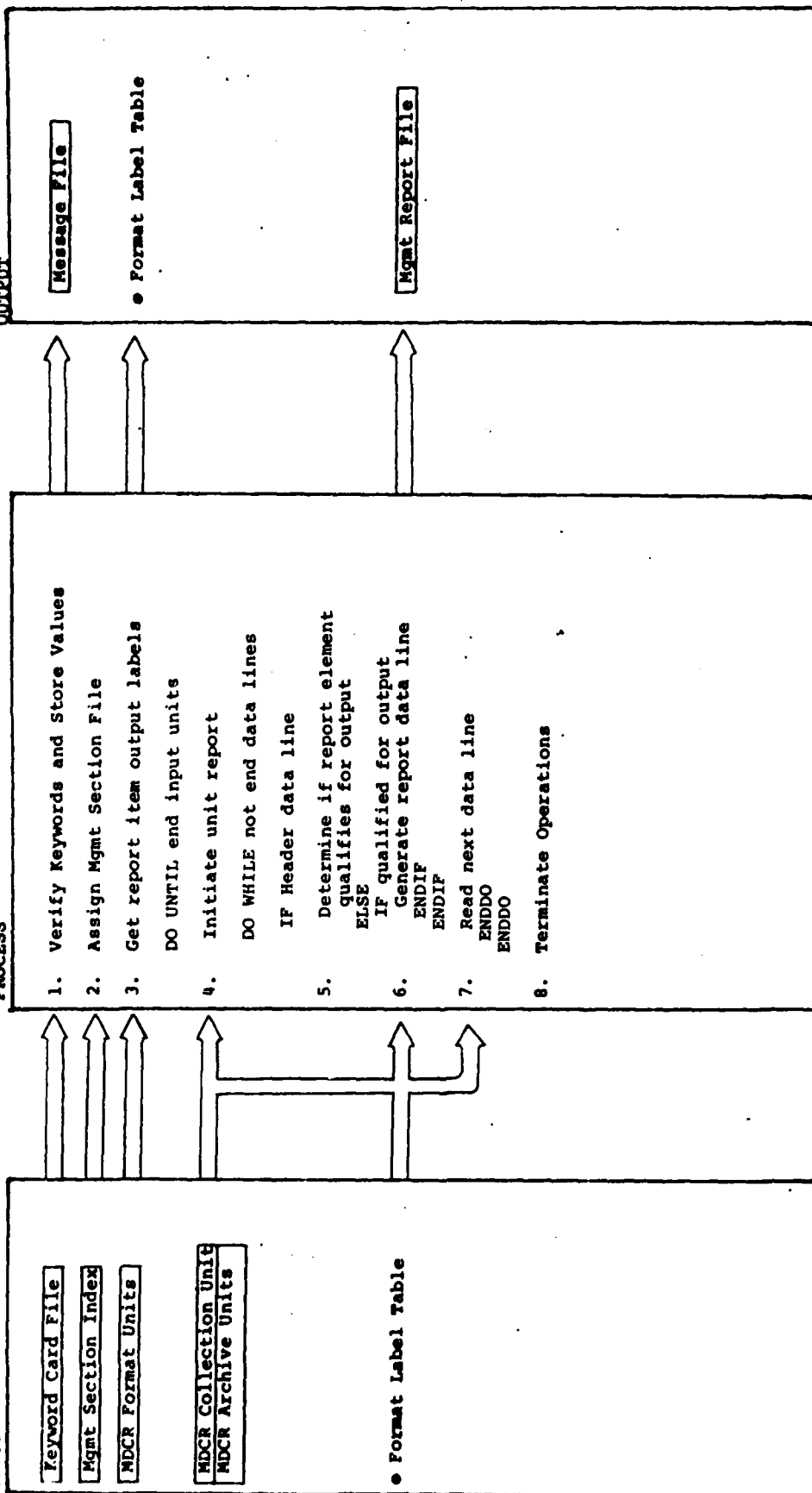
Diagram ID: 1.1.4.4.2

PRMD - Print Management Data

INPUT

PROCESS

OUTPUT



10 PRIME-ITEM-NBR PIC X(3).

10 FORMAT-ITEM-LABEL PIC X(48).

The above table is referenced from the Format Level Table to which the following description applies:

• 01 FORMAT-LEVEL-TABLE.

05 FILLER OCCURS 5 TIMES.

10 START-INDEX PIC S9(9) COMP.

10 END-INDEX PIC S9(9) COMP.

The five format levels are (as referenced under paragraph 2.2.1.1.4.3.1 pertaining to the Collect Management Data module): system; subsystem; module, job and unit. The start and end indexes bracket the item label entries in the FORMAT-LABEL-TABLE against which the data item number read from the MDCR Collection or Archive unit (whose format parallels the New Collection File described under paragraph 2.2.1.1.4.3.1.2, item b) is matched with the PRIME-ITEM-NBR entries corresponding to the report element level being processed.

c. Branching and Error Conditions

The following table delineates the branching and error conditions which apply to HIPO diagram 1.1.4.4.2.

Function Reference	Condition Code	Message Category	Program Action	Note
1	2,19	ERR	Perform Process #8	1
2	6	ADV	Perform Process #8	1
3	11	ERR	Perform Process #8	1
4	26	ERR	Perform Process #8	1
7	18	N/A	Perform Process #4	
8	0	N/A	Normal Exit	
8	7	ERR	Error Exit	

Note:

(1) Error exit in Process #8 will be taken.

2.2.1.1.5 Output Processing

Output processing functions are performed to print an index (INDEX), print source data (SOURCE), print data by author (AUTHOR), print documentation (DOCUMENT) and scan for a character string (CSCAN). The program modules whose operations are invoked by these functions are described below.

2.2.1.1.5.1 PRXX - Print an Index

The module PRXX, which corresponds to the PSL function ** INDEX, prints a status report for a section of a PSL library. The report contains information pertaining to the section as a whole, as well as an alphabetical listing of certain data items pertaining to each unit of the section found in the section index.

a. Program Operations

HIPO diagram 1.1.5.1 describes operations performed. The specific data items printed and the source of each one are shown in Table 2-1. The number of free (unused) blocks in the section is determined by scanning the PSL-FREE-BLOCK-SWS in the PSL-CONTROL-BLOCK's of the section. Each bit (six bits per byte) of the PSL-FREE-BLOCK-SWS represents one block allocated to the section. Each block of the section (except the first control block - block 0) is represented by one bit. A bit value 1 indicates that the corresponding block is free; a bit value 0 indicates that the block is in use. The number of bits with value 0 are counted as the PSL-FREE-BLOCK-SWS are scanned.

b. Data File and Table Descriptions

The data items contained in the index report are shown in Table 2-1.

Diagram No.: 1.1.5.1

INPUT

[Parameter table]
Input card file
PSL library section file
Control blocks
Index entries
Accounting record

Name: PRXX - Print an Index

PROCESS

1. Verify keywords and store values (OBKW).
2. Assign section file (ASFL).
3. Setup header.
4. Calculate blocks in section used.
5. Get first block of first unit in index (RDX, RDBK).
6. IF index is empty
Write page for empty index.
ENDIF
7. DO WHILE not end of index
Start new page if necessary.
8. Setup line data for unit.
9. Write line.
10. Get first block of next unit in index. (RDX, RDBK)
ENDDO
11. Release section file (RLFL):

OUTPUT

[Parameter table]

Unit listing file

Message File

<u>Item Name</u>	<u>Source of Data</u>	<u>Conversion Applied</u>
1. HEADER-PROJECT-NAME	PARAMETER-TABLE	
2. HEADER-MONTH	DATE-TIME (System date/time)	
3. HEADER-DAY	DATE-TIME (System date/time)	
4. HEADER-YEAR	DATE-TIME (System date/time)	
5. HEADER-SECTION-SIZE	PSL-CONTROL-BLOCK	
6. HEADER-PAGE-NBR	Sequential Count	
7. HEADER-LIBRARY-NAME	PARAMETER-TABLE	
8. HEADER-HOUR	DATE-TIME (System date/time)	
9. HEADER-MINUTE	DATE-TIME (System date/time)	
10. HEADER-BLOCKS-USED	Count of zero bits in PSL-FREE-BLOCKS-SWS in PSL-CONTROL-BLOCK(s)	
11. HEADER-SECTION-NAME	PARAMETER-TABLE	Table SECTION-TABLE
12. PRINT-UNIT-NAME	REQUESTED-INDEX-ENTRY	
13. PRINT-UNIT-TYPE	REQUESTED-INDEX-ENTRY	Table UNIT-TYPE-WORDS
14. PRINT-INCL-COUNT	Unit ACCOUNTING-INFORMATION	
15. PRINT-VERSION	Unit ACCOUNTING-INFORMATION	
16. PRINT-MODIF	Unit ACCOUNTING-INFORMATION	
17. PRINT-UPD-MONTH	Unit ACCOUNTING-INFORMATION	

Table 2-1. Index Report Data Items and Source (1 of 2)

<u>Item Name</u>	<u>Source of Data</u>	<u>Conversion Applied</u>
18. PRINT-UPD-DAY	Unit ACCOUNTING-INFORMATION	
19. PRINT-UPD-YEAR	Unit ACCOUNTING-INFORMATION	
20. PRINT-UPD-HOUR	Unit ACCOUNTING-INFORMATION	
21. PRINT-UPD-MINUTE	Unit ACCOUNTING-INFORMATION	
22. PRINT-MGR-NAME	Unit ACCOUNTING-INFORMATION	
23. PRINT-LANGUAGE	Unit ACCOUNTING-INFORMATION	
24. PRINT-NBR-LINES-IN-UNIT	Unit ACCOUNTING-INFORMATION	
25. PRINT-ORIG-MONTH	Unit ACCOUNTING-INFORMATION	
26. PRINT-ORIG-DAY	Unit ACCOUNTING-INFORMATION	
27. PRINT-ORIG-YEAR	Unit ACCOUNTING-INFORMATION	

2-180

Table 2-1. Index Report Data Items and Source (2 of 2)

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Subsequent processing is bypassed	
1	19	ERR	Subsequent processing is bypassed	
1	32	ERR	Subsequent processing is bypassed	
2	6	ADV	Subsequent processing is bypassed	1
4,5,10	33	PSL	Subsequent processing is bypassed	
5,10	50	N/A	Normal end of processing	

Note:

- (1) The required library section file could not be assigned. See description of ASFL for conditions which cause file assignment to fail.

2.2.1.1.5.2 PRSD - Print Source Data

The module PRSD, which corresponds to the PSL function ** SOURCE, produces a printed listing, punch card or tape copy of any unit in a PSL library section that is in card-image format. This includes units in the SOURCE, PDL, LINK, JOB, TEST, MGMT and TEXT sections. If the unit is from the SOURCE or the PDL section and the unit is written as a supported structured language (Structured COBOL, Structured JOVIAL or Structured FORTRAN), the structured code is indented to highlight the control structures used in the code. For card or tape output, one header record is written.

a. Program Operations

HIPO diagram 1.1.1.2 describes operations performed. For the printed listing option, module PRUN is called to produce the standard PSL source listing and to perform structured programming checking for the structured languages.

b. Data File and Table Descriptions

1. UNIT-LISTING-FILE

The format and contents of the standard unit listing report are described with the module PRUN.

2. UNIT-CARD-FILE and UNIT-TAPE-FILE

The UNIT-CARD-FILE and UNIT-TAPE-FILE contain card-image records of each line of each specified unit. In front of each unit is written one header record, the UNIT-HEADER-RECORD. The data items contained in this record are:

05 FILLER PIC X(25) VALUE ALL".".

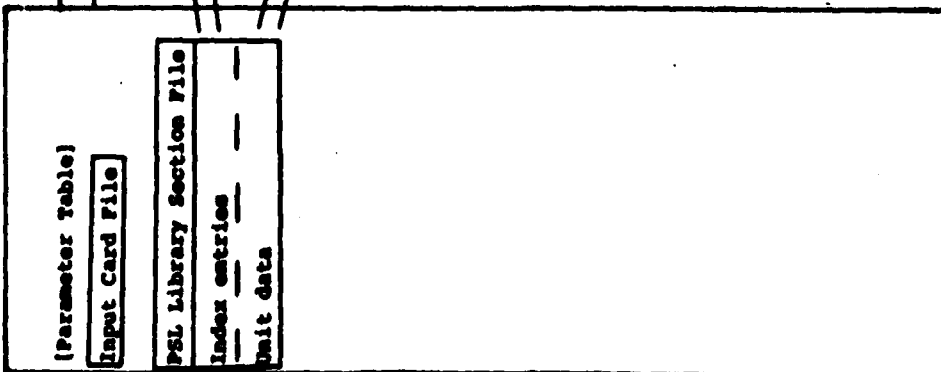
05 HEADER-UNIT-NAME PIC X(30).

- name of unit whose source records follow

05 FILLER PIC X(25) VALUE ALL".".

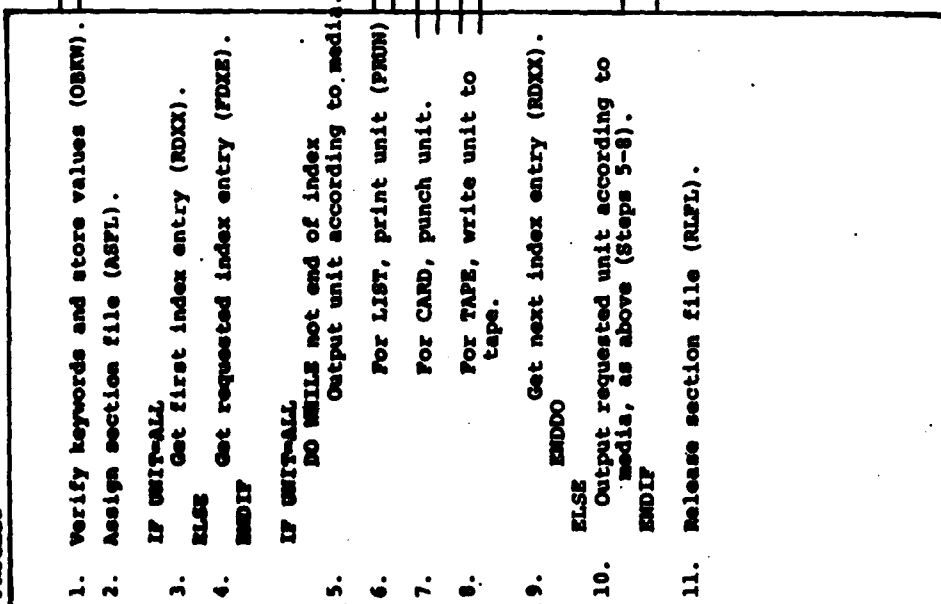
Diagram ID: 1.1.5.2

INPUT

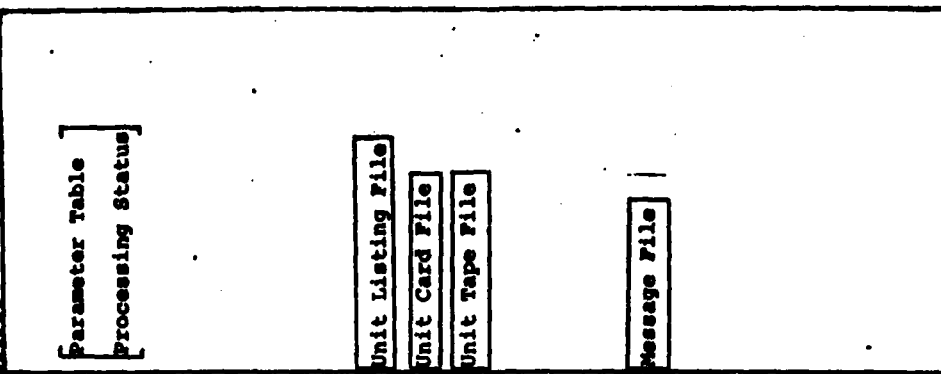


Name: PRSD - Print Source Data

PROCESS



OUTPUT



c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Subsequent processing is bypassed	
1	19	ERR	Subsequent processing is bypassed	
1	32	ERR	Subsequent processing is bypassed	
1	10	ERR	Subsequent processing is bypassed	
2	6	ADV	Subsequent processing is bypassed	1
3,9	50	N/A	Normal end of processing	
4	26	ERR	Subsequent processing is bypassed	
5-8,10	18	N/A	Normal; continue with next unit	
5-8,10	47	PSL	Discontinue current unit; continue with next unit	2
5-8,10	48	PSL	Discontinue current unit; continue with next unit	2

Notes:

- (1) Unable to assign require library-section file. See description of ASFL for conditions which cause file assignment to fail.
- (2) Unable to initialize to read line or to read line. See description of RDUN for conditions which cause read or read initialization to fail.

2.2.1.1.5.3 PRAU - Print Author Report

The PRAU module, which corresponds to the PSL function ** AUTHOR, prints alphabetical listing of data items pertaining to those units of a PSL library section whose originating programmer or last updating programmer is a specified name. Optionally, a source listing of each such unit can be printed. The source option is available only for those sections whose unit are in card-image format. These are SOURCE, PDL, LINK, JOB, MGMT, TEXT, and TEST sections.

a. Program Operations

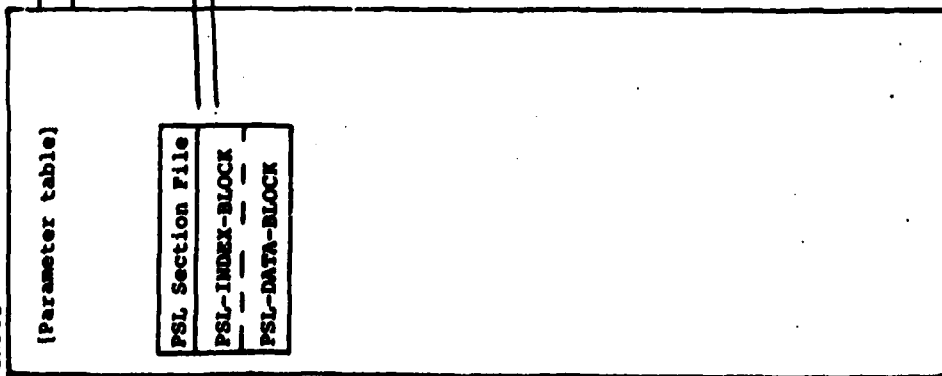
HIPO diagram 1.1.5.3 describes the operations performed. User options determine whether units to be listed should be those with a specific originator, update programmer or either. When originator is specified, the index listing will include the name of the update programmer for each unit printed. Also when update programmer is selected, the name of the originator is printed on the index list for each unit. If both update programmer and originator are specified, they must be the same name, and the index listing will include for each unit either originator or update programmer, whichever is different from the specified value.

b. Data File and Table Descriptions

The data items contained in the author report are shown in Table 2-2.

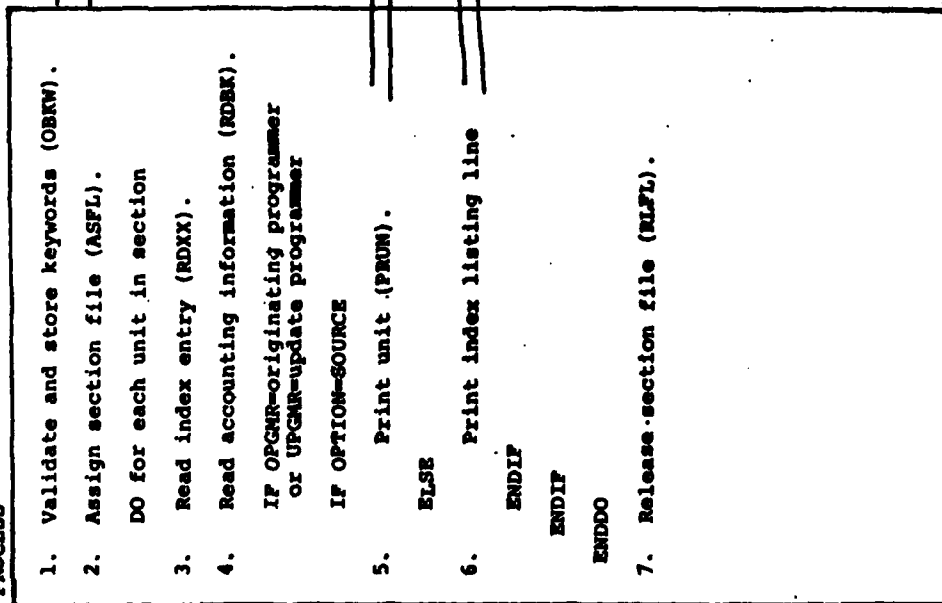
Diagram ID: 1.1.5.3

INPUT

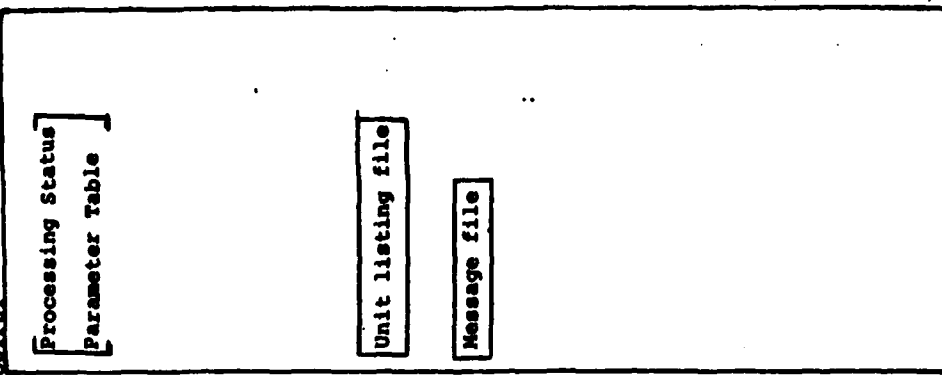


Name - PRAU - Print Author

PROCESS



OUTPUT



<u>Item Name</u>	<u>Source of Data</u>	<u>Conversion Applied</u>
1. HEADER-PROJECT-NAME	PARAMETER-TABLE	
2. HEADER-MONTH	DATE-TIME (System date/time)	
3. HEADER-DAY	DATE-TIME (System date/time)	
4. HEADER-YEAR	DATE-TIME (System date/time)	
5. HEADER-PGMR-LABEL	User Option	
6. HEADER-PGMR-NAME	User Specified	
7. HEADER-PAGE-NBR	Sequential Count	
8. HEADER-LIBRARY-NAME	PARAMETER-TABLE	
9. HEADER-HOUR	DATE-TIME (System date/time)	
10. HEADER-MINUTE	DATE-TIME (System date/time)	
11. HEADER-SECTION-NAME	PARAMETER-TABLE	Table SECTION-TABLE
12. PRINT-UNIT-NAME	REQUESTED-INDEX-ENTRY	
13. PRINT-UNIT-TYPE	REQUESTED-INDEX-ENTRY	Table UNIT-TYPE-WORDS
14. PRINT-INCL-COUNT	Unit ACCOUNTING-INFORMATION	
15. PRINT-VERSION	Unit ACCOUNTING-INFORMATION	
16. PRINT-MODIF	Unit ACCOUNTING-INFORMATION	

Table 2-2. Author Report Data Items (1 of 2)

<u>Item Name</u>	<u>Source of Data</u>	<u>Conversion Applied</u>
17. PRINT-UPD-MONTH	Unit ACCOUNTING-INFORMATION	
18. PRINT-UPD-DAY	Unit ACCOUNTING-INFORMATION	
19. PRINT-UPD-YEAR	Unit ACCOUNTING-INFORMATION	
20. PRINT-UPD-HOUR	Unit ACCOUNTING-INFORMATION	
21. PRINT-UPD-MINUTE	Unit ACCOUNTING-INFORMATION	
22. PRINT-PCMR-TYPE	UPD (update) if originator specified; ORG (originator) if update specified.	n/a
23. PRINT-PCMR-NAME	Unit ACCOUNTING-INFORMATION	n/a
24. PRINT-LANGUAGE	Unit ACCOUNTING-INFORMATION	n/a
25. PRINT-NBR-LINES-IN-UNIT	Unit ACCOUNTING-INFORMATION	n/a
26. PRINT-ORIG-MONTH	Unit ACCOUNTING-INFORMATION	n/a
27. PRINT-ORIG-DAY	Unit ACCOUNTING-INFORMATION	n/a
28. PRINT-ORIG-YEAR	Unit ACCOUNTING-INFORMATION	n/a

2-1-68

Table 2-2. Author Report Data Items (2 of 2)

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Subsequent processing is bypassed	
1	19	ERR	Subsequent processing is bypassed	
1	32	ERR	Subsequent processing is bypassed	
1	10	ERR	Occurs only for SOURCE option; subsequent processing is bypassed	
1	155	ERR	Subsequent processing is bypassed	
2	6	ADV	Subsequent processing is bypassed	1
3	50	N/A	Normal end of processing	
4	33	PSL	Subsequent processing is bypassed	

Note:

- (1) The required library section file could not be assigned. See description of ASFL for conditions which cause file assignment to fail.

