AD-A031 461 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
STRUCTURED PROGRAMMING TRANSLATORS. VOLUME V. STRUCTRAN-2 SYSTE--ETC(U)
AUG 76 C GANNON, R A MELTON F30602-75-C-0245
UNCLASSIFIED RADC-TR-76-253-VOL-5 NL

1 OF 1
AD A031461

END

DATE FILMED
12-76

RADC-TR-76-253, Vol V (of five)
Final Technical Report
August 1976

STRUCTURED PROGRAMMING TRANSLATORS
STRUCTRAN-2 System Design and Implementation Manual

General Research Corporation

Approved for public release;
distribution unlimited.

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

This report consists of the following volumes:

I - Final Report
II - STRUCTRAN-1 User's Manual
III - STRUCTRAN-1 System Design and Implementation Manual
IV - STRUCTRAN-2 User's Manual
V - STRUCTRAN-2 System Design and Implementation Manual

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and approved for publication.

APPROVED: Donald L. Mark

DONALD L. MARK
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-76-253-Vol [redacted] 5	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): STRUCTURED PROGRAMMING TRANSLATORS. Volume V. STRUCTRAN-2 System Design and Implementation Manual.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. May 75 — Jan 76
6. PERFORMING ORG. REPORT NUMBER: N/A	
7. AUTHOR(s): C. Gannon; R. A. Melton	8. CONTRACT OR GRANT NUMBER(s): F30602-75-C-0245
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Research Corporation, P. O. Box 3587, Santa Barbara CA 93105	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 32010314
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441	12. REPORT DATE: August 1976
	13. NUMBER OF PAGES: 56
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

16 AF-3201

17 320103

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

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Donald L. Mark (ISIS)

DMAAC Project Engineer: Ms. Opal Power

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

Structured Programming
Precompilers
Translators
Software Tools

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

The STRUCTRAN-2 System Design and Implementation Manual describes the overall design of the STRUCTRAN-2 translator, and the organization and contents of the STRUCTRAN-2 data base. This manual is extended primarily for use in STRUCTRAN-2 software maintenance.

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

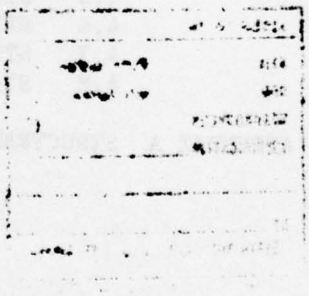
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

402754 LB

CONTENTS

ILLUSTRATIONS

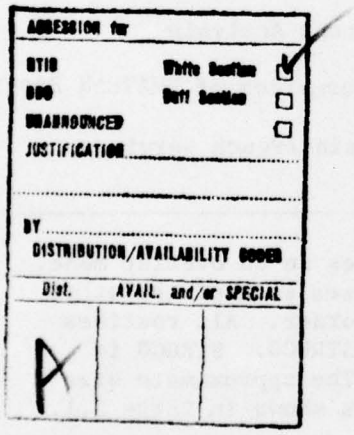


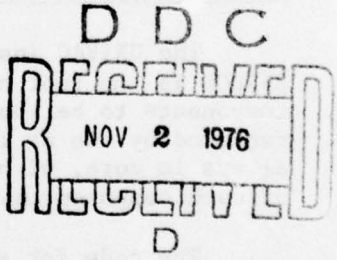
1 INTRODUCTION

The STRUCTRAN structured programming translators are tools for enabling structured programming in FORTRAN-based software developments. STRUCTRAN-1 translates programs written in DMATRAN into pure FORTRAN programs. STRUCTRAN-2 translates ("structurizes") programs written in FORTRAN into a structured form expressed in DMATRAN.

The STRUCTRAN-2 software has been developed on the Control Data Corporation (CDC) 6400 computer at the General Research Corporation (GRC) facility in Santa Barbara, California. Following this software development phase, STRUCTRAN-2 has been installed on the UNIVAC 1108 at the Defense Mapping Agency Aerospace Center (DMAAC), St. Louis, Missouri. The software development phase utilized the CDC FORTRAN run compiler, version 2.3 under the GOLETA operating system for the CDC 6400. The installation phase utilized the UNIVAC 1108 FORTRAN V compiler with the UNIVAC 1108 operating system.

This document describes the overall design of the STRUCTRAN-2 software (Sec. 2), and the organization and contents of the STRUCTRAN-2 data base (Sec. 3). The STRUCTRAN-2 software components are described in Sec. 4. A narrative description of each module contained in the components is included in Appendix A. The details of the STRUCTRAN-2 organization, intermodule dependencies, and intramodule control structure are contained in the STRUCTRAN-2 Software Analysis Collection documentation.





2 STRUCTRAN-2 PROGRAM OVERVIEW

2.1 STRUCTRAN-2 PROGRAM ORGANIZATION

STRUCTRAN-2 is organized into eight components, each of which consists of a hierarchy of interrelated modules with specific processing capabilities. These components are used in a definite order to translate FORTRAN to DMATRAN. Table 2.1 shows the STRUCTRAN-2 components; detailed descriptions of each component's function appear in Sec. 4.

TABLE 2.1

STRUCTRAN-2 COMPONENTS

Component	Memory (Octal)	Function
STRUC0	77K	Storage Manager, Data Base Services, Support Subroutines, and Control
STRUC1	3K	Initialization of Data Base
STRUC2	14K	Lexical Analysis
STRUC3	14K	FORTRAN Parsing
STRUC4	13K	Graph Identification
STRUC5	16K	Graph Analysis
STRUC6	20K	Formation of DMATRAN Program
STRUC7	5K	Print/Punch Services

The UNIVAC installation of STRUCTRAN-2 operates in an overlay mode. STRUC0 is the root of the overlay structure and causes each of the other components to be loaded and executed in the proper order. All routines required by two or more components are included in STRUC0. STRUC0 is always in core, with at most one other component. The approximate size of each component as overlayed on the UNIVAC 1108 is shown in Table 2.1.

The code for all STRUCTRAN-2 components was developed by GRC personnel utilizing structured programming techniques throughout.

2.2 STRUCTRAN-2 OPERATION

Each STRUCTRAN-2 component is devoted to a unique task; communication between components is through a common data base called a data base library. The data base library is initially created by STRUC1, modified by STRUC2, STRUC3, STRUC4, and STRUC5, and utilized by STRUC6 and STRUC7. The sequence in which the components are executed is shown in Fig. 2.1, which uses DMATRAN to express a high-level, structured representation of STRUCTRAN-2 operation.

```
 PROCEDURE STRUCTRAN2 CONTROL
 INVOKE ( STRUC1 - INITALIZE DATA BASE )
 DO WHILE ( END OF THE CURRENT MODULE HAS NOT BEEN REACHED )
  .  INVOKE ( STRUC2 - PERFORM LEXICAL SCAN OF MODULE TO IDENTIFY
1.           KEYWORDS AND VARIABLES )
  .  IF ( END OF THE CURRENT MODULE HAS NOT BEEN REACHED ) THEN
  .  .  INVOKE ( STRUC3 - PERFORM PARSE OF MODULE TO IDENTIFY STATEMENT
1.  .           TYPES )
  .  .  INVOKE ( STRUC7 - PRINT FORTRAN MODULE )
  .  .  INVOKE ( STRUC4 - IDENTIFY PROGRAM GRAPH )
  .  .  INVOKE ( STRUC5 - RECOGNIZE STRUCTURED FORMS IN PROGRAM GRAPH AND
1.  .           ABSTRACTLY REPRESENT AS A STRUCTURED PROGRAM )
  .  .  INVOKE ( STRUC6 - USE STRUCTURED FORM REPRESENTATION TO REWRITE
1.  .           MODULE IN DMATRAN )
  .  .  INVOKE ( STRUC7 - PUNCH DMATRAN MODULE )
  .  END IF
 END WHILE
 END
```

Figure 2.1. Description of STRUCTRAN-2 Control

STRUC0 consists of the main program for STRUCTRAN-2, all the routines it causes to be loaded, and all routines which are referred to directly or indirectly in two or more components. Since STRUC1 through STRUC7 communicate through the data base library, all data base service routines and storage manager routines are in STRUC0. Also all support routines performing commonly utilized functions are in STRUC0. The data base library is initialized by STRUC1. STRUC2 inputs the FORTRAN text to be structurized, adding it to the data base library after performing a lexical analysis to recognize FORTRAN key words and variables. STRUC3 reads the lexically analyzed text from the data base library and parses each statement, adding statement type and symbol usage information to the data base library. STRUC4 utilizes all of the information accumulated so far in the data base library to identify the program graph of the module. This is added to the data base library in tabular form. STRUC5 performs a reduction type analysis to recognize structured forms in the graph of the module. It updates the information on the data base library to indicate the structured form of the module. STRUC6 uses this representation to build the DMATRAN version of the original FORTRAN program. STRUC7 is used to print and punch statement text from the data base library. Since many routines may be contained in a single UNIVAC 1108 FORTRAN V program unit (due to the internal subroutine capability) the iteration indicated in Fig. 2.1 is necessary to process modules consisting of more than one routine.

2.3 STRUCTRAN-2 DATA STRUCTURES

In performing its structurization STRUCTRAN-2 utilizes the data structures shown in Table 2.2. The FORTRAN source text to be structurized is input on LIN. A random access data base library is created on LIBNEW. The FORTRAN source is listed on LOUT, translated DMATRAN source is written on LPUNCH, and a high level audit trail is output to LDEBUG. File activity performed by STRUCTRAN-2 is summarized in Fig. 2.2.

Examples of each type of output which is meaningful to the user appear in the STRUCTRAN-2 User's Manual.

TABLE 2.2

FILES USED IN STRUCTRAN-2 PROCESSING

Logical Unit Number	File Name	Data Structure	Mode	Storage Form	Record Format	Recommended Allocation
6LLIBNEW	LIBNEW	FORTRAN Program Description	Binary	Random	System Standard (installation-dependent)	Scratch file (mass storage)
6LOUTPUT	LOUT	Reports	BCD	Sequential	Maximum 132 characters per line	System printer
6LLDEBUG	LDEBUG	Reports	BCD	Sequential	Maximum 132 characters per line	Scratch file (mass storage)
6LLPUNCH	LPUNCH	DMATRAN Translated Text	BCD	Sequential	Card image	System punch or permanent file*
5LINPUT	LIN	FORTRAN Text	BCD	Sequential	Card image	Card reader or permanent file*

*Permanent files must have been previously created.

NOTE: The above files, along with other intermediate scratch files, are defined in subroutine FLINIT.

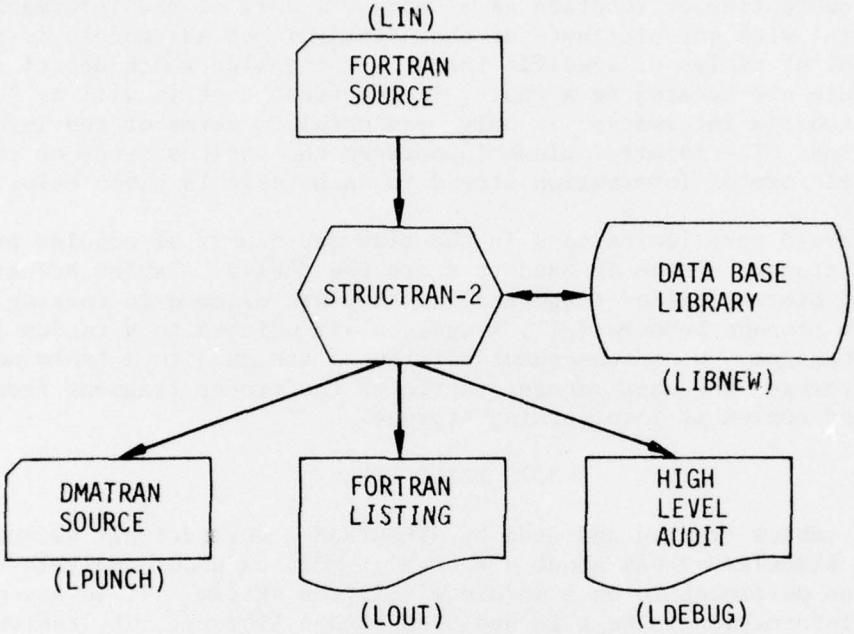


Figure 2.2. STRUCTRAN-2 Data Flow

3 STRUCTRAN-2 DATA BASE

3.1 INTRODUCTION

The STRUCTRAN-2 system is intended to provide the nucleus of a generalized system for translation of FORTRAN programs to DMATRAN. To provide the greatest possible flexibility in supporting STRUCTRAN-2 translation objectives, it is necessary to preserve detailed information about the internal structure and organization of individual elements of the program text being analyzed. For convenience, we refer to a UNIVAC 1108 FORTRAN V program, an internal or external subroutine or function as a "module". All of the information necessary to deal with any attribute or characteristic of any module is embodied in a series of tables of specific format. The tables which describe a particular module are treated as a whole, in the sense that it will be the case that one table's information is only meaningful in terms of the information in other tables. The interrelationship between the various types of tables and the general form of information stored in each table is given below.

To avoid core limitations in the size and number of modules processed, a virtual storage system is used to store the tables. Tables are assigned to predefined blocks (called fragments) as they are created in working storage. As working storage becomes full, fragments are written to a random library by the data manager. When subsequent references are made to a table no longer in working storage, the data manager retrieves the proper fragment from the library and copies it into working storage.

3.2 TABLE TYPES

The tables created and used by *STRUCTRAN-2 collectively* define all information STRUCTRAN-2 has about a system of modules under analysis and the information pertinent to each module within the system. It is assumed that all this information is kept in one STRUCTRAN-2 library. The tables used by STRUCTRAN-2 can then be categorized in the following way:

a. Those which describe the contents of the library (system tables)
b. Those which pertain to specific modules (module tables)
c. Those which support token storage (text tables)

Each table is in one of two formats: fixed length (meaning a fixed number of words per entry or block) or variable length (meaning a variable number of words per entry or block). Each table is divided into fixed-length segments called fragments; one fragment is the unit of transfer to the auxiliary storage device which holds the library. We will discuss each type's general nature in turn. Specific uses of each word of a table are found in the table format on the referenced page.

3.2.1 Permanent System Tables

The tables used to describe library contents are listed below.

Library Header Table (LHT): fixed table. The Library Header Table contains a fixed set of global library data that is the same for all libraries. The table is always located in the first twenty words of the random file that contains the library and provides pointers to the Entry Point Table, which is the directory table for the library.

Module Location Table (MLT): fixed table. The Module Location Table stores the library address and working storage address of each module descriptor block known to the system. The MLT is not located on the library, but is constructed in working storage as analysis proceeds. The MLT is in ascending order by module number assigned at run time.

Entry Point Table (EPT): fixed table. The Entry Point Table is used for storing the names and locations of modules within the library. It is searched for finding the run-time module number of a named module. The entries in the EPT are created in the order in which modules are assigned to the library.

Fragment Directory Table (FDT): fixed table. The Fragment Directory Table is used to link library fragments belonging to one table. It is also used to store the working storage location of a fragment if it is resident in core. The entries in the FDT are in fragment order.

3.2.2 Module Tables

The single-module tables and blocks describe, in aggregate, all of the fixed characteristics of each FORTRAN module. These tables are listed below:

Module Descriptor Block (MDB): fixed table. The Module Descriptor Block specifies certain parameters pertaining to a single module in the system. It also contains Library Manager pointers to other tables pertaining to this module. The MDB must be present in the working storage when using any of the other module tables. There is only one entry for each module in this table, hence there is no implied order.

Statement Descriptor Block (SDB): fixed table. The Statement Descriptor Block table describes the attributes of each statement within the pertinent module. There is an entry in the table for each statement, containing type codes, length information, block structure pointers and other descriptive information. The SDB table is in ascending order by statement number.

Statement Block (SB): variable table. The Statement Block table contains the parsed source text in token pointer form for the pertinent module. There is one Statement Block for each statement in the source program text; each block is a variable length word-pair list. In addition to the source text which is stored as a string of token pointers, the Statement Block contains token keyword descriptors and pointers to the Symbol Table. The SB table is in ascending order by statement number and by word-pair within a statement.

Symbol Table (STB): fixed table. The Symbol Table contains an entry for each symbol defined within the pertinent module. Type, mode, pointers and other descriptive information for each symbol are recorded.

DD-Path Table (DDP): fixed table. The DD-Path (Decision-to-Decision Path) Table contains the module's program graph. There is an entry for each decision-to-decision path in the program, with the beginning statement number, the ending statement number and other useful information. The DDP table is in ascending order by DD-path number.

DD-Path/Statement Table (DS): variable table. The DD-Path/Statement Table has a variable length entry for every DD-path in the pertinent module. Each entry is the list of statements comprising the corresponding DD-path. The DS Table is in ascending order by DD-path number, and in statement execution order within a DD-path.

