RETRAN — A Program for One-Dimensional Transient Thermal-Hydraulic Analysis of Complex Fluid Flow Systems

CCM-5
Volume 2: Programmer's Manual
Research Projects 342 and 889

Computer Code Manual, December 1978

Prepared by

ENERGY INCORPORATED
330 Shoup Avenue
Idaho Falis, Idaho 83401

Principal Investigators
C. E. Peterson
M. P. Paulsen
J. A. McClure
D. S. Fjeld
K. D. Richert

Prepared for

Electric Power Research Institute
3412 Hillview Avenue
Palo Alto, California 94304

EPRI Project Manager
L. J. Agee
Nuclear Power Division

LEGAL NOTICE

This report was prepared by Energy Incorporated (EI) as an account of work sponsored by the Electric Power Research Institute, Inc. (EPRI). Neither EPRI, members of EPRI, EI, or any person acting on behalf of either: (a) makes any warranty or representation, express or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infri[illegible] privately owned rights; or (b) assumes any liabilities with respect to the u[illegible] of, or for damages resulting from the use of, any information, apparatus method or process disclosed in this report.

ABSTRACT

RETRAN represents a new computer code approach for analyzing the thermal-hydraulic response of Nuclear Steam Supply Systems (NSSS) to hypothetical Loss of Coolant Accidents (LOCA) and Operational Transients. In contrast to the "conservative" approach, RETRAN provides "best estimate" solutions to hypothetical LOCAs and Operational Transients. RETRAN is a computer code package developed from the RELAP series of codes, from reference data, and from extensive analytical and experimental work previously conducted relative to the thermal-hydraulic behavior of light-water reactor systems subjected to postulated accidents and operational transient conditions. The RETRAN computer code is constructed in a semimodular and dynamic dimensioned form where additions to the code can be easily carried out as new and improved models are developed. This report (the second of a four volume computer code manual) describes the programming aspects of the RETRAN code. The three companion volumes describe the theory and numerical algorithms, the user input and code output, and the verification and qualification performed with RETRAN.

CONTENTS

8 010

ILLUSTRATIONS

TABLES

I. INTRODUCTION

RETRAN is a best estimate computer code used to predict the behavior of complex thermal-hydraulic systems subjected to postulated transient conditions. RETRAN was developed from the best-estimate portions of RELAP4/003 Update 85[I-1], which were extensively modified, and complemented with a large number of model improvements and model additions.

The RELAP4 series of codes and their derivative codes have historically required a large amount of main memory storage for loading and executing. The large amount of often poorly utilized main memory required for execution has resulted in two significant disadvantages.

First, turnaround time can be slow. Since RELAP type codes can monopolize large amounts of main memory for lengthy periods of time, many computer facilities are reluctant to run such codes during peak computer usage hours.

Secondly, billing algorithms are generally weighted such that memory charges are a significant fraction of the total cost for codes requiring large amounts of memory; since algorithms for I/O time charges are based on the amount of main memory used for execution, the I/O charges can be costly, especially when data tape manipulation is required.

The RETRAN Code Package contains significant improvements in efficient usage of main memory storage and data tape manipulation. The Code Package is dynamically dimensioned, so the main memory required for any given problem depends on the nodalization used for the problem and the RETRAN modeling options that are used. Additionally, the RETRAN Code Package is semi-modular, allowing data to be stored over specific blocks of unused coding and/or the release of unused main memory, which contains coding for inactive program options or excess bulk storage. The flexibility of the semi-modular feature is somewhat inhibited by deficiencies or constraints imposed by some operating systems.

The Programmer's Manual discusses the coding conventions utilized in the RETRAN Code Package as well as the techniques used for dynamic dimensioning, semimodularization, and data tape structure and manipulation. Also contained in the manual is the general program flow from the executive subprogram RMAIN to each of the program modules, program flow within each module, a description of the specific purpose of each subroutine and function and their argument lists, subcode packages utilized and computer dependencies.

II. GENERAL CODING PHILOSOPHY

Experience from previous code development efforts has shown that as the code becomes more sophisticated, execution time and machine storage requirements increase such that the desired level of sophistication cannot be utilized under any reasonable financial constraints. It has therefore been one of the key tenets of the RETRAN development to provide a code which is usable and not a financial liability, thus providing the motive for the dynamic dimensioning and semi-modularization features in RETRAN.

To improve the efficiency of RETRAN, as compared to other large computer codes, almost by definition dictates the use of computer dependent extensions to standard FORTRAN IV compilers in addition to assembly language programs. These non-standard FORTRAN or non-FORTRAN features make it possible for the programmer to take advantage of special hardware and software features designed to improve the efficiency of program execution. Use of such non-standard and assembly language subprograms can reduce the portability of code. However, judicious programming can ensure portability of the source code, particularly between IBM and CDC computers. The approach used in the RETRAN Code Package is to isolate the required non-standard FORTRAN and assembly language programs into what is referred to as an environmental library, and to use FORTRAN coding conventions in the mainline source program which are standard for the various computer manufacturers. This approach does require the development and maintenance of environmental libraries for the various computer systems available, but ensures portability of the RETRAN Code Package source code which is much more voluminous than the environmental libraries. The environmental libraries provide the capability of using FORTRAN programming compatible with compilers supplied by different computer manufacturers, yet accommodate special hardware and software which may be available on one computer and not another. Thus, the environmental library can be viewed as an extension to the standard FORTRAN IV programming language.

1.0 PROGRAMMING CONVENTIONS

This section is not designed to spell out explicit programming conventions which must be adhered to when modifying the RETRAN source code, but rather to summarize the general guidelines followed during the development of the code. Some of the guidelines may or may not seem like normal programming conventions, depending upon the reader's familiarity with the FORTRAN supported by various computer vendors. However, the guidlines have proven to be very useful while maintaining a single version of RETRAN which was operational on many computers and operating systems.

1.1 RETRAN Code Package Programming Language

The source code for the RETRAN Code Package is written completely in the ANS FORTRAN IV programming language. All machine or system dependencies which require the use of computer manufacturers' extensions to ANS FORTRAN or to assembly language programs are isolated from the RETRAN source code via use of a machine or system-dependent environmental library which contains the non-standard coding. By isolating the computer dependencies in the environmental library, it is possible to generate a source code that may be used to install the RETRAN Code Package on a variety of machines with a minimal effort, while retaining the ability to take advantage of system dependent features designed to increase the overall efficiency of the code. Another advantage to using the scheme described above is that it is much easier to ensure that the actual coding, when installed on different machines, is in as close agreement as practicable within the limitations of different system architectures and operating philosophies.

1.1.1 CDC FORTRAN Compiler

The FORTRAN Extended 4.4 compiler has been used to translate the RETRAN Code Package source code for use on CYBER 70, CYBER 170, and 6000 Series Computer Systems[II.1-1] No problems have been encountered when using the default optimization level (OPT=1). Only minor difficulties have been encountered when using optimization level two (OPT=2). In some instances, the larger subroutines such as EDATA1, EDATA2, EDATA3, EDATA4, or DNBM will not compile at OPT=2, as the result of an error in the optimizer logic. Since an approximate five percent decrease in running time can be achieved through use of the level two optimization, the source code is generally compiled at OPT=2, the object decks saved

(not using the A option of FTN), and the subroutines which contained compilation errors are then recompiled at OPT=1 and placed in the previous object deck file via the COPYL utility program. The use of additional options which may be set on this FTN control card should be determined according to the programmer's needs. A field length of 150K is generally adequate to compile any subroutine contained in the RETRAN Code Package.

1.1.2 IBM FORTRAN Compiler

Compilation of the RETRAN Code Package source code for use on an IBM 360/3x computer is facilitated by use of the FORTRAN IV H-Extended Compiler[II.1-2]. Use of the 2.2 level H-Extended compilers (PGM=IFEAAB) on the RETRAN source code has demonstrated several compiler errors which are catastrophic to compilation and/or actual execution of RETRAN. Consequently, use of the 2.2 level H-Extended compiler should be considered only if the following IBM-supplied updates have been incorporated.

P49288	P54297
P50693	P56626
P52463	P56633
P52465	P56650
P52485	P58582
P54293	P58585
P54295	P60353

The optimization used during compilation is generally specified as OPT(2). Other options generally specified in the PARM field of the EXEC statement include NOALC or no automatic alignment; GOSTMT which includes internal statement numbers used by the traceback feature; and AUTODBL(12220) which activates the API (Automatic Precision Increase) facility. All other options are set according to the programmer's needs. A region of 300K is generally adequate to compile any subroutine in the RETRAN Code Package.

1.2 RETRAN Environmental Library

The environmental library consists of a group of subcodes designed to perform specific tasks such as:

- Free form input processing - INP
- Dynamic dimensioning - FTB
- Water state properties - STH20
- Multiple curve plotting - PLOTMC
- Buffer I/O - BUFOUT

in addition to miscellaneous subroutines which invert matrices, form masks, and perform Boolean arithmetic. A detailed discussion of the RETRAN environmental library can be found in Section VII.

Unlike the source code for the RETRAN Code Package, several versions of the environmental library are required. Each of the libraries is tailored to the particular computer and operating system for which it is designed.

Since some features designed to perform a specific function on a given computer may not be available on another computer, some trivial subprograms are required. As an example, dynamic field length or region adjustment is a very useful feature. However, only CDC computers allow for dynamic field length adjustment during a given job step. Consequently, the CDC versions of REDUCE and EXPAND used, respectively, to release main memory or to obtain more main memory contain coding which exercises the memory macro and edits region and storage information, while the IBM version only edits storage information based upon a fixed region size specified at the beginning of the job step.

The environmental libraries are rather small compared with the RETRAN source code. Additionally, once they have been checked out they should require little, if any, maintenance unless new features are added or existing features modified.

1.2.1 CDC Environmental Library Programming Languages

The environmental library used for the CYBER 70, CYBER 170, and 6000 Series Computer Systems is written in FORTRAN and COMPASS. All COMPASS subprograms are coded so that they may be batched to the FTN 4.4 compiler with FORTRAN subprograms. The FTN 4.4 compiler will direct an intermixed COMPASS input stream to the COMPASS 3.3 assembler provided an IDENT card is placed as the first card of a COMPASS deck. An END card must be the last card of a deck, and both the IDENT and END must begin in column 11 of the card images.

Several of the COMPASS programs require a system text file to provide the assembler with the macro expansions. The required system text library has been named CPUTEXT at all installations encountered at the time of writing this document, but it is possible that the required library name could be different for some facilities. In any event, the system text file name should be supplied on the FTN control card by use of the system text parameter S, e.g., S=CPUTEXT, for intermixed FORTRAN and COMPASS decks. The COMPASS 3.3 assembler may be used to assemble COMPASS decks; however, it is generally much more convenient to use the FTN 4.4 compiler.

The FORTRAN subprograms are compiled using the OPT=2 optimization feature. No difficulties have been encountered while using the OPT=2 optimization level for the environmental library.

The object decks or relocatable binary records generated by the compiler are processed through the EDITLIB utility to construct a user library[II.1-3]. That library is then made available to the segmented loader by specific direction in the loader control statements for a job loading the RETRAN Code Package. Use of the environmental library in the user library format has significant advantages due to the method in which object decks are loaded and external references are satisfied. For the specifics refer to the LOADER or CYBER LOADER reference manuals[II.1-4, II.1-5].

1.2.2 IBM Environmental Library Programming Languages

The environmental library used for the IBM System/360 or System/370 computers is written in both FORTRAN and assembly language. Capabilities do not exist in the current IBM Program Product line to intermix FORTRAN and assembly language source text to a single program containing multiple language translating features.

Consequently, FORTRAN source code must be translated via the FORTRAN compiler and assembly language source code via the assembler, both of which may be executed in the batch mode.

The FORTRAN source code is compiled using the FORTRAN H-Extended compiler and options described in Section II.1.1.2, with the exception of the Automatic Precision Increase feature, which must be set to AUTODBL(NONE). Assembly of the

assembly language source code is facilitated via the level F or H assemblers, PGM=IEV90 or PGM=IEUASM, respectively and the SYS1.MACLIB system text data set (specified on SYSLIB DD card).

Each of the environmental library subprograms is made a member of a partitioned data set by use of the linkage editor PGM=IEWL[II.1-6]. All entry points are defined as aliases during the link edit step. The library data set is concatenated with the SYS1.FORTXLIB data set on the SYSLIB DD card during the link edit step used to generate a RETRAN Code Package load module. External references are then satisfied from the system FORTRAN library and the RETRAN environmental library.

1.3 Internal Documentation

Documentation for the RETRAN Code Package is included in "RETRAN - A Program for One-Dimensional Transient Thermal-Hydraulic Analysis of Complex Fluid Flow Systems, Volume I: Equations and Numerics; Volume II: Programmer's Manual; Volume III: User's Manual; and Volume IV: Applications Manual." Another level of documentation is also readily available. This additional documentation consists of a listing of the FORTRAN source coding, which internally provides basic code documentation via the liberal use of comment statements. Comments are inserted at the beginning of each subroutine or function defining the role each subprogram and/or entry point plays in the RETRAN Code Package. Along with each subprogram definition, the argument list is also defined, noting the specific definition of the arguments, whether they are input, output, or modified internally (both input and output). Each of the dynamically allocated storage arrays (see Section II.2) is also described by means of the RETRAN file directory which is given in Appendix A, and which is also included in the main program RMAIN.

Each well-defined task required during execution of one of the RETRAN Code Package program modules has been coded into a single concise subprogram. Within each subprogram comments have been included to aid in the understanding of the various blocks of coding. All program modules include a main or primary driver which in turn calls an input processing driver, an initialization driver if required, followed by a driver which directs the actual execution of a problem. As an example, execution from the RETRAN program module is controlled by the

driver subroutine RETRAN. Subroutine RETRAN in turn serially delegates the responsibility of input processing to the input processing driver INRTRN; generating initial conditions or initial values to the driver STSTAT; and finally execution of the transient thermal hydraulics via the driver TRAN. Few, if any, calculations are performed in the driver subroutines over the minimum required to direct the program flow. The simplicity of the drivers, coupled with comments describing such redirection in program flow provides a very useful and necessary level of documentation, very similar to a general flow chart.

An over-use of comments can make the source coding difficult to follow, so one should be careful of over-using comments. On the other hand, nothing is more frustrating than trying to understand even the simplest of coding which is void of comments. A guideline followed during the development of the RETRAN Code Package is that comments should be added when they can add clarity and aid in the understanding of a block of coding, particularly for those who may not be intimately familiar with the code.

1.4 Machine Flag

Several situations were encountered where differences between the various computers could not be completely isolated in the environmental library, particularly Hollerith character manipulation. Consequently, a logical variable (IBMRUN) was added to the code, which dynamically flags whether the code is executing on an IBM or CDC computer. This is determined automatically by use of the environmental library function LOCF, which is described in detail in Section VII. Briefly, LOCF returns the core address of a variable or subroutine entry point. If the difference between the core address of two consecutive real variables is eight, then an IBM computer is being used since eight bytes are needed for each double precision variable. If the difference is one, a CDC computer is being used. This flag is then used to direct the program flow through the machine dependent coding. The machine dependent coding will be executed only for the particular machine for which it is designed, but must also be compatible with the FORTRAN compiler used on the machine for which it is not designed.

The example in Table II.1-1 is typical of the character manipulations found in the input subroutines of the RETRAN Code Package. For CDC computers (.NOT.IBMRUN),

Table II.1-1

TYPICAL EXAMPLE OF RETRAN CHARACTER MANIPULATION

```
      DATA BLANK       /8H        /
      DATA IZER / 4H0000 /, ININ / 4H9999 /
C
C
C
C     SET UP CHARACTER MANIPULATION MASKS
      NBITS = 30                                                    }
      NBYTE = 6                                                     }
      IF (.NOT.IBMRUN) GO TO 2                                      } A
      NBITS = 32                                                    }
      NBYTE = 8                                                     }
    2 BMASK = MASKF (NBITS)                                         }
      EMASK = MASKF (-NBITS)                                        } B
      LEN = -NBITS                                                  }
      IF (.NOT.IBMRUN) LEN = LFN - NBYTE                            }
      CMASK = MASKF (LEN)                                           }
      AMASK = DAND (BLANK,CMASK)                                    } C
      LEN = 4*NBYTE                                                 }
      AMASK = DSL (AMASK,LEN)                                       }
      AMASKN = MASKF (-NBYTE)                                       }
      IF (IBMRUN) AMASKN = DSL (AMASKN,NBITS)                       }
      IF (.NOT.IBMRUN) IZER = ISL (IZER,NBYTE)                      }
      IF (.NOT.IBMRUN) ININ = ISL (ININ,NBYTE)                      } D
      IZER = IAND (IZER,AMASKN)                                     }
      ININ = IAND (ININ,AMASKN)                                     }
      ALOCAL = DAND (RSTOR(IDXVAR,BMASK)                            }
      BLOCAL = RSTOR(IDXVAR+1)                                      }
      IF (.NOT.IBMRUN) BLOCAL = DSL (BLOCAL,NBITS)                  } E
      BLOCAL = DAND (BLOCAL,BMASK)                                  }
      RSTOR(LOC) = DOR (ALOCAL,DSR(BLOCAL,NBITS))                   }
```

1758 025

the word length is 60 bits for real, integer, and logical word types, while on IBM computers real type variables are 64 bits, and 32 bits for logical and integer type variables. Integers and logical type variables are equivalenced to real variables (for use by the FTB subcode), such that the integers and logicals share the first 32 bits of the corresponding real type words for problems run on IBM computers. Experience has shown that very little additional memory or execution time overhead is required for this technique. The following descriptions refer to Table II.1-1.

A: The number of bits per Hollerith character and the number of bits per halfword are defined for either the CDC or IBM computer on which a problem is being run. For this discussion consider a full word to be 64 bits on IBM computers, since this is in effect the method used by FTB, i.e., arrays RSTOR and ISTOR are equivalenced such that RSTOR(K) has the same address as ISTOR(1,k) as opposed to ISTOR(K) for CDC computers.

B: Turn on all bits in the top half of BMASK and zero all bits in the bottom half. Do the opposite for EMASK. All masks are constructed by use of the Environmental Library function MASKF, which is described in detail in Section VII.

C: Form AMASK such that the bottom 4 Hollerith characters are binary zeros and the remaining 4 or 6 characters are Hollerith blanks for IBM or CDC, respectively.

D: Form two integer-type masks (60 bit CDC, 32 bit on IBM) where the bottom characters are either a Hollerith zero or nine and the remaining upper portion of the word is binary zero filled.

E. Take the Hollerith field from the top half of RSTOR(IDXVAR) and merge it with the integer in RSTOR(IDXVAR+1) (actually ISTOR (1,IDXVAR+1) or ISTOR(IDXVAR+1)), and merge them into a full word such that the Hollerith portion resides in the top and the integer in the bottom.

There may be cases encountered in the future where it is required to know whether the code is being executed on an IBM or CDC computer. If so, the logical flag IBMRUN, which is set in the main program RMAIN, is readily available to any subprogram since it resides in the FTB labeled common block.

1.5 Programming Suggestions

During the development of the RETRAN Code Package, several FORTRAN programming guidelines were established to maintain portability of the source code between CDC and IBM machines. Several of the guidelines are normal elements of good FORTRAN programming while others tend to eliminate problems resulting from idiosyncrasies of the various versions of FORTRAN supported by the computer manufacturers. The guidelines are summarized below.

1) Include all separators between fields in a format statement,

```
100 FORMAT (I6,A8)
```

rather than

```
100 FORMAT (I6A8).
```

2) Avoid use of PRINT and NAMELIST I/O statements. They generally require system routines not loaded into the programs in order to minimize core requirements. Use of PRINT and NAMELIST statements is generally catastrophic. The convention is to use formatted WRITE statements to FORTRAN Unit 6 (file OUTPUT for CDC or SYSOUT=A for IBM).

3) FORTRAN binary READ-WRITE operations are also avoided for reasons much the same as given above. Additionally, the environmental library contains a FORTRAN callable binary read-write subroutine which has been tailor-made for use in the RETRAN Code Package.

4) Avoid use of a function in an I/O list:

```
IT=IABS(IK)
WRITE (6,100) IT,HOLER
```

rather than

```
WRITE (6,100) IABS(IK),HOLER
```

5) Variables used to store Hollerith fields greater than 4 characters in length should be typed REAL (elevated to 8 byte words for IBM).

6) Assume that the output mode of a variable in an I/O list is determined by the variable type, rather than format field.

7) Avoid use of Hollerith replacement such as

```
NONO=8HNOWORKY?
```

A more universally acceptable approach is to limit use of Hollerith literals to DATA and FORMAT statements.

8) Limit the magnitude of large literal floating point constants to exponents of 75 and small literal floating point numbers to exponents of -75.

This will help prevent unnecessary overflow and underflow problems.

9) Avoid the use of actual numerical values (or literal values) as passed subroutine or function arguments. It is better to pass a variable of the proper type, which has been initialized using a data statement or literal replacement.

10) ANS FORTRAN intrinsic functions used to force mode conversion are not universally acceptable to all FORTRAN IV compilers, and some are actually remnants of earlier levels of the FORTRAN language. Mode conversion is automatic during a replacement, or by merely equating two variables of different type.

1.6 FORTRAN Library Intrinsic and External Functions

A summary of the basic intrinsic functions and external functions satisfied from the FORTRAN library of a given system at load time are given in Tables II.1-2 and II.1-3, respectively. The tables include only the functions used in the RETRAN Code Package source program, and do not include extensions such as shifting, masking, and Boolean arithmetic which is required by the environmental library.

TABLE II.1-2

BASIC INTRINSIC FUNCTIONS[1]

Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Absolute Value	|A|	1	ABS	Real	Real
			IABS	Integer	Integer
			DABS	Double	Double
Truncation	Sign of A times largest integer $\leq$ |A|	1	AINT	Real	Real
			INT	Real	Integer
			IDINT	Double	Integer
Remaindering	A1-[A1/A2] A2 Brackets indicate largest integer $\leq$A1/A2	2	AMOD	Real	Real
			MOD	Integer	Integer
			DMOD[2]	Double	Double
Choosing largest value	Max(A1, A2,...)	$\geq$2	AMAX0	Integer	Real
			AMAX1	Real	Real
			MAX0	Integer	Integer
			MAX1	Real	Integer
			DMAX1	Double	Double
Choosing smallest	Min(A1, A2,...)	$\geq$2	AMIN0	Integer	Real
			AMIN1	Real	Real
			MIN0	Integer	Integer
			MIN1	Real	Integer
			DMIN1	Double	Double
Float	Conversion from integer to floating point	1	FLOAT	Integer	Real
			DFLOAT[2]	Integer	Double

TABLE II.1-2 (Cont'd)

Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Fix	Conversion from real to integer Same as INT	1	IFIX[3]	Real	Integer
Transfer of sign	Sign of A2 with A1	2	SIGN	Real	Real
			ISIGN	Integer	Integer
			DSIGN	Double	Double

1. Functions with double precision arguments or results are used for IBM versions not using the Automatic Precision Increase (API) compiler option. For CDC versions or IBM versions using API, only single precision functions are used. For IBM applications, Integer implies INTEGER*4, Real implies REAL*4, and Double implies REAL*8.

2. Not ANS FORTRAN

3. Not suggested for use since there is not a double precision form. IBM H-extended with API option turned on may accept. However, with FORTRAN IV, a function is not required for mode conversion. INTEG = FLOATP is preferable since mode conversion automatically occurs when FLOATP is typed Real or Double and INTEG is typed Integer.

1758 030

TABLE II.1-?

BASIC EXTERNAL FUNCTIONS[1]

Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Exponential	e^A	1	EXP	Real	Real
		1	DEXP	Double	Double
Natural	loge(A)	1	ALOG	Real	Real
Logarithm	A>0	1	DLOG	Double	Double
Common	log10(A)	1	ALOG10	Real	Real
Logarithm	A>0		DLOG10	Double	Double
Trigonometric	sin(A)	1	SIN	Real	Real
Sine		1	DSIN	Double	Double
Trigonometric	cos(A)	1	COS	Real	Real
Cosine		1	DCOS	Double	Double
Trigonometric	tan(A)	1	TAN	Real	Real
Tangent		1	DTAN	Double	Double
Square	$(A)^{1/2}$	1	SQRT	Real	Real
Root	$A \geq 0$	1	DSQRT	Double	Double
Arctangent	arctan (A)	1	ATAN	Real	Real
		1	DATAN	Double	Double
	arctan (A1/A2)	2	ATAN2	Real	Real
	$A1^2+A2^2 \neq 0$	2	DATAN2	Double	Double
Arccosine	arccos (A)	1	ACOS[2]	Real	Real
	$A \leq 1$	1	DARCOS	Double	Double
Arcsine	arcsin (A)	1	ASIN[3]	Real	Real
	$A \leq 1$	1	DARSIN	Double	Double

1. Functions with double precision arguments or results are used for IBM versions not using the Automatic Precision Increase (API) compiler option. For CDC versions or IBM versions using the API option, single precision functions are used. For IBM applications, Integer implies INTEGER*4, Real implies REAL*4, and Double implies REAL*8.

2 Alias for ARCOS on IBM when using H-extended compiler.

3 Alias for ARSIN on IBM when using H-extended compiler.

1758 031

In general, only the INTEGER and REAL type functions are of interest since the CDC code operates as single precision and the AUTODBL feature of the IBM H-extended compiler will elevate the single precision REAL*4 coding (default) to REAL*8 and also convert single precision functions to their double precision equivalent. If the AUTODBL feature is not used on IBM versions, a text editor must be used to convert single precision functions to their double precision equivalents.

2.0 DYNAMIC STORAGE ALLOCATION TECHNIQUE

RETRAN, from a dynamic dimensioning viewpoint, is a code in which the core storage requirements depend strictly on the size of the problem and the options that are used. The concept of dynamic dimensioning is such that the storage for variables describing the problem is allocated as the input data is read. The amount of storage allocated for data depends strictly on the size of the problem and the options that are used.

Several techniques were examined which afforded dynamic dimensioning capabilities, with varying degrees of flexibility. The FTB Dynamic Core Allocation and Scratch I/O Package was selected as a means of implementing dynamic dimensioning because of the flexibility and control of system resources the package allows the programmer to exercise. The FTB package allows execution time allocation of main memory, bulk memory and mass storage as files which may be used for the duration of the program, and expanded or reduced as required. Temporary storage can be deleted at will, thus freeing resources which may be re-assigned as needed. With the FTB package and the input data a user normally supplies to describe system nodalization, storage requirements may be computed and assigned dynamically, thus minimizing resource requirements (main memory in particular). Additionally, any data file may be easily accessed by any subprogram in the RETRAN code package. A description of the FTB package is included in Section VII of this manual.

2.1 Description of File Structures

Basic to RETRAN is what will be designated from this point on as "files". A file is defined as a characteristic set of variables describing one particular feature of the problem such as volumes, junctions, pumps, heat conductors, etc.

As an example of the file structure, the file (abbreviated) containing variables describing a RETRAN control volume is shown in Table II.2-1. In this example, there are twenty-five variables used for the description of one volume.

The data required to completely describe a control volume is referred to as a set, where the control volume file is comprised of an individual set for each volume. A fixed amount of data is often required to describe a particular

TABLE II.2-1

TYPICAL FILE SET STRUCTURE

WORD	VARIABLE	
(01) =	IBUB	= BUBBLE DATA INDEX
(02) =	IREAD	= VOLUME DATA RETRIEVAL INDEX
(03) =	P	= VOLUME PRESSURE
(04) =	TEMP	= VOLUME TEMPERATURE
(05) =	HORX	= VOLUME QUALITY
(06) =	V	= VOLUME (FT3)
(07) =	ZVOL	= VOLUME HEIGHT
(08) =	ZM	= MIXTURE LEVEL
(09) =	JTPMV	= TWO PHASE FRICTION INDEX
(10) =	FLOWA	= VOLUME FLOW AREA
(11) =	DIAMV	= EQUIVALENT DIAMETER OF FLOW AREA
(12) =	ELEV	= ELEVATION AT THE BOTTOM OF THE VOLUME
(13) =	SATP	= SATURATION PRESSURE
(14) =	SATT	= SATURATION TEMPERATURE
(15) =	SATVF	= SATURATION SPECIFIC VOLUME OF LIQUID
(16) =	SATVG	= SATURATION SPECIFIC VOLUME OF GAS
(17) =	SATUF	= SATURATION LIQUID SPECIFIC INTERNAL ENERGY
(18) =	SATUG	= SATURATED GAS SPECIFIC INTERNAL ENERGY
(19) =	SATHF	= SPECIFIC ENTHALPY OF SATURATED LIQUID
(20) =	SATHG	= SPECIFIC ENTHALPY OF SATURATED GAS
(21) =	VL	= SPECIFIC LIQUID VOLUME
(22) =	VS	= SPECIFIC GAS VOLUME
(23) =	UW	= INTERNAL SPECIFIC ENERGY
(24) =	HW	= VOLUME SPECIFIC ENTHALPY
(25) =	GASH	= GAS SPECIFIC ENTHALPY

feature or component such as a volume; consequently, the set is fixed length. Files vary in length from one set to many sets, depending on the nodalization specified by the user.

For a single volume problem, storage is allocated for one set of volume variables. For a multi-volume problem, storage is allocated for NVOL sets of volume variables where NVOL is the number of volumes. Some files may have many sets of variables in a file such as the volume file, the junction file, or the heat conductor file, while others always contain only one set of data. An example which always has a single set is the data file describing the overall system parameters for the problem being modeled, or the system file.

Another variation to the file structure is "subfiles". As an example of subfiles, consider the fill option in RETRAN. The major file for the fill option is similar to the files described above, and contains a fixed length set of data describing each fill. However, each fill has a fill table of flow, enthalpy, and pressure versus time or adjacent volume pressure. These tables can vary in length. The major file and subfile are shown in Table II.2-2. In this example, word 6 (IDXFLT) of file 14 gives the index of the first word of the array containing fill table data. The fill table consists of pairs of values of time (or pressure) and flow. All file and subfile indices denote the locations of the files and subfiles in the array RSTOR in the FTB common block. Let IDX14 be the index of the fill data set (file 14). Then the N^{th} pair of values for time (or pressure) and flow can be written as TIME = RSTOR (IDXFLT (IDX14) + 2*(N-1)) and FLOW = RSTOR (IDXFST (IDX14) + 2*N - 1).

The idea of maintaining a fixed length for each set of variables residing within a major file allows easy access to the various sets of data and will be illustrated in the following sections. However, information that can be variable in length, such as data tables, are not well suited for the fixed length, set oriented file structure discussed above. Consequently, files for variable length arrays are assigned as subfiles, which are indirectly addressed via the subfile index or pointer. The index is generally saved as an element of the major file set, such as IDXFLT, IDXENT and IDXPRS in the previous example. Subfiles are used for all information that is variable length as it applies to a particular code option.

Table II.2-2

MAJOR FILE AND SUBFILE CONTENT

```
*** FILID(14) = FILLER - CONTAINS FILL DATA
*** FILE FILLER HAS NFILL DATA STES.  A DATA SET IS SHOWN BELOW
 WORD
 (01) = NFILL  = NUMBER OF PAIRS IN FILL TABLE
 (02) = ITFILL = TRIP NUMBER CONTROLLING FILL
 (03) = JX     = INDEPENDENT VARIABLE TYPE(-1=DIFF PRES,0=TIME,1=VOL PRESSURE)
 (04) = JY     = DEPENDENT VARIABLE TYPE (0=LB/SEC,1=GAL/MIN)
 (05) = IFILL  = CURRENT TABLE POSITION
 (06) = IDXFLT = INDEX OF FILE CONTAINING FILL TABLE
 (07) = PFILLO = PREVIOUS TIME STEP FILL PRESSURE
 (08) = STHIDX = MEMORY INDEX FOR FILL STATE PROPERTIES
 (09) = IDXENT = INDEX OF ENTHALPY SUBFILE
 (10) = IDXPRS = INDEX OF PRESSURE SUBFILE - DEFINED ONLY IF JX .GE. 0

*** FILID FILTBL IS A SUBFILE OF FILLER, CONTAINS FILL TABLE
 WORD
 (01)     = FILTBL(1)    = TIME OR PRESSURE
 (02)     = FILTBL(1)    = FLOW
  .
  .
  .
(2*NFILL-1) = FILTBL(NFILL) = TIME OR PRESSURE
(2*NFILL)   = FILTBL(NFILL) = FLOW

*** ENTHALPY CORRESPONDENCE SUBFILE
   (EACH ITEM CORRESPONDS TO W VS T OR P PAIR IN FILTBL)
 WORD
 (01) = FILENT(1) = FILL ENTHALPY
 (02) = FILENT(2) =
  .
  .
  .
 (NFILL) = FILENT(NFILL) = FILL ENTHALPY
```

1758 036

Table II.2-2 (Cont'd)

*** PRESSURE CORRESPONDENCE SUBFILL
(EACH ITEM CORRESPONDS TO W VS T OR P PAIR IN FILTBL)
WORD
(01) = FILPRS(1) = FILL PRESSURE
(02) = FILPRS(2) =

(NFILL) = FILPRS(NFILL) = FILL PRESSURE

The file describing the heat conductors is another example of the use of subfiles. Each heat conductor can be divided into as many temperature nodes and material regions as desired by the user. Consequently, the number of variables describing the nodalization in each heat conductor may vary. Quantities identified with heat conductor nodes and conductor regions are kept in subfiles of the major heat conductor file as shown in the file description illustrated Table II.2-3.

TABLE II.2-3

MULTIPLE SUBFILE USAGE

*** FILID(20) = SLABHT - HEAT CONDUCTOR DESCRIPTIONS AND DATA
*** FILE SLABHT HAS NSLB DATA SETS. A DATA SET IS SHOWN BELOW

WORD

(01) = IVSL = INDEX NO. OF VOLUME AT LEFT SURFACE OF COND. (NEW)
(02) = IVSR = INDEX NO. OF VOLUME AT RIGHT SURFACE OF COND. (NEW)
(03) = IGOM = GEOMETRY INDEX
(04) = ISB = STACK INDICATOR
(05) = IMCL = LEFT SURFACE INDICATOR FOR HEAT TRANS. CORRELATION
(06) = IMCR = RIGHT SURFACE INDICATOR FOR HEAT TRANS. CORRELATION
(07) = AHTL = HEAT TRANS. AREA AT LEFT CONDUCTOR SURFACE
(08) = AHTR = HEAT TRANS. AREA AT RIGHT CONDUCTOR SURFACE
(09) = VOLS = VOLUME OF HEAT CONDUCTOR
(10) = HDML = HYDRAULIC DIAMETER OF VOLUME ON LEFT OF COND.
(11) = HDMR = HYDRAULIC DIAMETER OF VOLUME ON RIGHT OF COND.
(12) = DHEL = HEATED EQUIVALENT DIAMETER ON LEFT OF COND.
(13) = DHER = HEATED EQUIVALENT DIAMETER ON RIGHT OF COND.
(14) = CHNL = CHANNEL LENGTH ON LEFT OF COND.
(15) = CHNR = CHANNEL LENGTH ON RIGHT OF COND.
(16) = SE = STORED ENERGY IN CONDUCTOR
(17) =
(18) = FCHL = CRITICAL HEAT FLUX AT LEFT COND. SURFACE
(19) = FCHR = CRITICAL HEAT FLUX AT RIGHT COND. SURFACE
(20) = HTCL = HEAT TRANSFER COEFFICIENT AT LEFT COND. SURFACE
(21) = HTCR = HEAT TRANSFER COEFFICIENT AT RIGHT COND. SURFACE
(22) = PHIL = HEAT FLUX AT LEFT COND. SURFACE
(23) = PHIR = HEAT FLUX AT RIGHT COND. SURFACE
(24) = SLEN = EQUIVALENT LENGTH TO HEAT COND.
(25) = WQCL = HEAT TRANS. RATE TO FLUID AT LEFT COND. SURFACE
(26) = WQCR = HEAT TRANS. RATE TO FLUID AT RIGHT COND. SURFACE
(27) = IBCL = LEFT BOUNDARY CONDITION INDICATOR

TABLE II.2-3 (Cont'd)

(28) = IBCR = RIGHT BOUNDARY CONDITION INDICATOR
(29) = IHTL = HEAT TRANS. MODE AT LEFT COND. SURFACE
(30) = IHTR = HEAT TRANS. MODE AT RIGHT COND. SURFACE
(31) = ISCO = CORE NUMBER
(32) = TL = LEFT SINK TEMPERATURE, F
(33) = TR = RIGHT SINK TEMPERATURE, F
(34) = AZL = AZL THROUGHT HZR ARE COEFFICIENTS FOR CHF
(35) = AX1L = CORRELATIONS FOR LEFT SIDE
(36) = BZL =
(37) = CZL =
(38) = EZL =
(39) = EZ1L =
(40) = FZL =
(41) = HZL =
(42) = WED = WETTED EQUIVALENT DIAMETER, IN.
(43) = IDXTP = INDEX OF FILE CONTAINING TP FOR EACH NODE
(44) = IDXAP = INDEX OF FILE CONTAINING AP FOR EACH NODE
(45) = IDXTPC = INDEX OF FILE CONTAINING ITPC FOR EACH REGION
(46) = IDXTPK = INDEX OF FILE CONTAINING ITPK FOR EACH REGION
(47) = IDXTPX = INDEX OF FILE CONTAINING ITPX FOR EACH REGION
(48) = OLSLBN = OLD CONDUCTOR NUMBER
(49) = IVSLOL = OLD VOLUME NUMBER (LEFT SIDE)
(50) = IVSROL = OLD VOLUME NUMBER (RIGHT SIDE)
(51) = AZR = AZR THROUGH HZR COEFFICIENTS FOR CHF CORRELATIONS
(52) = AZ1R = FOR RIGHT SIDE
(53) = BZR =
(54) = EZR =
(55) = EZ1R =
(56) = FZR =
(57) = FZR =
(58) = HZR =
(59) =
(60) =
(61) = PHOL = HEAT FLUX AT LEFT COND. SURFACE (OLD TIME STEP)

8 040

TABLE II.2-3 (Cont'd)

(62) = PHOR = HEAT FLUX AT RIGHT COND. SURFACE (OLD TIME STEP)
(63) = GL = LEFT COOLANT FLOW RATE
(64) = GR = RIGHT COOLANT FLOW RATE

*** FILE SLBTP IS A SUBFILE OF SLABHT. SLBTP CONTAINS VARIABLE TP FOR EACH NODE IN A HEAT CONDUCTOR. INDEX OF FILE SLBTP IS WORD (43) OF FILID(20)

WORD

(01) = NNODE = NUMBER OF TEMPERATURE NODES IN THE HEAT CONDUCTOR
(02) = TP = NODE 1 TEMPERATURE
(03) = TP = NODE 2 TEMPERATURE
.
.
.
(XX-1) = TP = NODE XX-1 TEMPERATURE
(XX) = TP = NODE XX TEMPERATURE

*** FILE SLABAP IS A SUBFILE OF SLABHT. SLABP CONTAINS VARIABLE AP FOR EACH NODE IN A HEAT CONDUCTOR. INDEX OF FILE SLBAP IS WORD (44) OF FILID(20)

WORD

(01) = AP = -2*SURFACE WEIGHT*THERMAL CONDUCTIVITY (FOR NODE 1)
(02) = AP = -2*SURFACE WEIGHT*THERMAL CONDUCTIVITY (FOR NODE 2)
.
.
.
(XX) = AP = -2*SURFACE WEIGHT*THERMAL CONDUCTIVITY (FOR NODE XX

*** FILE SLITPC IS A SUBFILE OF SLABHT. SLBAP CONTAINS VARIABLE ITPC FOR EACH REGION OF A HEAT CONDUCTOR. INDEX OF FILE SLITPC IS WORD(45) OF FILID(20)

1758 041

TABLE II.2-3 (Cont'd)

WORD

(01) = ITPX = CURRENT POSITION IN TPX TABLE FOR REGION 1
(02) = ITPX = CURRENT POSITION IN TPX TABLE FOR REGION 2
.
.
.
(XX) = ITPX = CURRENT POSITION IN TPX TABLE FOR REGION XX

WORD

(01) = ITPC = CURRENT POSITION IN TPC TABLE FOR REGION 1
(02) = ITPC = CURRENT POSITION IN TPC TABLE FOR REGION 2
.
.
.
(XX) = ITPC = CURRENT POSITION IN TPC TABLE FOR REGION XX

*** FILE SLITPK IS A SUBFILE OF SLABHT. SLITPK CONTAINS VARIABLE ITPK FOR EACH REGION OF A HEAT CONDUCTOR. INDEX OF FILE SLITPK IS WORD(46) OF FILID(20)

WORD

(01) = ITPK = CURRENT POSITION IN TPK TABLE FOR REGION 1
(02) = ITPK = CURRENT POSITION IN TPK TABLE FOR REGION 2
.
.
.
(XX) = ITPK = CURRENT POSITION IN TPK TABLE FOR REGION XX

*** FILE SLITPX IS A SUBFILE OF SLABHT. SLIPTX CONTAINS VARIABLE ITPX FOR EACH REGION OF A HEAT CONDUCTOR. INDEX OF FILE SLITPX IS WORD(47) OF FILID(20)

178 042

There are approximately 56 major files and many subfiles available for use in RETRAN. The files that are used and the amount of storage required is determined at execution time as the input data is processed. The philosophy of every file structure corresponds with one of the examples given previously. All of the files and their descriptions are listed in Appendix A and in subroutine RMAIN.

2.2 Accessing Files and Variables in Files

There are four variables kept to describe and access each major file and the variable sets in the file. These variables are the SETSIZ, FILSIZ, FILIDX, and FILID and reside in the FTB labeled common block. The SETSIZ is the number of variables in each set of data in a file. The FILSIZ is the total number of words in the file, i.e. the SETSIZ times the number of sets. The FILIDX is the "index" of the first word in the file. The FILIDX is the offset from the base address of the labeled common array to the first word in the file. The FILID is a floating point number, assigned as a unique identifier for the file at the time of storage allocation.

The storage allocated for files is reserved through a subroutine named RESERV from the FTB package. A call to RESERV is made for every file that is created for any particular run. The FILSIZ, SETSIZ and FILID are defined before the call to RESERV. RESERV allocates the requested amount storage and returns the FILIDX.

All major files are given an integer number or relocatable file ID for reference. For instance, the volume file is file 5 and the junction file is file 6. These numbers are shown on the file description listing in Appendix A.

2.2.1 Multiple Type Variable Common Storage

Storage assigned by the FTB subroutines resides in a common area that is defined by the FTB labeled common block. The size of the common storage area is defined by the difference between the field length or region size and the relative address corresponding to the first word of the FTB common block. Upon examination of the FTB common block definition illustrated in Table II.2-4, it should be noted very little space appears to be set up. In actuality, the FTB

Table II.2-4

ILLUSTRATION OF FTB COMMON BLOCK DEFINITION

THE FIRST 100 WORDS ARE USED AS A COMMUNICATIONS BLOCK BY THE FTB ROUTINES. THE ADDITIONAL SPACE IS USED TO STORE PROGRAM RELOCATABLE FILE LOCATION AND DESCRIPTION. ARRAY BSTOR IS USED TO HOLD THE FIRST LINK FOR THE DYNAMIC STORAGE ALLOCATION.

```
COMMON / FTB / RSTOR(340), BSTOR(200), LASTAD
REAL FILID(60), RSTOR(340), FSTOR(340)
INTEGER FILIDX(60),FILSIZ(60), SETSIZ(60)
INTEGER ISTOR(340)
LOGICAL LSTOR(340)
LOGICAL IBMRUN
EQUIVALENCE (LSTOR(97),IBMRUN)
EQUIVALENCE (RSTOR(1),ISTOR(1),LSTOR(1),FSTOR(1))
EQUIVALENCE (RSTOR(101),FILID(1)), (RSTOR(161),FILIDX(1)),
           (RSTOR(221),SETSIZ(1)), (RSTOR(281),FILSIZ(1))
```

1758 044

common block provides the space required for communication between the FTB subroutines, the FILID, FILIDX, SETSIZ and FILSIZ array described above, and 200 words to hold the first FTB link (refer to Section VII for description). All space assigned by FTB actually resides after the common block. In other words, all storage is spilled across the common block boundary, which provides a reference address.

Implicit in the storage philosphy used in the RETRAN Code Package is the requirement that different type variables, i.e. REAL, INTEGER and LOGICAL, can reside anywhere in the common storage area. Further, an array index must reference the same physical core location whether the array is REAL, LOGICAL, or INTEGER. This requirement poses no difficulty for computers where the word length for the various variable types is the same, such as encountered for both IBM and CDC single precision. While single precision arithmetic is adequate for CDC applications (60 bit words), it is not adequate for IBM applications. The approach used for IBM applicatons is to use double precision REAL*8 words (64 bit words). This would cause misalignment when INTEGER*4 numbers are equivalenced to REAL*8 numbers. For example, consider the following IBM FORTRAN coding:

```
REAL*8 RSTOR(2)
INTEGER*4 ISTOR(2)
EQUIVALENCE (RSTOR(1), ISTOR(1)
RSTOR(1) = 1.
ISTOR(2) = 1
```

The desired result is to store a real (64 bit) number in the first word of the array and an integer (32 bit) number in the second word of the array. The FORTRAN Compiler, however, assumes that the location of ISTOR(2) is 4 bytes (32 bits) after ISTOR(1). This would cause ISTOR(2) to be stored in the second half of RSTOR(1), thus destroying the contents of RSTOR(1). To avoid this problem, the IBM H-Extended compiler has an Automatic Precision Increase facility that can be used to elevate the single precision REAL variables to REAL*8 variables, and also pad LOGICAL and INTEGER variables equivalenced to the promoted variables to achieve the desired alignment. Use of the Automatic Precision Increase facility allows the physical coding to be the same for both IBM and CDC machines, since it forces consecutively addressed LOGICAL and INTEGER elements of an array to be offset by 64 bits rather than the normal 32 bits.

2.2.2 Fetching Variables in the Input Segments

Variables in the files throughout the input segments are referenced or defined using three mnemonics for the variable storage array. These mnemonics are RSTOR for real numbers, ISTOR for integers, and LSTOR for logicals. These three variables are dimensioned and equivalenced to the first word of labeled common. Fetching a variable is a matter of computing the proper subscript combined with the appropriate mnemonic, RSTOR, ISTOR, or LSTOR.

To demonstrate how indices are computed in the input segment refer to the volume file description, Table II.2-5. To obtain the thermodynamic pressure for volume 1, the index of the first word in the file would have to be obtained. This index is FILIDX(5), where the subscript 5 refers to relocatable File Number 5. This index is also the index of the first word of the first set of variables Pressure P is the third word in this set of variables. Consequently, P(1) would be represented by RSTOR (FILIDX(5)+2). To fetch P for the fourth volume requires accessing the fourth set of variables in the volume file. P(4) is represented by RSTOR(FILIDX(5) +3*SETSIZ(5)+2).

To illustrate this concept in further detail, an example of a DO loop as typically coded in most FORTRAN applications and the same DO loop as it will appear in RETRAN, is shown in Table II.2-6.

2.2.3 Fetching Variables in the Execution Segment

A different technique is used for the indexing scheme in the execution segment. This technique is based on equivalencing a group of words having meaningful mnemonics with the variable storage array according to the position dependence defined by the file directory.

To illustrate this method, recall that the data storage array is named RSTOR. The equivalencing shown in Table II.2-5 would be performed for the volume file.