2.2.1.1.5.4 PRDC - Print Document

The module PRDC, which corresponds to the PSL function ** DOCUMENT, prints document text incorporating card-image units stored in PSL libraries, according to user specifications. Units in the SOURCE, PDL, LINK, JOB, TEST, MGMT, and TEXT sections may be printed. User subfunction cards provide specifications for spacing, pagination, headers and page eject.

a. Program Operations

HIPO diagram 1.1.5.4 describes operations performed. Line spacing is controlled so that if more than one user specification for spacing between text lines, spacing after header, spacing between units and absolute spacing all apply to a single line, the spacing used will be the maximum value of any one of these. Multiple specifications for spacing of a single line are never cumulative.

b. Data File and Table Descriptions

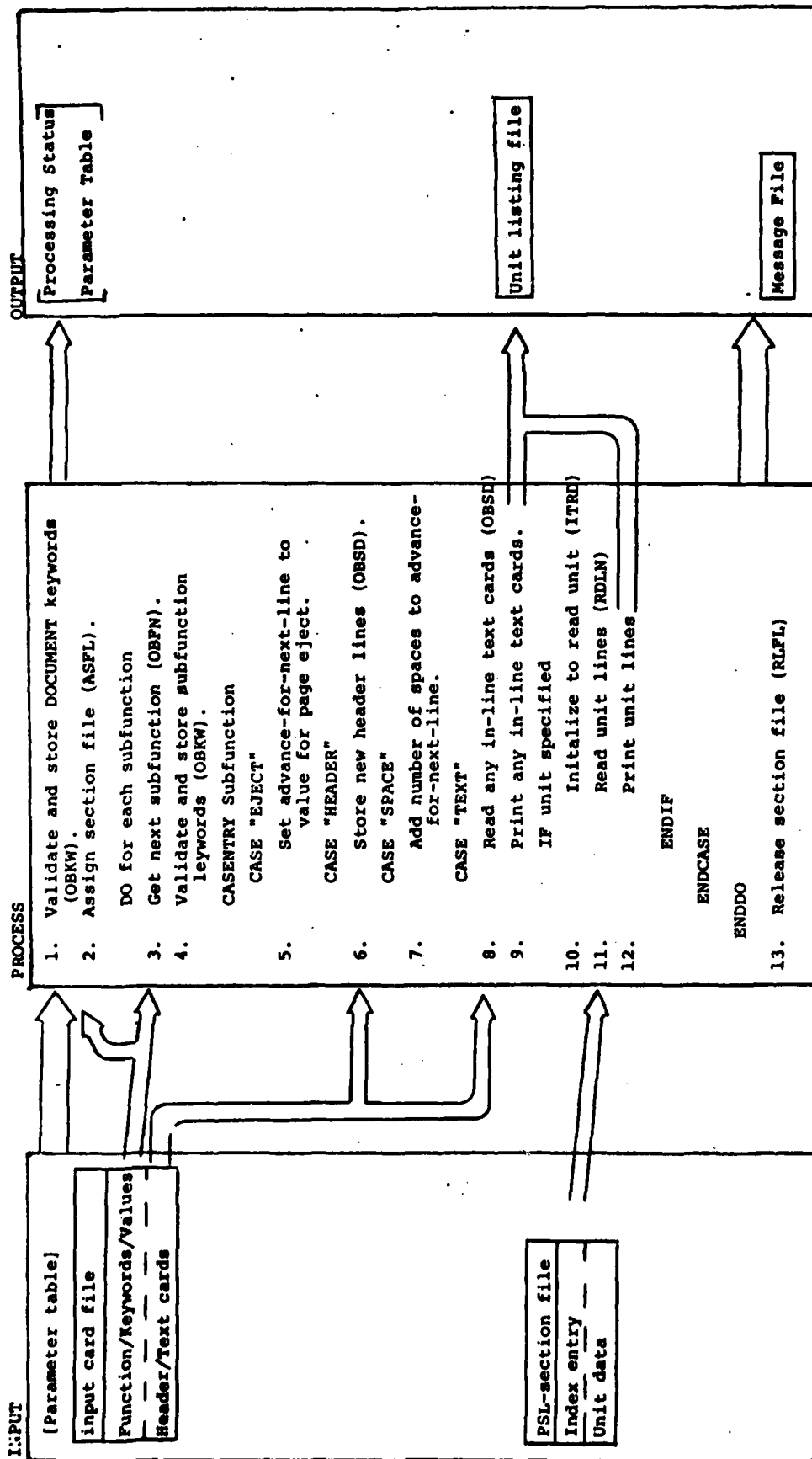
1. Document Report

The document report is the primary output of the PRDC module. Its contents are completely determined by user specification cards. The user may specify that lines be printed on the report in the following ways:

- (a) Header lines to be printed may be inserted in the control card input stream after a ** HEADER subfunction.
- (b) Text lines to be printed may be inserted in the control card input stream after a ** TEXT subfunction.
- (c) One or more units stored in a PSL library section may be requested for printing with the ** TEXT subfunction.
- (d) Insertion of blank lines may be requested with the ** SPACE subfunction.
- (e) Page eject may be requested with the ** EJECT subfunction.

Diagram ID: 1.1.5.4

Name - PRDC - Print Document



c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
3	1	N/A	Normal return to caller	
1,4	2	ERR	Continue processing; may prevent access to and thus printing of a unit	
1,4	19	ERR	Continue processing; may prevent access to and thus printing of a unit	
1	32	ERR	Continue processing; may prevent access to and thus printing of a unit	
1	10	ERR	Continue processing; units from library will not be printed	
2	6	ADV	Continue processing; units from library will not be printed	1
6	148	ERR	Excess header lines ignored	
10	47	ERR	Continue processing; unit from library will not be printed	2
11	48	ERR	Bypass remainder of unit being processed; continue with next subfunction	2

Notes:

- (1) Unable to assign library section file. See description of module ASFL for specific conditions which cause "unable to assign file".
- (2) Unable to initialize or read line of unit. See description of module RDUN for specific conditions which cause "unable to read".

2.2.1.1.5.5 SCSG - Scan for String

The module SCSG, which corresponds to the PSL function ** CSCAN, scans for the occurrence of a specific character string in any unit of a section. It lists the unit name and corresponding lines of code for each occurrence found.

a. Program Operations

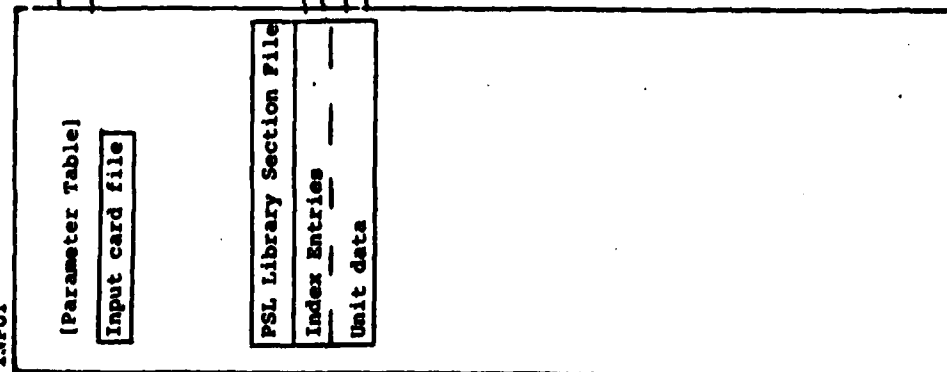
HIPO diagram 1.1.5.5 describes the operations performed.

b. Data File and Table Descriptions

The data items contained in the string scan report are shown in Table 2-3.

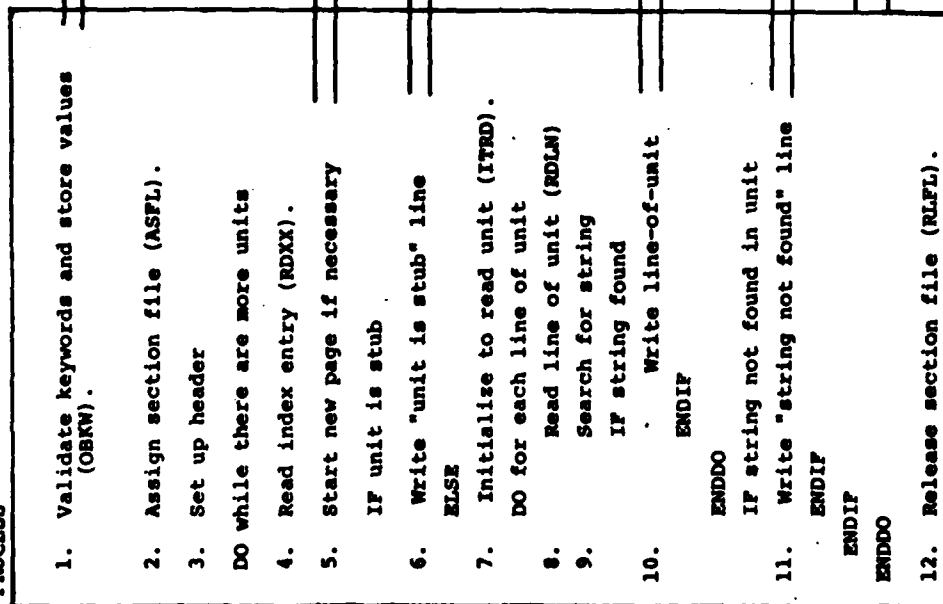
Diagram ID: 1.1.5.5

INPUT

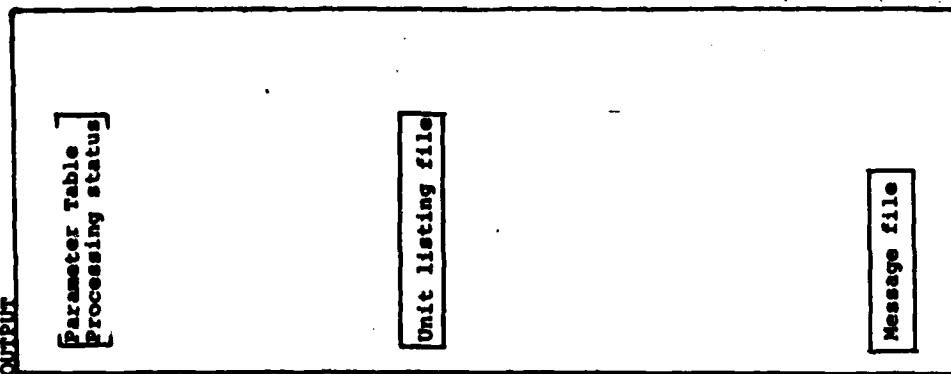


Name - SCSG - Scan for String

PROCESS



OUTPUT



<u>Item Name</u>	<u>Source of Data</u>	<u>Conversion Applied</u>
1. HEADER-PROJECT-NAME	PARAMETER-TABLE	n/a
2. HEADER-STRING	User Supplied	n/a
3. HEADER-PAGE-NBR	Sequential Count	
4. HEADER-LIBRARY-NAME	PARAMETER-TABLE	
5. HEADER-MONTH	DATE-TIME (System date/time)	n/a
6. HEADER-DAY	DATE-TIME (System date/time)	n/a
7. HEADER-YEAR	DATE-TIME (System date/time)	n/a
8. HEADER-SECTION-NAME	PARAMETER-TABLE	Table SECTION-TABLE
9. HEADER-HOUR	DATE-TIME (System date/time)	n/a
10. HEADER-MINUTE	DATE-TIME (System date/time)	n/a
11. LINE-UNIT-NAME	REQUESTED-INDEX-ENTRY from RDX	
12. LINE-LINE-NBR	Number of line which contains specified string.	Counted as lines are read.
or		
12. Word "NONE" or word "STUB"	Generated if unit is stub or contains no occurrences of specified string.	
13. LINE-COL-NBR	Column number where first character of string was found.	Computed as line is scanned.
14. LINE-LISTING	Line of unit where string was found.	

Table 2-3. Data Items from the String Scan Report

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Subsequent processing is bypassed	
1	19	ERR	Subsequent processing is bypassed	
1	32	ERR	Subsequent processing is bypassed	
1	10	ERR	Subsequent processing is bypassed	
2	6	ADV	Subsequent processing is bypassed	1
4	50	N/A	Normal end of processing	
7	47	PSL	Subsequent processing is bypassed	2
8	48	PSL	Discontinue searching of current unit; continue processing next unit	2

Notes:

- (1) Unable to assign library section file. See description of module ASFL for specific conditions which cause "unable to assign file".
- (2) Unable to initialize or read line of unit. See description of module RDUN for specific conditions which cause "unable to read".

2.2.1.1.6 General Functions

General functions are utilized to establish PSL parameters (PARAM Function), write job control cards (JCL Function) and perform job procedures (PERFORM Function). The program modules invoked by these functions are described in the following paragraphs.

2.2.1.1.6.1 ESPA - Establish Parameters

The ESPA establishes the print classification and the project, library, section, password and programmer name parameters that remain in effect until changed by ensuing PSL Function keyword input values that are processed to the Parameter Table.

a. Program Operations

HIPO diagram 1.1.6.1 depicts the program operations of the ESPA module. The Keyword Card file is read to obtain the keyword and keyword values for the PARAM Function. If any error is determined, the Keyword Card file is read until another PARAM Function request is found or the end of the input is reached since erroneous parameter table values are presumed to affect all following PSL Functions until corrected by another PARAM Function input.

b. Data File and Table Descriptions

The ESPA module establishes values in the following table used by all PSL Function modules.

01	PARAMETER-TABLE.	
05	SECTION-REFERENCE.	
10	PROJECT-NAME	PIC X(12).
10	LIBRARY-SECTION-NAME.	
15	SECTION-CODE	PIC X(1).
15	LIBRARY-NAME	PIC X(7).
05	PASSWORD	PIC X(12).
05	PROGRAMMER-NAME	PIC X(12).
05	USERID	PIC X(12).

The USERID value is established by the BCTL module not by the ESPA module.

c. Branching and Error Conditions

The following branching and error conditions apply to HIPO diagram 1.1.6.1:

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Continue to process #2	
	19	ERR	Continue to process #2	

2.2.1.1.6.2 WRJB - Write Job Control Cards

The WRJB module writes user-provided job control cards to the spawned job file.

a. Program Operations

HIPO diagram 1.1.6.2 depicts the program operations of the WRJB module. Although no keyword input is associated with the JCL Function, it is technically required that the OBKW module entry point be called to establish the LAST-PRIME-KW condition prior to calling the OBSD entry point for reading the source data (i.e., user-supplied job control card) inputs. The OBSD entry point in the OBKW module is called to obtain any source data cards that may be present. Although a test is made for an error return from the call to the OBKW entry point, such an error could only be caused by a PSL system/computer malfunction. The only expected exit is that which sets the processing status to spawn a job upon return to the BCTL module.

b. Data File and Table Descriptions

No special data file or table descriptions are utilized.

c. Branching and Error Conditions

No special branching or error conditions are utilized.

Diagram ID: 1.1.6 /

Name: ESPA - Establish Parameters

Description: General Function (PARAM)

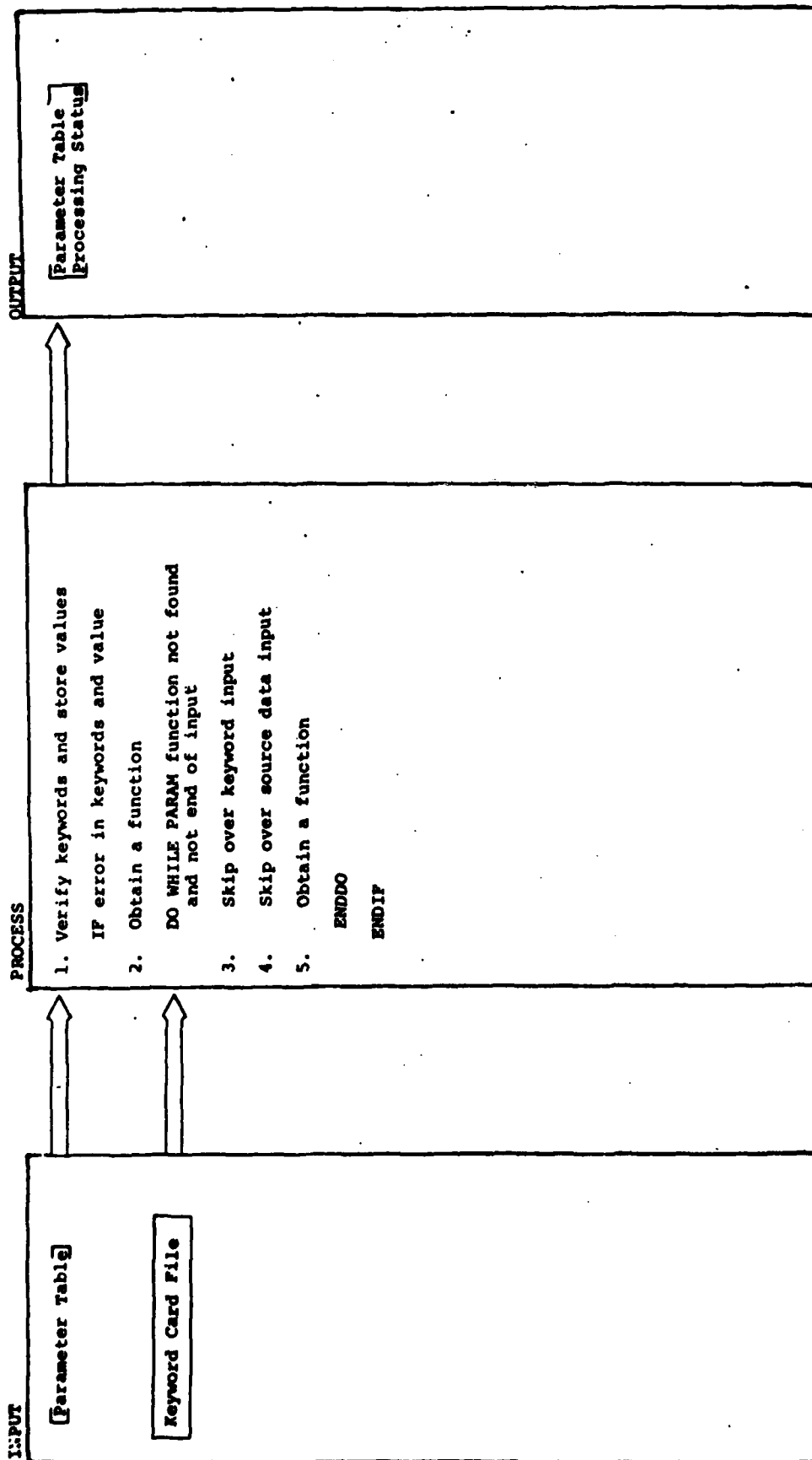
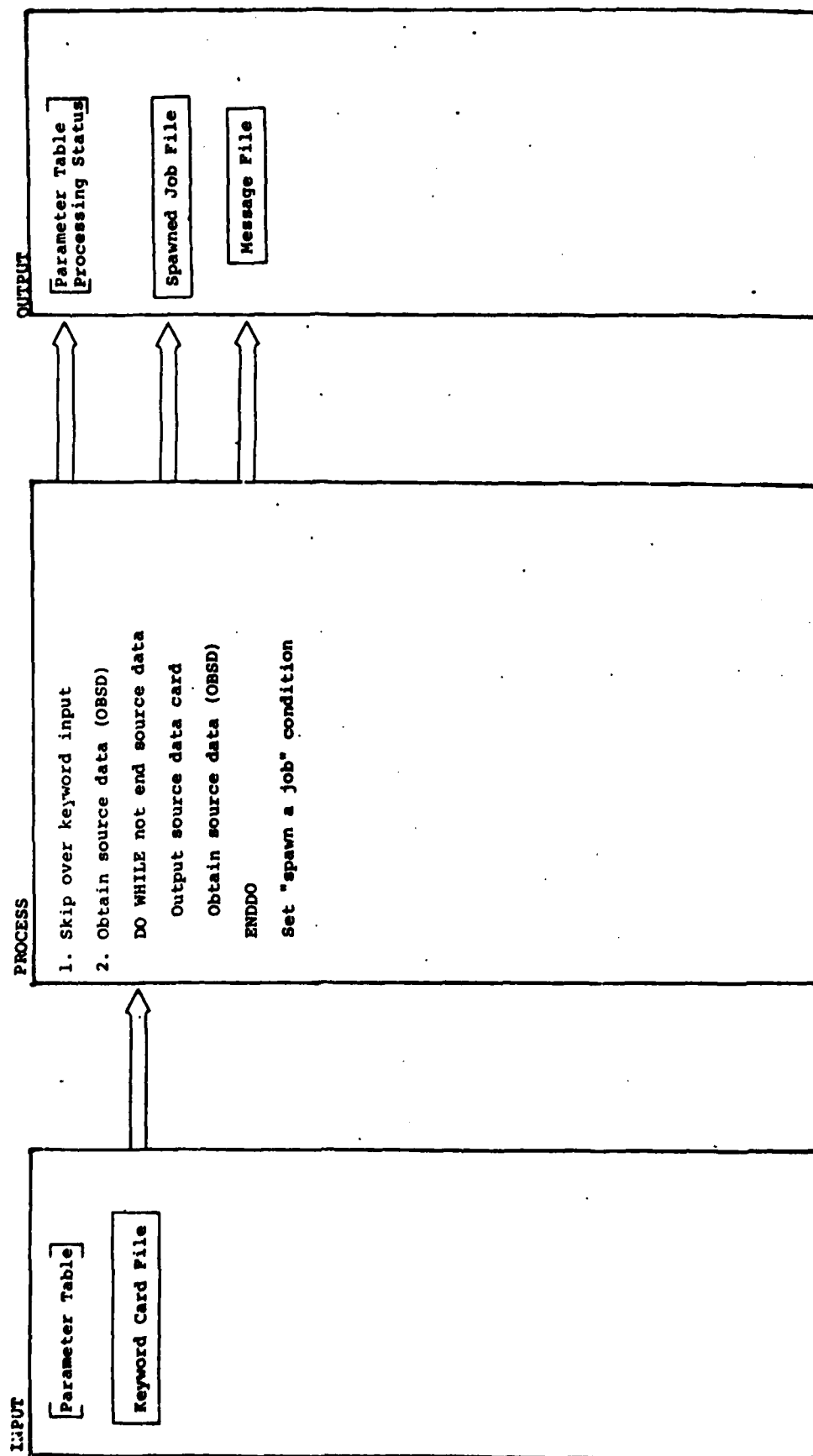


Diagram ID: 1.1.6.2 Name: WRJB - Write Job Control Cards Description: General Function (JCL)



2.2.1.1.6.3 PFJB - Perform Job Control Cards

A user-designated job procedure is retrieved and processed to modify the job control cards (when indicated by self-contained flagword specifications) with information derived from keyword value inputs provided in conjunction with the user of the PERFORM Function. The resultant job control cards (both modified and unmodified) are written to the Spawn-Job file. Independent data files that are referenced through flagword and keyword value specifications are created and/or modified as may be required. File accounting information is correspondingly added and/or updated in the related PSL section storage. Resultant, spawned job activities may subsequently utilize the referenced data files and accounting information in the performance of user and PSL-related operations.

a. Program Operations

1. General Function (PERFORM)

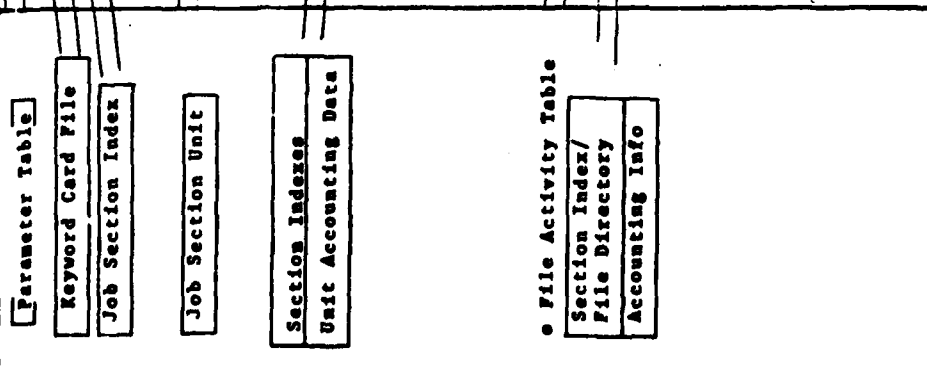
HIPO diagram 1.1.6.3 depicts the top level procedure in PFJB program operations. The keyword value inputs to the PERFORM Function are processed and provided that no errors are detected in that input, the job procedure designated by the PROCEDURE keyword input value is retrieved from the Job section of the project library designated by the PROJECT and LIBRARY keyword or parameter-table input values. Each job card is read and examined for a flagword specification in columns 73 through 80. If a flagword is present, it is processed to perform a job control card modification prior to writing the card to the Spawn-Job file, else the job control card is written, unmodified, to the Spawn-Job file. If no errors are detected through the above processing, the data output file references are noted to create/modify the referenced files with the given (or default) FMS parameter values and to establish/update accounting information for each file so referenced. The processing status parameter is set to spawn a job on normal exit; otherwise, it is set to indicate the inability to complete processing due to a previously noted error.