3.2.3 Temporary Module Tables

Various tables are used internally in given components and do not contain information passed to other components. Such temporary tables provide efficiency in operation and flexibility in size constraints. These tables are listed below:

Label/Statement Table (LBTL): fixed table. The Label/Statement Table contains an entry for each labeled executable statement in the module. Given a label in some form of GOTO statement, it allows the corresponding statement number to be efficiently found.

Current Program Graph Tables (ITEMP and JTEMP): fixed tables. The Current Program Graph Tables are used in the graphical reduction process.

Program Graph Connection Table (CN): variable table. The Program Graph Connection Table is used to efficiently store and retrieve information describing the current program graph.

Program Graph Connection Directory Table (CNAX): fixed table. This table serves as the directory for the variable length CN table.

3.2.4 Text Tables

The system breaks down the module source text into its individual tokens (keyword, variable name, delimiter, constant, operator) and stores the text by saving these tokens. The character strings that make up the tokens are stored individually in a token list for an entire library, and the table entries that refer to tokens contain pointers to the particular character strings in the token list. The following permanent tables are used for storing text in this way. Figure 3.1 shows pointer relationships between tables.

Token String Table (TKL): variable table. The full text of each token is stored in a table entry of a variable number of words. On the UNIVAC 1108 with 6 characters per word all strings of 1 to 6 characters are stored in one word (with blank fill), 7 to 12 characters in two words, and so on. The string address is the word number (starting from the beginning of the list) of the first word of the table entry. No other information but character strings is stored in the token string table.

Token Directory Table (TOK): fixed table. To simplify and speed up Token String Table searches, a Token Directory Table is created. This fixed table contains four words per entry describing a particular Token String Table entry:

Word 1 contains one word of token text (UNIVAC 1108 implementation has the first six characters of the string)

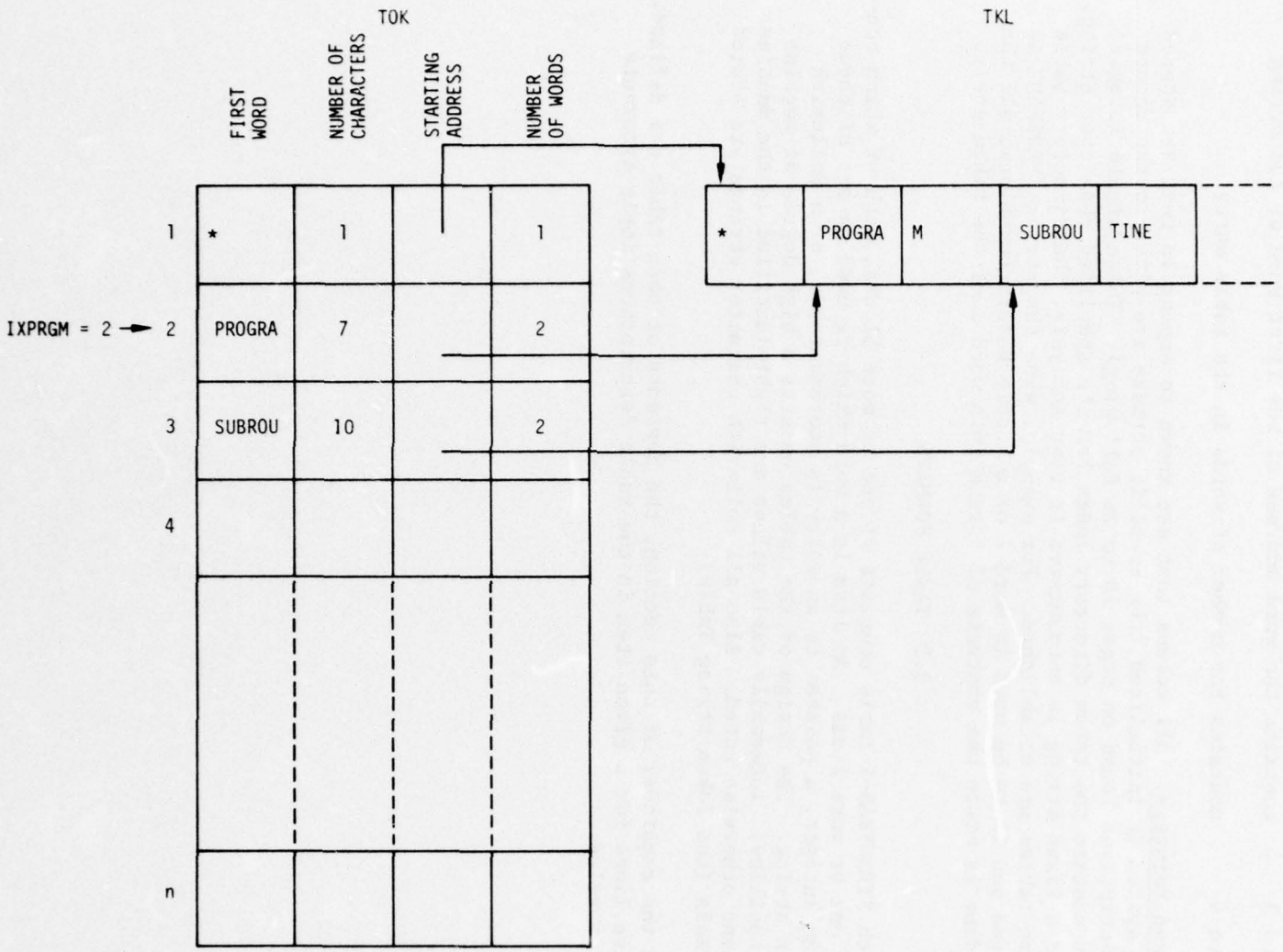


Figure 3.1. Pointer Relationships for Character String Storage and Retrieval

Word 2 contains the number of characters in the complete string in the Token String Table

Word 3 contains the word address of the first word of the string

Word 4 contains the number of words in the table entry

Fixed Pointers. All tokens that are known to appear in text are stored when the system is initialized (the specific strings are given in the common block descriptions found on pages 20 to 26 following). These simple integer variables contain the token directory index for the specified character string. Thus when a fixed string is encountered in text analysis, the directory table and string tables are established. For example, when the string "PROGRAM" is encountered and is to be used in word 2 of a Module Descriptor Block, all that need be done is store the contents of IXPRGM into word 2 of the table entry.

3.3 TABLE FORMATS

Each STRUCTRAN-2 table consists of one or more blocks, each of which consists of one or more items. An item is a word which is used as one of three types: an integer, a pointer to an entry in another table, or a Hollerith character string. The design of the tables ensures a high degree of machine transportability: internally table values are right-justified in the machine word unless otherwise noted. Also all Hollerith character strings are stored in one table (the Token String Table).