This technique serves several functions. First, it sets up a set of offsets relative to the first word in a set of variables, which the compiler uses to modify the base address . This offset or address "mask" allows all words within one set to be fetched with a single index relative to the first word of labeled

8 046

TABLE II.2-5

TYPICAL FILE EQUIVALENCE MASK

```
FILID(05) - VOLUME QUANTITIES

REAL LIQH, MIXQ, LIQM, MIXV, LIQV
INTERGER PHASE, OLVLN
LOGICAL GFLAG

DIMENSION      IBUB(1),  IREAD(1),     P(1),    TEMP(1),      HORX(1),
*     V(1),    ZVOL(1),    ZM(1), JTPMV(1),   FLOWA(1),     DIAMV(1),
*  ELEV(1),    SATP(1),   SATT(1), SATVF(1),   SATVG(1),     SATUF(1),
* SATUG(1),   SATHF(1), SATHG(1),    VL(1),      VS(1),       UW(1),
*    HW(1),    GASH(1)

  EQUIVALENCE
*(ISTOR(01),IBUB(1))     , (ISTOR(02),IREAD(1))   , (RSTOR(03),P(1))       ,
*(RSTOR(04),TEMP(1))     , (RSTOR(05),HORX(1))    , (RSTOR(06),V(1))       ,
*(RSTOR(07),ZVOL(1))     , (RSTOR(08),ZM(1))      , (RSTOR(09),JTPMV(1))   ,
*(RSTOR(10),FLOWA(1))    , (RSTOR(11),DIAMV(1))   , (RSTOR(12),ELEV(1))    ,
*(RSTOR(13),SATP(1))     , (RSTOR(14),SATT(1))    , (RSTOR(15),SATVF(1))   ,
*(RSTOR(16),SATVG(1))    , (RSTOR(17),SATUF(1))   , (RSTOR(18),SATUG(1))   ,
*(RSTOR(19),SATHF(1))    , (RSTOR(20),SATHG(1))   , (RSTOR(21),VL(1))      ,
*(RSTOR(22),VS(1))       , (RSTOR(23),UW(1))      , (RSTOR(24),HW(1))      ,
*(RSTOR(25),GASH(1))
```

1758 047

TABLE II.2-6

A COMPARISON OF TYPICAL FORTRAN CODING AND RETRAN CODING WITH PRIMARY VARIABLES

Typical Form

```
      DO 10 I = 1, NVOL
      TPMV (I) = 1.0E0
      IFAN(I)  = 1
   10 JVISC(I) = 1
```

RETRAN Form

```
      ISIZ = SETSIZ(05)
      IDXVOL  = FILIDX(05)
      DO 10 I = 1,NVOL
      RSTOR (IDXVOL+51) = 1.0E0
      ISTOR (IDXVOL+53) = 1
      ISTOR (IDXVOL+54) = 1
   10 IDXVOL = IDXVOL + ISIZ
```

common. The "mask" eliminates the need to compute indices for each word in a set, thus realizing a net savings in computing time. The equivalencing technique also allows the use of meaningful mnemonics which make the code more readable for debugging purposes and later model changes.

An example of a typical FORTRAN DO loop and the corresponding RETRAN DO loop is shown in Table II.2-7 to illustrate the indexing scheme used in the execution segment.

The subscript I in the RETRAN DO loop represents the index relative to the first word of labeled common for each set of variables as it loops over NVOL sets of volume data, incremented by the SETSIZ for each loop. The "mask" supplies the offset for variables within a set.

There is one equivalencing "mask" for each major file. The "mask" for a file is in each subroutine in which variables from the file are referenced.

2.3 Addition of New Variables to the Files

Addition of new variables to the file structure is a fairly simple procedure. There are two options open to the programmer for variable additions.

First, the new variables can be added to existing files. For instance, if the new variable describes a volume and there is one of these variables for each volume, the set can be lengthened to include the new variable. This involves changing the SETSIZ in the subroutine that reserves the storage for the volume file to the new length. Then the "mask" for the volume file has to be changed so the new variable mnemonic is dimensioned and equivalenced to the next higher address in the common storage array RSTOR, so as to reflect its proper positional dependence.

The second option involves creating a new file to contain the added variables. To accomplish this task requires determining the appropriate input subroutine in which storage should be defined for the new file and allocating storage space with a call to RESERV. The file structure has to be defined so it is compatible with the dynamic dimensioning scheme. An equivalencing "mask" should then be added to subroutines in the execution segment in which the new variables are

TABLE II.2-7

A COMPARISON OF TYPICAL FORTRAN CODING AND RETRAN CODING WITH EQUIVALENCE MASKS

Typical Form

```
    DO 290 I = 1, NVOL
    AVED(I)  = 1.0EO/SPVZ(I)
    FMASS(I) = V(I)*AVED(I)
    GASM(I)  = FMASS(I)
290 CONTINUE
```

RETRAN Form With Equivalence Masking

```
    IDX5  = FILIDX(5)
    ISIZ5 = SETSIZ(5)
    IEND5 = IDX5 + ISIZ5*(NVOL-1)
    DO 290 I = IDX5,IEND5, ISIZ5
    AVED(I) = 1.0EO/SPVZ(I)
    FMASS(I) = V(I)*AVED(I)
    GASM(I)  = FMASS(I)
290 CONTINUE
```

used. Any modification made to the file structure system should be reflected in the file directory found in subroutine RMAIN which is also listed Appendix A of this manual.

3.0 SEMI-MODULAR CODING TECHNIQUE

The motive for developing the semi-modular programming technique used in the RETRAN Code Package was the desire to minimize the memory required to execute any given problem. Semi-modularization, as defined for use in the RETRAN Code Package, invloves splitting the coding into concise and well-defined blocks of coding designed to perform a specific task, where each block consists of one or several subroutines. These blocks are interfaced to the mainline code via conditional calls (called only if the option is used) and the FTB common storage area. Logic required to direct the overall program flow is relatively simple, especially compared to that involved when coding for multiple tasks is lumped into a single subroutine as is done in some large codes. Splitting the optional program models into blocks makes it a simple task to omit options from the load modules which are not used.

3.1 Dynamic Memory Management

It was originally envisioned that a pre-processor or executive code could be used to generate problem-oriented loader directives by scanning the users input data and determining the optional program blocks used. The unused options could then be omitted from the load module at execution time (link edit required at execution time), by leaving the unused options unsatisfied. This scheme was abandoned due to the additional cost incurred by requiring a link edit step for each job and the added complexity of the job.

The dynamic dimensioning and semi-modularization schemes utilized in the RETRAN code package are intimately related through interfacing with the FTB package. This integrated scheme of dynamic dimensioning and semi-modularization is referred to as dynamic memory management. Since FTB assigns data files from a single scratch array according to the attributes of a file reservation request, proper assignment of the scratch array origin can allow FTB access to the bulk of the user's area in memory during execution. In RETRAN, the FTB scratch array includes most of the physical coding as well as the area of memory normally available for data storage, as illustrated by Figure II.3-1. The FTB scratch array is placed in labeled common which is loaded at the end of the root segment or overlay, in effect making the scratch area length equal to the field length less the length of the root segment. An FTB file is created which reserves

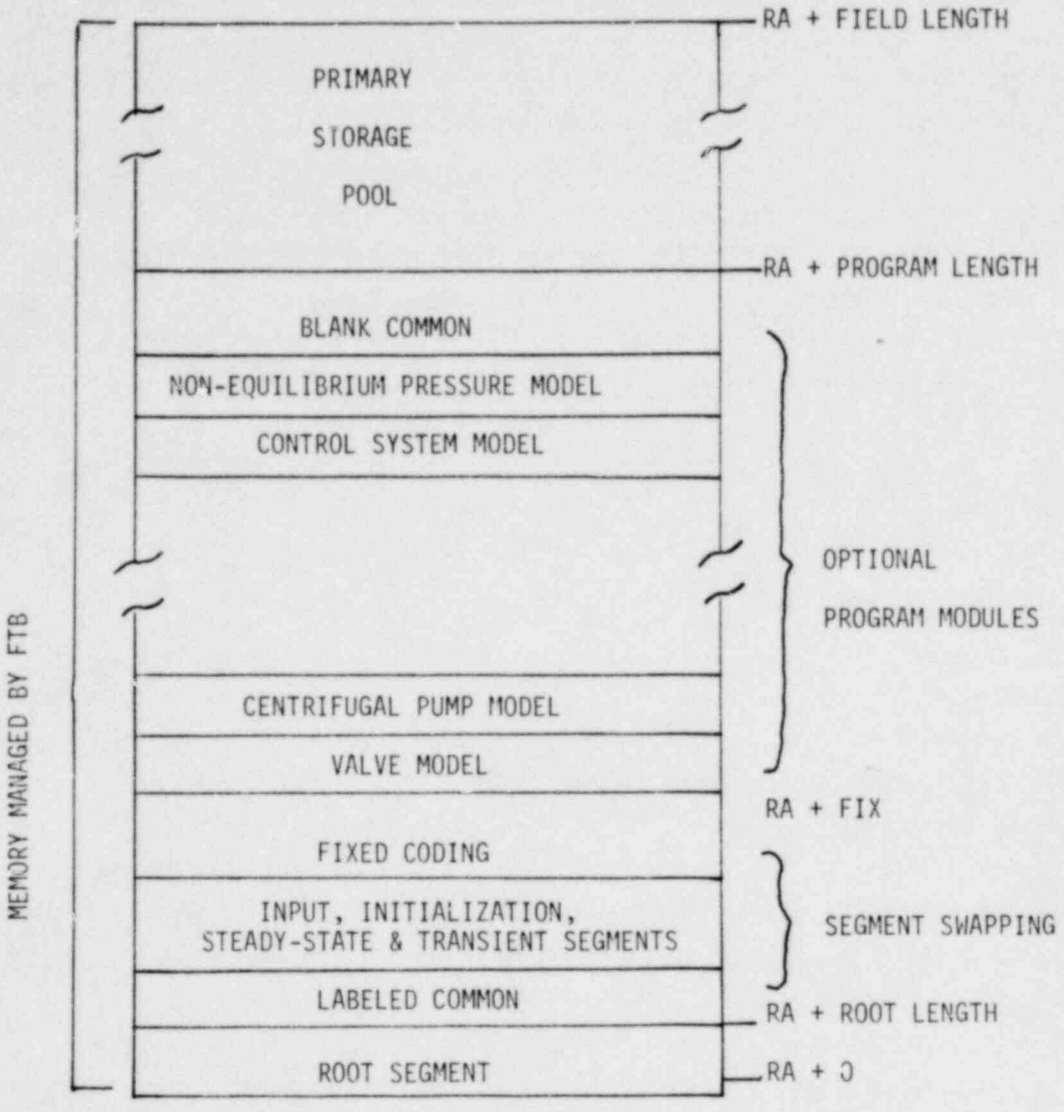

Figure II.3-1 RETRAN Dynamic Memory Management Scheme

memory extending from the end of labeled common through address RA + FIX. This area in memory is not used to store data, but the file prevents actual data files from being over-stored on required coding residing between labeled common and address RA + FIX. The optional program modules reside in memory between addresses RA + FIX and RA + PROGRAM LENGTH. An FTB file is reserved in the appropriate area of memory for each block of coding associated with a particular module whose use is optional. Upon scanning the input data supplied to describe the RETRAN problem, unused options are isolated and the files used to protect the corresponding areas of memory deleted, thus freeing otherwise dead code areas for data storage. The dynamic memory management scheme in effect allows generation of efficient problem-oriented load modules at execution time without the use of a loader and in a manner completely transparent to the user. In effect, the scheme dynamically maximizes the utilization of memory.

The key to the dynamic memory managment scheme used in the RETRAN Code Package is that the programmer must be able to determine the order in which the optional program blocks are loaded. By ordering the optional program blocks it is possible to minimize the degree to which the core is fractured, thus maximizing the utilization of the space occupied by unused options for data storage. The ordering also makes it much easier to ascertain the area of memory occupied by unused as well as active option program blocks which must be protected.

The discussion of the dynamic memory management technique given above was directed toward the RETRAN and RESTRT program modules, which contain large blocks of optionally used coding. Program modules REEDIT and PLOTER contain no optional programs; consequently, the logic required to reserve FTB files to protect the required coding is greatly simplified. Only a single area in memory must be located (the area above the dashed line and below the solid line in Figure II.3-2) for the PLOTER and REEDIT coding file, while all remaining memory is available for storage. Figure II.3-2 illustrates the relative core requirements for each of the program modules in the RETRAN Code Package.

3.2 Computing System Related Deficiencies

The semi-modularization feature just described was originally developed on a CDC computer and relied heavily on the segmented loader. During the conversion from CDC to IBM computing systems, an IBM 360/195 with an OS operating system

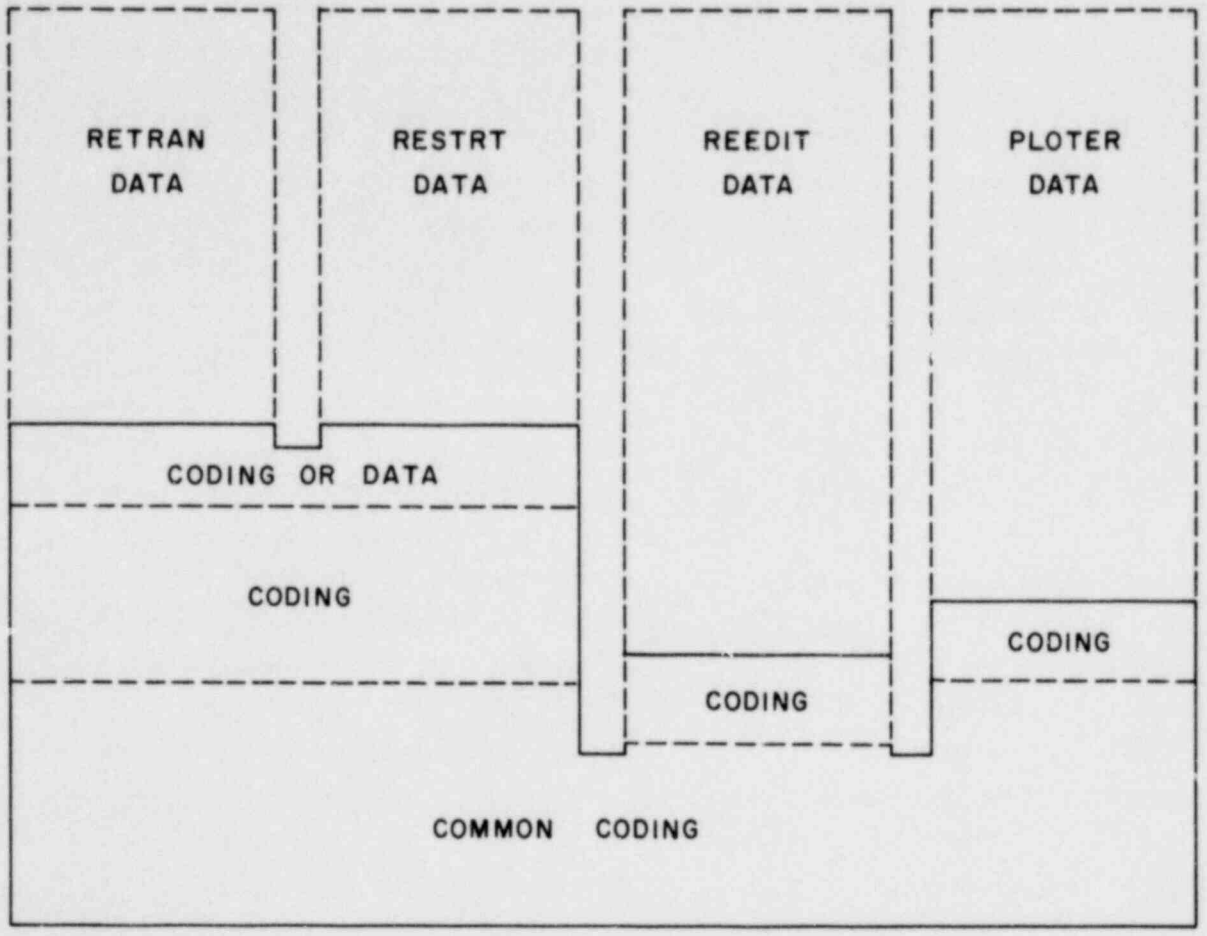

Figure II.3-2 RETRAN Code Package Core Requirements

1758 055

was used. At this time some difficulties were encountered due to the primative nature of the IBM overlay loader as compared to the CDC segmented loader. In particular, subroutines could only reside in one segment or overlay and no way was available to specify the loading sequence in a given overlay. Also, the limitation of a maximum of four regions proved to be a liability.

As a matter of expediency, the early IBM versions of the RETRAN code package were overlayed, but the dynamic memory management feature was not fully activated, i.e. the areas of memory containing unused code options were not used for data storage. A scheme was devised whereby the full dynamic memory management capability could be activated by splitting the subroutines for a given option into two overlays. The first overlay would contain a single subroutine used to locate the optional program block, while the second and succeeding overlay would contain the additional subroutines required for the optional program block. The subroutine in the second overlay could then be loaded in random order without affecting the ability to locate the origin of the optional coding block defined by the first overlay.

Prior to the actual implementation of the dynamic memory management scheme described above for IBM computing systems, it was discovered that a significant degradation in the code running time was being observed on MVS operating systems. It was later discovered that the code was spending an inordinate amount of time in the overlay entry tables, effectively increasing running time by a factor of 3 to 4. This was perplexing, particularly since no segment swapping was taking place. Consequently, the overlay structure was reduced to a minimum and running time improved by a factor of 3 to 4. At this point it was decided to abandon the dynamic memory management scheme as a result of the difficulties encountered with the overlay loader on MVS systems, particularly since the largest portion of prospective RETRAN users with IBM computing systems run under an MVS operating system.

The lack of any real overlay structure for the IBM/MVS version significantly increases the amount of virtual storage required (typically increased from 700K to 1300K), but the effective cost of the increased overhead is more than compensated for by the decrease in running time that is realized. A significant portion of the code which is used briefly (input subroutines), or not at all (unused program options), resides on mass storage and is seldom, if at all, paged into or out of real memory.

The original overlay structure, with partial dynamic memory managment, has been retained for use on IBM computing systems operating under an OS operating system. No degradation in performance has been observed using this overlay structure on OS where the overlay loader functions as described in the documentation[II.1-6]. Section VIII discusses the various overlay structures used.

III. RETRAN AND RESTRT THERMAL HYDRAULICS MODULES

The segmentation of the RETRAN and RESTRT modules can best be described by following a problem through the program structure for input processing and execution. The program structure for restart and initial runs is very similar, so the segmentation for each will be discussed concurrently.

The discussion in the following sections is directed to applications on CDC computing systems, where the segmented loader is used extensively[II.1-4, II.1-5]. As noted in Section II.3.2, deficiencies in the IBM overlay loader [II.1-5], particularly when MVS is used, have precluded use of the overlay feature. As a result, the content of the following information does not apply directly to IBM versions of the RETRAN Code Package. However, should the loader difficulties be isolated and corrected at a later date, the information presented below may be useful in developing a detailed overlay structure for IBM computing systems.

1.0 BASIC RETRAN CODE PACKAGE MODULES

Segment RMAIN is the root segment of RETRAN (refer to Figure III.1-1). Segment RMAIN is a fixed length block of coding containing subroutines that are resident in core for all phases of the problem. These subroutines are used in restart runs, initial runs, reediting runs, and plotting runs. RMAIN acts as the driver for the RETRAN Code Package.

The segment INPUT is loaded first resulting from a call to subroutine INPUT from subroutine RMAIN. Segment INPUT initializes the FTB package and the variable data storage array. Subroutine INPUT reads and processes the problem dimensions data card to determine what type of problem is to be run (restart, initial run, re-edit, or plot). This information is then returned to subroutine RMAIN which selects the appropriate tree to overlay Segment INPUT. If the problem is a plotting run, the segments in the Tree MOD3 are loaded. If the problem is a re-edit run, the segments in the Tree MOD2 are loaded. The segmented program structures for re-edit and plotting are described in Sections IV and V respectively. If the problem is a restart or initial run, segments in the Tree MOD1 and Tree OPTIONS are loaded. The segmentation for TREE MOD1 is shown in Figure III.1-2 and a schematic of Tree OPTIONS is shown in Figure III.1-3.

The base of the Tree MOD1 is the Segment RETRAN. Subroutine RETRAN is the driver for the RETRAN module. Segment TRAN contains subroutines that are required by the transient portion of the code. Segment CPYPLT is used for restart problems only. Its function is to copy archived data to the point of restart from the restart tape to a new data tape. Subroutines in Segment REDUCE reduce the field length and release any unused contiguous block of main memory. Segment PRNPLT contains subroutines that are used for printer plots and are loaded after all calculations are completed. Tree INMODS is the input processing portion of the code for both restart and initial runs. A schematic of the Tree INMODS is shown in Figure III.1-4. The Tree INITAL is the steady state initialization portion of the code. A schematic of the Tree INITAL is shown in Figure III.1-5.

1758 060

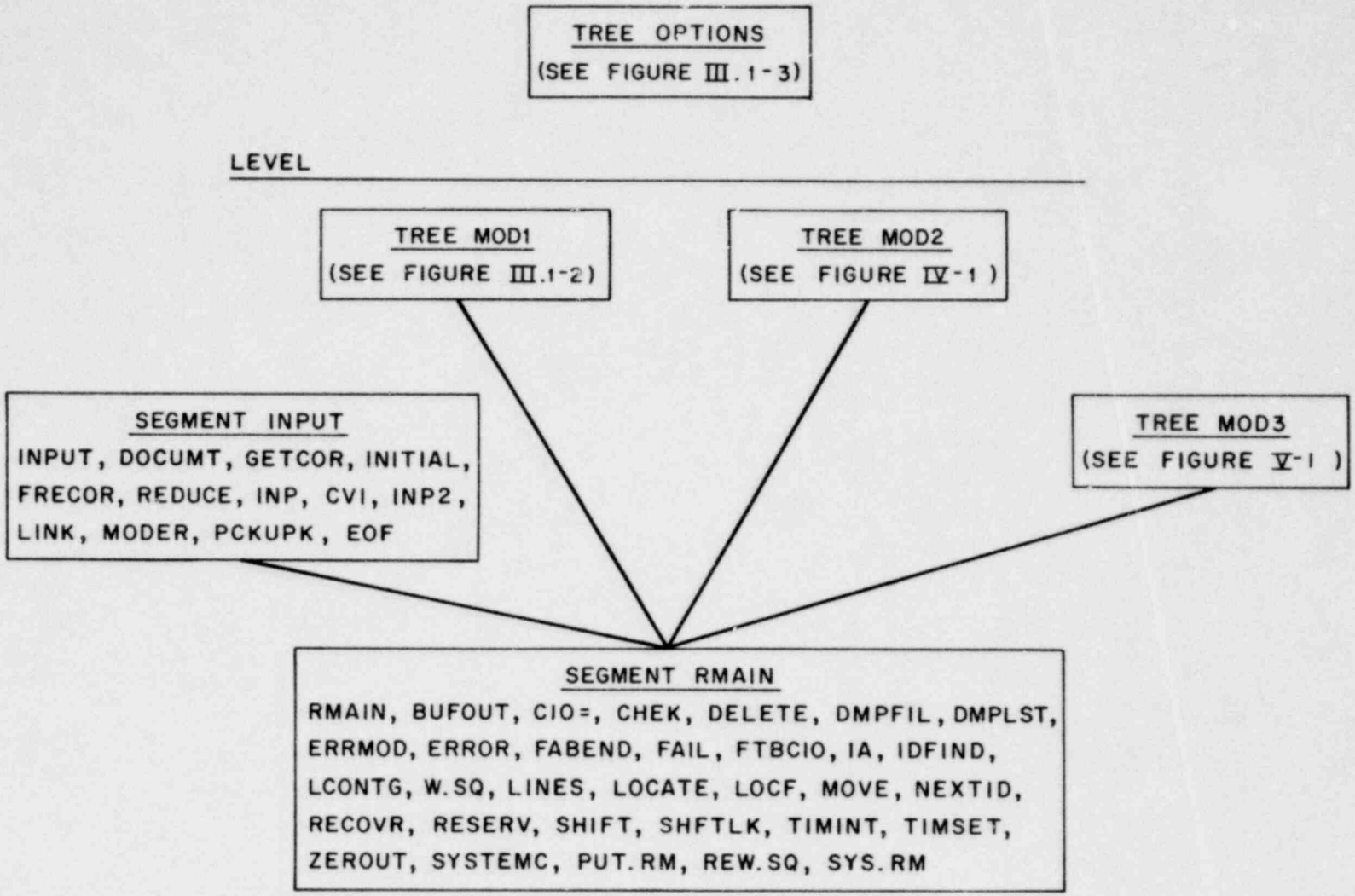

Figure III.1-1 Schematic of the Tree RMAIN

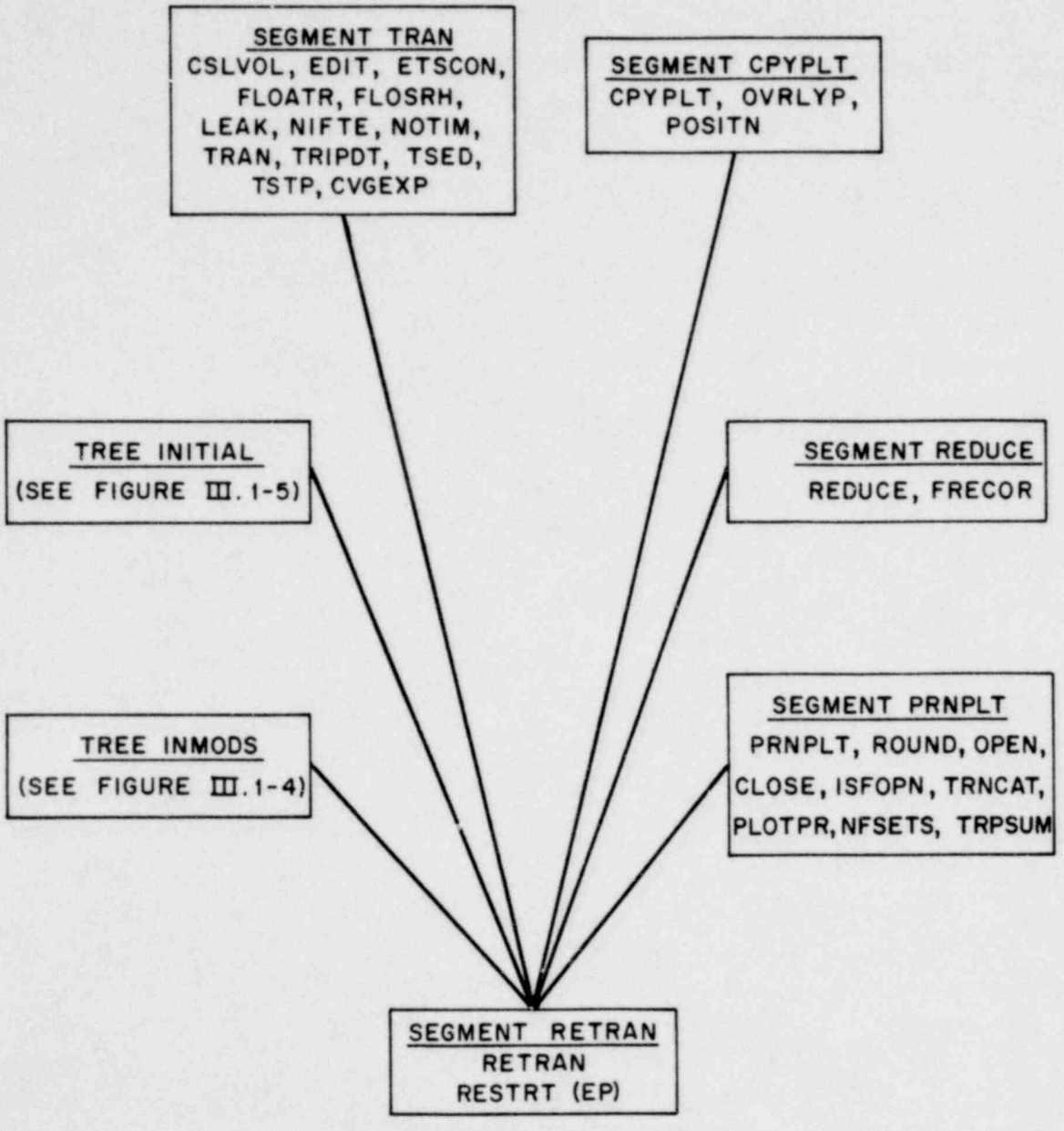

Figure III.1-2 RETRAN and RESTRT Program Module Segmentation Tree MOD1

1758 042

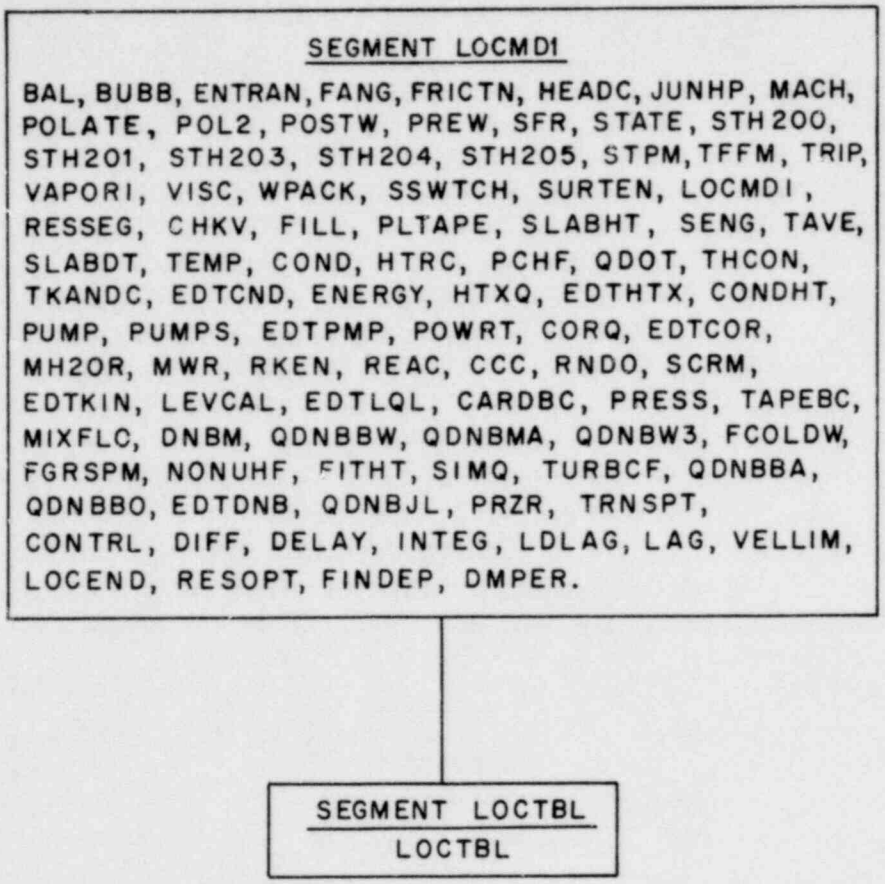

Figure III.1-3 Schematic of the Tree OPTIONS

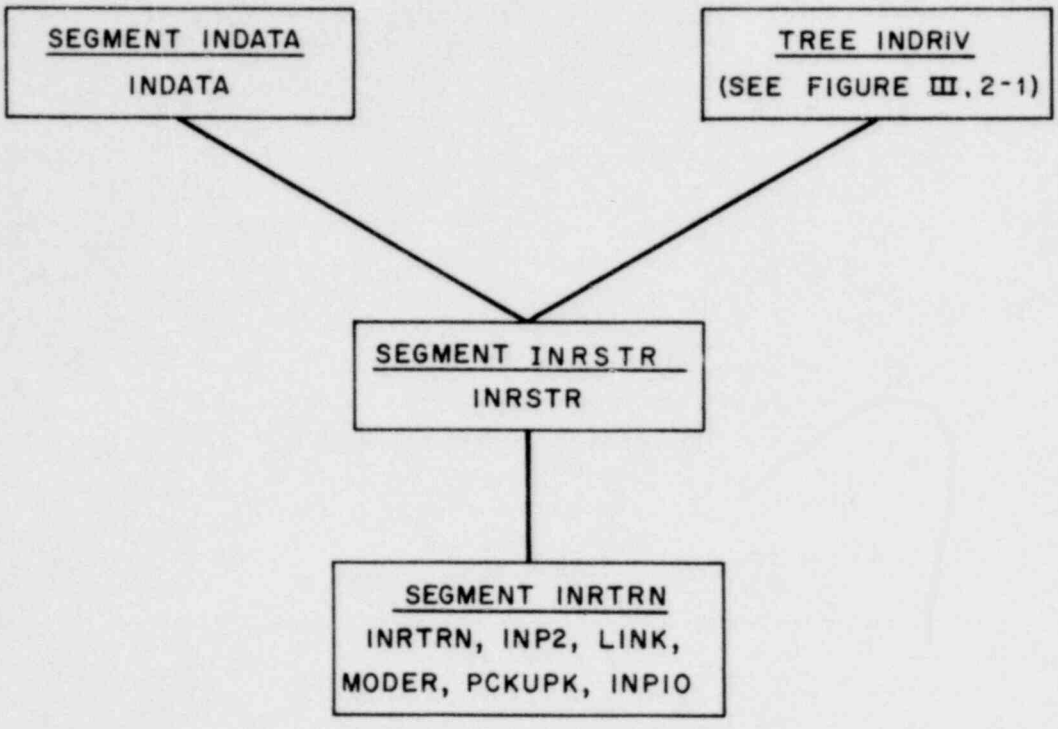

Figure III.1-4 Schematic of the Tree INMODS

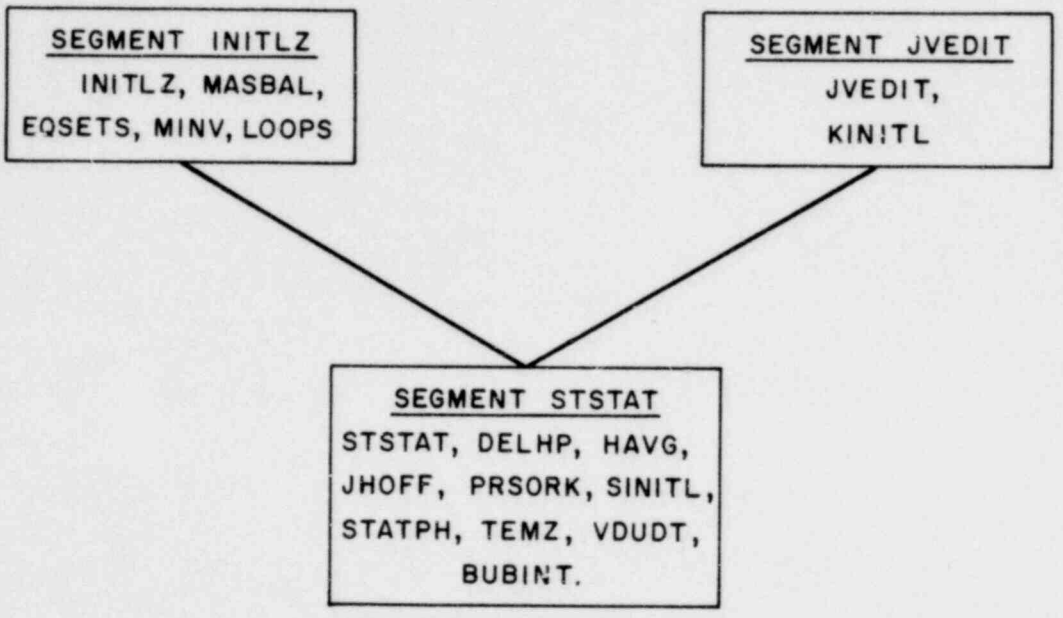

Figure III.1-5 Schematic of the Tree INITAL

2.0 INPUT PROCESSING SEGMENTATION

The segments containing the subroutines that process the input data are contained in the Tree INMODS (see Figure III.1-4).

2.1 Input Processing Segmentation for an Initial Run

For initial runs, segments INRSTR and INDATA are not used. Segment INRTRN contains subroutines from the FTB package and INP package that are used for both restart and initial run input processing. Subroutine INRLPE reads the label for a plot tape if one is to be generated, and then calls subroutine INTRAN. Subroutine INTRAN is in segment INTRAN, which is the base for the TREE INDRIV (refer to Figure III.2-1). Subroutine INTRAN is the driver for the input processing subroutines. Each of the segments branching from the segment INTRAN are used only once; therefore, each segment overlays the previously used segment. After all the input data is processed, control is returned to segment RETRAN (Figure III.1-2) at which time the Segments in the Tree INITAL overlay the Tree INMODS.

2.2 Segmentation for Restart Input Processing

For a restart run, the original input data is retrieved from the data tape and combined with the minimal input data supplied with restart. This set of input data is then processed as if it were an initial run so the dynamic data storage is set up.

Subroutine INRSTR reads the label on the tape containing the restart data. INRSTR then reserves a file for the original problem data to be stored. INRSTR then calls subroutine INDATA in segment INDATA. INDATA retrieves the original problem input data from tape. The input data is then processed in segments of the Tree INDRIV as if it were an initial run.

Control is then returned to subroutine RETRAN in segment RETRAN (Figure III.1-2). RETRAN calls subroutine CPYPLT which retrieves the problem data at the appropriate restart time from the data tape. Subroutine OVRLYP is than called by RETRAN. OVRLYP overlays the data storage files with the restart data. The combination of original initial input data and the restart data provides all the information necessary to continue the transient calculations.

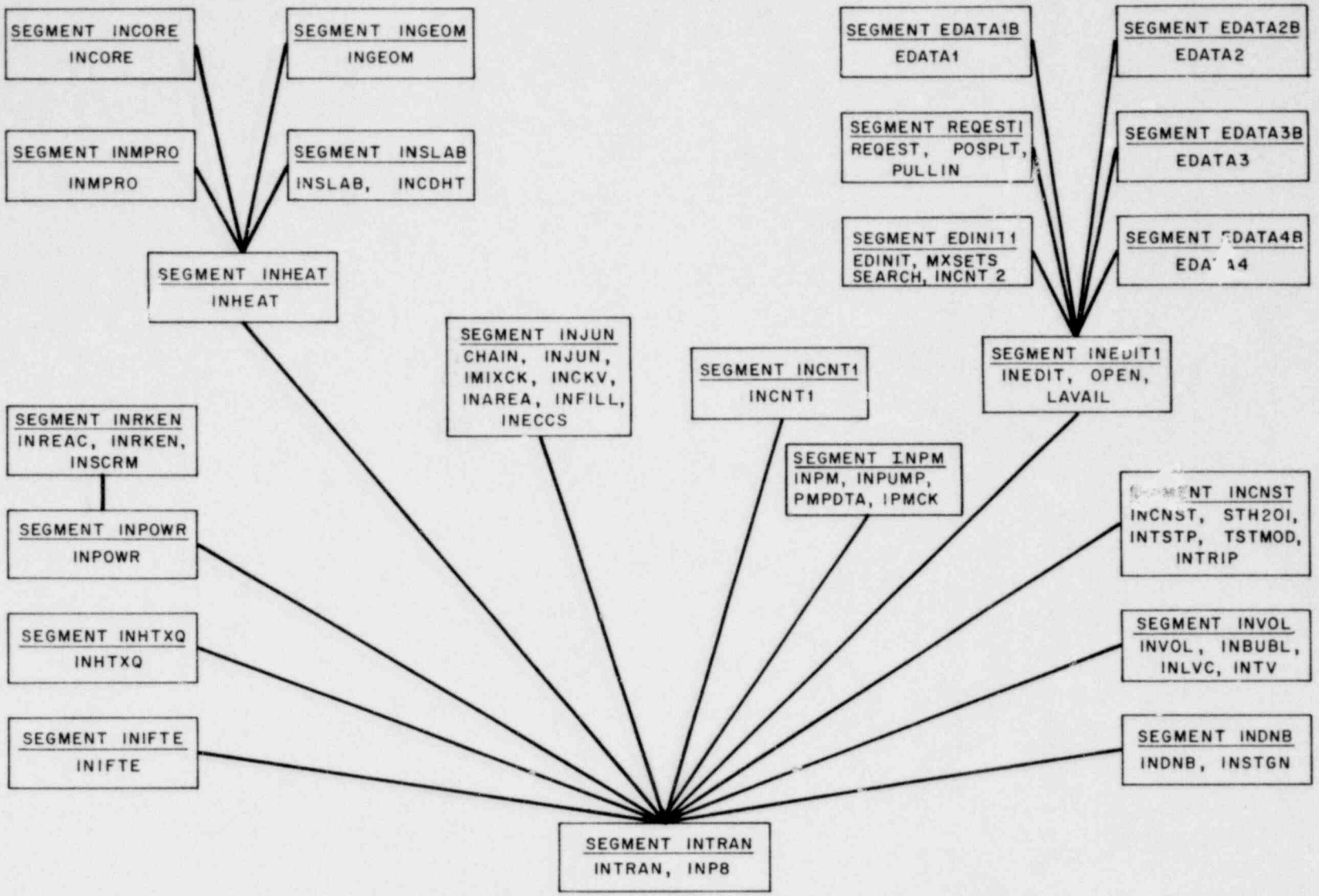

Figure III.2-1 Schematic of the Tree INDRIV

3.0 STEADY-STATE INITIALIZATION SEGMENTATION

Segments from the Tree INITAL are loaded after the input processing has been completed. Tree INITAL is needed for initial runs only and not for restarting. A schematic of the Tree INITAL is shown in Figure III.1-5. The driver for the steady-state initialization package is subroutine STSTAT in segment STSTAT. Steady-state initialization and the transient calculations share a common block of subroutines. These subroutines are located in the Tree OPTIONS (Figure III.1-3). Subroutines in the Tree INITAL are exclusively used for steady-state initialization and are overlayed by Segment TRAN after the steady-state calculations are complete.

4.0 EXECUTION SEGMENTATION

The subroutines in the Segment TRAN (Figure III.1-2) are loaded for transient execution. Subroutines in the segment TRAN are necessary for execution of all problems and are in core at all times during transient execution.

The Tree OPTIONS (See Figure III.1-3) is present in core through input processing, steady state initialization, and transient calculations. The Segment LOCMD1 contains a block of subroutines that are needed for all initial runs and restart runs. The optional subroutines are located after subroutine LOCMD1 as part of the semimodularization scheme. (See Section II.3.0).

IV. REEDIT DATA TAPE EDITING MODULE

The REEDIT module is essentially a stand-alone computer program which is included in the RETRAN Code package to facilitate the generation of edits of RETRAN data tapes. REEDIT is fully dynamically dimensioned and also contains dynamic field length reduction features. The REEDIT module segmentation is shown in Figure IV-1. Subroutine REEDIT in segment REEDIT is the driver for this module.

The discussion in the following sections is directed to applications on CDC computing systems, where the segmented loader is used extensively[II.1-4, II.1-5]. As noted in Section II.3.2, deficiencies in the IBM overlay loader [II.1-5], particularly when MVS is used, have precluded use of the overlay feature. As a result, the content of the following information does not apply directly to IBM versions of the RETRAN Code Package. However, should the loader difficulties be isolated and corrected at a later date, the information presented below may be useful in developing a detailed overlay structure for IBM computing systems.

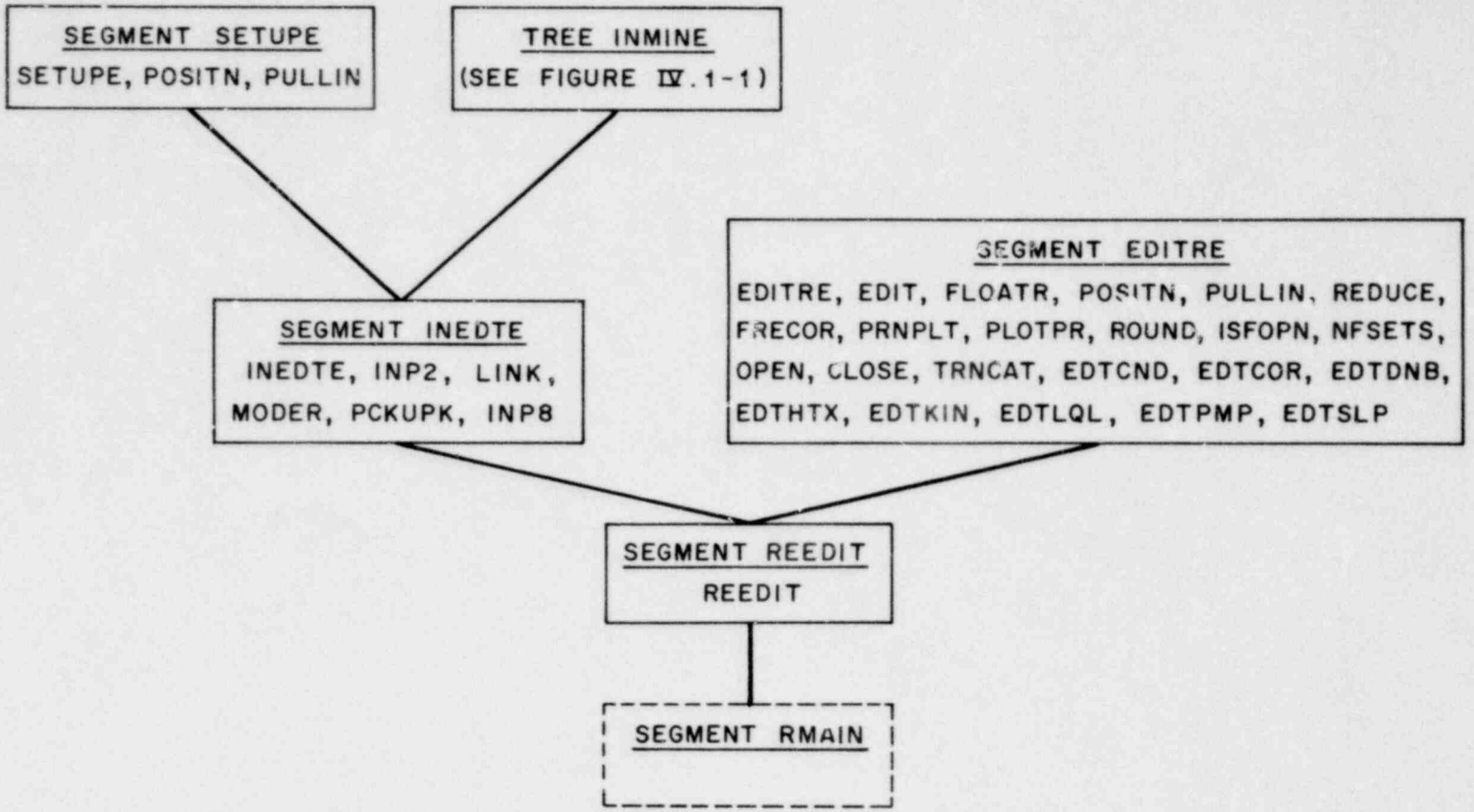

Figure IV-1 REEDIT Program Module Segmentation Tree MOD2

1.0 INPUT SEGMENTATION

The input segmentation as well as the overall segmentation for the program module REEDIT is brief, due to the simplicity of REEDIT. The storage allocation performed by the input segments resides in the region ranging from the end of the longest load configuration to the end of the field length. Permanent storage arrays ("files") are assigned the preferred end of core (end nearest the executable coding), while temporary scratch files are assigned to the least preferred end of core.

Subroutine INEDTE in Segment INEDTE is the driver for the input processing. INEDTE reads the minimal input data supplied for re-editing. INEDTE calls subroutines in Segment SETUPE which set up relocatable files to store the data from tape, read the header record off the tape, position the code at the correct data record and read the data record. INEDTE then calls INEDIT in the Tree INMINE (See Figure IV.1-1). The primary function of the segments in the Tree INMINE is to set up edit headings. Once the input segments have been executed control is returned to subroutine REEDIT. The field length is reduced so as to free all unused core that resides above the permanently assigned files.