2. Process Keyword Input

HIPO diagram 1.1.6.3.1 depicts the program operations associated with processing the keyword value input provided with the PERFORM Function. Successive keyword names and values are obtained and each keyword name is validated against a keyword table for the PERFORM Function. The keyword value is then validated in accordance with the requirements imposed for the particular keyword input. When a FILE keyword (i.e., FILE1,

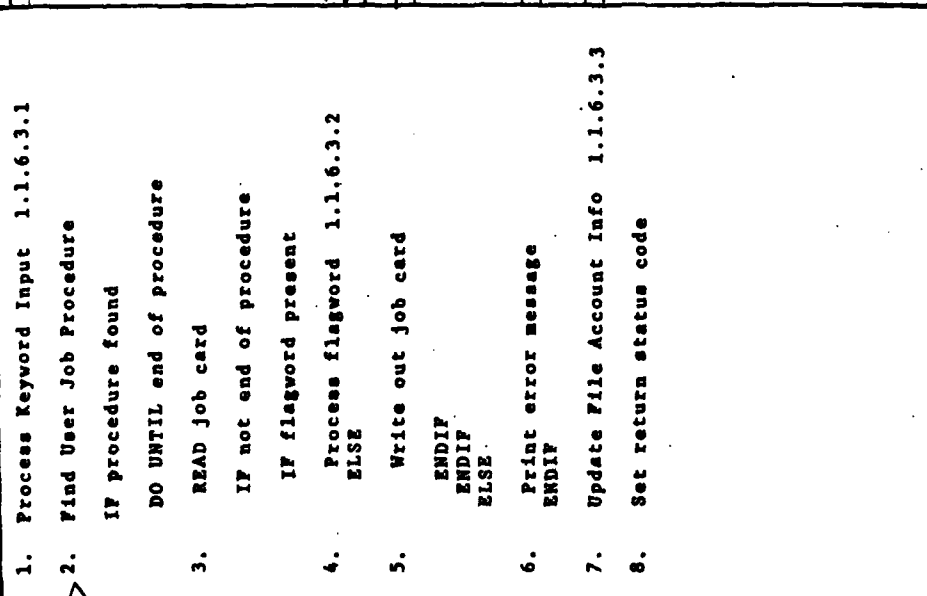
Diagram ID: 1.1.6.3

INPUT



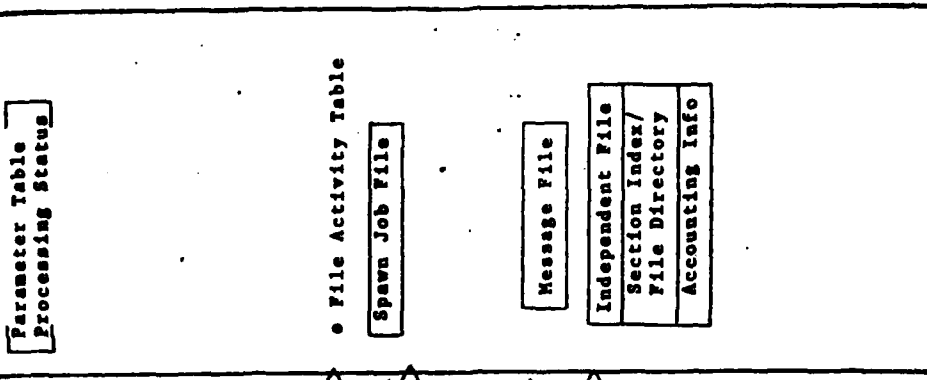
Name: PPJB - Perform Job Control Cards

PROCESS

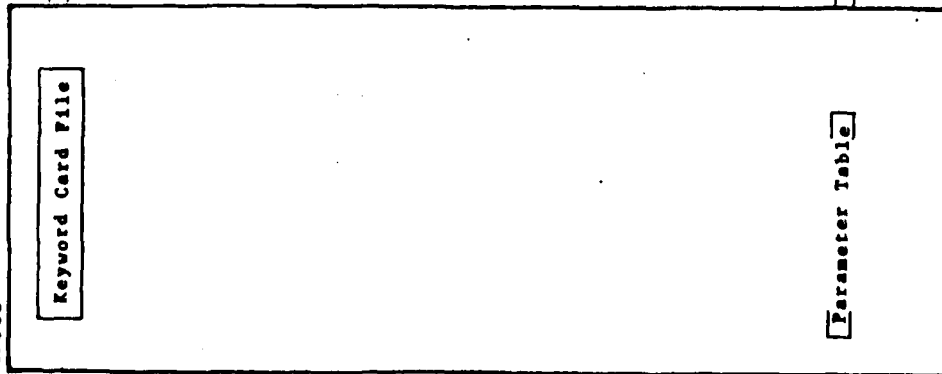


Description: General Function
(PERFORM)

OUTPUT



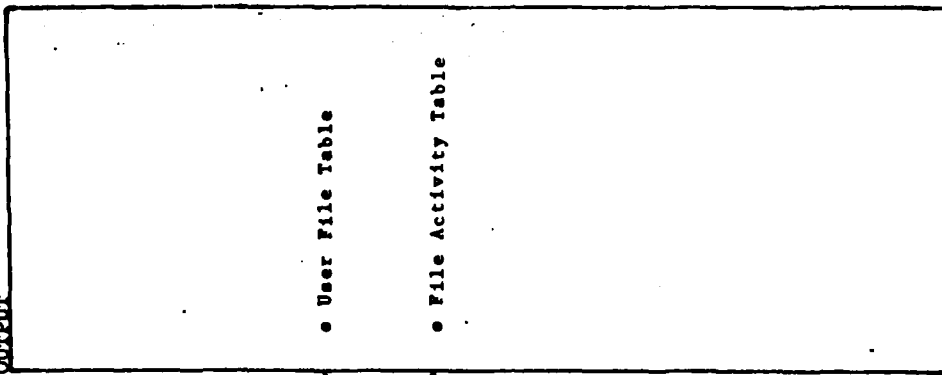
INPUT



PROCESS

```
DO UNTIL End Keyword Input
1. Obtain keyword input
2. Validate keyword name
   IF file keyword
3. Validate keyword specification
   IF first keyword value
4. Validate and store file name
   ELSE
   IF second keyword value
5. Validate and store allocation code
   ENDDIF
   ENDDIF
   ELSE
6. Validate and store keyword value
   as required
   ENDDIF
   IF keyword value error
7. Print error message
   ENDDIF
   ENDDO
8. Check for required keyword input values
```

OUTPUT



FILE2, etc.) is input, the validation process requires that two (2) keyword values be input for each individual FILE keyword name. These two keyword values will be obtained in two successive input records (refer to OBKWE program operation in subsection 2.2.1.2.6.2). The FILE-NBR parameter is initially set to zero to indicate that the FILE keyword specification is to be validated. The FILE keyword name is further checked for a valid fifth character (i.e., 1 through 9) to which the FILE-NBR parameter is set after ensuring that the keyword value input is not the last keyword value (i.e., only keyword value) provided with that FILE keyword name. If it is not the only value provided (i.e., there is a second value forthcoming), it is stored in INPUT-FILE-NAME of the User File Table as an adjunct of the File Activity Table. The next keyword value obtained will have the same FILE keyword name (i.e., since the previous keyword value was not the last) and that second value must also be the last keyword value (i.e., there must be no third, fourth, etcetera values supplied). If this condition is met, the second value is checked for an appropriate allocation code and stored in the File Activity Table. When fewer or more than two keyword input values are provided, it is noted as a keyword value format error. A final check for required keyword input values (Process #8) is made to ensure that the minimum keyword input has been provided.

3. Process Flagword

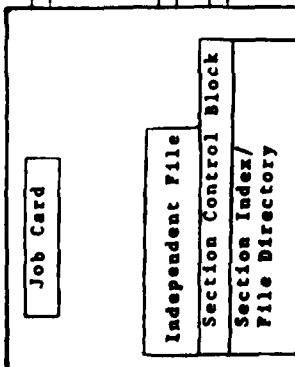
HIPO diagram 1.1.6.3.2 depicts the program operations for processing the flagword specification provided by the user in the given job control card. The flagword character string in columns 73 through 80 is scanned to obtain the flagword name and validate it against a pre-defined FLAGWORD-TABLE listing the flagword names that reference FMS files. If not in that table then the flagword must be CARDS or else be an error. The CARDS flagword signifies that source data cards following the PERFORM Function keyword cards are to be written to the Spawn-Job file between a @DATA,I card and a @END card. If the flagword is found in the table, the flagword value is checked for an appropriate value. If no error is incurred, the PSL section referenced by the flagword is checked to ensure that the section has been previously created and that the section password is valid, when required. It is also determined whether or not keyword value input provides a file name for the given flagword-section reference. If not, a missing keyword error is indicated; otherwise the specific file reference is verified and the job control card is modified provided that no error has been determined.

Diagram ID: 1.1.6.3.2

Name: PFJB - Perform Job Control Cards

Description: Process Flagword

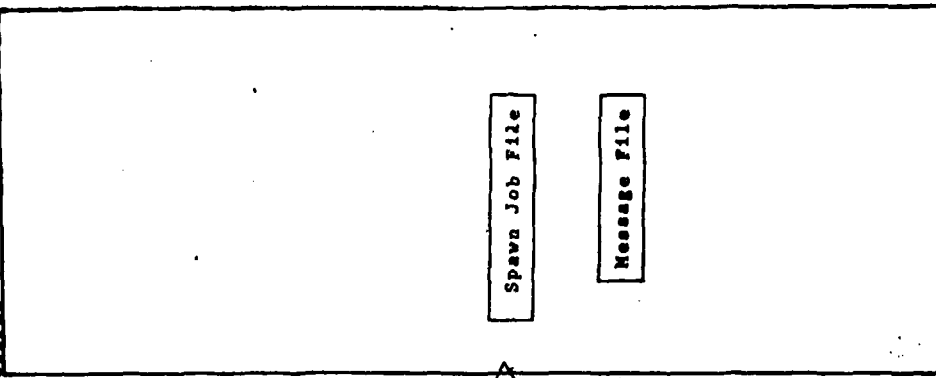
INPUT



PROCESS

1. Obtain flagword name
IF flagword in table
2. Obtain flagword value
IF valid access code
3. Verify PSL section reference
4. Verify PSL file reference 1.1.6.3.2.1
5. Modify job control card
ELSE
6. Print error message
ENDIF
ELSE
IF flagword = CARDS
7. Write @DATA,1 card
8. Write source data cards
9. Write @END card
ELSE
10. Print error message
ENDIF
ENDIF

OUTPUT



4. Verify File Reference

HIPO diagram 1.1.6.3.2.1 depicts the program operations that determine the status of the referenced file. The file name is derived from the UNIT keyword value if the processed flagword designates a PSL section name and the specified unit name is searched for in the unit index of that section, i.e., the USE-DIRECTORY-SW is set to "false" to cause called subroutines to search the unit index rather than the file directory. When a FILE flagword is processed, however, the file name is derived from the corresponding FILE keyword value and the PSL User section file directory is searched since the USE-DIRECTORY-SW is set to "true". If the file name is found, the unit type is validated as being independent. If no error is incurred and the access code designated in the flagword specification indicates an output activity, the FILE-ACTION-SW is set to cause the updating of the existing file accounting information for that particular file as noted in the FILE-ACTIVITY-TABLE. Retrieval is made of the current file accounting information to check whether a UNIT-KEY exists (i.e., not equal spaces) and if it does, whether it matches the KEY keyword input value provided with the PERFORM Function. When no current file is found corresponding to the requested file name, the PSL section options are validated to ensure that the creation of independent files is permitted for that section. If no error is incurred and the access code designated in the flagword specification indicates an output activity, the FILE-ACTION-SW is set to cause the creation of the given file as noted in the FILE-ACTIVITY-TABLE. If the referenced FILE-ACTIVITY-SW now indicates that the file is to be created, then the current reference to that file is valid, otherwise the current reference is a read-only action against a file not previously written and therefore not previously created. A UNIT-NOT-FOUND message is given to indicate an error.

5. Update File Account Information

HIPO diagram 1.1.6.3.3 depicts the program operations that create the Independent files and update the PSL accounting information. The FILE-ACTIVITY-TABLE contains all the needed data to perform the actions required. The FILE-ACTIVITY-SW indicates when an output action is contemplated against a particular file that necessitates an update of the related PSL section data. If the file is to be created, such action is taken and the file accounting information is stored in the section and the file name is added to the section index

Diagram ID: 1.1.6.3.2.1

Name: PFJB - Perform Job Control Cards

Description: Verify file reference

INPUT

Section Index/
File Directory

• File Activity Table

Accounting Info

Section Control Block

PROCESS

1. Find referenced file name in PSL
section directory or index
IF found
2. Validate unit type
IF access code NOT = read only
3. Set file-action-sw to modify file
account info
4. Validate unit key
ENDIF
ELSE
5. Validate section options
IF access code NOT = read only
6. Set file-action-sw to create file
account info
ENDIF
IF NOT file-to-be-created
7. Print error message
ENDIF

OUTPUT

• File Activity Table

Message File

Diagram ID: 1.1.6.3.3

INPUT

• File Activity Table

Independent File

Section Index/Directory

FMS Parameter File

Accounting Data Block

Name: PFJB - Perform Job Control Cards

Description: Update file
account info

PROCESS

DO UNTIL end of Unit File Activity Table
IF file being written

1. Assign PSL section

IF file to be created

2. Create file

3. Add accounting info

4. Add file name to section index
or directory
ELSE

IF FMS parameters input

5. Modify file

ENDIF

6. Modify accounting info

ENDIF

7. Release PSL section

ENDIF

ENDDO

OUTPUT

Independent File

Accounting Data Block

Section Index/Directory

Independent File

Accounting Data Block

or file directory (dependent on the USE-DIRECTORY-SW criteria discussed in the previous paragraph). If the file already exists, its FMS parameters are updated when FMS keyword input values are provided, and the accounting information is updated to reflect the projected, spawned job action to change the independent file contents.

b. Data File and Table Descriptions

The following tables are utilized to control Independent file creation and PSL accounting information update activities:

01	FILE-ACTIVITY-TABLE.	
05	FILLER	PIC X(114) VALUE LOW-VALUES.
05	FILLER	PIC X(19) VALUE SPACES.
01	A-FILLER	REDEFINES FILE-ACTIVITY-TABLE.
05	FILE-ACTION-SW	OCCURS 19 TIMES.
		PIC S9(9) COMP.
	88 FILE-NOT-USED	VALUE ZERO.
	88 FILE-EXISTS	VALUE 1.
	88 FILE-TO-BE-CREATED	VALUE 2.
	88 FILE-ACTIVE	VALUES 1, 2.
05	UNIT-FILE-SECTION-CODE	OCCURS 19 TIMES
		PIC X.
01	USER-FILE-TABLE	VALUE SPACES.
05	INPUT-FILE-NAME	OCCURS 9 TIMES
		PIC X(30).

The FILE-ACTIVITY-TABLE is initialized by VALUE statements to set all FILE-ACTION-SW values to zero, to set user file allocation codes to spaces, to set PSL section file allocation codes to sequential (S) except for the Load section which is set to random (R), and to set all section codes to spaces. The nineteen (19) table entries correspond to the nine (9) user files and ten (10) PSL sections, in that order. The USER-FILE-TABLE is an adjunct to the above table in that INPUT-FILE-NAME is stored for the nine user files corresponding to the first nine entries in the FILE-ACTIVITY-TABLE. The following table is utilized to process flagwords that make reference to user or PSL section files:

```

01 FLAGWORD-VALUES.
05 FW-FILE1      PIC X(6) VALUE "FILE1".
05 FW-FILE2      PIC X(6) VALUE "FILE2".
05 FW-FILE3      PIC X(6) VALUE "FILE3".
05 FW-FILE4      PIC X(6) VALUE "FILE4".
05 FW-FILE5      PIC X(6) VALUE "FILE5".
05 FW-FILE6      PIC X(6) VALUE "FILE6".
05 FW-FILE7      PIC X(6) VALUE "FILE7".
05 FW-FILE8      PIC X(6) VALUE "FILE8".
05 FW-FILE9      PIC X(6) VALUE "FILE9".
05 FW-SOURCE     PIC X(6) VALUE "SOURCE".
05 FW-OBJECT     PIC X(6) VALUE "OBJECT".
05 FW-PDL        PIC X(6) VALUE "PDL".
05 FW-LINK       PIC X(6) VALUE "LINK".
05 FW-LOAD       PIC X(6) VALUE "LOAD".
05 FW-JOB        PIC X(6) VALUE "JOB".
05 FW-TEST       PIC X(6) VALUE "TEST".
05 FW-TEXT       PIC X(6) VALUE "TEXT".
05 FW-MGMT       PIC X(6) VALUE "MGMT".
05 FW-USER       PIC X(6) VALUE "USER".
01 FLAGWORD-LIST REDEFINES FLAGWORD-VALUES.
05 FLAGWORD-TABLE OCCURS 19 TIMES
                   INDEXED BY FT-INDEX
                   PIC X(6).

```

The order of user and PSL section file flagwords determines the position of related information in the FILE-ACTIVITY-TABLE.

c. Branching and Error Conditions

The following branching and error conditions apply to HIPO diagram 1.1.6.3.

Function Reference	Condition Code	Message Category	Program Action	Note
1	2	ERR	Skip to Process #8	1
	19	ERR	at conclusion of	2
	32	ERR	process step	3
	82	ERR		4
4	26	ERR	Skip to process #8	5
	32	ERR	at conclusion of	6
	83	ERR	process step	7
	84	ERR		8
	85	ERR		9
6	81	ERR	Skip to process #8	

c. Branching and Error Conditions (continued)

Function Reference	Condition Code	Message Category	Program Action	Note
7	205	FMS	Exit procedure	10
8	7 16	ERR ADV	Error exit Normal exit	

NOTES:

- (1) Determined in Process #2 of HIPO diagram 1.1.6.3.1
- (2) Determined in Process #7 of HIPO diagram 1.1.6.3.1
- (3) Determined in Process #8 of HIPO diagram 1.1.6.3.1
- (4) Determined in Process #3 of HIPO diagram 1.1.6.3.1
- (5) Determined in Process #7 of HIPO diagram 1.1.6.3.2.1
- (6) Determined in Process #3 of HIPO diagram 1.1.6.3.2
- (7) Determined in Process #10 of HIPO diagram 1.1.6.3.2
- (8) Determined in Process #6 of HIPO diagram 1.1.6.3.2
- (9) Determined in Process #5 of HIPO diagram 1.1.6.3.2.1
- (10) Determined in Process #2 of HIPO diagram 1.1.6.3.3

2.2.1.2 Support Routines

The modules in this subdivision of the PSL system lend special support to the functional processors previously described.