In the remainder of this section, the contents of each table are defined. Successive lines for a given item in the table descriptions imply alternate meaning or value.

```
*OK START
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C                 PERMANENT SYSTEM TABLES

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     L I B R A R Y   H E A D E R   T A B L E

C     WORD      DEFINITION
C     ----      ----------

C       1       PASSWORD NAME OF LIBRARY
C       2       ACCESS DIRECTIVE FOR A RUN - HOLLERITH(H) FORMAT
C                ... 3HNEW        -NOT WRITTEN BEFORE
C                ...4HREAD    -CANNOT BE CHANGED DURING RUN
C                ... 3HR/W        -CAN BE CHANGED DURING RUN
C       3
C       4       WORDS USED ON LIBRARY
C       5       SIZE OF LIBRARY FRAGMENT
C       6       NUMBER OF MODULES ON LIBRARY
C       7
C       8       FIRST MDB FRAGMENT ON LIBRARY
C       9       NUMBER OF FRAGMENTS USED ON LIBRARY
C      10       NUMBER OF ENTRIES IN FRAGMENT OF EPT
C      11       FIRST FDT FRAGMENT ON LIBRARY
C      12       NUMBER OF ENTRY POINTS IN MODULES ON LIBRARY
C      13       NUMBER OF ENTRIES IN FRAGMENT OF FDT
C      14       FIRST EPT FRAGMENT ON LIBRARY
C      15
C      16
C      17
C      18       DATE LIBRARY OPENED NEW
C      19       TOTAL NUMBER OF TIMES LIBRARY WRITTEN
C      20       DATE LAST CLOSED AFTER BEING CHANGED

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     M O D U L E   L O C A T I O N   T A B L E

C     WORD      DEFINITION
C     ----      ----------

C       1       LIBRARY ADDRESS OF MODULE MDB                         INTEGER
C       2       CORE ADDRESS OF MODULE MDB ( 0 IF NOT RESIDENT )      INTEGER
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     E N T R Y   P O I N T   T A B L E

C     WORD      DEFINITION
C     ----      ----------

C       1       NAME OF ENTRY POINT                                   TOK POINTER
C       2       RUN-TIME NUMBER OF MODULE                             INTEGER
C       3       NAME OF MODULE OWNING ENTRY POINT                     TOK POINTER
C       4       STATEMENT NUMBER ON WHICH ENTRY OCCURS                SDB POINTER
C       5       NUMBER OF ENTRY WITHIN MODULE                         INTEGER
C       6       INDEX OF MODULE MDB ON LIBRARY                        INTEGER
C       7       LIBRARY ADDRESS FOR MDB OF MODULE.                    INTEGER

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     F R A G M E N T   D I R E C T O R Y   T A B L E

C     WORD      DEFINITION
C     ----      ----------

C       1       POINTER TO NEXT LIBRARY FRAGMENT OF TABLE ( 0 IF NONE )
C                                                                     INTEGER
C       2       CORE FRAGMENT NUMBER IF LIBRARY FRAGMENT RESIDENT     INTEGER
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C                    PERMANENT MODULE TABLES

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C      M O D U L E   D E S C R I P T O R    T A B L E

C      MODULE DESCRIPTOR BLOCK (MDB).  THERE IS ONE MDB FOR EACH MODULE
C      ON A LIBRARY.  EACH MDB CONTAINS 100 WORDS OF INFORMATION DESCRIBING
C      NUMERIC PROPERTIRES OF THE MODULE OR SERVING AS POINTERS TO OTHER
C      TABLES DESCRIBING THE MODULE IN GREATER DETAIL.  THEY HAVE THE
C      FOLLOWING MEANING. . .

C      WORD      DEFINITION
C      ----      ----------

C        1       NAME OF MODULE                                     TOK POINTER
C        2       TYPE OF MODULE( IXPRGM , IXSUB , IXFNCT , IXBLCK )  TOK POINTER
C        3       MODE OF FUNCTION ( IXREAL , IXINTG , IXLGCL , IXCMPX ,
C                                   IXDBL , IXTPLS , IXBL IF NOT A FUNCTION).
C                                                                   TOK POINTER
C        4
C        5       TOTAL NUMBER OF STATEMENT WORD PAIRS               INTEGER
C        6
C        7       NUMBER OF STATEMENTS (NUMBER OF SDB ENTRIES)       INTEGER
C        8       NUMBER OF SYMBOLS (NUMBER OF STB ENTRIES)          INTEGER
C        9       NUMBER OF ARGUMENTS IF SUBROUTINE OR FUNCTION      INTEGER
C       10       NUMBER OF FIRST EXECUTABLE STATEMENT               INTEGER
C       11       NUMBER OF ENTRIES TO MODULE (IF SUBROUTINE OR FUNCTION)
C                                                                   INTEGER
C       12
C       13
C       14
C       15       NUMBER OF DEPENDENT MODULE TABLE BLOCKS            INTEGER
C       16
C       17       NUMBER OF DD-PATH BLOCKS                           INTEGER
C       18       LANGUAGE DIALECT( IXUNV )                          TOK POINTER
C       19
C       20       LANGUAGE ( IXFTRN )                                TOK POINTER
C       21
C       22
C       23
C       24       TOTAL NUMBER OF WORDS IN DS TABLE                  INTEGER
C       25       NUMBER OF EXTERNALS                                INTEGER
C       26
C       27       NUMBER OF EXECUTABLE STATEMENTS                    INTEGER
C       28       NUMBER OF COMMON STATEMENTS                        INTEGER
C       29       NUMBER OF EQUIVALENCE STATEMENTS                   INTEGER
C       30       NUMBER OF READ STATEMENTS                          INTEGER
C       31       NUMBER OF WRITE STATEMENTS                         INTEGER
C       32
C       33
C      34-50     SPARE
C      51-100    RESERVED FOR STORAGE MANAGER USE
C                                                                   TB ORIGIN
C                                                                   SB ORIGIN
C                                                                   SDB ORIGIN
C                                                                   STB ORIGIN
C                                                                   DDP ORIGIN
C                                                                   DS ORIGIN
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     S T A T E M E N T   D E S C R I P T O R   T A B L E

C     STATEMENT DESCRIPTOR BLOCK (SDB).  THERE IS ONE SDB FOR EACH STATEMENT
C     IN A MODULE.  EACH SDB CONTAINS 10 WORDS OF INFORMATION DESCRIBING
C     THE STATEMENT. THE STATEMENT SEQUENCE NUMBER SERVES AS THE INDEX
C     TO THIS TABLE.

C     WORD      DEFINITION
C     ----      ----------
C       1       STATEMENT LABEL                                      TOK POINTER

C       2       TYPE OF STATEMENT  - COMPARE WITH THE TOKEN POINTER VARIABLES
C               INDICATED.  HOLLERITH VALUE ASSOCIATED WITH EACH TOKEN POINTER
C               VARIABLE IS SHOWN.

C               TYPE                                TKL       HOLLERITH
C                                                   POINTER   VALUE
C                                                   VARIABLE

C               ---------------------------------------------------------------

C               ARITHMETIC IF                       IF3        8HIF-THREE
C               ASSIGN                              IXASS      6HASSIGN
C               ASSIGNMENT                          IXASGM    10HASSIGNMENT
C               ASSIGNED GOTO                       IAGT      10HGOTO-ASSGN
C               ASSIGNED GOTO                       IXASGT     8HASS-GOTO
C               BACKSPACE                           IXBKSP     9HBACKSPACE
C               BLOCK                               IXBLK      5HBLOCK
C       A       BLOCKDATA                           IXBLCK     9HBLOCKDATA
C       N       BUFFERIN                            IXBUFI     8HBUFFERIN
C       S       BUFFEROUT                           IXBUFO     9HBUFFEROUT
C       I       CALL                                IXCALL     4HCALL
C               CALL-EXIT                           IXCALE     9HCALL-EXIT
C               COMMON                              IXCMMN     6HCOMMON
C       X       COMPLEX                             IXCMPX     7HCOMPLEX
C       3       COMMENT                             IXCMMT     7HCOMMENT
C       .       COMPUTED GOTO                       ICGT      10HCOMP. GOTO
C       9       CONTINUE                            ICONT      8HCONTINUE
C               DATA                                IXDATA     4HDATA
C               DIMENSION                           IXDMSN     9HDIMENSION
C       F       DOUBLE                              IXDBL      6HDOUBLE
C       O       DO                                  IXDO       2HDO
C       R       END                                 IEND       3HEND
C       T       END-TOKEN                           IXENDT     9HEND-TOKEN
C       R       EQUIVALENC                          IXEQVL    10HEQUIVALENC
C       A       EXIT                                IXEXIT     4HEXIT
C       N       EXTERNAL                            IXEXTN     8HEXTERNAL
C               FORMAT                              IXFRMT     6HFORMAT
C               FUNCTION                            IXFNCT     8HFUNCTION
C       S       GOTO                                IGT        4HGOTO
C       T       LOGICAL IF                          IFT        2HIF
C       A       IF-ASSIGN                           IFASS      9HIF-ASSIGN
C       T       IF-ASSIGNMENT                       IFASGM    10HIF-ASSGMNT
C       E       IF-ASSIGNED GOTO                    IFAGT     10HIF-ASSGN-G
C       M       IF-BACKSPACE                        IFBACK    10HIF-BACKSPA
C       E       IF-CALL                             IFCALL     7HIF-CALL
C       N       IF CALL EXIT                        IFCLXT    10HIFCALLEXIT
C       T       IF-ENDFILE                          IFENDF    10HIFENDFILE
```

```
C               IF-GOTO                                 IFGT        7HIF-GOTO
C               IF COMPUTED GOTO                        IFGTC       9HIF-GOTO-C
C     T         IF-IF                                   IXIF        5HIF-IF
C     Y         IF-PAUSE                                IFP         8HIF-PAUSE
C     P         IF-PRINT                                IFPRNT      8HIF-PRINT
C     E         IF-PUNCH                                IFPNCH      8HIF-PUNCH
C     S         IF-READ                                 IFREAD      7HIF-READ
C               IF-REWIND                               IFRWND      9HIF-REWIND
C               IF-RETURN                               IFR         9HIF-RETURN
C               IF-STOP                                 IFS         7HIF-STOP
C               IF-WRITE                                IFWRTE      8HIF-WRITE
C               INTEGER                                 IXINTG      7HINTEGER
C               I/O-WRITE                               IXIOWR      9HI/O-WRITE
C               I/O-ACTION                              IXIOAC     10HI/O-ACTION
C               I/O-READ                                IXIORD      8HI/O-READ
C               LOGICAL                                 IXLGCL      7HLOGICAL
C               PAUSE                                   IXPAUS      8HPAUSE
C               PRINT                                   IXPRNT      5HPRINT
C               PUNCH                                   IXPNCH      5HPUNCH
C               READ                                    IXREAD      4HREAD
C               REAL                                    IXREAL      4HREAL
C               RETURN                                  IRET        6HRETURN
C               REWIND                                  IXRWND      6HREWIND
C               TWO BRANCH LOGICAL IF                   IF2         6HIF-TWO
C               STOP                                    ISTOP       4HSTOP
C               STATEMENT FUNCTION                      IXSTFX     10HSTMT. FUNC
C               SUBROUTINE                              IXSUB      10HSUBROUTINE
C               TO                                      IXTO        2HTO
C               WRITE                                   IXWRIT      5HWRITE

C               ------------------------------------------------------------

C     E         MULTIPLE ASSIGNMENT                     IXASNM     10HASSGN-MULT
C     X         BUFFER                                  IXBFFR      6HBUFFER
C     T         BUFFERI                                 IFBUFI     10HIF-BUFFERI
C     E         BUFFERO                                 IFBUFO     10HIF-BUFFERO
C     N         DECODE                                  IXDCD       6HDECODE
C     S         ENCODE                                  IXENCD      6HENCODE
C     I         ENTRY                                   IENT        5HENTRY
C     O         IF-DECODE                               IFDCDE      9HIF-DECODE
C     N         IF-ENCODE                               IFENCD      9HIF-ENCODE
C     S         NAMELIST                                IXNLST      8HNAMELIST
C               PROGRAM                                 IXPRGM      7HPROGRAM

C               ------------------------------------------------------------

C               ACCUMULATOR OVERFLOW                    IXACUM     11HACCUMULATOR
C               DIVIDE CHECK                            IXDVDE      6HDIVIDE
C     C         END-OF-FILE                             IXEOF       3HEOF
C     D         IF-END-OF-FILE                          IFEOF       6HIF-EOF
C     C         IF-UNIT                                 IFUNIT      7HIF-UNIT
C     -         IF-ACCUMULTOR OVERFLOW                  IFA        10HIF-ACCUMUL
C     R         IF-DIVIDE CHECK                         IFD         9HIF-DIVIDE
C     U         IF-QUOTIENT CHECK                       IFQ        10HIF-QUOTIEN
C     N         IF-SENSE-LIGHT                          IFSL       10HIF-SENSE-L
C               IF-SENSE-SWITCH                         IFSS       10HIF-SENSE-S
C               INPUT/OUTPUT CHECK                      IXIOCK      7HIOCHECK
C               LIGHT                                   IXLIGT      5HLIGHT
C               SENSE                                   IXSENS      5HSENSE
C               SENSELIGHT                              IXSNSL     10HSENSELIGHT
```

```
C               SWITCH                                  IXSWCH      6HSWITCH
C               UNIT                                    IXUNIT      4HUNIT
C
C               ------------------------------------------------------------

C       C       CALL WITH MULTIPLE RETURNS              ICALLR     10HCALL-RETRN
C       D       IF-CALL WITH MULTIPLE RETURNS           IFCLLR      9HIF-CALL-R
C       I       IMPLICIT                                IXIMPL      8HIMPLICIT
C       -       LEVEL                                   IXLEVL      5HLEVEL
C       F       OVERLAY                                 IXOVLY      7HOVERLAY
C       X       SUBROUTINE WITH MULTIPLE RETURNS        IXSUBR     10HSUBR-RETRN
C       T
C
C               ------------------------------------------------------------

C       B       BYTE                                    IXBYTE      4HBYTE
C       D       DEFINE                                  IXDEFI      8HDEFINE
C       D       DEFINE-RANDOM FILE                      IXDFRN     10HDEFINE-RAN
C       -       FILE                                    IXFILE      4HFILE
C       1       FIND                                    IXFIND      4HFIND
C       1       FIND-RANDOM RECORD                      IXFDRN      8HFIND-RAN
C               IF-READ WITH CONDITIONAL BRANCHES       IFRDC       9HIF-READ-C
C               IF-WRITE WITH CONDITIONAL BRANCHES      IFWTC      10HIF-WRITE-C
C               INTEGER (2 BYTES)                       IXINT2      9HINTEGER2
C               LOGICAL (2BYTES)                        IXLGC1      9HLOGICAL1
C               READ WITH CONDITIONAL BRANCHES          IREADC      9HREAD-COND
C               READ-RANDOM RECORD                      IXRDRN      8HREAD-RAN
C               REAL (4 BYTES)                          IXRL4       6HREAL4
C               REAL (8 BYTES)                          IXRL8       6HREAL8
C               WRITE WITH CONDITIONAL BRANCHES         IWRTEC     10HWRITE-COND
C               ------------------------------------------------------------


C       3       STATEMENT TYPE CODE -  TYPE OF STATEMENT FOR RXVP PROCESSING
C               FUNCTIONS.
C
C                    CODE                               INTEGER

C                      0 = NOT PARSED OR COMMENT
C                      1 = NON-EXECUTABLE
C                    100 = EXECUTABLE ( NON-DECISION )
C                    200 = EXECUTABLE ( DECISION )
C       4       LENGTH OF STATEMENT BLOCK (IN TOKENS)      INTEGER

C       5       SB ORIGIN OF STATEMENT TOKEN LIST          SB POINTER

C       6       INTRA-STATEMENT STRUCTURE POINTER          SDB POINTER
C               ...FOR FORTRAN...
C                    FOR IF STATEMENTS, THE WORDPAIR INDEX WHERE
C                         CONSEQUENCE OF IF BEGINS, OR FIRST LABEL
C                    FOR DO STATEMENT, THE STATEMENT NUMBER OF TARGET
C                         CONTINUE
C                    FOR DO-TARGET CONTINUES, THE STATEMENT NUMBER OF THE
C                         CORRESPONDING DO STATEMENT.

C       7       POINTER TO SYMBOL TABLE LOCATION FOR LABEL  STB POINTER

C       8       SPARE
C       9       SPARE
```

```
      10     SPARE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     S T A T E M E N T     T A B L E

C     STATEMENT BLOCK (SB).  THERE IS ONE SB ENTRY FOR EACH STATEMENT IN THE
C     MODULE.  THE LENGTH OF THE SB ENTRY DEPENDS OF THE NUMBER OF TOKENS
C     IN THE STATEMENT.  FOR EXAMPLE . . .
C
C              IF ( IX .EQ. JX ) GOTO 100
C
C     CONSISTS OF 8 TOKENS AND IS STORED IN SB FORMAT AS . . .
C

      ...........................................................
      .        .          .        .        .        .          .        .        .
      .IF      .(         .IX      ..EQ.    .JX      .)         .GOTO    .100     .
      .        .          .        .        .        .          .        .        .
      ...........................................................
      .        .          .        .        .        .          .        .        .
      .KEYWORD.DELIMITER.STB PT.OPERATOR.STB PT.DELIMITER.KEYWORD.STB PT.
      .        .          .        .        .        .          .        .        .
      ...........................................................
C     EACH TOKEN IS REPRESENTED AS A WORD PAIR.  THE TOP ELEMENT OF EACH
C     WORD PAIR IS A TOK POINTER TO THE ORIGINAL SOURCE TEXT FOR THE TOKEN
C     (STORED IN THE TKL TABLE).    THE BOTTOM WORD PAIR IS AN STB POINTER
C     IF THE TOKEN REPRESENTS A SYMBOL WHICH WAS ENTERED IN THE SYMBOL
C     TABLE, OR A TOK POINTER. THE STATEMENT SEQUENCE NUMBER SERVES AS THE
C     INDEX TO THIS TABLE.

C     WORD     DEFINITION
C     ----     ----------

C     THERE IS A WORDPAIR FOR EACH TOKEN IN THE STATEMENT TEXT
C        OF THE FOLLOWING FORMAT

C       1      TOKEN TABLE POINTER FOR SYMBOL TEXT          TOK POINTER

C       2      SYMBOL CATEGORY...
C                       TOKEN TABLE POINTER FOR 9HDELIMITER/8HOPERATOR/
C                         7HKEYWORD, IF APPROPRIATE         TOK POINTER
C                       OTHERWISE, SYMBOL TABLE POINTER     STB POINTER
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     S Y M B O L   T A B L E

C     SYMBOL TABLE BLOCK (STB).   THERE IS ONE STB ENTRY FOR EACH SYMBOL
C     (VARIABLE,ARRAY,CONSTANT,LABEL,COMMON BLOCK,FILENAME,EXTERNAL,OR
C     FUNCTION) IN THE MODULE.  THE ENTRY CONTAINS INFORMATION ABOUT THE
C     SYMBOL WHICH SUMMARIZES ITS USE THROUGHOUT THE ENTIRE MODULE
C     (NOT JUST ONE STATEMENT).

C     WORD     DEFINITION
C     ----     ----------

C       1       SYMBOL TOKEN TABLE POINTER                           TOK POINTER

C       2      CLASS OF SYMBOL (IXLOCL,IXPRMT,IXCMMN,IXEXTN,IENT)  TOK POINTER

C       3      TYPE OF SYMBOL (IXCONS,IXVRBL,IXARAY,IXLABL,IXEXTN,
C                  IENT,IXFLNM,IXFNCT)                               TOK POINTER
```

```
C       4       ARITHMETIC MODE (IXREAL,IXHOLL,IXINTG,IXLABL,IXLGCL,
C                   IXCMPX,IXDDBL,IXTPLS,IXFLNM)                    TOK POINTER

C       5       BLOCK NAME FOR SYMBOL (FOR BLOCK DATA OR
C                   COMMON BLOCK). (<NAME> OR BLANK IF CLASS IS 6HCOMMON
C                   BUT OTHERWISE BLANK)                            TOK POINTER

C       6       FOR ARRAY, INTEGER DIMENSIONALITY (0,1,2,3).        INTEGER
C               FOR EXTERNAL, THE NUMBER OF ACTUAL PARAMETERS IN FIRST
C                   OCCURANCE IN MODULE.                            INTEGER
C               FOR LABEL, STATEMENT NUMBER OF DEFINITION.          SDB POINTER

C     7,8,9     1ST, 2ND AND 3RD INTEGER DIMENSIONAL EXTENT OR ZERO
C               NEGATIVE IMPLIES VARIABLE DIMENSION                 INTEGER

C      10       SPARE
C      11       SPARE

C      12       STATEMENT NUMBER OF FIRST USE OF SYMBOL.            SDB POINTER
C      13       NUMBER OF OCCURANCES OF SYMBOL (TOTAL)              SDB POINTER
C      14       STATEMENT NUMBER OF LAST USE OF SYMBOL.             SDB POINTER

C     15-19     SPARE

C     CLASS == STORAGE ORIENTED
C     TYPE  == INTERNAL SYNTACTIC TREATMENT ORIENTED
C     MODE  == COMPUTATIONAL TREATMENT

C     NOTES...

C     (1) BY CONVENTION, STB(1) IS THE MODULE NAME.

C     (2) STB(2) IS THE FIRST FORMAL PARAMETER, IF ANY.

C     (3) THE OTHER STB(.) ARE ASSIGNED IN ORDER OF THEIR OCCURANCE WITHIN
C         THE MODULE.

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     D - D     P A T H     T A B L E

C     D-D PATH BLOCK (DDP).    THERE IS ONE DDP ENTRY FOR EACH DECISION OUTWAY
C     IN THE MODULE, AND ONE DDP ENTRY FOR EACH REDUCTION PERFORMED
C     DURING THE SE/SE SUBGRAPH ANALYSIS.  THE DDPATH NUMBER IS THE
C     INDEX TO THIS TABLE.

C     WORD      DEFINITION
C     ----      ----------
C       1       INITIAL STMT NUMBER                         INTEGER
C       2       FINAL STMT NUMBER                           INTEGER
C       3       NUMBER OF DS ENTRIES                        INTEGER
C       4       TYPE OF SE/SE
C                   0 == DDPATH
C                   1 == LINEAR COMPOSITION
C                   2 == PARALLEL COMPOSITION
C                   3 == SELF-LOOP COMPOSITION
C                   4 == REPEAT AT LEAST ONCE ITERATION COMPOSITION
C                   5 == SPARE
C                   6 == RESTRUCTURED LOOP EXIT (1ST HALF)
C                   7 == RESTRUCTURED LOOP EXIT (2ND HALF)
C                   8 == SINGLE ENTRY/MULTIPLE EXIT (SE/ME) COMPOSITION
```

```
C        5      PREDICATE INDEX                              INTEGER
C        6      SPARE
C        7      SPARE
C        8      SPARE
C        9      NUMBER OF STATEMENTS ON THE DD-PATH.         INTEGER
C       10      DS TABLE ORIGIN OF DDPATH/STATEMENT LIST
                OR SE/SE COMPOSITION LIST                    INTEGER

C     NOTES...

C     (1)  DD PATH BLOCKS SORTED IN ORDER OF (1) INITIAL STATEMENT NUMBER,
C          AND (2) PREDICATE INDEX.

C     (2)  ENTRY DDP IS ALWAYS DDP NO. 1.

C     (3)  EXIT DDP(S) ALWAYS END(S) ON HIGHEST ASSIGNED NODE NUMBER.

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     D S   T A B L E.  THERE IS ONE DS ENTRY FOR EACH DDP ENTRY.  THE
C     DDPATH NUMBER IS THE INDEX TO THIS TABLE.

C     D

C          VARIABLE LENGTH TABLE STORING EXECUTION ORDER STATEMENT SEQUENCES
C          FOR EACH DD-PATH.                                   SDB POINTER

C          SE/SE-TOSE/SE DECOMPOSITION TABLE                   DDP POINTER
C              VARIABLE LENGTH TABLE STORING THE COMPONENTS OF AN SE/SE.
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C                    TEMPORARY MODULE TABLES

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     L A B E L / S T A T E M E N T   T A B L E

C     LABEL/STATEMENT BLOCK (LBTL).  THERE IS ONE ENTRY IN THE LBTL TABLE
C     FOR EACH LABEL IN THE MODULE.  THE LABEL NAME IS THE INDEX TO THIS
C     TABLE.

C     WORD      DEFINITION
C     ----      ----------
C       1       LABEL NAME                                          TOK POINTER
C
C       2       STATEMENT THE LABEL IS DEFINE AT                    SDB POINTER
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     C U R R E N T   G R A P H   T A B L E
C
C     CURRENT GRAPH BLOCK (ITEMP/JTEMP).  THERE IS ONE ENTRY IN ITEMP OR
C     JTEMP FOR EACH ARC IN THE CURRENT PROGRAM GRAPH.
C     WORD      DEFINITION
C     ----      ----------
C       1       INITIAL STATEMENT NUMBER                            SDB POINTER
C       2       FINAL STATEMENT NUMBER                              SDB POINTER
C       3       DDP ENTRY DESCRIBING THIS SE/SE                     DDP POINTER

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     C O N N E C T I O N   T A B L E
C
C     CONNECTION BLOCK (JCN).  THERE IS ONE JCN ENTRY FOR EACH ITEMP/JTEMP
C     ENTRY.  THE ITH JCN ENTRY LISTS THOSE ARCS WHICH ARE CONNECTED TO
C     THE ITH ENTRY IN THE ITEMP/JTEMP TABLE.
C
C               VARIABLE LENGTH ENTRY STORING GRAPH CONNECTIONS
C                                                     ITEMP/JTEMP POINTER
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     C O N N E C T I O N   D I R E C T O R Y   T A B L E
C
C     CONNECTION AUXILIARY BLOCK (JCNAX).
C     THERE IS ONE JCNAX ENTRY FOR EACH JCN ENTRY.
C
C     WORD      DEFINITION
C     ----      ----------
C       1       SPARE
C       2       NUMBER OF CONNECTIONS                       INTEGER
C       3       JCN TABLE ORIGIN OF CONNECTION LIST         JCN POINTER
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     C O M M O N   B L O C K   D E S C R I P T I O N S
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     SINGLE ALPHABETIC CHARACTERS
      COMMON/ALPHA/ IXA,IXB,IXC,IXD,IXE,IXF,IXH,IXL,IXN,IXO,IXR,IXS,
     *              IXX,IXY,IXZ
C
      TOK POINTER     CHARACTER STRING
      VARIABLE        POINTED TO
      ------------    --------------
C     IXA             1HA
C     IXB             1HB
C     IXC             1HC
C     IXD             1HD
C     IXE             1HE
C     IXF             1HF
C     IXH             1HH
C     IXL             1HL
C     IXN             1HN
C     IXO             1HO
C     IXR             1HR
C     IXS             1HS
C     IXX             1HX
C     IXY             1HY
C     IXZ             1HZ
C
C     RELATIONAL AND LOGICAL OPERATORS
      COMMON/RELATN/IXALSO,IXAND ,IXEQ   ,IXGE   ,IXGT   ,IXLE   ,IXLT   ,
     *              IXNE   ,IXNOT ,IXOR1 ,IXPNEG,IXPOS
C
      TOK POINTER     CHARACTER STRING
      VARIABLE        POINTED TO
      ------------    --------------
C     IXALSO          9H.ANDALSO.
C     IXAND           5H.AND.
C     IXBEQ
C     IXBGE
C     IXBGT
C     IXBNE
C     IXEQ            4H.EQ.
C     IXGE            4H.GE.
C     IXGT            4H.GT.
C     IXLE            4H.LE.
C     IXLT            4H.LT.
C     IXNE            4H.NE.
C     IXNOT           5H.NOT.
C     IXOR1           4H.OR.
C     IXPNEG          5H.NEG.
C     IXPOS           5H.POS.
C
C
```

```
C      ARITHMETIC OPERATORS
       COMMON/OPERAT/IXDIVD,IXDLM6,IXEQL ,IXEXP ,IXMNUS,IXMULT,IXPLUS

C
       TOK POINTER     CHARACTER STRING
       VARIABLE        POINTED TO
       ------------    --------------
C      IXDIVD          1H/
C      IXDLM6          1H≤
C      IXEQL           1H=
C      IXEXP           2H**
C      IXMNUS          1H-
C      IXMULT          1H*
C      IXPLUS          1H+
C
C

C      UTILITARIAN CONSTANTS
       COMMON/UTILTY/IXBLNK,IXDBUG,IXFALS,IXNO  ,IXOFF ,IXON  ,IXOR  ,
      *              IXTRUE,IXYES

C
       TOK POINTER     CHARACTER STRING
       VARIABLE        POINTED TO
       ------------    --------------
C      IXBLNK          1H
C      IXDBUG          5HDEBUG
C      IXFALS          0
C      IXNO            0
C      IXOFF           0
C      IXON            1
C      IXOR            2HOR
C      IXTRUE          1
C      IXYES           1
C
C

C      MISCELLANEOUS DELIMITERS
       COMMON/MISC  /IXBL  ,IXCOMA,IXDLM1,IXDLM2,IXDLM3,IXDLM4,
      *              IXDLM5,IXDOLL,IXPAR ,IXPARD,IXPERD,IXPRDS,IXPRIM

C
       TOK POINTER     CHARACTER STRING
       VARIABLE        POINTED TO
       ------------    --------------
C      IXBL            1H
C      IXCOMA          1H,
C      IXDLM1          1H>
```

```
C      IXDLM2          1H*
C      IXDLM3          1H^
C      IXDLM4          1H<
C      IXDLM5          1H]
C      IXDOLL          1H$
C      IXPAR           1H(
C      IXPARD          1H)
C      IXPERD          1H.
C      IXPRDS          3H) )
C      IXPRIM          1H#
C
C

C      DIALECT REFERENCE SET
       COMMON/DIALCT/IXASC ,IXCFTN,IXCRUN,IXFTRN,IXIFTN,IXPDP ,IXSTRN,
      *              IXUNV

C
       TOK POINTER     CHARACTER STRING
       VARIABLE        POINTED TO
       ------------    --------------
C      IXASC           3HASC
C      IXCFTN          7HCDC-FTN
C      IXCRUN          7HCDC-RUN
C      IXFTRN          7HFORTRAN
C      IXIFTN          6HIFTRAN
C      IXPDP           3HPDP
C      IXSTRN          9HSTRUCTRAN
C      IXUNV           6HUNIVAC
C
C

C      HOLLERITHS USED IN REPORTING
       COMMON/REPORT/IXALL ,IXARAY,IXASGD,IXCONS,IXDTRG,IXEROR,IXFLNM,
      *              IXFNXC,IXFTAL,IXHOLL,IXKWN1,IXLABL,IXLOCL,IXNON1,
      *              IXNOTP,IXNUSD,IXOCTL,IXPRMT,IXSET ,IXTPLS,IXTYPE,
      *              IXUSDN,IXUSED,IXVRBL,IXWARN,IXWRTN

C
       TOK POINTER     CHARACTER STRING
       VARIABLE        POINTED TO
       ------------    --------------
C      IXALL           3HALL
C      IXARAY          5HARRAY
C      IXASGD          8HASSIGNED
C      IXCONS          8HCONSTANT
C      IXDTRG          7HDO-TARG
C      IXEROR          7H*ERROR*
C      IXFLNM          8HFILENAME
C      IXFNXC          10HFUNCT-CALL
C      IXFTAL          7H*FATAL*
C      IXHOLL          9HHOLLERITH
C      IXKWN1          8H*UNKNOWN
C      IXLABL          5HLABEL
C      IXLOCL          5HLOCAL
C      IXNON1          6H*NONE*
C      IXNOTP          10HNOT PARSED
C      IXNUSD          6HUNUSED
C      IXOCTL          5HOCTAL
C      IXPRMT          9HPARAMETER
C      IXSET           3HSET
C      IXTPLS          8HTYPELESS
```

```
C     IXTYPE          4HTYPE
C     IXUSDN          6HUNUSED
C     IXUSED          4HUSED
C     IXVRBL          8HVARIABLE
C     IXWARN          7HWARNING
C     IXWRTN          7HWRITTEN
C     IXXALL          7H**ALL**
C
C     LOGICAL CONSTANTS
      COMMON/LOGICL/IXFLSE,IXPF   ,IXT    ,IXTTRU
C
      TOK POINTER     CHARACTER STRING
      VARIABLE        POINTED TO
      ------------    --------------
C     IXFLSE          7H.FALSE.
C     IXPF            3H.F.
C     IXT             3H.T.
C     IXTTRU          6H.TRUE.
C
C
C     STANDARD ANSI  FORTRAN
      COMMON/ANSI   /IAGT   ,ICGT   ,IEND   ,IFAGT ,IFASGM,IFASS ,IFBACK,
     *               IFCALL,IFCLXT,IFENDF,IFGT  ,IFGTC ,IFP   ,IFPNCH,
     *               IFPRNT,IFR    ,IFREAD,IFRWND,IFS   ,IFT   ,IFWRTE,
     *               IF2    ,IF3   ,IGT   ,IRET  ,ISTOP ,IXASGM,IXASGT,
     *               IXASS ,IXBKSP,IXBLCK,IXBLK ,IXBUFI,IXBUFO,IXCALE,
     *               IXCALL,IXCMMN,IXCMMT,IXCMPX,IXCONT,IXDATA,IXDBL ,
     *               IXDMSN,IXDO   ,IXENDF,IXEQVL,IXEXIT,IXEXTN,IXFNCT,
     *               IXFRMT,IXGO   ,IXLGCL,IXIOAC,IXIF  ,IXIOWR,IXIORD,
     *               IXINTG,IXPAUS,IXPNCH,IXPRNT,IXREAD,IXREAL,IXRWND,
     *               IXSTFX,IXSUB ,IXTO  ,IXWRIT
C
      TOK POINTER     CHARACTER STRING
      VARIABLE        POINTED TO
      ------------    --------------
C     IAGT            10HGOTO-ASSGN
C     ICGT            10HCOMP. GO TO
C     IEND            3HEND
C     IFAGT           10HIF-ASSGN-G
C     IFASGM          10HIF-ASSGMNT
C     IFASS           9HIF-ASSIGN
C     IFBACK          10HIF-BACKSPA
C     IFCALL          7HIF-CALL
C     IFCLXT          10HIFCALLEXIT
C     IFENDF          10HIF-ENDFILE
C     IFGT            7HIF-GOTO
C     IFGTC           9HIF-GOTO-C
C     IFP             8HIF-PAUSE
C     IFPNCH          8HIF-PUNCH
C     IFPRNT          8HIF-PRINT
C     IFR             9HIF-RETURN
C     IFREAD          7HIF-READ
C     IFRWND          9HIF-REWIND
C     IFS             7HIF-STOP
C     IFT             2HIF
C     IFWRTE          8HIF-WRITE
C     IF2             6HIF-TWO
C     IF3             8HIF-THREE
```

```
C      IGT             4HGOTO
C      IRET            6HRETURN
C      ISTOP           4HSTOP
C      IXASGM          10HASSIGNMENT
C      IXASGT          8HASS-GOTO
C      IXASS           6HASSIGN
C      IXBKSP          9HBACKSPACE
C      IXBLCK          9HBLOCKDATA
C      IXBLK           5HBLOCK
C      IXBUFI          8HBUFFERIN
C      IXBUFO          9HBUFFEROUT
C      IXCALE          9HCALL-EXIT
C      IXCALL          4HCALL
C      IXCMMN          6HCOMMON
C      IXCMMT          7HCOMMENT
C      IXCMPX          7HCOMPLEX
C      IXCONT          8HCONTINUE
C      IXDATA          4HDATA
C      IXDBL           6HDOUBLE
C      IXDMSN          9HDIMENSION
C      IXDO            2HDO
C      IXENDF          7HENDFILE
C      IXEQVL          11HEQUIVALENCE
C      IXEXIT          4HEXIT
C      IXEXTN          8HEXTERNAL
C      IXFNCT          8HFUNCTION
C      IXFRMT          6HFORMAT
C      IXGO            2HGO
C      IXLGCL          7HLOGICAL
C      IXIOAC          10HI/O-ACTION
C      IXIF            5HIF-IF
C      IXIOWR          9HI/O-WRITE
C      IXIORD          8HI/O-READ
C      IXINTG          7HINTEGER
C      IXPAUS          5HPAUSE
C      IXPNCH          5HPUNCH
C      IXPRNT          5HPRINT
C      IXREAD          4HREAD
C      IXREAL          4HREAL
C      IXRWND          6HREWIND
C      IXSTFX          10HSTMT.
C      IXSUB           10HSUBROUTINE
C      IXTO            2HTO
C      IXWRIT          5HWRITE
C
C     COMMONLY IMPLEMENTED FORTRAN EXTENSIONS
      COMMON/COMEXT/IENT   ,IFBUFF,IFBUFI,IFBUFO,IFDCDE,IFENCD,IXASNM,
     *              IXBFFR,IXDCD  ,IXENCD,IXNLST,IXPRGM
C
      TOK POINTER     CHARACTER STRING
      VARIABLE        POINTED TO
      ------------    --------------
C     IFNT            5HENTRY
C     IFBUFF          8HIFBUFFER
C     IFBUFI          10HIF-BUFFERI
C     IFBUFO          10HIF-BUFFERO
C     IFDCDE          9HIF-DECODE
C     IFENCD          9HIF-ENCODE
C     IXASNM          10HASSGN-MULT
C     IXBFFR          6HBUFFER
C     IXDCD           6HDECODE
```

```
C      IXENCD          6HENCODF
C      IXNLST          8HNAMELIST
C      IXPRGM          7HPROGRAM
C
C      CDC-RUN COMPILER
       COMMON/CDCRUN/IFA    ,IFD    ,IFEOF ,IFQ    ,IFSL   ,IFSS   ,IFUNIT,
      *              IXACUM,IXDVDE,IXEOF ,IXIOCK,IXLIGT,IXQUOT,IXSENS,
      *              IXSNSL,IXSWCH,IXUNIT

C
       TOK POINTER     CHARACTER STRING
       VARIABLE        POINTED TO
       ------------    --------------
C      IFA             10HIF-ACCUMUL
C      IFD             9HIF-DIVIDE
C      IFEOF           6HIF-EOF
C      IFQ             10HIFQUOTIENT
C      IFSL            10HIF-SENSE-L
C      IFSS            10HIF-SENSE-S
C      IFUNIT          7HIF-UNIT
C      IXACUM          11HACCUMULATOR
C      IXDVDE          6HDIVIDE
C      IXEOF           3HEOF
C      IXIOCK          7HIOCHECK
C      IXLIGT          5HLIGHT
C      IXQUOT          8HQUOTIENT
C      IXSENS          5HSENSE
C      IXSNSL          10HSENSELIGHT
C      IXSWCH          6HSWITCH
C      IXUNIT          4HUNIT
C
C      CDC-6000 FORTRAN EXTENSIONS
       COMMON/CDCEXT/ICALLR,IFCLLR,IXIMPL,IXLEVL,IXOVLY,IXSUBR

C
       TOK POINTER     CHARACTER STRING
       VARIABLE        POINTED TO
       ------------    --------------
C      ICALLR          10HCALL-RETRN
C      IFCLLR          9HIF-CALL-R
C      IXIMPL          8HIMPLICIT
C      IXLEVL          5HLEVEL
C      IXOVLY          7HOVERLAY
C      IXSUBR          10HSUBR-RETRN
C
C      PDP-11 FORTRAN
       COMMON/PDP11 /IFRDC ,IFWTC ,IREADC,IWRTEC,IXBYTE,IXDEFI,IXDFRN,
      *              IXFDRN,IXFILE,IXFIND,IXINT2,IXLGC1,IXRDRN,IXRL4 ,
      *              IXRL8

C
       TOK POINTER     CHARACTER STRING
       VARIABLE        POINTED TO
       ------------    --------------
C      IFRDC           9HIF-READ-C
C      IFWTC           10HIF-WRITE-C
C      IREADC          9HREAD-COND
C      IWRTEC          10HWRITE-COND
C      IXBYTE          4HBYTE
```

```
C       IXDEFI          6HDEFINE
C       IXDFRN          10HDEFINE-RAN
C       IXFDRN          8HFIND-RAN
C       IXFILE          4HFILE
C       IXFIND          4HFIND
C       IXINT2          8HINTEGER2
C       IXLGC1          8HLOGICAL1
C       IXRDRN          8HREAD-RAN
C       IXRL4           5HREAL4
C       IXRL8           5HREAL8
C
C

C       UNIVAC COMMON
        COMMON/UNIVAC/IXABNL,IXDELT,IXEDIT,IXINCL,IXSTRT

C
        TOK POINTER      CHARACTER STRING
        VARIABLE         POINTED TO
        ------------     --------------
C       IXABNL          8HABNORMAL
C       IXDELT          6HDELETE
C       IXEDIT          4HEDIT
C       IXINCL          7HINCLUDE
C       IXSTRT          5HSTART
C

C       INTERNAL COMMON HOLLERITHS
        COMMON/INTRNL/IXDLMT,IXENDT,IXKYWD,IXOPTR

C
        TOK POINTER      CHARACTER STRING
        VARIABLE         POINTED TO
        ------------     --------------
C       IXDLMT          9HDELIMITER
C       IXENDT          9HEND-TOKEN
C       IXKYWD          7HKEYWORD
C       IXOPTR          8HOPERATOR

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


*DK END
```