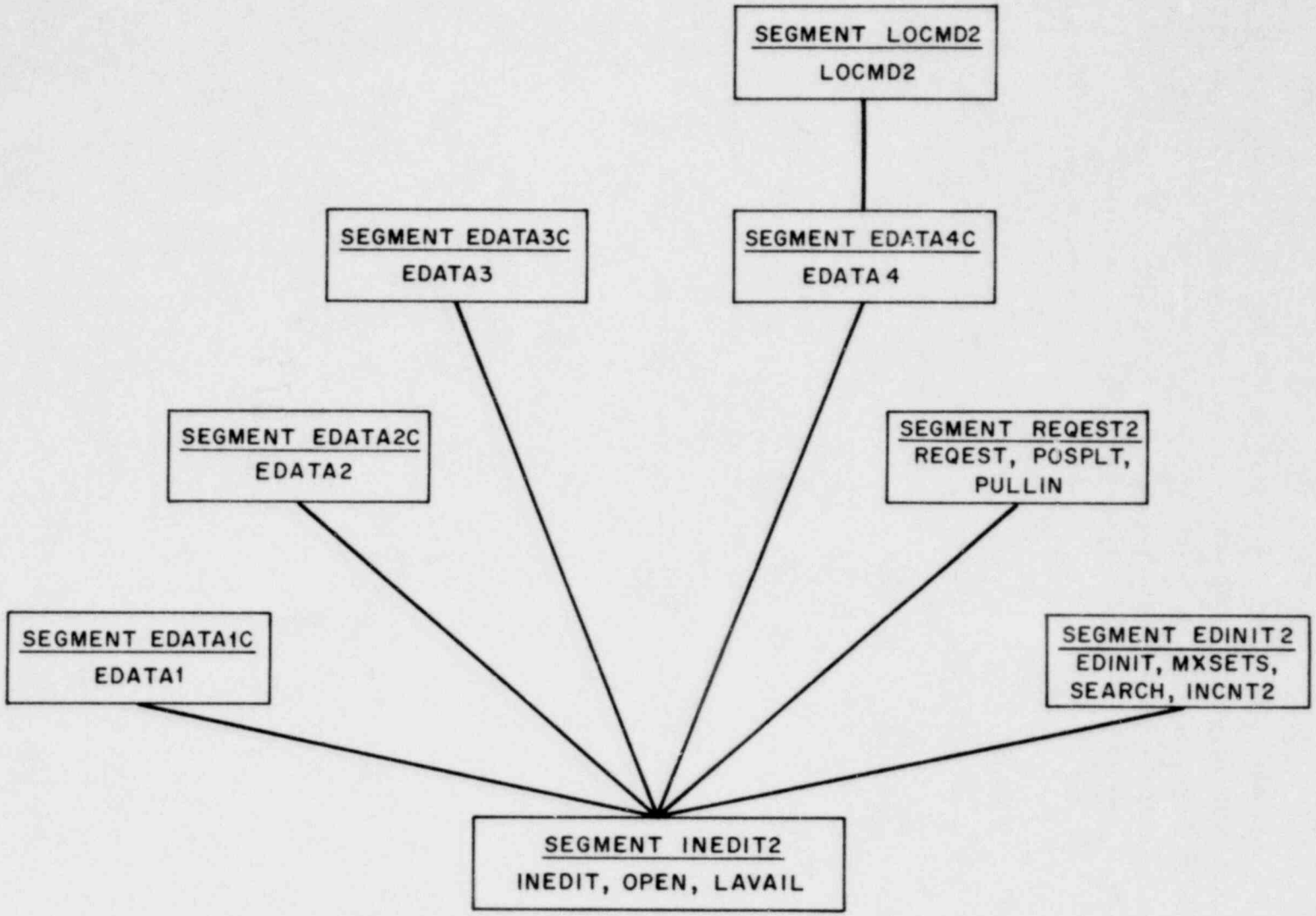

Figure IV.1-1 Schematic of the Tree INMINE

2.0 EXECUTION SEGMENTATION

The execution segment EDITRE (see Figure IV-1) controls the data tape editing process. Subroutine EDITRE selects the appropriate data records to be edited, controls data tape positioning and checking performed by subroutine POSITN, and calls the data retrieval subroutine PULLIN. EDITRE is the driver subroutine for the data tape editing function performed by REEDIT.

V. PLOTER PLOTTING MODULE

The PLOTER module is the portion of the RETRAN Code package to facilitate generation of plots from RETRAN data tapes, RELAP4 data tapes, and stranger data tapes.

The PLOTER module shares a common pool of subroutines with the RETRAN code package in the root segment RMAIN. The PLOTER module contains subroutines that are unique to this module and some subroutines that are present in other modules. A schematic of the segmentation of the PLOTER module is shown in Figures V-1 and V-2. Subroutine PLOTER is the driver for the plotting package and is called from subroutine RMAIN.

The discussion in the following section is directed to applications on CDC computing systems, where the segmented loader is used extensively[II.1-4, II.1-5]. As noted in Section II.3.2, deficiencies in the IBM overlay loader [II.1-5], particularly when MVS is used, have precluded use of the overlay feature. As a result, the content of the following information does not apply directly to IBM versions of the RETRAN Code Package. However, should the loader difficulties be isolated and corrected at a later date, the information presented below may be useful in developing a detailed overlay structure for IBM computing systems.

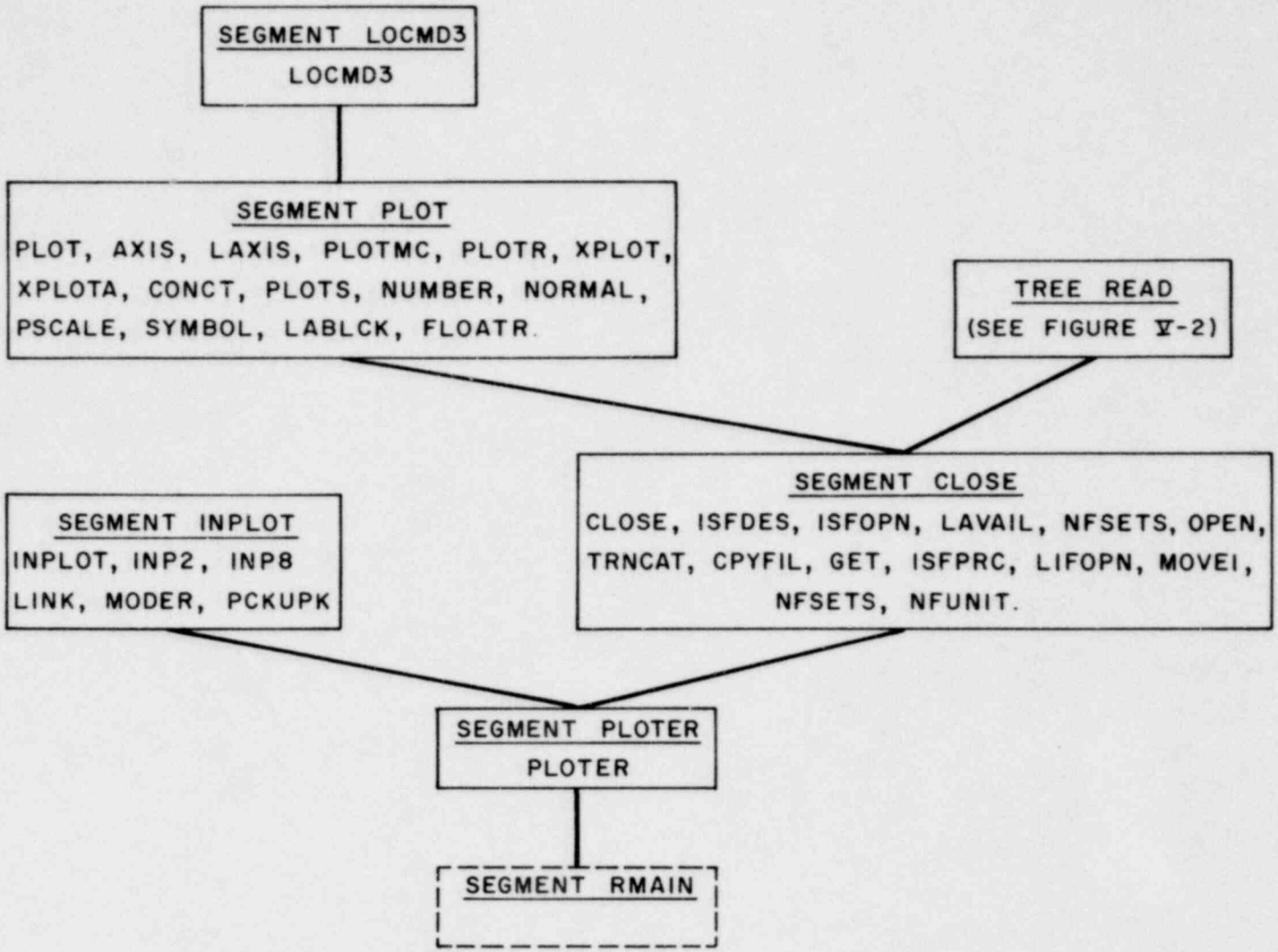

Figure V-1 PLOTER Program Module Segmentation Tree MOD3

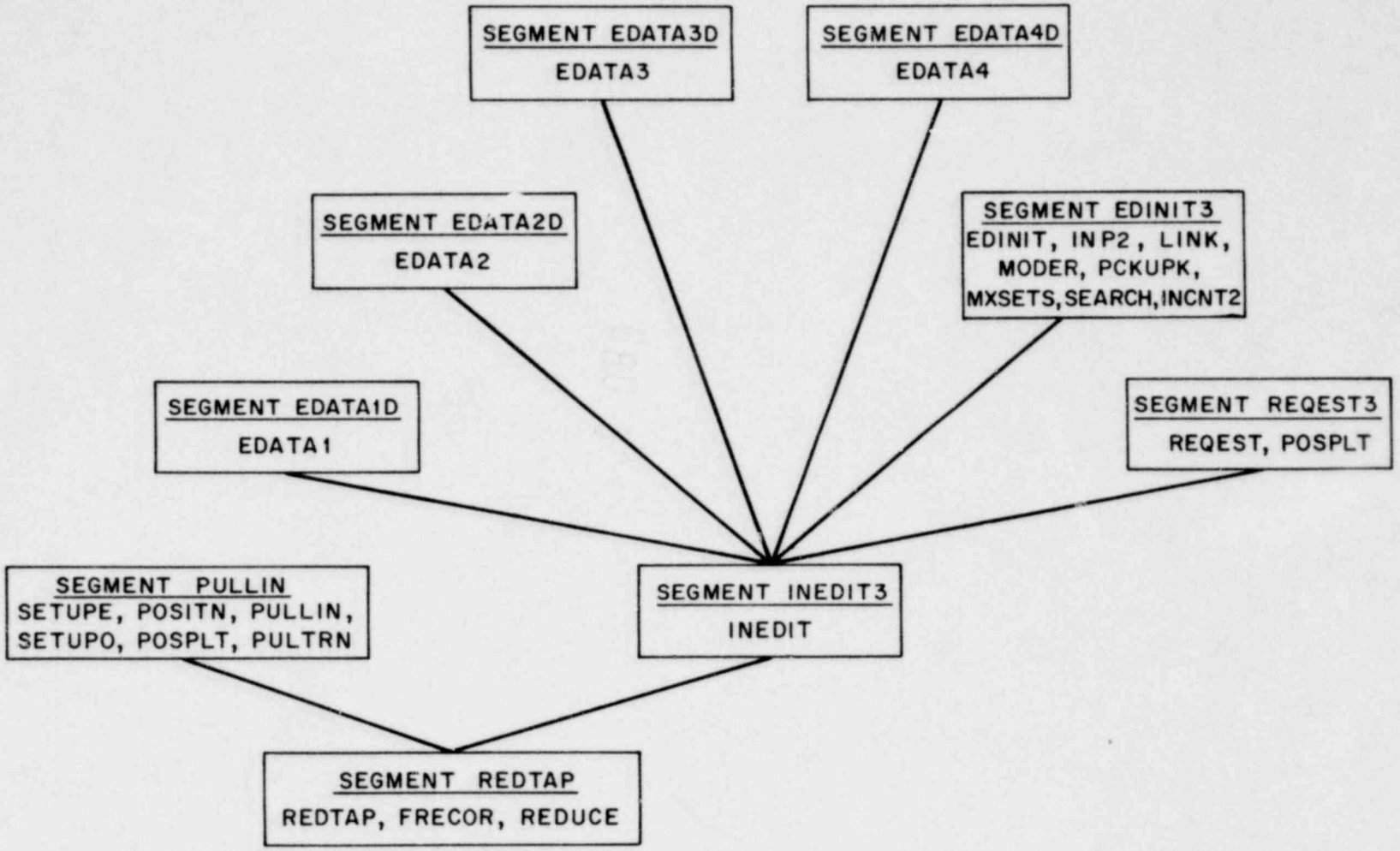

Figure V-2 Schematic of the Tree READ

1.0 INPUT SEGMENTATION

Subroutine PLOTER calls subroutine INPLOT in segment INPLOT. Subroutine INPLOT reads, checks and edits the input card data. INPLOT also sets up the permanent storage arrays and some temporary scratch files. Control is then returned to subroutine PLOTER.

2.0 EXECUTION SEGMENTATION

After the input is processed, subroutine REDTAP is called. REDTAP (See Figure V-2) is the driver for reading data records from a data set or multiple data sets. The subroutine selects the requested plot variables from each data record and stores the plot variables for the plotting subroutines.

The function of subroutine INEDIT in Segment INEDIT and the subroutines it calls is basically to set up plotting variable headings and units.

Subroutines in Segment PULLIN set up abbreviated permanent storage arrays to store information from each data record, position the code at the appropriate data record and then read the data records.

Subroutines in Segment PLOT perform the actual plotting after the requested plot variables have been retrieved from the data tape.

VI. RETRAN DATA TAPE

The RELAP series of codes have historically contained features for archiving problem solution data on magnetic tape. RETRAN, not unlike its predecessors, also contains the capability of generating data tapes for use with companion program modules RESTRT, REEDIT, and PLOTER. RETRAN data tape generation and usage is based upon a new and more efficient technique (with respect to I/O time consumption and usable data density) than employed in versions of the RELAP program.

1.0 DATA TAPE DESCRIPTION

RETRAN data tapes are standard labeled magnetic tapes and are composed of three types of records; a header record which is the first record on all data tape volumes, data records containing problem solution results and a trailer record written as the terminal record for a RETRAN data set. A data set is defined as a continuum of solution results from time zero to problem end time. A RETRAN data set may consist of a single volume (reel of magnetic tape), or it may consist of multiple volumes. Each volume of a multiple volume data set will have a header record as the first record, followed by a series of data records. Only the last volume of a data set will be terminated by a trailer record. If an end of volume is encountered while writing a data record, the volume is unloaded, a new volume mounted, a header label written and the full data record written on the new volume.

The feature whereby each volume is initiated with a header record, followed by complete data records (excluding any partial record at the end of a volume), negates the dependency of a given volume of a multiple volume set on any predecessor volumes for RESTRT problems and possibly some REEDIT and PLOTER problems. For example, a RESTRT problem which is to continue a RETRAN solution at a time corresponding to a data record in the fourth volume of a RETRAN data set, requires that only the fourth volume be mounted. This volume independent feature thus negates the need for reading volumes one through four merely to be able to position the data tape as required by RELAP. Additionally, RELAP performs a reel to reel copy as the positioning operation is performed, doubling the number of magnetic tapes the user must account for. RESTRT performs a reel to reel copy only for the volume needed to continue the problem (volume four in the example), thus minimizing I/O time and the number of tapes for which the user is accountable. A continuous data set is preserved by use of the volumes preceding the volume containing the restart data record (volumes one through three) from the previous problem solution, plus any additional volumes generated by RESTRT (volume four at a minimum). At least one data tape is generated for any RESTRT problem.

1.1 Header Record Description

RETRAN data tape header records consist of problem identification and description data, dimension specifications and the original problem input data. Header records contain blocks of specified information, where each block of information must be present, but the actual length of each block varies according to problem requirements such as dimensions and models used. Table VI.1-1 describes the contents of a data tape header record.

The information contained in the header record is used to verify that the desired data set has been requested, in addition to providing original problem detail for the program modules using the data set, and to ensure that the archived data is consistent with the requested usage. Header information is also used to set up buffers used in retrieving data from a data tape and to provide RESTRT with a physical description of the system being modeled through the original input data.

1.2 Data Record Description

RETRAN data records contain all of the time dependent data necessary to continue a problem solution through use of program module RESTRT and all information required to obtain edits and plots of archived data through use of program modules REEDIT and PLOTER. The use of a single data record for restarting, editing and plotting is a departure from the philosophy utilized in all versions of the RELAP code, where both plot and restart records were saved. Use of a single data record in the RETRAN Code Package was facilitated by expanding the list of edit variables (major and minor edit parameters) to meet user requests and by adding a minimal number of additional parameters required for restart only. No time invariant parameters such as geometry and material properties, except component node numbers used for editing, are saved in the data records. These constants are obtained at restart time in the same manner as is done for a normal RETRAN problem.

RETRAN data records are composed of a three-word prefix header followed by the actual solution data. The prefix header is an integral part of the data record consisting of

TABLE VI.1-1

HEADER RECORD DESCRIPTION

Word	Description of Contents
1 to 4	Header Label - This information specifies that tape is a RETRAN DATA TAPE.
5 to 19	Documentation Header - Information for documenting the version of RETRAN from which the data tape was created.
20	Date - Calendar creation date of the data tape.
21 to 29	Problem Title - User supplied title for the archived data.
30	LEN1 - Length of the problem dimension file.
31 to 30+LEN1	Problem Dimension Specification - Problem dimension file (Relocatable file 2) from original problem.
31 + LEN1	LEN2 - Length of the Data Record Description information.
32 + LEN1 to 31 + LEN1 + LEN2	Data Record Description - Relocatable file 43. This file is described later in this section.
32 + LEN1 + LEN2	LEN3 - Length of the original problem input data.
33 + LEN1 + LEN2 to 32 + LEN1 + LEN2 + LEN3	Original Problem Input Data - This card data is in INP table format.

(1) Hollerith word, DATA REC
(2) Integer data record number greater than or equal to one
(3) Integer data record length not including the prefix header.

The content of a data record is specified by the Data Record Description (DRD) file (relocatable file 43 in Appendix A). The DRD file is generated after the input data is processed. The DRD file specifies a set of relocatable files to be included in the data record, selected from a set of predetermined relocatable files. The predetermined list consists of optional as well as required files. While required files are always included in a data record, optional files are either included or excluded from data records depending upon whether or not the specific option is used by the problem.

Files specified in the DRD file are not included in a data record in their entirety, but rather in abbreviated form. The abbreviated form of each file is given in a subfile of the DRD file called the Abbreviated File Description (AFD). The AFD contains a set of data groups consisting of one word specifying the number of words from the corresponding file in the DRD list which are to be included in a data record, followed by a list of FTB addresses (relative to the first word of labeled common) for parameters to be included in the data record. Such an abbreviated description is required for each file specified in the DRD file and redefines a file's set structure for data record storage purposes. The abbreviated description for an optional file is not included in the data record if the option is not used. Files composed of multiple sets (refer to Section II) require an AFD for only the first set, since a given parameter for sets other than the first may be fetched by using the first set offset and adding the product of the set number minus one and the set size. The resultant is then summed with the index of the first word in the file to obtain the index of the desired parameter.

The DRD is the key to reading and writing data records for restarting, plotting, and re-editing. A thorough understanding of this file and how it is set up is necessary if any new variables are added that require archiving on a data tape.

There is a set of four variables, NFIL, LFIL, KFIL, and MFIL kept in the DRD file to describe each relocatable file that is archived on a data tape. (Refer to relocatable file 43 description). NFIL is the relocatable file number for

major files. For subfiles, NFIL is the relocatable file number associated with the subfile and is negative to indicate it is a subfile. LFIL is the number of variables in a set for the abbreviated relocatable file. KFIL is the number of sets in the relocatable file (refer to Section II). MFIL is a flag to indicate if the relocatable file is a major file (MFIL=0), a major file that has a subfile archived (MFIL=1), or a subfile. In the case of a subfile, MFIL is defined such that the top half of the word is the offset in the abbreviated major file to the index of the subfile and the bottom half of the word contains the offset to the index of the subfile for fully described major files. The set of four variables for each relocatable file is defined in subroutines EDATA1, EDATA2, EDATA3, and EDATA4.

The set of four EDATA subroutines are intimately tied in with the DRD file, the AFD, minor editing, and major editing. The function of these subroutines is to set up information for the DRD file, obtain offsets for the AFD, determine which optional relocatable files are to be included in a data record, set up edit headings, define minor edit request mnemonics, and define units for each variable available for editing. A section of coding from EDATA3 with variable definitions is shown in Table VI.1-2.

Modification of data record contents is facilitated by changing the appropriate EDATA subroutine. Changes may include adding files to a data record or merely expanding an existing abbreviated file description.

Any modifications to the DRD file or AFD subfile will be propagated throughout the RETRAN Code Package since RETRAN data tape usage is facilitated through the DRD and AFD files.

TABLE VI.1-2

A SECTION OF CODING FROM EDATA3

```
C
C
C       LIQUID LEVEL VARIABLE    (RELOCATABLE FILE 32)
   60   II = II + 1
        IF (NLVC(I2).LE.0) GO TO 70   Check to see if this file is to be
        I = 1                         included in a data record.
        IOLD = IX
        Y(1,IX) = FLAG32(I)           Information needed for each variable.
        LOC = LOCF(ZLVC  (1)) - IAD
        Y(2,IX) = ALOC
        Y(3,IX)  = HDA32(I)
        Y(4,IX)  = HDB32(I)
        Y(5,IX) = IUNIT(5)


        I = I + 1                     Counter for the number of variables
                                      in the file.
        IX = IX + 1                   Counter for the number of variables
                                      in a plot record.
        Y(1,IX) = FLAG32(I)           Minor edit mnemonic.
        LOC = LOCF(LVCVOL(1)) - IAD
        Y(2,IX) = ALOC                Offset of the variable in the file.

        Y(3,IX) = HDA32(I)            Heading for the variable for editin
        Y(4,IX) = HDB32(I)            purposes.
        Y(5,IX) = IUNIT(1)            Units of the variable for editing
        IX = IX + 1                   purposes.

                                      Information Needed for Each File

        IDXF = FILIDX(32)             FTB index for the file.
        Y(INDEX1,1) = AIDXF
        INDEX1 = INDEX1 + 1
        IFIL = 0
```

178 089

TABLE VI.1-2 (Cont'd)

Y(INDEX2,1) = AIFIL INDEX2 = INDEX2 + 1	Region deck indicator, 0 = Only one set in the file, -1 multiple sets sequential ordering > 0 = offset for region check for files with multiple sets that can have non-sequential region numbers.
Y(INDEX3,1) = JFIL(II) INDEX3 = INDEX3 + 1 LFIL = IX - IOLD	Hollerith defining which feature of the code variable is associated with.
Y(INDEX4,1) = ALFIL INDELX4 = INDEX4 + 1 KFIL = FILSIZ(32)/SETSIZ(32)	Length of a set of variables in the abbreviated file.
Y(INDEX5,1) = AKFIL INDEX5 = INDEX5 + 1 NFIL = 32	Number of sets of variables in the file.
Y(INDEX6,1) = ANFIL INDEX6 = INDEX6 + 1 MFIL = 0	Relocatable file number (>0), subfile if (<0), where major file is given by absolute value.
Y(INDEX7,1) = AMFIL INDEX7 = INDEX7 + 1 ANOVAR = Y(2,1) NOVAR = NOVAR - 1 Y(2,1) = ANOVAR	Variable indicating if this is a major file (=0) with no subfiles written to the data tape, if it is a major file with a subfile (=1) written to a data tape, or if it is a subfile (≠0). If a subfile, the top half of word contains the offset from beginning of the fully described major file to the index pointing to the subfile. The bottom half word contains the offset to the index for abbreviated major file.

1758 090

2.0 DATA TAPE GENERATION

RETRAN data tapes are generated by subroutine PLTAPE in program module RETRAN and by subroutines CPYPLT and PLTAPE in program module RESTRT. Subroutine PLTAPE writes header records at the beginning of each new tape volume, subsequent data records and a trailer record upon problem termination. PLTAPE also processes all tape mount, unload and return requests.

Subroutine CPYPLT is used in RETRAN problems to perform a reel to reel copy of all information on the data tape used for restart to a new data tape. The new data tape contains a header record and all data records to the point at which the original problem is restarted. All subsequent data records are written by subroutine PLTAPE. The first call to PLTAPE for a RESTRT problem bypasses the tape mount request and header record processing and immediately writes the data record for the restart time. All subsequent data record processing is identical to that performed for a RETRAN problem.

Table VI.2-1 illustrates the use of the DRD file and AFD subfile to generate a data record.

The outermost loop is over all files which may be included in a data record, the next loop is over all sets of a file, and the innermost loop is over all members of the AFD.

TABLE VI.2-1

TYPICAL CODING USED TO WRITE A DATA RECORD

```
      IDXPRC = FILIDX(43)
      LPRC = FILSIZ(43)
      NOFILS = ISTOR(IDXPRC)
      IDXADD = ISTOR(IDXPRC+01)
      NSET = ISTOR(IDXPRC+2)
      MAX = NSET*NOFILS
      IDXSCR = ISTOR(FILIDX(38))
      LSCRAT = FILSIZ(38)
C
C  WRITE ABREVIATED DATA RECORD FILES
      IDADR1 = IDXADD
      INDEX = IDXSCR - 1
      DO 80 I = 3,MAX,NSET
      ID = ISTOR(IDXPRC+I)
      ISETSZ = ISTOR(IDXPRC+I+1)
      IF (ID.GT.0) ISETSZ = SETSIZ(ID)
      NS = ISTOR(IDXPRC+I+2)
      IS = ISTOR(IDADR1)
      L = 0
      K = 0
      DO 70 J = 1,NS
      DO 60 M = 1,IS
      K = K + 1
   60 RSTOR(INDEX+K) = RSTOR(ISTOR(IDADR1+M)+L)
      ITEST = K+IS
      IF (J.LT.NS .AND. ITEST.LE.LSCRAT) GO TO 70
      ITEST = IS
      IS = K
      CALL BUFOUT (IUNIT,ISTOR(IDXSCR),IS,IOK,LAST,IPRU)
      IF (IOK.EQ.3) GO TO 30
      IF (IOK.NE.1 .OR. IS.NE.LAST) GO TO 440
      IS = ITEST
      K = 0
   70 L = L + ISETSZ
   80 IDADR1 = ISTOR(IDADR1) + IDADR1 + 1
```

1758 092

3.0 DATA TAPE USAGE

RETRAN data tapes provide the data interface between program module RETRAN and the RESTRT, REEDIT and PLOTER program modules. Use of a RETRAN data tape by all program modules is facilitated through use of a common pool of data tape processing subroutines. A summary of these data tape processing subroutines and their function is given in Table VI.3-1. Each of the subroutines listed in Table VI.3-1 utilizes the binary I/O environmental subcode BUFOUT (discussed in Section VII) to perform the required I/O operations.

The four program modules use RETRAN data tapes for various purposes. Consequently, several different FORTRAN unit numbers are used by the program modules. A summary of the FORTRAN unit numbers and the corresponding data set requirements is given for the four program modules in Table VI.3-2.

TABLE VI.3-1

DATA TAPE PROCESSING SUBROUTINE DESCRIPTIONS

Subroutine	Description
CHEK	Used by all program modules to mount a tape and to ensure that the requested data tape is a RETRAN data tape and that the first sixteen characters of the original and current problem titles match (title check not made for PLOTER problems).
CPYPLT	Used to generate a duplicate copy of the RETRAN data tape used to restart a problem. The copy operation is terminated at the last data record prior to the restart data record. The duplicate copy is then used to store new problem solution data and ensures a continuous RETRAN data set for each individual problem. This feature is used by program module RESTRT only.
INDATA	Used by program module RESTRT to retrieve the original problem input data from the header record of a RETRAN data set.
OVRLYP	Used to overlay abbreviated file data contained in a data record onto corresponding fully described files. This feature is used by program module RESTRT only.
PLTAPE	Used by program modules RETRAN and RESTRT to generate a RETRAN data set.
POSITN	Used by all program modules to position a RETRAN data set at a specified data record. The data set is positioned after the three word header prefix which is used to ensure proper positioning.
PULLIN	Used by program modules REEDIT and PLOTER to move a data record into the appropriate abbreviated files reserved by subroutine SETUPE. Subroutines EDITRE and REDTAP then retrieve specified information from these abbreviated files by use of the AFD subfile created by INEDIT.

TABLE VI.3-1 (Cont'd)

SETUPE	Used by program modules REEDIT and PLOTER to reserve space for the abbreviated files contained in a given data record structure. Subroutine SETUPE defines the space (main memory) into which subroutine PULLIN moves a data record.
TAPEBC	Used by program modules RETRAN and RESTRT to retrieve time dependent boundary conditions and power histories from a RETRAN data set.

TABLE VI.3-2

RETRAN DATA TAPE FORTRAN UNIT NUMBER CROSS REFERENCE[1]

Program Module	Unit Number	Description
RFTRAN	12	Read a previously generated RETRAN data tape which is used to supply time dependent thermodynamic boundary conditions for a volume(s) in the current problem and/or to supply the power history for the current problem. (READ Only)
	14	Unit upon which a RETRAN data tape is written if requested by input data. (WRITE Only)
RESTRT	12	Same as for RETRAN program module above.
	13	Read a previously generated RETRAN data tape. The data tape contains the information required to restart a RETRAN problem. (READ Only)
	14	Tape mounted on Unit 13 is copied onto Unit 14 out to (but not including) the data record containing the requested restart information. From the restart point on, data records are written as requested thru input data. (WRITE Only)
REEDIT	13	A RETRAN data set containing information for which printed edits and/or printer plots are desired is read on this Unit. (READ Only)
PLOTER	13	A RETRAN data set (or multiple data sets) containing information to be plotted is read on this unit. (READ Only)

1) CDC logical file names are TAPEXX and IBM DD names are FTXXF001, where XX corresponds to the Unit Number given above.

4.0 COMPATIBILITY WITH RELAP DATA TAPES

Due to the improved data format and storage philosophy adopted in the RETRAN Code Package, only RETRAN data tapes provide valid input to program modules RETRAN, RESTRT, and REEDIT. In other words, RELAP3 and RELAP4 plot-restart tapes are not compatible with the data format and content required by the three program modules noted above. However, an option has been added to the PLOTER program module which allows PLOTER to be used to plot RELAP3, RELAP4 and specially formulated "Stranger" data tapes, as well as RETRAN data tapes. Use of PLOTER program module for plotting RELAP and Stranger data tapes is possible only if:

(1) The RELAP3 or RELAP4 plot record structure has not been modified from the released version;
(2) The record structure for Stranger tapes (given in following Sections) is adhered to;
(3) The recording technique is consistent with that discussed for subprogram BUFOUT in Section VII.

The subroutines written to extend the capability of the PLOTER program module and allow use of RELAP and Stranger data tapes are summarized in Table VI.4-1. All I/O for the extensions to PLOTER is performed as a READ only operation on FORTRAN Unit number 13.

4.1 RETRAN Stranger Data Tape Format and Structure

RETRAN Stranger data tapes are binary data tapes generated by any program, subject to the format specifications given in Table VI.4-2. The data tape may be written using any FORTRAN BUFFER I/O program used consistently with the record and blocking structure discussed for subprogram BUFOUT in Section VII.

TABLE VI.4-1

EXTENDED PLOT TAPE PROCESSING SUBROUTINES

Subroutine	Description
CHEK	Used by PLOTER to determine the data tape type to set the tape type flag as follows: Type 1 RETRAN Data Tapes 2 RELAP4/003 Plot-restart Tapes 3 RELAP4/002 Plot-restart Tapes 4 RELAP3 Plot-restart Tapes 5 Stranger Data Tapes
POSPLT	Used by PLOTER to position Type 2, 3 and 4 tapes at a specified plot record. The data set is positioned after the single header word (4HPLOT).
PULLIN	Used by PLOTER to move a stranger tape record into the appropriate abbreviated files reserved by subroutine SETUPE. Note: Stranger tape formatting is set-oriented, as is RETRAN data tape format.
PULTRN	Used by PLOTER to read Type 2, 3 and 4 records which are array oriented, i.e., A(1) immediately precedes A(2), and transposes the data into a set oriented form as it is moved into the files reserved by subroutine SETUPO.
SETUPO	Used by PLOTER to reserve the abbreviated files required to store Type 2, 3 and 4 records.

TABLE VI.4-2

RETRAN STRANGER DATA TAPE FORMAT

RECORD 1: Must be at least 8 but not more than 16 (A8) HOLLERITH WORDS in length

First three words must be

WORD(1) = 8HRETRANbS

WORD(2) = 8HTRANGERb b = blank

WORD(3) = 8HDATAbTAP

WORD(4) = 8H
.
.
.
WORD(8) = 8H

} Available for tape documentation

WORD(9) = 8H
.
.
.
WORD(16) = 8H

} Optionally available for tape documentation

TABLE VI.4-2 (Cont'd)

RECORD 2: WORD(1) = Number of data files per plot record

WORD(2) = Set size of file number 1

WORD(3) = Number of sets in file number 1

.

.

.

.

WORD(N*NOFILS) = Set size of file number NOFILS

WORD(N*NOFILS+1) = Number of sets in file number NOFILS

Example of a File:

M(1) V(1) P(1) T(1) X(1) . . . . Set 1

M(2) V(2) P(2) T(2) X(2)

M(3) V(3) P(3) T(3) X(3)

M(4) V(4) P(4) T(4) X(4) . . . . Set 4

Set size = 5

A file may consist of 1 set of n parameters

8 100

TABLE VI.4-2 (Cont'd)

RECORD 3: Data file heading (A8)

(First word of axis label)

WORD(1) = 8HXXXXXXXX Heading for file Number 1

(or 4HXXXX)

.

.

.

.

WORD (NOFILS) = 8HXXXXXXXX Heading for file Number NOFILS

(or 4HXXXX)

The data file heading is a single Hollerith word generally describing the data file. A file of one set should have an 8 character heading, while multiple set files should use a 4 character heading. The remaining 4 characters will be added to the heading word automatically. For example, consider the previous example, where the file contains control volume data. A heading word of bVOL will be expanded to bVOLbbb4 for plot requests for volume 4 (set 4).

TABLE VI.4-2 (Cont'd)

RECORD 4: WORD(1) = 4HXXXX Plot request flag

WORD(2) = Position in set minus 1

WORD(3) = 8HXXXXXXXX 1st word of description

WORD(4) = 4HXXXX 2nd word of description

WORD(5) = 8HXXXXXXXX parameter units

.

.

.

.

.

and so on where each member of a set is described by a 5 word alphanumeric field as shown above.

NOTE: Plot request must be set to 4HTIMX for problem time (words 2, 7, 12, etc.)

8 102

TABLE VI.4-2 (Cont'd)

RECORDS 5 AND UP:

WORD(1) = 4HPLOT

WORD(2) = File 1 set 1

.

. .

. set N

WORD(NSETS(1)*SETSIZ(1) +1) = File 2 set 1

. .

. .

. .

. .

. set N

for all files in record

8 103

VII. ENVIRONMENTAL SUBCODE PACKAGES

The environmental subcode packages used in RETRAN are a collection of subroutines designed to facilitate data management, storage, and retrieval in an efficient manner.

Five subcode packages will be described in this section: the FTB package; the INP package; water property table interpolation routines; the plotting routines; and the BUFFER I/O routines. In addition, the Boolean algebra, shift and mask functions are described.

1.0 FTB

The FTB package is a set of subroutines for controlling dynamic storage allocation in central memory, bulk core, and disk data sets. The concepts utilized in the FTB code package are based on the INEL environmental code package[VII.1-1].

1.1 File Organization

The basic entity used by FTB in organizing data is a file whose entries normally have some internal relationship. Files may be further subdivided into sets which are generally that portion of a file which must be available to the program at any one time. The two types of files that may be defined are reserve files and process files. The principle difference between the two is the method of accessing the data in them. Reserve files may be accessed randomly and thus are required to reside in random access memory (RAM). Random access to disk is not permitted because of the high I/O overhead. A process file is processed one set at a time by calls on FTB and the program has access only to the current set. Process files can be processed sequentially or randomly. The sequential mode processes files sequentially starting with the first set. Process files can be stored in main memory or as disk data sets but only main memory files can be processed randomly.

Each file is identified by a unique number (ID) which must be positive or negative but not zero. All communication with FTB concerning the file is through this ID. FTB locates and assigns the requested space (if available) on the designated unit when the file is created. If the file is later deleted, the space is returned to the pool and becomes available for allocation to other files. Storage obtained for a reserve file is similar to a singly subscripted array in a DIMENSION statement except that the length can be sized for the particular problem being run. A set size must be defined for process files and storage is obtained as a multiple of the set size.

The main memory available for a problem is defined at execution time through an REL card for CDC or the REGION parameter for IBM. Excess storage on CDC is automatically released. The location of scratch memory begins at the end of the root segment and extends to the end of the requested region. Overlay segments reside within the scratch memory and active segments are protected from overwriting.

All remaining region space is available for data files on CDC systems. Characteristics of the linkage editor on IBM systems do not permit unused space within the overlay to be recovered. Thus data files on IBM systems are located at the end of the overlay.

The FTB routines use reserve areas in main memory for their internal bookkeeping. These reserve areas, called links, are 200 words long and contain the file descriptions for 50 files with each file description using four words. The first link is established during FTB initialization and the first description in the first link describes the link itself. Whenever a new file description would use the last (50th) description in a link, another link is established which uses the last description in the previous link for its description. The new file is described in the first description of the new link. The links are chained together through their descriptions and links are created and deleted as needed. Links use a file identification of 0.0 and user files cannot use 0.0 for a file ID.

A call to the RESERV subroutine describes a reserve file and obtains the required main memory storage. A call to DSCRIB describes a process file, but no storage for the file is obtained. The first call to OPEN in the write mode for a described file obtains storage space on the requested unit. Once the file has been written, the file may be opened for reading or rewriting using the same storage space. Process files use an extended description in addition to the standard description. For main memory, the extended description is two words long and precedes the file. The extended description is kept until the file is deleted. For disk files, the extended description is three words long and precedes the buffers used in reading or writing disk files. A contiguous region of main memory is used for the extended description and the buffers. This space is acquired when the file is opened and released when the file is closed. The sets in process files are processed by calls to PROC1, PROC2, PROC3, and PROC4. When all the sets have been processed, the file must be closed by a call to CLOSE. If not all the sets are processed, the file can be closed by a call to TRNCAT.

Storage in main memory is allocated in contiguous space as near as possible to one end of the space. For disks, space is always allocated toward the beginning of the available space. For core space, a parameter in the initialization call

indicates whether the beginning or the end of core is the preferred end of storage. Links, process files in main memory, extended descriptions and buffers are always positioned as near as possible toward the preferred end of main memory. Reserve files can be positioned toward either end of main memory. When using the FTB package, some foresight should be used to prevent excessive storage fragmentation especially in main memory. The SHIFT subroutine can be used to change the size of a reserve file and to reposition the file toward either end of main memory. Whenever a request for main memory storage is made, an attempt is first made to reposition links toward the preferred end of main memory and thus links can never fragment the main memory.

1.2 FTB Subroutines Calls and Functions

All subroutine and function arguments are of the integer type except for the file identification (ID) which is real. The array A used in the following descriptions is defined by

```
COMMON/FTB/A

REAL A(1)
```

and is positioned at the end of the root segment as described above.

The FTB package contains tests for I/O errors, invalid subroutine usage, and incorrect specification of subroutine call parameters. The list of possible error messages is given in Section VII.1.5. All errors cause abnormal termination of the load module.

1.2.1 CALL INITAL (LOWHI)

This call initializes the FTB package and must be the first call to the package. The INITAL subroutine performs the initialization discussed above. Checks are made as described and there must be at least 200 words of memory available to establish the first link. If an FTB subroutine is called before the call of INITAL, the program is abnormally terminated.

If LOWHI equals 1, the beginning of core storage is the preferred end, and if LOWHI equals 2, the high end of core storage is the preferred end.

1.2.2 CALL RESERV (ID, FILSIZ, LOHI, INDEX)

This call is used to define a reserve file with file identification ID and to reserve the storage space in fast core. The ID must be unique, that is, no file with that ID can already be described. The size of the reserve file is given by FILSIZ. If LOHI is 1, the reserved area is as near to the preferred end as possible, and if LOHI is 2, the reserved area is as near to the opposite end as possible. INDEX is returned such that A(INDEX) is the first location of the reserved area. The first argument, ID, is real, the rest integer.

1.2.3 CALL SHIFT (ID, FILSIZ, LOHI, INDEX)

This subroutine is used to move and/or change the length of a reserve file. FILSIZ may be larger than, smaller than, or equal to the original size and LOHI need not be the same as the original value. If LOHI is 1, the file is shifted as near as possible to the preferred end of main memory, and if LOHI is 2, the file is shifted as near as possible to the opposite end. If LOHI is 3, the origin of the file is not changed and only the size of the file is changed. INDEX is set such that A(INDEX) is the origin of the new storage area. When searching for an area large enough to hold the file, the space currently occupied by the file is considered available. Shifting a file does not destroy the contents of the file.

1.2.4 CALL DELETE (ID)

This call is used to delete all bookkeeping in the links and all storage associated with the reserve or process file with identification ID.

1.2.5 CALL DMPFIL (ID, FORMAT)

This subroutine is used to dump or print on Fortran Unit 6 the reserve file or a closed process file named ID. FORMAT is 0 for an octal (CDC) or hexadecimal (IBM) printout and 1 for floating point printout.

1.2.6 CALL DMPLST

This subroutine is used to dump bookkeeping information stored in the links. Once the INITAL subroutine has been successfully called, all errors in FTB usage call this routine before calling the abnormal termination routine.

1.2.7 LAVAIL (UNIT)

This function returns the number of locations available in UNIT if UNIT is main memory and the number of blocks available in UNIT if UNIT is a disk data set.

1.2.8 LCONTG (UNIT)

This function returns the length of the largest contiguous space currently available in UNIT. For disk data sets, the length is the product of the largest number of contiguous blocks available and the maximum number of words that can be stored in a block.

1.2.9 NEXTID (0)

This function returns the absolute value of the largest ID currently described or reserved plus 1 in floating point format. This function can be used to guarantee that unique file ID's are being used. The argument 0 is a dummy argument. NEXTID must appear in a REAL specification statement.

1.2.10 NFUNIT (ID)

This function returns the unit number of the file ID.

1.2.11 ISFDES (ID)

This function returns 1 if file ID has been described, 0 otherwise.

1.2.12 ISFOPN (ID)

This function returns 1 if file ID is open, 0 if closed.

1.2.13 CALL DSCRIB (ID, SETSIZ, NOSETS, UNIT)

This subroutine describes a process file which must be the first operation on a new process file. ID is the file identification which must be unique. SETSIZ is the set size and NOSETS is the number of sets. UNIT is 1 for fast core, 2 for bulk core and 3 through 7 for disk data sets (DDNAMEs FTB15F01 to FTB19F01 for IBM computers and FILE1 to FILE5 for CDC computers). The units 3 through 6 can have only one file open at a time while unit 7 has no such restriction.

1.2.14 CALL OPEN (ID, MODE)

This subroutine is used to open a process file. ID is the file identification and the file to be opened must be described. MODE is 1 for read, 2 for write, 3 for read-write, and 4 for random access. If MODE is 1, 2, or 3, the sets of the file must be processed sequentially by PROC calls. If MODE is 4, the unit specified in DSCRIB must have been 1 or 2 and the sets are processed randomly by GET and PUT calls described below. Storage is allocated to this file in UNIT if MODE is 2 or 4 and the file has not been opened before. It is an error if the first OPEN for a file is with MODE equal to 1 or 3. Rewriting a file is not an error and no additional storage is allocated. Random processing may be preceded or followed by sequential processing. When MODE is 1 or 3, OPEN initiates the first read if the file is a disk file.

1.2.15 CALL PROC1 (ID1, INDEX1)
CALL PROC2 (ID1, INDEX1, ID2, INDEX2)
CALL PROC3 (ID1, INDEX1, ID2, INDEX2, ID3, INDEX3)
CALL PROC4 (ID1, INDEX1, ID2, INDEX2, ID3, INDEX3, ID4, INDEX4)

These calls are used to process the next set of one, two, three, or four files which have been opened in read, write, or read-write mode. ID1, ID2, ID3, and ID4 are the file identifications. For the read mode, INDEX is set such that A(INDEX) is the location of the first word of the set. For the write mode, A(INDEX) is the first word of the location into which the program should place the next set. For read-write mode, A(INDEX) is the location of the first word of the set to be altered. The PROC routines handle the blocking and deblocking of sets into blocks, buffer switching, and the initiation and checking of read and

write operations. Overlapped I/O operations are used so that CPU processing can proceed concurrently with I/O operations. PROC1, PROC2, and PROC3 are entry points to the PROC4 subroutine.

1.2.16 CALL GET (ID, SETNO, LOC)

This subroutine is used to obtain set numbered SETNO from a file which has been opened in random mode. The set is moved from the file to location LOC.

1.2.17 CALL PUT (ID, SETNO, LOC)

This subroutine is used to store set numbered SETNO by moving information from location LOC into file ID which has been opened in random mode. The set must be in location LOC when the call is given. PUT is an entry point in the GET subroutine.

1.2.18 CALL CLOSE (ID)

This call is used to close file ID. A file opened in random mode may be closed at any time. A file opened in read, write, or read-write mode may not be closed by the CLOSE subroutine until all sets have been processed. When writing disk files, a flag is set such that the next request for fast core space forces the last write to completion.

1.2.19 CALL TRNCAT (ID)

This call also closes file ID, but not all sets need to be processed at the time of call. If the exact number of sets of a file to be written is not known at the time it is described, the file may be described with NOSETS equal to an upper bound and processed normally. When TRNCAT is called in the write mode, NOSETS is decreased to the number of sets actually processed and any extra storage is released as the file is closed. The deletion of storage for the unprocessed sets occurs during initial write or rewrite operations. If the file is open for reading, TRNCAT may be called to close the file before all the sets have been processed, but the remaining sets are not lost. TRNCAT may not be called for files being processed in read-write or random mode.

1.2.20 CALL CPYFIL (ID1, ID2)

This subroutine copies process file ID1 to process file ID2. ID2 must be described, the set size and number of sets of the two files must be the same, and ID1 cannot equal ID2. Both files cannot reside on the same disk unit and both files must be closed when CPYFIL is called.

1.2.21 NFSETS (ID)

This function returns the number of sets in file ID.

1.2.22 MXSETS (SETSIZ, UNIT)

This function returns the maximum number of sets of size SETSIZ which will fit into the space remaining on UNIT.

1.2.23 ISFPRC (ID)

This function returns 0 if file ID is a reserve fi[illegible] 1 if file ID is a process file.

1.3 File Description, Extended Description, and Record Formats

1.3.1 File Description Format

Files are described using four REAL words stored in special 200-word reserve files named "links" with ID's of 0.0. The file description format is given in Table VII.1-1. Link descriptions are chained together in forward and backward directions. Of the four words for a file description, on a CDC computer the first is treated as REAL and the other three are packed with two 30-bit integers. The IBM format is similar with the first word REAL*8 and the remaining words 32 bit integers. In the format description, A(I) is the first word of the description. For IBM Computers, the variable IA(N,K) accesses the upper half of A(K) if N is 1 and the lower half if N is 2. For CDC computers, IA is a function which returns the integer packed in the high order 30 bits of A(K) if N is 1, and the integer packed in the low order 30 bits if N is 2. The function IA is described in detail in Section VII.1.4.6.

TABLE VII.1-1

FTB FILE DESCRIPTION FORMAT

A(I)	File identification, 0.0 if file description for a link.
IA(1,I+1)	The length of a reserve file, the setsize of a process file, or the index of the last file in previous link if a link description of other than first link.
IA(2,I+1)	Zero if a reserve file and the number of sets if a process file.
IA(1,I+2)	Zero if reserve file or closed process file, the index of the extended description as a positive quantity if the file is open, or the index of the extended description as a negative quantity if the file is open for copying.
IA(2,I+2)	Unit of the file, 1 for main core, 2 for ECS or bulk core, and 3-7 for disk data sets.
IA(1,I+3)	Zero if no storage assigned; index of first storage location if reserve file or main core or bulk core process file; first block number if disk file. If a link description, this index is also the location of the next link.
IA(2,I+3)	Amount of storage used by file in words if in memory and in blocks if a disk data set.

1.3.2 Extended Description Format

The extended description is two REAL words long for fast core or bulk core process files with storage assigned and is three REAL words long for disk process files that are open. The extended array is pointed to by the index in IA(1,I+2) for the file description when the file is open. The extended description of a fast or bulk core file with storage assigned is also pointed to by IA(1,I+3) of the file description. In Table VII.1-2, A(J) is the first word of the extended description. All words of the extended description are treated as integers.

1.3.3 Block Formats

The first word of each block contains the file ID and the low order half of the second word contains the block number. Sets are blocked into the remainder of the block until the next set would exceed the blocksize. A full block is always written to or read from disk. The high order half of the second word and the last part of the block not used for sets are undefined and contain whatever was left from the previous use of those core locations.

1.4 Additional Subroutines Used by FTB Package

The following subroutines are part of the FTB package, but not directly called by the user: ERROR, IDFIND, LOCATE, SHFTLK, GETCOR, FTBIO, DMPER, FRECOR, and IA. A COMMON storage area named FTB is used for communication between FTB subroutines. The subroutines are coded in Fortran except for GETCOR, FTBIO, FRECOR, and IA which are coded in assembly language.

The FTB common block must be in the root segment of the load module.

1.4.1 CALL DMPER (M, B)

This subroutine, which is only used on CDC computers, interprets and edits the status information on direct access units. If M is nonzero, the device statistics are passed from GETDSK (entry point in GETCOR), via the B array and saved. If M = 0, the device statistics are edited.

8.1.15

TABLE VII.1-2

FTB EXTENDED FILE DESCRIPTION FORMAT

IA(1,J)	Mode of processing, 1 for read, 2 for write, 3 for read-write, and 4 for random processing.
IA(2,J)	Current set number.
IA(1,J+1)	Index of set.
IA(2,J+1)	Length of storage used for extended description and buffers. This and the following two integer words are not used for fast core or bulk core files.
IA(1,J+2)	Number of sets per block.
IA(2,J+2)	Block number.

1.4.2 CALL ERROR (ERR)

This subroutine is called if an FTB error is detected. The error number ERR is printed with a message.

1.4.3 CALL FRECOR

This subroutine reduces the job field length to the minimum required amount for CDC processors. It is inactive on IBM processors.