2.2.1.2.1 BKSC - Backup Section

A PSL library section is stored on a Backup Tape file which is opened and closed by the BCTL module at the beginning and end respectively of the PSL job.

a. Program Operations

HIPO diagram 1.2.1 depicts the top-level operations of the BKSC module. Backup records are identified and written in a specific order beginning with record type 1 containing section header information and ending with record type 7 signifying end-of-section. Every unit identified in the section index is backed-up. Independent file data requires that the file parameters be queried (through the Interface Routine QYFLE shown in HIPO diagram 1.4.1.6) for information which is pertinent to restoring the independent file. The returned information is transposed into a format to be used as input for recreating that file at a later time (as done through the Interface Routine CRFLE depicted in HIPO diagram 1.4.1.3). Independent files are dynamically allocated and deallocated through the Interface Routines ALFLE and DAFLE before and after respectively opening and closing file backup operations (refer to HIPO diagrams 1.4.1.1 and 1.4.1.4).

b. Data File and Table Descriptions

Refer to the Restore Library (RSLB) module described under paragraph 2.2.1.1.4 for details on the Backup Tape File format.

c. Branching and Error Conditions

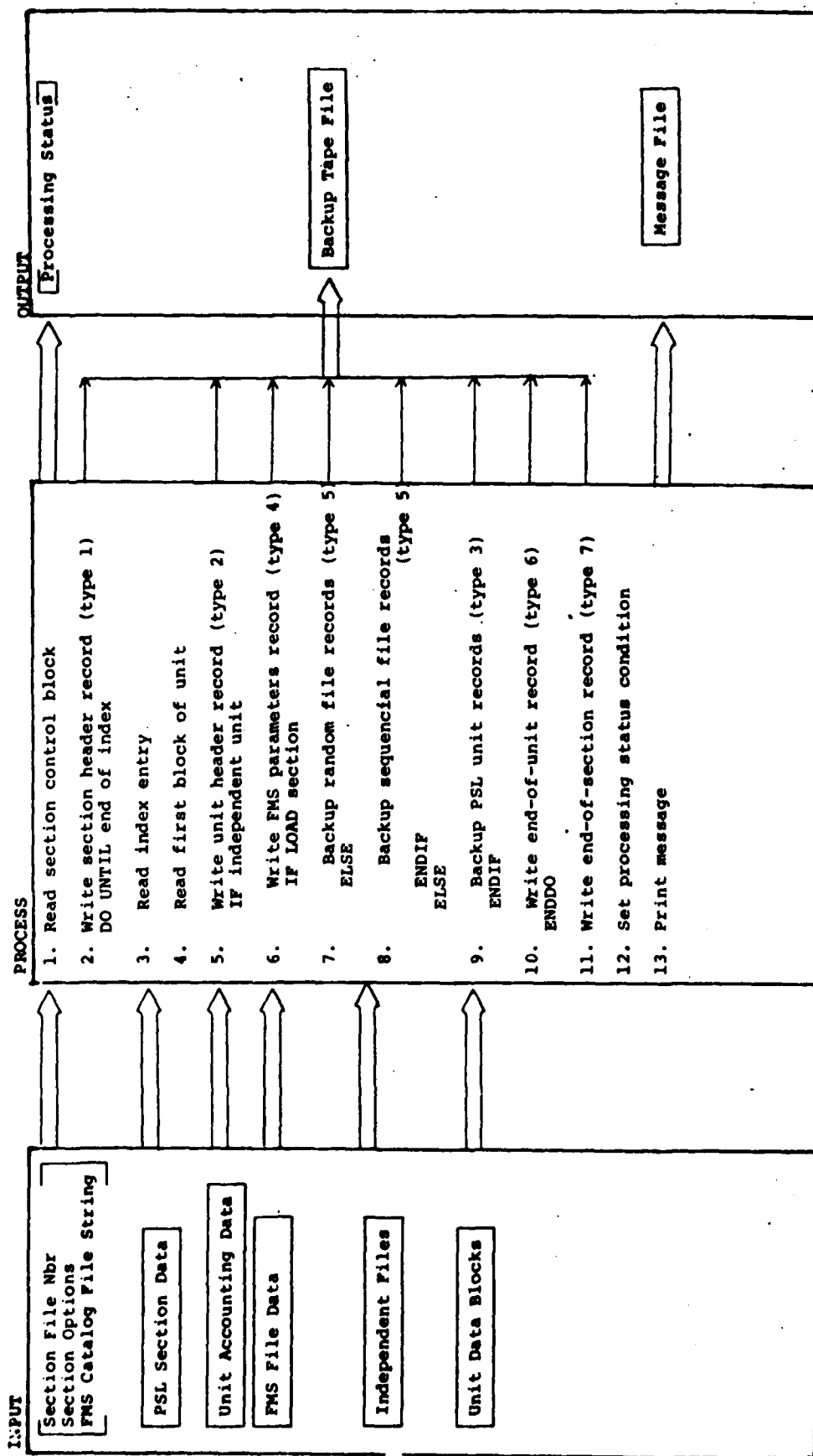
The following branching and error conditions are significant to BKSC module operation:

Function Reference	Condition Code	Message Category	Program Action	Note
3	51		Bypass Process #'s 4-10	1
6	152	ERR	Continue normally	2
12	7	ERR	Error Exit	
12	99	INF	Normal Exit	3

Diagram ID: 1.2.1

Name: BKSC - Backup Section

Description: Support Routine



Notes:

- (1) End-of-index condition terminates DO loop
- (2) FMS parameters set to zero indicating that FMS default parameters are to be used in restore operation unless otherwise overridden.
- (3) Condition code reset to zero before exiting.

2.2.1.2.2 Unit Processing

Unit processing support is provided to process lower units referenced by an INCLUDE statement and to delete PSL section file units as required or requested. The program modules whose operations perform these actions are described below.

2.2.1.2.2.1 PCLU - Process Lower Units

The PCLU module is called to update the higher (i.e., including) unit list maintained for a lower (i.e., included) unit and to add a stub unit when no real unit exists in the library section being processed.

a. Program Operations

HIPO diagram 1.2.2.1 depicts the program operations performed by the PCLU module. The first step in processing lower (i.e., INCLUDED) units is to find the lower unit name in the library section index. If found, the existing lower unit accounting information is updated provided that the unit type value in the index entry indicates an included type unit and the unit is not to be deleted. If the lower unit changes from a singly included to a multiply included (or vice versa) or from a real, non-included to a singly included unit (or vice versa), the unit type value is changed to reflect the new status in both the index entry and the unit accounting information. If the lower unit is a stub and is no longer included (i.e., the current include count equals 1 and the PROCESS-TYPE input parameter value indicates that a reference to the lower unit is being deleted), then the stub unit will be deleted since it is no longer referenced. In contrast, if the lower unit were "real" (i.e., was previously ADDED by the PSL system user), it would not be deleted since the deletion of real units is a prerogative of the PSL user.

When an existing lower unit initially becomes multiply included (i.e., transitions from singly included to multiply included) a higher-unit block is built to contain a list of the higher unit names in which the lower unit is included. The initial number of such names is two but as more INCLUDE statements make reference to the lower unit, this list is increased and later, as INCLUDE statements are deleted, the list is diminished. Since this list of higher unit names is maintained in alphabetic sequence, an added higher unit reference must be placed into the list at an appropriate point. On the other hand, deleted higher unit reference requires an equivalent entry in the current list be eliminated. The ensuing operation involves the reading of INPUT-HUB (i.e., current higher unit) entries and writing of OUTPUT-HUB (i.e., updated higher unit) entries to accomplish the stated objective. The process is one which requires precise

control over the input of current higher unit blocks and the assignment and output of new higher unit blocks. If the requisite number of a new PSL storage block can be assigned to complete the process of re-writing the higher unit list, then the PSL storage blocks holding the current (i.e., old) higher unit list are released, otherwise the operation is terminated with an error return to the calling program which most usually responds by flagging the INCLUDE statement being processed to denote the inability to complete lower unit processing on that statement.

When a lower unit changes from multiply to singly included, the existing higher-unit block is released and the remaining higher unit reference is stored in the accounting information area. The higher unit block (HUB) pointer in lower unit index entry is made zero to reflect the absence of HUB storage, otherwise this pointer denotes the first block assigned to HUB storage. The next-block-pointer for that HUB points to the next HUB and so on until a next-block-pointer equal to zero indicates that no further HUB storage is assigned to that lower unit.

Lastly, when a INCLUDE statement makes reference to a lower unit which does not exist in the given library section, a stub unit entry is added to the unit index of that section and accounting data for that stub unit is generated and written to an assigned storage block which is referenced by the data-block-pointer value in the index entry for that lower unit stub.

b. Data File and Table Description

The PSL-HIGHER-UNIT-BLOCK description is obtained from the PSL Copy Library (CPYLIB) and is used to describe the format of INPUT-HUB and OUTPUT-HUB data.

```
01 PSL-HIGHER-UNIT-BLOCK.  
   05 HUB-BLOCK-NBR  
   05 HUB-NEXT-BLOCK-NBR  
   05 HUB-NBR-OF-ENTRIES  
   05 HUB-HIGHER-UNIT-LIST.  
      10 HUB-HIGHER-UNIT  
                                           OCCURS 25 TIMES  
                                           PIC X(30).
```

It will be noted that the maximum number of entries per HUB is twenty-five (25). The list of higher unit names is terminated by the first blank entry in the list. When the HUB is fully utilized (i.e., HUB-NBR-OF-ENTRIES=25), the HUB-NEXT-BLOCK-NBR value is zero if no more entries exist, else the value is non-zero and indicates the HUB in which the list of higher unit names is continued.

Diagram ID: 1.2.2.1

Name: PCLU - Process Lower Units

Description: Support Routine

I:PUT

Section File Nbr
Section Options
Process Type
Higher Unit Name
Lower Unit Name
Programmer Name
Input Language

Section Index
Lower Unit Block
Higher Unit Block

PROCESS

1. Find lower unit in section index
IF lower unit found
2. Update unit accounting info
3. Update list of higher units
IF stub and no longer INCLUDED
4. Delete lower unit
ELSE
IF lower unit type changes
5. Change unit type in section index
and unit accounting record
ENDIF
ENDIF
ELSE
IF process type is ADD
Add lower unit stub to section
ENDIF
ENDIF
IF processing status not normal
Print error message
ENDIF
- 6.

OUTPUT

Processing Status

Higher Unit Block

Section Index

Lower Unit Block

Message File

c. Branching and Error Conditions

The following branching and error conditions apply to HIPO diagram 1.2.2.1:

Function Reference	Condition Code	Message Category	Program Action	Note
1	26	ERR	Exit module	1
	38	PSL	Exit module	2
2,6	39	ERR		3
	40	ERR		3
4	70	ADV		

NOTES:

- (1) Only if PROCESS-TYPE-DELETE condition exists.
- (2) Not expected in normal PSL operations.
- (3) Secondary error indication due to a previous, primary error indication received from a called module.

2.2.1.2.2.2 DLUN - Delete a Unit

The DLUN module is called to release the PSL data blocks and delete the entry of a given unit in a specified PSL section.

a. Program Operations

HIPO diagram 1.2.2.2 depicts the program operations of the DLUN module. The first data block of the given unit is read to obtain the unit key from the accounting information and validate the input unit key when required. If the requested unit type is independent, random or sequential, the existing FMS file is purged. Else if the unit is in PDL or SOURCE section storage, the unit data lines are read and each INCLUDE statement is processed via the PCLU module to eliminate references to the deleted unit from the higher unit lists of the lower (i.e., INCLUDED) units.

If the unit to be deleted is not itself included in a higher unit, then all the data block for that unit are released and its entry is deleted from the unit index if it is an independent type unit or from the file directory (USE-DIRECTORY-SW = TRUE) if it is a random or sequential type unit established via the PFJB module in the execution of the PERFORM function. Otherwise, the unit to be deleted is converted to a stub unit by releasing all assigned data blocks other than the first and re-writing the first data block with initialized accounting information (retaining the date-originated) and changing the unit index entry to reflect the change of unit type.

b. Data File and Table Descriptions

No special data files or tables are utilized in DLUN module program operations.

c. Branching and Error Conditions

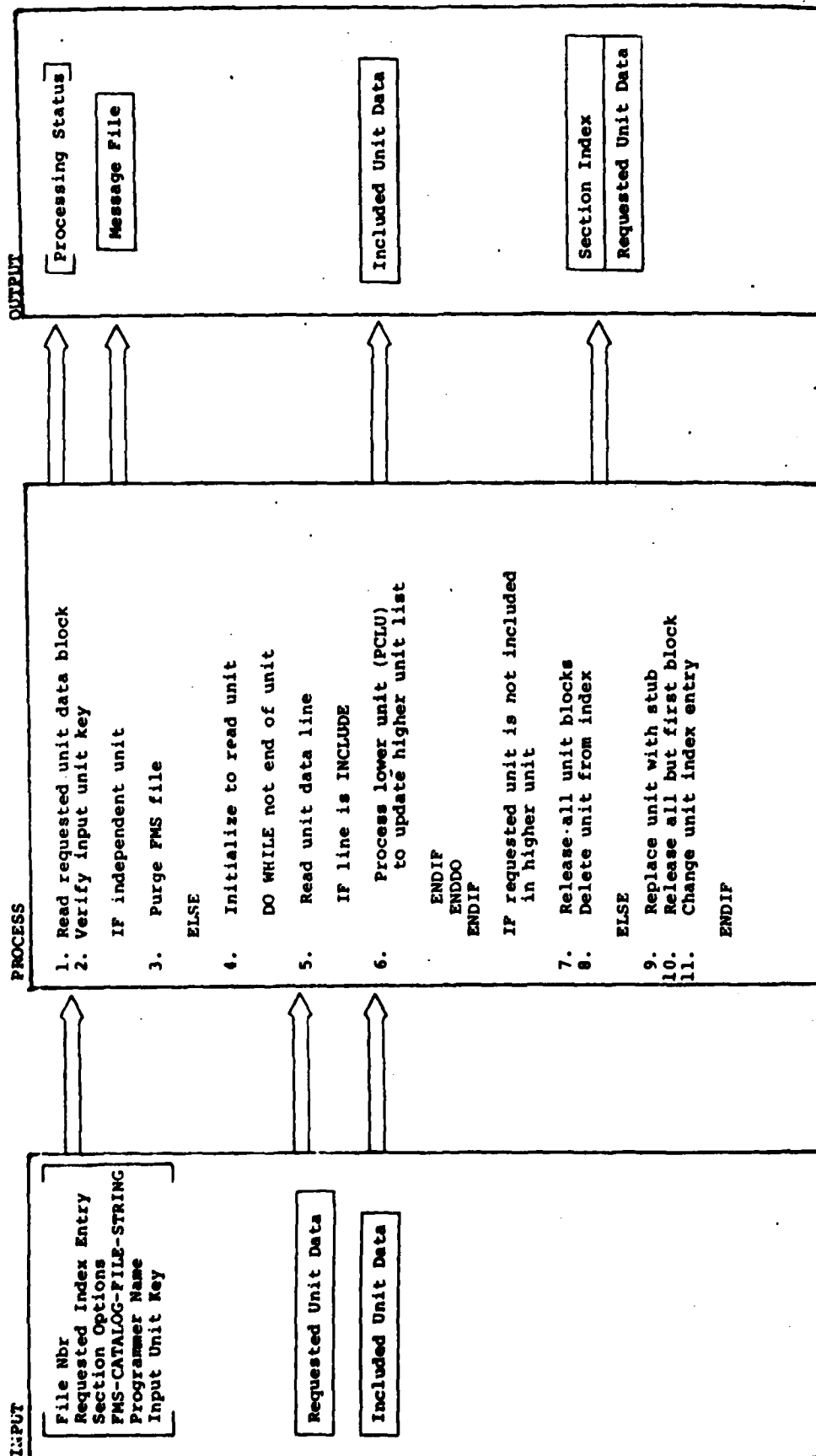
The following branching and error conditions apply to HIPO diagram 1.2.2.2:

Function Reference	Condition Code	Message Category	Program Action	Note
2	9	ERR		
5	55	ERR		1
9	96	PSL		2

NOTES:

- (1) This error cannot occur since the INCLUDE statement has previously been checked; if it had been an error it would have been flagged with a percent sign (%) prefix and not now recognized as an INCLUDE (i.e., since it would appear as %INCLUDE).
- (2) This error is not expected to occur in normal PSL operations.

Diagram ID: 1.2.2.2 Name: DLJN - Delete a Unit Description: Support Routine (Unit Processing)



2.2.1.2.3 PRUN - Print a Unit

The PRUN module is used in support of the print source data (SOURCE) function to determine and call the appropriate print routine for the given unit language. The PRUN module is also used in support of UNIT MAINTENANCE processing.

a. Program Operations

HIPO diagram 1.2.3 depicts the program operation of the PRUN module. In step 2, the unit's accounting information supplies data to the header lines. In step 3, the LANGUAGE-ENTRIES of the LANGUAGE-LINK-TABLE is searched to determine the appropriate language dependent print routine to call. If the unit type is equal to UTYPE-MD-FORMAT or UTYPE-MD-PLAN, the unit name is used instead of the UNIT-LANGUAGE. Steps 4 through 9 are processed according to the result of the search in step 3, which used the existing unit language as the object of the search.

b. Data File and Table Descriptions

- LANGUAGE-LINK-TABLE.
 - see Section 3 for description of "LANGUAGE-LINK-TABLE".
- PRINT-UNIT-OPTIONS.
 - see Section 3 for description of "PRINT-UNIT-OPTIONS".
- CATALOG-NAME.
 - use to build FMS-LIB-SEC-NAME for independent units.

```
05 FILLER                                PIC X(1) VALUE "C".
05 CAT-LIB-SEC-NAME                      PIC X(8).
```

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	47	PSL	Subsequent processing bypassed	1
4	See description of module PRNI for possible conditions.			
5	See description of module PRCB for possible conditions.			

Diagram ID: 1.2.3

Name: PRUN - Print a Unit

Description: Support Routine

INPUT

File-Mbr
Requested-Index-Entry
Section-Options
Print-Unit-Options
FMS-CATALOG-FILE-STRING

PROCESS

1. Initialize to read unit
2. Setup header data
3. Establish print format
CASENTRY Format-Mbr
CASE 1.
4. Print non-indented (PRMI) 1.2.3.3
CASE 2.
5. Print structured COBOL (PRCS) 1.2.3.1
CASE 3.
6. Print structured FORTRAN (PRFT) 1.2.3.2
CASE 4.
7. Print structured JOVIAL (PRJV) 1.2.3.4
CASE 5.
8. Print MDCR format unit (PRFM) 1.2.3.3
CASE 6.
9. Print MDCR plan unit (PRPL) 1.2.3.3
ENDCASE
IF processing status not normal
and not structured program error
10. Print error message
ENDIP

OUTPUT

Processing-Status

Unit-Listing-File

Message-File

c. Branching and Error Conditions (continued)

Function Reference	Condition Code	Message Category	Program Action	Note
6	See description of module PRFT for possible conditions.			
7	See description of module PRJV for possible conditions.			
8	See description of module PRNI for possible conditions.			
9	See description of module PRNI for possible conditions.			
10	66	ERR	Unit not printed Control returned to calling program	

NOTE:

- (1) Unable to initialize read. See description of RDUN for detailed list of causes for this condition.

2.2.1.2.3.1 PRCB - Print COBOL

The PRCB module prints structured COBOL source data in an appropriately indented format. The indentation algorithm only applies to executable source code.

a. Program Operations

HIPO diagram 1.2.3.1 depicts the program operations of the PRCB module. In step 4, if the end of unit is detected, the message "UNIT HAS NO DATA LINES" will be written on the unit listing by the unit message line. If the unit is a STUB, the message "UNIT IS STUB" will be written on the unit listing. In step 5, the NBRLINES-OPTION of the PRINT-UNIT-OPTIONS data group is interrogated for the maximum number of lines to be printed per page. In step 6, the preparation of the print line is determined generally by the first word (WORD) of the source line. If the WORD of the source line is contained in the WORD-TABLE, it has an associated WORD-TYPE and WORD-NBR. Since COBOL is not a free formatted language, certain areas (MARGINS) of the data line are not inspected for executable code. If the CHECK-SP-SW switch of the SECTION-OPTIONS is on, each unit line will be checked for adherence to structured programming principles. Also in step 6, the WORD-TYPE and WORD-NBR are used to dictate indentation and direct structure verb processing (structure, nesting and stacking). In step 7, the unit line is written to the UNIT-LISTING-FILE according to the calculated indentation. Print line overflow processing is activated if printing extends beyond column 100 on the listing line. In step 10, the INCLUDING-UNIT-LIST-PTR of the INDEX-ENTRY issued to point to the PSL-HIGHER-UNIT-BLOCK, which stores the unit names, in which the unit being printed is included. In step 11, if a structure error has been detected, then all the error flags relating to structuring and section options are added together.

b. Data File and Table Descriptions

• UNIT-LISTING-FILE

The UNIT-LISTING-FILE is the formatted output for the listing of the COBOL unit being printed. See Section 3 description of data group UNIT-LISTING-RECORD.

77 WORD-TYPE

PIC S9(9) COMP.

- assigned value representing the function of the associated word.

88	WORD-NOT-IN-TABLE	VALUE 1.
88	WORD-IS-STRUCTURE	VALUE 2.
88	WORD-IS-CONDITIONAL	VALUE 3.
88	WORD-IS-VERB	VALUE 4.
88	WORD-IS-NOISE-WORD	VALUE 5.
88	WORD-IS-INCLUDE	VALUE 6.

01 ADJUSTMENT-VALUES.

- number of print columns specified for indentation
based on structure word or condition.

05	ADJ-FOR-IF-CODE	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-CASE-CODE	PIC S9(9) COMP VALUE 8.
05	ADJ-FOR-NEW-CASE	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-DO-CODE	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-CONDITION-CODE	PIC S9(9) COMP VALUE 8.
05	ADJ-FOR-CONDITION-WORD	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-CONTINUATION	PIC S9(9) COMP VALUE 12.
05	ADJ-FOR-OVERFLOW	PIC S9(9) COMP VALUE 60.

01 FIGURE-STACK.

- records the nesting of structure figures and also used
to check syntax of structured figures.

05	FILLER	OCCURS 50 TIMES.
10	FIGURE-IN-STACK	PIC XX.
	88 FIGURE-IN-STACK-IS-DO	VALUE "DO".
	88 FIGURE-IN-STACK-IS-IF	VALUE "IF".
10	ELSE-PREVIOUSLY-FOUND-SW	PIC S9(9) COMP.
	88 ELSE-PREVIOUSLY-FOUND	VALUE 1.

01 LINE-OF-DATA.

- represents standard format for COBOL statements input.

05	LINE-COLS-1-7	PIC X(7).
05	AREA-A-B.	
	10 AREA-A	PIC X(4).
	10 AREA-B	PIC X(61).
05	FILLER REDEFINES AREA-A-B.	
	10 LINE-COLS-8-16	PIC X(9).
	88 AREA-A-WORD-IS-ON	VALUES ".ON "
		" .ON "
		" .ON "
		" .ON".

```

      88 AREA-A-WORD-IS-OFF      VALUES ".OFF  "
                                   " .OFF  "
                                   " .OFF  "
                                   " .OFF".
01  10 FILLER                    PIC X(56).
    05 IDENTIFICATION-AREA      PIC X(8).
    05 FILLER REDEFINES LINE-OF-DATA.
    05 LINE-CHAR                OCCURS 80 TIMES
                                   PIC X.
      88 LINE-CHAR-IS-BLANK      VALUE SPACE.
      88 LINE-CHAR-IS-BLANK-OR-PUNCTUAT
                                   VALUES SPACE "",""."";".
      88 LINE-CHAR-IS-PERIOD     VALUE ".".
      88 LINE-CHAR-IS-QUOTE      VALUE QUOTE.
      88 CONT-FIELD-USED         VALUES "-"*"/".
      88 LINE-CHAR-IS-PAGE-EJECT VALUE "/"".

01  WORD-TABLE.

    - table stores all words considered in the indentation
      of structured CODE. Each word is classified and
      assigned representative numbers.

05  FILLER                      PIC X(12) VALUE "      ".
05  FILLER                      PIC 9 VALUE 1.
05  FILLER                      PIC 9 VALUE 1.
05  FILLER                      PIC X(12) VALUE "ACCEPT ".
05  FILLER                      PIC 9 VALUE 4.
05  FILLER                      PIC 9 VALUE 1.
05  FILLER                      PIC X(12) VALUE "ADD    ".
05  FILLER                      PIC 9 VALUE 4.
05  FILLER                      PIC 9 VALUE 1.
05  FILLER                      PIC X(12) VALUE "ALTER  ".
05  FILLER                      PIC 9 VALUE 4.
05  FILLER                      PIC 9 VALUE 1.
05  FILLER                      PIC X(12) VALUE "AT     ".
05  FILLER                      PIC 9 VALUE 5.
05  FILLER                      PIC 9 VALUE 1.
05  FILLER                      PIC X(12) VALUE "CALL   ".
05  FILLER                      PIC 9 VALUE 4.
05  FILLER                      PIC 9 VALUE 1.
05  FILLER                      PIC X(12) VALUE "CANCEL ".
05  FILLER                      PIC 9 VALUE 4.
05  FILLER                      PIC 9 VALUE 1.
05  FILLER                      PIC X(12) VALUE "CASE   ".
05  FILLER                      PIC 9 VALUE 2.
05  FILLER                      PIC 9 VALUE 5.