4 STRUCTRAN-2 SOFTWARE COMPONENTS

Each of the following sections describes a STRUCTRAN-2 component: its function, the modules in the component, and the data structures used by the component. Additional information is provided in the STRUCTRAN-2 Software Analysis Collection **documentation.**

4.1 STRUCO

STRUCO consists primarily of the storage management function for STRUCTRAN-2. STRUCTRAN-2 is designed to handle large programs. This imposes a requirement for access to a large data base in mass storage; the desire for fast, efficient operations on active data establishes a need for a core-resident working storage capable of holding the module data tables for two or three modules at the same time. The large data base is maintained in random access files called libraries, with each library holding a collection of tables for related modules. Each table is fragmented into a fixed block size. In core the working storage consists of allocated blocks of storage which contain whole or partial active module data tables. Data transfer between the libraries and the working storage area, and between working storage and analysis programs, is controlled by the storage manager system.

The organization of the storage management component is shown in a functional diagram in Fig. 4.1. The interface between the analysis components and the active data tables in the working storage is handled by the access interface subfunction, which permits the storage and retrieval of data using a

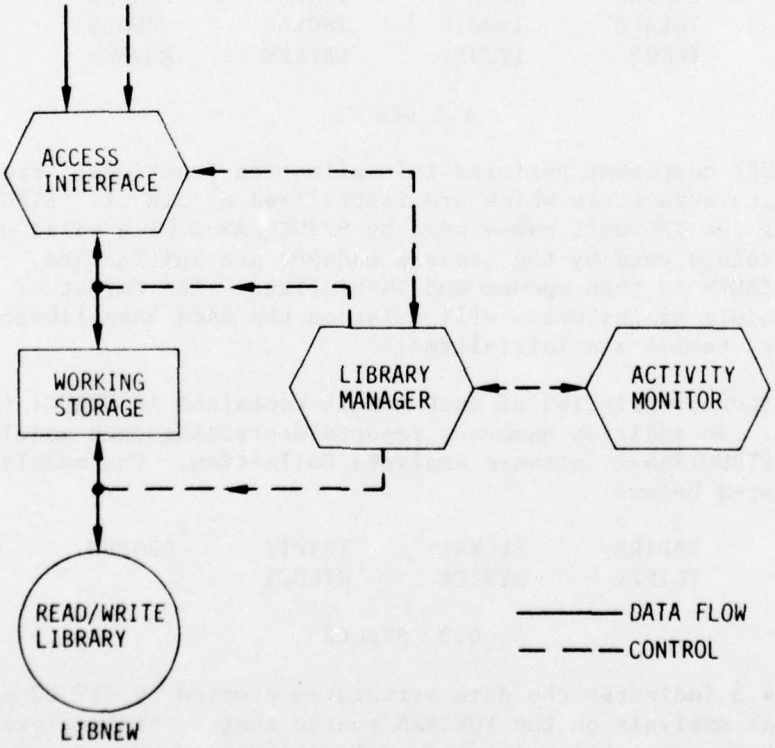


Figure 4.1. Storage Manager Organizations

module number, data table number and table block number to identify specific items. If the desired module tables are not present in the working storage when an access interface call is made, the library manager subfunction automatically responds by creating data tables of a new module or retrieving existing tables from the proper library. If space must be made in the working storage, the activity monitor subfunction tells which table fragment is a likely candidate for dumping from core. The library manager also handles the building of libraries by storing revised table fragments or newly entered module table fragments into the data base.