1.4.4 CALL FTBIO (UNIT, A, NREC)

This subroutine does the random I/O on direct access devices for FTB process files. The unit number must be between one and five for CDC and between fifteen and nineteen for IBM, A is the buffer address, and NREC is the record number to be moved.

1.4.5 CALL GETCOR

This subroutine requests the maximum field length available at the beginning of a job and initializes part of the FTB parameters. On CDC processors, the available field length is defined by the region size (if any) on the job card and by an RFL control card. An RFL control card is required to prevent automatic field length adjustment.

On IBM processors, memory can be allocated in both fast and bulk core if the latter is available. The length of fast core and the length and position of bulk core depends on the XREG and PARM fields on the EXEC card and the configuration of the operating system. The XREG specification has the form XREG=(aK,bK) and the PARM field required for FTB has the form PARM='c,d', where a, b, c, and d are integer decimal numbers. If HIARCHY support is generated into the operating system aK bytes of HIARCHY 0 storage and bK bytes of HIARCHY 1 storage are allocated to the job step. If b is zero or bK is missing, or if HIARCHY support is not present, no HIARCHY 1 storage is allocated to the step. If a nonzero bK is specified and HIARCHY support is not present, aK + bK bytes of fast storage are allocated. Conditional GETMAIN requests (for subpool 0) are issued for as much space as available in HIARCHYS 0 and 1. If HIARCHY storage 1 is obtained by GETMAIN, that storage becomes the bulk core. Of the HIARCHY 0 space obtained

by GETMAIN, c*1024 or cK bytes at the end of the space are returned by a FREEMAIN request to provide storage for other possible GETMAIN requests issued during execution of the load module such as LOAD and LINK macros and Fortran I/O buffers. If HIARCHY 1 storage is available, the remaining HIARCHY 0 storage becomes the fast core. If HIARCHY storage 1 is not available, the amount of HIARCHY 0 storage remaining is compared to dK; if the amount remaining is less than dK, that amount becomes fast core and the length of bulk core is zero. If the amount of HIARCHY storage 0 remaining is greater than dK, the length of fast core is dK and the amount greater than dK is used for bulk core. If a PARM field is not supplied, a default field of '8,400' is used. The FTB initialization subroutine edits on unit 6 the length of fast core and bulk core in units of eight byte words and the quantity INDEX, where A(INDEX) is the origin of bulk core. The fast core and bulk core storage areas are set to zero.

Disk units are also located and requested in this subroutine. On CDC systems, the status of the available disks is examined and the units having the most space are selected. The status information is saved for later editing. Logical file names used on CDC are FILEx where $1 \leq x \leq 5$. Logical records are fixed at 512 words each. Either NOS/BE or SCOPE 3.4 operating systems may be used provided a conditional assembly flag is properly set. Its value is 1 for SCOPE 3.4 and 3 for NOS/BE. The five disk data sets on IBM systems are designated FTB15F01 through FTB19F01. The minimum DD statement specifications for data sets used for FTB are the UNIT, SPACE, and DISP. UNIT must specify a direct access unit and the VOL. Channel and device separation specifications can be used. The SPACE parameter must be defined in terms of tracks or cylinders and only the primary specification is used. The SPACE parameter must be specified even if DISP=OLD since internal tables of FTB are based on SPACE information. A data set can be kept or preallocated data sets can be used, but files within FTB are scratch files and files cannot be saved on data sets past load module termination. The SPACE parameter can be varied to suit the requirements of each application.

Files are stored in data sets and transferred to and from fast core in blocks which contain an integral number of sets. If no blocksize information is entered in the DD statement, a default blocksize of one track of the assigned device truncated to an integral number of REAL*8 words is used. A blocksize of up to two tracks or the limit of 32760 bytes permitted by BSAM can be specified on the

DD statement. The specified blocksize is truncated to an integral number of REAL*8 words and if the blocksize is greater than one track, the track overflow indicator is set. Disk space is utilized in full tracks and each block starts at the beginning of a track. Even if the blocksize is specified as less than half a track, only one block is written in each track. It is recommended that the user either default to one track or specify two tracks.

Data sets on the first four units have the restriction that only one FTB file can be processed or open at any one time in order to minimize seek time. This restriction does not apply to the fifth unit.

1.4.6 Integer Function IA (J, N, ITEM)

This function, which is only used on CDC Computers, facilitates the conversion of environmental routines between IBM and CDC systems where mixed double precision and integer arrays are used. Four byte words are used in the IBM version for many variables and defined as follows:

```
REAL*8 A(1)
INTEGER IA (2,1)
EQUIVALANCE (A(1), IA(1,1)).
```

This coding on an IBM computer permits the packing of two four byte integer words into one eight byte real word by use of the array IA. To accomplish the same results on a CDC computer, IA is defined to be a FORTRAN Function instead of an array.

This function stores integer items as 30-bit words. J is an integer with a value of 1 if the item is in the high order 30 bits and 2 if in the low order 30 bits. N is the index (A(N)) where the item is located, and ITEM is a value to be stored. The argument, ITEM, is present only when storing an item.

1.4.7 CALL IDFIND (ID, I1)

This subroutine finds the location in the link tables of the FTB file ID. I1 is the index of the file description if one exists, otherwise I1 is zero.

1.4.8 CALL LOCATE (UNIT, SIZE, IDX)

This subroutine finds a contiguous space of length SIZE on device UNIT. Pointers are moved to preserve the space. The starting location is returned as IDX.

1.4.9 CALL REDUCE (LEN, LEM, IEDIT)

This subroutine, which is only used on CDC computers, releases unused core or (as an entry) expands the field length without destroying the FTB links. LEN is the new SCM field length if greater than 0. IEDIT is an edit flag, if 0, no edit of field length is given. LEM is the new LCM field length.

1.4.10 CALL SHFTLK

This subroutine attempts to shift the FTB links to the preferred end of core.

1.5 Error Messages

Most, but not all, FTB errors print an error message on Fortran Unit 6. All errors cause an abnormal termination.

The following messages can be issued during FTB initialization:

ARGUMENT ERROR TO FTB INITIALIZATION.

All other FTB error messages are given by

ERROR NUMBER xx FROM FTB PACKAGE.

followed by a list of the information in the links and then an abnormal termination. The error number given in the error message refers to the following error comments:

1 Another file is already open on disk unit.
2 INITIAL must be first call to FTB
4 Reserve file cannot be closed or truncated.

8 120

5 Files in read-write or random modes cannot be truncated.

6 File ID wrong on block just read, probably disk seek error or program error destroyed buffer.

7 Block number wrong on block just read, probably disk seek error or program error destroyed buffer.

8 File is a reserve file and shouldn't be.

9 File is already open.

11 File is not described.

12 File is not open.

13 File has not been written and has no storage assigned.

14 File already described.

15 Files in copy request on same disk unit.

16 File size is less than or equal to zero or is too large.

17 Incorrect format for DMPFIL.

19 File ID is zero.

20 File ID's for copy not unique.

22 Subroutine parameter for setting preferred end of fast core is incorrect.

23 Processing mode is random and unit is disk or PROC has been called.

24 Incorrect processing mode specified in subroutine argument.

25 Processing is not random and should be.

27 No space available for buffers in fast core.

29 No space available for file in disk data sets.

30 No space available for file in fast core.

31 Number of sets less than or equal to zero.

32 Not all sets processed on call to CLOSE.

34 Not enough space in fast core for links.

36 Set number out of range in call to GET or PUT.

37 Set size greater than that allowed by blocksize of disk data set.

38 Set size less than or equal to zero.

39 The number of sets in the files specified for a copy are different.

40 The set sizes for files specified for a copy are different.

41 Calls to PROC exceeds number of sets in file.

42 Unit number not between 1 and 7.

43 File is not a reserve file and should be.

47 Enough space is available in fast core, bulk core, or disk, but it is not contiguous space.

48 File size cannot be increased without moving file.

2.0 INP

The INP set of subroutines constitutes a convenient data input package for use with Fortran programs. The INP package is based on a similar Subroutine Package in the INEL Environmental library. To the user, the package offers: free form input; card numbers to identify data cards; automatic removal of cards containing duplicate card numbers; arbitrary ordering of input cards except for duplicate, continuation, and terminator cards; arbitrary use of comment cards and comments on data cards; ease of preparing cases in which only moderate changes are made from case to case; and a listing of the card data. For the programmer, the package is a convenient method for implementing a highly user-oriented data input scheme and includes: extensive checking of the amount and mode (integer, floating point, or alphanumeric) of data; automatic expansion of sequential and overlay type of input data; deletion of superfluous cards; checking whether extraneous input has been entered; and the ability to detect several errors during input checking.

2.1 User Aspects of INP Package

This section describes the INP package as seen by the program user.

2.1.1 Data Deck Organization

The data deck contains input for one or more problem sets. No relationship is assumed between problem sets. Each problem set consists of one or more cases in which the input data for cases other than the first consist of the data from the previous case plus modification cards entered for the present case. Input data for cases are separated by slash cards; the final case is terminated by a period card instead of a slash card. The period card also serves as the separator between problem sets. A slash card has a (/) as the first non-bank character on a card; a period card has a (.) as the first non-blank character. Comments may follow the slash and period on slash and period cards.

A list containing a card sequence number and the card image of each card is printed at the beginning of printed output for each case. The card sequence number starts at one for each case. The first line of the list contains "LISTING OF INPUT DATA FOR CASE n", where n is the case number.

2.1.2 Title Card

A title card is designated by an equal sign (=) as the first non-blank character on a card. The remainder of the card can have any alphanumeric characters. The information on the title card and the current date are printed at the top of every page following the input data listing. One title card should be entered for each case. If more than one title card is entered in a case, the contents of the last title card are used for the page heading. The heading contains only the date if no title card is entered for a case.

2.1.3 Comments Cards

An asterisk (*) or a dollar sign ($) appearing as the first non-blank character identifies the card as a comment card. Any information may be entered on the remainder of the card. Blank cards are treated as comment cards. There is no processing of comment cards other than listing them in the card list.

2.1.4 Data Cards

All cards other than title cards, comment cards, slash cards or period cards are considered data cards. The data cards contain a varying number of fields which may be decimal integer, decimal floating point, alphanumeric, octal, or hex. The rules for specifying fields are as follows.

Blanks preceding and following fields are ignored. A decimal field is started by either a digit (0 through 9), a sign (+ or -), or a decimal point (.). A comma or a blank (with one exception noted below) terminates the decimal field. The decimal field has a number part, and optionally an exponent part. A decimal field without a decimal point or an exponent is a decimal integer field. A field with either a decimal point or an exponent or both is a decimal floating point field. A decimal floating point field without a decimal point is assumed to have a decimal point immediately in front of the first digit. The exponent denotes the power of ten to be applied to the number part of the field. The exponent part is a sign, an E or D, or an E or D and a sign followed by a number giving the power of ten. Rules for decimal floating point numbers are identical to those for entering data in Fortran E or F formatted fields except that no blanks (one exception) are allowed between characters. Floating point data

punched by Fortran programs can be read. To permit this, a blank following an E or D denoting an exponent is treated as a plus sign. Acceptable ways of entering floating point numbers are illustrated by the following six fields all containing the quantity 12.45,

12.45, +12.45 1245+2 1.245+1, 1.245E1 1.245E+1

When entering a decimal zero for either an integer or floating point quantity, a zero can be written in either form. Thus, a floating point zero can be entered simply as 0 without a decimal point. A field starting with a non-blank character other than a digit, sign, comma, period or decimal point, asterisk, dollar sign, slash, or apostrophe is considered a default alphanumeric field. The field is terminated by a comma or the end of the card. All characters except commas are allowed and imbedded blanks are considered part of the alphanumeric field and do not terminate the field. Blanks extending from the last non-blank character of an alphanumeric field to the end of the card are not considered part of the field. An alphanumeric field can also be specified by enclosing the field within apostrophes (') for IBM and quotes (" for CDC. A blank or comma must follow the terminating apostrophe. The apostrophe field can be used to specify an alphanumeric field beginning with one of the special characters, e.g., '6% ENRICHED FUEL'. (Hex and octal fields permitted by CVI are treated as alphanumeric fields by INP. CVI is the environmental subroutine used by the INP package to process free format data on cards as discussed in Section VII.2.2.11.)

Data on a card may be continued on a continuation card by entering a plus sign as the first non-blank character on the continuation card. A field starting on a card must be completed on that card and may not continue to the next card. The plus sign indicating the continuation card is not considered part of the first data field on the continuation card and may be placed alone or adjacent to the first data field. Continuation cards themselves may be continued. In subsequent processing, data on continuation cards are treated as if the data were all entered on one card.

Comment information may follow the data fields on any data card (including cards that are continued) by preceding the comments with an asterisk or dollar sign. A default alphanumeric field preceding a comment must be terminated by a comma or the comment information is considered part of the alphanumeric field.

When card format errors are detected, lines containing a $ located under the character causing the error and a comment giving the card column of the error are printed. A field containing an error is converted as an alphanumeric field of $$$$$$$. An error flag is set and input processing continues, but the job can be aborted at the end of input processing. Usually another error is produced by a routine attempting to process the erroneous data.

The first field on a data card is treated as a card number which must be a positive decimal integer number. If the first field has an error or is not a positive decimal integer, the card number is replaced by the current card sequence number, an error statement is printed, and the error flag is set. Data on the card is not used and the card will be identified by the card sequence number if the list of unused data cards is printed. Continuation cards do not have card numbers since they are considered an extension of the first card. After each card number and the accompanying data are converted, the card number is compared to previously entered card numbers. If a matching card number is found, the data entered on the previous card is replaced by the data of the current card. If the card being processed contains only a card number, the card number and the data entered on the previous card are deleted. If a card causes replacement or deletion of data, a statement is printed indicating that the card is a replacement card.

The list of card numbers and associated data used in a case can be passed to the next case. Cards entered for the next case are added to the passed list or act as replacement cards depending on the card number. The resulting input to cases following the first case is the same as if all previous slash cards were removed from the input to the problem set.

2.2 Programming Use of the INP Package

The INP package contains fifteen subroutines or entry points which can be called by the programmer using the package. One call is issued to INP for each case in order to read and convert the data for the case, to replace or remove duplicate cards, and to form a sorted table of card numbers cross-referenced to a list. The list contains data words obtained from the cards and mode words generated during input conversion. Because of the sorted table, the order of cards in the data deck is not important. The subroutine LINK accesses the table and can

located one card at a time. The subroutine MODER is used to check the appropriate mode of the data against a specified list, also one card at a time. The INP2 subroutine is used to check data and to move data from the list to a specified array by using calls to LINK and MODER. It can be used to process data from a single card or from a set of cards numbered within a specified range and to check for minimum and maximum numbers of items and appropriate mode. The subroutine INP4 executes repeated calls to INP2 and modifies the call parameters to INP2 by specified amounts. Function INP8 can be used to determine whether there are cards in the list that have not been referenced by LINK and thus also INP2. The function INP9 deletes cards from the table and list which have been referenced by LINK and INP2. The function INP10 deletes selected cards from the table and list.

The following sections describe programming requirements for using the INP package. In the following descriptions, calling parameters are named for their integer or floating point format; that is, integer quantities have I, J, K, L, M, or N as their first character and names beginning with any other character are floating point quantities. Symbols appearing in the calling sequences are unique and when the same symbol appears in two or more calling sequences, the symbol has the same definition in each appearance. The symbol is usually completely described only in its first appearance, but the definitions are summarized in the Array and Variable Summaries (Section VII.2.3.2 and VII.2.3.3). Calling parameters that are marked with an asterisk both convey information to the subroutine and return information from the subroutine and thus the parameter can have a different value on exit than it had on entry. The error messages referenced in the descriptions are listed in the Error Message Summary Section VII.2.3.4. Programming errors such as improper calling parameters cause an abnormal termination by calling the FABEND[VII.1-1] routine.

Input data cards can have a mixture of integer, floating point, and alphanumeric data and the INP subroutine converts and stores all data into the list as words, which are defined to be sixty bit quantities for CDC and sixty-four bit quantities for IBM. The INP2 subroutine, after checking the data against a specified list for the correct mode, moves the data to a specified array if the move flag is set. The accessing of data from an array containing mixed integer and floating point data is easily accomplished by equivalencing integer and floating point names to the array containing the mixed data.

2.2 1 CALL INP (XL1, NL1, TITLE*, NCASE*, NDATA*, ISW*)

One call to INP reads all the input cards for the next case. That is, each call to INP reads cards from Fortran Unit 5 from its current position until either a slash card or a period card is read, or the end of the data set is encountered.

The quantity NCASE is incremented by one. NCASE should be set to zero before the first call to INP. If a period card terminates the case, the sign of NCASE is set to minus. The calling program can test the sign of NCASE to determine whether this case is the last case of a problem set or another case follows. If NCASE is negative indicating the end of a problem set, NCASE should be reset to zero before calling INP for the first case of the next problem set.

As input cards are read, they are printed without modification. Before printing the first card, the first heading line is printed at the top of a page and followed by "LISTING OF INPUT DATA FOR CASE n", where n is NCASE after it has been incremented by one. Subsequent pages of the input data listing have only the first heading line at the top of an output page. As title cards are read, the title information with the equal sign removed overlays the title storage array TITLE, which was initialized to blanks. TITLE should be at least 12 words in length. The editing note above is performed if ISW $\neq$ 0 upon entry to INP. If ISW = 0, no printing is performed.

The parameter XL1 is a REAL array of length NL1 and is used to store the list and table plus a control word. The control word is stored in XL1(1); the list is stored starting at XL1(2) and extends upward in the array; the table is stored starting at XL1(NL1) and extends downward in the array. The space between the list and table is used for temporary working space. Data card information is converted to binary form through calls to the DCVIC entry to the CVI subroutine. Binary information from a data card that is not a continuation card is stored starting at the beginning of the temporary work space and the mode indicators are stored beginning at the middle of the temporary work space. The binary information and mode indicators for continuation cards are stored following the information and mode indicators for the preceeding card. As each individual data card is processed, there must be 40 words between the end of the list or the last converted binary quantity and the beginning of the mode indicators. This space is necessary to prevent the binary quantities from overstoring the mode indicators and the mode indicators from overstoring the table; 40 words

are necessary since that is the largest number of quantities that can be entered on an 80 column card. After the data card and any continuation cards have been converted the binary data and mode indicators are stored as if the data were entered on one card and subsequent processing can proceed as if only one card was entered. A table word is then constructed, consisting of the card number or the card sequence number if the card number is illegal, an indicator specifying whether a card number or card sequence number is stored, an indicator specifying whether a card format error was detected, a pointer to where the binary information is stored, the number of words on the card other than the card number, and a use indicator which is set to off. If there are data other than the card number on the card, the binary information is moved downward one word eliminating the card number from the list, and the corresponding mode indicators returned from CVI are converted to two bit indicators and stored in a packed form, 30 indicators per word, following the binary information. The table words, one for each card, constitute the table, and the list is made up of the converted binary information and the mode indicators.

The new card number is then compared against the current card numbers stored in the table. If a duplication is found, the table word containing the card number is replaced by the new table word. Space occupied by the replaced list data is retrieved by shifting list data downward over the replaced data when the number of words on the replaced card and the new card are different or simply by overstoring the replaced list data with the new list data when the number of words is the same. Table pointers are updated when list data are moved. When a replacement card contains only a card number, the card acts as a deletion card; the table word is deleted and the list space is retrieved by shifting the list over the deleted data. A message, "CARD ABOVE IS REPLACEMENT CARD", is printed below any card that replaces or deletes a data card. A card containing only a card number that is not a duplicate card number has no effect on the table and list, and no message is printed.

A normal return from the INP subroutine is made if a period or slash card is read or an end of data set is encountered after at least one input card was read. Before a normal return, the table is sorted by card number and is moved adjacent to the list and the number of words in the list and the number of words in the table are packed into the control word at XL1(1). Upon normal exit, the absolute value of NDATA is set equal to the number of words needed in XL1 to

hold the control word, list, and table. The number of words needed is one plus the number of words in the list plus the number of words in the table. The sign of NDATA is set minus if no data cards (cards other than title, comments, slash or period cards) are entered for a case, and in normal usage this indicates that no input data were entered and that a succeeding case may have input identical to the preceding case.

On entry to INP, NDATA indicates whether the array XL1 contains data from a previous case. If NDATA is equal to or less than zero, XL1 contains no data from a previous case and the table and list are assumed empty. If NDATA is greater than zero, XL1 is assumed to contain data from a previous problem in the same format as that upon exit from INP. That is, XL1 contains a control word containing the number of words in the list and table followed by the list and table. The table is moved to the end of XL1 with the use indicators in the table set to off and the input cards for the current case are then processed as described above.

The parameter ISW controls the list edit and indicates the return status. If ISW is zero when INP is called, the card list edit is suppressed. The card list edit is provided if ISW is nonzero. If ISW is zero on return, a normal return was made and no errors were detected during the processing of the input cards. If ISW is one, the end of data set was encountered when trying to read the first data card of a case and INP returned immediately. This is the normal exit path if NCASE is not zero. If ISW is two, a normal return was made, but card format errors were detected and the list of input cards contains one or more of Error Messages 3 through 6. The usual practice in this case is to continue checking the input data for additional errors but execution is terminated after input checking is completed. If ISW is three, the array XL1 is not large enough to process the input data as indicated by Error Message 1 or 2, and INP returned immediately.

2.2.2 CALL INP2 (XL1, XL2, L3)

The array XL1 contains the list and table data. The array XL2 is the array into which the data specified in the call is to be moved. The array L3 contains specifications as follows:

L3(1) IC1, first card number.

L3(2) ±IC2, last card number. IC1 and IC2 specify the set of card numbers of the data that are to be moved into XL2. If IC2 is zero, only the card with card number IC1 is specified. If IC2 is nonzero, cards with card number c, IC1 $\leq$ c $\leq$ IC2, are requested. Not all the cards in the range of c need be present. If IC2 is positive, the card numbers that are present within the range must be sequential and are processed in sequential order beginning with IC1. If card numbers IC1, IC1+1, IC1+2 ..., IC1+a are present where IC1+a is the last sequential card number, a card with number c_x, IC1+a + 2 $\leq$ c_x $\leq$ IC2, is an error and causes Error Message 8 to be printed. If IC2 is negative, the cards need not be sequential and are processed in increasing order.

L3(3) MIN, the minimum number of items to be processed. Error Message 9 is printed if fewer items are processed.

L3(4) MAX, the maximum number of items to be processed; ignored if zero. Error Message 10 is printed if more items are processed.

L3(5) NJ, the number of words to skip between items in XL2; usually zero.

L3(6) ±J*. J if positive is the starting location in XL2; input item n is stored in XL2(J+(NJ+1)*(n-1)). If J is negative, no data is moved, but all checking is performed.

L3(7), L3(8), L3(9), ... An array defining the integer, floating point, or alphanumeric format expected on the cards. Format information is defined in MODER description. Errors cause Error Messages 7, 11, or 12 to be printed.

LINK is used to locate each card and MODER is used to check the format.

On exit, L3(6) or J is set to NMOVE, the number of items moved, if no errors were found and J was positive on entry. L3(6) or J can be zero indicating no cards with the specified card numbers are present if L3(3) or MIN is set to zero. If any card requested contained a card format error (as detected by INP) or if any of the tests specified in the L3 array failed, L3(6) or J is set to -1 on exit. If J was negative on entry, L3(6) or J is set to -NMOVE on exit if no errors were found. Only the first error in the specified set of cards is

found and processing is terminated after the appropriate error message is printed. Error Message 13 and 14 can be printed by INP2 in addition to those noted in the definition of L3.

2.2.3 CALL INP4 (IC1, ±IC2, MIN, MAX, NJ, J*, IC3, NTIMES, NEWJ, XL1, XL2, L5)

Subroutine INP4 makes NTIMES calls on INP2. An abnormal termination occurs if NTIMES is zero or negative. For the first call on INP2, IC1, IC2, MIN, MAX, NJ, and J are as described for INP2 and L5 is the format array for checking the mode of the data. For the following calls, IC1 and IC2 are increased by IC3, and J, if positive, is increased by NEWJ. J is changed upon exit as in the INP2 description. IC1 and IC2 on exit have the same value as on entry. Maximum size of the L5 array is 40.

2.2.4 CALL INP5 (IC1, ±IC2, IC3, ±N1, ±NMIN, ±NMAX, ±NSTORE, NTIMES, NEWJ, J*, XL1, XL2, L5, XL6, NL6)

The subroutine INP5 is similar to INP4 in that it makes NTIMES calls on INP2, but is more powerful in that it accepts self-expanding data of the sequential or overlay type. The basic unit of input data is a vector, $\underline{S}$, of N1 components where N1 is defined as an input argument. If N1 is positive, the data is of sequential form where $(\underline{S}_k, n_k)$, k = 1, ... K means that the vector $\underline{S}_k$ is to be repeated $n_k - n_{k-1}$ times, i.e., expanded into vectors $n_{k-1} + 1$ through n_k. The variable k is the number of vectors on the data cards with n_o = NMIN , $n_k > n_{k-1}$, and $n_k \leq$ NMAX .

The vectors, $\underline{S}_i$, form a two-dimensional array with elements, $s_{k,i}$, where $1 \leq k \leq$ N1 , NMIN $\leq i \leq$ NMAX . This array can be stored into XL2 in two different modes where one mode is the transpose of the other mode. If NSTORE is positive, $s_{k,i}$ is stored in XL2(J+(k-1) + NSTORE*(i-1)) and NSTORE should be greater than or equal to N1 . If NSTORE is negative, $s_{k,i}$ is stored in XL2(J+(i-1) + NSTORE *(k-1)) and the proper size of NSTORE depends on NMIN and NMAX. This is equivalent to having a two dimensional array defined as: DIMENSION SS(NSTORE,n) and EQUIVALENCE (XL2(J),SS(1,1)). If NSTORE is positive, $s_{k,i}$ is stored in SS(K,I), and if NSTORE is negative, $s_{k,i}$ is stored in SS(I,K).

As an example, let

```
N1     = 2
NMIN   = 0
NMAX   = 10
NSTORE = 10
NTIMES = 1
NEWJ   = 0
```

and let the vectors $\underline{S}_k$ be given by

$\underline{S}_1 = (1.0, 10.0)$, $\underline{S}_2 = (2.0, 20.0)$, $\underline{S}_3 = (3.0, 30.0)$,

and n_k be given as $n_1 = 2$, $n_2 = 4$, $n_3 = 10$. These data on a card could appear as follows:

```
xxxxxx   1.0,10.0,2   2.0,20.0,4   3.0,30.0,10
```

where xxxxxx is the card number. The expanded data would be stored in core as a two dimensional matrix SS(K,I) and would appear as:

```
 1.0   1.0   2.0   2.0   3.0   3.0   3.0   3.0   3.0   3.0
10.0  10.0  20.0  20.0  30.0  30.0  30.0  30.0  30.0  30.0
```

Now, let

```
N1     = 2
NMIN   = 0
NMAX   = 10
NSTORE = -10
NTIMES = 1
NEWJ   = 0
```

and use the same data as above. Since NSTORE is negative the expanded matrix is stored as the transpose, i.e., SS(I,K) and would appear in core as

1.0 10.0
1.0 10.0
2.0 20.0
2.0 20.0
3.0 30.0
3.0 30.0
3.0 30.0
3.0 30.0
3.0 30.0
3.0 30.0

If N1 is negative, the data is of the overlay form $(m_k, \underline{S}_k, n_k)$, k = 1, ...K where $\underline{S}_k$ is overlaid on an initial set of vectors beginning at the m_k'th vector and extending through the n_k'th vector, with $m_k \leq n_k$, min $m_k \geq$ NMIN , and max $n_k \leq$ NMAX . For either type there results a sequence of expanded input data of the form $\underline{S}_i$, $i_o \leq i \leq$ NMAX , where certain of the $\underline{S}_i$ may be missing in case of overlay expansion. Here $i_o =$ NMIN + 1 or NMIN for sequential and overlay types respectively. For overlay data, a positive NMIN requires that the lower limit be included while a negative NMIN can be used for negative indexing. For both types a positive NMAX is used to require that the upper limit be included while a negative NMAX only specifies an upper bound. The initial vectors being overlaid may be null since the complete matrix can be overlaid initially.

As an example of the overlay feature, let

N1 = -2
NMIN = 1
NMAX = 10
NSTORE = 2
NTIMES = 1
NEWJ = 0

and let the vectors $\underline{S}_k$ be given by

$\underline{S}_1 = (1.0, -1.0)$, $\underline{S}_2 = (2.0, -2.0)$, $\underline{S}_3 = (3.0, -3.0)$.

Assume a data card is given by

```
xxxxxx  1,1.0,-1.0,10  4,2.0,-2.0,8  5,3.0,-3.0,7 .
```

The first set overlays all ten vectors in SS(K,I) which then appears as

```
 1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0
-1.0  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
```

The second set of data on the card overlay vectors four through eight with the new vector $\underline{S}_2$. The matrix SS(K,I) is now given as

```
 1.0   1.0   1.0   2.0  2.0   2.0   2.0   2.0   1.0   1.0
-1.0  -1.0  -1.0  -2.0 -2.0  -2.0  -2.0  -2.0  -1.0  -1.0
```

The third set of data overlays vectors five through seven in SS(K,I) which then becomes

```
 1.0   1.0   1.0   2.0   3.0   3.0   3.0   2.0   1.0   1.0
-1.0  -1.0  -1.0  -2.0  -3.0  -3.0  -3.0  -2.0  -1.0  -1.0.
```

If INP5 was called as in the third example but with NSTORE = -10, the result would be the transpose of the last matrix above. Note that the value of NSTORE and N1 must be consistent with the matrix ordering.

IC1, IC2, IC3, NTIMES, J, NEWJ, and L5 are handled as in INP4. J on exit contains not the amount of expanded data, but the amount of data on the cards. The array XL6 of length NL6 is used for temporary storage and must be large enough to hold the unexpanded input data for one card set IC1 $\leq$ c $\leq$ IC2. Additional error checks are made because of the form of the input, and Error Messages 15 through 20 can be printed. Subroutine INP6 is called for error processing. If the parameters N1, NSTORE, or NTIMES are less than or equal to zero, an abnormal termination occurs.

2.2.5 CALL INP6 (IC1, IC2, ICARD, ITEM, XL1)

Subroutine INP6 can be entered when the program using the INP package finds that the N2'th item on a set of cards IC1 $\leq$ c $\leq$ IC2 processed by INP2 was in error. On exit, ICARD is the card number and ITEM is the number of the field on the card containing the error.

2.2.6 CALL INP7 (ICARD, ITEM)

Subroutine INP7 simply prints Error Message 22 stating that item ITEM on card ICARD is in error. This subroutine can be used to print error information obtained from INP6. This subroutine is not called by other INP package routines.

2.2.7 CALL LINK (IC, IX, N3, N4, XL1)

Subroutine LINK searches the table and list array, XL1, for card IC. The subroutine exits with IX equal to the card number in the table next larger than IC unless such a card does not exist and then IX equals -1. On exit, if N4 equals 0, card IC is not in the table; if N4 < 0, a format error detected by INP is on Card IC; and if N4>0, there are N4 data fields excluding the card number on card IC and the data are stored sequentially beginning at XL1(N3). A use flag is set on the table entry for IC if it is found. LINK issues no error messages.

2.2.8 CALL MODER (XL1, L3, N3, N4, N5, N6)

Subroutine MODER checks N4 items of data stored sequentially beginning at XL1(N3) for appropriate format. The format specification begins at L3(7). The specification starting at 7 is consistent with the format specification for INP2. The format entries are -1 for alphanumeric, 0 for integer, and 1 for floating point. Cyclic repetition for two or more entries can be condensed by prefixing the repeated format by ±N, where the magnitude is the number of items repeated, the sign is positive if the cycle is to be reset for each entry, and the sign is negative if it is to pick up at the stopping point of the previous entry. To allow starting within the format specification, N6 is the number of items previously checked with the current format. On exit, N5 = 0 if no errors were found; 0 < N5 < 10000 if item N5 should have been an integer but was not; N5 < 0 if item -N5 should have been a floating point number, but was not; and N5 > 10000 8 135

10000 if item N5 - 10000 should have been alphanumeric, but was not. A decimal zero is considered either integer or floating point as required to satisfy mode tests. MODER issues no error message. MODER calls INPUPK which is part of the INP package.

2.2.9 FUNCTION INP8 (NPRINT, XL1)

Function INP8 returns the number of cards that have not been processed by LINK. This is done by counting the number of table entries in XL1 that do not have the use flag set on. If NPRINT is 1, the card numbers of the unprocessed cards are listed under Error Message 21, while if NPRINT is 0, no output is printed. The list of unprocessed cards will include any cards that have an invalid card number; the card sequence number is printed in place of the card number and the sequence number is preceded by asterisks to distinguish it from a card number.

2.2.10 FUNCTION INP9 (XL1) and FUNCTION INP10 (XL1, IC1, IC2)

Functions INP9 and INP10 delete table entries and associated data and mode information from the array XL1. INP9 deletes cards that have been referenced at least once by LINK; that is, table entries with the use bit set on. INP10 deletes all cards, IC1 $\leq$ c $\leq$ IC2. When cards are deleted, all holes created by deletion are squeezed out, the table entries are adjusted accordingly, and the control word is updated. Both functions return the new length of the table, list and control word. When all cards have been deleted, the length required in the XL1 array is one--the length required for the control word.

When a card is deleted, a hole in the table and list is created, and the remaining tables and data must be moved to regain the storage made available. In order to move the remaining table and list only once and not use storage outside the array XL1, the holes are marked with the bit pattern 37777777777777777777 for CDC computers and 7FFFFFFFFFFFFFFF for IBM computers as they are formed. After all cards are deleted, holes are located by testing for the special bit pattern. Thus, the bit pattern used for marking holes must not be allowed as a data item. The CDC bit pattern corresponds to a floating point positive infinity. The IBM bit pattern corresponds to a floating point number of approximately 7×10^{75}.

2.2.11 CALL DCVIC (BCD, BIN, ICOND, NUM, NCH)

This subroutine converts a data card from a character format in the appropriate binary form. DCVIC is an entry point to the CVI subroutine[VII.1-1].

BCD Array containing card to be converted in 10A8 format.

BIN Array containing NUM converted items

ICOND Array containing code for converted items
- 0 a zero result (+0 and -0 allowed)
- 1 integer conversion
- 2 floating conversion
- 3 hex conversion
- 4 octal conversion
- -(character count) Hollerith conversion (uses more words in BIN if more than 8 characters)

NUM number of converted items

NCH 0 means no error
>0 column position of error

2.2.12 CALL INPPCK (WORD, NOITMS, ITEMS)
CALL INPUPK (WORD, NOITMS, ITEMS)

These subroutines pack (INPPCK) or unpack (INPUPK) the mode flags from DCVIC as 2-bit integers. These mode flags are compared to the mode flags passed to subroutine MODER by INP2 and a message is written on the output unit if any have a mismatch.

WORD packed information
NOITMS number of items to pack (unpack)
ITEMS list of items to pack (unpack).

2.2.13 CALL INPSHF(XL1)

This subroutine, used only on IBM systems, checks each data double word (8 bytes) in the XL1 list and table array. If the double word is found to contain an

integer (4 bytes), this integer is copied from the low order 4 bytes into the high order 4 bytes as well. This double copy of each integer entry in the INP list and table array is required on IBM systems for compatability between the INP subroutines and the integer padding performed by the AUTODBL option of the IBM H-Extended Fortran Compiler (see Section II.1.1.2).

2.3 INP Summary

Following are the INP package calls, summary of parameters, list of error messages, and structures of control word, table words, and mode words.

2.3.1 Summary of INP Package Calls

```
CALL INP (XL1, NL1, TITLE*, NCASE*, NDATA*, ISW)
CALL INP2 (XL1, XL2, L3)
     with L3(1) = IC1, L3(2) = ±IC2, L3(3) = MIN, L3(4) = MAX, L3(5) = NJ,
     L3(6) = J*, L3(7) and on equivalent to L5(1) and on.
CALL INP4 (IC1, ±IC2, MIN, MAX, NJ, J*, IC3, NTIMES, NEWJ, XL1,
     XL2, L5)
CALL INP5 (IC1, ±IC2, IC3, ±N1, ±NMIN, ±NMAX, ±NSTORE, NTIMES,
     NEWJ, J*,XL1, XL2, L5, XL6, NL6)
CALL LINK (IC, IX, N3, N4, XL1)
CALL MODER (XL1, L3, N3, N4, N5, N6)
CALL INP6 (IC1, IC2, N2, ICARD, ITEM, XL1)
CALL INP7 (ICARD, ITEM)
FUNCTION INP8 (NPRINT, XL1)
FUNCTION INP9 (XL1)
FUNCTION INP10 (XL1, IC1, IC2)
CALL DCVIC (BCD, BIN, ICOND, NUM, NCH)
CALL INPPCK (WORD, NOITMS, ITEMS)
CALL INPUPK (WORD, NOITMS, ITEMS)
CALL INPSHF (XL1)
```

8 138

2.3.2 Array Summary

L3 Array used for specifications to INP2.

L5 Equivalenced to L3(7). Array used to define appropri. e mode of data fields on card or set of cards excluding the card numbers. An entry of -1 is for alphanumeric fields, an entry of 0 is for integer fields, and an entry of 1 is for floating point fields. If a format repeats beyond a point, prefix the repeated format in L5 by $\pm N$, $N \geq 2$, where N is the number of items repeated. Use N positive to reset the repeat cycle at the beginning of each card, or use N negative to allow the cycle to overlap cards. Within a cycle, all elements must be 0 or ± 1, and only one cycle is allowed. Size of L5 array is 40. Array for INP2 starting at L3(7) is not limited.

XL1 Array containing control word, converted data from cards, mode indicators, and table entries.

XL2 Array into which data is to be moved.

XL6 Array for temporary storage used in INP5.

2.3.3 Variable Summary

BCD Array containing card to be conv..ted in 10A8 format.

BIN Array containing NUM converted items.

IC Card number desired.

ICOND Array containing code for converted items

0 a zero result (+0 and -0 allowed)

1 integer conversion

2 floating conversion

3 hex conversion

4 octal conversion

-(character count) Hollerith conversion (uses more words in BIN if more than 8 characters)

IC1, ±IC2 Define card numbers of a set, $IC1 \leq c \leq IC2$. If IC2 is zero, only one card, IC1, is requested. If IC2 > 0, card numbers must

be sequential, c = IC1, IC1 + 1, ..., IC1 + a $\leq$ IC2, and if IC1 + a is the last card in sequence, the next larger card c_x must not be in the range IC1 + a + 2 $\leq$ c $\leq$ IC2. If IC2 is negative, cards need not be sequential and are taken in increasing order. As used for INP10, all cards c, IC1 $\leq$ c $\leq$ IC2 are deleted.

IC3 — Added to IC1 and IC2 to define next set for use in INP4 and INP5.

ICARD — Number of card containing error item N2.

INP8 — Result of function call is the number of cards in table not processed by LINK, INP2, INP4, or INP5.

ITEM — Item number on CARD of item N2 on set of cards in error.

ITMS — List of items to pack (unpack).

IX — Card number in table next larger than IC. If no such card is present IX is returned as -1.

J — On entry, if J is positive, store data beginning at XL2(J); if J is negative, do not move data into XL2 but check data. On exit, J is set to -1 if an error was found; if positive on entry and no errors were found, it is set to +MOVE; if negative on entry and no errors were found, it is set to -NMOVE.

MAX — Maximum number of data items in a set of cards.

MIN — Minimum number of data items in a set of cards.

±N1 — N1 = number of elements in the input vector S; data is sequential type if N1 is positive and overlay type if N1 is negative.

N2 — The number of the data field in error in call to INP6.

N3 — On return from LINK, XL1(N3) contains first data word if card is present.

N4 Number of data words on card IC located by LINK if N4 is positive; if N4 is zero, card IC is not in table; if N4 is negative, format error was found on card IC.

N5 On exit from MODER, N5 = 0 if format correct; 0 < N5 < 10000 if item N5 should have been integer but was not; N5 < 0 if item -N5 should have been floating point, but was not; and N5 > 10000 if item N5-10000 should have been alphanumeric but was not.

N6 On entry to MODER, the number of previously checked items. This is used to located proper starting position in L5.

NCASE On entry, previous case number which should be non-negative and zero if the first case. On exit, is the current case number, negative if the last case of a problem set.

NCH 0 means no error.
>0 gives column position of error.

NDATA On entry, NDATA > 0 calls for adding data to previous list and table in XL1, and NDATA $\leq$ 0 indicates no previous list and table is present and the new data is complete in itself.

NEWJ Added to J for subsequent set in INP4 and INP5.

NJ Number of words to skip in XL2 between items. INP2 stores data item n in XL2(J + (NJ + 1)*(n-1)). Usually NJ is zero.

NL1 Size of XL1 on entry to INP.

NL6 Size of XL6; must be large enough to hold data from one set in call to INP5.

±NMAX Upper limit for sequential and overlay data; if NMAX is positive limit must be included.

±NMIN If sequential type, NMIN = n_o; if overlay type, min $m_k \geq$ NMIN, with NMIN positive requiring that the lower limit be included.

NOITMS Number of items to pack (unpack).

NPRINT If 1, list unprocessed card numbers, if zero, do not list them.

±NSTORE Used to control storage of data in INP5. See INP5 description (2.2.4)

NTIMES Number of sets of cards to process in call to INP4 or INP5.

NUM Number of converted items.

WORD Packed information.

2.3.4 Error Message Summary

All error comments are preceded by eight asterisks to facilitate recognition of error comments in the midst of regular program output. The lower case letters represent call parameters or symbols used in the subroutine descriptions and actual values are substituted in the output.

1. INSUFFICIENT STORAGE ALLOCATION FOR PREVIOUS DATA, PROCESSING TERMINATED.
2. INSUFFICIENT STORAGE FOR DATA, PROCESSING TERMINATED.
3. $ (placed under column in error) $ POINTS TO CARD ERROR AT COL. i
4. END OF FILE ENCOUNTERED BEFORE END(.) CARD.
5. CONTINUATION CARD INDICATED, BUT NO PREVIOUS DATA CARD. TREATED AS NEW DATA CARD.
6. UNRECOGNIZABLE CARD NUMBER
7. WORD n5 ON CARD ic SHOULD BE IN ALPHANUMERIC FORMAT
8. CARD c+a+1 MISSING IN SEQUENCE
9. TOO FEW NUMBERS ON CARDS ic1 THROUGH ic2
10. TOO MANY NUMBER ON CARDS ic1 THROUGH ic2
11. WORD n5 ON CARD ic SHOULD BE IN INTEGER FORMAT
12. WORD n5 ON CARD ic SHOULD BE IN FLOATING POINT FORMAT
13. CARDS ic1 THROUGH ic2 MISSING

14. ILLEGAL FORMAT ON CARD ic
15. m NUMBERS ON CARDS ic1 THROUGH ic2 ARE NOT A MULTIPLE OF n1
16. ITEM m ON CARD ic IS LESS THAN MINIMUM ALLOWED OF nmin
17. ITEM m ON CARD ic EXCEEDS MAXIMUM ALLOWED FOR nmax
18. ERROR IN LIMITS OF THE SET BEGINNING AT ITEM m ON CARD ic
19. LOWER LIMIT OF nmin NOT INCLUDED ON CARDS ic1 THROUGH ic2
20. UPPER LIMIT OF nmax NOT INCLUDED ON CARDS ic1 THROUGH 1c2
21. THE FOLLOWING CARDS WERE NOT USED
22. ITEM item ON CARD card IN ERROR

2.3.5 Word Structure Used in XL1

2.3.5.1 Control Word Structure

The control word stored in XL1(1) consists of two 30 bit (CDC) or 32 bit (IBM) integer words. Integer word 1 contains the length of the table; integer word 2 contains the length of the list containing the binary data and the mode indicators.

2.3.6 Table Word Structure

Bits are numbered from the right, starting with 0.

(CDC version)

Bit 59	Use flag; 0 if card not processed, 1 if processed
Bit 58-30	Card number or sequence number
Bit 29	0 if bits 58-30 contain card number, 1 if sequence number
Bit 28	0 if no format errors on card, 1 if errors
Bit 27-10	Index in XL1 of first data word of card associated with this table entry.
Bit 10-0	Number of words on card excluding card number.

(IBM Version)

Bit 0	Use flag; 0 if card not processed, 1 if processed.
Bits 1-29	Card number or sequence number.
Bit 30	0 if bits 0-29 contain card number, 1 if sequence number.

Bit 31	0 if no format errors on card, 1 if errors.
Bits 32-47	Index in XL1 of first data word of card associated with this table entry.
Bits 48-63	Number of words on card excluding card number.

Card numbers are limited to 536,870,911 (2^{29}-1) and card numbers greater than this cause Error Message 6 to be printed.

2.3.7 Mode Indicator Word Structure

For each card (continuation cards are considered as part of the first card), the mode indicator words are stored immediately following the last data word. The mode indicators are 0 for an integer or floating point zero, 1 for a nonzero integer, 2 for a nonzero floating point quantity, and 3 for an alphanumeric quantity.

3.0 WATER PROPERTY TABLE INTERPOLATION ROUTINES

Tables of steam and water properties were generated using the 1967 International Formulation Committee (IFC) formulation for industrial use of the properties of steam[VII.3-1,VII.3-2] as coded in the ASTEM package[VII.3-3] and written on a data set in the proper format for RETRAN. Properties stored in the tables as functions of pressure and temperature include specific volume, internal energy, coefficient of thermal expansion, isothermal compressibility, and the isopiestic heat capacity. The data spacing in the tables is variable permitting the density to be greatest where the most accuracy is required, but the tables must be regenerated to change the data distribution.

State properties in the data tables are in the SI units. The nomenclature used in this section and the units of the properties are:

T	Temperature	Kelvin (K)
p	Pressure	Pascal (Pa) = Newton/meter^2 (N/m^2) = Joule/meter^3 (J/m^3)
v	Specific volume	meter^3/kilogram (m^3/kg)
u	Specific internal energy	Joule/kilogram (J/kg)
h	Enthalpy	Joule/kilogram (J/kg)
$\beta = \frac{1}{v}\frac{\partial v}{\partial T}_p$	Coefficient of thermal expansion	Kelvin^{-1} (K^{-1})
$\kappa = \frac{1}{v}\frac{\partial v}{\partial P}_T$	Isothermal compressibility	Pascal^{-1} (Pa^{-1})
$c_p = \frac{\partial h}{\partial T}_p$	Isopiestic heat capacity	Joule/kilogram-Kelvin (J/kg-K)
f	Saturated liquid subscript	
g	Saturated vapor subscript	
sat	Saturated pressure subscript	

The data file contains five tables packed into a single-dimensioned array. The first two tables are temperatures and pressures obtained from input data; the third table contains saturation properties as a function of the saturation temperatures in the temperature table; the fourth table is a separate saturation table as a function of the saturation pressures in the pressure table; and the fifth table is a two-dimensional table containing the single-phase properties as

a function of the temperatures and pressures in the first two tables. The tables are generated and stored using full word floating-point quantities where a full word is a 64-bit or 60-bit item for IBM and CDC machines, respectively.

In the description of the tables, A is used for the array symbol, NT is the number of temperatures entered, and NP is the number of pressures entered into the table. NS is the number of input temperatures not above the critical temperature, and NS2 is the number of input pressures not above the critical pressure. The table storage is as follows:

(1) The temperatures in increasing order, as obtained from the input data, are stored in A(1) through A(NT). The temperature can be considered to be stored in any array dimensioned T(NT), where T(1) is equivalenced to A(1).

(2) The pressures in increasing order, as obtained from the input data, are stored in A(NT + 1) through A(NT + NP). The pressures can be considered to be stored in any array dimensioned P(NP), where P(1) is equivalenced to A(NT + 1).

(3) The saturation properties as a function of temperature are stored in A(NT + NP + 1) through A(NT + NP + NS*11). The saturation properties are stored as an array dimensioned B(11,NS), where B(1,1) is equivalenced to A(NT + NP + 1). The saturation values in B(N,1), $1 \leq N \leq 11$, are a function of the temperature in T(I). The correspondence between the B array and the saturation properties are:

B(1,I)	P	B(7,I)	v_g
B(2,I)	v_f	B(8,I)	u_g
B(3,I)	u_f	B(9,I)	β_g
B(4,I)	β_f	B(10,I)	κ_g
B(5,I)	κ_f	B(11,I)	c_{pg}
B(6,I)	c_{pf}		

(4) The saturation properties as a function of pressure are stored in A(NT + NP + NS*11 + 1) through A(NT + NP + NS*11 + NS2*11). The saturation values in C(N,J), $1 \leq N \leq 11$, are a function of the pressure

in P(J). The correspondence between the C array and the saturation properties is the same as for the B array except that C(1,J) is the saturation temperature instead of the saturation pressure.