```

05	FILLER	PIC X(12) VALUE "CASENTRY	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC X(12) VALUE "CLOSE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "COMPUTE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "COPY	".
05	FILLER	PIC 9 VALUE 6.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "DATA	".
05	FILLER	PIC 9 VALUE 5.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "DELETE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "DISABLE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "DIVIDE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "DO	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 9 VALUE 8.	
05	FILLER	PIC X(12) VALUE "ELSE	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC X(12) VALUE "ELSECASE	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 9 VALUE 6.	
05	FILLER	PIC X(12) VALUE "ENABLE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "END	".
05	FILLER	PIC 9 VALUE 3.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "ENDCASE	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC X(12) VALUE "ENDDO	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 9 VALUE 9.	
05	FILLER	PIC X(12) VALUE "ENDIF	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 9 VALUE 3.	

05	FILLER	PIC X(12) VALUE "END-OF-PAGE "	.
05	FILLER	PIC 9 VALUE 3.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "ENTER	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "EOP	".
05	FILLER	PIC 9 VALUE 3.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "ERROR	".
05	FILLER	PIC 9 VALUE 5.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "EXAMINE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "EXIT	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "GENERATE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "GO	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "IF	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "INCLUDE	".
05	FILLER	PIC 9 VALUE 6.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "INITIATE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "INSPECT	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "INVALID	".
05	FILLER	PIC 9 VALUE 3.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "MERGE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "MOVE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "MULTIPLY	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	

05	FILLER	PIC X(12) VALUE "NO	".
05	FILLER	PIC 9 VALUE 5.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "NOTE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "ON	".
05	FILLER	PIC 9 VALUE 5.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "OPEN	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "OVERFLOW	".
05	FILLER	PIC 9 VALUE 3.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "PERFORM	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "READ	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "RECEIVE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "RELEASE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "RETURN	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "REWRITE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "SEARCH	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "SEEK	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "SEND	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "SET	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "SIZE	".
05	FILLER	PIC 9 VALUE 5.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "SORT	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	

05	FILLER	PIC X(12) VALUE "START	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "STOP	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "STRING	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "SUBTRACT	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "SUPPRESS	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "TERMINATE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "THEN	".
05	FILLER	PIC 9 VALUE 5.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "UNSTRING	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "USE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "WHEN	".
05	FILLER	PIC 9 VALUE 3.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "WRITE	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC X(12) VALUE "%INCLUDE	".
05	FILLER	PIC 9 VALUE 6.	
05	FILLER	PIC 9 VALUE 1.	
01	FILLER REDEFINES WORD-TABLE.		
05	WORD-TABLE-ENTRIES	OCCURS 70 TIMES	
		ASCENDING KEY IS WORD-IN-TABLE	
		INDEXED BY WORD-INDEX.	
	10 WORD-IN-TABLE	PIC X(12).	
	10 WORD-TYPE-IN-TABLE	PIC 9 .	
	10 WORD-NBR-IN-TABLE	PIC 9 .	
*THIS REDEFINITION ASSUMES THAT A FRESH COPY OF THE MODULE			
* WILL BE LOADED BEFORE EACH CALL			
01	PSL-HIGHER-UNIT-BLOCK REDEFINES WORD-TABLE		
	COPY PSL-HIGHER-UNIT-BLOCK.		

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
2,8	48	PSL	Subsequent processing bypassed	1
10	33	PSL	Subsequent processing bypassed	2

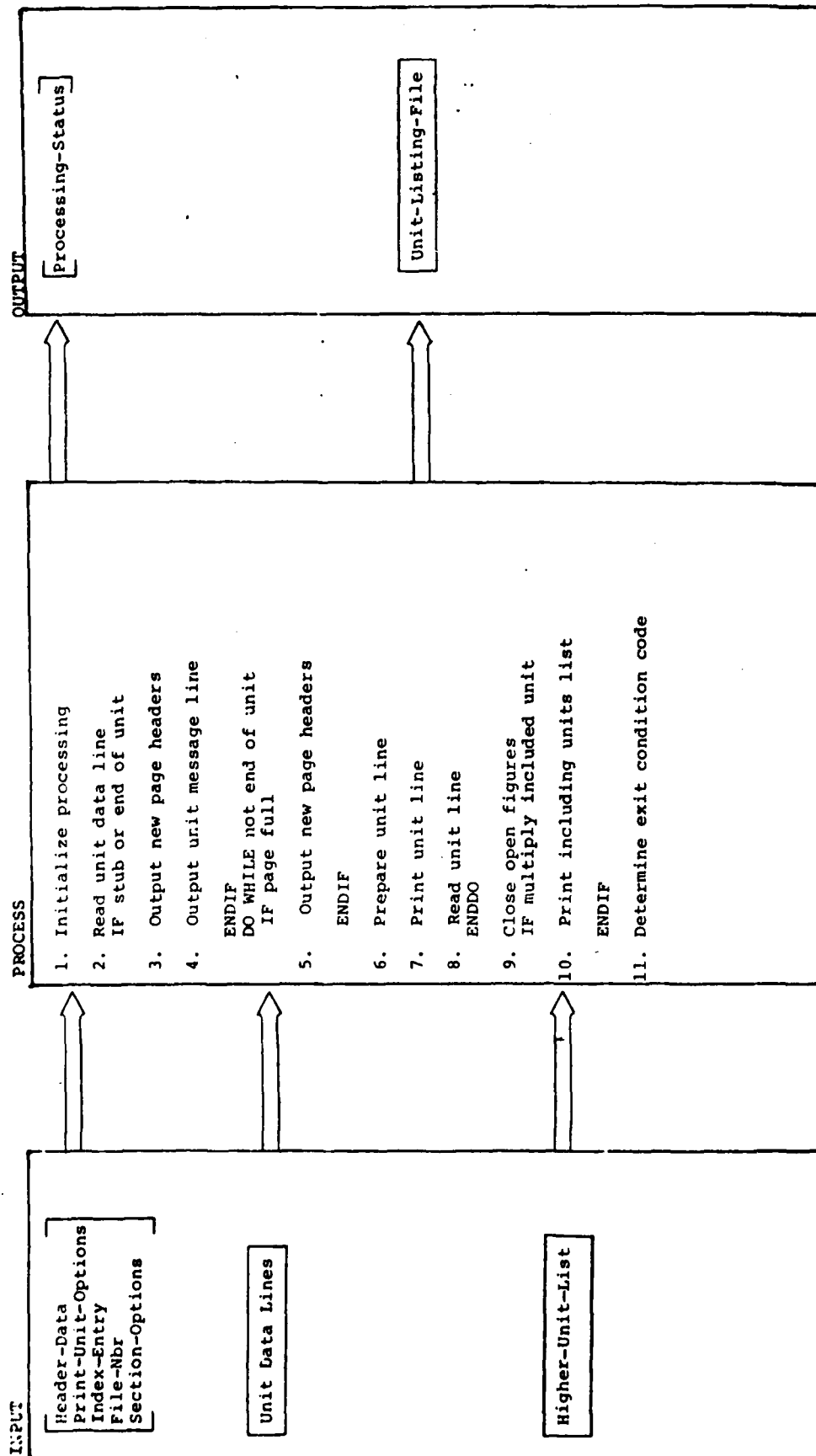
NOTES:

- (1) Unable to read line. See description of RDUN for detailed list of causes for this condition.
- (2) Unable to read block. See description of RDBK for detailed list of causes for this condition.

Diagram ID: 1.2.3.1

Name: PRCB - Print COBOL

Description: Support Routine



2.2.1.2.3.2 PRFT - Print FORTRAN

The PRFT module prints structured FORTRAN source data in an appropriately indented format.

a. Program Operations

HIPO diagrams 1.2.3.2 and 1.2.3.2.1 depict the program operations of the PRFT module. In step 2, if the end of the unit is detected, the message "UNIT HAS NO DATA LINES" will be written on the unit listing. If the unit is a STUB, the message "UNIT IS STUB" will be written on the unit listing. In step 3, the preparation of the print line is determined generally by the scanning of FORTRAN statements. Since FORTRAN is a free format language with many delimiters, each unit line is edited before indentation processing takes place. If first word of the edited FORTRAN statement is contained in the WORD-TABLE-ENTRIES of the FORTRAN-WORD-TABLE, it has an associated PROCESSING-GROUP and PROCESSING-GROUP-NBR. The PROCESSING-GROUP and PROCESSING-GROUP-NBR are used to dictate indentation and direct structure verb processing (structure, nesting and stacking). The unit line is written to the UNIT-LISTING-FILE according to the calculated indentation. Print line overflow processing is activated, if printing extends beyond column 100 on the listing line. In step 4, the INCLUDING-UNIT-LIST-PTR of the INDEX-ENTRY is used to point to the PSL-HIGHER-UNIT-BLOCK, which stores the including unit names of the unit being printed. In step 5, if a structure error has been detected, then all the error flags relating to structuring and section options are added together.

b. Data File and Table Descriptions

• UNIT-LISTING-FILE

The UNIT-LISTING-FILE is the formatted output for the listing of the FORTRAN unit being printed. See Section 3 description of data group UNIT-LISTING-RECORD.

01 ADJUSTMENTS-TO-COLUMN.

- number of print columns specified for indentation based on structure words or conditions.

05	ADJ-FOR-BLOCK	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-CASE	PIC S9(9) COMP VALUE 8.
05	ADJ-TO-CENTER-OF-PAGE	PIC S9(9) COMP VALUE 50.
05	ADJ-FOR-CONTINUATION	PIC S9(9) COMP VALUE 12.
05	ADJ-FOR-DO	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-IF	PIC S9(9) COMP VALUE 4.

```

05 ADJ-TO-LEFT-MARGIN      PIC S9(9) COMP VALUE 7.
05 ADJ-FOR-LINE-OVERFLOW   PIC S9(9) COMP VALUE 60.
05 ADJ-FOR-NEWCASE         PIC S9(9) COMP VALUE 4.
05 ADJ-FOR-UNSTRUC-DO      PIC S9(9) COMP VALUE 4.

01 COMPRESSED-STATEMENT.
    - stores edited FORTRAN statements to be processed.

05 COMP-STATEMENT-CHAR      PIC X
                             OCCURS 1320 TIMES.
01 FILLER REDEFINES COMPRESSED-STATEMENT.
05 COMP-STATEMENT-1-11      PIC X(11).
05 FILLER                   PIC X(1309).

01 CURRENT-WORD.
    - word corresponding to the first word of the compress
      statement; use for classification by search of
      FORTRAN-WORD-TABLE.

05 CURRENT-WORD-CHAR        PIC X
                             OCCURS 14 TIMES.
01 FILLER REDEFINES CURRENT-WORD.
05 CURRENT-WORD-1-6.
    88 WORD-IS-ASSIGN        VALUE "ASSIGN".
    10 CURRENT-WORD-1-4.
        88 WORD-IS-GOTO      VALUE "GOTO".
        88 WORD-IS-THEN      VALUE "THEN".
        15 CURRENT-WORD-1-2  PIC X(2).
            88 WORD-IS-DO     VALUE "DO".
            15 FILLER         PIC X(2).
        10 FILLER            PIC X(2).
05 FILLER                   PIC X(8).

01 FIGURE-STACK.
    - records the nesting of structure figures; used to
      check syntax of structure figures.

05 FILLER                   OCCURS 50 TIMES.
    10 FIGURE-IN-STACK       PIC 9.
        88 FIGURE-IN-STACK-IS-IF      VALUE 1.
        88 FIGURE-IN-STACK-IS-DO      VALUE 2.
        88 FIGURE-IN-STACK-IS-CASE    VALUE 3.
        88 FIGURE-IN-STACK-IS-BLOCK   VALUE 4.
        88 FIGURE-IN-STACK-IS-UNSTRUC-DO  VALUE 5.
    10 FIGURE-ELSE-FOUND-SW PIC 9.
        88 FIGURE-ELSE-FOUND VALUE 1.
    10 FIGURE-LABEL          PIC X(5).

```

01 FORTRAN-WORD-TABLE.

- table stores all words considered in the indentation of structured code. Each word is classified and assigned representative numbers.

05	LENGTH-OF-WORD-TABLE	PIC S9(9) COMP VALUE 55.
05	WORD-TABLE-VALUES.	
10	FILLER	PIC X(11) VALUE "DOUBLEPRECI".
10	FILLER	PIC 9 VALUE 6.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 2.
10	FILLER	PIC X(11) VALUE "EQUIVALENCE".
10	FILLER	PIC 9 VALUE 6.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 2.
10	FILLER	PIC X(11) VALUE "SUBROUTINE ".
10	FILLER	PIC 9 VALUE 6.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 2.
10	FILLER	PIC X(11) VALUE "BACKSPACE ".
10	FILLER	PIC 9 VALUE 7.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 2.
10	FILLER	PIC X(11) VALUE "BLOCKDATA ".
10	FILLER	PIC 9 VALUE 6.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 3.
10	FILLER	PIC X(11) VALUE "CHARACTER ".
10	FILLER	PIC 9 VALUE 6.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 2.
10	FILLER	PIC X(11) VALUE "DIMENSION ".
10	FILLER	PIC 9 VALUE 6.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 2.
10	FILLER	PIC X(11) VALUE "PARAMETER ".
10	FILLER	PIC 9 VALUE 6.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 4.
10	FILLER	PIC X(11) VALUE "ABNORMAL ".
10	FILLER	PIC 9 VALUE 6.
10	FILLER	PIC 9 VALUE 1.
10	FILLER	PIC 9 VALUE 2.
10	FILLER	PIC X(11) VALUE "CASEELSE ".
10	FILLER	PIC 9 VALUE 3.
10	FILLER	PIC 9 VALUE 3.
10	FILLER	PIC 9 VALUE 3.

10	FILLER	PIC X(11) VALUE "CASENTRY	".
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC X(11) VALUE "CONTINUE	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "ELSECASE	".
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "ENDBLOCK	".
10	FILLER	PIC 9 VALUE 4.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "ENDUNTIL	".
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "ENDWHILE	".
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "EXTERNAL	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "FUNCTION	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "IMPLICIT	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "NAMELIST	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "COMPLEX	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "DOUNTIL	".
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 1.	

10	FILLER	PIC X(11) VALUE "DOWHILE	".
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC X(11) VALUE "ENDCASE	".
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC 9 VALUE 4.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "ENDFILE	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "INTEGER	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "LOGICAL	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "ASSIGN	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "CASEOF	".
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC X(11) VALUE "COMMON	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "DECODE	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "ENCODE	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "FORMAT	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "INVOKE	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	

10	FILLER	PIC X(11) VALUE "RETURN	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "REWIND	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "BLOCK	".
10	FILLER	PIC 9 VALUE 4.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "ENDDO	".
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "ENDIF	".
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "ENTRY	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "PAUSE	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "PRINT	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "PUNCH	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "WRITE	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "CALL	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 1.	
10	CASE-WORD	PIC X(11) VALUE "CASE	".
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC X(11) VALUE "DATA	".
10	FILLER	PIC 9 VALUE 6.	

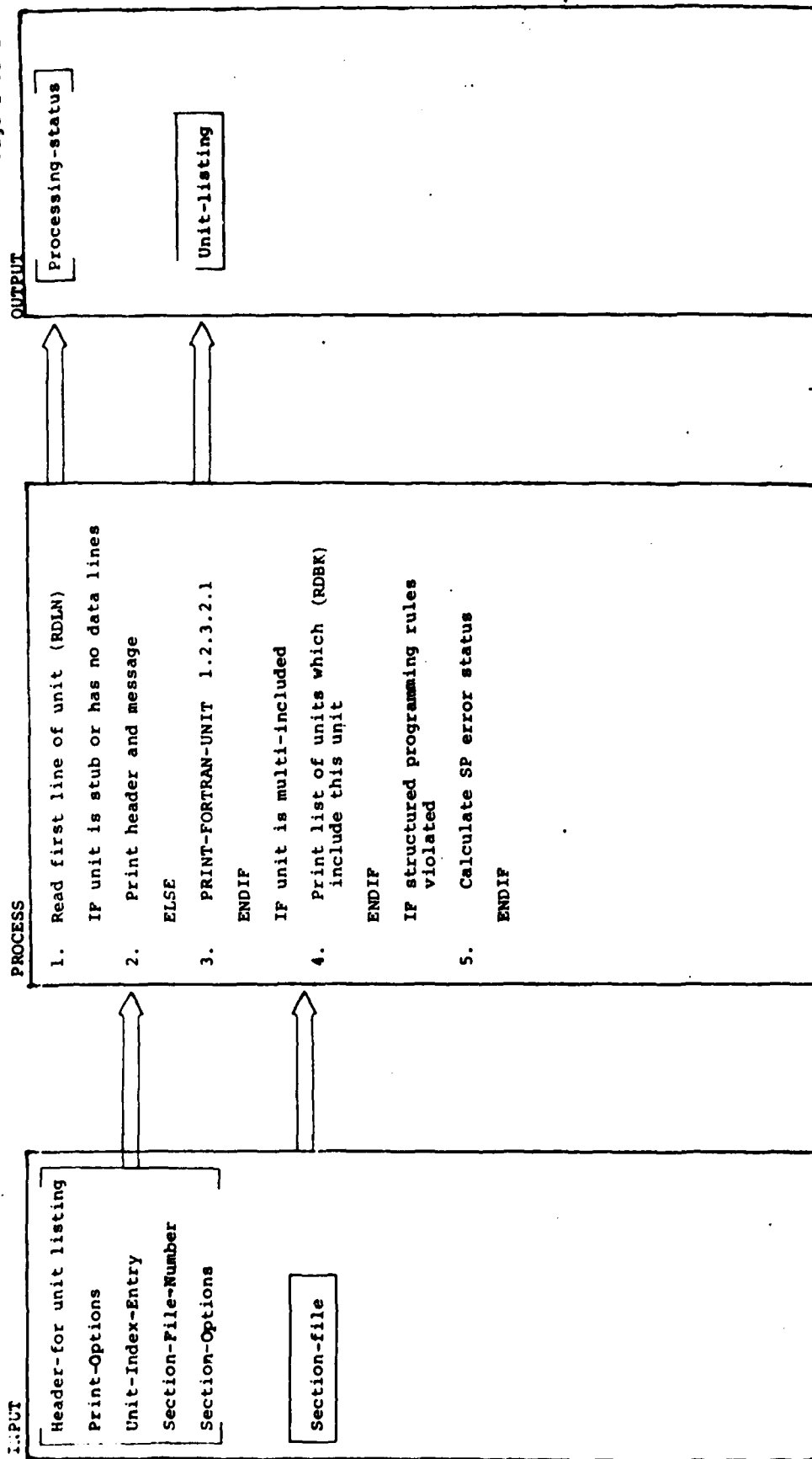
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "ELSE	".
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC 9 VALUE 3.	
10	FILLER	PIC X(11) VALUE "GOTO	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "READ	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "REAL	".
10	FILLER	PIC 9 VALUE 6.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "STOP	".
10	FILLER	PIC 9 VALUE 7.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 2.	
10	FILLER	PIC X(11) VALUE "END	".
10	FILLER	PIC 9 VALUE 8.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 3.	
10	DO-WORD	PIC X(11) VALUE "DO	".
10	FILLER	PIC 9 VALUE 5.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 6.	
10	IF-WORD	PIC X(11) VALUE "IF	".
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 1.	
10	FILLER	PIC 9 VALUE 5.	
05	FILLER REDEFINES WORD-TABLE-VALUES.		
10	WORD-TABLE-ENTRIES	OCCURS 55 TIMES	
		INDEXED BY WT-INDEX.	
15	WT-WORD.		
20	WT-WORD-CHAR	PIC X	
		OCCURS 11 TIMES	
		INDEXED BY WTW-INDEX.	
15	WT-PROC-GROUP	PIC 9.	
15	WT-PROC-GROUP-NBR	PIC 9.	
15	WT-VERIF-TEST	PIC 9.	

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1,3	48	PSL	Subsequent processing bypassed	1
4	33	PSL	Subsequent processing bypassed	2

NOTES:

- (1) Unable to read line. See description of RDUN for detailed list of causes for this condition.
- (2) Unable to read block. See description of RDBK for detailed list of causes for this condition.



INPUT

Header-for unit listing
Print-Options
Unit-Index-Entry
Section-File-Number
Section-Options

Section-file

PROCESS

1. Get first statement (RDLN)
2. DO for each FORTRAN statement
IF statement is not comment
3. Remove all blanks and scan
for punctuation
4. Determine statement type
5. Determine number of columns
to indent
6. IF SP exceptions specified
Check for SP exceptions
ENDIF
7. Adjust indentation
ENDIF
8. Write statement on listing with
page eject as needed
9. Get next FORTRAN statement (RDLN)
FNDDO
10. Check for open structure figures.

OUTPUT

Unit-listing

SP-ERROR-FLAGS

2.2.1.2.3.3 PRNI - Print with No Indentation

The PRNI module prints data with no automatic indentation. In general, unstructured data (or management data outputs for which no special print routine is implemented) are printed without changing the existing indentation.

a. Program Operations

HIPO diagram 1.2.3.3 depicts the program operations of the PRNI module. In step 4, if the end of unit is detected, the message "UNIT HAS NO DATA LINES" will be written on the unit listing by the unit message line. If the unit is a STUB the message "UNIT IS STUB" will be written on the unit listing. In step 5, the NBRLINES-OPTION of the PRINT-UNIT-OPTIONS data group is interrogated for the maximum number of lines to be printed per page. In step 8, the INCLUDING-UNIT-LIST-PTR of the INDEX-ENTRY is used to point to the PSL-HIGHER-UNIT-BLOCK, which stores the unit names, in which the unit being printed is included.

b. Data File and Table Description

All significant data descriptions are described in Section 3 of this manual.

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
2,7	48	PSL	Subsequent processing bypassed	1
8	33	PSL	Subsequent processing bypassed	2

NOTES:

- (1) Unable to read line. See description of RDUN for detailed list of causes for this condition.
- (2) Unable to read block. See description of RDBK for detailed list of causes for this condition.

Diagram ID: 1.2.3.3

Name: PRNI - Print with No Indentation Description: Support Routine

INPUT

Header-Data
Print-Unit-Options
Index-entry
File-Nbr
Section-Options

Unit Data Lines

Higher-Unit-List

PROCESS

1. Initialize processing
2. Read unit data line
IF end of unit data or stub unit
3. Output new page heading
4. Output unit message line
ENDIF
DO WHILE not end of unit
- IF new page required
5. Output new page heading
ENDIF
6. Output unit data line
7. Read unit data line
ENDDO
- IF multiply included unit
8. Output including units list
ENDIF

OUTPUT

Processing-Status

Unit-Listing-File

2.2.1.2.3.4 PRJV Print-JOVIAL

The PRJV module prints structured JOVIAL source data in an appropriately indented format.

a. Program Operations

HIPO diagram 1.2.3.4 depicts the program operations of the PRJV module. In Step 4, the unit message states whether the unit being printed is a stub unit or the unit has no lines of data. In Step 6, the first word (string of non-blank characters) is used to key the indentation process. The WORD-TABLE is searched for the presence of the first word. The presence or absence of the word in the WORD-TABLE plus the associated WORD-TYPE will determine the indentation format. If the WORD-TYPE indicates a structure verb, then its presence is recorded in the FIGURE-STACK. In the case of indentation, structure verbs and continuation lines will effect the indentation of the next unit line, as well as, the current unit line. Also in Step 6, if any structure verb being recorded in the FIGURE-STACK causes an inconsistent entry to be attempted, an error message will be generated on the unit listing. If the unit being printed resides in the PDL or SOURCE sections and the SPCHECK option has been specified, then the unit data line will be checked for adherence to structured programming principles. In Step 7, if indentation would cause a unit line to be print beyond print column 100 on the unit listing, then the unit line is left justified beginning at print column 40. In Step 9, an error message will be generated on the unit listing for every unresolved entry in the FIGURE-STACK.

b. Data File and Table Description

1. UNIT-LISTING FILE

The UNIT-LISTING file is used to format the listing of the contents of the stored PSL units. The format of the UNIT-LISTING is described in Section 3.