A narrative description of each module contained in STRUC0 is contained in Appendix A. In addition numerous reports describing each module can be found in the STRUCTRAN-2 Software Analysis Collection. The modules making up STRUC0 are listed below:

MAIN	LBZERO	ACTFRG	ACTOLD	KSTMTS	LGTWRD
WSGWRD	UPDEPT	WSGFRG	LBTGET	IHLCON	WARN
ACTMOD	LBTPUT	ACTICH	MDBGET	TRACE	ERPERF
PUTBLK	SYSADD	WSPWRD	LBREAD	PRMGXV	PRMDB
NMBARG	MKVTAB	LBWRIT	NFRGSZ	ISRTAB	MOVEWD
LPTBLK	MAKTAB	LBMOVE	MAKMOD	IGTBIT	GETFRG
LGTBLK	UNPX	MDBPUT	TOKCMP	COMPZ	ITSFRG
WSPFRG	ACTDMP	CRTFRG	IBAPAR	PUTWRD	FATAL1
SPRYWD	GETLBT	DMPFRG	KFLOWS	DMPMOD	PRDBG
MAKFRG	ISERCH	SLEDIT	GETLST	LENTAB	GETFOR
ICHFRG	PUTLST	PUTBIT	JGET	IGTTOK	IGTWRD
GETMOD	IMAKEP	EROR	JPUT	FINISH	GETBLK
GETTAB	TOKADD	ISDOTR	ISCLAS	NUMODS	ACTDAT
EDIT10	IFEOF	IFUNIT	LBFREE	MOVECH	REPLC1
XMIT					

4.2 STRUC1

The STRUC1 component performs initialization functions. Figure 4.2 indicates the data structures which are initialized by STRUC1. STRUC1 first assigns all of the I/O unit names used by STRUCTRAN-2 with valid unit numbers. Then various values used by the storage manager are initialized. The random access file LIBNEW is then opened and initialized. The format of the various library and module tables which will exist on the data base library is defined, and the library tables are initialized.

A narrative description of each module contained in STRUC1 is contained in Appendix A. In addition numerous reports describing each module can be found in the STRUCTRAN-2 Software Analysis Collection. The modules making up STRUC1 are listed below:

PRLBT	LBDIRS	SBLKSZ	TBINIT	LBOPEN	AIINIT
LBINIT	FLINIT	STRTER	STRUC1		

4.3 STRUC2

Figure 4.3 indicates the data structures created by STRUC2 as it performs a lexical analysis on the FORTRAN source text. This analysis identifies each FORTRAN keyword, label, operator, delimiter, constant, or variable as a token. The FORTRAN source text to be structurized is read in one statement at

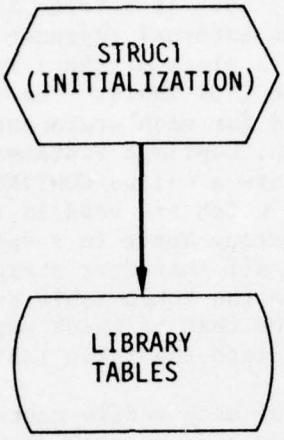


Figure 4.2. STRUC1 Data Structures

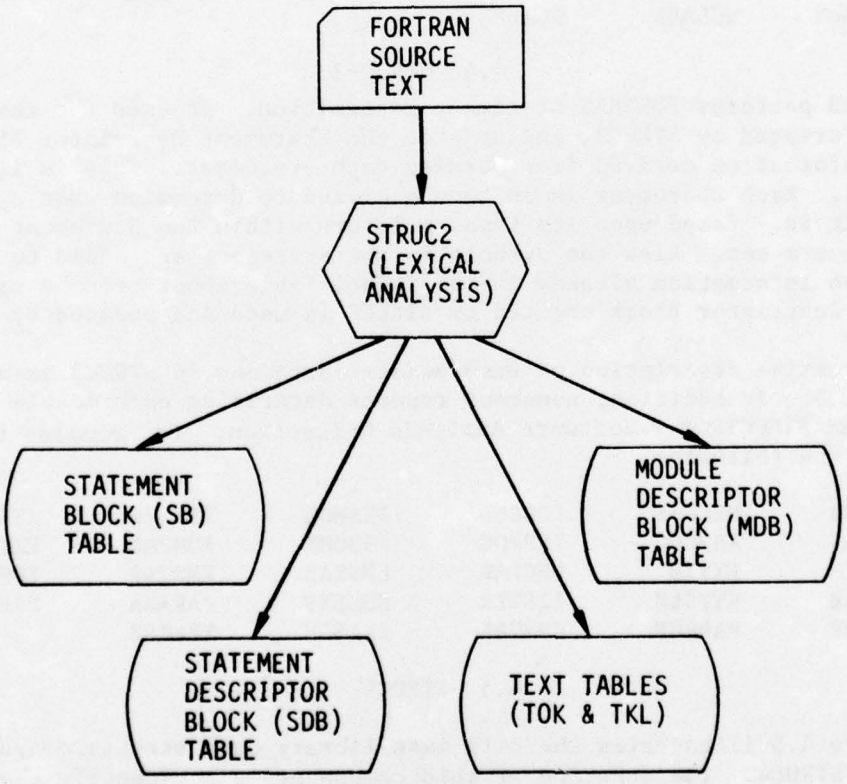


Figure 4.3. STRUC2 Data Structures

a time during the lexical analysis. If a token is not already in the token table it is added to it, and an internal representation of the statement containing token table pointers (in place of the original character string data) is saved in the Statement Block (SB) Table. The Statement Descriptor Block (SDB) Table is also initialized for each statement. During this initial pass through the FORTRAN source text, CONTINUE statements may be added to the source text so that each DO-loop targets a unique CONTINUE statement. For purposes of efficiency FORTRAN keywords which are used in recognizing FORTRAN statement types are added to the **Token String Table** in a specific order by STRUC2. For enhanced machine independence, all character strings which are used internally or used in reports are added to the token table by STRUC2. This is performed by the module TOKNIT, the source text of which explicitly indicates which character strings are inserted into the token table.

A narrative description of each module contained in STRUC2 is contained in Appendix A. In addition, numerous reports describing each module can be found in the STRUCTRAN-2 Software Analysis Collection. The modules making up STRUC2 are the following:

STRUC2	LEXPDP	OVRFLO	IHVALU	LEXEDT	LADJST
PUTFOR	LEXFIX	TOKSET	PSTPOP	POSTAK	CLASFY
DOLABL	LEXFR1	TOKNIT	ADMODL	MAINCM	ADCOND
ADLABLS	NULABL	SLABEL			

4.4 STRUC-3

STRUC3 performs FORTRAN statement recognition. It uses the text representations created by STRUC2, and updates the Statement Descriptor Block Table with the information derived from parsing each statement. This is illustrated in Fig. 4.4. Each statement is in turn analyzed to determine what type of statement it is. Based upon its type, pointers within the Statement Descriptor Block Table are set. Also the symbols in the statement are added to the Symbol Table or the information already in the Symbol Table about them is updated. The Module Descriptor Block created by STRUC2 is used and updated by STRUC3.

A narrative description of each module contained in STRUC3 is available in Appendix A. In addition, numerous reports describing each module can be found in the STRUCTRAN-2 Software Analysis Collection. The modules included in STRUC3 are the following:

STRUC3	MXCLAS	IOCOND	ITSMOD	ITSTYP	ITRGBL
ISETK	ASPROC	IFPROC	ISNONX	FUNPRC	EXTFIO
ISIF	EXTIO	DECLAR	LNSTAB	KNSTAB	IOPROC
INSTAB	STTBLK	I1STEX	NUMEXS	PARASA	PARUNV
PARPDP	PARRUN	KNSTBL	ISASGN	FPARSE	

4.5 STRUC4

Figure 4.5 illustrates the data base library data structures used and created by STRUC4. The function of this component is to identify the program graph of the module, using the information which has already been saved in the data base library. Each statement in turn is examined to see if it corresponds to a node in the graph from which one or more arcs (DD-paths) originate. For each arc originating at a node, the statements on that arc up until the next node (decision statement) on that path are determined. A temporary

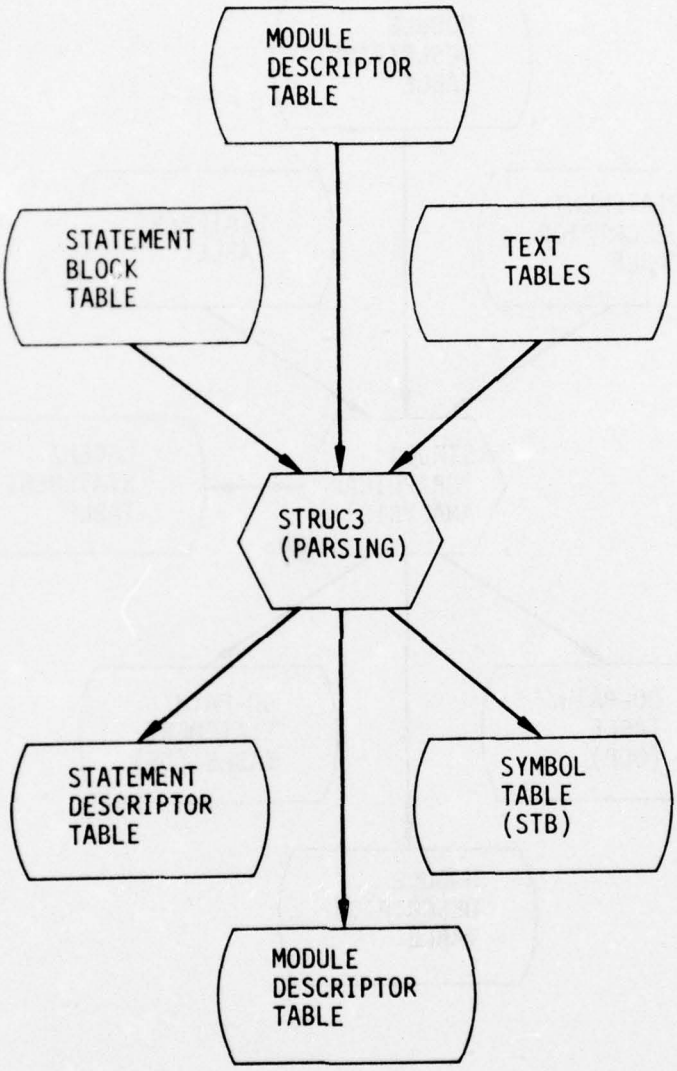


Figure 4.4. STRUC3 Data Structures

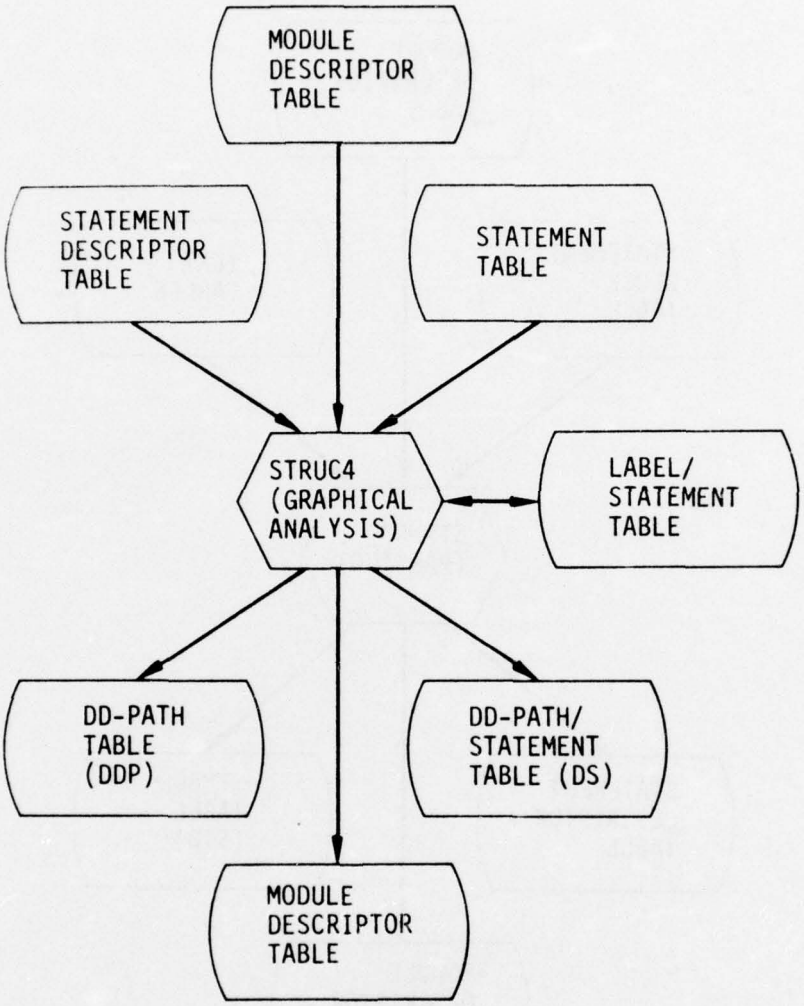


Figure 4.5. STRUC4 Data Structures

table, the Label/Statement Table, is created and used to efficiently associate labels with statement numbers. Each arc of the program graph is described by an entry in the DD-Path Table; the statements on each arc are saved in the DD-Path/Statement Table. The Module Descriptor Block is updated by STRUC4.

A narrative description of each module contained in STRUC4 is available in Appendix A. In addition, numerous reports describing each module can be found in the STRUCTRAN-2 Software Analysis Collection. The modules in STRUC4 are the following:

STRUC4	LSTEX	ITSCJX	ITSFNC	LABSTM	ABRTRN
NXTEX	STMOUX	BRCNT	GETBIT	DPSTMT	FRTRN2
PDPFT2	CDCRN2	FJNC2	STCK	FJNC1	PDPFT1
CDCRN1	FRTRN1	NOTRCH	NXTSTM	POPDDP	BEGDDP
FNDEND	DDPGEN	PRDDP	KJONES		

4.6 STRUC5

STRUC5 identifies and reduces elementary structured forms in the program graph of the module being analyzed. To do this it uses the information stored in the DD-Path and DD-Path/Statement Tables. It also updates the information in these tables to reflect the structured forms identified. This is indicated in Fig. 4.6. Several temporary tables are created and used by STRUC5 during this process. The identification of structured forms proceeds from the inside out. When a structured form (a single entry/single exit structure consisting of several arcs) is recognized it is removed from the graph by replacing it with a single arc with the same entry and exit nodes. This allows other basic constructs to be identified. As this reduction process is performed, the DD-Path and DD-Path/Statement Tables are updated with the types of structured forms identified. Two temporary tables are used to represent the current program graph during the reduction process. Also temporary connection tables are used to efficiently analyze the more complicated graphical structures which may be encountered.

A narrative description of each module in STRUC5 is contained in Appendix A. In addition, numerous reports describing each module can be found in the STRUCTRAN-2 Software Analysis Collection. The modules in STRUC5 are the following:

STRUC5	BAKPTH	NPTH	NODSTK	FNDCON	INTPOP
INTSTK	NODPOP	MLTWHL	IBORCD	ITORIF	NPAR
NBEF	NAFT	REORDR	LOOPCK	INTRVL	MAKCON
ADDSLF	REMENT	SUBSCH	ISITUN	IBORNT	ADDJNC
ADDBRO	ADDLST	ADDLIN	SEME	ADDWHL	PRTEMP
ADDPAR	REMBRO	SHRINK			

4.7 STRUC6

STRUC6 uses all of the information saved so far to build a DMATRAN version of the original FORTRAN module. Figure 4.7 indicates the data structure usage. The DMATRAN source code is written into the Statement Block for a new module, and the Statement Descriptor Block Table and Module Descriptor Block Table are initialized so that the DMATRAN program may be output by standard print and punch routines.