(5) The single-phase properties as a function of temperature and pressure are stored in A(NT + NP + NS*11 + NS2*11 + 1) through A(NT + NP + NS*11 + NS2*11 + NT*NP*5). The single-phase properties are stored as an array dimensioned D(5,NT,NP), where D(1,1,1) is equivalenced to A(NT + NP + NS*11 + NS2*11 + 1). The values D(N,I,J), $1 \leq N \leq 5$, are a function of the temperature in T(I) and the pressure in P(J). The correspondence between the D values and the properties are:

D(1,I,J)	v	D(4,I,J)	κ
D(2,I,J)	u	D(5,I,J)	c_p
D(3,I,J)	β		

The data file containing the steam-water property data consists of two binary records. The first record contains NT, NP, NS, and NS2. The second record contains the A array which has NTOT elements, where NTOT = NT + NP + NS*11 + NT*NP*5.

The STH2ØI subroutine resides on the RETRAN source and is a special purpose subroutine used to retrieve the tables needed by the other STH2Ø subroutines. STH2ØI uses the first record of the state properties data set to initialize the STH2ØC common block. Additionally, STH2ØI reserves a block of central memory for the A array, and then moves the table from the state properties data set into the A array.

The water property file resides on a disk and is brought into RETRAN by calling subroutine STH2ØI. This call is of the form "CALL STH2ØI" and must precede any reference to the other water property subroutines. It need be executed only once unless the data array is destroyed. On CDC systems, the file has W-type records with I-type blocking and on IBM systems, the file has a VS record format with a maximum blocking of 800 bytes. On IBM systems, the water property file is read from DDNAME FT01F001 for disk data sets or FT11F001 for tape data sets. The corresponding file names for CDC systems are TAPE1 and TAPE11, respectively.

The integer variable, IT, is set by the STH2Ø subroutines that can compute properties in the liquid, two-phase, and vapor states. IT is set to 1 for the liquid state, 2 for the two-phase state, 3 for the vapor state below the critical temperature, and 4 for the state above the critical temperature. Water above the critical pressure, but below the critical temperature, is considered to be in the liquid state.

S is an array of twenty-three floating-point words containing both input to and output from the subroutines. The assignment of properties to the S array is:

S(1)	T	S(13)	u_f
S(2)	P	S(14)	u_g
S(3)	v	S(15)	h_f
S(4)	u	S(16)	h_g
S(5)	h	S(17)	β_f
S(6)	β	S(18)	β_g
S(7)	κ	S(19)	κ_f
S(8)	c_p	S(20)	κ_g
S(9)	X(quality)	S(21)	c_{pf}
S(10)	P_{sat}	S(22)	c_{pg}
S(11)	v_f	S(23)	indices
S(12)	v_g		

S(6) through S(8) are undefined if IT is returned as 2. S(9) contains 0.0 if IT is returned as 1, contains the quality if IT is returned as 2, and contains 1.0 if IT is returned as 3 or 4. S(10) is the saturation pressure corresponding to the temperature in S(1) if IT is returned as 1 through 3 and is undefined if IT is returned as 4. S(11) through S(22) are undefined if IT is not returned as 2. The S array is used for working storage and undefined elements may be changed during subroutine execution. On entry, the indices in S(23) are used to start the table search, if they are valid. On return, S(23) contains the indices obtained by the table search. Execution time can be minimized if the table indices returned are saved and used subsequently to start a table search. The subroutines do not fail when invalid indices are entered in S(23) because the table search will then start at the beginning of the table.

An error flag, ERR, is returned FALSE if the input quantities are within the range of the tables and TRUE otherwise.

The STH2Ø1, STH2Ø2, STH2Ø3, STH2Ø4, and STH2Ø5 subroutines all use similar table search and interpolation procedures. Linear interpolation is always used for liquid properties and linear or reciprocal interpolation is used for vapor properties based on the perfect gas relations, $PV = RT$, $u = C_v T$, $h = C_p T$, $\beta = T^{-1}$, and $\kappa = P^{-1}$. For example, since $v = RTP^{-1}$, linear interpolation is used for the temperature dependence and reciprocal interpolation is used for the pressure dependence. The linear interpolation is

$$Y = Y_1 + \frac{(Z - Z_1)(Y_2 - Y_1)}{(Z_2 - Z_1)} \quad , \qquad \text{(VII.3-1)}$$

and the reciprocal interpolation is

$$Y = Y_1 + \frac{(Z - Z_1) Z_2 (Y_2 - Y_1)}{(Z_2 - Z_1) Z} \quad . \qquad \text{(VII.3-2)}$$

The saturation pressure as a function of temperature is always computed using the K function of the IFC formulation. The STH2ØØ subroutine is just this evaluation. Enthalpy values are always obtained from $h = u + Pv$.

3.1 CALL STH2ØØ (T, P, ERR)

Subroutine STH2ØØ computes the saturation pressure, P, given the temperature, T, as input. The temperature must be in the range $273.16K \leq T \leq 647.30K$. This subroutine does not use the A array and can be called before STH2Ø1 is called.

3.2 CALL STH2Ø1 (A, S, ERR)

Subroutine STH2Ø1 computes the saturated water properities given temperature and quality as input. The temperature is entered in S(1) and must be greater than or equal to either T(1) or C(1,1) and less than or equal to either T(NS) or C(1,NS2). The quality, X, is entered in S(9) and must be in the range $0.0 \leq X \leq 1.0$. S(3) through S(5) return values for the two-phase mixture and S(11)

through S(22) return values for saturated liquid and saturated vapor. S(2) and S(10) are returned equal. IT would always be 2 for this call and thus is not included in the argument list.

For the STH2Ø1 subroutine, the saturation pressure is obtained from the K function. Both the saturated table as a function of temperature and the saturated table as a function of pressure are searched and the nearest bracketing values are used for interpolation. The saturated liquid properties are obtained by linear interpolation on temperature. The saturated vapor specific volume is obtained from

$$r = \frac{T - T_1}{T_2 - T_1} \quad , \quad v = (1 - r) \frac{P_1}{T_1} v_1 + r \frac{P_2}{T_2} v_2 \quad \frac{T}{P} \quad . \quad \text{(VII.3-3)}$$

Linear interpolation on temperature is used for saturated vapor internal energy. The saturated vapor thermal expansion is obtained from reciprocal interpolation on temperature; the saturated vapor compressibility is obtained from reciprocal interpolation on pressure; and the saturated vapor heat capacity is obtained from linear interpolation on temperature. The two phase values of specific volume, internal energy, and enthalpy are obtained from the saturated liquid and vapor values and the quality, e.g.,

$$v = (1 - x) v_f + x v_g \qquad \text{(VII.3-4)}$$

3.3 CALL STH2Ø2 (A, S, ERR)

Subroutine STH2Ø2 computes saturated water properties given pressure and quality as input. The pressure is entered in S(2) and must be greater than or equal to the triple point pressure (611.2 Pa). The saturation temperature is returned in S(1) and the other elements of S are set as in STH2Ø1. STH2Ø2 is an entry point in the STH2Ø1 subroutine.

For the STH2Ø2 subroutine, the saturation temperature is estimated from the input pressure, using equations given in SHARE Program 1095 converted to SI units. The equations are given below with the values in parentheses being the original values for temperature in deg.F and pressure in lb_f/in^2.

$$C_1 \leq P < C_2 \qquad\qquad C_2 \leq P \leq C_4 \qquad\qquad \text{(VII.3-5a,5b)}$$

$$T = \sum_{i=o}^{8} A_i \left[\ln(C_3 P)\right]^i \qquad T = \sum_{i=o}^{5} B_i \left[\ln(C_5 P)\right]^i \qquad \text{(VII.3-6a,6b)}$$

where

$A_0 = 274.9043833(35.157890)$ $\qquad$ $B_0 = 6669.352222(11545.164)$

$A_1 = 13.66254889(24.592588)$ $\qquad$ $B_1 = -4658.899(-8386.0182)$

$A_2 = 1.176781611(2.1182069)$ $\qquad$ $B_2 = 1376.536722(2477.7661)$

$A_3 = -.189693(-0.34144740)$ $\qquad$ $B_3 = -201.9126167(-363.44271)$

$A_4 = 8.74535666\times10^{-2}(0.15741642)$ $\qquad$ $B_4 = 14.82832111(26.690978)$

$A_5 = -1.7405325\times10^{-2}(-3.132958\times10^{-2})$ $\qquad$ $B_5 = -0.4337434056(-0.78073813)$

$A_6 = 2.147682333\times10^{-3}(3.8658282\times10^{-3})$ $\qquad$ $C_1 = 1378.951459(0.2)$

$A_7 = -1.383432444\times10^{-4}(-2.4901784\times10^{-4})$ $\qquad$ $C_2 = 3102640.782(450.)$

$A_8 = 3.800086611\times10^{-6}(6.8401559\times10^{-6})$ $\qquad$ $C_3 = 1.450377377\times10^{-3}(10.)$

$C_4 = 2.212\times10^{7}(3206.2)$ $\qquad$ $C_5 = 1.450377377\times10^{-4}(1.0)$

The estimated temperature is then used by STH2Ø1 to compute the saturation pressure. If the computed pressure does not converge to within 1.0×10^{-4} Pa of the input pressure, the reciprocal of the temperature derivative of the K function is used to predict the new temperature to be used in the next pass thru STH2Ø1. STH2Ø2 is merely an extension of STH2Ø1, allowing saturation pressure calls to use the K function, thus improving the consistency between pressure-quality and temperature-quality steam table entries. STH2Ø1 and STH2Ø2 use a common table search and interpolation procedure.

8 151

3.4 CALL STH2Ø3 (A, S, IT, ERR)

Subroutine STH2Ø3 computes single-phase water properties given temperature and pressure as input. The temperature, T, is entered in S(1) and must be within the range $T(1) \leq T \leq T(NT)$. The pressure, P, is entered in S(2) and must be within the range $0 < P \leq P(NP)$ for the vapor state and $P(1) \leq P \leq P(NP)$ for the liquid state. IT is never set to 2 because temperature and pressure cannot determine a two-phase condition. The single-phase quantities are returned in S(3) through S(8), and S(9) is set to either 0.0 or 1.0 corresponding to the liquid or vapor state, respectively.

For the STH2Ø3 subroutine, the temperature and pressure tables are searched to bracket the input temperature and pressure, and the bracketing temperature and pressure values and the corresponding values from the single phase table are used in a two-dimensional interpolation procedure. The two-dimensional interpolation procedure consists of two temperature interpolations, one for each bracketing pressure, followed by a pressure interpolation. When the temperatures and pressures bracket the saturation line, not all the single phase values are in the same phase. When this occurs, the appropriate saturation values are used in place of the single phase values that are in the wrong phase. Depending on how the temperatures and pressures bracket the saturation line, saturation values can be used in one or both temperature interpolations or may entirely replace one of the temperature interpolations. For determining whether the state is liquid or vapor, the input pressure is compared to the saturation pressure; the vapor state is assumed in the indeterminate case when the input pressure equals the saturation pressure. Linear interpolation on both temperature and pressure is used for the liquid state. For the vapor state, linear interpolation on temperature is used for all quantities except thermal expansion where reciprocal interpolation is used; for pressure interpolation, reciprocal interpolation is used for specific volume and compressibility and linear interpolation is used for the others.

When the input pressure is below the first pressure table entry and the temperature is such that the state is liquid, ERR is set TRUE. With the same pressure input, but with temperature such that the state is vapor, the water properties can be computed if the input temperature is bounded by the temperature table. For input temperatures above the saturation temperature corresponding to the

first pressure table entry, the internal energy, thermal expansion, and heat capacity are obtained from temperature interpolation using the first pressure values and the specific volume and compressibility are given by

$$r = \frac{T - T_1}{T_2 - T_1} \ , \ v = (1 - r) \frac{v_1}{T_1} + r \frac{v_2}{T_2} \quad \frac{P_1 T}{P} \ , \ \kappa = P^{-1} \quad . \ (VII.3\text{-}7)$$

Saturation values are used in place of the lower single phase values if the lower single phase values are in the liquid phase. For input temperatures below the saturation temperature corresponding to the first table pressure value, the saturation values corresponding to the input temperature are used for internal energy and heat capacity, the perfect gas laws $\beta = T^{-1}$ and $\kappa = P^{-1}$ are used for thermal expansion and compressibility, and the specific volume is computed as for saturated vapor conditions in STH2Ø1 except that the temperature and pressure are not saturation values.

3.5 CALL STH2Ø4 (A, S, IT, ERR)

Subroutine STH2Ø4 computes water properties given temperature and specific volume as input. The temperature, T, is entered in S(1) and must be within the range $T(1) \leq T \leq T(NT)$. The range of specific volume depends on the state. If the temperature and specific volume indicate the liquid state, the resultant pressure, P, must be within the range $P(1) \leq P \leq P(NP)$. If the temperature and specific volume indicate the superheated state, the resultant pressure must be less than P(NP). S(6) through S(8) are undefined if IT is returned as 2. S(9) contains 0.0 if IT is returned as 1, contains the quality if IT is returned as 2, and contains 1.0 if IT is returned as 3 or 4. S(10) is the saturation pressure corresponding to the temperature in S(1) if IT is returned as 1 through 3 and is undefined if IT is returned as 4. S(11) through S(22) are undefined if IT is not returned as 2. The S array is used for working storage and undefined elements may be changed during subroutine execution.

For STH2Ø4, the input specific volume is compared to the saturated liquid and vapor values corresponding to the input temperature to determine whether the state is liquid, two phase, or vapor. The interpolation procedure for two phase is similar to that for STH2Ø1 with the quality determined from

$$x = \frac{v - v_f}{v_g - v_f} \qquad \text{(VII.3-8)}$$

For single phase states, the temperature table is searched until the temperature is bracketed and then the specific volumes for the lower bracketing temperature are searched until the input specific volume is bracketed. Two temperature interpolations are made as in STH2Ø3, but because specific volume is not one of the table coordinates, the results of the temperature interpolations may not bracket the input specific volume. When this occurs, the table search is moved in the appropriate direction. When the temperature interpolations bracket the input specific volume, the specific volume interpolation is performed. The procedures for handling the saturation line, determining the state and type of interpolation and the handling of states with pressure below the first table pressure are similar to those in STH2Ø3 with specific volume and pressure exchanging roles.

3.6 CALL STH2Ø5 (A, S, IT, ERR)

Subroutine STH2Ø5 computes water properities given pressure and enthalpy as input. The pressure, P, is entered in S(2) and must be within the range $P(1) \leq P \leq P(NP)$. The enthalpy, h, is entered in S(5). The operation range of STH2Ø5 is further bounded by the temperature range $T(1) \leq T \leq T(NT)$. On return from an STH2Ø5 call, IT is appropriately set to reflect the phase conditions. S(6) through S(8) are undefined if IT is returned as 2. S(9) contains 0.0 if IT is returned as 1, contains the quality if IT is returned as 2, and contains 1.0 if IT is returned as 3 or 4. S(10) is the saturation pressure corresponding to the temperature in S(1) if IT is returned as 1 through 3 and is undefined if IT is returned as 4. S(11) through S(22) are undefined if IT is not returned as 2. The S array is used for working storage and undefined elements may be changed during subroutine execution.

For STH2Ø5, the input pressure is used with a quality of 0.5 in a call to STH2Ø2. If the input enthalpy lies between h_f and h_g (returned from STH2Ø2), the two-phase properties are computed, IT is set to 2 and a return made.

Single phase properties are computed with a temperature iteration and STH2Ø3, where the appropriate value of IT is determined using the input enthalpy and h_f and h_g returned from the initial call to STH2Ø2. The input pressure, the temperature estimate from the previous temperature iteration, and the phase index are then used in a call to STH2Ø3. If the enthalpy returned from STH2Ø3 is not converged to the input enthalpy a new temperature estimate is made and a new iteration performed. This iterative procedure is continued until

$$\frac{h_3 - h_I}{h_I} \leq \varepsilon$$

where

h_I = input enthalpy

h_3 = enthalpy returned from STH2Ø3

ε = convergence parameter.

If a convergent solution is not reached in 20 iterations, an error message is written and ERR set TRUE.

4.0 PLOTMC MULTIPLE CURVE PLOT PACKAGE

The PLOTMC subcode package was developed from the PLOTR4M program and is designed to provide a versatile environment in which graphic representation of problem solutions (or data) may be generated[VII.4-1]. PLOTMC is generalized to the extent that the user may select the length of the independent and dependent axes, choose between linear and logarithmic axes, specify the axes scaling or allow PLOTMC to perform scaling; overlay multiple curves on a single frame using one or more dependent variable axes to provide the curve scaling, and allow curve identifying symbols to be included on multiple curve frames. All axis labels are user supplied as is the title field which is placed at the top of each frame. PLOTMC is written in a manner which is consistent with the CalComp basic software for electromechanical plotters[VII.4-2]. In fact, PLOTMC contains all of the basic software required to generate graphics, with the exception of subroutine PLOT (and entry point PLOTS). A description of PLOT and PLOTS can be found in Reference[VII.4-2]. The PLOTMC subcode may be used on virtually any host computer to generate on-line or off-line plot data sets for any of the many electromechanical plotters available, by merely supplying the appropriate PLOT subroutine and its referenced subroutines at load time.

4.1 Programming Use of the PLOTMC Package

The PLOTMC subcode package consists of both FORTRAN and assembly language subroutines. A single call to the driver subroutine PLOTMC is required for each frame to be generated, which includes the independent axis and label, the first dependent axis and label, the title at the top of the frame and the first curve. Any additional dependent axes or curves which are to be added (overlayed) to the basic frame, require a separate call to PLOTMC for each curve or curve and axis to be added. Communication between the host plot program and PLOTMC is facilitated through use of the TOLP labeled common block and the PLOTMC argument list. Both the TOLP common block and the argument list must be initialized prior to each call to PLOTMC. The PLOTMC argument list is discussed in the following section.

The TOLP common block contains variables which are used to flag the plot function to be performed and to supply frame dimensions and scaling information. A description of the TOLP common block is given in Table VII.4-1. As noted in

TABLE VII.4-1

TOLP COMMON BLOCK DESCRIPTION

Variable	Type	Description
OFLAG	LOGICAL	LOGICAL flag used to determine whether a complete frame including all axis grids, axis labels, title and curve is to be generated (.TRUE.), or if only a curve is to be overlayed on an existing frame, which may include addition of a new dependent axis (.FALSE.).
ISYMB	INTEGER	Code used to add curve identifying symbols if ISYMB is greater than unity (see Figure VII.4-1, ISYMB=FLAG+1).
XPLTBK(1)	REAL	Ten word array used for communication between PLOTMC subroutines. Independent axis origin or starting value which will appear as the first annotation on the axis.
XPLTBK(2)		Independent axis scaling factor.
XPLTBK(3)		Dependent axis origin or starting value which will appear as the first annotation on the axis.
XPLTBK(4)		Dependent axis scaling factor.
XPLTBK(5)		Independent axis length (inches).
XPLTBK(6)		Dependent axis length (inches).

TABLE VII -1 (Cont-d)

XPLTBK(7)		Axes type flag 1.0 = LIN IND AXIS AND LIN DEP AXIS 2.0 = LIN IND AXIS AND LOG DEP AXIS 3.0 = LOG IND AXIS AND LIN DEP AXIS 4.0 = LOG IND AXIS AND LOG DEP AXIS
XPLTBK(8)		Reciprocal scale factor if independent axis is logarithmic.
XPLTBK(9)		Reciprocal scale factor if dependent axis is logarithmic.
XPLTBK(10)		Flag used to bypass scaling for multiple curve overlays using common scaling factors. < 1.0 DO SCALING ≥ 1.0 BYPASS SCALING
LIMS(1)	REAL	Four word array used to scale curve. Minimum value of independent variable (-1.E75 indicates automatic scaling).
LIMS(2)		Maximum value of independent variable (1.E75 indicates automatic scaling).
LIMS(3)		Minimum value of dependent variable (-1.E75 indicates automatic scaling).
LIMS(4)		Maximum value of dependent variable (1.E75 indicates automatic scaling).

158 158

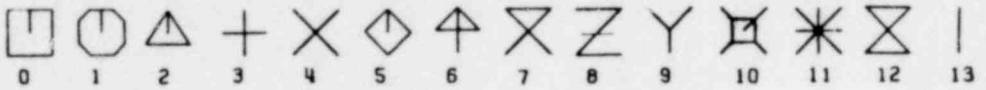

Figure VII.4-1 Centered Plot Symbols

8 159

Table VII.4-1, the variables OFLAG, ISYMB, and the LIMS array must be defined in the host program prior to each call to PLOTMC, while the XPLTBK array is used for communication between the subroutines in the PLOTMC subcode. Prior to the first call to PLOTMC, the plot output device must be opened via a call to PLOTS. The plot origin should also be set before calling PLOTMC.

4.1.1 CALL PLOTMC (NP,XX,YY,SIZ,LOG,NX,LBLX,NY,LBLY,NT,TITL)

A single call to PLOTMC is required to generate the base frame and first curve. Each additional curve or dependent axis and curve which may be added to the base frame requires an additional call to PLOTMC.

The integer argument NP is used to specify the number of independent-dependent variable pairs to be used in generating a curve. Both of the floating point arrays, XX and YY, contain NP values of the independent and dependent variables, respectively. The two-word floating point array SIZ is used to specify the length of the dependent and independent axes in inches. SIZ(1) is used to pass the requested length for the independent axis to the PLOTMC package while SIZ(2) is similarly used for the dependent axis.

The two-word LOG integer array is a multiple use array which is defined in the calling subroutine to specify axes types; i.e., linear or logarithmic, and to control scaling and frame advancement for overlaying curves. Interpretation of the LOG multiple use array is found below.

LOG(1)	<	0	Linear independent axis
	=	0	Error, skip this request
	>	0	Logarithmic independent axis
LOG(2)	<	0	Linear dependent axis
	>	0	Logarithmic dependent axis
LOG(1)	=	1	Advance frame after this curve request is complete
	>	1	Do not advance frame

1758 160

LOG(2)	=	1	Calculate new scale factors and draw new dependent axis
	>	1	Allow room for multiple dependent axes (1st curve)
			Use previous scale factor and axis after the base frame is generated (after 1st curve)

The Hollerith character arrays used to supply axis labels and titles to the PLOTMC subcode consist of ten and eight character words for CDC and IBM computer applications, respectively. Integer words NX, NY, and NT are used to specify the respective number of characters in the independent axis label, the dependent axis label and the problem title. The axis labels and title Hollerith strings are passed to PLOTMC via the independent axis array LBLX, the dependent axis array LBLY and the TITL title array.

4.1.2 CALL XPLOT (XX,YY,NP,KK,JJ)

Subroutine XPLOT is called by PLOTMC and is designed to request axis scaling if required, draw the curve, and to add the optional curve identifying symbols if specified. The values of the dependent and independent variables are contained in the two floating point arrays XX and YY, respectively. Both arrays contain NP*KK values which are used in pairs, i.e., a dependent variable value and the corresponding independent variable, to generate a curve consisting of line segments drawn between the data points. The integer KK is the skip count used to define the DO loop increment used in drawing the curve. NP is generally set to the total number of points in the dependent and independent variable arrays, in which case the skip count KK is set to unity. JJ is not currently used.

4.1.3 CALL XPLOTA (AFG,NX,LBLX,NY,LBLY,NT,TITL)

The functions performed by XPLOTA include drawing the axes and grids, annotating the axes, and writing the title. XPLOTA is called directly by PLOTMC to generate a base frame and subsequently called for each dependent axis added to the base frame. Location of the dependent axis to be drawn is specified by the value of the floating point variable AFG. For the first call to XPLOTA for a given frame, AFG is zero. On subsequent calls to XPLOTA, used to add an additional dependent axis to the left of the previous axis, AFG specifies the location at which the new axis is to be drawn. Arrays LBLX, LBLY and TITL contain the Hollerith character strings which are to be used as the independent and

dependent axis labels and the plot titles, respectively. The number of characters in the independent axis, dependent axis and problem title character strings are specified through NX,NY and NT.

4.1.4 CALL PSCALE (L,KK,NP,ARG,SIZE,ORG,DELTA,XYLOG,CYCL)

Subroutine PSCALE is an extended version of the SCALE subroutine, designed to perform scaling for linear or logarithmic axes (SCALE is for linear only). The integer flag L is used in conjunction with the LIMS array in the TOLP common block to determine whether the user has specified the range of a particular axis or whether the range is to be computed by PSCALE. L is either specified as 1 or 2 for the independent or dependent axes, respectively. The product of the skip indicator (KK) and the number of points to be plotted (NP) gives the number of dependent variables or independent variables in the array ARG. ARG contains independent variables when L=1 and dependent variables when L=2. The skip indicator is used as a DO loop increment when examining the KK*NP data items, starting at the first item. The length of the axis is specified in inches by the variable SIZE. The starting value of the axis is returned from PSCALE in ORG and the scaling factor computed by PSCALE is returned in DELTA. The input variable XYLOG is used to determine if logarithmic scaling is to be done (see XPLTBK(7) in Table VII.4-1). For cases where logarithmic scaling is performed, CYCL is returned as the inverse scale factor.

4.1.5 CALL AXIS (X,Y,LABEL,NC,SIZE,ANGLE,ORG,DELTA,N)

The AXIS subroutine is used to draw the axis lines, one inch grid marks, grid annotations, and axis labels for linear axes. Separate calls to AXIS are required for each dependent and independent axis drawn. The coordinates of the axis line's starting point are given by X and Y. LABEL contains the NC character axis label Hollerith character string, which is centered and placed parallel to the axis line. The sign of NC determines on which side of the axis line the scale grid lines and labeling information are drawn. If the sign of NC is positive, all annotation appears on the counterclockwise side of the axis line (normally desired for the dependent axis), while all annotation appears on the clockwise side of the axis line for negative values of NC (normally desired for the independent axis). Axis length and orientation are specified by SIZE, which is the axis length in inches, and ANGLE, which is the angle in degrees (positive or negative).

The starting value of the axis, which will appear at the first tick mark on the axis, is given by ORG. DELTA represents the number of data units per inch of axis and is used to increment the value of ORG to obtain the annotations for subsequent tick marks. Control of the precision of the numbers annotated with the axis grids is obtained by use of the integer N. For a description of N, refer to the discussion given in Section VII.4.1.7 describing the NUMBER subroutine.

4.1.6 CALL LAXIS (X,Y,LABEL,NC,SIZE,ANGLE,ORG,DELTA,N)

The LAXIS subroutine is similar to the AXIS subroutine, except LAXIS is used to generate logarithmic axes and AXIS draws linear axes. Use of the LAXIS subroutine is identical to the AXIS subroutine discussed in the previous section, with the exception of the integer N. For LAXIS, N is used as a flag to turn the annotation and label drawing feature on or off. If N is non-zero, the annotations and label will be added to the axis and grid, while only the axis and grid are drawn if N is zero.

4.1.7 CALL NUMBER (X,Y,HEIGHT,F,THETA,N)

The NUMBER subroutine converts a floating-point number to the appropriate decimal equivalent so that the number may be plotted in fixed point format or scientific notation. X and Y provide the coordinates of the lower left-hand corner of the first character to be produced. The pen is up while moving to this point. Annotation may be continued following the location at which the last annotation ended. Continuation occurs when X and/or Y equals 999.0, and may be applied to X and Y independently. HEIGHT is the height in inches of the character to be plotted. For best results, HEIGHT should be a multiple of seven times the plotter increment size, although other values are acceptable. The width of a character, including spacing, is typically the same as the height. The floating point number to be converted and plotted is contained in F. THETA is the angle at which the characters are to be plotted. N controls the precision of the conversion of the floating point number F. If the value of N is greater than zero, it specifies the number of digits to the right of the decimal point that are to be converted and annotated after rounding. For example, the floating point number -1.33499 x 10' will be plotted as -13.350 if N=3. When N=0, only the numbers integer portion and decimal point are plotted after rounding (-13.),

while only the integer portion is plotted if N=-1 (-13). A value of N $\leq$ -2 indicates that the annotation is to be in scientific notation and that there will be N characters to the right of the decimal point. N=-2 will result in -1.33E+01 being plotted in the above example.

4.1.8 CALL NORMAL (X,Y,HEIGHT,F,THETA,M)

Subroutine NORMAL is called by NUMBER to draw floating point numbers in scientific notation. NORMAL may also be called directly to draw floating point numbers in scientific notation. Use of NORMAL is identical to the use of NUMBER discussed above, with the exception of the last argument, M. M must be a positive integer ranging from one to five inclusively, which specifies the number of digits to be annotated to the right of the decimal point. For the example discussed above for subroutine NUMBER, where N=-2; NUMBER merely sets M=-N in the NORMAL argument list (passing all other arguments to NORMAL as input to NUMBER) and then calls NORMAL and returns upon return from NORMAL.

4.1.9 CALL SYMBOL (X,Y,HEIGHT,IBCD,THETA,NC)

The SYMBOL subroutine produces plot annotation by use of either of two calling formats. The standard calling format is used to draw text such as titles and legends, while the special call is used to draw special centered symbols such as a box, octagon or triangle for plotting data points. Refer to Figure VII.4-1 for the list of symbols plotted when using the special call. For both the standard and special calls to SYMBOL, HEIGHT is the height of the character or characters to be plotted and THETA is the angle at which the character or characters are to be drawn. The height of the characters should be a multiple of seven times the plotter increment size. Heights which are not a multiple of seven times the increment size are acceptable, but best results will generally be achieved if the rule is followed.

The standard call to SYMBOL is made by using a positive value of NC which then specifies the number of characters contained in the IBCD array that are to be annotated. If NC is greater than zero, characters to be plotted must be left-justified in the first element of IBCD. If NC is zero, one character is drawn, using a single character which is right-justified in the first element of IBCD.

For all standard calls, where NC is greater than or equal to zero, X and Y are the coordinates, in inches, of the lower left-hand corner of the first character to be drawn.

The special calls to SYMBOL are determined by a negative value of NC. If NC=-1, the pin is up during a move to coordinates X and Y, after which a single symbol is produced. For values of NC=-2, the pen is down during the move to coordinates X and Y, where a single symbol is produced. The X and Y coordinates represent the geometric center of the character to be produced on a special call. IBCD is an integer flag used to specify the symbol to be plotted on a special call. Figure VII.4-1 illustrates the special symbols which may be drawn and their corresponding flag (IBCD) values.

5.0 BIT MANIPULATION FUNCTIONS

Quite often it is necessary to modify the content of a word in storage at the bit level, to merge several words into a single word or to modify character strings. To perform such manipulations, it is necessary to have the capability to make logical sums and products, as well as logical and arithmetic shifts. These functions are not available in the ANS FORTRAN IV programming language. A set of FORTRAN callable assembly language functions which provide bit manipulation capabilities are described in the following sections.

All of the real bit manipulation functions discussed in the following sections must be declared as real functions in calling subroutines, if their default type is not real. The real specification applies to CDC and IBM applications where the Automatic Precision Increase facility of the H-extended compiler is used. If the API feature of the H-extended compiler is not used for the calling subroutines, the real functions must be typed as REAL*8.

5.1 Logical Sum

The logical sum (OR) of A and B is defined as follows. A bit position in the result is set to one if the corresponding bit position of one or both operands A and B are one; otherwise, the result bit is set to zero. An example of the logical sum of two 32 bit words is:

```
    A   01000110 00110011 11111111 10101010
    B   00000000 11100011 00000111 01010100
  Sum   01000110 11110011 11111111 11111110
```

The extension of the above example to words of a different length is obvious.

5.1.1 I = IOR(A,B)

The IOR integer function returns the logical sum of the two integer words A and B as an integer result (returned in general register 0 for IBM and register X6 for CDC). For IBM applications the integer function assumes 32 bit integers, while integers are 60 bits in length on CDC computers.

5.1.2 DA = DOR(DB,DC)

The DOR real function returns the logical sum of the two real words DB and DC as a real result (returned in floating register 0 for IBM and X6 for CDC). The arguments DB and DC, as well as the result are 64 bit quantities (double precision) on IBM computers and 60 bit quantities (single precision) on CDC computers.

5.2 Logical Product

The logical product (AND) of A and B is defined as follows: A bit position in the result is set to one if both of the corresponding bit positions of A and B are one; otherwise, the result bit is set to zero. An example of the logical product of two 8 byte words is:

A	11110000 11111111 00000000 10101010 01010101 11001100 00110011 11100010
B	10101010 01010101 11110000 11110000 00001111 11111111 00000000 01010101
Sum	10100000 01010101 00000000 10100000 00000101 11001100 00000000 01000000

5.2.1 I = IAND(A,B)

The IAND integer function returns the logical product of the two integer words A and B as an integer result (returned in general register 0 for IBM and X6 for CDC). The integers are 32 bits on IBM and 60 bits on CDC.

5.2.2 DA = DAND(DB,DC)

The DAND real function returns the logical product of the two real words DB and DC as a real result (returned in floating register 0 for IBM and X6 for CDC). The arguments DB and DC, as well as the result are 64 bit quantities (double precision) on IBM computers and 60 bit quantities (single precision) on CDC computers.

5.3 Logical Left Shift

The logical left shift works slightly differently on IBM and CDC computers. In the case of logical left shifts on IBM computers, the N high order bits are lost (shifted off the end of the register) and N low order bits are zeroed for a

logical left shift of N bits. Two examples of 15 bit logical left shifts of 32 bit words for IBM systems are as follows.

```
Before    01010101 01010101 01010101 01010101
After     10101010 10101010 10000000 00000000

Before    11111111 11111111 11111111 11111111
After     11111111 11111111 10000000 00000000
```

Logical left shifts on CDC computers are not "end-off" as in the case of IBM computers, but rather "end around" shifts. In other words, bits shifted off the high end of the register are transfered to the low end of the register as if the register were circular. An example of a 15 bit logical left shift for a CDC computer is shown below.

```
Before    11111111110001100000000000000000000000001111100000000000000000
After     00000000000000000000000001111100000000000000001111111111100011
```

5.3.1 I = ISL(X,N)

The ISL integer function returns the integer X shifted left N bits (returned in general register 0 for IBM and X6 for CDC). For IBM applications the integer function assumes 32 bit integers, while integers are 60 bits in length on CDC computers.

5.3.2 DA = DSL(X,N)

The DSL real function returns a real value of X, shifted left N bits (returned in floating register 0 for IBM and register X6 for CDC). The argument X as well as the result are 64 bit quantities (double precision) on IBM computers and 60 bit quantities (single precision) on CDC computers.

5.4 Arithmetic Right Shift

The net result of an arithmetic right shift is identical for both IBM and CDC computers. In other words, the low N bits of a word are shifted off the end of a register and the N high order bits are set to zero if the word is postive, or

the N high order bits are set to one if the word is negative (sign bit originally set), where N is the number of bit positions to be shifted. Two examples of 15 bit arithmetic right shifts are illustrated below for 32 bit words.

```
Before    11111111 00000000 00000000 11111111
After     11111111 11111111 11111110 00000000

Before    01011111 00000111 11111111 11111111
After     00000000 00000000 10111110 00001111
```

5.4.1 I = ISR(X,N)

The ISR integer function returns the integer X arithmetically shifted right N bits (returned in general register 0 for IBM and X6 for CDC). For IBM applications, the integer function assumes 32 bit integers, while integers are 60 bits in length on CDC computers.

5.5 Bit Mask Generation

Bit masks are often used in conjunction with the shift, logical product and logical sum functions to merge portions of several computer words or to isolate a portion of a given word in memory. Two examples of a 30 bit mask in a 60 bit word are given below.

```
High 111111111111111111111111111111000000000000000000000000000000
Low  000000000000000000000000000000111111111111111111111111111111
```

5.5.1 A = MASKF(N)

The MASKF function generates an N bit mask in the real word A, where A is a 64 bit word on IBM computers and a 60 bit word on CDC computers (returned in floating point register 0 for IBM and register X6 for CDC). For positive values of N, an N bit high order mask is generated, while an N bit low order mask is generated if N is negative, as illustrated in the previous example. The MASKF function must be typed as REAL in the calling program (see additional discussion in section VII.5).

5.6 Character String Comparison

CALL CØMPAR (A(I), B(J), NCHAR),

where A(I) = address of first character string
B(J) = address of second character string
NCHAR = number of characters to test.

This subroutine compares a character string beginning at location A(I) with another character string beginning at location B(J). The comparison is made one character at a time and terminated after all requested characters are tested or after the first inequality is found. The number of characters is returned unchanged if both character strings match and is set to zero if an inequality is present.

6.0 BUFFER I/O SUBROUTINE PACKAGE

The BUFIO package is a collection of interrelated subroutine entry points providing an interface between the Fortran program and I/O operations on disk or tape units. This package provides dynamic buffering for I/O units, issues requests for and releases hardware devices, does end-of-volume processing, and provides for efficient data transmission. Nine units are available for Fortran users. Units 1-5 are scratch disks on CDC and can be disk or tape on IBM. Units 11-14 are tape drives. These unit number designations do not necessarily correspond to Fortran unit numbers.

Records written by BUFIO on CDC hardware are compatible with Record Manager[VII.1-1] definitions for sequential files, block type C, and record type F. In addition W type records can be read as input. All records written on IBM hardware are type VBS, units 1-5 have a logical record length of 796 bytes and blocksize of 800 bytes while units 11-14 are 7292 bytes and 7296 bytes respectively.

Functions provided by the subroutine include reading and writing data, file positioning, and requesting and releasing devices. Entry points are provided to check if operations are complete for systems using asynchronous I/O such as IBM.

Since I/O operations on CDC systems and IBM systems are fundamentally different in their concept, this subroutine is machine dependent. The calling sequences are the same so that differences are transparent to a user. The CDC operation uses CIO rather than Record Manager and all requests are issued with recall. The IBM operation uses the BSAM access method of asynchronous I/O. These operations must be checked for completion since a request for action only initiates the operation.

A logical record is determined by the data transmitted by a write request. This length can be of any size and the calling program is responsible for deblocking the logical records when reading. A system record on the CDC system is terminated by inserting a level number ranging from 0 to 17_8, the latter being an end-of-file flag. On the IBM system, each write request produces a record with its record descriptor word. Buffers are filled in both cases as long as data is being transmitted to form physical records. The last physical record may be short. When reading, less data than a record may be requested, but in no case will more data be transmitted per request than exists in one record.

6.1 Writing Data

Data is transmitted to double I/O buffers in the IBM version and a single circular buffer in the CDC version by making calls to the BUFOUT routine. The data is assumed to be in logical records with the calling program responsible for control of the record size. Thus, any length record may be transmitted. No internal record terminators are appended to the data going to the CDC hardware device except when specifically requested by the calling program. Variable length record descriptor words are appended to each IBM logical record.

The calling sequence for a write request is as follows:

```
CALL BUFOUT (UNIT, A, NEXT, CHEK, LAST, VSN)
```

where

UNIT = An integer unit number
A = beginning location of data to write
NEXT = number of words to transmit (8 byte words assumed on IBM)
CHEK = status flag (returned) (Integer)
LAST = number of words written (returned)
VSN = Volume serial number of tape if unit is a tape drive (returned) (real) (REAL*8 on IBM)

Status codes returned in CHEK are as follows:

CHEK = 1 no error
CHEK = 2 error - likely an attempt to read or write after previous alternate operation and no intervening rewind.
CHEK = 3 end-of-record or end-of-file encountered
CHEK = 4 invalid unit number.

If the unit is a tape drive, the VSN is returned in the variable VSN after the first call to BUFOUT. The variable must accommodate six characters. The variable is ignored for disk files.

The I/O buffers are obtained for a file when the first data transmission request is issued. The buffer is an FTB file of 1025 words for CDC and 1600 bytes for IBM (800 bytes for each buffer). For this call to BUFOUT, CHEK is used to pass LOHI to the call to RESERV, i.e., if CHEK is 1, the buffer will be at the preferred end of core, if CHEK is 2, the buffer will be at the opposite end of core.

Data transmission is not forced to completion until an end-of-record or end-of-file is requested. Any file being written must be terminated by a call to BUFWEF to clear the buffers. The calling sequence for this entry is

```
CALL BUFWEF (UNIT, A, LEVEL, CHEK)
```

where

UNIT = unit number
A = dummy variable
LEVEL = level number for end-of-record
CHEK = status flag.

On CDC computers an end-of-record may have any level number between 0 and 16_8. A level number of 17_8 is equivalent to an end-of-file. If successful, the status flag is returned as 3.

No level number is written on IBM systems, but a short record is written and the buffers cleared.

6.2 Reading Data

Data is received from the I/O buffers by making calls to the BUFIN routine. Deblocking of records is the responsibility of the calling program. Data is transmitted in the size records requested unless an end-of-record is encountered on CDC or a segment end on IBM in which case a truncated record is transmitted. Once initiated, reading ahead on the unit continues until an end-of-record is encountered or the I/O buffers are full.

The calling sequence for the read request is as follows:

```
CALL BUFIN (UNIT, A, NEXT, CHEK, LAST, VSN)
```

where

UNIT = An integer unit number
A = beginning location where data is to be stored
NEXT = number of words requested to be read (8 byte words assumed on IBM)
CHEK = status flag (returned) (Integer)
LAST = number of words actually read (returned)
VSN = Volume Serial Number of tape if unit is a tape drive (returned) (real) (REAL*8 on IBM)

The status flag and buffer assignment are the same as for writing data.

6.3 Requesting Devices

Magnetic tapes units or catalogued disk files must be requested before attempting to read or write on them. No request is necessary for scratch disk units. Only two types of requests are permitted; the first is for a scratch file to be opened in the write mode with the write ring inserted, and the second is for existing files to be opened in the read mode and no write ring present. Writing on an existing file is not permitted. If this latter operation is desired, the file may be copied to a scratch unit and writing continued after the copy is finished.

The calling sequence for a file request is as follows:

```
CALL BUFREQ (UNIT, VSN, DEN, LABEL, DATE, CHEK)
```

where

UNIT = unit number
VSN = alphanumeric Volume Serial Number if file exists or is a disk unit.
= 0 or blank if scratch tape

8 174

DEN = alphanumeric recording density left adjusted (H format) (HI = 556 bpi, HY = 800 bpi, 7 trk drives), (GE = 6250 bpi, PE = 1600 bpi, HD = 800 bpi, 9 trk drives). For IBM hardware, DEN = blank designates data set resides on disk.

LABEL = alphanumeric tape label or data set DSNAME (up to 17 characters in two words)

DATE = Julian creation date if old tape. Julian date returned if scratch tape. Date in H format.

CHEK = status flag returned on both CDC and IBM. An 8-byte field passed for IBM systems with Hollorith label type passed in high order 4 bytes and file sequence number in low order 4 bytes. Label types permitted are "BLP" - bypass label processing; "SL" - standard labels, and "NL" - no label.

A catalogued disk file on CDC systems can be attached to a job through the request function on a read only condition. No provision is made for cataloging a new file. The calling sequence is as follows:

```
CALL BUFREQ (UNIT, VSN, CY, PFNAME, ID, CHEK)
```

where

UNIT = unit number (1-5)

VSN = 4HDISK

CY = cycle number (highest cycle available is used if CY = 0).

PFNAME = perm file name (2 words). Data set name on IBM.

ID = users ID.

CHEK = return code.

6.4 File Positioning

Several file positioning requests may be issued from the calling program. These include rewinding, unloading a reel, releasing the drive unit, spacing forward n records, and spacing forward n files.

The calling sequence is as follows:

```
CALL BUFSKP (UNIT, REC, FILE, CHEK)
```

8 175

where

UNIT = unit number
REC = depends upon operation
FILE = depends upon operation
CHEK = status flag.

The requested operation is defined by the parameters REC and FILE. A negative value for FILE defines a rewind operation. Possible combinations include

	FILE < 0
REC greater than 0,	rewind file
REC equal to 0,	rewind and release unit
REC less than 0,	rewind and unload reel.

A rewind is required after writing a file if the file is to be read back. Disk files which are released are returned to the system and become available to other jobs. Releasing tape files decrements the unit count on the CDC job card as well as freeing the drive for other jobs. The tape drive on IBM is not released as it is a queued device.

A positive or zero value for FILE defines a skip operation. On CDC computers FILE contains the level number of records to skip. This value can vary from 0-17 octal, where 17_8 defines an end-of-file. The level number is only a flag for IBM and no end-of-file is written except at an end-of-information. The variable REC contains the number of records of the specified level to skip. The record count is decremented on IBM at the same physical location as the level number on CDC records.

A check of end-of-record is made before skipping. If the system is at an end-of-record of the requested level, the record count is reduced by one. If this results in a zero record count, control is returned to the calling program without skipping any records.

8 176

7.0 EXTENDED I/O SUBROUTINES

Several extended I/O subroutines have been written to ensure the compatibility of FORTRAN programs with both IBM and CDC compilers. The extensions allow for consistent calling of in-core read and write operations, as well as output to the operator's console and the dayfile on CDC computers or the job allocation edit on IBM computers. These FORTRAN callable I/O extensions are discussed in the following sections.

7.1 Incore Read/Write Subroutines

7.1.1 CALL INCORW (NCHAR, FORMT, ARRAY, IOLIST, NWRDS)

The INCORW subroutine is written in FORTRAN and uses the CDC extension to FORTRAN, ENCODE and DECODE in the CDC version, and the INFILQ subroutine (documented below) for IBM applications. On a call to INCORW, NCHAR Hollerith characters from NWRDS words in the I/O list IOLIST, are written into the array ARRAY according to the format specified in the FORMT array. NCHAR and NWRDS are integers, while all of the arrays are real (60 bit words on CDC and 64 bit words on IBM). A sample usage of INCORW is given below, where the integer INT is converted into a 4 character Hollerith representation with blank fill in the low 4 characters.

```
DIMENSION REAL(1), ARRAY(1), FORMT(1)
EQUIVALENCE (REAL(1), INT)
DATA FORMT/  8H(A4,4X) /, INT/ 1333/
NCHAR = 8
NWRDS = 1
CALL INCORW (NCHAR, FORMT, ARRAY, REAL, NWRDS)
```

7.1.2 CALL INCORR (NCHAR, FORMT, ARRAY, IOLIST, NWRDS)

INCORR is an entry point in the INCORW subprogram, and its use is similar to that described above for INCORW. The difference is that for INCORR, the information is read from the array ARRAY and transmitted to the IOLIST array, much like a FORTRAN read which transmits data from a file buffer (corresponding to ARRAY), to the I/O list.

7.1.3 CALL INFILQ(ARRAY, NBYTES)

INFILQ is an IBM Fortran callable incore read and write subroutine. It modifies IBCOM# in such a way that the next read or write request causes information in the user's list to be placed in a core buffer. This routine performs functions similar to ENCODE and DECODE statements allowed in FORTRAN for CDC computers. INFILQ is called by the IBM versions of INCORW and INCORR in much the same way as ENCODE and DECODE are called by the CDC versions of INCORW and INCORR. On a call to INFILQ, ARRAY is the name of a temporary buffer sufficiently large to contain the information to be written, and NBYTES is an integer constant (or variable) which specifies the size of the buffer in bytes. A sample usage of INFILQ is given as follows:

```
      DIMENSION A(20), WD(80)
      READ(5,100) WD
      CALL INFILQ (A,80)
      WRITE (10,100) (WD(I),I=1,80)
100 FORMAT (80A1)
      CALL INFILQ (A,80)
      READ (10,101) X, Y, Z
101 FORMAT(3F8.5)
```

Several items to consider when using INFILQ include:

1. A call to INFILQ must precede each dummy read or write.
2. Any unit number (1-99) may be used in the dummy read or write statements.
3. No DD card is required for the unit used in the dummy statement.
4. The PRINT b, list and READ b, list statements, were b is the format statement number, are also permissable.

7.2 Extended Message Subroutine

7.2.1 CALL REMARK (17H MESSAGE GOES HERE)

The REMARK subroutine places a message in the dayfile on CDC computers and in the input JCL stream edit on IBM computers. The REMARK subroutine is very

useful on highly segmented subroutines, since it allows the programmer to edit when a new segment is loaded and entered. REMARK also allows the programmer to note error conditions that terminate a problem. The maximum message length is 80 characters. A message exceeding the maximum length is truncated, while a message shorter than the maximum must have binary zeros in the two Hollerith character positions following the message. These zeros are automatically supplied when a Hollerith constant is used as the parameter, as illustrated above. The REMARK subroutine is not physically present in the CDC environmental library and is actually obtained from the FORTRAN library, but it does physically reside in the IBM environmental library since it generally is not present in standard IBM FORTRAN libraries.

8.0 SYSTEM INTERROGATION SUBROUTINES

It is quite often necessary to interrogate the computer operating system to obtain information such as the current date, elapsed CPU time and remaining CPU or I/O time. Such information is not available through ANS standard FORTRAN subprograms; consequently, the following subprograms were written to allow consistent calling procedures for both IBM and CDC applications.