01 WORD-TYPE PIC S9(9) COMP.

The WORD-TYPE is assigned to each word in the WORD-TABLE. If a word being compared to the WORD-TABLE is not located, a value of "1" is assigned to the WORD-TYPE. The WORD-TYPE is use to direct processing of unit-line. The other values WORD-TYPE might take are:

- 2 - structure-verb
- 3 - conditional word

- 4 - verb
- 5 - noise-word
- 6 - include
- 7 - data declaration
- 8 - label

- 01 ADJUSTMENT-VALUES.

The ADJUSTMENT-VALUES indicate the indentation of the unit data line after the appropriate word has been detected.

05	ADJ-FOR-LABEL	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-DATA	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-IF-CODE	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-CASE-CODE	PIC S9(9) COMP VALUE 8.
05	ADJ-FOR-NEW-CASE	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-DO-CODE	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-CONDITION-CODE	PIC S9(9) COMP VALUE 8.
05	ADJ-FOR-CONDITION-WORD	PIC S9(9) COMP VALUE 4.
05	ADJ-FOR-CONTINUATION	PIC S9(9) COMP VALUE 12.
05	ADJ-FOR-OVERFLOW	PIC S9(9) COMP VALUE 60.

- 01 FIGURE-STACK.

The FIGURE-STACK is use to record the nesting of structure figures within the unit being printed

05	FILLER	OCCURS	50 TIMES
10	FIGURE-W-STACK	PIC X(3)	
10	ELSE-PREVIOUSLY-FOUND-SW	PIC S9(9) COMP.	

- 01 WORD-TABLE.

The WORD-TABLE is a fixed value, PSL defined, table of reserve words for the JOVIAL language, used to classify the first word of each unit data line for further processing.

01 WORD-TABLE.

05	FILLER	PIC X(12) VALUE"	".
05	FILLER	PIC 9 VALUE 1.	
05	FILLER	PIC 99 VALUE 1.	
05	FILLER	PIC X(12) VALUE "ADD	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "ARRAY	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	

05	FILLER	PIC X(12) VALUE "ASSIGN	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "BEGIN	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 15.	
05	FILLER	PIC X(12) VALUE "CASE	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 5.	
05	FILLER	PIC X(12) VALUE "CASEENTRY	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 4.	
05	FILLER	PIC X(12) VALUE "CLOSE	".
05	FILLER	PIC 9 VALUE 8.	
05	FILLER	PIC 99 VALUE 18.	
05	FILLER	PIC X(12) VALUE "COMMON	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "DEFINE	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "DIRECT	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 13.	
05	FILLER	PIC X(12) VALUE "DO	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 8.	
05	FILLER	PIC X(12) VALUE "ELSE	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 2.	
05	FILLER	PIC X(12) VALUE "ELSECASE	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 6.	
05	FILLER	PIC X(12) VALUE "ENCODE	".
05	FILLER	PIC 9 VALUE 8.	
05	FILLER	PIC 99 VALUE 18.	
05	FILLER	PIC X(12) VALUE "END	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 12.	
05	FILLER	PIC X(12) VALUE "ENDCASE	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 7.	
05	FILLER	PIC X(12) VALUE "ENDDO	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 9.	
05	FILLER	PIC X(12) VALUE "ENDIF	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 3.	

05	FILLER	PIC X(12) VALUE "FILE	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "FOR	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "GOTO	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "IF	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 1.	
05	FILLER	PIC X(12) VALUE "IFEITH	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 10.	
05	FILLER	PIC X(12) VALUE "IN	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "INCLUDE	".
05	FILLER	PIC 9 VALUE 6.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "INPUT	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "IN/OUT	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "IO	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "ITEM	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "JOVIAL	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 14.	
05	FILLER	PIC X(12) VALUE "MODE	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "MONITOR	".
05	FILLER	PIC 9 VALUE 8.	
05	FILLER	PIC 99 VALUE 18.	
05	FILLER	PIC X(12) VALUE "OPEN	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "ORIF	".
05	FILLER	PIC 9 VALUE 2.	
05	FILLER	PIC 99 VALUE 11.	
05	FILLER	PIC X(12) VALUE "OUT	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	

05	FILLER	PIC X(12) VALUE "OUTPUT	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "OVERLAY	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "PROC	".
05	FILLER	PIC 9 VALUE 8.	
05	FILLER	PIC 99 VALUE 18.	
05	FILLER	PIC X(12) VALUE "PROGRAM	".
05	FILLER	PIC 9 VALUE 8.	
05	FILLER	PIC 99 VALUE 18.	
05	FILLER	PIC X(12) VALUE "REMQUO	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "RETURN	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "SHUT	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "START	".
05	FILLER	PIC 9 VALUE 8.	
05	FILLER	PIC 99 VALUE 18.	
05	FILLER	PIC X(12) VALUE "STOP	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "STRING	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "SWITCH	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "TABLE	".
05	FILLER	PIC 9 VALUE 7.	
05	FILLER	PIC 99 VALUE 17.	
05	FILLER	PIC X(12) VALUE "TERM	".
05	FILLER	PIC 9 VALUE 8.	
05	FILLER	PIC 99 VALUE 18.	
05	FILLER	PIC X(12) VALUE "TEST	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "WAIT	".
05	FILLER	PIC 9 VALUE 4.	
05	FILLER	PIC 99 VALUE 19.	
05	FILLER	PIC X(12) VALUE "%INCLUDE	".
05	FILLER	PIC 9 VALUE 6.	
05	FILLER	PIC 99 VALUE 19.	
01	FILLER REDEFINES WORD-TABLE.		
05	WORD-TABLE-ENTRIES	OCCURS 53 TIMES ASCENDING KEY IS WORD-IN-TABLE INDEXED BY WORD-INDEX.	

```

10 WORD-IN-TABLE          PIC X(12).
10 WORD-TYPE-IN-TABLE     PIC 9.
10 WORD-TYPE-IN-TABLE     PIC 99.
* THIS REDEFINITION ASSUMES THAT A FRESH COPY OF THE MODULE
* WILL BE LOADED BEFORE EACH CALL
01 PSL-HIGHER-UNIT-BLOCK REDEFINES WORD-TABLE
   COPY PSL-HIGHER-UNIT-BLOCK.

```

c. Branching and Error Conditions

Function Ref.	Condition Code	Message Category	Program Action	Note
2	48	PSL	Subsequent processing by passed	1
	18	N/A	end-of-unit code returned to caller	
10	33	PSL	file not assigned	2
11	N/A	N/A	any structure errors flag are added to the processing status	

Note:

- (1) Unable to read a line from the PSL library stored unit. See description of RDUN for the specific cause of "Unable to read line for unit" condition.
- (2) Unable to read a block of the PSL library section file see description of ACBK for specific causes of "Unable to read a block" condition.

Diagram ID: 1.2.3.4

INPUT

Header-for-unit-listings
Print-unit-options
Index-entry
File-number
Section-options

Unit-date-lines

Higher-unit-list

Name: PRJV - Print JOVIAL

PROCESS

1. Initialize processing
2. Read unit data line
- IF stub or end-of-unit
3. Output new-page-header
4. Output unit-message-line
- ENDIF
- DO while not (end-of-unit or
- end-of-index)
- IF page full
5. Output new-page-header
- ENDIF
6. Prepare unit line
7. Print unit line
- IF no error
8. Read unit data line
- ENDIF
- ENDDO
- IF end-of-unit or end-of-index
9. Close open figures
- IF unit multiple included
10. Print including units
- ENDIF
11. Determine return-code status
- ENDIF

Description: Support Routine

OUTPUT

Processing-Status

Unit-listing-file

2.2.1.2.4 Top-Down-Read

Top-down-read support is provided to sequentially read the source code of a module including the code contained in units referenced by INCLUDE statements. The program modules which perform this type of operation are described below.

2.2.1.2.4.1 RDCM - Read for Compile

The RDCM module is repetitively called to read the data lines of a given top unit whose code may reference INCLUDED units residing in multiple project libraries. Lines of code are sequentially retrieved from the top and included units and returned to the calling program.

a. Program Operations

HIPO diagram 1.2.4.1 depicts the program operations performed by the RDCM module. If this is the initial time called, the Keyword Card file is opened for input and read to process the keyword OBJECT which denotes that the source code is to be compiled and its object code stored, the UNIT keyword value which gives the name of the top unit and the LIBSEC keyword value which gives the name of a project library to be searched for the top unit and its included code. If an OBJECT keyword card is processed, a message specifying the project library section and unit name in which the object code will be stored is printed. The named unit is then searched for in the project library source section(s) designated on the processed LIBSEC card(s). Information specifying the project library section in which the unit was found, its current version/modification, and last date of update is returned to the calling program in conjunction with a start-of-unit status code.

The STACK-PTR value is incremented to one in the preceding operation and the unit name is moved to STACK-UNIT-NAME and zero is moved to the STACK-LINE-NBR as the first entry of the STACK-TABLE. When the RDCM module is next called, a line of code from the top unit is read and the STACK-LINE-NBR is incremented by one to indicate the current line number being processed for the unit at that STACK-PTR position, and the retrieved line of code is returned to the calling program. If a line contains an INCLUDE statement, the unit referenced in the INCLUDE statement is found using the same search procedures employed for the top unit. After checking to see that the INCLUDED unit is not already in the Stack Table (which would indicate an INCLUDED unit error condition), the STACK-PTR is incremented by one and the unit name is stored, its STACK-LINE-NBR initialized to zero and return made with a start-of-unit status code. The INCLUDED unit is next read and if it contains any INCLUDE statements, the foregoing procedure is repeated and the STACK-PTR value is incremented to reflect the current level of INCLUDE statement nesting and point to the current unit and line number being read. When the end of the unit is reached for the current unit, the STACK-PTR is decremented by one and return

is made to the calling program with an end-of-unit status code set, except that when the STACK-PTR is decremented to zero, an end-of-module status code is returned and all PSL section files previously assigned for the library search are released.

It will be noted that when in the search for an INCLUDED unit a stub unit is found, the search is continued until a real unit is found or all the libraries are examined. This action is taken to ensure that a PSL-generated stub in one library does not prevent a real unit, maintained by the user in a second library, from being found. Only when two (or more) versions of a real unit exists in multiple project libraries does the order of library scan determine the version which will be used.

b. Data File and Table Descriptions

The following data files and/or tables are of special significance in program operations of the RDCM module:

01	STACK-TABLE.	
05	STACK-ENTRY	OCCURS 50 TIMES.
10	STACK-UNIT-NAME	PIC X(30).
10	STACK-LINE-NBR	PIC S9(9) COMP.
01	STACK-PTR	PIC S9(9) COMP.
88	STACK-IS-EMPTY	VALUE ZERO.
88	STACK-IS-FULL	VALUE 50.

The use of the preceding STACK-TABLE and STACK-PTR data entries is discussed under program operations. It will be noted that the INCLUDED unit nesting limit is set at fifty (50). This limit is set much beyond what would normally be encountered (even in very large programs) to preclude reaching that limit and giving an error return.

c. Branching and Error Conditions

The following branching and error conditions apply to HIPO diagram 1.2.4.1:

Function Reference	Condition Code	Message Category	Program Action	Note
3	77	MSG	Set end-of-module status	
	78	MSG	Set end-of-module status	
4	76	MSG	Set initialize-error status	1

c. Branching and Error Conditions (continued)

<u>Function Reference</u>	<u>Condition Code</u>	<u>Message Category</u>	<u>Program Action</u>	<u>Note</u>
6	79	MSG	Set include-error status	
	80	MSG	Set include-error status	
7	77	MSG	Set include-error status	
	78	MSG	Set include-error status	
9	76	MSG	Set initialize-error status	1

NOTES:

- (1) This is a secondary error caused by an error return from the ITRD module where a PSL error message is issued describing the primary cause of error.

Diagram ID: 1.2.4.1

Name: RDCM - Read for Compile

Description: Support Routine

INPUT

Keyword Card File
Multi-library Sections

PROCESS

- IF no units are in stack
1. Read keyword/value file.
2. Save unit name and library-section names.
3. Find unit in libraries
4. Initialize unit
ELSE
5. Get unit pointers from stack
6. Read next line of unit
IF line is INCLUDE
7. Find unit in libraries
IF stub
8. Set status to stub
ENDIF
IF real
9. Initialize unit
ENDIF
ENDIF
IF end-of-unit
IF not end-of-module
10. Status is unit-finished
Remove unit from stack
IF stack is empty
11. Status is end-of-module
ENDIF
ENDIF
IF end-of-module
12. Release all files
ENDIF

OUTPUT

Line of data
Unit line number
Unit name
Project name
Library name
Unit Version Modif.
Unit Update Data Time
Processing Status

2.2.1.2.4.2 RDPS - Read for Program Structure

The RDPS module is repetitively called to scan the data lines of a given top unit whose code may reference INCLUDED units residing in multiple project libraries. INCLUDED and CALLED unit references are found and accounting information is returned to the calling program.

a. Program Operations

HIPO diagram 1.2.4.2 depicts the program operations of the RDPS module which utilizes unit search and scan procedures that are equivalent to the RDCM module operations previously described. One difference in RDPS module procedures is that only the keyword LIBSEC is processed from the Keyword Card file while the calling program (essentially limited to the PRPS module) provides the name of the top unit through a USING parameter of the RDPS module. Another difference is that, although lines of code are sequentially read as in the RDCM module, return to the calling program is made only when an INCLUDE or a CALL statement is found. A further difference is noted in the processing of the INCLUDED unit reference. That is, when the included unit type indicates a multi-included unit, the included unit code is not scanned. However, the calling program may, as accomplished in PRPS, later pass the name of the multi-included unit to RDPS as the "top unit" to further scan its code for INCLUDE and CALL statements. A final difference is that when a CALL statement is found, only the referenced unit name is returned to the calling program (i.e., no accounting information is returned). Again, the referenced unit may later have its name passed to RDPS as a "top unit" to start a scan of its code, if such is required. Beyond these differences the program operations of the RDPS module are essentially the same as the RDCM module.

b. Data File and Table Descriptions

Same as RDCM module description under paragraph 2.2.1.2.4.1.

c. Branching and Error Conditions

Same as RDCM module description under paragraph 2.2.1.2.4.1.

Diagram ID: 1.2.4.2

Name: RDPS - Read for Program Structure

Description: Support Routine

INPUT

Top unit name
Keyword Card File

PROCESS

```
IF no units are in stack
1. Read keyword card file
2. Save library-section names
3. Find unit in libraries
4. Initialize unit
   ENDIF

DO WHILE not unit break
5. Get unit pointers from stack
6. Read line of unit
   IF line is INCLUDE
7. Find unit in libraries
   IF stub or multi-included
8. Set status accordingly
   Indicate unit break
   Get accounting information
   ENDIF
   IF real
9. Initialize unit
   Indicate unit break
   ENDIF
   ENDIF
   IF line is CALL
10. Indicate unit break
   ENDIF
   IF end-of-unit
11. IF stack is empty
   Status is end-of-module
   ELSE
12. Status is unit-finished
   Remove unit from stack
   ENDIF
   ENDIF
   ENDDO
   IF end-of-module
13. Release all files
   ENDIF
```

OUTPUT

Unit name
Project name
Library name
Unit type
Accounting information
Data line status

2.2.1.2.5 Process Message

Message processing support is provided in three separate print message operations. The Print Message (PRMS) module exclusively supports the operations under BCTL. A complete set of standard error messages is made available through the subordinate PRM1, PRM2, PRM3 and PRM4 modules. The PRMSI module is equivalent to the PRMS module except that a lesser number of standard message is required by the spawned job operations which it supports. The Print Errors (PRER) module is utilized in addition to the PRMSI module in spawned job operations to support non-standard error message output requirements.

2.2.1.2.5.1 PRMS - Print a Message

The module PRMS is responsible for printing the majority of messages generated by the PSL System. The module PRMS writes messages, sequentially, to the MESSAGE-FILE as it is called. The PRMS module calls one of four subroutine modules (PRM1, PRM2, PRM3 and PRM4) in order to build the printable messages. Each PSL message is composed of a three character message class code, a message number, stored message text and may contain data supplied by the calling routine. The calling routine is responsible for supplying the message number and any message data to the module PRMS.

a. Program Operations

HIPO diagram 1.2.5.1 describes the program operation. In step 2, the MESSAGE-NUMBER-TABLE is searched for entry number which matches the MESSAGE-NUMBER. The matching table entry is composed of a message class code and a message text code. The message class code from the table entry is translated into a three character code for printing by the CLASS-TABLE. In step 4 through step 10, the message text code determines which subroutine is selected to build the remaining portion of the print message. In step 8 through step 10, the subroutine which handles multiple line messages is called for each line of the message. In step 11, if a message number has no entry in the MESSAGE-NUMBER-TABLE, see Appendix for description, a PSL message is built containing supplied message data. In step 12, the message line built by the subroutines is written to the MESSAGE-FILE.

b. Data File and Table Descriptions

1. INPUT ARGUMENTS

MESSAGE-NUMBER PIC S9(9) COMP.

- Number which corresponds to message text as it appears in Appendix C in User's Manual.

MESSAGE-DATA PIC X(100)

- Data to be inserted into message supplied by the calling routine.

MESSAGE-FILE

This file contains the printed messages generated by the PSL System executions.

Diagram ID: 1.2.5.1

Name: PRMS - Print a Message

Description: Support Routine

INPUT

MESSAGE-NUMBER
MESSAGE-DATA

▲ MESSAGE-NUMBER-TABLE

▲ Print-msg-line

▲ MESSAGE-LENGTH

PROCESS

```
IF line count greater than 53
1. write header line
  ENDIF
2. Search message-number-table for
   message-class
3. Move message-class to print-msg-line
   Casentry text description from
   message-number-table
   → Case 1, 2, 6
4. Initialize print-type-1 PRM1E 1.2.5.1.1
   Case 3, 5
5. Initialize print-type-2 PRM2E 1.2.5.1.2
   Case 4
6. Initialize print-type-3 PRM3E 1.2.5.1.3
   Case 7
7. Move 2 to message-length
   → DO UNTIL message-length is 0
8. Initialize print-type-4 PRM4E 1.2.5.1.4
9. Write print-line-4
10. Subtract 1 from message-length
    ENDDO
    Case 8
11. Initialize print line with
    message-data only
    ENDCASE
IF text description code not = 7
12. Write print-line
    ENDIF
```

OUTPUT

▲ MESSAGE-LINE

MESSAGES

MESSAGE-FILE

PRINT-MSG-LINE

• 01 PRINT-MSG-LINE

05 PRINT-MSG-LINE

10 FILLER PIC X(3).

10 PRINT-ID PIC X(3).

- Message class three letter code.

10 PRINT-MSG-NBR PIC X(3).

- Message number

10 FILLER PIC X(2).

10 PRINT-TEXT-DATA PIC X(109).

- Contains the body of the message with
text and data.

MESSAGE-LINE PIC X(109).

- Contains body of message with text and data
built by appropriate subroutine.

• 01 CLASS-TABLE.

05 FILLER PIC X(3) VALUE "ADV".

05 FILLER PIC X(3) VALUE "ERR".

05 FILLER PIC X(3) VALUE "FMS".

05 FILLER PIC X(3) VALUE "INF".

05 FILLER PIC X(3) VALUE "PSL".

05 FILLER PIC X(3) VALUE "UND".

05 FILLER PIC X(3) VALUE SPACE.

01 FILLER REDEFINES CLASS-TABLE.

05 CLASS-NAME OCCURS 7 TIMES PIC X(3).

- Contains three character codes for message class.

c. Branching and Error Conditions

NOT APPLICABLE POINT MESSAGE ROUTINES.

2.2.1.2.5.1.1 PRM1 - Print Type 1 Message

The module PRM1 is responsible for building a PSL message in which; 1) the first ten characters of the message is comprised of stored message text, 2) the message body requires a unique print format, and 3) the message is composed solely of stored message text. The calling routine is responsible for supplying the MESSAGE-NUMBER and MESSAGE-DATA. The module PRM1 outputs the constructed message line. The MESSAGE-NUMBER is the connecting element between the predefined print line formats, stored message text and the format of input message data.

a. Program Operations

HIPO diagram 1.2.5.1.1 describes the program operation. In step 2, if the print line format requires no input message data, only the related stored message text is used to build the message. In step 4, if the print line format specifies that the first ten characters of the message contains stored message text, then the appropriate message text and message data combination is used to build the message. In step 6, if the print line format has a unique layout, the appropriate stored message text and message data combination is used to build the message.

b. Data File and Table Descriptions

1. INPUT ARGUMENTS

MESSAGE-NUMBER PIC S9(9) COMP.

- number which corresponds to return code of calling routine.

MESSAGE-DATAA PIC X(100).

- data associated with return code condition format determined by message number.

2. OUTPUT ARGUMENTS

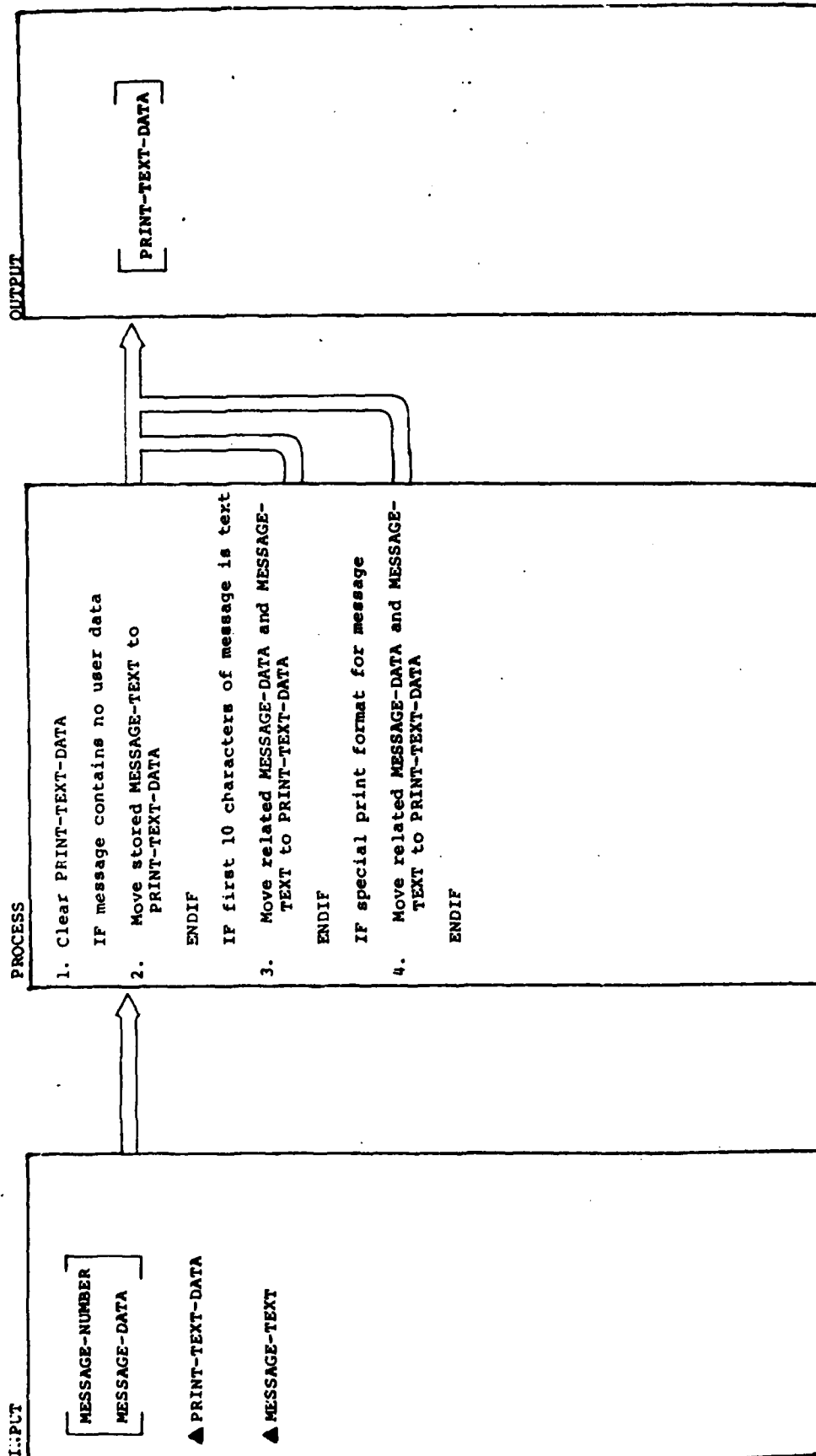
PRINT-TEXT-DATAA PIC X(109).