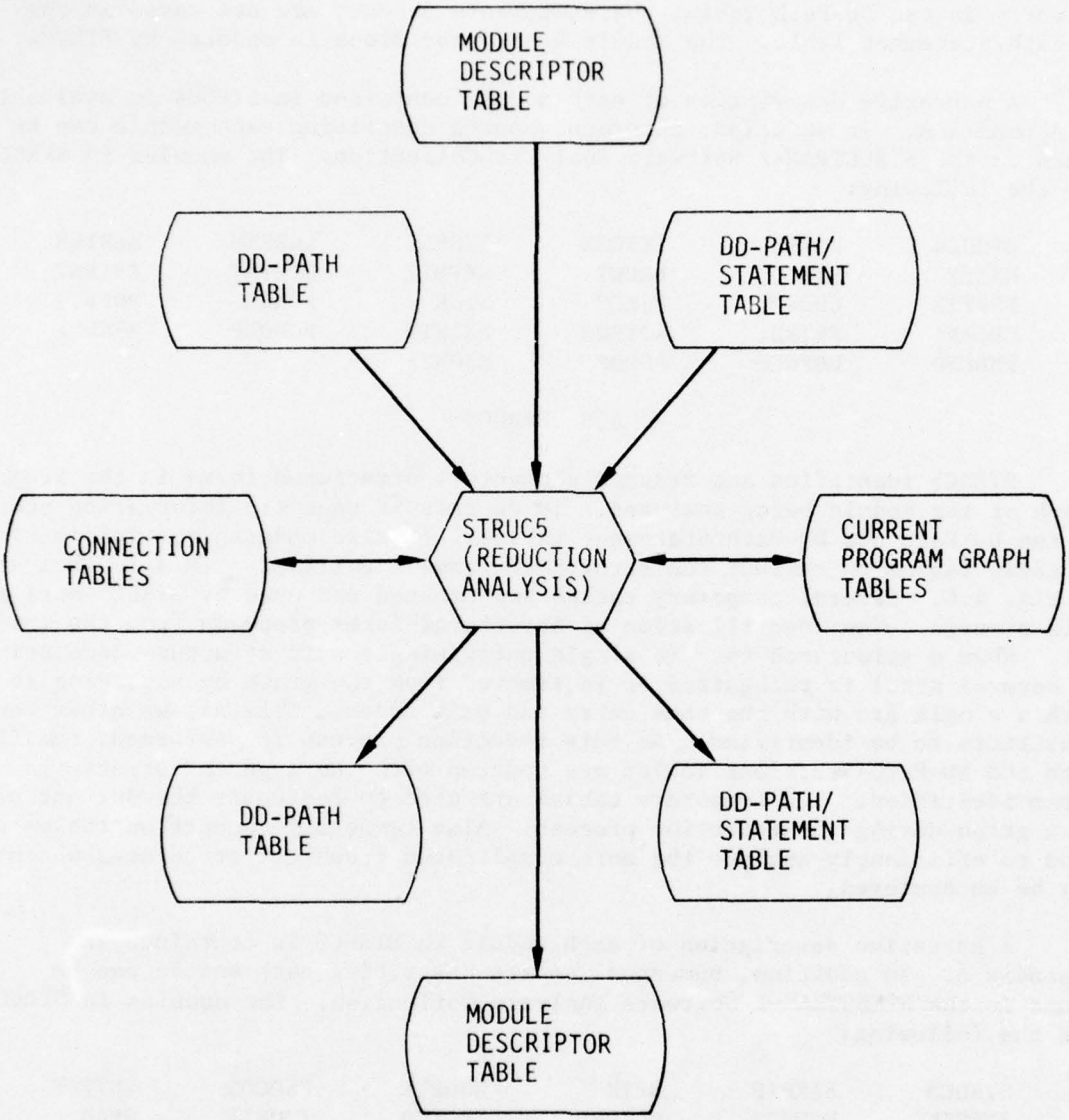


Figure 4.6. STRUC5 Data Structures

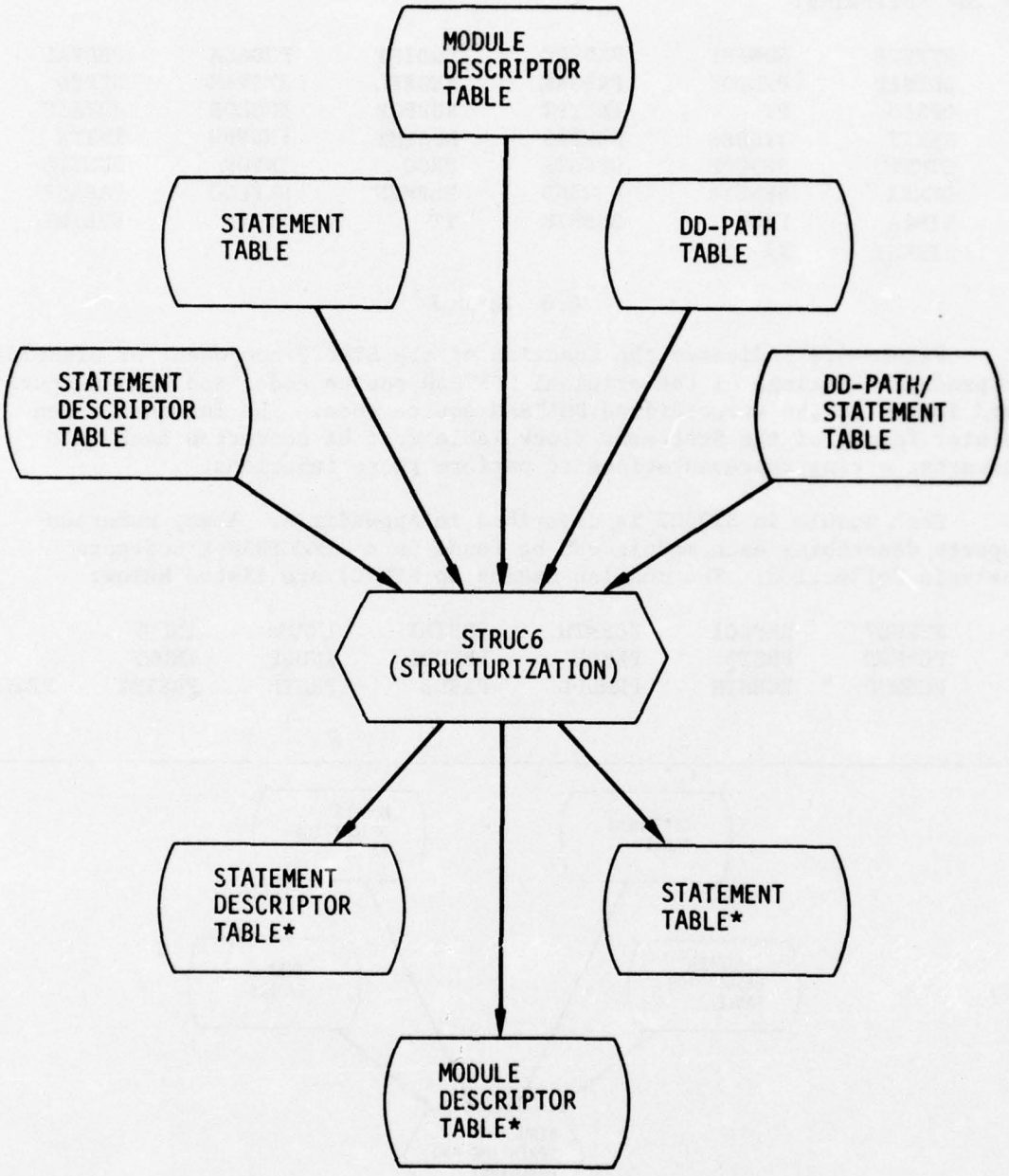


Figure 4.7. STRUC6 Data Structures

A narrative description of each module in STRUC6 is contained in Appendix A. In addition, numerous reports describing each module can be found in a STRUCTRAN-2 Software Analysis Collection. The modules in STRUC6 are the following:

STRUC6	NUNAME	PRDSPC	PRDIFT	PRDASA	PRDVAL
DDPRED	PRDPOP	PRQSTK	NEGREL	ITSVAR	NPRED
DPRED	PT	INITPT	SUBPOP	ENCLOS	JHVALU
NEXIT	ITSDES	NEGPRD	NUSTMT	FNDPRD	INITX
STMTOU	SESPOP	SESSTK	PROC	INVOK	DUNTIL
SEMEX	NEWFIR	NEWSND	ELEMEN	SLFLOO	PARALE
LINEA	PROG	SUBSTK	TT	INITT	SUBINT
STRUCT	FINDSB				

4.8 STRUC7

Figure 4.8 indicates the function of the STRUC7 component of STRUCTRAN-2. It produces listings of the original FORTRAN source code, and also outputs card images of the structurized DMATRAN source code. The internal token pointer format of the Statement Block Table must be converted back into character string representations to perform these functions.

Each module in STRUC7 is described in Appendix A. Also, numerous reports describing each module can be found in a STRUCTRAN-2 Software Analysis Collection. The modules making up STRUC7 are listed below:

STRUC7	REPLC1	PCHMTH	PRSTMT	INDWN	INDUP	
PCHMOD	PRSTB	PRSDB	PRMODL	INDUP	INDWN	
PCHMOD	PCHMTH	PRMODL	PRSDB	PRSTB	PRSTMT	PRTOKS

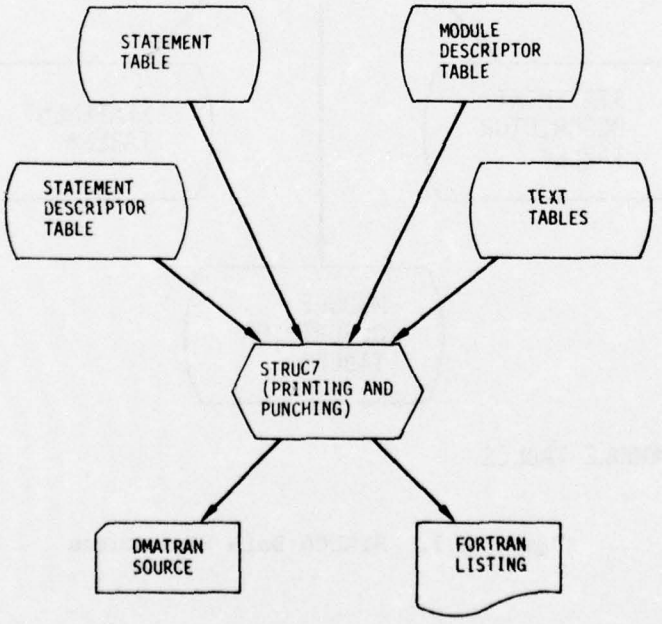


Figure 4.8. STRUC7 Data Structures

APPENDIX A

STRUCTRAN-2 MODULE FUNCTION DESCRIPTIONS

ABRTRN determines the initial statements on DD-paths which begin with abnormal returns from CALLs or I/O conditional statements.

ACTDAT is an Activity Monitor procedure. It removes a given module from the array of active modules (LACTIV).

ACTDMP is an Activity Monitor procedure. It removes entries from the working storage fragment directory list. If the parameter MODULE is negative, its absolute value is the fragment being deactivated. If the parameter is positive, it is the module being deactivated. In this case, a search of the working storage fragment directory is made until a fragment for the given module is found. In both cases, the entry in the working storage fragment directory is deactivated and the working storage fragment number, and its corresponding library fragment number is returned. In the second case, if no fragment for the module is found, a fragment number of zero is returned.

ACTFRG is an Activity Monitor procedure. If the given working storage fragment is a fragment of a library table (less than LBFRGS), the working storage directory is updated to contain the proper corresponding library fragment pointers. If the fragment is in the module area of working storage, the directory cell is made "most active" and the library fragment pointers are updated, if necessary.

ACTICH is an Activity Monitor procedure. This procedure sets an indicator in the working storage fragment directory that the given fragment has been changed since being fetched into core. This is done by making the library pointer negative.

ACTMOD is an Activity Monitor procedure. This procedure inserts and removes modules in the active module list (LACTIV). If the given module number is zero, the least active module is removed from the list. Its module number and MDB location in the working storage is returned. If a module number is given, it is inserted into the list if it is not already present. If it is necessary to remove a module from the list, its module number (NOLD) and MDB location (MDBLOC) is returned. (If it is not necessary to remove a module, NOLD is returned with a value of zero.) The given module is moved to the top of the list (LACTIV) to signify that it is the most active module.

ACTOLD is an Activity Monitor procedure. ACTOLD finds a likely fragment to remove from the working storage. To do this, it searches the working storage fragment directory for the least active fragment. It returns the fragment number and its corresponding library fragment and releases the entry in the fragment directory.

ADCOND checks READ and WRITE statements for END= and ERR= and calls RWCOND if either is found.

ADDBRO searches the current program graph for examples of arcs which have the same initial and final nodes, replaces them by a single entry/single exit (SESE) structure, and records the reduction by adding an entry to the DDPATH Table.

ADDJNC searches the current program graph for elementary examples of borrowed code arcs, replaces them by a well-structured SESE structure, and records the reduction in the DDPATH Table.

ADDLIN searches the current graph representation for examples of vertices which have only one inway and one outway, replaces such examples by a single SESE structure, and indicates the reduction by adding an entry to the DDPATH Table.

ADDLST summarizes the final state of the current program graph as the root of the SESE tree.

ADDPAR controls the reduction process for simple, non-iterative SESE structures. The process terminates when no further reductions can be performed.

ADDSLF removes self-loop arcs which have only one exit arc from the current program graph, and records the reduction in the SESE tree.

ADDWHL converts basic repeat-at-least-once loops to an elementary SESE form and records the reduction.

ADLBLS adds the statement label from a FORTRAN DO statement to the label stack and calls EROR with error number 5102 if the stack overflows. ADLBLS checks to see if the DO target is a CONTINUE statement. If it is not, then a CONTINUE statement is added and a label defined.

ADMODL initiates the lexical analysis of a new module and stores such information as source language type, statement blocks, statement labels, date of first analysis, number of statements, module name and intra-statement pointers for each module on the library.

AIINIT is a Starter Initialization procedure. It initializes storage used by the access interface and the Activity Monitor.

ASPROC parses assignment statements, calls and statement functions. The Symbol Table Block (STB) and Statement Descriptor Block (SDB) on the library are updated as a result of this parsing.

BAKPTH backs up the last path which was found in a subset of the current program graph until an arc which was not previously taken is found.

BEGDDP determines whether any DD-paths begin at ISTMT. If so, a stack is set up to indicate the initial statements on each DD-path.

BRCNT counts the number of branch labels in IF-COMPUTED-GOTO, COMPUTED-GOTO statements, IF-ASSIGNED-GOTO, and ASSIGNED-GOTO statements.

CDCRN1 tests for hardware-IF statements (IF-SENSE-L, IF-SENSE-S, IF-ACCUMUL, IF-QUOTIENT, IF-DIVIDE, IF-EOF). If any are found, the initial statements beginning each DD-path are returned.

CDCRN2 also tests for hardware-IF statements and and sets IACT to 1 if any are found.

CLASFY identifies delimiters, operators and FORTRAN keywords in unprocessed text stream.

COMPZ converts A1 characters into packed A6 format.

CRTFRG is a Library Manager routine. It zeros out a new fragment on a given library and returns the fragment number. If necessary, it creates an additional fragment for the library's fragment directory table.

DDPGEN controls the generation of the original program graph and builds the DDPATH Table. The statements in the module are analyzed sequentially to see how many DD-paths originate at a given statement and to find the statements on each such DD-path.

DDPRED constructs the predicate for a given DD-path number by calling PRDVAL with the computed predicate index.

DECLAR processes statements which are identified as declarations (DIMENSION, TYPE, DOUBLE PRECISION, etc.). The STB in the library is updated for each symbol in the declarations.

DMPFRG is a Library Manager procedure. DMPFRG removes a given working storage fragment from core. If the fragment has been altered since being fetched into core, it is written to the given library fragment. If necessary, it is removed from the access interface "active fragment" storage. The fragment directory table is changed to indicate the module is no longer in the working storage.

DMPMOD is a Library Manager routine. It removes all data fragments pertaining to a given module from the working storage, including the MDB, and sets the Module Location Table to indicate the module is no longer in the working storage.

DOLABL computes unique statement numbers for CONTINUE statements which were added as DO-targets during the previous pass through the source text.

DPRED calls DDPRED with its own formal parameters.

DPSTMT computes the list of statement numbers which reside on a specified set of DD-paths.

DUNTIL constructs a DO UNTIL or END UNTIL statement for the structurized module, depending upon the value of NBLOK(3).

EDIT10 edits imbedded blanks out of A6 character strings, saving a leading blank character and padding the last word with trailing blanks.