8.1 CALL DATE (D)

The current date is returned as the value of argument D in the form 10HbMM/DD/YYb for CDC computers and 8HMM/DD/YY for IBM computers, where b denotes a blank. MM is the number of the month, DD is the day of the month and YY is the year. The value returned is Hollerith data and may be output using an A format specification. D is a 60 bit word for CDC applications and a 64 bit word for IBM. The DATE subroutine is not physically present in the CDC environmental library and is actually obtained from the FORTRAN library. It does, however, physically reside in the IBM environmental library since it generally is not present in standard IBM FORTRAN libraries.

8.2 Interval Timer Routine

The interval timer routine provides the user with the means of determining the elapsed time of his task. Entry points to the interval timing routine are discussed in the following sections.

Initially TIMSET is called which sets a programmed timer to a zero reference time interval. The time interval is associated with the task that was active when the subroutine was called, and the timer interval is incremented only when the associated task is active.

The subroutine maintains two nested time measures. The outer measure is usually used for the total execution time for a set of data, and the inner measure is used for timing loops.

8.2.1 CALL TIMSET(X)

The call to TIMSET is used to set the time interval zero reference point, where X is a dummy argument (not used).

8.2.2 CALL TIMEL(X)

TIMEL is used to obtain the inner measure of time. X is returned as seconds in floating point form (60 bits for CDC, 64 bits for IBM) as the elapsed time from the last call to either TIMSET, TIMEL, or TIMER.

8.2.3 CALL TIMER(X)

TIMER is used to obtain the outer measure of time. X is returned as seconds in floating point form as the elapsed time from the last call to either TIMSET or TIMER.

8.3 L = NOTIM(ACPU, AIO)

The NOTIM logical function is used to determine if the job step is nearing its allotted time limit. Since CDC operating systems provide for both CPU and I/O time limits, both time limits are considered in the argument list. However, for IBM applications no I/O time limits apply and AIO should be carried as a dummy variable. The arguments ACPU and AIO are both input and returned (modified internally) quantities to NOTIM. On input to NOTIM, ACPU and AIO should specify the minimum amount of CPU and I/O time in seconds, respectively, that must be available to the job step. If less time is available than either of the two input limits, NOTIM is set to true. Conversely, if there is more CPU and I/O time available than that specified by ACPU and AIO, NOTIM is set to false. On return from NOTIM, ACPU contains the remaining CPU time in excess of the limit specified on INPUT to NOTIM and AIO contains the remaining I/O time in excess of the limit specified in input to NOTIM. Both ACPU and AIO are 60 bit floating point words for CDC applications and 64 bit floating point words for IBM.

9.0 MISCELLANEOUS SUBROUTINES

Various functions and procedures are often required for use by FORTRAN programs which are not standard ANS FORTRAN routines. These subprograms are not subject to a great deal of change or modification and may be used by several programs. As a result, such subroutines are included in an environmental library, which may be viewed as an extension to the FORTRAN library. These extensions are described in subsequent sections.

9.1 I = LOCF(A)

The LOCF function is used to obtain the address of variable A which may be any type. The integer address returned (60 bits on CDC and 32 bits on IBM), is the core address relative to the load point, corresponding to the location in memory occupied by A. For IBM applications, the address returned is in bytes, while the returned address is in units of 60 bit words for CDC computers. The LOCF function is not physically present in the CDC environmental library, but is obtained from the FORTRAN library. The LOCF function is not available as a FORTRAN library supplied routine for IBM application and is therefore physically present in the IBM environmental library.

9.2 X = FLOATR(A)

The FLOATR function is used to ensure that the value of A stored in location X is in floating point format. X is a 60 bit floating point number for CDC computers and a 64 bit floating point number for IBM computers. A may be either a floating point number or an integer. If A is a floating point number and FLOATR is being used in an IBM program, A must be a 64 bit floating point number. Integer values of A are 60 bit quantities on CDC computers and 32 bit quantities on IBM computers.

9.3 I = FINDEP (ENTRY,NAME)

The integer function FINDEP is used to obtain the address at which the primary entry point ENTRY is loaded. The address returned is relative to the load point and is in bytes for IBM use and 60 bit words for CDC. The Hollerith character string corresponding to the entry point name is returned in the NAME argument.

Since the Hollerith names can be six characters in length, NAME should be a 64 bit word for IBM applications. The entry point must be specified as external in any subroutine calling FINDEP.

9.4 CALL MOVE (A(I), B(J), NUM)

Subroutine MOVE transfers data from one array A, to a second array B. The contents of the first array are not modified unless the arrays overlap. The move operation may start at either the beginning of the arrays (a forward move) or at the end of the arrays (a backward move). The forward move is the faster of the two types and should always be chosen if the arrays do not overlap. In the case of overlap, the move direction depends on how the arrays are overlapped.

A backward overlapped move must be specified where the second array, B, begins at some element of the first array, A, such as

```
EQUIVALENCE (B(1), A(2)).
```

A forward overlapped move must be specified where A begins at some element of B, such as

```
EQUIVALENCE (A(1), B(2)).
```

Both A and B are comprised of 60 bit words on CDC computers (floating point or integer) and 64 bit floating point words on IBM computers. NUM is a signed integer whose absolute value specifies the number of words to move. A positive value of NUM indicates a forward move and a negative value, a backward move.

9.4.1 CALL MOVEI (A(I), B(J), NUM)

The MOVEI subroutine is used to move integer arrays, but is otherwise identical to MOVE. In the case of CDC computers where the integer and floating point word lengths are identical, calls to MOVE and MOVEI are equivalent. For IBM usage, the MOVEI subroutine is used to move 32 bit words, either floating point or integer, as opposed to the MOVE subroutine which moves 64 bit words.

9.4.2 CALL MOVEC (A(I), B(J), NUM)

The MOVEC subroutine is used on IBM computers only and performs moves at the character (byte) level. NUM indicates the number of characters (bytes) to be moved.

9.5 CALL ZEROUT (A(I), NUM)

Subroutine ZEROUT sets an arbitrary number of sequential real words to zero. The actual array type (INTEGER, LOGICAL, REAL, REAL*8) is not important; however, for IBM applications it must be recalled that ZEROUT assumes 8 byte words. NUM is the number of words to zero out (60 bit words on CDC and 64 bit words on IBM), and A(I) provides the starting address for the zero procedure and must be a double word boundary for IBM use.

9.6 CALL FABEND

Calling the FABEND subroutine causes an abnormal termination of the job step. A core dump is printed if a SYSUDUMP or SYSABEND DD card is included in the job step JCL for the IBM version. No modifications are made to the save areas so that the save area trace can show subroutines that were previously called.

The CDC version also provides a traceback of the subroutine calling sequence and generates a dump of the subroutine that calls FABEND.

9.7 CALL MINV (A, N, D, L, M)

Subroutine MINV is called to invert the order N matrix contained in array A. The method of solution is basically a double pivotal Gauss-Jordan reduction method with the pivot terms being selected from the largest eligible elements of the matrix. Upon entering the subroutine, A contains the matrix to be inverted, and when return is made to the calling program, A contains the inverse matrix. The area should be dimensioned to accommodate the maximum dimensions of a matrix to be inverted. All of the arrays and variables in the argument list are composed of 60 bit words for CDC applications and 64 bit words for IBM, with the exception of the integer N (32 bits on IBM), which indicates the order (dimension) of the matrix to be inverted. The determinant of the matrix is returned in D and the N element vectors L and M provide work space required by the double pivotal Gauss-Jordan reduction method.

VIII. OTHER CODE DETAIL

Previous sections have addressed the general coding philosophies used in developing the RETRAN Code Package, as well as some problem areas encountered during the development. Additionally, the overall code structure and the interfacing between the dynamic dimension and semi-modularization features have been discussed in some detail. However, several topics, such as code maintenance or code modification procedures and the loading schemes used for the various computing systems, have not been presented. These topics are the subjects of the following discussion.

1.0 CODE MAINTENANCE

During the development of the RETRAN Code Package, a single master copy of the source code was maintained for use on both IBM and CDC computers. The single master insured that the code versions installed on both IBM and CDC computers were in as close agreement as practically possible considering the different system architectures and operating philosophies. By maintaining a single master source, it was also much easier to insure that subsequent modifications to the code were not inadvertantly omitted from the source for a given machine.

The master copy of the source was maintained on a CDC computer in a compressed binary form, compatible with the UPDATE program[VIII.1-1]. To generate a RETRAN source for IBM application, an update was run and a card image source file generated by use of the UPDATE program. This source contained all *DECK, *COMDECK and *CALL cards that would be required to create a new compressed binary source, or OLDPL, for UPDATE. The source was then processed by a text editing program to comment out the CDC PROGRAM card; convert quotes (") to apostrophes ('); substitute a blank for any $ encountered in column one and add a C in column one of the following card;[1] and to add sequence information in columns 73 thru 80 for the UPD program discussed in Appendix B. The resulting source produced by the text editor is the IBM source code transmitted to IBM users, while the compressed binary file in OLDPL form is transmitted to CDC users.

1.1 RETRAN Maintenance on CDC Computing Systems

Maintenance of the RETRAN source code is facilitated through use of the UPDATE program as described in Reference VIII.1-1. The OLDPL or compressed binary source can not be processed directly by the compiler. Thus, the source must

---

[1]The $ in column one is treated as a comment card identifier by the FTN compiler. Consequently, $ and anything following are treated as comments. In CDC FORTRAN, entry point argument lists are assumed to be the same as the primary entry point, while IBM requires seperate argument list specifications for each entry. The IBM entry specifications are prefixed by a $ in column one on CDC versions, but the prefix is dropped for IBM version and the following card (CDC entry), is commented.

first be processed by the UPDATE program which generates a card image file containing the expansions of all comdecks, activated by *CALL cards. The resulting card image file is then processed by the compiler.

As noted earlier, the RETRAN source and the source for the environmental library reside on separate files, both of which are in OLDPL form.

1.2 RETRAN Maintenance on IBM Computing Systems

The source code for use on IBM systems is designed for the UPD program described in Appendix B, and in general can not be input to the compiler as is. UPD must be used as an intermediate step to provide the comdeck expansion in the auxiliary file that supplies the input to the compiler. Use of an intermediate UPD step may seem like an unnecessary complication, however, once the *COMDECK feature is understood, it becomes evident that code maintenance or modification is greatly simplified. As an example, RETRAN contains some 200 subroutines, and virtually all of these contain the FTB labeled common block which resides in comdeck BKCM. Without the comdeck, or a similar feature, coding changes to the FTB common block would require that each of the subroutines be similarly modified. Using comdecks, only a single copy of the coding exists and it resides in the comdeck. The *CALL in effect issues a request to UPD to insert the comdeck contents in the auxiliary file card image stream (*CALL is not placed in the auxiliary file). As a result, there is only one physical copy of redundant coding to modify, and any subsequent *CALL statements will result in the most recent copy being inserted in the auxiliary file. The comdeck feature is described in the UPD Manual in Appendix B.

Both the RETRAN source and the environmental library reside on separate files in a form acceptable to UPD.

2.0 OVERLAY DIRECTIVES

The RETRAN Code Package utilizes the overlay or segmented loading features on both IBM and CDC computing systems. Little correspondence exists between the IBM and CDC loader directives, owing to the different operating philosophies around which the loader software was developed. In the course of implementing RETRAN on several different processors and operating systems, problem areas were encountered on IBM and CDC computers, in that loader directives may work satisfactorily for one system configuration and not another for a given vendor, i.e. IBM or CDC. The various loading schemes that have been used successfully during the development of RETRAN, and their known range applicability, are given in the following sections.

2.1 CDC CYBER 70/170 and 6000 Series Computing Systems

Two segmented loaders have been used on CDC hardware, where both perform the same functions in a similar manner, but each has its own idiosyncrasies. In particular, the two loaders are the "old" Cyber Loader documented in Reference II.1-4 and the "new" Cyber Loader documented in Reference II.1-5. The loader directives given in Table VIII.2-1 have been used successfully on the "old" and the "new" Cyber Loaders provided PSR 452 has been implemented in the "new" loader.

Errors exist in the PSR 439 level of the "new" loader which cause catastrophic failures at execution time. Basically, the error occurs when a given subroutine resides in several segments. The loader loads the subroutine into the multiple segments properly, but the linkage is often performed incorrectly. Linkage is generally made with a copy of the loaded subroutine that resides in a segment that conflicts with the calling subroutine. The conflicting segment is not swapped into memory and as a result, the return jump branches to the address where the conflicting subroutine would reside, provided it was in memory. At this point the code "terminates" in occasionally unpredicable ways. The loader directives illustrated in Table VIII.2-2 were generated as a temporary measure to bypass the error in the loader, but result in a significantly larger field length requirement. An approximate increase in memory of 30K octal is realized when the directives in Table VIII.2-1 are used as opposed to those in Table VIII.2-2.

TABLE VIII.2-1

CYBER LOADER SEGLOAD DIRECTIVES-FINAL FORM

```
*                                ROOT SEGMENT
*                                ************
          GLOBAL     STH20C,FTB
          GLOBAL     Q8.IO.,FCL.C.,REW.FO,JMPS.RM,A0B.RM,CON.RM
          TREE       RMAIN-(INPUT,MOD1,MOD2,MOD3)
 INPUT    INCLUDE    INPUT,DOCUMT,GETCOR,INITAL,FRECOR,REDUCE
 INPUT    INCLUDE    INP,CVI,INP2,LINK,MODER,PCKUPK,EOF
 RMAIN    INCLUDE    RMAIN,BUFOUT,CIO=,CHEK,DELETE,DMPFIL,DMPLST
 RMAIN    INCLUDE    ERRMOD,ERROR,FABEND,FAIL,FTBIO,IA,IDFIND,LCONTG
 RMAIN    INCLUDE    LOCATE,LOCF,MOVE,NEXTID,RECOVR,RESERV,SHIFT
 RMAIN    INCLUDE    SHFTLK,TIMINT,TIMSET,ZEROUT
 RMAIN    INCLUDE    SYSTEMC,PUT.RM,REW.SQ,SYS.RM
*
*                              PLOTER MODULE
*                              *************
 MOD3     TREE       PLOTER-(INPLOT,PLOTS)
 INEDITD  TREE       INEDIT3-(EDATA1D,EDATA2D,EDATA3D,EDATA4D,EDINIT3,REQ
,EST3)
 PLOTS    TREE       TRNCAT-(READ,PLTS)
 PLTS     TREE       PLOT-LOCMD3
 READ     TREE       REDTAP-(PULLIN,INEDITD)
 EDATA1D  INCLUDE    EDATA1
 EDATA2D  INCLUDE    EDATA2
 EDATA3D  INCLUDE    EDATA3
 EDATA4D  INCLUDE    EDATA4
 EDINIT3  INCLUDE    EDINIT,INP2,LINK,MODER,PCKUPK,MXSETS,SEARCH,INCNT2
 INEDIT3  INCLUDE    INEDIT
 INPLOT   INCLUDE    INPLOT,INP2,INP8,LINK,MODER,PCKUPK
 LOCMD3   INCLUDE    LOCMD3
 PLOT     INCLUDE    PLOT,AXIS,LAXIS,PLOTMC,PLOTR,XPLOT,XPLOTA
 PLOT     INCLUDE    PLOTS,NUMBER,NORMAL,PSCALE,SYMBOL,LABLCK,FLOATR
 PLOTER   INCLUDE    PLOTER
```

TABLE VIII.2-1 (Cont'd)

```
 PULLIN    INCLUDE   SETUPE,POSITN,PULLIN,SETUPO,POSPLT,PULTRN
 REDTAP    INCLUDE   REDTAP,FRECOR,REDUCE,ISFOPN,LAVAIL,NFSETS
 REQEST3   INCLUDE   REQEST,POSPLT
TRNCAT     INCLUDE   TRNCAT,ISFDES
/
*
*                              REEDIT MODULE
*                              *************
 MOD2      TREE      REEDIT-(INEDT,EDITRE)
 INEDT     TREE      INEDTE-(SETUPE,INMINE)
 INMINE    TREE      INEDIT2-(EDATA1C,EDATA2C,EDATA3C,REEDATA,EDINIT2,REQ
,EST2)
 REEDATA   TREE      EDATA4C-LOCMD2
 EDATA1C   INCLUDE   EDATA1
 EDATA2C   INCLUDE   EDATA2
 EDATA3C   INCLUDE   EDATA3
 EDATA4C   INCLUDE   EDATA4
 EDINIT2   INCLUDE   EDINIT,MXSETS,SEARCH,INCNT2
 EDITRE    INCLUDE   EDITRE,EDIT,FLOATR,POSITN,PULLIN,REDUCE,FRECOR
 EDITRE    INCLUDE   PRNPLT,PLOTPR,ROUND,ISFOPN,NFSETS,TRNCAT
 EDITRE    INCLUDE   EDTCND,EDTCOR,EDTDNB,EDTHTX,EDTKIN,EDTLGL
 EDITRE    INCLUDE   EDTPMP
 INEDTE    INCLUDE   INEDTE,INP2,LINK,MODER,PCKUPK,INP8
 INEDIT2   INCLUDE   INEDIT
 LOCMD2    INCLUDE   LOCMD2
 REEDIT    INCLUDE   REEDIT
 REQEST2   INCLUDE   REQEST,POSPLT,PULLIN
 SETUPE    INCLUDE   SETUPE,POSITN,PULLIN
/
*
*                          RETRAN AND RESTRT MODULES
*                          *************************
```

TABLE VIII.2-1 (Cont'd)

```
 MOD1      TREE       RETRAN-(INMODS,TRAN,INITAL,PRNPLT,CPYPLT,REDUCE)
 INMODS    TREE       INRTRN-INRSTR-(INDATA,INDRIV)
 INDRIV    TREE       INTRAN-(INCNST,INVOL,INJUN,INPM,INHTXQ,INPWR,INHT,IN
,IFTE,INEDITB,INDNB,INCNT1)
 INEDITB   TREE       INEDIT1-(EDATA1B,EDATA2B,EDATA3B,EDATA4B,EDINIT1,REG
,EST1)
 INHT      TREE       INHEAT-(INCORE,INGEOM,INMPRO,INSLAB)
 INITAL    TREE       STSTAT-(INITLZ,JVEDIT)
 INPWR     TREE       INPOWR-(INRKEN)
           LEVEL      *** ALL OPTIONS LIVE IN SECOND LEVEL ABOVE ENTRY
*                     POINT LOCMD1. SUBROUTINE ORDERING ABOVE LOCMD1 IS
*                     EXPILICITLY CONNECTED TO CODING IN SUBR. RESOPT.
 OPTIONS   TREE       LOCTBL-(LOCMD1,LOGO)
 CPYPLT    INCLUDE    CPYPLT,OVRLYP,POSITN
 EDATA1B   INCLUDE    EDATA1
 EDATA2B   INCLUDE    EDATA2
 EDATA3B   INCLUDE    EDATA3
 EDATA4B   INCLUDE    EDATA4
 EDINIT1   INCLUDE    EDINIT,MXSETS,SEARCH,INCNT2
 INCNST    INCLUDE    INCNST,STH2OI,INTSTP,TSTMOD,INTRIP,INCNT1
 INCORE    INCLUDE    INCORE
 INDATA    INCLUDE    INDATA
 INDNB     INCLUDE    INDNB,INSTGN
 INEDIT1   INCLUDE    INEDIT
 INGEOM    INCLUDE    INGEOM
 INHEAT    INCLUDE    INHEAT
 INHTXQ    INCLUDE    INHTXQ
 INIFTE    INCLUDE    INIFTE
 INITLZ    INCLUDE    INITLZ,MASBAL,EQSETS,MINV,LOOPS
 INJUN     INCLUDE    CHAIN,IMIXCK,INJUN,INCKV,INAREA,INFILL
 INMPRO    INCLUDE    INMPRO
 INPM      INCLUDE    INPM,INPUMP,PMPDTA,IPMCK
```

TABLE VIII.2-1 (Cont'd)

```
INPOWR    INCLUDE   INPOWR
INRKEN    INCLUDE   INSCRM,INREAC,INRKEN
INRTRN    INCLUDE   INRTRN,INP2,LINK,MODER,PCKUPK,INP10
INRSTR    INCLUDE   INRSTR
INSLAB    INCLUDE   INSLAB,INCDHT
INTRAN    INCLUDE   INTRAN,INP8
INVOL     INCLUDE   INVOL,INBUBL,INLVC,INTV
JVEDIT    INCLUDE   JVEDIT,KINITL
LOCMD1    INCLUDE   BAL,BUBB,ENTRAN,FANG,FRICTN,HEADC,JUNHP,MACH,PNONEQ
LOCMD1    INCLUDE   POLATE,POL2,POSTW,PREW,SFR,STATE,STH200,STH201
LOCMD1    INCLUDE   STH203,STH204,STH205,STPM,TEMLIQ,TFFM,TRIP,VAPOR1
LOCMD1    INCLUDE   VISC,WPACK,SSWTCH,SURTEN,FLOATR
*         OPTIONS START HERE
LOCMD1    INCLUDE   LOCMD1,RESSEG
LOCMD1    INCLUDE   CHKV
LOCMD1    INCLUDE   FILL
LOCMD1    INCLUDE   PLTAPE
LOCMD1    INCLUDE   SLABHT,SENG,TAVE,SLABDT,TEMP,COND,HTRC,PCHF,QDOT
LOCMD1    INCLUDE   THCON,TKANDC,EDTCND
LOCMD1    INCLUDE   ENERGY
LOCMD1    INCLUDE   HTXG,EDTHTX
LOCMD1    INCLUDE   CONDHT
LOCMD1    INCLUDE   PUMP,PUMPS,EDTPMP
LOCMD1    INCLUDE   POWRT
LOCMD1    INCLUDE   CORG,EDTCOR
LOCMD1    INCLUDE   MH2OR,MWR
LOCMD1    INCLUDE   RKEN,REAC,CCC,RNDO,SCRM,EDTKIN
LOCMD1    INCLUDE   LEVCAL,EDTLQL
LOCMD1    INCLUDE   CARDBC
LOCMD1    INCLUDE   PRESS
LOCMD1    INCLUDE   TAPEBC
LOCMD1    INCLUDE   MIXFLO
```

TABLE VIII.2-1 (Cont'd)

```
LOCMD1    INCLUDE   DNBM,QDNBBW,QDNBMA,QDNBW3,FCOLDW,FGRSPM,NONUHF
LOCMD1    INCLUDE   FITHT,SIMQ,QDNBBA,QDNBBO,QDNBJL,EDTDNB
LOCMD1    INCLUDE   PRZR
LOCMD1    INCLUDE   TRNSPT
LOCMD1    INCLUDE   CONTRL,DIFF,DELAY,INTEG,LDLAG,LAG,VELLIM
LOCMD1    INCLUDE   LOCEND,RESOPT,FINDEP,DMPER
LOCTBL    INCLUDE   LOCTBL
LOGO      INCLUDE   LOGO
PRNPLT    INCLUDE   PRNPLT,ROUND,ISFOPN,TRNCAT,PLOTPR,NFSETS,TRPSUM
REDUCE    INCLUDE   FRECOR,REDUCE
REQEST1   INCLUDE   REQEST,POSPLT,PULLIN
RETRAN    INCLUDE   RETRAN
STSTAT    INCLUDE   DELHP,HAVG,JHOFF,PRSORK,SINITL,STATPH,STSTAT,TEMZ
STSTAT    INCLUDE   VDUDT,BUBINT
TRAN      INCLUDE   CSLVOL,CVGEXP,CVGIMP,EDIT,ETSCON,FLOSRH
TRAN      INCLUDE   LEAK,NIFTE,NOTIM,TRAN,TRIPDT,TSED,TSTP
          END       RMAIN
```

TABLE VIII.2-2

CYBER LOADER SEGLOAD DIRECTIVES-INTERIM FORM

```
*                                ROOT SEGMENT
*                                ************
          GLOBAL     Q8.IO.,FCL.C.,REW.FO,JMPS.RM,AOB.RM,CON.RM
          GLOBAL     STH2OC,FTB
          TREE       RMAIN-(INPUT,MOD1,MOD2,MOD3)
 RMAIN    INCLUDE    ALOG,DMPFIL,DMPLST
*
*                                 PLOTER MODULE
*                                 *************
 MOD3     TREE       PLOTER-(INPLOT,PLOTS)
 PLOTS    TREE       CLOSE-(READ,PLTS)
 PLTS     TREE       PLOT-LOCMD3
 READ     TREE       REDTAP-(PULLIN,SETUPO)
 PLOT     INCLUDE    PLOT,PLOTS,SYMBOL,LABLCK,LAXIS,PLOTMC,PLOTR,XPLOTA
 PLOT     INCLUDE    ISFDES,AXIS,NORMAL,NUMBER,PSCALE,XPLOT
*
*                                 REEDIT MODULE
*                                 *************
 MOD2     TREE       REEDIT-(INEDT,EDITRE)
 INEDT    TREE       INEDTE-(SETA,INP8)
 SETA     TREE       SETUPE-LOCMD2
*
*
*                          RETRAN AND RESTRT MODULES
*                           ************************
 MOD1     TREE       RETRAN-(INMODS,TRAN,INITAL,PRNPLT,CPYPLT)
 INMODS   TREE       INRTRN-INRSTR-(INDATA,INDRIV)
 INDRIV   TREE       INTRAN-(INCNST,INVOL,INJUN,INPM,INHTXQ,INPWR,INHT,IN
,IFTE,INDNB,INCNT1,INCNT2)
 INHT     TREE       INHEAT-(INCORE,INGEOM,INMPRO,INSLAB)
 INITAL   TREE       STSTAT-(INITLZ,JVEDIT)
 INPWR    TREE       INPOWR-(INRKEN)
```

TABLE VIII.2-2 (Cont'd)

```
          LEVEL      *** ALL OPTIONS LIVE IN SECOND LEVEL ABOVE ENTRY
*                    POINT LOCMD1. SUBROUTINE ORDERING ABOVE LOCMD1 IS
*                    EXPILICITLY CONNECTED TO CODING IN SUBR. RESOPT.
 OPTIONS  TREE       LOCTBL-(LOCMD1,LOGO)
 STSTAT   INCLUDE    SINITL,TEMZ,LOOPS
 LOCMD1   INCLUDE    BAL,BUBB,ENTRAN,FANG,FRICTN,HEADC,JUNHP,MACH
 LOCMD1   INCLUDE    POLATE,POL2,POSTW,PREW,SFR,STATE,STH200,STH201
 LOCMD1   INCLUDE    STH203,STH204,STH205,STPM,TFFM,TRIP,VAPOR1
 LOCMD1   INCLUDE    VISC,WPACK,SSWTCH,SURTEN
*         OPTIONS START HERE
 LOCMD1   INCLUDE    LOCMD1,RESSEG,CHKV,FILL
 LOCMD1   INCLUDE    PLTAPE,SLABHT,SENG,TAVE,SLABDT,TEMP
 LOCMD1   INCLUDE    COND,HTRC,PCHF,QDOT,THCON,TKANDC,ENERGY,HTXQ
 LOCMD1   INCLUDE    CONDHT,PUMP,PUMPS,POWRT,CORQ,MH2OR,MWR,RKEM,REAC,CCC
 LOCMD1   INCLUDE    RNDO,SCRM,LEVCAL,CARDBC,PRESS,TAPEBC,MIXFLO
 LOCMD1   INCLUDE    DNBM,QDNBBW,QDNBMA,QDNBW3,FCOLDW,FGRSPM
 LOCMD1   INCLUDE    NONUHF,FITHT,SIMQ,QDNBBA,QDNBBO,QDNBJL
 LOCMD1   INCLUDE    PRZR,TRNSPT,CONTRL,DIFF,DELAY,INTEG,LDLAG,LAG,VELLIM
 LOCMD1   INCLUDE    LOCEND,RESOPT,FINDEP,DMPER
          END     RMAIN
```


8 195

The error in the "new" Cyber Loader was corrected between PSR 439 and PSR 452. It should also be noted that the directives in Table VIII.2-2 will not work on the "old" loader since the segment structure is not completely defined.

2.2 IBM 360/370 Computing Systems With OS or MVS Operating Systems

The overlay feature of the IBM loader described in Reference II.1-6, is used at installations running OS or MVS operating systems. However, the loader directives that are used on MVS are significantly different than those used on OS operating systems in order to bypass an apparent flaw in the implementation of the overlay feature on MVS. The details of the problem are given in Section II.3.2. Tables VIII.2-3 and VIII.2-4 illustrate the overlay directives used with OS and MVS operating systems, respectively.

TABLE VIII.2-3

OS OVERLAY DIRECTIVES

```
      INCLUDE  ADD(RETRAN)
INSERT  MAIN,CHEK,FAIL,TIMINT,DSL,DSR,MOVE,BUFOUT,CLOSE,DELETE,DMPFIL
INSERT  DMPLST,DSCRIB,ERROR,FABEND,FINDEP,IDFIND,ISFDES,ISFOPN,LAVAIL
INSERT  LCONTG,LOCATE,NFSETS,NEXTID,OPEN,PROCES,PROC4,REDUCE,RESERV
INSERT  SHFTLK,SHIFT,TRNCAT,ZEROUT,IBCOM#,LOCF,DAND,INFILG,DCR
INSERT  TIMSET
OVERLAY R102
INSERT  INPUT,CVI,DATE,GETCOR,INITAL,INP,INPSHF
OVERLAY R102
INSERT  RETRAN,STH201,STH203,STH204,TRPSUM
OVERLAY R103
INSERT  INRTRN,INRSTR,INTRAN
OVERLAY R104
INSERT  INVOL,INBUBL,INLVC,INTV,INCNST,STH201,INTSTP,TSTMOD,INTRIP
INSERT  INCNT1
OVERLAY R104
INSERT  CHAIN,IMIXCK,INJUN,INCKV,INAREA,INFILL
OVERLAY R104
INSERT  INPM,INPUMP,PMPDTA,IPMCK
OVERLAY R104
INSERT  INHEAT,INCORE,INGEOM,INMPRO,INSLAB,INCDHT,INHTXG
OVERLAY R104
INSERT  INPOWR,INREAC,INRKEN,INSCRM,INDNB,INIFTE
OVERLAY R104
INSERT  INEDIT,EDATA1,EDATA2,EDATA3,EDATA4,EDINIT,REQEST
OVERLAY R104
INSERT  INDATA,CPYPLT,OVRLYP
OVERLAY R103
INSERT  STSTAT,HAVG
OVERLAY R104
INSERT  STATPH,INITLZ,MASBAL,EGSETS,PRSORK,SINITL,TEMZ,JHOFF,JVEDIT
INSERT  KINITL,VDUDT,DELHP,MINV,LOOPS
```

TABLE VIII.2-3 (Cont'd)

```
OVERLAY R103
INSERT  TRAN,TSTP,NOTIM
OVERLAY R104
INSERT  FLOSRH,LEAK,NIFTE,CSLVOL,CVGEXP,ETSCON,TRIPDT
INSERT  TSED,PLTAPE
OVERLAY R103
INSERT  PRNPLT,PLOTPR,ROUND
OVERLAY R102
     INSERT  LAXIS,PLOTMC,SYMBOL,XPLOTA,XPLOT,PLOTR
OVERLAY R103
     INSERT  INPLOT
OVERLAY R201(REGION)
INSERT  INP2,INP4,INP5,INP6,INP7,INP8,INP10,LINK,MODER,INPUPR
OVERLAY R201
INSERT  EDIT,EDTCND,EDTCOR,EDTDNB,EDTHTX,EDTKIN,EDTLQL,EDTPMP
OVERLAY R301(REGION)
INSERT  BAL,BUBB,ENTRAN,FANG,FRICTN,HEADC,JUNHP,MACH,POLATE
INSERT  POL2,POSTW,PREW,SFR,STATE,STH2O5,STPM,TFFM,TRIP
INSERT  VAPOR1,VISC,WPACK,CHKV,FILL,SLABHT
INSERT  SENG,TAVE,SLABDT,TEMP,COND,HTRC,PCHF,QDOT,SURTEN,THCON
INSERT  TKANDC,ENERGY,HTXQ,CONDHT,PUMP,PUMPS,POWRT,CORQ,MH2OF,MWR
INSERT  RKEN,REAC,CCC,RNDO,SCRM,LEVCAL,CARDBC,PRESS,TAPEBC,MIXFLO
INSERT  DNBM,QDNBBW,QDNBMA,QDNBW3,FCOLDW,FGRSPM,NONUHF,FITHT
INSERT  SIMQ,QDNBBA,QDNBBO,QDNBJL,PRZR,TRNSPT,CONTRL,DIFF
INSERT  DELAY,INTEG,LDLAG,LAG,VELLIM
     OVERLAY R401(REGION)
     INSERT  BLKCM
     ENTRY   MAIN
     NAME RETRAN(R)
```

TABLE VIII.2-4

MVS OVERLAY DIRECTIVES[1]

```
INCLUDE  ADD(RETRAN)
INSERT MAIN
OVERLAY END(REGION)
INSERT BLKCM
ENTRY MAIN
NAME RETRAN(R)
```

---

[1]An IEW0152 error will occur but will not affect execution. LET must be specified on link edit step.

8 200

8 201

IX. SUBROUTINE AND FUNCTION SUBPROGRAM DEFINITIONS

A brief definition of subprograms in the RETRAN source code is included in this section. Each subprogram is listed with its argument list. This is followed by a description of the subprogram, a description of each item in the argument list and the subprograms that call it.

```
**********************************************************************

   SUBROUTINE BAL (DT,STDATA)

SUBROUTINE BAL OBTAINS THE THERMODYNAMIC STATE PROPERTIES FOR
ALL NORMAL VOLUMES ON A U-V STATE PLANE, PLUS CALLS SUBROUTINES
THAT OBTAIN STATE PROPERTIES FOR TIME DEPENDENT VOLUMES AND NON-
EQUILIBRIUM VOLUMES (PRESSURIZER MODEL).

BAL IS CALLED FROM SUBROUTINES TRAN, STSTAT, AND JUNHP.

DT       = CURRENT TIME STEP SIZE                    (INPUT)
STDATA   = ARRAY TO HOLD THE THERMODYNAMIC STATE PROPERTIES
           RETURNED FROM STATE ROUTINE               (INPUT)

**********************************************************************

   SUBROUTINE BUBB (I1,I2,DEL,H,C,X,INITL,DT,WFRAC)

SUBROUTINE BUBB DEFINES JUNCTION ENTHALPY AND QUALITY PLUS
PARAMETER C FOR A NON-HOMOGENEOUS VOLUME BUBBLE MASS INTEGRATION.

BUBB IS CALLED FROM SUBROUTINES DELHP,BUBINT, AND
JUNHP.

I1       = VOLUME FILE INDEX                         (INPUT)
I2       = JUNCTION FILE INDEX                       (INPUT)
DEL      = ESTIMATED GAS MASS CHANGE                 (INPUT)
H        = ENTHALPY AT JUNCTION                      (OUTPUT)
C        = FRACTION OF GAS REMOVED BY THE JUNCTION FROM
           ABOVE MIXTURE                             (OUTPUT)
X        = QUALITY AT JUNCTION                       (OUTPUT)
INITL    = INITIAL CONDITION INDEX, 0 = TRANSIENT, 1 = IC
           A CALL TO BUBB WITH INITL = 1 CAN ALSO BE USED TO SKIP
           ENTHALPY TRANSPORT WHEN ONLY C IS WANTED  (INPUT)
DT       = TIME STEP SIZE                            (INPUT)
WFRAC    = MASS FRACTION OF WATER IN JUNCTION FLOW   (OUTPUT)

**********************************************************************

   SUBROUTINE BUBINT (IDXV,ISZV,ITERST)

SUBROUTINE BUBINT INITIALIZES THE BUBBLE RISE MODEL FOR STEADY
STATE.

BUBINT IS CALLED FROM SUBROUTINE STSTAT.
```

```
IDXV      = INDEX OF A SCRATCH FILE USED FOR STEADY STATE TO STORE
            VOLUME VARIABLES                          (INPUT)
ISZV      = SETSIZ OF THE SCRATCH FILE                (INPUT)
ITERST    = NUMBER OF ITERATIONS LEFT TO ITERATE IN STEADY STATE
                                                      (INPUT)
```

****

SUBROUTINE CARDBC

SUBROUTINE CARDBC OBTAINS THERMODYNAMIC BOUNDARY CONITIONS
FROM TABULAR INPUT FOR TIME DEPENDENT VOLUMES.

CARDBC IS CALLED BY SUBROUTINES INVOL, BAL AND RESOPT.

****

SUBROUTINE CCC (X,CA,CB,CC)

SUBROUTINE CCC CALCULATES EXPONENTIAL COEFICIENTS OF X.

CCC IS CALLED FROM SUBROUTINE RKEN.

```
X         = REAL NUMBER                             (INPUT)
CA        = EXPONENTIAL COEFFICIENTS OF X           (OUTPUT)
CB        = EXPONENTIAL COEFFICIENTS OF X           (OUTPUT)
CC        = EXPONENTIAL COEFFICIENTS OF X           (OUTPUT)
```

****

SUBROUTINE CHAIN (IKP,JKP,IETA,NUETA,NTOT1,NTRM1,NCHAIN)

SUBROUTINE CHAIN SETS UP THE JUNCTION CONNECTION MATRIX TO BE USED
BY SUBROUTINE NIFTE FOR EVALUATION OF THE CONSERVATION EQUATIONS.

CHAIN IS CALLED FROM SUBROUTINE INJUN.

```
IKP       = ARRAY CONTAINING VOLUME NUMBER OF THE VOLUMES ON THE
            "FROM" SIDE OF THE JUNCTION               (INPUT)
JKP       = ARRAY CONTAINING VOLUME NUMBERS OF THE VOLUMES ON THE
            "TO" SIDE OF THE JUNCTION                 (INPUT)
IETA      = ARRAY CONTAINING VOLUME CHAINS            (INPUT)
NUETA     = ARRAY CONTAINING THE NUMBER OF VOLUMES
            IN EACH CHAIN                             (INPUT)
NTOT1     = NUMBER OF JUNCTIONS                       (INPUT)
NTRM1     = NUMBER OF VOLUMES                         (INPUT)
NCHAIN    = NUMBER OF ITEMS IN JUNCTION CONNECTION ARRAY  (OUTPUT)
```

****

SUBROUTINE CHEK (IUNIT,ITYP,LABL,DEN)

SUBROUTINE CHEK MOUNTS THE REQUESTED DATA TAPE, CHECKS THE TAPE LABEL AND VSN, AND DETERMINES THE DATA TAPE TYPE.

CHEK IS CALLED FROM SUBROUTINES REDTAP, EDITRE, INEDTE OVRLYP, INRSTR, AND TAPEBC.

IUNIT	= TAPE UNIT NUMBER	(INPUT)
ITYP	= 1 FOR A RETRAN DATA TAPE	(OUTPUT)
	= 2 FOR RELAP4/003 UPDATE 54 AND UP, DATA TAPES	
	= 3 FOR RELAP4/002 DATA TAPES	
	= 4 FOR RELAP3 DATA TAPES	
	= 5 FOR RETRAN STRANGER DATA TAPES	
LABL	= ARRAY CONTAINING DATA TAPE LABEL, VSN AND CREATION DATE	(INPUT)
DEN	= DENSITY OF THE DATA WRITTEN ON TAPE	(INPUT)

****

SUBROUTINE CHKV (I,DT)

SUBROUTINE CHKV DETERMINES THE JUNCTION FLOW AREA FOR JUNCTION AREAS THAT ARE CONTROLLED BY ANY TYPE OF VALVE.

CHKV IS CALLED FROM SUBROUTINES INITLZ, PREW, AND RESOPT.

I	= JUNCTION FILE INDEX	(INPUT)
DT	= TIME STEP SIZE	(INPUT)

****

SUBROUTINE COND (L1,L2,DT)

SUBROUTINE COND ACTS AS A DRIVER FOR OBTAINING HEAT CONDUCTOR TEMPERATURE PROFILES, HEAT TRANSFER COEFFICIENTS, AND HEAT GENERATION RATES.

COND IS CALLED BY SUBROUTINE SLABHT.

L1	= FILE INDEX OF CONDUCTOR AT BOTTOM OF STACK	(INPUT)
L2	= FILE INDEX OF CONDUCTOR AT TOP OF STACK	(INPUT)
DT	= RETRAN TIME STEP SIZE	(INPUT)

****

SUBROUTINE CONDHT (K,I)

SUBROUTINE CONDHT COMPUTES HEAT TRANSFER COEFFICIENTS FOR THE CONDENSING HEAT TRANSFER MODEL.

8 205

CONDHT IS CALLED FROM SUBROUTINES SLABHT, SINITL, AND RESOPT.

K	= VOLUME FILE INDEX	(INPUT)
I	= CONDENSING HEAT TRAN. SET INDEX	(INPUT)

****

SUBROUTINE CONTRL (INIT,NTS,ETIME,DT)

SUBROUTINE CONTRL IS THE DRIVER FOR ALL THE CONTROL SYSTEM ELEMENTS. THE ORDER OF CALCULATION FOR EACH ELEMENT IS DETERMINED BY THE ORDER IN WHICH THE ID OF THE ELEMENT IS STORED IN ARRAY IDC IN FILE 54.

CONTRL IS CALLED BY SUBROUTINES TRAN, STSTAT, AND RESOPT.

INIT	= 0 INITIALIZATION PASS	(INPUT)
	= 1 NORMAL PASS	
NTS	= NUMBER OF ACTUAL RETRAN TIME STEPS	(INPUT)
ETIME	= ELAPSED PROBLEM TIME	(INPUT)
DT	= CURRENT TIME STEP TO BE USED BY CONTROL SYSTEM, IDENTICAL TO TIME STEP USED BY REST OF RETRAN	(INPUT)

****

SUBROUTINE CORQ (M,NS,K,VOL)

SUBROUTINE CORQ CALCULATES INTERNAL HEAT GENERATION RATE AND MODERATOR HEATING RATE FOR EACH ACTIVE CONDUCTOR.

CORQ IS CALLED BY SUBROUTINES SINITL AND COND.

M	= GEOMETRY FILE INDEX	(INPUT)
NS	= NODE NUMBER AT RIGHT SURFACE	(INPUT)
K	= CORE FILE INDEX	(INPUT)
VOL	= VOLUME OF CONDUCTOR, FT3	(INPUT)

****

SUBROUTINE CPYPLT (IUNIT,JUNIT)

SUBROUTINE CPYPLT IS USED TO GENERATE A DUPLICATE COPY OF THE RETRAN DATA TAPE USED TO RESTART THE PROBLEM. THE COPY OPERATION IS TERMINATED AT THE LAST DATA RECORD PRIOR TO THE RESTART DATA RECORD TO INSURE A CONTINUOUS DATA SET FOR EACH INDIVIDUAL PROBLEM. DATA RECORDS ARE COPIED FROM IUNIT TO JUNIT.

CPYPLT IS CALLED BY SUBROUTINE RETRAN.

1758 206

```
IUNIT     = UNIT NUMBER FOR DATA TAPE BEING READ
            FOR RESTART                          (INPUT)
JUNIT     = UNIT NUMBER FOR DATA TAPE BEING CREATED
            DURING A RESTART                     (INPUT)
```

****

SUBROUTINE CSLVOL

SUBROUTINE CSLVOL DETERMINES WHICH VOLUMES ARE CAUSAL, THAT IS VOLUMES FOR WHICH STATE CALLS ARE TO BE MADE.

CSLVOL IS CALLED BY SUBROUTINE NIFTE.

****

SUBROUTINE DELAY (INPUT,OUTPUT,DLAY,L,BOX,INDEX,GAIN)

SUBROUTINE DELAY MODELS A TIME DELAY ELEMENT. THE INPUT IS SAMPLED AT EQUALLY SPACED TIME INTERVALS AND STORED IN AN ARRAY. THE OUTPUT IS CALCULATED BY LINEARLY INTERPOLATING AMONG VALUES STORED IN THE ARRAYS. SAMPLES OF INPUT ARE DISCARDED WHEN THEY CORRESPOND TO A TIME EARLIER THAN THE CURRENT TIME MINUS THE TIME DELAY.

DELAY IS CALLED BY SUBROUTINE CONTRL.

```
INPUT     = VALUE AT INPUT SIGNAL AT THE CURRENT TIME STEP  (INPUT)
OUTPUT    = VALUE OF THE OUTPUT SIGNAL AT THE CURRENT
            TIME STEP     (OUTPUT)
DLAY      = LENGTH OF TIME DELAY IN SECONDS                 (INPUT)
L         = NUMBER OF SAMPLES. NOTE: TIME INTERVAL BETWEEN SAMPLES
            EQUAL DLAY/L                                    (INPUT)
BOX       = STORAGE ARRAY REPRESENTING STATE OF THE
            DELAY                                           (INPUT/OUTPUT)
INDEX     = ON ENTRY TO SUBROUTINE DELAY, INDEX IS THE ADDRESS
            OF BOX(1) RELATIVE TO BOXX(1), ARRAY IN CALLING
            SUBROUTINE CONTRL BEFORE RETURNING TO CONTRL, INDEX
            IS INCREASED BY AN AMOUNT EQUAL TO THE LENGTH OF THE
            BOX ARRAY.                                      (INPUT/OUTPUT)
GAIN    = GAIN OF THE CONTROL BLOCK
```

****

SUBROUTINE DELHP (I,J)

SUBROUTINE DELHP COMPUTES THE JUNCTION ENTHALPY RISE DUE TO SEPARATION OR ENTHALPY TRANSPORT. THIS SUBROUTINE IS USED FOR STEADY STATE INITIALIZATION ONLY.

DELHP IS CALLED BY SUBROUTINE HAVG.

```
I         = RELOCATABLE VOLUME FILE INDEX
```

8 207

```
J         = RELOCATABLE JUNCTION FILE INDEX

**********************************************************************

   SUBROUTINE DIFF (INPUT,OUTPUT,GAIN,BOX)

SUBROUTINE DIFF MODELS A DIFFERENTIATOR FOR THE CONTROL SYSTEM.

DIFF  IS CALLED BY SUBROUTINE CONTRL.

INPUT     = VALUE AT INPUT SIGNAL AT THE CURRENT TIME STEP (INPUT)
OUTPUT    = VALUE OF THE OUTPUT SIGNAL AT THE CURRENT
            TIME STEP (OUTPUT)
GAIN      = GAIN OF DIFFERENTIATOR                          (INPUT)
BOX       = USED TO STORE THE PREVIOUS VALUE OF
            THE INPUT   (INPUT/OUTPUT)

**********************************************************************

   SUBROUTINE DNBM

SUBROUTINE DNBM IS DIRECTS THE FLOW OF THE DNBR COMPUTATIONS.

DNBM IS CALLED FROM SUBROUTINES TRAN, STSTAT AND RESOPT.

**********************************************************************

   SUBROUTINE DOCUMT (DOC)


SUBROUTINE DOCUMT DEFINES THE DOCUMENTATION HEADERS PRINTED
AT THE TOP OF EACH MAJOR EDIT.

DOCUMT IS CALLED BY SUBROUTINE INPUT.

DOC = ARRAY CONTAINING DOCUMENTATION HEADER              (OUTPUT)

**********************************************************************

   SUBROUTINE EDATA1 (MODE,Y,IX,IXI,INDEX1,JFIL,IUNIT,II)


SUBROUTINE EDATA1 SETS UP INFORMATION FOR EDIT VARIABLES AND
VARIABLES ARCHIVED ON DATA TAPE FROM RELOCATABLE FILES.
INFORMATION DEFINED IN EDATA1 IS INTIMATELY TIED IN WITH MAJOR
EDITS,MINOR EDITS, AND DATA TAPE MANIPULATION.

EDATA1 IS CALLED BY SUBROUTINE INEDIT.

MODE      = 1 FOR RETRAN OR RESTRT              (INPUT)
```

```
          = 2 FOR READIT
          = 3 FOR PLOTER
Y         = SCRATCH ARRAY USED TO STORE INFORMATION DESCRIBING
            THE RELOCATABLE FILES AND VARIABLES IN THE FILES (OUTPUT)
IX        = COUNTER FOR THE NUMBER OF VARIABLES DESCRIBED (OUTPUT)
IXI       = BEGINNING INDEX IN THE Y SCRATCH ARRAY WHERE THE
            INFORMATION DESCRIBING THE RELOCATABLE FILES AND
            VARIABLES IS STORED (OUTPUT)
INDEX1    = AN INDEX IN THE SCRATCH ARRAY Y USED AS A POSITION
            INDICATOR TO STORE INFORMATION DESCRIBING THE RELOC-
            ATABLE FILES    (INPUT)
JFIL      = AN ARRAY CONTAINING GENERAL HEADINGS.  THIS ARRAY IS
            DEFINED IN SUBROUTINE INEDIT    (INPUT)
IUNIT     = AN ARRAY CONTAINING VARIABLE UNITS.  THIS ARRAY IS
            DEFINED IN SUBROUTINE INEDIT    (INPUT)
II        = COUNTER FOR THE NUMBER OF FILES DESCRIBED  (OUTPUT)

**********************************************************************

   SUBROUTINE EDATA2 (MODE,Y,IX,IXI,INDEX1,JFIL,IUNIT,II)


SUBROUTINE EDATA2 SETS UP INFORMATION FOR EDIT VARIABLES AND
VARIABLES ARCHIVED ON DATA TAPE FROM RELOCATABLE FILES.
INFORMATION DEFINED IN EDATA2 IS INTIMATELY TIED IN WITH MAJOR
EDITS, MINOR EDITS, AND DATA TAPE MANIPULATION.

EDATA2 IS CALLED BY SUBROUTINE INEDIT.

MODE      = 1 FOR RETRAN OR RESTRT          (INPUT)
          = 2 FOR REEDIT
          = 3 FOR PLOTER
Y         = SCRATCH ARRAY USED TO STORE INFORMATION DESCRIBING
            THE RELOCATABLE FILES AND VARIABLES IN THE FILES  (OUTPUT)
IX        = COUNTER FOR THE NUMBER OF VARIABLES DESCRIBED (OUTPUT)
IXI       = BEGINNING INDEX IN THE Y SCRATCH ARRAY WHERE THE
            INFORMATION DESCRIBING THE RELOCATABLE FILES AND
            VARIABLES IS STORED     (INPUT)
INDEX1    = AN INDEX IN THE SCRATCH ARRAY Y USED AS A POSITION
            INDICATOR TO STORE INFORMATION DESCRIBING THE RELOC-
            ATABLE FILES                    (INPUT)
IUNIT     = AN ARRAY CONTAINING VARIABLE UNITS.  THIS ARRAY IS
            DEFINED IN SUBROUTINE INEDIT    (INPUT)
JFIL      = AN ARRAY CONTAINING GENERAL HEADINGS   THIS ARRAY IS
            DEFINED IN SUBROUTINE INEDIT    (INPUT)
II        = COUNTER FOR THE NUMBER OF FILES DESCRIBED  (OUTPUT)

**********************************************************************

   SUBROUTINE EDATA3 (MODE,Y,IX,IXI,INDEX1,JFIL,IUNIT,II)
```

8 209

SUBROUTINE EDATA3 SETS UP INFORMATION FOR EDIT VARIABLES AND VARIABLES ARCHIVED ON DATA TAPE FROM RELOCATABLE FILES. INFORMATION DEFINED IN EDATA3 IS INTIMATELY TIED IN WITH MAJOR EDITS, MINOR EDITS, AND DATA TAPE MANIPULATION.