- print line format for message format determined by message number.

c. Branching and Error Conditions

NOT APPLICABLE FOR PRINT MESSAGE ROUTINES.

Diagram ID: 1.2.5.1.1 Name: PRMIE - Print Type 1 Message Description: Support Routines



2.2.1.2.5.1.2 PRM2 - Print Type 2 Message

The module PRM2 is responsible for building a PSL message in which the first twenty characters or the first fifty characters of the message is comprised of stored message text. The calling routine is responsible for supplying the MESSAGE-NUMBER and MESSAGE-DATA. The module PRM2 outputs the constructed message line. The MESSAGE-NUMBER is the connecting element between the predefined print line format, the store message text, and the format of the input message data.

a. Program Operations

HIPO diagram 1.2.5.1.2 describes the program operation. In step 2, if the print line format requires that the first twenty characters of the message consist of stored message text, then the appropriate message text and message data combination is used to build the message. In step 3, if the print line format requires that the first fifty characters of the message consist of stored message text, then the appropriate message text and message data combination is used to build the message.

b. Data File and Table Descriptions

1. INPUT ARGUMENTS

MESSAGE-NUMBER PIC S9(9) COMP.

- number which corresponds to return code of calling routine.

MESSAGE-DATAA PIC X(100).

- data associated with return code condition format determined by message number.

2. OUTPUT ARGUMENTS

PRINT-TEXT-DATAA PIC X(109).

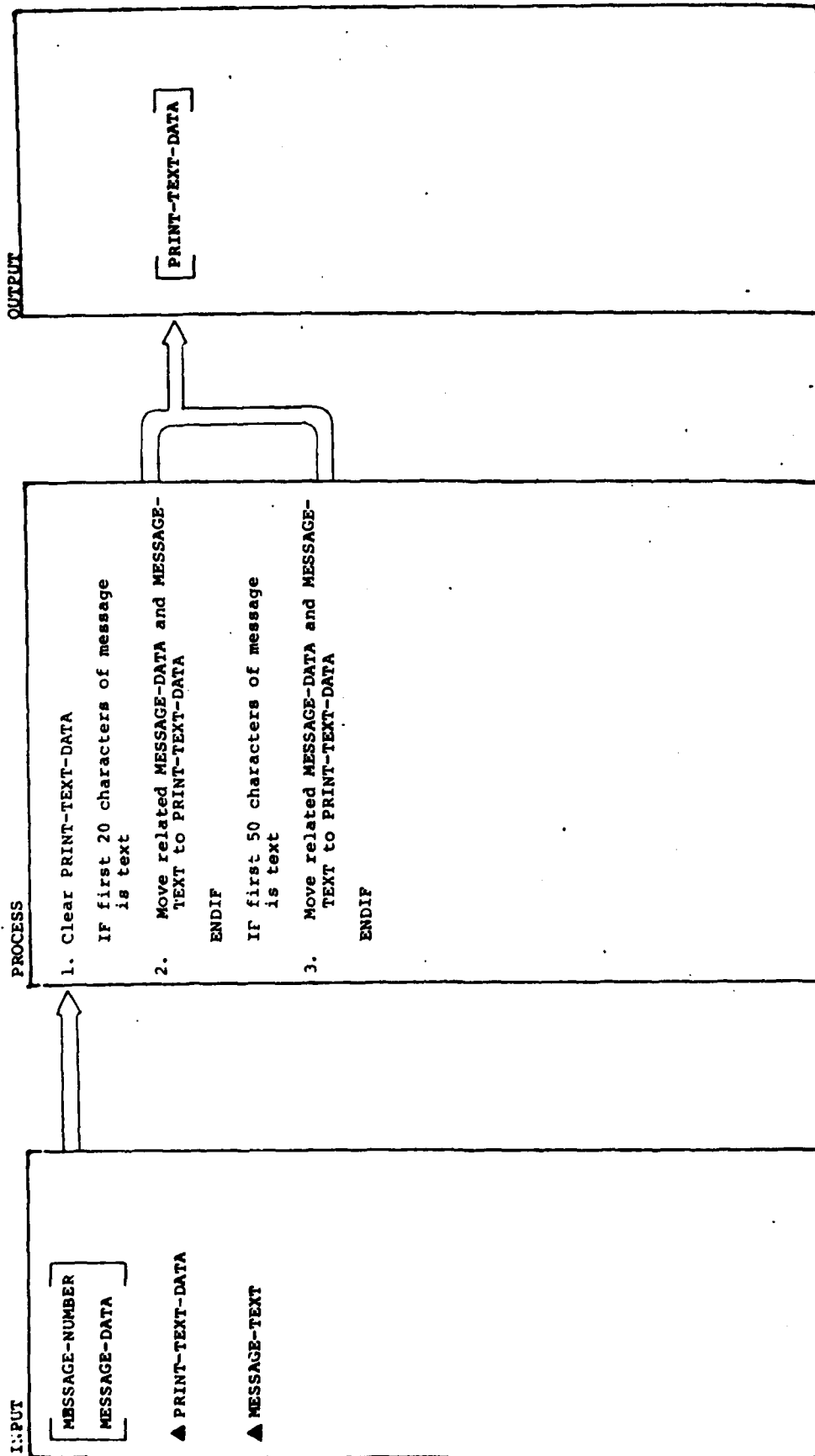
- print line format for message format determined by message number.

c. Branching and Error Conditions

NOT APPLICABLE FOR PRINT MESSAGE ROUTINES.

Diagram ID: 1.2.5.1.2

Name: PRM2E - Print Type 2 Message Description: Support Routines



2.2.1.2.5.1.3 PRM3 - Print Type 3 Message

The module PRM3 is responsible for building a PSL message in which the first 35 characters of the message is comprised of stored message text. The calling routine is responsible for supplying the MESSAGE-NUMBER and MESSAGE-DATA. The module PRM3 outputs the constructed message line. The message number is the connecting element between the predefined print line format, stored message text, and the format of the input message data.

a. Program Operations

HIPO diagram 1.2.5.1.3 describes the program operation. In step 2, if the print line format specifies that the first 35 characters of the message contains stored message text, then, the appropriate message text and message data combination is used to build the message line.

b. Data File and Table Descriptions

1. INPUT ARGUMENTS

MESSAGE-NUMBER PIC S9(9) COMP.

- number which corresponds to return code of calling routine.

MESSAGE-DATAA PIC X(100).

- data associated with return code condition format determined by message number

2. OUTPUT ARGUMENTS

PRINT-TEXT-DATAA PIC X(109).

- print line format for message format determined by message number.

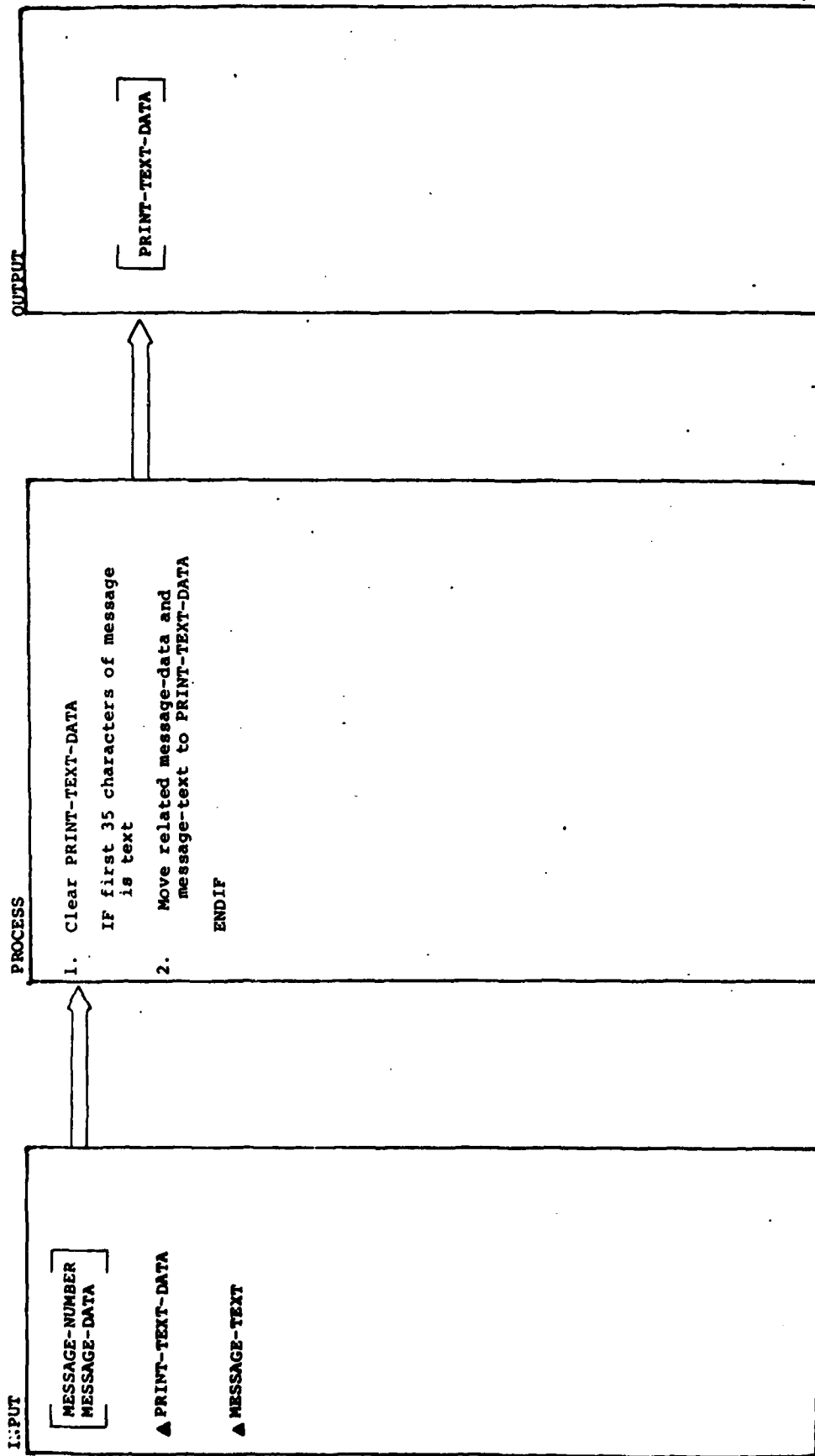
c. Branching and Error Conditions

NOT APPLICABLE FOR PRINT MESSAGE ROUTINES.

Diagram ID: 1.2.5.1.3

Name: PRM3E - Print Type 3 Message

Description: Support Routine



2.2.1.2.5.1.4 PRM4 - Print Type 4 Message

The module PRM4 is responsible for building multiple line PSL messages. The print line format of module PRM4 is a composite of the PRINT-TEXT-DATA definitions described in modules PRM1, PRM2, and PRM3. The calling routine is responsible for supplying the MESSAGE-NUMBER, MESSAGE-DATA and MESSAGE-LENGTH. The module PRM4 outputs the constructed message line. The message number is the connecting element between the predefined print line format, stored message text, and the format of the input message data.

a. Program Operations

HIPO diagram 1.2.5.1.4 describes the program operation. In step 2, the first message line is initialized with the message text and message data that are associated with the MESSAGE-NUMBER. In step 3, if the message number is one which supplies file and catalog data, turn the FILE-STRING-SW on. In step 4, if the FILE-STRING-SW is on, initialize the second message line with file string message data and file string print format. In step 6, initialize the second message line with the message data and stored message text associated with the message-number.

PGMR-NAME

- name of programmer performing update.

MOD-NBR

b. Data File and Table Descriptions

1. INPUT ARGUMENTS

MESSAGE-NUMBER PIC S9(9) COMP.

- number which corresponds to return code of calling routine.

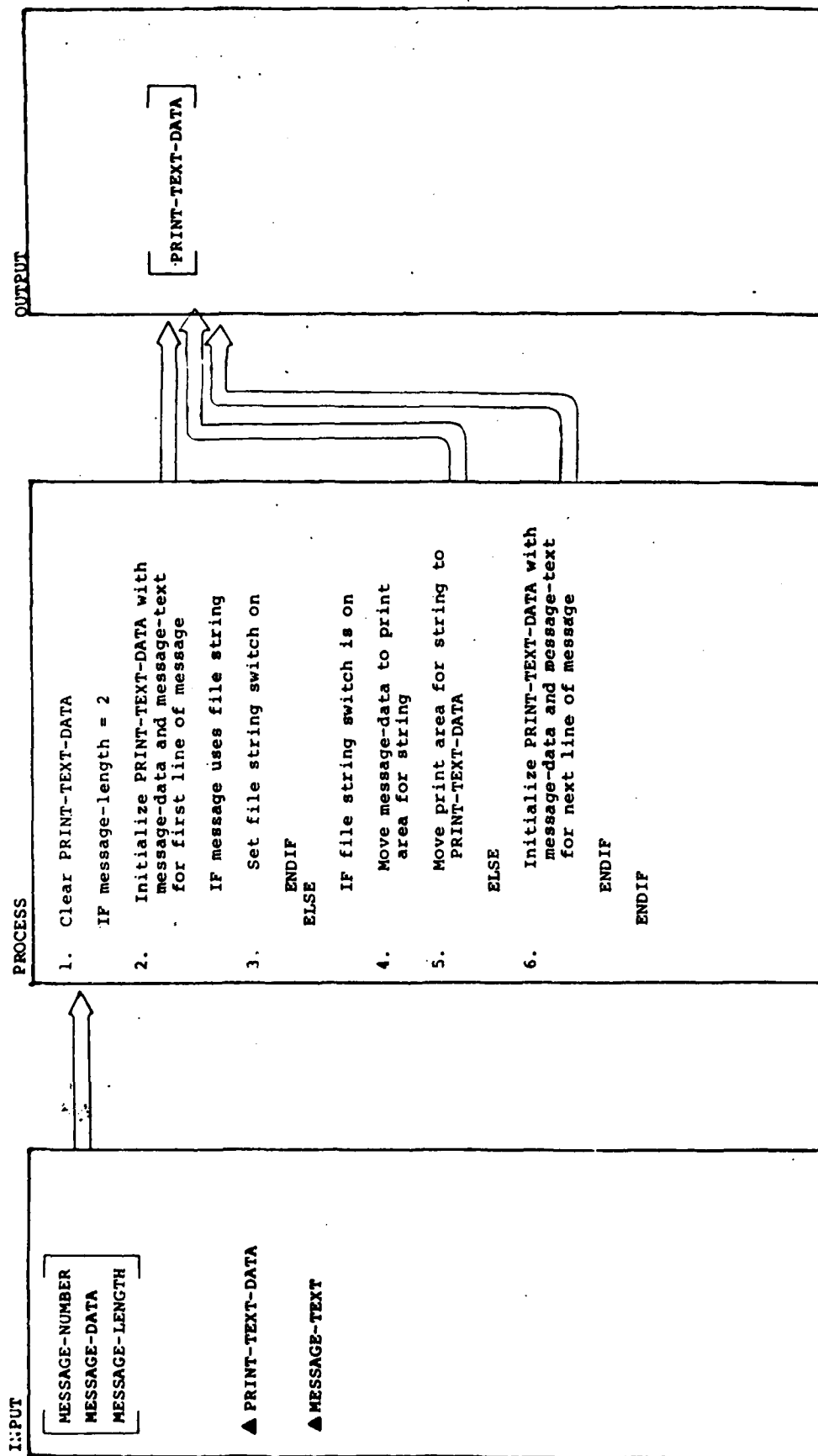
MESSAGE-DATA PIC X(100).

- data associated with return code condition format determined by message number.

Diagram ID: 1.2.5.1.4

Name: PRM4E - Print Type 4 Message

Description: Support Routine



MESSAGE-LENGTH PIC S9(9) COMP.

- the number of lines to be printed in
multi-line message.

PRINT-TEXT-DATAA PIC X(109).

- print line format for message format
determined by message number.

c. Branching and Error Conditions

NOT APPLICABLE FOR PRINT MESSAGE ROUTINES.

2.2.1.2.5.2 PRER - Print Errors

The PRER module is utilized by PSL spawned job modules to output general error messages for which no standard text has been pre-determined.

a. Program Operations

HIPO diagram 1.2.5.2 depicts operations of the PRER module. The input parameters specify the content of the required error message. These inputs are formatted for output to an error message listing file.

b. Data File and Table Descriptions

1. Message Listing File

```
01 PRINT-LINE.  
   05 MSG-LABEL-FIELD      PIC X(4).  
   05 MSG-NBR-FIELD        PIC Z(4)  
   05 FILLER                PIC X(2).  
   05 UNIT-LABEL-FIELD     PIC X(5).  
   05 UNIT-NAME-FIELD      PIC X(30).  
   05 FILLER                PIC X(2)  
   05 LINE-LABEL-FIELD     PIC X(5).  
   05 LINE-NBR-FIELD       PIC Z(5).  
   05 FILLER                PIC X(63).
```

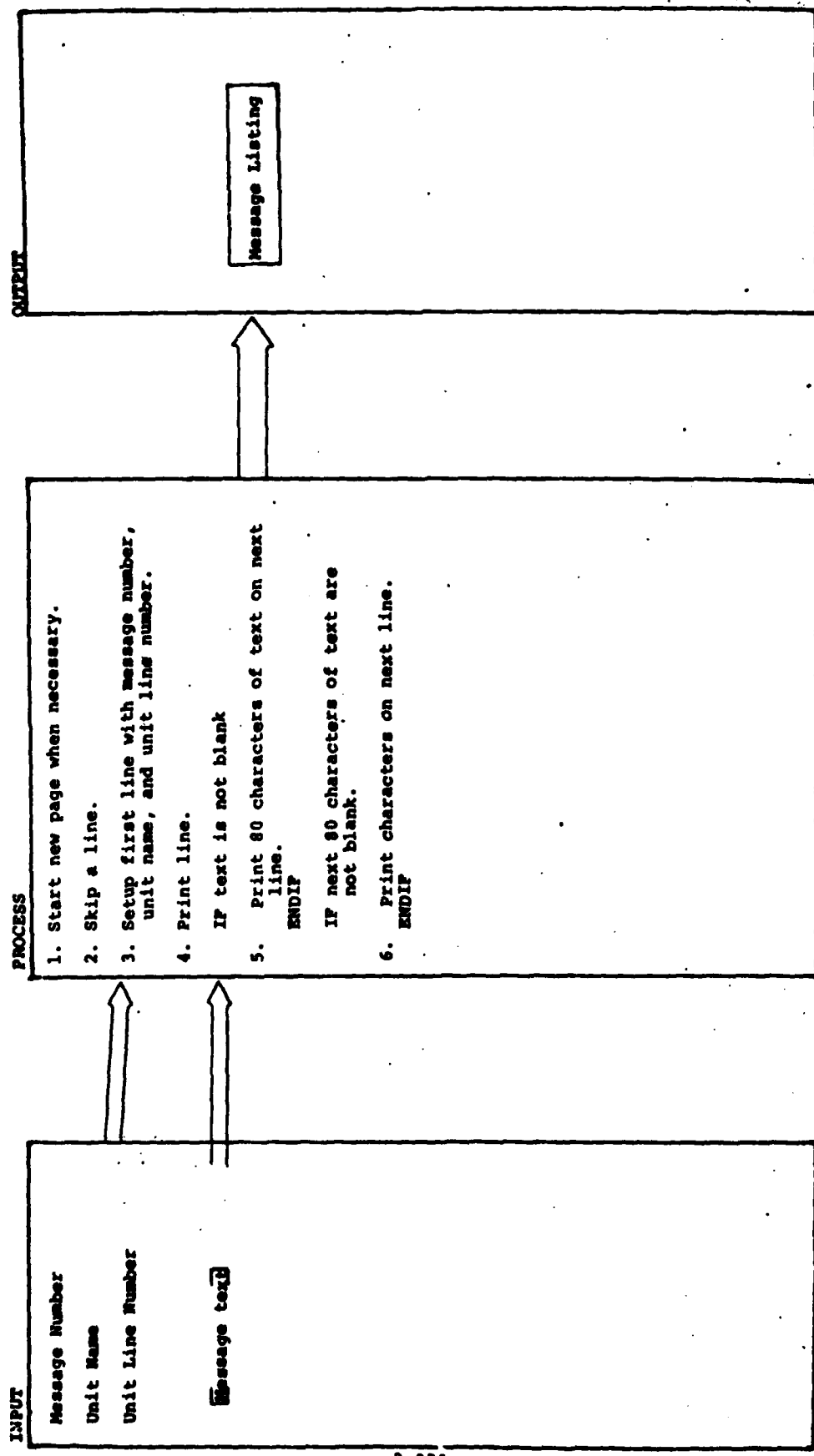
c. Branching and Error Conditions

Not applicable.

Diagram ID: 1.2.5.2

Name: PRER - Print Error

Description: Support Routine



2.2.1.2.5.3 PRMSI - Print Message in Spawned Job

The module PRMSI is responsible for printing the messages generated by the PSL System in a spawned job environment. The module PRMSI is designed to write messages sequentially, to the MESSAGE-FILE as it is called. The modules which utilize the PRMSI module in the spawned job environment are those modules that are contained in the PSL Procedures described in Appendix D of the User's Manual. The calling routines are responsible for supplying the MESSAGE-NUMBER and MESSAGE-DATA. The MESSAGE-NUMBER is the connecting element between the pre-defined print line format, stored message text and the format of the input message data.

a. Program Operations

HIPO diagram 1.2.5.3 describes the program operation. In step 1, if the MESSAGE-NUMBER is not defined under the message categories listed in Appendix C of the User's Manual, an "undefined" message written to the MESSAGE-FILE. In step 3, if the message to be written consist of multiple lines, each line is initialized and written, singularly. In step 6, the MESSAGE-NUMBER determines the PRINT-TEXT-DATA format, the stored message text, and the MESSAGE-DATA definition with which to initialize message line. The message line processing is similar to the processing of modules PRM1, PRM2, PRM3 and PRM4.

b. Data File and Table Descriptions

1. INPUT ARGUMENTS

MESSAGE-NUMBER PIC S9(9) COMP.

- number which corresponds to message text as it appears in Appendix C in User's Manual.

MESSAGE-DATA PIC X(100).

- data to be inserted into message supplied by the calling routine.