ELEMEN generates the structurized statement text for statements on a specified DD-path. ELEMEN adds each statement except the last one on a DD-path to the DMATRAN program being written, modifying the statement when necessary (as for ASSIGN statements).

ENCLOS encloses a given string of length L1, stored in token array LIST1, with a set of parentheses and stores the new string in token array LIST.

EROR increments the error counter by one and passes the identifying error number (an input argument) to the error printing routine, PRMGXV.

ERPERF is a print routine which outputs the error processing statistics.

EXTFIO determines if a FORTRAN extended I/O statement is an IF-READ conditional or an IF-WRITE conditional and stores the appropriate statement type and pointer to the read or write token in the SDB. EXTFIO calls IOPROC which processes the I/O statement.

EXTIO determines if an extended I/O statement is a READ or WRITE conditional and stores the appropriate statement type and pointer to the parameter list in the SDB. EXTIO calls IOPROC which processes the I/O statement.

FATAL1 prints the fatal error message, calls the error printing routine, PRMGXV, with the error number, calls the traceback routine, TRACE, and sets the number of fatal errors to one.

FINDSB performs a tree traversal of the SESE tree. During this process any large segments of code which would otherwise be duplicated are noted so that the BLOCK...END BLOCK and INVOKE statements can be used to control code expansion. The mechanism for doing this is an internal subroutine stack implemented in the SUBSTK routine.

FINISH closes the libraries and saves files prior to termination of each processing step.

FJNC1 determines the initial statements on a DD-path which originates at a junction node in the program graph.

FJNC2 determines whether ISTMT is a junction node which terminates the DD-path currently being formed.

FLINIT initializes the standard files used and prints the file names and descriptions.

FNDCON builds a compact table of arc connections in the current program graph. The program graph may be interpreted as a directed graph in one of two directions.

FNDEND builds a label table for each module for subsequent processing and verifies the presence of an END statement in the module. If there is no END statement, EROR is called with error number 8002, but processing continues.

FNDPRD performs the minimum tree traversal required to determine the DD-paths associated with a decision node in the SESE tree. The DD-paths are then used to build the actual predicate required.

FPARSE parses a module which has already been stored in the statement blocks and builds appropriate tables. FPARSE loops over the module statements and determines if a statement is a comment or an assignment. Statement labels are stored in the SDB and STB. Depending upon which FORTRAN compiler is being used, the statement is further parsed by PARRUN (RUN compiler), PARPDP (PDP-11 FORTRAN), PARUNV (Univac), or PARASA (ASA FORTRAN). The language dialect, number of executable statements, and number of the first executable statement are stored in the MDB. STTBLK is called to construct block pointers.

FRTRN1 determines whether ISTMT is an ANSI Standard statement type which begins one or more DD-paths. If so, it determines the initial statements on each such DD-path according to the statement type.

FRTRN2 finds the next executable statement for the following conditions: a CONTINUE statement which is not a DO target, a GOTO label statement, a RETURN, END, CALL EXIT, or STOP statement, or any statement not an IF-type. It finds the next statement in a DD-path when the current statement is one of these types.

FUNPRC determines if a subprogram is a block data (labelled or unlabelled) subroutine, a subroutine, program, or function. The number of arguments, if any, are determined and the MDB updated accordingly. The STB is updated with the arithmetic mode of the subprogram, the number of actual parameters and the use class.

GETBIT gets the ith bit from a bit string. GETBIT calls routine UNPX.

GETBLK is an Access Interface procedure used for retrieving a block of information from a given module table. The procedure first checks to see that the module is active for that table. It then makes certain the proper fragments of that table are in the working storage. It returns the proper block in a given array.

GETFOR puts the next complete FORTRAN statement into array LIST, in A1 format. Leading and imbedded blanks are preserved, but trailing blanks are removed. GETFOR reads the statements from unit LUNIT. If a statement uses more than 20 cards, EROR is called with error number 7502.

GETFRG is a Library Manager routine which activates the proper fragment for a given module table. It threads its way through the proper library fragment table until it locates the correct fragment. If that fragment is not already in the working storage, it retrieves it from the library, dumping another fragment from the working storage if necessary. Access interface "active fragment" data is updated.

GETLBT is a Library Manager procedure which activates the proper fragment for a given library table. If a fragment for the table type is active, that fragment is rolled out of the working storage. The proper fragment is located in different ways. FDT fragments for the read only library are stored

successively on the workspace library. Other libraries' FDTs have their fragments spaced along the library. If one fragment of the FDT has 250 entries, there is a fragment for the FDT every 250th fragment (always starting at fragment one). For the entry point table, the last cell in the fragment contains a pointer to the next fragment. Once the proper fragment is located, it is brought into the working storage and the access interface "active fragment" storage is updated.

GETLST is an Access Interface procedure used for retrieving a variable-length block of information from a given module table. The procedure first checks to see that the module is active for that table. It then gets the proper table fragment into the working storage and retrieves the list out of it. If the list spans more than one fragment, successive fragments are retrieved. It returns the list in a given array.

GETMOD is a Library Manager procedure. It fetches the Module Descriptor Block for a given module into the working storage. If necessary it dumps another module from the working storage to make room. Access interface "active table" pointers are updated.

GETTAB is a Library Manager procedure. For a given table of a given module, after insuring the module is resident in the working storage, it updates the access interface "active table" pointers.

IBAPAR is a function which returns an integer value pointer to the first balancing right parenthesis for the first left parenthesis in a given LIST of length L.

IBORCD is a logical function which is TRUE or FALSE depending on whether a borrowed code construct has been found.

IBORNT determines whether a self-loop construct has been found.

ICHFRG is a Library Manager procedure. It is called when a table is changed. If the module to which the active table is on the read-only library, ICHFRG moves all information for that module to the read/write library. All data for the module is first dumped from the working storage. Then, using a buffer fragment in the working storage, each fragment for each table is moved. Finally the MDB for the module is put onto the read/write library.

IFEOF is a function which checks for an end-of-file condition on unit LUNIT and returns a 1 for an EOF and a 0 otherwise.

IFPROC parses the ANSI Standard IF statement types. It decides which type of IF statement is being analyzed and extracts and saves appropriate information for each type.

IFUNIT is a function which checks to see if unit LUN is busy and returns when LUN is in ready status.

IGTBIT is a function which returns the ith bit of array IARRAY. IGTBIT uses routine UNPX.

IGTTOK is a function which returns the first word of the specified block of the token table in A6 format.

IGTWRD is a function which returns a word from the specified table, block and index. If the specified index or block are beyond the length of the table, or if the table has not been made, calls to EROR with errors 5204, 5205 or 5206 will result.

IHLCON converts an integer constant (ITEM) into a left-adjusted Hollerith constant in A6 format. If used as a function, ITEM is destroyed.

IHVALU is a function which computes the integer value of its Hollerith integer argument, INPUT. INPUT is not destroyed during execution of IHVALU.

IMAKEP is a Library Manager procedure. It creates a new module in the JAVS system, setting up a new entry in the Module Location Table.

INDUP is a function which returns a YES or NO value depending upon whether the input parameter, ITYPE, represents DMATRAN statement types IF THEN, DO WHILE, ELSE, DO UNTIL, and BLOCK.

INDWN performs the same purpose as INDUP, except that INDWN checks for DMATRAN statement types END WHILE, END IF, END UNTIL, and END BLOCK.

INITPT initializes the stack for walking the SESE tree and building an appropriate predicate.

INITT initializes the stack which is used in walking the SESE tree.

INITX builds the statement NEXIT = 0 and passes it to NUSTMT to add to the restructured module.

INSTAB is a function which searches the STB for a specified symbol, IARRAY(1). If it is found, INSTAB = YES, the block number is returned in MTH, and the contents of the MTH block are returned in IARRAY. If the symbol is not found, the contents of IARRAY are added to the end of the STB.

INTPOP takes an integer off the top of stack INSTAK and stores it in INT. If the stack is empty, then INT = 0.

INTRVL identifies the next interval in the current program graph. The purpose for this is to identify non-trivial iteration structures which have more than one exit vertex.

INTSTK puts the input parameter INT on the top of stack INSTAK. If the stack overflows, EROR is called with error number 8013.

INVOK adds an appropriate INVOKE statement to the DMATRAN program being written. The appropriate block name to use is generated by INVOK.

IOCOND processes statements which are potential I/O conditional statements in order to send the labels which follow END and ERR to subroutine KNSTBI.

IOPROC parses the various ANSI Standard input/output statements. The statement type is identified and Symbol Table entries are created or updated appropriately.

ISASGN is a function which returns a YES or NO value depending upon whether a statement is an assignment statement.

ISCLAS is a function which returns a 0 if the input symbol is alphabetic, 1 if it is a digit, 2 if it is a FORTRAN special character (+ - * / ** ,), and 3 if it is another symbol.

ISDOTR is a function which returns a YES or NO value depending upon whether the ith statement in a specified module is a DO target.

ISERCH is a function which searches the LIST array for the first occurrence of the specified symbol, ISYM, and returns the subscript of the matching element in LIST. ISERCH is 0 if there is no match.

ISETK is a function which returns a value that represents the word-pair pointer to the first variable in a READ or WRITE argument list.

ISIF is a function which returns a YES or NO value depending upon whether a statement is an IF-THEN statement type.

ISITUN determines whether a portion of the current program graph can be represented as a DO UNTIL type of iteration structure.

ISNONX is a function which returns a YES or NO value depending upon whether a statement is a nonexecutable statement.

ISRTAB is an Access Interface procedure for searching a given data base table for an item with a given value. It searches item INDEX from block IBEG to block IEND. It first insures the proper table is active and retrieves the first fragment of table information to be searched into the working storage. It brings in successive fragments of the table to search if it does not find the value. When an item with a proper value is found, its block number is returned. If no item is found before IEND block is reached, a value of zero is returned.

ITORIF is a dummy routine.

ITRGBL is a function which returns the statement number that is the target of a WHILE, ELSE, IF-THEN, or FORTRAN DO statement. If no target can be found, EROR is called with error number 6201, indicating that there is no BLOCK structure, and ITRGBL is set to 0 .

ITSCJX determines whether a comment statement should be treated as a junction node and thus begin or end DD-paths.

ITSDES is a function which returns a YES or NO value depending upon whether the specified statement contains a predicate.

ITSFNC determines whether an arbitrary statement should be treated as a junction node and thus begin or end DD-paths.

ITSFRG is an access interface routine which returns the fragment number and fragment block for a given table and block number.

ITSMOD is a function which returns the value *UNKNOWN.

ITSTYP is a function which determines the type of a given symbol: either a variable, constant, or unknown type.

ITSVAR is a function which returns a 1 if the input parameter ITEM is in the STB for the given module and a 0 if it is not in the STB.

I1STEX is a function which returns the statement number for the first executable statement in a given module. The statement number is written to file LOUT. If there are no executable statements, an appropriate message is written to LOUT, and I1STEX is returned as 0 .

JGET is a function which extracts the Lth symbol from a packed A6 format character string, LIST, and returns the character in A1 format.

JHVALU is a function which adds an integer J (0-9) to the token table. If J is not 0-9, then EROR is called with error number 8021, a range error.

JPUT inserts the leftmost character of LIST into position LET of the packed character string (LIST) originating at L .

KFLOWS computes the number of graphical flows in an SESE construct.

KJONES is the driver for DD-path generation. KJONES calls DDPGEN to generate the DD-paths and PUTWRD to store the number of paths in the MDB. The number of DD-paths is written to LOUT. If the compiler language of the given module is not found to be FORTRAN, then EROR is called with error number 501. If the STATUS flag was set in DDPGEN, indicating abnormal graphical analysis, EROR is called with error number 502.

KNSTAB is called for a given ith word and mth statement of a module. If the given symbol is not yet in the STB, it is added to the STB and the arithmetic mode, symbol type, and symbol class determined. If the symbol is already in the STB, the symbol type and class are updated, if the symbol is an external.

KNSTBL is called with a given label in a module. If the label is not already in the STB, it is added, along with the statement number of its first occurrence. If the label was already in the STB, the number of its occurrences is incremented by one, and the statement number of its last occurrence is updated. If the label is not a DO-target, its STB use class is typed as used.

KSTMTS computes the number of statements in an SESE construct.

LABSTM converts LABEL (which was obtained as a label in some FORTRAN control statement) to the corresponding statement number at which the label is defined.

LADJST left-adjusts a list by removing all leading occurrences of a specific symbol, ISYM.

LBDIRS is a Library Manager procedure which builds and initializes Library Directories within the STRUCTRAN-2 startup sequence. The Fragment Directory Table for the read-only library is first moved to the workspace library, for it must be changed during the STRUCTRAN-2 run. The Module Location Table is then constructed, entering the modules off the read-only library and/or the modules off the read/write library. If both a read-only and read/write library exist, as modules are entered from the latter, a check is made to see if that module was already entered from the read-only library. If it was, the MLT entry is updated to point to the newest version; if not, a new MLT entry is made. The run-time module numbers for modules on the read/write library are entered into its EPT. For modules taken off the read-only library, their run-time numbers are equal to their library index.

LBFREE is a Library Manager procedure. It releases a module from the working storage and can be called by a user.

LBINIT is a Starter Initiator procedure. It initializes variables used by the Library Manager.

LBMOVE is a Library Manager procedure. It moves a fragment on one library to a fragment of another library using a given fragment in the working storage as a buffer.

LBOPEN is a Library Manager procedure which opens libraries to be used in a run during the startup sequence. It is passed the names of the read-only library (NAMOLD) and the read/write library (NAMNEW). If either name is blank, it is assumed the user is not supplying a library of that type for the present run. The read-only library is opened first, if necessary. Its library header is read off the library into array LIBTAB. The read/write library is then opened. If the user has not supplied one, a scratch read/write library is opened. For a user library to be created this run (or in the case of a scratch library), the Library Header Table is built. For an later library, it is read off the library. An alter library must be opened with the same name as it was created with in a previous run or a fatal error will result.

LBREAD is a random I/O read routine. It reads NWDS words from library LIBNUM, starting at relative address LIBADD and stores the data into array CORARY, starting at position CORLOC. If LIBNUM is not a legal library, EROR is called with error number 5313.

LBTGET is a Library Manager procedure. For a given library table, it retrieves the table length and first fragment index from the library header information storage.

LBTPUT is a Library Manager procedure. For a given library table, it stores the table length and first fragment index in the library header information storage.

LBWRIT is a random I/O write routine. It writes NWDS words into library LIBNUM, starting at address LIBADD. The data comes from array CORARY, starting at position CORLOC. If LIBNUM is not a legal library number, EROR is called with error number 5314.

LBZERO zeroes out a given library fragment.

LENTAB is an Access Interface function which returns the number of words in a block of a given table.

LEXEDT takes LS number of symbols in A1 format and edits them into A6 word-pairs for the SB. Each newly edited word is added to the token table.

LEXFIX is a logical function which returns TRUE if the Ith word in the given SB contained 1H'. LEXFIX calls LEXPDP which performs the search for 1H'.

LEXFR1 constructs the SB for FORTRAN blank- or special character-delimited programs.

LEXPDP performs the proper lexical analysis for special characters such as 1H1 and 1HC.

LGTBLK is a Library Manager procedure used for retrieving a block of information from a given library table. It first insures the proper fragment is in the working storage and then returns the proper block of that fragment in a given array.

LGTWRD is a Library Manager procedure which retrieves an item from a block of a given library table. It first insures the proper fragment is in the working storage and then returns the chosen item from that fragment.

LINEA is a dummy routine which returns upon entry.

LNSTAB searches the SB for a given statement, starting with word position IBASE and ending with position ILONG. If the symbol category (lower wordpair of the SB) is 0, then KNSTAB is called to see if the symbol resides in the STB. For entry LNSTBL, KNSTB1 is called to check if the label resides in the STB.

LOOPCK examines an interval as identified by INTRVL. It determines whether the interval contains an iteration, and if it does converts it into a single exit form by making all exit arcs pass through one of the exit vertices.

LPTBLK is a Library Manager procedure used for storing a block of information into a given library table. It first retrieves or creates the proper fragment into the working storage and then stores the data block into it.

LPTWRD is a Library Manager procedure which stores an item of a block into a given library table. The proper fragment is first retrieved or created in the working storage and then the item is stored into it.

LSTEX is a function which returns the statement number of the executable statement just prior to the given ISTMT.

MAIN is the main program which invokes each overlay.

MAINCM is a labelled block data routine. MAINCM contains labelled commons for all global variables. Most of the variables are set in data statements in subroutine TOKNIT. MAINCM contains the data statements used to initialize the number of errors and warnings allowable and the file declarations.

MAKCON builds a compact table which represents the current program graph. The purpose in doing this is to be able to efficiently analyze the flow through a subset of the current program graph.

MAKFRG is a Library Manager routine which appends a new fragment to a given module table. A new fragment is created on the proper library, the end of the original table is found, and the Fragment Directory Table is updated to link the new fragment to the table end.