EDATA3 IS CALLED BY SUBROUTINE INEDIT.

```
MODE     = 1 FOR RETRAN OR RESTRT              (INPUT)
         = 2 FOR REEDIT
         = 3 FOR PLOTER
Y        = SCRATCH ARRAY USED TO STORE INFORMATION DESCRIBING
           THE RELOCATABLE FILES AND VARIABLES IN THE FILES  (OUTPUT)
IX       = COUNTER FOR THE NUMBER OF VARIABLES DESCRIBED (OUTPUT)
IXI      = BEGINNING INDEX IN THE Y SCRATCH ARRAY WHERE THE
           INFORMATION DESCRIBING THE RELOCATIVE FILES AND
           VARIABLES IS STORED     (INPUT)
INDEX1   = AN INDEX IN THE SCRATCH ARRAY Y USED AS A POSITION
           INDICATOR TO STORE INFORMATION DESCRIBING THE RELOC-
           ATABLE FILES                          (INPUT)
JFIL     = AN ARRAY CONTAINING GENERAL HEADINGS. THIS ARRAY IS
           DEFINED IN SUBROUTINE INEDIT    (INPUT.
IUNIT    = AN ARRAY CONTAINING VARIABLE UNITS.  THIS ARRAY IS
           DEFINED IN SUBROUTINE INEDIT    (INPUT)
II       = COUNTER FOR THE NUMBER OF FILES DESCRIBED  (OUTPUT)
```

****

SUBROUTINE EDATA4 (MODE,Y,IX,IXI,INDEX1,JFIL,IUNIT,II)

SUBROUTINE EDATA4 SETS UP INFORMATION FOR EDIT VARIABLES AND VARIABLES ARCHIVED ON DATA TAPE FROM RELOCATABLE FILES. INFORMATION DEFINED IN EDATA4 IS INTIMATELY TIED IN WITH MAJOR EDITS, MINOR EDITS, AND DATA TAPE MANIPULATION.

EDATA4 IS CALLED BY SUBROUTINE INEDIT.

```
MODE     = 1 FOR RETRAN OR RESTRT              (INPUT)
         = 2 FOR REEDIT
         = 3 FOR PLOTER
Y        = SCRATCH ARRAY USED TO STORE INFORMATION DESCRIBING
           THE RELOCATABLE FILES AND VARIABLES IN THE FILES  (OUTPUT)
IX       = COUNTER FOR THE NUMBER OF VARIABLES DESCRIBED (OUTPUT)
IXI      = BEGINNING INDEX IN THE Y SCRATCH ARRAY WHERE THE
           INFORMATION DESCRIBING THE RELOCATIVE FILES AND
           VARIABLES IS STORED     (INPUT)
INDEX1   = AN INDEX IN THE SCRATCH ARRAY Y USED AS A POSITION
           INDICATOR TO STORE INFORMATION DESCRIBING THE RELOC-
           ATABLE FILES                          (INPUT)
JFIL     = AN ARRAY CONTAINING GENERAL HEADINGS. THIS ARRAY IS
           DEFINED IN SUBROUTINE INEDIT    (INPUT)
IUNIT    = AN ARRAY CONTAINING VARIABLE UNITS.  THIS ARRAY IS
           DEFINED IN SUBROUTINE INEDIT    (INPUT)
```

8 210

```
II        = COUNTER FOR THE NUMBER OF FILES DESCRIBED  (OUTPUT)

*****************************************************************

   SUBROUTINE EDINIT (IDXCRD,LEDT,MODE,NOFILS,IDXF,IFIL,JFIL,LFIL
                     ,KFIL,NFIL,MFIL,DD)


SUBROUTINE EDINIT READS AND CHECKS MINOR EDIT REQUEST OR PLOT
VARIABLE REQUEST CARDS, PLUS FINISHES SETTING UP THE DATA RECORD
DESCRIPTION FILE (RELOCATABLE FILE 43) AND THE ABBREVIATED FILE
DESCRIPTION.

EDINIT IS CALLED BY SUBROUTINE INEDIT.

IDXCRD    = INDEX OF THE DATA CARD ARRAY     (INPUT)
LEDT      = LOGICAL FLAG TO INDICATE IF INPUT EDITING SHOULD BE
            DONE    (OUTPUT)
MODE      = 1 FOR RETRAN OR RSTRT            (INPUT)
          = 2 FOR REEDIT
          = 3 FOR PLOTER
NOFILS    = NUMBER OF FILES IN THE ABBRIEVIATED FILE DESCRIPTION
                                             (INPUT)
IDXF      = AN ARRAY CONTAIN THE RELOCATABLE FILE INDICES FOR EACH
            FILE IN THE DATA RECORD DESCRIPTION.  (INPUT)
IFIL      = AN ARRAY CONTAINING REGION CHECK PARAMETER FOR MINOR
            EDIT AND PLOT REQUESTS.  = -1 MULTIPLE AND SEQUENTIAL
            SETS, => 0 MULTIPLE AND NONSEQUENTIAL SETS, = 0 ONLY
            ONE SET                          (INPUT)
JFIL      = AN ARRAY CONTAINING GENERAL COMPONENT HEADINGS, DEFINED
            IN SUBROUTINE INEDIT             (INPUT)
LFIL      = AN ARRAY CONTAINING THE LENGTH OF A SET FOR EACH FILE
            IN THE DATA RECORD DESCRIPTION  (INPUT)
KFIL      = AN ARRAY CONTAINING THE NUMBER OF SETS IN EACH FILE IN
            THE DATA RECORD DESCRIPTION     (INPUT)
NFIL      = AN ARRAY CONTAINING THE RELOCATABLE FILID FOR EACH FILE
            IN THE DATA RECORD DESCRIPTION  (INPUT)
MFIL      = AN ARRAY INDICATING IF THE RELOCATABLE FILE IN THE DATA
            RECORD DESCRIPTION IS A MAJOR FILE OR A SUBFILE.  0 =
            MAJOR FILE WITHOUT A SUBFILE, 1 = MAJOR FILE WITH A
            SUBFILE, >1 = SUBFILE WITH NFIL EQUAL TO THE OFFSET
            FOR THE REGION CHECK PARAMETER IN THE MAJOR FILE(INPUT)
DD        = THE ARRAY THAT CONTAINS THE EDIT HEADINGS FOR ALL
            VARIABLES IN THE DATA RECORD DESCRIPTION        (INPUT)

*****************************************************************

   SUBROUTINE EDIT (MAJC,MINC,PLTC,LABL,PEND,IUNIT)


SUBROUTINE EDIT IS THE DRIVER FOR MAJOR AND MINOR EDITING.
```

EDIT IS CALLED BY SUBROUTINES TRAN AND EDITRE.

```
MAJC     = MAJOR EDIT FLAG                          (INPUT)
MINC     = MINOR EDIT FLAG                          (INPUT)
PLTC     = DATA TAPE WRITE FLAG                     (INPUT)
LABL     = DATA TAPE LABEL WRITE FLAG               (INPUT)
PEND     = END OF RUN FLAG                          (INPUT)
IUNIT    = UNIT NUMBER GIVEN TO A RESTART TAPE      (INPUT)
```

****

SUBROUTINE EDITRE

SUBROUTINE EDITRE CONTROLS THE DATA TAPE EDITING PROCESS FOR MODULE REEDIT.

EDITRE IS CALLED FROM SUBROUTINE REEDIT.

****

SUBROUTINE EDTCND (ISUBC,IVSLOL)

SUBROUTINE EDTCND EDITS CONDUCTOR PARAMETERS FOR MAJOR EDITS.

EDTCND IS CALLED BY SUBROUTINE EDIT.

```
ISUBC    = OFFSET USED TO PICK UP TEMP. DIST. SUBFILE   (OUTPUT)
OVSLOL   = OFFSET TO OBTAIN VARIABLE IVSL FROM HEAT CONDUCTOR
           FILE   (OUTPUT)
```

****

SUBROUTINE EDTCOR (ISUBCR,IVSLOL)

SUBROUTINE EDTCOR EDITS CORE CONDUCTOR PARAMETERS FOR MAJOR EDITS.

EDTCOR IS CALLED BY SUBROUTINE EDIT.

```
ISUBCR   = OFFSET USED TO PICK UP TEMP. DIST. SUBFILE   (INPUT)
IVSLOL   = OFFSET USED TO OBTAIN VARIABLE IVSL FROM HEAT CONDUCTOR
           FILE                                        (INPUT)
```

****

SUBROUTINE EDTDNB (NDELT)

SUBROUTINE EDTDNB EDITS AUXILIARY DNB PARAMETERS FOR MAJOR EDITS.

EDTDNB IS CALLED BY SUBROUTINE EDIT.

NDELT = NUMBER OF DNB PARAMETERS EDITED (OUTPUT)

****

SUBROUTINE EDTHTX (IVOL,ISV)

SUBROUTINE EDTHTX EDITS NON-CONDUCTING HEAT EXCHANGER PARAMETERS.

EDTHTX IS CALLED BY SUBROUTINE EDIT.

IVOL = VOLUME FILE INDEX
ISV = VOLUME FILE SETSIZ

****

SUBROUTINE EDTKIN

SUBROUTINE EDTKIN EDITS THE REACTOR KINETICS PARAMETERS.

EDTKIN IS CALLED BY SUBROUTINE EDIT.

****

SUBROUTINE EDTLQL

SUBROUTINE EDTLQL EDITS THE LIQUID LEVEL CALCULATION PARAMETERS FOR MAJOR EDITS.

EDTLQL IS CALLED FROM SUBROUTINE EDIT.

****

SUBROUTINE EDTPMP

SUBROUTINE EDTPMP EDITS CENTRIFUGAL PUMP PARAMETERS FOR MAJOR EDITS.

EDTPMP IS CALLED BY SUBROUTINE EDIT.

****

SUBROUTINE ENERGY (DT,IBRNCH)

SUBROUTINE ENERGY EVALUATES THE VOLUME HEAT SOURCES AND SINKS FOR THE SYSTEM BY CALLING SUBROUTINES FOR THE CONDUCTION SOLUTION

AND/OR NON-CONDUCTING HEAT EXCHANGERS.

ENERGY IS CALLED BY SUBROUTINES TRAN, STSTAT, INITLZ, AND RESOPT.

DT	= RETRAN TIME STEP	(INPUT)
IBRNCH	= 1 FOR STEADY STATE INITIALIZATION	(INPUT)
	= 2 FOR TRANSIENT	(INPUT)

****

SUBROUTINE ENTRAN (IVOL,J,DT,HNEW)

SUBROUTINE ENTRAN CALCULATES JUNCTION ENTHALPY USING ENTHALPY TRANSPORT. JUNCTION ENTHALPY IS CORRECTED FOR HEAT ADDITION FROM VOLUME AVERAGE ENTHALPY.

ENTRAN IS CALLED FROM SUBROUTINE BUBB.

IVOL	= VOLUME FILE INDEX	(INPUT)
J	= JUNCTION FILE INDEX	(INPUT)
DT	= TIME STEP SIZE	(INPUT)
HNEW	= NEW JUNCTION ENTHALPY	(OUTPUT)
	INPUT VALUE IS JUNCTION ENTHALPY AS CALCULATED BY BUBB	

****

SUBROUTINE EQSETS (IDXV,ISZV,MEQS)

SUBROUTINE EQSETS DETERMINES THE NUMBER OF INDEPENDENT EQUATION SETS NECESSARY TO CALCULATE VOLUME ENTHALPIES FOR STEADY STATE INITIALIZATION AND SETS UP FLAGS FOR EACH EQUATION SET.

EQSETS IS CALLED BY SUBROUTINE INITLZ.

IDXV	= INDEX OF A SCRATCH ARRAY USED TO CONTAIN VOLUME PARAMETERS FOR STEADY STATE INITIALIZATION	(INPUT)
ISZV	= SET SIZE OF THE SCRATCH ARRAY	(INPUT)
MEQS	= NUMBER OF EQUATION SETS	(OUTPUT)

****

SUBROUTINE ETSCON

SUBROUTINE ETSCON DOES THE EDITING OF THE TIME STEP CONTROL SUMMARY.

ETSCON IS CALLED BY SUBROUTINE TRAN.

```
**********************************************************************

   SUBROUTINE FAIL

SUBROUTINE FAIL SETS AN ERROR FLAG.

**********************************************************************

   FUNCTION  FANG (G,D,VIS,IFAN)

FUNCTION FANG COMPUTES THE FANNING FRICTION FACTOR AS A FUNCTION
OF REYNOLDS NUMBER.

FANG IS CALLED BY SUBROUTINE STPM.

G        = MASS FLUX                          (INPUT)
D        = EQUIVALENT FLOW AREA DIAMETER      (INPUT)
VIS      = FLUID VISCOSITY                    (INPUT)
IFAN     = MEMORY INDEX FOR FANNING CALCULATION  (INPUT) (OUTPUT)

**********************************************************************

   SUBROUTINE FCOLDW (GCORE,PR,FCW,QUAL)

SUBROUTINE FCOLDW COMPUTES FACTOR FOR COLD WALL EFFECT.

FCOLDW IS CALLED BY DNBM.

GCORE    = MASS FLUX          (INPUT)
PR       = PRESSURE           (INPUT)
FCW      = FACTOR FOR COLD WALL EFFECT     (OUTPUT)
QUAL     = QUALITY            (INPUT)

**********************************************************************

   SUBROUTINE FGRSPM(FS,GCORE,TDC)

SUBROUTINE FGRSPM COMPUTES THE GRID SPACER MIXING EFFECT ON CHF.

FGRSPM IS CALLED BY SUBROUTINE DNBM.

FS    =  GRID SPACER MIXING FACTOR                    (OUTPUT)
SCORE =  LOCAL MASS FLUX                              (INPUT)
TDC   =  THERMAL DIFFUSION COEFFICIENT                (INPUT)

**********************************************************************

   FUNCTION FILL(I)
```

FUNCTION FILL INTERPOLATES IN THE FILL TABLES TO OBTAIN FLOW, PRESSURE AND ENTHALPY FOR POSITIVE FILLS AND FLOW FOR NEGATIVE FILLS. FLUID STATE PROPERTIES ARE ALSO OBTAINED FOR POSITIVE FILLS VIA A CALL TO STH2O5.

FILL IS CALLED BY SUBROUTINES FLOSRH, BUBINT, INITLZ, JUNMP AND RESOPT.

I = JUNCTION RELOCATABLE FILE INDEX. (INPUT)
FILL = JUNCTION MASS FLOW RATE (OUTPUT)

****

SUBROUTINE FITHT

SUBROUTINE FITHT COMPUTES CORE POWER PROFILE SHAPE BASED ON CORE NODE VOLUMES.

FITHT IS CALLED BY SUBROUTINE DNBM.

****

SUBROUTINE FLOSRH (DT)

SUBROUTINE FLOSRH COMPUTES TERMS FOR THE MOMENTUM EQUATION AND MAKES A PREDICTIVE CALCULATION OF CHOKED FLOW.

FLOSRH IS CALLED BY SUBROUTINE TRAN.

DT = RETRAN TIME STEP SIZE (INPUT)

****

SUBROUTINE FRICTN (BF,BR,RA1,RAK,RA2,MVN)

SUBROUTINE FRICTN COMPUTES FORWARD AND REVERSE GEOMETRIC FRICTION COEFFICIENTS FOR SHARP EDGE AREA CHANGES. TWO AREA CHANGES ARE CONSIDERED FROM THE SOURCE VOLUME TO THE JUNCTION AND THEN FROM THE JUNCTION TO THE SINK VOLUME.

FRICTN IS CALLED BY SUBROUTINES PREW AND MIXFLO.

BF = FORWARD FRICTION MULTIPLIER BASED ON JUNCTION AREA (OUTPUT)
BR = REVERSE FRICTION MULTIPLIER BASED ON JUNCTION AREA (OUTPUT)
RA1 = RECIPROCAL INLET AREA (INPUT)
RAK = RECIPROCAL JUNCTION AREA (INPUT)
RA2 = RECIPROCAL OUTLET AREA (INPUT)

```
MVN       = FLAG TO USE ZUBER EQUATION FORM = 4 (INPUT)
```

****

```
   SUBROUTINE HAVG (IDXV,ISZV,MEQS)

SUBROUTINE HAVG COMPUTES VOLUME AVERAGE ENTHALPIES
REQUIRED FOR STEADY STATE FLOW WITH ENERGY ADDITIONS
- SOLVES C*HBAR = G.  HAVG IS USED FOR STEADY
STATE INITIALIZATION ONLY.

HAVG IS CALLED BY SUBROUTINES INITLZ AND STSTAT.

IDXV      = INDEX OF A SCRATCH FILE NEEDED FOR STEADY STATE VOLUME
            VARIABLES                                  (INPUT)
ISZV      = SETSIZ OF VOLUME SCRATCH                   (INPUT)
MEQS      = NUMBER OF EQUATION SETS                    (INPUT)
```

****

```
   SUBROUTINE HEADC

SUBROUTINE HEADC COMPUTES HEAD TERMS - EITHER VOLUME
GEOMETRIC CENTER OR CENTER OF MASS COORDINATES
MAY BE USED.  FOR GEOMETRIC CENTER CALCULATION
CMAS = HALF*ZVOL.

HEADC IS CALLED BY SUBROUTINE PREW.
```

****

```
   SUBROUTINE HTRC (AA,BB,TSUR,ISIDE,LL,II)


SUBROUTINE HTRC PROVIDES THE SURFACE TEMPERATURE AND
SURFACE FLUX BOUNDARY CONDITIONS FROM A GENERALIZED
BOILING CURVE FOR THE HEAT CONDUCTION SOLUTION.

HTRC IS CALLED BY SUBROUTINES TEMP AND TEMZ.

AA        = COEFFICIENTS OF THE CONDUCTION SOLUTION (INPUT)
BB        = IN THE FORM Q=AA*TSUR+BB                (INPUT)
TSUR      = SURFACE TEMPERATURE                     (INPUT/OUTPUT)
ISIDE     = HEAT CONDUCTOR SIDE                     (INPUT)
          = RIGHT SIDE
          = 2 LEFT SIDE
LL        = HEAD CONDUCTOR FILE INDEX               (INPUT)
II        = 0 FOR INITIALIZATION                    (INPUT)
          = 1 FOR TRANSIENT
```

8 217

```
**********************************************************************

   FUNCTION HTXQ (I,PTHERM,DELT,WAVE,TEMP,QLOSS,HE,TIMEZ)

FUNCTION HTXQ COMPUTES THE HEAT REMOVAL OR ADDITION
VIA A NON-CONDUCTING HEAT EXCHANGER.

HTXQ IS CALLED BY SUBROUTINE ENERGY.

I        = HEAT EXCHANGER INDEX NUMBER      (INPUT)
PTHERM   = THERMAL POWER                    (INPUT)
DELT     = TIME STEP SIZE                   (INPUT)
WAVE     = VOLUME AVERAGED FLOW             (INPUT)
TEMP     = VOLUME TEMPERATURE               (INPUT)
QLOSS    = TOTAL RATE OF HEAT REMOVAL       (INPUT/OUTPUT)
HE       = ENERGY EXTRACTED BY HEAT EXCHANGER   (OUTPUT)
TIMEZ    = TRANSIENT TIME (SEC)             (INPUT)
HTXQ     = HEAT REMOVED                     (OUTPUT)

**********************************************************************

   SUBROUTINE IMIXCK


SUBROUTINE IMIXCK CHECKS USER INPUT OF THE MIXING INDEX, MVMIX,
AND DETERMINES NEW ARRAYS IJ AND IK FOR JUNCTIONS THAT MIX
MOMENTUM.
      MVMIX(I) = 0, COMPRESSIBLE SINGLE STREAM FLOW, NO MIXING
   IF MVMIX(I) = 1, JUNCTION I MOMENTUM IS MIXED WITH ANOTHER
                    JUNCTION IN THE "FROM" VOLUME
      MVMIX(I) = 2, JUNCTION I IS MIXED IN THE "TO" VOLUME
      MVMIX(I) = 0, INCOMPRESSIBLE SINGLE STREAM FLOW
ONLY TWO FLOW RATHES CAN BE MIXED AND THEN ONLY ON ONE SIDE
   (EITHER THE VOLUME "FROM" OR THE VOLUME "TO"). MIXED FLOWS MUST
   BE MIXED ON EITHER THE INLET OR THE OUTLET SIDE OF A VOLUME
   THEN IJ(I) = J AND IJ(J) = I
THE IK ARRAY IS THE VOLUME NUMBERS FOR MIXING, OUTLET POSITIVE
   AND INLET NEGATIVE
A FILL IS REVERSED AND MIXED WITH ANOTHER JUNCTION
   IJ(J) = -I
IF A FILL JUNCTION IS SINGLE STREAM FLOW BUT REVERSED
   IN DIRECTION THEN INPUT MVMIX AS -2 (FILL)
   AND IJ(I) =IS SET TO -I

IMIXCK IS CALLED BY SUBROUTINE INJUN.

**********************************************************************

   SUBROUTINE INAREA

SUBROUTINE INAREA READS AND CHECKS THE GENERALIZED
```

178 218

TABULAR DATA CARDS (12XXYY).

INAREA IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INBUBL

SUBROUTINE INBUBL READS AND CHECKS THE BUBBLE
RISE MODEL DATA CARDS (06XXX1).

INBUBL IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INCDHT

SUBROUTINE INCDHT READS AND CHECKS THE SPECIFIED
HEAT TRANSFER COEFFICIENT CARDS (15000X).

INCDHT IS CALLED BY SUBROUTINE INHEAT.

****

SUBROUTINE INCKV

SUBROUTINE INCKV READS AND CHECKS VALVE DATA CARDS
(11XXX0).

INCKV IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INCNST (NOGO)

SUBROUTINE INCNST READS AND CHECKS THE PROBLEM CONTROL
AND CONSTANTS DATA CARD (010005).

INCNST IS CALLED BY SUBROUTINE INTRAN.

NOGO = ERROR FLAG (INPUT/OUTPUT)

****

SUBROUTINE INCNT1

SUBROUTINE INCNT1 READS ALL THE 70XXXX CARDS USED TO SPECIFY
THE CONTROL SYSTEM MODEL, SETS UP FILES 53 AND 54, AND CALCULATES

INDICES NEEDED BY TRANSIENT CALCULATIONS.

INCNT1 IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INCNT2

SUBROUTINE INCNT2 CALCULATES ADDRESSES RELATIVE TO ISTOR(1) WHICH COULD NOT BE DONE BY INCNT1. THESE ADDRESSES ARE THE ADDRESSES OF VARIABLES TO BE MONITORED BY THE CONTROL SYSTEM AND THE ADDRESSES OF AREA VS. TIME TABLES USED BY THE FNG AND FN2 CONTROL BLOCKS.

INCNT2 IS CALLED BY SUBROUTINE INEDIT.

****

SUBROUTINE INCORE

SUBROUTINE INCORE READS AND CHECKS THE CORE SECTION DATA CARDS (16XXX0).

INCORE IS CALLED BY SUBROUTINE INHEAT.

****

SUBROUTINE INDATA (IUNIT,JUNIT)

SUBROUITNE INDATA COPIES HEADER LABEL AND PROBLEM DESCRIPTION DATA FROM IUNIT TO JUNIT AND STORES THE ORIGINAL PROBLEM INPUT DATA IN FILE ID 1.0. THIS SUBROUTINE IS USED FOR RESTART RUNS ONLY.

INDATA IS CALLED BY SUBROUTINE INRSTR.

IUNIT = UNIT TO READ DATA FROM (INPUT)
JUNIT = UNIT TO WRITE DATA TO (INPUT)

****

SUBROUTINE INDNB

SUBROUTINE INDNB READS AND CHECKS THE DNBR DATA CARDS (8001XX,8002XX,8003XX, AND 8004XX).

INDNB IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INEDIT (IDXCRD,LEDT,MODE,ITYP,KUNIT)

SUBROUTINE INEDIT IS THE DRIVER FOR CREATION OF THE DATA RECORD DESCRIPTION FILE (RELOCATABLE FILE 43).

INEDIT IS CALLED BY SUBROUTINES INEDTE, REDTAP, INTRAN.

```
IDXCRD    = INPUT FILE DATA INDEX    (INPUT)
LEDT      = .TRUE. FOR RESTARTING WITH MINOR EDIT REQUESTS.
            OTHERWISE .FALSE.        (OUTPUT)
MODE      = 1 FOR RELAP/E OR RESTRT/E              (INPUT)
          = 2 FOR REDIT/E
          = 3 FOR PLOTR/E
ITYP      = 2 FOR RELAP4/003 DATA TAPES            (INPUT)
          = 3 FOR RELAP4/002 DATA TAPES
          = 4 FOR RELAP3 DTAT TAPES
          = 5 FOR STRANGER DATA TAPES
KUNIT     = TAPE UNIT NUMBER                       (INPUT)
```

****

SUBROUTINE INEDTE

SUBROUTINE INEDTE READS AND CHECKS INPUT DATA FOR THE REEDIT MODULE.

INEDTE IS CALLED BY SUBROUTINE REEDIT.

****

SUBROUTINE INFILL

SUBROUTINE INFILL READS AND CHECKS THE FILL DATA CARDS (13XXYY).

INFILL IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INGEOM (L,IDXTGM,IG,NR,IW,IDXR,IDXS,IDXVP,RPOI,SL, SR,IR,M)

SUBROUTINE INGEOM READS AND CHECKS HEAT CONDUCTOR GEOMETRY CARDS (17XXYY).

INGEOM IS CALLED BY SUBROUTINE INHEAT.

```
L         = GEOMETRY NUMBER          (INPUT)
IDXTGM    = INDEX OF TEMPORARY BUFFER FILE FOR READING IN GEOMETRY
            DATA                     (INPUT)
IG        = GEOMETRY TYPE            (OUTPUT)
```

```
NR        = NUMBER OF REGIONS      (OUTPUT)
IW        = EDIT FLAG              (INPUT)
IDXS      = INDEX OF FILE CONTAINING SURFACE AREA WEIGHT FOR THIS
            CONDUCTOR GEOMETRY     (INPUT)
IDXR      = INDEX OF FILE CONTAINING GEOMETRY REGION VARIABLES
            (INPUT)
IDXVP     = INDEX OF FILE CONTAINING NODAL VOLUMES FOR THIS
            CONDUCTOR GEOMETRY     (INPUT)
RROI      = INITIAL FUEL ROD RADIUS     (OUTPUT)
SL        = LEFT CONDUCTOR SURFACE AREA PER UNIT HEIGHT (OUTPUT)
SR        = RIGHT CONDUCTOR SURFACE AREA PER UNIT HEIGHT (OUTPUT)
IR        = INPUT ERROR INDICATOR       (OUTPUT)
M         = NODE AT RIGHT SURFACE       (OUTPUT)

**********************************************************************

   SUBROUTINE INHEAT

SUBROUTINE INHEAT IS ESSENTIALLY THE DRIVER FOR
PROCESSING HEAT CONDUCTOR INPUT DATA.

INHEAT IS CALLED BY SUBROUTINE INTRAN.

**********************************************************************

   SUBROUTINE INHTXQ


SUBROUTINE INHTXQ READS AND CHECKS THE NON-
CONDUCTING HEAT EXCHANGER DATA CARDS (21XXYY).

INHTXQ IS CALLED BY SUBROUTINE INTRAN.


**********************************************************************

   SUBROUTINE INIFTE

SUBROUTINE INIFTE RESERVES A FILE FOR THE SCRATCH
ARRAYS USED IN SUBROUTINE NIFTE.

INIFTE IS CALLED BY SUBROUTINE INTRAN.


**********************************************************************

   SUBROUTINE INITLZ (IDXV,ISZV,IDXV,ISZV,MEGS)

SUBROUTINE INITLZ OBTAINS INITIAL GUESSES FOR
STATE PROPERTIES, LOSS COEFFICIENTS, PRESSURES,
ENTHALPIES AND HEAT CONDUCTION FOR STEADY-STATE
INITIALIZATION.  IF STEADY-STATE INITIALIZATION
IS NOT USED THIS SUBROUTINE DIRECTS THE INITIALAZTION OF THE
```

PROBLEM

INITLZ IS CALLED BY SUBROUTINE STSTAT.

IDXJ = INDEX OF A SCRATCH FILE CONTAINING JUNCTION VARAIBLES
USED FOR STEADY-STATE INTIALIZATION (INPUT)
ISZV = SET SIZE OF THE JUNCTION SCRATCH FILE (INPUT)
IDXV = INDEX OF A SCRATCH FILE CONTAINING VOLUME VARIABLES
USED FOR STEADY-STATE INITIALIZATION. (INPUT)
ISZV = SET SIZE OF THE VOLUME SCRATCH FILE (INPUT)
MEQS = NUMBER OF INDEPENDENT ENERGY EQUATION SETS (INPUT)

****

SUBROUTINE INJUN

SUBROUTINE INJUN READS AND CHECKS JUNCTION DATA
CARDS (08XXXY).

INJUN IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INLVC

SUBROUTINE INLVC READS AND CHECKS EQUIVALENT LIQUID
LEVEL VOLUME CALCULATION DATA CARD (060000).

INLVC IS CALLED BY SUBROUTINE INBUBL.

****

SUBROUTINE INMPRO (M,NK,NC,NX,IDXK,IDXC,IDXX)

SUBROUTINE INMPRO READS AND CHECKS, THERMAL HEAT
CAPACITY DATA CARDS (19XXYY), THERMAL CONDUCTIVITY
DATA CARDS (18XXYY), AND LINEAR EXPANSION COEFFICIENT
DATA CARDS (20XXYY).

INMPRO IS CALLED BY SUBROUTINE INHEAT.

M = MATERIAL NUMBER (INPUT)
NK = NUMBER OF ENTRIES FOR THERMAL CONDUCTIVITY TABLE
(OUTPUT)
NC = NUMBER OF ENTRIES FOR HEAT CAPACITY (OUTPUT)
NX = NUMBER OF ENTRIES FOR EXPANSION COEFFICIENT (OUTPUT)
IDXK = FILE INDEX OF THERMAL CONDUCTIVITY TABLE (INPUT)
IDXC = FILE INDEX OF HEAT CAPACITY TABLE (INPUT)
IDXX = FILE INDEX OF THERMAL EXPANSION COEFFICIENT (INPUT)

8 223

****

SUBROUTINE INPLOT

SUBROUTINE INPLOT PROCESSES INPUT DATA AND RESERVES
FILES FOR PLOTTING.

INPLOT IS CALLED BY SUBROUTINE PLOTER.

****

SUBROUTINE INPOWR

SUBROUTINE INPOWR DETERMINES WHAT TYPE OF POWER
GENERATOR MODEL IS DESIRED AND CALLS THE
APPROPRIATE SUBROUTINE TO READ THE CARD DATA.

INPOWR IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INPM

SUBROUTINE INPM READS AND CHECKS PUMP DESCRIPTION
DATA CARDS (090XXY), PUMP HEAD MULTIPLIER DATA CARDS
(091XXY), PUMP TORQUE MULTIPLIER DATA CARDS (092XXY),
PUMP STOP DATA CARDS (095XX1), PUMP MOTOR TORQUE
DATA CARDS (097XXY), AND THE PUMP CURVE SET INPUT
DATA CARD (100000).

INPM IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INPUMP (K,NC,IDXP,IDXBUF,NAME,FILIDS)

SUBROUTINE INPUMP READS THE PUMP HEAD AND TORQUE DATA CARDS
(10XYYZ).

INPUMP IS CALLED BY SUBROUTINE INPM.

K	= PUMP CURVE SET NUMBER	(INPUT)
NC	= NUMBER OF CURVES	(INPUT/OUTPUT)
IDXP	= INDEX OF CURVE SET FILE	(INPUT)
IDXBUF	= INDEX OF FILE TO READ DATA	(INPUT)
NAME	= ARRAY TO PASS PUMP IDENTIFICATION FOR EDIT	(OUTPUT)
FILIDS	= ARRAY TO PASS PUMP FILE ID'S	(OUTPUT)

****

SUBROUTINE INPUT (FIRST,LSST,NCASE,IGO)

8 224

SUBROUTINE INPUT READS THE PROBLEM DIMENSION CARD
(01000Y). INPUT ALSO INITIALIZES FTB AND SETS UP THE
CARD DATA ARRAY VIA A CALL TO INP.

INPUT IS CALLED BY SUBROUTINE RMAIN.

```
FIRST    = FIRST PROGRAM FLAG      (OUTPUT)
LSST     = LAST PROGRAM FLAG       (OUTPUT)
NCASE    = CASE NUMBER             (OUTPUT)
IGO      = PROBLEM TYPE            (OUTPUT)
         = 1 RELAP/E
         = 2 RESTRT/E
         = 3 REDIT/E
         = 4 PLOTR/E
```

****

SUBROUTINE INREAC

SUBROUTINE INREAC READS AND CHECKS THE DENSITY
REACTIVITY TABLE DATA CARDS (1420XX), THE DOPPLER
TABLE DATA CARDS (1430XX), AND THE REACTIVITY COEFFICIENT
DATA CARDS (140XX0).

INREAC IS CALLED BY SUBROUTINE INRKEN.

****

SUBROUTINE INRKEN

SUBROUTINE INRKEN READS AND CHECKS THE KINETICS CONSTANTS
DATA CARDS (140000) AND THE DELAY NEUTRON DATA
OVERIDE CARD (140001)

INRKEN IS CALLED BY SUBROUTINE INPOWR.

****

SUBROUTINE INRSTR (IUNIT,JUNIT)

THIS SUBROUTINE DIRECTS INPUT AND INITIALIZATION OF RESTRI
PROBLEMS. TWO INPUT DATA SETS ARE USED, THE MINIMAL
RESTART DECK SUPPLIED BY THE USER AND THE ORIGINAL DATA
STORED ON THE PLOT TAPE TO BE USED FOR RESTART.

INRSTR IS CALLED BY SUBROUTINE RETRAN.

```
IUNIT    = TAPE UNIT NUMBER OF RESTART DATA TAPE     (INPUT)
JUNIT    = TAPE UNIT NUMBER OF WHICH NEW DATA TAPE IS WRITTEN
```

****

SUBROUTINE INRTRN

SUBROUTINE INRTRN CHECKS AND EDITS THE PROBLEM DIMENSION DATA AND READS THE LABEL FOR A PLOT TAPE IF ONE IS SUPPLIED (CARD 010040).

INRTRN IS CALLED BY SUBROUTINE RETRAN.

****

SUBROUTINE INSCRM

SUBROUTINE INSCRM READS AND CHECKS THE SCRAM TABLE DATA CARDS (141XYY).

INSCRM IS CALLED BY SUBROUTINES INPOWR, AND INRKEN.

****

SUBROUTINE INSLAB

SUBROUTINE INSLAB READS, CHECKS AND EDITS THE HEAT CONDUCTOR DATA CARDS (15XXXY).

INSLAB IS CALLED BY SUBROUTINE INHEAT.

****

SUBROUTINE INSTGN

SUBROUTINE INSTGN READS AND CHECKS THE STEADY-STATE INITIALIZATION CONVERGENCE CRITERIA DATA CARD (230000), AND THE STEADY-STATE POWER REMOVAL SYSTEM DATA CARDS (230XXY).

INSTGN IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INTEG(INPUT,OUTPUT,GAIN)

SUBROUTINE INTEG MODELS A RECTANGULAR INTEGRATOR.

INTEG IS CALLED BY SUBROUTINE CONTRL.

```
INPUT  = VALUE OF INPUT SIGNAL AT THE CURRENT TIME STEP    (INPUT)
OUTPUT = VALUE OF OUTPUT SIGNAL AT THE CURRENT TIME STEP (OUTPUT)
GAIN   = INTEGRATOR GAIN                                   (INPUT)
```

****

SUBROUTINE INTRAN (NOGO)

SUBROUTINE INTRAN DIRECTS THE CALLING OF INPUT PROCESSING SEGMENTS FOR RETRAN AND RESTRT MODULES.

INTRAN IS CALLED BY SUBROUTINE INRTRN AND INRSTR.

```
NOGO     = SYSTEM ERROR FLAG      (OUTPUT)
```

****

SUBROUTINE INTRIP (IDXCRN,LTRP,LRST)

SUBROUTINE INTRIP READS AND CHECKS THE TRIP CONTROL DATA CARDS (04XXX0).

INTRIP IS CALLED BY SUBROUTINE INTRAN.

```
IDXCRN = INDEX OF THE FILE CONTAINING INPUT DATA CARDS      (INPUT)
LTRP   = .TRUE. IF RESTART WITH NEW TRIPS SUPPLIED
         OTHERWISE .FALSE.                                  (INPUT)
LRST   = .TRUE. IF RESTART RUN, OTHERWISE .FALSE.           (INPUT)
```

****

SUBROUTINE INTSTP (IDXCRD,LTRP,LRST)

SUBROUTINE INTSTP READS AND CHECKS THE TIME STEP DATA CARDS (03XXX0).

INTSTP IS CALLED BY SUBROUTINE INTRAN.

```
IDXCRD = FILE INDEX OF ARRAY CONTAINING INPUT DATA          (INPUT)
LRST   = .TRUE. IF A RESTART RUN WITH NEW TIME STEPS
         SUPPLIES, OTHERWISE .FALSE.                        (INPUT)
LRST   = .TRUE. IF RESTART RUN, OTHERWISE .FALSE.           (INPUT)
```

****

SUBROUTINE INTV

SUBROUTINE INTV READS AND CHECKS THE TIME DEPENDENT VOLUME DATA CARDS (07XXXY) AND THE BOUNDARY CONDITION TAPE REQUEST DATA CARDS (01002Y).

INTV IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE INVOL

SUBROUTINE INVOL READS AND CHECKS VOLUME DATA CARDS (05XXXY). THIS SUBROUTINE ALSO OBTAINS INITIAL STATE PROPERTIES FOR VOLUME THERMODYNAMIC CONDITIONS DESCRIBED WITH ANYTHING OTHER THAN A PRESSURE AND ENTHALPY ENTRY.

INVOL IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE IPMCK

SUBROUTINE IPMCK PERFORMS FURTHER CHECKING ON THE PUMP INPUT DATA PLUS INITIALIZES PARAMETERS NEEDED BY THE PUMP MODEL.

IPMCK IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE JHOFF

SUBROUTINE JHOFF COMPUTES THE ENTHALPY BIAS (FOR FILL JUNCTIONS FLAGGED AS BIAS JUNCTIONS) SUCH THAT DU/DT IS ZERO. THIS SUBROUTINE IS USED FOR STEADY-STATE INITIALIZATION ONLY.

JHOFF IS CALLED BY SUBROUTINE STSTAT.

****

SUBROUTINE JUNHP(DT,IBH1)

SUBROUTINE JUNHP PERFORMS TWO FUNCTIONS, IF IBH1 = 2 JUNHP DEFINES JUNCTION ENTHALPY AND QUALITY AND COMPUTES DU/DM, IF IBH1 = 1 JUNHP DOES THE VOLUME BUBBLE MASS INTEGRATION.

JUNHP IS CALLED BY SUBROUTINES BAL AND PREW.

8 228

```
DT      = RETRAN TIME STEP                                          (INPUT)
IBH1    = 2 FOR JUNCTION ENTHALPY CALCULATION, = 1 FOR
          BUBBLE MASS INTEGRATION                                   (INPUT)
```

****

```
   SUBROUTINE JVEDIT (IDXS,ISZS,IDXB)

SUBROUTINE JVEDIT EDITS ACTUAL JUNCTION, VOLUME, HEAT
CONDUCTOR AND BUBBLE RISE DATA ACTUALLY BEING USED
AFTER INITIALIZATION OF THE PROBLEM.

JVEDIT IS CALLED BY SUBROUTINE STSTAT.

IDXS    = INDEX OF A SCRATCH FILE USED TO STORE HEAT CONDUCTOR
          DATA FOR STEADY-STATE INITIALIZATION                      (INPUT)
ISZS    = SET SIZE OF THE SCRATCH FILE CONTAINING HEAT
          CONDUCTOR DATA.                                           (INPUT)
IDXB    = INDEX OF A SCRATCH FILE USED TO STORE BUBBLE RISE
           DATA FOR STEADY-STATE INITIALIZATION                     (INPUT)
```

****

```
   SUBROUTINE KINITL


SUBROUTINE KINITL COMPUTES INITIAL CONDITIONS FOR THE POINT
KINETICS MODEL.

KINITL IS CALLED BY SUBROUTINE STSTAT.
```

****

```
   SUBROUTINE LABLCK (LABL,NC,IBMRUN)

SUBROUTINE LABLCK DETERMINES HOW MANY CHARACTERS ARE IN
THE AXIS LABEL FOR PLOTTING AND PACKS LABEL CHARACTERS
INTO A10 FORMAT FOR CDC MACHINES.

LABLCK IS CALLED BY SUBROUTINE PLOTR.

LABL    = ARRAY CONTAINING AXIS LABEL                (INPUT/OUTPUT)
NC      = NUMBER OF CHARACTERS IN THE LABEL          (OUTPUT)
IBMRUN = .TRUE. IF RUN ON IBM, .FALSE. IF CDC        (INPUT)
```

****

```
   SUBROUTINE LAG (INPUT,OUTPUT,TAU,GAIN)
```

```
SUBROUTINE LAG MODELS A LAG COMPENSATION NETWORK, WHICH HAS
A LAPLACE TRANSFORM OF 1/(1+S*TAU).

LAG IS CALLED BY SUBROUTINE CONTRL.

INPUT  = VALUE AT INPUT SIGNAL AT THE CURRENT TIME STEP     (INPUT)
OUTPUT = VALUE OF THE OUTPUT SIGNAL AT THE CURRENT TIME STEP
         (OUTPUT)
TAU    = LAG TIME CONSTANT                                  (INPUT)
GAIN   = GAIN OF THE LAG BLOCK                              (INPUT)

*****************************************************************

   SUBROUTINE LDLAG (INPUT,OUTPUT,TLEAD,TLAG,GAIN,BOX,INDEX)

SUBROUTINE LDLAG MODELS A LEAD-LAG COMPENSATION NETWORK, WHICH
HAS A LAPLACE TRANSFORM OF (1+TLEAD*S)/(1+TLAG*S).

LDLAG IS CALLED BY SUBROUTINE CONTRL.

INPUT  = VALUE AT INPUT SIGNAL AT THE CURRENT TIME STEP     (INPUT)
OUTPUT = VALUE OF THE OUTPUT SIGNAL AT THE CUTTENT TIME STEP
         (OUTPUT)
TLEAD  = LEAD TIME CONSTANT                                 (INPUT)
TLAG   = LAG TIME CONSTANT                                  (INPUT)
GAIN   = GAIN OF NETWORK
BOX    = STORAGE ARRAY REPRESENTING STATE OF THE
         LEAD LAG ELEMENT                          (INPUT/OUTPUT)
INDEX  = ON ENTRY TO SUBROUTINE, INDEX IS THE ADDRESS
         OF BOX(1) RELATIVE TO BOX (1), ARRAY IN CALLING
         SUBROUTINE CONTRL BEFORE RETURNING TO CONTRL, INDEX
         IS INCREASED BY AN AMOUNT EQUAL TO THE LENGTH OF THE
         BOX ARRAY                                 (INPUT/OUTPUT)

*****************************************************************

   FUNCTION  LEAK (H,P,I,DGDP,ICHOK,ITYPE)
FUNCTION LEAK RETURNS TABLE VALUE OF MASS FLUX GIVEN SOURCE
ENTHALPY, PRESSURE AND CHOKING TYPE INDICATOR.

LEAK IS CALLED BY SUBROUTINE FLOSRH

H        = SOURCE ENTHALPY          (INPUT)
P        = SOURCE PRESSURE          (INPUT)
I        = JUNCTION NUMBER          (INPUT)
DGDP     = CHANGE IN MAXIMUM FLOW RATES PER UNIT CHANGE IN
           STAGNATION PRESSURE      (OUTPUT)
ICHOK    = LIQUID PHASE CHOKING CONDITION     (INPUT)
ITYPE    = CHOKING TYPE INDICATOR             (OUTPUT)
LEAK     = TABLE VALUE OF MASS FLUX (LBM/FT2 - SEC)     (OUTPUT)
```

8 230

****

SUBROUTINE LEVCAL

SUBROUTINE LEVCAL PERFORMS THE LIQUID LEVEL CALCULATION.
LEVCAL SUMS LIQUID MASSES OVER SELECTED VOLUMES TO COMPUTE
AN EQUIVALENT HEIGHT.

LEVCAL IS CALLED BY SUBROUTINES BAL AND RESOPT.

****

SUBROUTINE LOCEND

THIS IS A DUMMY SUBROUTINE USED ONLY TO FIND LOAD POINTS.

LOCEND IS CALLED BY SUBROUTINE RESOPT.

****

SUBROUTINE LOCMD1 (IAD)

SUBROUTINE LOCMD1 IS USED TO OBTAIN THE BEGINNING ADDRESS FO
SEGMENT LOCMD1.

LOCMD1 IS CALLED BY SUBROUTINES INDATA AND INRTRN.

IAD = BEGINNING ADDRESS OF SEGMENT LOCMD1 (OUTPUT)

****

SUBROUTINE LOCMD2(IAD)

SUBROUTINE LOCMD2 IS USED TO OBTAIN THE BEGINNING ADDRESS
OF SEGMENT LOCMD2.

LOCMD2 IS CALLED BY SUBROUTINE REEDIT.

IAD = BEGINNING OF SEGMENT LOCMD1 (OUTPUT)

****

SUBROUTINE LOCMD3(IAD)

SUBROUTINE LOCMD3 IS USED TO OBTAIN THE BEGINNING
ADDRESS OF SEGMENT LOCMD3.

LOCMD3 IS CALLED BY SUBROUTINE PLOTER.

8 231

```
IAD     = BEGINNING ADDRESS OF SEGMENT LOCMD3                    (OUTPUT)
```

****

SUBROUTINE LOCTBL

THIS SUBROUTINE IS USED TO LOCATE THE BEGINNING OF THE LEVEL 2 LOADER TABLE.

LOCTBL IS CALLED BY SUBROUTINES RMAIN AND RESSEG.