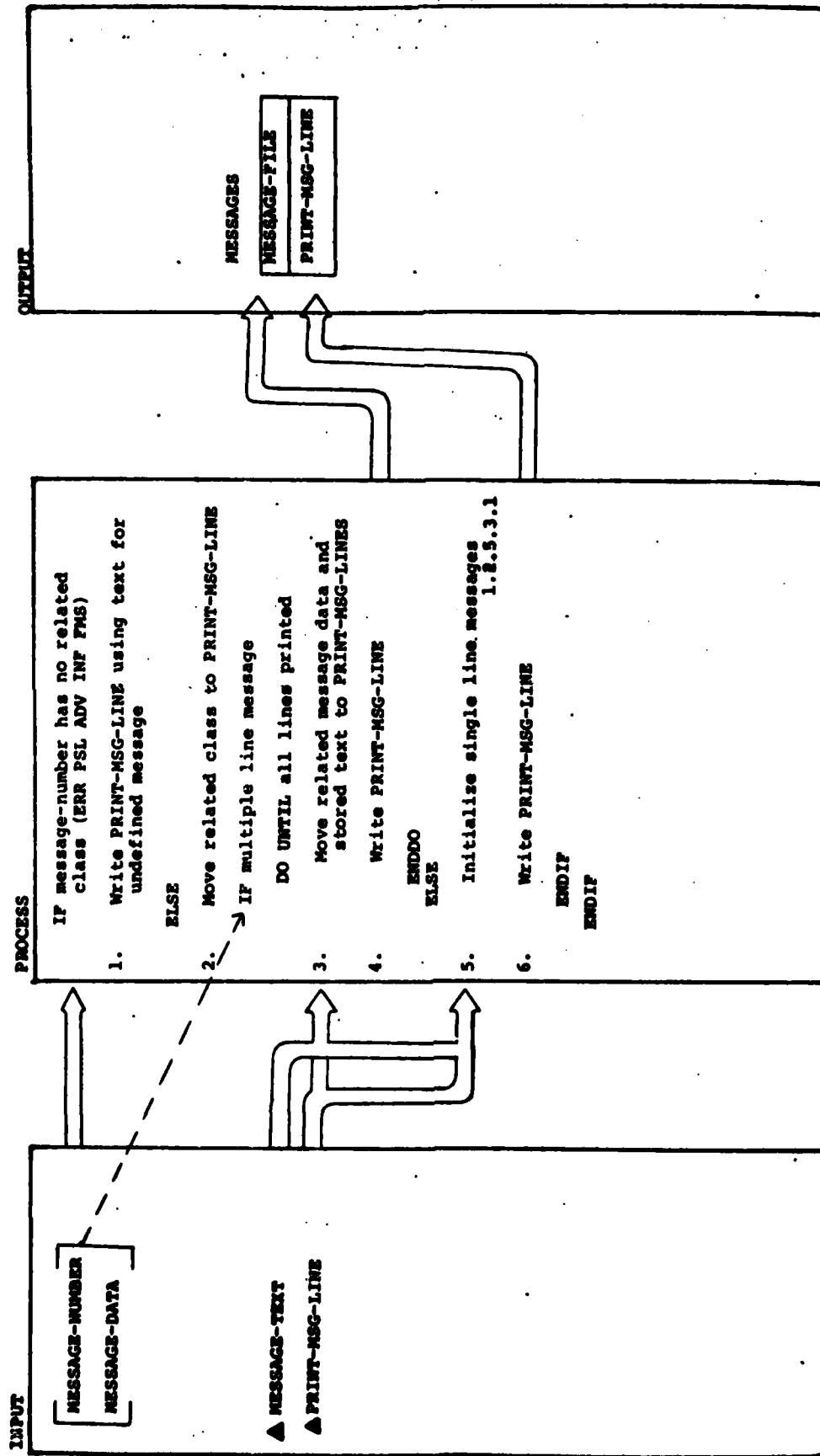
MESSAGE-FILE

- This file contains the printed messages generated by the PSL System executions.

Diagram ID: 1.2.5.3

Name: PRMSI - Print Message in Spawned Job

Description: Support Routine



• 01 PRINT-MSG-LINE

05 PRINT-MSG-LINE

10 FILLER PIC X(3).

10 PRINT-ID PIC X(3).

- Message class three letter code.

10 PRINT-MSG-NBR PIC X(3).

- Message number

10 FILLER PIC X(2).

10 PRINT-TEXT-DATA PIC X(109).

- Contains the body of the message with text and data.

MESSAGE-LINE PIC X(109).

- contains body of message with text and data built by appropriate subroutine.

• 01 CLASS-TABLE.

05 FILLER PIC X(3) VALUE "ADV".

05 FILLER PIC X(3) VALUE "ERR".

05 FILLER PIC X(3) VALUE "FMS".

05 FILLER PIC X(3) VALUE "INF".

05 FILLER PIC X(3) VALUE "PSL".

05 FILLER PIC X(3) VALUE "UND".

05 FILLER PIC X(3) VALUE SPACE.

01 FILLER REDEFINES CLASS-TABLE.

05 CLASS-NAME OCCURS 7 TIMES PIC X(3).

- Contains three character codes for message class.

c. Branching and Error Conditions

NOT APPLICABLE FOR PRINT MESSAGE ROUTINES.

2.2.1.2.6 Obtain Input Card

Support is provided by a three-entry point module whose program identification is OBIC (but whose module name is specified as OBKWE to facilitate program structure report generation). The three entry points correspond with the subordinate routines OBPN, OBKW, and OBSD described in the following paragraphs. These routines read and scan the input cards submitted by a PSL user to determine the next PSL function, keyword and source data, respectively.

2.2.1.2.6.1 OBFN - Obtain a Function

The module OBFN is responsible for obtaining the function or subfunction word from the INPUT-CARD file. The OBFN processing scans the users' control cards for the function or subfunction name. If the name is not found on the control card, OBFN processing will automatically read through the INPUT-CARD file scanning each control card until a function or subfunction is located. Once the function name has been obtained, OBFN processing initializes the INPUT-CARD-KEYWORD-VALUE area and scans the remainder of the control card for a non-blank character. The module OBFN only reads the INPUT-CARD file, it does not open or close the file.

a. Program Operations

HIPO diagram 1.2.6.1 describes the operation performed. In step 3, only cards containing the PSL control characters set of "***" are scanned for the function or subfunction word. Starting at card column 3 which contains a blank character, a scan for a non-blank character begins. In step 4 through step 6, each contiguous non-blank character is stored in a hold area. In step 7, if the word in the hold area has the proper length, then it is moved to the output area and the word found switch is turned on. In step 9, internal data areas are initialized for keyword processing (OBKW) is cleared. (see Section 2.2.1.2.6.2 for description).

b. Data File and Table Descriptions

INPUT-CARDS

This file contains the user's control cards and user data. The PSL run deck is identified as the INPUT-CARDS file.

• 01 INPUT-CARD

05 INPUT-CARD-COLUMN OCCURS 80 TIMES
PIC X.

- Card image of user control cards and user data control cards are distinguished from user data cards by the character set "***" in the first three position of the card.

1. OUTPUT ARGUMENT

INPUT-CARD-FUNCTION PIC X(12).

- The function word obtained from the PSL control card.

Diagram ID: 1.2.6.1

Name: OBPN - Obtain a Function

Description: Support Routine

INPUT

CONTROL-CARD-FILE
CONTROL-CARD

▲ SPECIAL-CHARACTER

PROCESS

```

1. IF end of control cards
   Set status to end cards
   ENDIF
   DO WHILE no function exist and not end
     of control cards

2. Read card file
   IF first three characters equal ".*"
     Starting at column three scan
     control card for non-blank
     characters
   IF non-blank characters found
     → DO WHILE non-blank character and
       length of word less than 13 and
       character is not delimiter (/,
       (,),",")
3. Move character to HOLD-AREA
   Add 1 to length of word
   Add 1 to card column
   ENDDO
   IF blank character and length of
   word is 1 thru 12
4. Move hold-area to input-card-
   function
5. Scan control cards for non-
   blank character
6. Initialize input-card-keyword-
   value
   ELSE
7. Print control card
8. Print message (PMS)
9. Read next control card
   ENDIF
   ELSE
10. Print control card (PMS)
11. Read next control card
   ENDIF
   ELSE
12. Read next control card
   ENDIF
   ENDDO

```

OUTPUT

[Processing Status]

MESSAGE

MESSAGE-FILE

[INPUT-CARD-FUNCTION]

▲ INPUT-CARD-KEYWORD-VALUE

PROCESSING-STATUS

PIC S9(9) COMP.

Return code - zero for normal status
- error code indicating cause of
failure of OBFN. See Section
2.2.1.2.6.1(c) for complete
list of codes.

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	14	INF	Subsequent processing bypassed	
3	13,71	ERR	Next input card is read, processing continues	
10	13	ERR	Next input card is read, processing continues	

2.2.1.2.6.2 OBKW - Obtain Keyword

The module OBKW is responsible for obtaining the keywords and associated values with PSL functional processing. A list of keywords is given in Figure 3-12 of the User's Manual. The calling routine, which is usually a functional processor, expects to receive the keyword and the associated value as input. The module OBKW reads through the INPUT-CARD file processing user's control cards. The module OBKW scans the user control cards storing and passing keywords and their associated values to the calling routines. Each call to the module OBKW updates pointers to the control card where to start the OBKW processing for the next call. The arrangement of keywords and values are dependent on the delimiter between words on the control card.

a. Program Operations

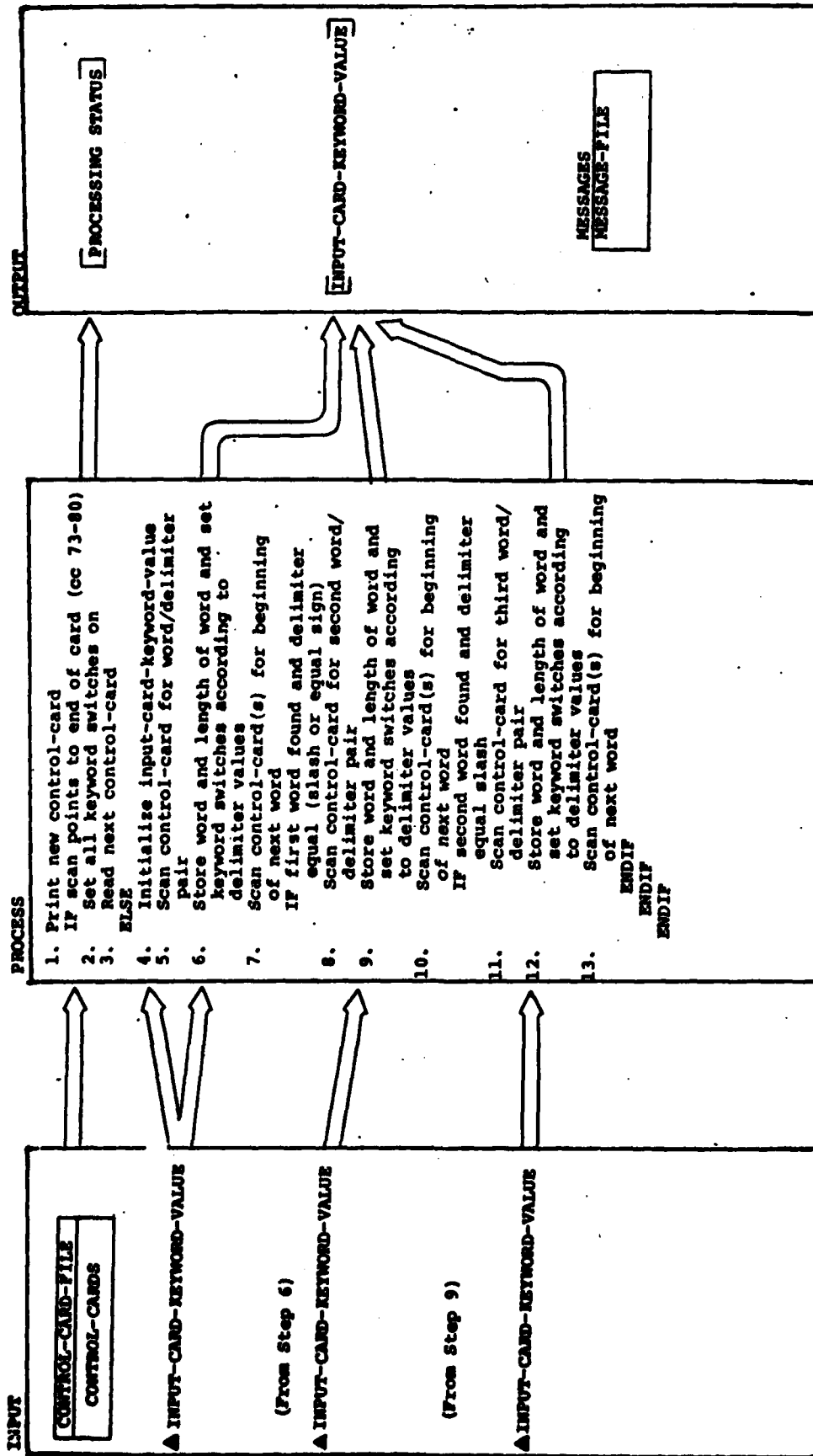
HIPC diagram 1.1.6.2 describes the operation performed. In step 1, if the control card has not been printed by previous processing, it is printed. In step 5, the OBKW processing scans across the card for the first word and delimiter that appears. In step 6, the first word and delimiter obtained is used to update INPUT-CARD-KEYWORD-VALUE. Storage of the data processed is dependent on the delimiter encountered. In step 8, the continued processing of OBKW is dependent on previous storage of data words. If the previous delimiter denotes that a keyword value is to follow, then processing continues to obtain second word delimiter pair. In step 11, if previous delimiter denotes that keyword value follows, then processing continues to obtain third word delimiter pair. In step 13, if the end of control card being scanned has been reached or the card is continued on the next card, the next card of the INPUT-CARD file is read, verified as a control card, and scanned for first character of first word.

b. Data File and Table Descriptions

INPUT-CARDS

- See section 2.2.1.2.6.1 for description

77	SPECIAL-CHARACTER	PIC X.
88	BLANK-CHARACTER	VALUE SPACE.
88	FUNCTION-DELIMITER	VALUE SPACE, QUOTE, "(", ")", "/", "=", ":", "
88	WORD-DELIMITER	VALUE "(", ")", "/", "=", ",", SPACE.
88	QUOTE-PRESENT	VALUE QUOTE.
88	END-CARD	VALUE "X".



88	EQUAL-SIGN	VALUE "=".
88	COMMA-DELIMITER	VALUE ",".
88	CPAREN-DELIMITER	VALUE ")".
88	SLASH-DELIMITER	VALUE "/".
88	OPAREN-DELIMITER	VALUE "(".
77	TRAILING-DELIMITER	PIC X.
88	TRAILING-SLASH	VALUE "/".
88	TRAILING-COMMA	VALUE ",".
88	TRAILING-EQUAL-SIGN	VALUE "=".
88	TRAILING-CPAREN	VALUE ")".
88	TRAILING-OPAREN	VALUE "(".
88	TRAILING-BLANK	VALUE SPACE.

- List of delimiters used in OBKW processing to support keywords and keyword values.

OUTPUT ARGUMENTS

• 01 INPUT-CARD-KEYWORD-VALUE

- This data group is used to store keywords, key-word values, length of words and set switches denoting the end of any particular data level stored.

05	PRIME-KEYWORD	
10	PRIME-KW-CHARACTER	OCCURS 12 TIMES PIC X(12).
05	FILLER	REDEFINES PRIME-KEYWORD
10	SHORT-PRIME-KEYWORD	PIC X(4).
10	FILLER	PIC X(8).

- Usually the first word following the function word on a control card. The delimiter which follow PRIME-KEYWORD is = or blank.

05	SUB-KEYWORD.	
10	SUB-KW-CHARACTER	OCCURS 12 TIMES PIC X(12).
05	FILLER	REDEFINES SUB-KEYWORD.
10	SHORT-SUB-KEYWORD	PIC X(9).
10	FILLER	PIC X(8).

- Usually presided by the delimiters ((OR, OR b) and followed by a slash delimiter.

05	LENGTH-OF-VALUE	PIC S9(9) COMP.

- Denotes the length of the stored keyword value

05	KEYWORD-VALUE.	
10	KW-VALUE-CHARACTER	OCCURS 48 TIMES PIC X.

05 FILLER REDEFINES KEYWORD-VALUE.
 10 SHORT-KEYWORD-VALUE PIC X(4).
 10 FILLER PIC X(44).

- Usually preceded by a delimiter (= OR, OR /)
 and followed by a delimiter (\$, comma, OR /,
 OR)).

05 LAST-IMRE-KW-SW PIC S9(9) COMP.

- Denotes when the PRIME-KEYWORD is obtained

05 LAST-SUB-KW-SW PIC S9(9) COMP.

- Denotes when the last SUB-KEYWORD is stored

05 LAST-KW-VALUE-SW PIC S9(9) COMP.

- Denotes the last keyword associated with a PRIME-
 KEYWORD or SUB-KEYWORD on the control card.

• 01 PROCESSING STATUS PIC S9(9) COMP.

Return code - zero for normal status

- error code indicating cause of failure of
 OBKW. See section 2.1.2.1.2.6.2(c) for
 complete list of codes.

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	20	N/A	Normal status Processing continues	
2	89	PSL	Subsequent processing bypassed	
5	19	ERR	Processing continues	
6	2	ERR	Processing continues	
7	13,71	ERR	Processing continues	

2.2.1.2.6.3 OBSD - Obtain Source Data

The module OBSD is responsible for obtaining user data from the INPUT-CARD file. The module OBSD is usually called by a PSL functional processor to obtain data line for a unit in a PSL library. User data which has the control characters "***" in the first columns must be preceded by ** DATA card and followed by an ** ENDATA card in order to be recognized as user data. Each call to the module OBSD returns a user data line.

a. Program Operations

HIPO diagram 1.2.6.3 describes the operation performed. In step 1, if an input card is read that has the control set (***) in the first three columns then the card has to be scanned for the word DATA. In step 3, the first non-blank character following the word DATA on the ** DATA card is stored in LIMIT-SPEC for later validation of ** ENDATA card. In step 1, if an input card is read that has the control set (***) in the first three columns, then the card has to be scanned for the word ENDATA. In step 12, the first non-blank character following the word ENDATA on the ** ENDATA card is compared to the LIMIT-SPEC. If the non-blank character matches LIMIT-SPEC then the end of user data has been reached, otherwise the card is considered user data.

b. Data File and Table Descriptions

INPUT-CARDS

- See section 3 for description.

• 01 LIMIT-SPEC

PIC X(3) VALUE HIGH VALUES

- Storage area for non-blank character followed word DATA on the ** DATA card

1. OUTPUT ARGUMENTS

SOURCE-DATA

PIC X(80).

- Card image of INPUT-CARD user data

PROCESSING STATUS

PIC S9(9) COMP.

Return code - zero normal status

- error code indicating cause of failure of OBSD. See section 2.2.1.2.6.3(c) for complete list of codes.

Diagram ID: 1.2.6.3

Name: OBSD - Obtain Source Data

Description: Support Routine

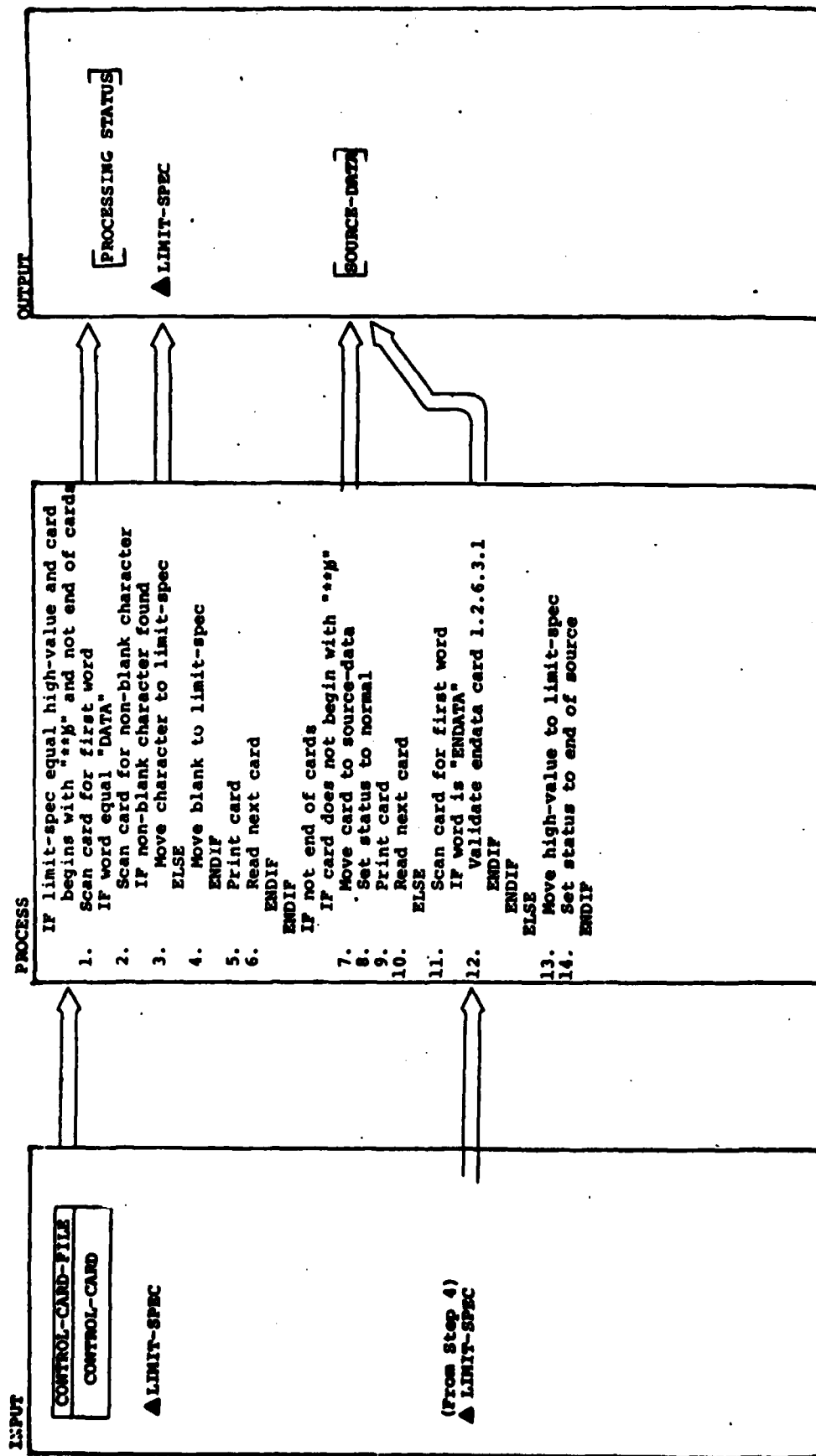


Diagram ID: 1.2.6.3.1

Name: OBSD - Validate ENDATA Card

Description: Support Routine

INPUT

CONTROL-CARD-FILE
CONTROL-CARD

▲ LIMIT-SPEC

PROCESS

1. Scan card for non-blank character
IF character equal limit-spec
or (limit-spec equal space and
no non-blank character)
2. Print cards
3. Read next card
4. Move high-values to limit-spec
5. Set status to end of source
ELSE
6. Move card to source-data
7. Set status to normal
8. Print card
9. Read next card
ENDIF

OUTPUT

▲ PROCESSING STATUS

▲ SOURCE-DATA

▲ LIMIT-SPEC

c. Branching and Error Conditions

Function Reference	Condition Code	Message Category	Program Action	Note
1	14	INF	Subsequent processing bypassed	
1,11	13,71	ERR	Next input card is read, processing continues	
5,9,12	20	N/A	Normal status Processing continues	
8	0	N/A	Normal status Processing continues	
12,24	28	N/A	End of source data Processing continues	