MAKMOD is a Library Manager procedure which establishes a new module MDB on a given library. A new MDB is added to the library's MDB chain, creating a new fragment if necessary. An **Entry Point Table** entry is added, updating it completely if the module information already exists on some other library. The Module Location Table is also updated.

MAKTAB is a Library Manager procedure which defines a new data base table type. For this the user supplies a table name, two items of the MDB to be used for pointers by the Library Manager (or two cells in the library header if it is a library table) and a table block length. The procedure returns the assigned table number which must be used in the subsequent **Access Interface** calls to that table.

MDBGET is a Library Manager procedure. For a given module table, it retrieves from the MDB the table's present maximum length and a first library fragment index.

MDBPUT is a Library Manager procedure. For a given module table, it stores in the MDB the table's maximum length and its first library fragment index.

MKVTAB is a Library Manager procedure which converts a data base table defined by a call to MAKTAB into a variable length table. For this the user supplies the assigned table number, the maximum number of blocks per entry in the table, the table number of an auxiliary table, two items in the auxiliary table that the Library Manager can use for pointers, and an offset where one wants the Library Manager to store its length cell.

MLTWHL implements the mechanism which remembers that an SEME iteration structure is being reduced.

MOVECH moves NCHARS number of characters from position 1 of array IARY1 to array IARY2, starting in position INDEX.

MOVEWD moves NWDS number of words from the IPOS position of IARRY to the IPOS1 position of IARRY1. MOVEWD calls system routine XMIT.

MXCLAS is a function which returns the maximum class (see ICLAS) of ITEM. Entry MNCLAS returns the minimum class of ITEM.

NAFT determines how many arcs and which ones follow a given arc in the current program graph.

NBEF determines how many arcs and which ones precede a given arc in the current program graph.

NEGPRD takes the contents of a predicate in word-pair array TEMP and negates it logically. NEGPRD calls NEGREL which negates each logical operator.

NEGREL substitutes the negative relation for a given logical or relational operator.

NEWFIR generates a new statement for the restructured module: NEXIT = j, where j is integer representing an appropriate exit value.

NEWSND generates a new statement for the restructured module: NEXIT = 0, used to guide the control flow.

NEXIT is a function which adds 'NEXIT' to the token table, if not already present, and returns the token table pointer to this value.

NFRGSZ is a Library Manager function which returns the working storage fragment size.

NMBARG is a function which returns the number of arguments in LIST if all parentheses balance. Otherwise, NMBARG points to the first balancing right parenthesis in LIST.

NODPOP pops the top element off stack NDSTAK and puts it into output argument NODE. If the stack is empty, NODE is returned as 0.

NODSTK puts an element, NODE, on the top of stack NDSTAK if NODE is nowhere in the stack. If NODE is already in the stack, control returns to the calling routine. If the stack overflows, EROR is called with error number 8012.

NOTRCH determines which statements in a module are structurally reachable and writes a message to LOUT indicating that all statements can be reached or which statements cannot be reached. NOTRCH uses subroutine DPSTMT to get the statements on each DD-path and utilizes bit vectors to account for reachable statements.

NPAR computes the list of brothers (parallel arcs) to a given arc in the current program graph.

NPRED returns a list of 5 words: (NEXIT = n) where n is the specified number of DD-paths.

NPTH determines the next path in the subset of the current program graph which is being analyzed. In order to efficiently perform a path analysis, FNDCON implicitly implements a stack for remembering which paths have been taken and finds the next arc in a path.

NULABL is a function which returns a FORTRAN label. The first label is 8999; each call to NULABL produces a label which is decremented by one from the last call.

NUMEXS is a function which returns the number of executable statements in a module.

NUMODS is a function which returns the number of modules active on the library.

NUNAME is a function which returns a value, starting at 9999 and decremented by one with each call, which is not already in the STB for the given module.

NUSTMT is used to write a new statement for the restructured module onto LOUT. The statement type and statement length L1 is input. The new statement list must be in array LIST1 (in common MTHST1) prior to calling NUSTMT. The new statement is added to the SB. For certain statement types, enclosing parentheses in the LIST1 expression are stripped off before generating the new statement in LIST.

NXTEX is a function which returns the number of the next executable statement following ISTMT.

NXTSTM determines the next statement on the DD-path which is being found, or indicates that the end of the DD-path has been found.

OVRFLO is called when a stack has been overflowed. OVRFLO calls EROR with error number 5400 and sets FLAG to TRUE.

PARALE adds IF THEN, ELSE, or END IF statements to the DMATRAN program being written. The predicate for IF THEN statements is found by calling FNDPRD. When possible, unnecessary statements are removed from the program being written.

PARASA is the ANSI X3.9 FORTRAN recognizer. PARASA is called with a module, statement number, and keyword. Depending upon the keyword, PARASA determines the statement type, number of entries (if a subprogram), the symbol use class, statement type code (executable, non-executable, comment, etc.), label (if any), intra-statement structure pointer and stores the statement in the SB.

PARPDP performs the parsing for READ and WRITE conditional statements, DEFINE FILE, IMPLICIT, BYTE, INTEGER, LOGICAL, and REAL statements.

PARRUN recognizes special FORTRAN statements, such as the hardware IF statements, SENSE LIGHT and SENSE SWITCH declarations. The statement type is stored in the SDB and the statement list in the SB.

PARUNV recognizes other special FORTRAN statements, such as DEFINE, ABNORMAL, INCLUDE, IF-PRINT. The statement type is stored in the STB and the statement list in the SB.

PCHMOD causes the statements in a module to be written onto LPUNCH in FORTRAN card image format. PCHMOD calls PCHMTH.

PCHMTH is called by PCHMOD with a specified module and statement number. PCHMOD writes the statement onto LPUNCH in standard FORTRAN card image form (unindented).

PDPFT1 checks a given statement for READ/WRITE or IF-READ/WRITE conditional statement type. If any of these statement types are found, IACT is set to 1, the labels within the statement text are stored, and the next executable statement is stored in the statement vector, STMVEC.

PDPFT2 merely sets a flag, IACT, to 1 if statement type NAME is a READ/WRITE conditional or IF-READ/WRITE conditional. This terminates the DD-path being found.

POPDDP removes the initial statements for the next DD-path from the DD-path stack.

POSTAK puts a symbol, ISTMT, of two words and a priority value on the top of ISTACK. If ISTACK has overflowed (its pointer is less than or equal to 0), then OVRFLO is called.

PRDASA is called with the module and statement numbers, statement type, and DD-path predicate index as input parameters. From this information, PRDASA constructs the restructured predicate statements and stores it in array LWORK.

PRDBG is a debugging printout routine. It writes the values of its six input arguments onto file LDBG in the format 5X, A10, 5I10.

PRDDP writes the DD-path table for a given module onto file LOUT.

PRDIFT checks for IFTRAN-IF, WHILE or ORIF statements and negates the expression associated with the second predicate index.

PRDPOP pops a 5-word predicate off SSTACK and returns it in array NBLOK.

PRDSPC is called upon detection of hardware IF statements. PRDSPC generates calls to FORTRAN library routines and replaces the hardware IF statement in the module's text with an appropriate call. The following replacements are made:

Old Statement	New Statement
IF ACCUMULATOR OVERFLOW N1,N2	CALL OVERFL(J)
IF QUOTIENT OVERFLOW N1,N2	CALL OVERFL(J)
IF DIVIDE CHECK N1,N2	CALL DVCHK(J)
IF (SENSE LIGHT I) N1,N2	CALL SLITET(I,J)
IF (SENSE SWITCH I) N1,N2	CALL SSWTCH(I,J)

where J is a new variable name generated by NUNAME. When PRDSPC returns to calling subroutine PRDVAL, the statement "IF (J .EQ. 1) THEN" is generated as the next source text statement.

PRDSTK puts a 5-word predicate (in MBLOK1, MBLOK2, MBLOK3, MBLOK4, MBLOK5 input arguments) onto stack SSTACK. If stack pointer IPTR is less than one, EROR is called with error number 8018.

PRDVAL is the driver routine to evaluate predicates. PRDVAL stores a left parenthesis in LWORK(1,1), computes the predicate expression by calling PRDASA, PRDIFT, PRDSPC, and stores the new expression in LWORK(1,2) to LWORK(1,LW-1), then stores a right parenthesis in LWORK(1,LW).

PRLBT is a print procedure which outputs the contents of the library headers for the read-only and read/write libraries.

PRMDB is a print procedure which outputs the partial contents of the Module Descriptor Blocks known in the system.

PRMGXV writes the error, warning and fatal error numbers and a message onto file LUNIT.

PRMODL writes an indented listing of a given module onto file LOUT. Statement numbers are listed at the far left of the printout. Indentation of statement is based upon control flow nesting. Each indentation increment, INDSTP, is 4 columns.

PROC controls the internal subroutine stack and generates BLOCK and END BLOCK statements at the proper times.

PROG copies a module from the library back onto the library under a new module number. The new module is then modified during the restructuring process. Interspersed format statements, if any, from the old module are collected and written at the end of the new module. The MDB, SDB, and SB (via STMTOU) are created for the new module.

PRSDB writes the SDB (Statement Descriptor Blocks) for a given module onto file LOUT.

PRSTB writes the STB (Symbol Table Blocks) for a given module onto file LOUT.

PRSTMT writes the Ith statement of a given module onto file LUN, which is an input parameter. The indentation level, INDENT, is also an input parameter.

PRTEMP is used for debugging purposes. PRTEMP writes the contents of a specified temporary table onto file LOUT.

PRTOKS writes the contents of the token table onto file LOUT. One token table is created for the entire library, thus there is no module specification.

PSTPOP places the top element of the stack ISTACK onto the output string IPOST. If the stack has underflowed FLAG is set to TRUE and control returns to the calling module.

PT implements the stack for walking the SESE tree and finding DD-paths which affect predicates to be formed.

PUTBIT puts IBIT into the Ith position of IARRAY using subroutine PACX.

PUTBLK is an Access Interface procedure for storing a block of information into a given module table. After checking to see the module is active for the table and indicating the table is going to be changed, another library fragment is appended to the table if necessary. Then the proper fragment is retrieved into the working storage and the given block is stored into it.

PUTFOR writes a statement into the output file.

PUTLST is an Access Interface procedure for storing a variable length block of information into a given module table. The procedure checks if this is a new block being put into the table or an update of an old block. If it is a new block, it sets the table pointers in the auxiliary table. It makes the module active for the table and indicates that the table is being changed. If necessary, new fragments are appended to the table to store a new list. The proper fragment is brought into the working storage and the list is stored into it. Successive fragments are brought in if the list spans multiple fragments.

PUTWRD is an Access Interface procedure for storing an item into a given block of a given module table. It insures the module is active for the table and indicates the table is being changed. It also appends a new library fragment to the table if necessary. The proper fragment is brought into the working storage and the item is stored into the proper block.

REMBRO removes brothers (parallel DD-paths) from the DDPATH table as it produces a reduced form of the program graph.

REMENT removes out-of-date entries from the current program graph representation.

REORDR reorders the current program graph representation so that it is ordered by the first element in each entry.

REPLC1 replaces all occurrences of A1 character ITHIS with A1 character ITHAT in LIST of length 1, where the LIST array contains only one character in the leftmost position of each word. Entry REPLC10 performs the same replacement in LIST, where LIST is a packed character string array. REPLC10 uses subroutines JGET and JPUT to search and replace the characters within each LIST word.

SBLKSZ is a Starter Initialization procedure. It calculates the number of blocks that will fit into one fragment for each table. It also sets up some access interface pointers for library tables. It returns the number of modules that can be active at any one time (one fragment's worth).

SEME identifies SEME structures in the current program graph and replaces them with a structurally similar construct which has only one exit vertex. This is done by choosing an exit vertex and redefining all exits to go through this vertex in a consistent manner.

SEMEX forms the proper predicates for a structurized multiple-exit loop and adds the DO WHILE and END WHILE statements to the DMATRAN program being written.

SESPOP pops 5 words off of SSTACK and stores them in array NBLOK. SESPOP is used in processing single-entry/single-exit blocks of code. If the stack is empty, LFLAG is set to FALSE.

SESSTK puts 5 words (input parameters MBLOK1, MBLOK2, MBLOK3, MBLOK4, MBLOK5) onto stack SSTACK. If the stack overflows, EROR is called with error number 8016.

SHRINK controls the generation of the SESE tree. Various reduction algorithms are applied to the current program graph, producing a new program graph and adding an entry to the DD-path table for each reduction performed. SHRINK terminates when the current program graph consists of only one entry, or when no more reductions can be performed.

SLABEL is called after a module has been restructured. SLABEL replaces the statement labels in any READ/WRITE conditional statement with new unique labels which correspond with the restructured code.

SLEDIT (also entry SL3DIT) edits a LIST array, removing words which have the value of 0 and changing the sign of words which are negative. If input parameter NAR equals 3, then the original and edited LIST are the same. If NAR equals 4 or more, the edited list is stored in array LSTOUT.

SLFLOO writes a WHILE or END WHILE statement in the structurized module, depending upon whether it is the beginning or ending of a self-loop.

SPRYWD consecutively fills an array, IARRAY, starting at position IPOS with a total of NWDS occurrences of the constant IVALUE.

STCK puts the value ISTMT on the top of stack ISTCK. If the stack overflows, EROR is called with error number 8026.

STMOUX stores N number of statement numbers into array STMVEC. The statement numbers are obtained from calls to subroutine LABSTM and are, therefore, statement numbers which correspond to a label.

STMTOU takes statement number ISTMT from the old (unstructured) module and stores the text into the current statement number pointer in the restructured module. The statement number pointer and cumulative length of SB for the new module are updated.

STRTER is the Starter Initialization driver. It calls procedures that initialize system storage and open the random libraries that are to be used during the STRUCTRAN-2 execution.

STRUCT controls the procedure of walking the SESE tree and producing the structurized program during the tree walk. INITT and TT implement stacks for a tree traversal, while the other routines referenced produce structured statements according to the type of reductions in the SESE tree.

STRUC1 - STRUC2 are driver routines for each overlay link.

STTBLK constructs the target pointers to the end of IF, ORIF, ELSE, WHILE blocks and DO targets for FORTRAN programs. The target statement number is stored in the intra-statement pointer word of the SDB. In FORTRAN programs, if the DO target is not a CONTINUE statement, WARN is called with warning number 5101.

SUBINT is used to initialize the SBSTAK pointer to 0 and to set the stack size to 50.

SUBPOP pops the top element off stack SBSTAK and stores it into output parameter IDDP. If the stack is empty, EROR is called with error number 8007.

SUBSCH is the routine which adds all SESE reductions to the DDPATH table as the SESE is being built. It is given the type of reduction and the entries being reduced (in LIST) and returns the entry (IENTRY) in the DDPATH table at which this information was stored.

SUBSTK puts input parameter IDDP onto the top of stack SBSTAK. EROR is called with error number 8006 if the stack overflows.

SYSADD is a Library Manager procedure which sets the number of system tables to those defined previously during execution.

TBINIT is a Library Manager procedure which initializes some system table storage and defines the system data base tables.

TOKADD is an integer function which adds the character string CHRLST of character length CHARS to the token table, if the string is not already present in the table. The token pointer is updated if an insertion is made.

TOKCMP is an integer function which returns the token pointer which corresponds to the input string CHRLST. TOKCMP equals 0 if the string is not in the token table.

TOKNIT is called once, and it initializes a set of token numbers to refer to a set of character strings. TOKNIT calls TOKSET which puts the character strings in the token table and returns an integer pointer. The initialized variables in TOKNIT are set to the token pointers.

TOKSET is an integer function which returns a token pointer value to the input character string, after adding the string to the token table.

TRACE is a dummy routine which returns upon entry.

TT controls the stack which is used in walking the SESE tree. The type of reduction represented in the SESE tree is used to determine the invocation which is placed on the stack.

UPDEPT is a Library Manager procedure which updates a module's entry in a Library Entry Point Table. If the module has been moved from the read-only library, necessary information is retrieved from the library's EPT. Otherwise the information is obtained from the module's MDB.

UNPX is a bit string manipulation routine. The bits starting at location II and ending at JJ of KERNL are moved to ISTRING, starting at location 1.

WARN increments the warning counter by one and call PRMGXV which prints the warning message and number. If the number of warnings exceeds NWARER (set in MAINCM), then EROR is called with error number 5000.

WSGFRG is a Library Manager procedure which reads a given library fragment into a given working storage fragment.

WSGWRD is a Library Manager procedure which reads a given number of words from a given library address to a given working storage address.

WSPFRG is a Library Manager procedure which writes a given working storage fragment into a given library fragment.

WSPWRD is a Library Manager procedure which writes a given number of working storage words onto a given library starting at a given address.

XMIT transmits the first I1 words of array I2 into the first I1 words of array I3. If I1 is negative, XMIT transmits the first word of I2 into the first +I1 words of array I3.