****

SUBROUTINE LOOPS (NLOOP)

SUBROUTINE LOOPS DETERMINES THE NUMBER OF INDEPENDENT FLOW NETWORKS AND LINKS THEIR ASSOCIATED HEAT SOURCES AND SINKS TO THEM.

LOOPS IS CALLED FROM SUBROUTINE INITLZ.

NLOOP = NUMBER OF INDEPENDENT FLOW NETWORKS (OUTPUT)

****

SUBROUTINE MACH (W,RA1,RAK,RA2,V1,VK,N2,C2,R1,RK,R2,JK,KK)

SUBROUTINE MACH COMPUTES THE COMPRESSIBLE EFFECT OF AN AREA CHANGE USING AN ISENTROPIC, CONSTANT VELOCITY MODEL.

MACH IS CALLED BY SUBROUTINES PREW AND MIXFLO.

```
W  = JUNCTION FLOW                                              (INPUT
RA1 = RECIPRICAL NORMAL INLET FLOW AREA                         (INPUT)
RAK = RECIPROCAL NORMAL JUNCTION FLOW AREA                      (INPUT)
RA2 = RECIPROCAL NORMAL OUTLET FLOW AREA                        (INPUT)
V1  = SPECIFIC VOLUME, INLET                                    (INPUT)
VK  = SPECIFIC VOLUME, MINIMUM VALUE ON INPUT, JUNCTION
      VALUE ON OUTPUT                                (INPUT/OUTPUT)
V2  = SPECIFIC VOLUME, OUTLET                        (INPUT/OUTPUT)
C2  = SONIC VELOCITY SQUARED                         (INPUT/OUTPUT)
R1  = DENSITY, INLET                                            (OUTPUT)
RK  = DENSITY, MAXIMUM VALUE ON INPUT, JUNCTION
      VALUE ON INPUT                                 (INPUT/OUTPUT)
R2  = DENSITY VOLUME, OUTLET                                    (OUTPUT)
JK  = SONIC CHOKING FLAG                                        (OUTPUT)
KK  = TABLE MEMORY INDEX                             (INPUT/OUTPUT)
```

****

SUBROUTINE MASBAL (IDXV,ISZV)

8 232

SUBROUTINE MASBAL COMPUTES JUNCTION MASS FLOWS NOT SUPPLIED
VIA USER INPUT AND CHECKS TO INSURE A STEADY STATE MASS
FLOW BALANCE.

MASBAL IS CALLED BY SUBROUTINE INITLZ.

IDXV = INDEX OF A SCRATCH ARRAY CONTAINING VOLUME QUANTITIES
FOR STEADY STATE INITIALIZATION (INPUT)
ISZV = SET SIZE OF THE VOLUME SCRATCH ARRAY (INPUT)

****

SUBROUTINE MH2OR (AVEX,CLTI,CTR,CTRL,GASM,QMWR,RLEN,RRO,DT,TS)

SUBROUTINE MH2OR SETS ARGUMENTS FOR SUBROUTINE MWR AND CONVERTS
THE OUTPUT FROM MWR TO A GENERATION RATE.

MH2OR IS CALLED BY SUBROUTINES SLABHT AND RESOPT.

AVEX = AVERAGE QUALITY (INPUT)
CLTI = INITIAL CLAD THICKNESS, FT. (INPUT)
CTR = DEPTH REACTION HAS PENETRATED CLADDING AT END OF ANY
TIME STEP, FT. (INPUT/OUTPUT)
CTRL = DEPTH REACTION HAS PENETRATED CLADDING AT END OF ANY
TIME STEP, FT. (OUTPUT)
GASM = GAS AMSS (INCLUDES BUBBLE MASS) (INPUT)
QMWR = HEAT GENERATED PER FOOT OF ROD, BUT/SEC. (INPUT)
RLEN = ROD LENGTH (INPUT)
RRO = ORIGINAL FUEL ROD RADIUS, FT. (INPUT)
DT = TIME STEP, SEC. (INPUT)
TS = CALCULATED SURFACE TEMPERATURE, DEC. F (INPUT)

****

SUBROUTINE MIXFLO (I,W,AI,BW,C2,JC,VE,VI,HKE,PST,RAI,RAK,RAL,
RVI,VI1,VI2,RVI1,RVI2,WMIX,FKGEOM,IRET)
SUBROUTINE MIXFLO PERFORMS THE MOMENTUM MIXING CALCULATIONS.

MIXFLO IS CALLED BY SUBROUTINE PREW.

I = JUNCTION INDEX (INPUT)
W = FLOW (INPUT)
AI = JUNCTION AREA (INPUT)
BW = FANNING FRICTION COEFFICIENT OF W**2 (OUTPUT)
C2 = MACH NUMBER SQUARED AT JUNCTION (INPUT/OUTPUT)
JC = SONIC CHOKING FLAG (INPUT)
VE = MOMENTUM (OUTPUT)
VI = SPECIFIC VOLUME (INPUT/OUTPUT)
HKE = CONVERSION FACTOR (INPUT)
PST = MIXED FLOW JUNCTION KINETIC ENERGY (OUTPUT)

```
RAI       = RECIPROCAL OF JUNCTION AREA                          (INPUT)
RAK       = RECIPROCAL OF VOLUME FLOW AREA ON "FROM" SIDE        (INPUT)
RAL       = RECIPROCAL OF VOLUME FLOW AREA ON "TO" SIDE          (INPUT)
RVI       = RECIPROCAL OF SPECIFIC VOLUME IN JUNCTION            (INPUT)
VI1       = SPECIFIC VOLUME AT JUNCTION INLET                (INPUT/OUTPUT)
VI2       = SPECIFIC VOLUME AT JUNCTION OUTLET               (INPUT/OUTPUT)
RVI1      = RECIPROCAL OF SPECIFIC VOLUME AT JUNCTION INLET      (INPUT)
RVI2      = RECIPROCAL OF SPECIFIC VOLUME AT JUNCTION OUTLET     (INPUT)
WMIX      = MIXED FLOW                (OUTPUT)
FKGEOM    = CALCULATION TYPE FLAG     (INPUT)
IRET      = RETURN FLAG               (OUTPUT)
```

****

```
   SUBROUTINE MWR (DRP,DRP1,QMWR,T,MSTA,DT,RO,CT)
```

SUBROUTINE MWR COMPUTES METAL WATER REACTION HEAT AND THE DEPTH THE METAL WATER REACTION HAS PENETRATED THE CLADDING.

MWR IS CALLED BY SUBROUTINE MH2OR.

```
DRP       = DEPTH REACTION HAS PENETRATED CLADDING     (INPUT)
            AT START OF ANY TIME STEP, INCHES
DRP1      = DEPTH REACTION HAS PENETRATED CLADDING     (OUTPUT)
            AT END OF ANY TIME STEP, INCHES
QMWR      = HEAT GENERATED BY M-W REACTION PER INCH    (OUTPUT)
            OF ROD, BTU/SEC
T         = SURFACE TEMPERATURE, DEG. RANKINE          (INPUT)
MSTA      = AMOUNT OF STEAM AVAILABLE FOR METAL-WATER
            REACTION IN A CORE REGION ON A PER INCH
            OF ROD BASIS                                 (INPUT)
DT        = TIME STEP, SEC                             (INPUT)
RO        = ORIGINAL FUEL PIN RADIUS, INCHES           (INPUT)
CT        = INITIAL CLAD THICKNESS, INCHES             (INPUT)
```

****

```
   SUBROUTINE NIFTE(DT,C,NSZ1,G,BA,XALPH,XGAM,OLDWP)
```

SUBROUTINE NIFTE NUMERICALLY INTEGRATES THE TRANSIENT EQUATIONS. THE FALSH-4 METHOD IS USED. THE SOLUTION OBTAINED IS FOR NON-CRITICAL JUNCTION FLOW, LEAK FLOW, TOTAL VOLUME ENERGY, TOTAL FLUID MASS, AND TOTAL VOLUME AIR MASS.

NIFTE IS CALLED BY SUBROUTINE FLOSRH.

```
DT        = TIME STEP                                  (INPUT)
C         = TEMPORARY STORAGE                          (INPUT)
NSZ1      = NUMBER OF JUNCTIONS                        (INPUT)
G         = TEMPORARY STORAGE                          (OUTPUT)
BA        = TEMPORARY STORAGE                          (OUTPUT)
XALPH     = TEMPORARY STORAGE                          (OUTPUT)
```

```
XGAM   = TEMPORARY STORAGE                          (OUTPUT)
OLDWP  = NOT USED

**********************************************************************

   SUBROUTINE OVRLYP (IUNIT)
SUBROUTINE OVRLYP OVERLAYS THE CURRENTLY POSITIONED DATA RECORD
ON THE FILES DEFINED IN INTRAN.

OVRLYP IS CALLED BY SUBROUTINE RETRAN.

IUNIT    = UNIT NUMBER OF DATA TAPE FROM WHICH TO RETREIVE OVERLAY
           DATA.        (INPUT)

**********************************************************************

   SUBROUTINE NONUHF (J,L,FC)

SUBROUTINE NONUHF COMPUTES NON-UNIFORM HEAT FLUX FACTOR.

NONUHF IS CALLED BY DNBM.

J        = LEVEL OF HEIGHT INDICATOR          (INPUT)
L        = FLAG FOR W-3 OR B&W                (INPUT)
FC       = NON-UNIFORM HEAT FLUX FACTOR       (OUTPUT)

**********************************************************************

   SUBROUTINE PCHF (G,CLI,DH,DR,HD,I,IVOL,K,L,FV,J,CHF)

SUBROUTINE PCHF OBTAINS CRITICAL HEAT FLUX FROM ONE OF SEVERAL
CORRELATIONS DEPENDING ON INPUT PARAMETERS.

PCHF IS CALLED BY SUBROUTINES SINITL AND SLABHT.

G        = MASS VELOCITY, LB/FT**2/HR                              (INPUT)
CLI      = CORE CHANNEL LENGTH, IN                                 (INPUT)
DH       = HEATED EQUIVALENT DIAMETER, IN                          (INPUT)
DR       = ROD DIAMETER, IN   (POSITIVE)                           (INPUT)
         = NEGATIVE OF PLATE HYDRAULIC DIAMETER, IF  (NEGATIVE)
HD       = HYDRAULIC DIAMETER, FT                                  (INPUT)
I        = INDEX FOR COEFFICIENTS                                  (INPUT)
IVOL     = VOLUME FILE INDEX                                       (INPUT)
K        = INLET JUNCTION INDEX                                    (INPUT)
L        = FLUID VELOCITY, FT/SEC                                  (INPUT)
FV       = FLUID VELOCITY                                          (INPUT)
J        = CORRELATION INDICATOR                                   (INPUT)
           J=0  MODIFIED BARNETT,BARNETT, AND B+W-2 COMBINATION
           J=1  GENERAL ELECTRIC
           J=2  SAVANNAH RIVER
```

8 235

```
CHF     = CRITICAL HEAT FLUX, BTU/FT**2/HR                    (OUTPUT)
```

****

SUBROUTINE PLOTER

SUBROUTINE PLOTER IS THE DRIVER FOR THE RETRAN PLOTTING PACKAGE.

PLOTER IS CALLED BY SUBROUTINE RMAIN.

****

SUBROUTINE PLOTPR (NDATA,X,Y,TITLE,XTITLE,YTITLE)

SUBROUTINE PLOTPR GENERATES PRINTER PLOTS FOR ALL REQUESTED MINOR EDIT VARIABLES.

PLOTPR IS CALLED BY SUBROUTINE PRNPLT.

```
NDATA   = NUMBER OF DATA POINTS                          (INPUT)
X       = INDEPENDENT VARIABLE ARRAY                     (INPUT)
Y       = DEPENDENT VARIABLE ARRAY                       (INPUT)
TITLE   = PLOT TITLE      (80 CHARACTERS MAX.)           (INPUT)
XTITLE  = X-AXIS TITLE    (32 CHARACTERS MAX.)           (INPUT)
YTITLE  = Y-AXIS TITLE    (32 CHARACTERS MAX.)           (INPUT)
```

****

SUBROUTINE PLOTR

SUBROUTINE PLOTR IS THE DRIVER FOR THE ACTUAL PLOTTING AFTER THE DATA HAS BEEN RETRIEVED FROM TAPE.

PLOTR IS CALLED BY SUBROUTINE PLOTER.

****

SUBROUTINE PLTAPE (IUNIT,LABL)

SUBROUTINE PLTAPE IS USED BY PROGRAM MODULE RETRAN AND RESTART TO GENERATE A RETRAN DATA TAPE.

PLTAPE IS CALLED BY SUBROUTINES EDIT AND RESOPT.

```
IUNIT   = TAPE UNIT DATA TAPE IS WRITTEN ON              (INPUT)
LABL    = FLAG TO WRITE TRAILER LABEL AND END OF FILE    (INPUT)
```

1758 236

```
**********************************************************************

   SUBROUTINE PMPDTA (K,IDXP,NAME,FILIDS,NC)

SUBROUTINE PMPDTA CONTAINS THE DATA FOR THE BUILT-IN PUMP CURVES
AND RESERVES THE FILES FOR THE BUILT-IN PUMP USED.

PMPDTA IS CALLED BY SUBROUTINE INPM.

K         = PUMP CURVE INDEX                                    (INPUT)
IDXP      = FILE INDEX OF BUILT IN CURVE SET                    (INPUT)
NAME      = PUMP CURVE NAME FOR EDITING                         (INPUT)
FILIDX    = FILE ID OF BUILT IN CURVE SET                       (OUTPUT)
NC        = FLAG TO INDICATE END OF CORE TO RESERVE FILE FOR BUILT
            IN CURVE SET                                        (INPUT)


**********************************************************************

   FUNCTION   POLATE (XY,XX,NN,KK,IR)

FUNCTION POLATE, GIVEN A VALUE FOR X, INTERPOLATES FOR Y FROM A
TABLE OF PAIRS OF X AND Y.

POLATE IS CALLED BY SUBROUTINES LEAK, KINITL, INMPRO, FANC, MACH
CHKV, FILL, SENG, TKANDC, HTXQ, CONDHT, PUMP, PUMPS, POWRI, CORQ,
REAC, SCRM, LEVCAL, AND CONTRL.

XY        = TABLE OF Y(1),X(1),Y(2),X(2), ... Y(NN),X(NN)   (INPUT)
XX        = THE GIVEN VALUE FOR X                           (INPUT)
NN        = THE NUMBER OF PAIRS OF ENTRIES IN XY            (INPUT)
            (INPUT/OUTPUT)
IR        = 1 IF XY IS A TABLE OF X(1),Y(1),X(2),....,XC(NN),Y(NN)
            (INPUT)
IR        = 2 IF XY IS A TABLE OF Y(1),X(1),Y(2),X(2),....,Y(NN)
            X(NN)
POLATE    = DEPENDENT VARIABLE                                (OUTPUT)


**********************************************************************

   SUBROUTINE POL2 (X,Y,Z,IXT,JXT,XP,YP,ZP,I,J)


SUBROUTINE POL2 IS A ROUTINE TO INTERPOLATE A FUNCTION OF TWO
VARIABLES.

POL2 IS CALLED BY SUBROUTINE TFFM.

X         = TABLE OF FIRST VARIABLE          (INPUT)
Y         = TABLE OF SECOND VARIABLE         (INPUT)
Z         = TABLE OF FUNCTIONAL VALUES       (INPUT)
IXT       = TOTAL NUMBER OF VALUES IN FIRST TABLE     (INPUT)
```

8 237

```
JXT       = TOTAL NUMBER OF VALUES IN SECOND TABLE     (INPUT)
XP        = GIVEN VALUE OF FIRST VARIABLE              (INPUT)
YP        = GIVEN VALUE OF SECOND VARIABLE             (INPUT)
ZP        = INTERPOLATED FUNCTIONAL VALUE              (OUTPUT)
I         = INDEX FOR INITIAL GUESS AND FINAL POSITION OF VAR. 1
            (INPUT/OUTPUT)
J         = INDEX FOR INITIAL GUESS AND FINAL POSITION OF VAR. 2
            (INPUT/OUTPUT)
```

****

SUBROUITNE POSITN (IUNIT,IPLOT,IEOV)

SUBROUTINE POSITN POSITIONS THE DATA TAPE ON UNIT NUMBER IUNIT, AFTER THE DATA RECORD HEADER PREFIX FOR DATA RECORD IPLOT.

POSITN IS CALLED BY SUBROUTINES OVRLYP, INEDTE AND REDTAP.

```
IUNIT  = UNIT NUMBER OF DATA TAPE                  (INPUT)
IPLOT  = DATA RECORD NUMBER                        (INPUT)
IEOV   = RETURN CONDITION FLAG.                    (OUTPUT)
       = -1 IF TRAILER LABEL ENCOUNTERED
       =  0 FOR NORMAL RETURN
       =  1 FOR END OF VOLUME ENCOUNTERED
```

****

SUBROUTINE POSPLT (IUNIT,ITYP,IPLOT,IEOV)

SUBROUTINE POSPLT LOCATES DATA RECORD ON RELAP3, RELAP4/02, RELAP4/03, OR RETRAN STRANGER TAPES.

POSPLT IS CALLED BY SUBROUTINE REDTAP.

```
IUNIT  = UNIT NUMBER OF DATA TAPE      (INPUT)
ITYP   = TYPE OF DATA TAPE FLAG        (INPUT)
IPLOT  = DATA RECORD NUMBER            (INPUT)
IEOV   = RETURN CONDITION FLAG         (OUTPUT)
       = -1 TRAILER LABEL ENCOUNTERED
       =  0 FOR NORMAL RETURN
       =  1 FOR END OF VOLUME ENCOUNTERED
```

****

SUBROUTINE POSTW (IC)

SUBROUTINE POSTW CALCULATES AVERAGE VOLUME MASS FLOW RATE.

POSTW IS CALLED BY SUBROUTINE FLOSRH.

```
DT        = TIME STEP SIZE                 (INPUT)
```

```
IC        = 1 FOR FIRST CALL (BEFORE FIRST CALL TO PREW)     (INPUT)
IC        = 2 FOR STEADY STATE ITERATION LOOP
IC        = 0 FOR TRANSIENT LOOP
```

****

FUNCTION POWRT (NODEL,DT)

FUNCTION POWRT DIRECTS THE TYPE OF POWER CALCULATES, E.G. EXPLICIT POWER VS. TIME, FROM DATA TAPE OR PAINT KINETICS.

POWRT IS CALLED BY SUBROUTINES TRAN, INITLZ, AND RESOPT.

```
NODEL     = POWER TYPE INDICATOR                    (INPUT)
DT        = TIME STEP SIZE                          (INPUT)
POWRT  = NORMALIZED POWER                           (OUTPUT)
```

****

SUBROUTINE PRESS (THERMO,STDATA,PHASE,ISW,ZM,ZVOL,IREAD)

SUBROUTINE PRESS DETERMINES THE THERMODYNAMIC STATE IN A VOLUME AND COMPUTES PARTIAL DERIVATIVES OF PRESSURE WITH RESPECT TO WATER MASS, AIRMASS, AND TOTAL ENERGY. PRESS DETERMINES THE STATE PROPERTIES FOR TIME DEPENDENT VOLUMES ONLY.

PRESS IS CALLED BY SUBROUTINES INVOL, BAL AND RESOPT.

```
THERMO = THERMODYNAMIC DATA TABLE                   (INPUT)
STDATA = ARRAY OF INPUT AND OUTPUT STATE DATA
(INPUT/OUTPUT)
PHASE  = PHASE OF WATER IN A VOLUME,                (OUTPUT)
         1 = SUBCOOLED
         2 = TWO PHASE
         3 = SUPERHEATED
         4 = CRITICAL
ISW    = ERROR FLAG , 0 = NO ERROR , 1 = ERROR      (OUTPUT)
ZM     = VOLUME MIXTURE LEVEL                       (INPUT)
ZVOL   = VOLUME HEIGHT                              (INPUT)
IREAD  = TIME DEPENDENT VOLUME NUMBER               (INPUT)
```

****

SUBROUTINE PREW (DT,IC)

SUBROUTINE PREW CALCULATES FRICTION AND MOMENTUM FLUX TERMS FOR ALL JUNCTIONS. PREW ALSO OBTAINS JUNCTION STATE PROPERTIES VIA A PRESSURE-ENTHALPY STATE CALL.

PREW IS CALLED BY SUBROUTINES TRAN, STSTAT AND INITLZ.

```
DT = RETRAN TIME STEP                    (INPUT)
IC = 0 FOR INITIAL CONDITIONS            (INPUT)
   = 1 FOR TRANSIENT FORM
```

****

SUBROUTINE PRNPLT

SUBROUTINE PRNPLT IS THE DRIVER FOR OBTAINING PRINTER PLOTS.

PRNPLT IS CALLED BY SUBROUTINE RETRAN.

****

SUBROUTINE PRSORK (IDXV,ISZV,IDXJ,ISZJ,ITERST,IUV)

SUBROUTINE PRSORK CALCULATES VOLUME PRESSURES OR JUNCTION LOSS COEFFICIENTS FOR STEADY-STATE INITIALIZATION.

PRSORK IS CALLED BY SUBROUTINE STSTAT.

```
IDXV   = INDEX OF A SCRATCH ARRAY CONTIANING VOLUME DATA FOR
         STEADY-STATE INITIALIZATION                    (INPUT)
ISZV   = SET SIZE OF THE VOLUME SCRATCH ARRAY           (INPUT)
IDXJ   = INDEX OF A SCRATCH ARRAY CONTAINING JUNCTION DATA FOR
         STEADY-STATE INITIALIZATION                    (INPUT)
ISZJ   = SET SIZE OF THE JUNCTION DATA SCRATCH ARRAY    (INPUT)
ITERST = NUMBER OF ITERATIONS LEFT FOR STEADY-STATE
         CALCULATIONS                                   (INPUT)
IUV    = INITIAL CALL FLAG, = 1 FIRST CALL TO PRSORK    (INPUT)
```

****

SUBROUTINE PRZR (I,DT,STDATA)

SUBROUTINE PRZR CALCULATES THE THERMODYNAMIC STATES OF TWO REGIONS IN A VOLUME ALLOWING NON-EQUILIBRIUM CONDITIONS BETWEEN REGIONS BUT CONSTRAINED TO EQUAL PRESSURE IN THE VOLUME.

PRZR IS CALLED BY SUBROUTINES BAL AND RESOPT.

```
I      = VOLUME FILE INDEX                       (INPUT)
DT     = RETRAN TIME STEP                        (INPUT)
STDATA = ARRAY OF INPUT AND OUTPUT STATE DATA    (INPUT/OUTPUT)
```

****

SUBROUTINE PULLIN (IUNIT,IEOV)

SUBROUTINE PULLIN IS USED BY MODULES PLOTER AND REEDIT, TO MOVE A DATA RECORDS INTO THE APPROPRIATE ABBREVIATED FILES RESERVED BY SUBROUTINE SETUPE.

PULLIN IS CALLED BY SUBROUTINES REQEST AND REDTAP.

```
IUNIT    = TAPE UNIT DATA TAPE IS TO BE READ FROM     (INPUT)
IEOV     = FLAG USED TO DETERMINE RETURN CONDITION, 0=NORMAL
           RETURN, 1=END OF VOLUME                    (OUTPUT)
```

****

SUBROUTINE PULTRN (IUNIT,ITYP,IPLOT,IEOV)

SUBROUTINE PULTRN IS USED BY MODULE PLOTER TO MOVE DATA RECORDS FROM RELAP4/02, RELAP4/03, AND RELAP3 DATA TAPES TO THE APPROPRIATE ABREVIATED FILES RESERVED BY SETUPO.

PULTRN IS CALLED BY SUBROUTINE REDTAP.

```
IUNIT  = TAPE UNIT DATA TAPE IS TO BE READ FROM        (INPUT)
ITYP   = TAPE TYPE FLAG                                (INPUT)
IPLOT  = DATA RECORD NUMBER                            (INPUT)
IEOV   = RETURN CONDITION FLAG, 1 = END OF VOLUME, 0 = NORMAL
         (OUTPUT)
```

****

SUBROUTINE PUMP (K,N,W,H,T)

SUBROUTINE PUMP RETURNING A NORMALIZED VALUE OF LEAD AND TORQUE (FROM THE HOMOLOGOUS PUMP CORVES) GIVEN FLOW A SPEED.

PUMP IS CALLED BY SUBROUTINE PUMPS.

```
K        = CURVE SET NUMBER                  (INPUT)
N        = INDEX OF PUMP DESCRIPTION SET     (INPUT)
W        = VOLUME FLOW                       (INPUT)
H        = PUMP HEAD, FT                     (OUTPUT)
T        = PUMP TORQUE                       (OUTPUT)
```

****

SUBROUTINE PUMPS (DT,START,INTEG)

SUBROUTINE PUMPS CALCULATES PUMP PRESSURES, TORQUES, SPEED, AND ENERGY ADDITION TO THE FLUID.

PUMPS IS CALLED BY SUBROUTINE PREW.

8 241

```
DT      = RETRAN TIME STEP                                    (INPUT)
START   = LOGICAL FOR INITIAL CONDITIONS (TRUE)               (INPUT)
INTEG   = PUMP SPEED INTEGRATION FLAG                         (INPUT)

**********************************************************************

   SUBROUTINE QDNBBA (G,X,HFG,QDNB)

SUBROUTINE QDNBBA COMPUTES CHF BY BARNETT CORRELATION.

QDNBBA IS CALLED BY SUBROUTINE DNBM.

G         = MASS FLUX          (INPUT)
X         = QUALITY            (INPUT)
HFG       = LATENT HEAT        (INPUT)
QDNB      = DNB HEAT FLUX      (OUTPUT)

**********************************************************************

   SUBROUTINE QDNBBO (P,X,G,HFG,QDNB)

SUBROUTINE QDNBBO COMPUTES CHF BY BOWRING CORRELATION.

QDNBBO IS CALLED BY SUBROUTINE DNBM.

P         = PRESSURE           (INPUT)
X         = QUALITY            (INPUT)
G         = MASS FLUX          (INPUT)
HFG       = LATENT HEAT        (INPUT)
QDNB      = DNB HEAT FLUX      (OUTPUT)

**********************************************************************

   SUBROUTINE QDNBBW (PR,HFG,GCORE,QDNB,QUAL)

SUBROUTINE QDNBBW COMPUTES CHF BY B+W-2 CORRELATION.

QDNBBW IS CALLED BY SUBROUTINE DNBM.

PR        = PRESSURE           (INPUT)
HFG       = LATENT HEAT        (INPUT)
GCORE     = MASS FLUX          (INPUT)
QDNB      = DNB HEAT FLUX      (OUTPUT)
QUAL      = QUALITY            (INPUT)

**********************************************************************

   SUBROUTINE QDNBJL (G,X,P,QDNB)
```

8 242

SUBROUTINE QDNBJL COMPUTES DNB HEAT FLUX BY JANSSEN-LEVY CORRELATION.

QDNBJL IS CALLED BY SUBROUTINE DNBM.

```
G     =  MASS FLUX                                          (INPUT)
X     =  QUALITY                                            (INPUT)
P     =  PRESSURE                                           (INPUT)
QDNB  = DNB HEAT FLUX                                       (OUTPUT)
```

****

```
   SUBROUTINE QDNBMA (GN,X,HFG,QDNB)
```

SUBROUTINE QDNBMA COMPUTES CHF BY MACBETH CORRELATION.

QDNBMA IS CALLED BY SUBROUTINE DNBM.

```
GN        = MASS FLUX          (INPUT)
X         = QUALITY            (INPUT)
HFG       = LATENT HEAT        (INPUT)
QDNB      = DNB HEAT FLUX      (OUTPUT)
```

****

```
   SUBROUTINE QDNBW3 (PR,H,QUAL,QDNB,HFSAT,GCORE)
```

SUBROUTINE QDNBW3 COMPUTES CHF BY W-3 CORRELATION.

QDNBW3 IS CALLED BY SUBROUTINE DNBM.

```
PR        = PRESSURE           (INPUT)
H         = ENTHALPY           (INPUT)
QUAL      = QUALITY            (INPUT)
QDNB      = DNB HEAT FLUX      (OUTPUT)
HFSAT     = ENTHALPY OF SATURATED LIQUID     (INPUT)
GCORE     = MASS FLUX          (INPUT)
```

****

```
   SUBROUTINE QDOT (A,B,CPF,CPG,CP,RF,RG,R,G,X,HD,HF,HG,PR,GCRIT,
                    QQ,TBULK,TS,TSAT,TSUR,IH,J,L,ITS)
```

SUBROUTINE QDOT CALCULATES SURFACE TEMPERTAURE FLUX AND HEAT TRANSFER COEFFICIENT, GIVEN A SPECIFIC HEAT TRANSFER CORRELATION AND CONDUCTION VALUES.

QDOT IS CALLED BY SUBROUTINE HTRC.

```
A, B   = COEFFICIENTS FROM THE CONDUCTION SOLUTION IN THE FORM
         QQ = A*TSUR + B                                    (INPUT)
CPF    = SPECIFIC HEAT CAPACITY FOR SATURATED LIQUID        (INPUT)
```

```
CPG     = SPECIFIC HEAT CAPACITY FOR SATURATED GAS                (INPUT)
CP      = SPECIFIC HEAT CAPACITY FOR SINGLE PHASE                 (INPUT)
RF      = DENSITY FOR SATURATED LIQUID, FT**3/LBM                 (INPUT)
RG      = DENSITY FOR SATURATED GAS                               (INPUT)
R       = SINGLE PHASE DENSITY                                    (INPUT)
G       = COOLANT FLOW RATE                                       (INPUT)
X       = VOLUME AVERAGE QUALITY                                  (INPUT)
HD      = HYDRAULIC DIAMETER, FT                                  (INPUT)
HF      = SATURATION LIQUID ENTHALPY                              (INPUT)
HG      = SATURATION GAS ENTHALPY                                 (INPUT)
PR      = VOLUME PRESSURE                                         (INPUT)
QCRIT   = CRITICAL HEAT FLUX                                      (INPUT)
QQ      = SURFACE HEAT FLUX                                       (OUTPUT)
TBULK   = WATER TEMPERATURE                                       (INPUT)
TSAT    = SATURATION TEMPERATURE                                  (INPUT)
TSUR    = SURFACE TEMPERATURE                                     (INPUT)
IH      = HEAT TRANSFER MODE --                                   (INPUT)
          1 = FORCED CONVECTION TO LIQUID
          2 = NUCLEATE BOILING
          3 = FORCED CONVECTION VAPORIZATION
          4 = FLOW TRANSITION BOILING
          5 = FLOW FILM BOILING
          6 = POOL FILM BOILING
          7 = POOL TRANSITION BOILING
          8 = FORCED CONVECTION TO GAS
          9 = DOUGALL-ROHSENOW FOR LOW PRESSURE FLOW FILM BOILING
         10 LOW FLOW CONVECTION TO LIQUID              COLLIER
         11 LOW FLOW NUCLEATE BOILING                  ROHSENOW
         12 FLOW FLOW TRANSITION BOILING               BROMLEY
         13 LOW FLOW CONVECTION TO STEM      SIEDER-TATE (TURBULENT)
                                             COLLIER  (LAMINAR)
J       = INDICATOR FOR FILM BOILING CORRELATION                  (INPUT)
          0 = USE GROENEVLED 5.9 FOR MODE 5
          1 = USE GROENEVELD 5.7 FOR MODE 5
L       = TYPE OF POOL BOILING, USED ONLY FOR MODES, IH=6 AND 7
          1 = TRANSITION POOL BOILING                           (OUTPUT)
          2 = LAMINAR POOL BOILING
ITS     = INDICATOR FOR SURFACE TEMPERATURE,
          0 = USE OLD VALUE, 1 = CALCULATE NEW VALUE            (INPUT)

**********************************************************************

   FUNCTION REAC

FUNCTION REAC COMPUTES THE TOTAL REACTIVITY FOR THE POINT
KINETICS MODEL.

REAC IS CALLED BY SUBROUTINE RKEN.

REAC  = TOTAL FEEDBACK REACTIVITY + SCRAM REACTIVITY          (OUTPUT)

**********************************************************************

   SUBROUTINE REDTAP
```

SUBROTUINE REDTAP IS THE DRIVER ROUTINE FOR READING DATA TAPES AND STORING THE REQUESTED PLOT VARIABLES FOR MODULE PLOTER.

REDTAP IS CALLED BY SUBROUTINE PLOTER.

****

SUBROUTINE REEDIT

SUBROUTINE REEDIT IS THE MAIN DRIVER FOR THE REEDIT MODULE.

REEDIT IS CALLED BY SUBROUTINE RMAIN.

****

SUBROUTINE RESOPT

SUBROUTINE RESOPT IS USED TO RESERVE SPACE FOR ACTIVE OPTIONS AND FREE SPACE OCCUPIED BY INACTIVE OPTIONS FOR STORAGE.

RESOPT IS CALLED BY SUBROUTINE RESSEG.

****

SUBROUTINE RESSEG (LOCMOD)

SUBROUTINE RESSEG SETS UP INFORMATION USED TO RESERVE SPACE FOR ACTIVE OPTIONS CODING AND FREE SPACE FOR INACTIVE OPTIONS CODING.

RESSEG IS CALLED BY SUBROUTINES PLOTER, REEDIT AND INRTRN

LOCMOD = BEGINNING OF THE SECOND LEVEL OF THE LOAD (INPUT)

****

SUBROUTINE REQEST (IUNIT,X,IX,IXI,ITYP)

SUBROUTINE REQEST GENERATES AN EDIT DATA TABLE TO BE USED BY EDINIT FOR RELAP3, RELAP4/02, RELAP4/03, AND RETRAN STRANGER DATA TAPE PLOT REQUESTS.

REQEST IS CALLED BY SUBROUTINE INEDIT.

IUNIT = TAPE UNIT CONTAINING DATA
X = SCRATCH ARRAY USED TO DESCRIBE VARIABLES AVAILABLE FOR EDITING. (OUTPUT)

8 245

```
IX      = COUNTER FOR THE NUMBER OF VARIABLES DESCRIBED     (OUTPUT)
IXI     = POINTER IN THE X ARRAY TO THE STARTING POSITION OF
          THE INDIVIDUAL VARIABLE DESCRIPTION               (OUTPUT)
ITYP    = DATA TAPE TYPE INDICATOR, 2=RELAP4/003, 3=RELAP4/002,
          4=RELAP3, 5=RETRAN STRANGER TAPE                  (INPUT)
```

****

SUBROUTINE RETRAN

SUBROUTINE RETRAN IS THE DRIVER FOR MODULE RETRAN.

RETRAN IS CALLED BY SUBROUTINE RMAIN.

****

SUBROUTINE RKEN (DTM)

SUBROUTINE RKEN PERFORMS THE REACTOR POINT KINETICS CALCULATIONS.

RKEN IS CALLED BY SUBROUTINE POWRT.

```
DTM = RETRAN TIME STEP              (INPUT)
```

****

SUBROUTINE ROUND (TMX,TMN,DELTA,VMAX,ORG,LAXE)

SUBROUTINE ROUND SCALES PLOTS TO GET THE APPROPRIATE POWER OF TEN.

ROUND IS CALLED BY SUBROUTINE PLOTPR.

```
TMX     = MAX VALUE OF DATA FOR SCALING     (INPUT)
TMN     = MIN VALUE OF DATA FOR SCALING     (INPUT)
DELTA   = LENGTH OF MAJOR AXIS DIVISION     (OUTPUT)
VMAX    = SCALE VALUE AT END                (OUTPUT)
ORG     = SCALE VALUE AT ORIGIN             (OUTPUT)
LAXE    = AXIS LENGTH                       (INPUT)
```

****

FUNCTION SCRM (NODEL)

FUNCTION SCRM RETURNS REACTIVITY FROM THE REACTIVITY VERSUS TIME SCRAM TABLE.

SCRM IS CALLED BY SUBROUTINE REAC.

```
NODEL = POWER TYPE CALCULATION INDICATOR            (INPUT)
```

1758 246

```
**********************************************************************

   SUBROUTINE SEARCH(IDXFLG,IDXVAR,IEDT,MODE,IDXF,IFIL,DD,J,K)

SUBROUTINE SEARCH VERIFIES REQUEST FLAGS(MINOR EDIT, PLOT, CONT.
BLK) AND REG NUMBERS AND RETURNS ADDRESS OF REQUESTED PARAMETER.

SEARCH IS CALLED BY SUBROUTINES EDINIT AND INCNT2.

IDXFLG = INDEX RELATIVE TO RSTOR(1) FOR REQUESTED PARAM    (OUTPUT)
IDXVAR = INDEX RELATIVE TO RSTOR(1) FOR REQUEST PAIR       (INPUT)
IEDT   = MINOR EDIT = 1, PLOT = 2, CONT BLK = 3            (INPUT)
MODE   = RETRAN-RESTRT = 1, REEDIT = 2, PLOTER = 3         (INPUT)
IDXF   = ARRAY CONTAINING FILE INDICES FOR AVAIL FILES     (INPUT)
IFIL   = ARRAY CONTAINING REGION CHECK FLAGS (CHECK TYPE)  (INPUT)
DD     = 5 X N ARRAY DESCRIBING PARAM AVAIL FOR REQUEST    (INPUT)
J      = POSITION IN IFIL OF VARIFIED REQUEST PAIR         (OUTPUT)
K      = POSITION IN DD OF VARIFIED REQUEST PAIR           (OUTPUT)

**********************************************************************

   SUBROUTINE SENG (L,MM,NJL,NJR,E)

SUBROUTINE SENG COMPUTES STORED ENERGY IN A HEAT CONDUCTOR

SENG IS CALLED BY SUBROUTINES SINITL AND COND.

L      = HEAT CONDUCTOR FILE INDEX                         (INPUT)
MM     = HEAT CONDUCTOR GEOMETRY FILE INDEX                (INPUT)
NJL    = NODE NUMBER ON LEFT SURFACE                       (INPUT)
NJR    = NODE NUMBER OF RIGHT SURFACE                      (INPUT)
E      = STORED ENERGY                                     (OUTPUT)

**********************************************************************

   SUBROUTINE SETUPE (IUNIT)

SUBROUTINE SETUPE CREATES ABBREVIATED FILES USED TO PROCESS
RETRAN DATA RECORDS FOR MODULES REEDIT AND PLOTER.

SETUPE IS CALLED BY SUBROUTINES REDTAP AND INEDTE.

IUNIT  = DATA TAPE UNIT NUMBER                 (INPUT)

**********************************************************************

   SUBROUTINE SETUPO (IUNIT,ITYP)
```

8 247

SUBROUTINE SETUPO CREATES ABBREVIATED FILES USED TO PROCESS RELAP3, RELAP4/02, RELAP4/03, AND RETRAN STRANGER TAPES FOR PLOTTING.

SETUPO IS CALLED BY SUBROUTINE REDTAP.

IUNIT = DATA TAPE UNIT NUMBER (INPUT)
ITYP = DATA TAPE TYPE INDICATOR, 2=RELAP4/03, 3=RELAP4/02, 4=REALP3, 5=RETRAN STRANGER TAPE (INPUT)

****

SUBROUTINE SFR

SUBROUTINE SFR DETERMINES THE FLOW REGIME IN EACH VOLUME.

SFR IS CALLED BY SUBROUTINE PREW.

****

SUBROUTINE SIMQ (A,B,N,KS)

SUBROUTINE SIMQ SOLVES N LINEAR SIMULTANEOUS EQUATIONS.

SIMQ IS CALLED BY SUBROUTINE FITHT.

A = MATRIX A FOR AX = B (INPUT/OUTPUT)
B = MATRIX B FOR AX = B (INPUT/OUTPUT)
N = RANK OF MATRIX A (INPUT)
KS = SINGULARITY INDICATOR OF SOLUTION OF AX = B (OUTPUT)

****

SUBROUTINE SINITL

SUBROUTINE SINITL IS THE DRIVER FOR THE TIME ZERO HEAT CONDUCTION SOLUTION.

SINITL IS CALLED BY SUBROUTINE ENERGY.

****

SUBROUTINE SLABDT (DT)

SUBROUTINE SLABDT DETERMINES WHICH HEAT CONDUCTORS ARE CAUSAL I.E. THE CAUSAL CONDUCTOR OPTION.

SLABDT IS CALLED BY SUBROUTINE ENERGY.

DT = RETRAN TIME STEP (INPUT)

****

SUBROUTINE SLABHT

SUBROUTINE SLABHT IS THE DRIVER FOR THE TRANSIENT HEAT CONDUCTION SOLUTION

SLABHT IS CALLED BY SUBROUTINE ENERGY.

****

SUBROUTINE SURTEN (T,SIGMA)

SUBROUTINE SURTEN COMPUTES SURFACE TENSION GIVEN STEAM TEMPERATURE.

SURTEN IS CALLED BY SUBROUTINE QDOT.

T	= STEAM TEMPERATURE (F)	(INPUT)
SIGMA	= SURFACE TENSION (LBF/FT)	(OUTPUT)

****

SUBROUTINE STATE (THERMO,STDATA,PHASE,ISW)

SUBROUTINE STATE DETERMINES THE THERMODYNAMIC STATE IN EACH VOLUME AND COMPUTES THE PARTIAL DERIVATIVE OF PRESSURE WITH RESPECT TO WATER MASS, AIR MASS AND TOTAL INTERNAL ENERGY.

STATE IS CALLED BY SUBROUTINES BAL AND PRZR.

THERMO	= THERMODYNAMIC DATA TABLE	(INPUT)
STDATA	= ARRAY OF INPUT AND OUTPUT STATE DATA	(INPUT/OUTPUT)
PHASE	= PHASE INDICATOR, 1=LIQUID, 2=TWO PHASE, 3=SUPERHEAT, 4=CRITICAL	(OUTPUT)
ISW	= ERROR SWITCH 0 =# NO ERROR, 1 =# ERROR	(OUTPUT)

****

SUBROUTINE STATPH

SUBROUTINE STATPH COMPUTES FLUID STATE PROPERTIES GIVEN PRESSURE AND ENTHALPY.

STATPH IS CALLED BY SUBROUTINES STSTAT AND INITLZ.

8 249

****

SUBROUTINE STH20I

SUBROUTINE STH20I READS THE WATER PROPERTY TABLE INPUT REQUEST CARD (010050), RESERVES A FILE TO CONTAIN THE DATA AND MOVES THE DATA INTO THE FILE.

STH20I IS CALLED BY SUBROUTINE INTRAN.

****

SUBROUTINE STPM

SUBROUTINE STPM COMPUTES A FANNING FRICTION FACTOR AND A TWO-PHASE FRICTION MULTIPLIER.

STPM IS CALLED BY SUBROUTINE PREW.

****

SUBROUTINE STSTAT

SUBROUTINE STSTAT CONTROLS THE PROGRAM FLOW FOR STEADY STATE INITIALIZATION.

STSTAT IS CALLED BY SUBROUTINE RETRAN.

****

SUBROUTINE TAPEBC

SUBROUTINE TAPEBC RETRIEVES VOLUME THERMODYNAMIC BOUNDARY CONDITIONS AND/OR POWER HISTORIES FROM RETRAN DATA TAPES.

TAPEBC IS CALLED BY SUBROUTINES INVOL, BAL AND RESOPT.

****

SUBROUTINE TAVE (L,M,NL,NRR,TA)

SUBROUTINE TAVE COMPUTES THE AVERAGE METAL TEMPERATURE OF A CORE HEAT CONDUCTOR.

TAVE IS CALLED BY SUBROUTINES SINITL, SENG, COND AND TKANDC.

L = HEAT CONDUCTOR FILE INDEX (INPUT)
M = HEAT CONDUCTOR GEOMETRY FILE INDEX (INPUT)

8 250

```
NL = MODE NUMBER AT LEFT SURFACE.                                (INPUT)
NRR = NODE NUMBERS AT RIGHT SURFACE                              (INPUT)
TA  = AVERAGE METAL TEMPERATURE                                  (OUTPUT)
```

****

```
   SUBROUTINE TEMP (DT,IC,M,QMWR,LL,ASS,V,TK,CR,Q,A,T,B)
```

SUBROUTINE TEMP SOLVES THE ONE-DIMENSIONAL TRANSIENT HEAT CONDUCTION EQUATION.

TEMP IS CALLED BY SUBROUTINE COND.

```
DT      = RETRAN TIME STEP SIZE                                  (INPUT)
IC      = CONDUCTOR INDEX                                        (INPUT)
M       = NUMBER OF NODES                                        (INPUT)
QMWR    = ADDITIONAL HEAT GENERATED PER FOOT OF ROD BY METAL
          WATER REACTION, BTU/SEC                                (INPUT)
LL      = HEAT CONDUCTOR FILE INDEX                              (INPUT)
ASS     = SURFACE AREA / 2 * DX                                  (INPUT)
V       = VOLUME, FT**3                                          (INPUT)
TK      = THERMAL CONDUCTIVITY, BTU/FT-SEC-F                     (INPUT)
CR      = VOLUMETRIC HEAT CAPACITY, BTU/FT**3-F                  (INPUT)
Q       = HEAT GENERATION RATE, BTU/FT**3-SEC                    (INPUT)
A       = S*TK, OLD TIME STEP VALUES                             (INPUT)
T       = TEMPERATURE, OLD TIME STEP VALUES, F
B       = WORKING SPACE FOR MATRIX SOLUTION                      (OUTPUT)
```

****

```
   SUBROUTINE TEMZ (M,IL,IR,TLX,TRX,SL,SR,HL,HR,S,V,Q,TK,A,T,B,
                    LL,FLP,FRP)
```

SUBROUTINE TEMZ SOLVES THE ONE-DIMENSIONAL STEADY-STATE HEAT CONDUCTION EQUATION.

TEMZ IS CALLED BY SUBROUTINE SINITL.

```
M       = NUMBER OF NODES                                        (INPUT)
IL      = INDICATOR FOR LEFT BOUNDARY CONDITION                  (INPUT)
IR      = INDICATOR FOR RIGHT BOUNDARY CONDITION                 (INPUT)
TLX     = LEFT SURFACE TEMPERATURE                               (INPUT)
TRX     = RIGHT SURFACE TEMPERATURE                              (INPUT)
SL      = SURFACE AREA/2, LEFT                                   (INPUT)
SR      = SURFACE AREA/2, RIGHT                                  (INPUT)
HL      = HEAT TRANSFER COEFICIENT, LEFT                         (INPUT)
HR      = HEAT TRANSFER COEFICIENT, RIGHT                        (INPUT)
S       = SURFACE AREA / 2 * DX                                  (INPUT)
V       = VOLUME                                                 (INPUT)
Q       = HEAT GENERATION RATE                                   (INPUT)
TK      = THERMAL CONDUCTIVITY                                   (INPUT)
A       = S*TK                                                   (OUTPUT)
```

8 251

```
T         = TEMPERATURE                                          (OUTPUT)
B         = SCRATCH SPACE FOR MATRIX SOLUTION                    (OUTPUT)
LL        = HEAT CONDUCTOR NUMBER                                (INPUT)
FLP       = FLUX AT LEFT SURFACE                                 (INPUT)
FRP       = FLUX AT RIGHT SURFACE                                (INPUT)
```

****

SUBROUTINE TFFM (G,PP,XX,TPF,IPR,IX1,IX2)

SUBROUTINE TFFM COMPUTES A TWO PHASE FRICTION MULTIPLIER.

TFFM IS CALLED BY SUBROUTINE STPM.

```
G         = MASS FLUX, LB/HR-FT**2                    (INPUT)
PP        = PRESSURE, LB/IN**2                        (INPUT)
XX        = QUALITY                                   (INPUT)
TPF       = TWO-PHASE FRICTION FACTOR MULTIPLIER      (OUTPUT)
IPR       = PRESSURE TABLE INDEX                      (INPUT/OUTPUT)
IX1       = FIRST QUALITY TABLE INDEX                 (INPUT/OUTPUT)
IX2       = SECOND QUALITY TABLE INDEX                (INPUT/OUTPUT)
```

****

SUBROUTINE THCON (N,T1,RHO1)

FUNCTION THCON COMPUTES THE THERMAL CONDUCTIVITY OF SATURATED LIQUID AND SATURATED AND SUPERHEATED VAPOR.

THCON IS CALLED BY SUBROUTINE QDOT.

```
N         = 1, SATURATED LIQUID    2,SATURATED AND SUPERHEATED
            VAPOR
            (INPUT)
T1        = TEMPERATURE                (INPUT)
RHO1      = DENSITY                    (INPUT)
THCON     = THERMAL CONDUCITIVITY (BTU/FT-HR-F)      (OUTPUT)
```

****

SUBROUTINE TIMINT (NUMBER)

SUBROUTINE TIMINT IS AN INTERNAL TIMING ROUTINE USED TO TIME VARIOUS FACTIONS OF RETRAN COMPUTATIONS.

TIMINT IS CALLED BY SUBROUTINES PLOTER, REEDIT, INRTRN, INDATA.

```
NUMBER    = TABLE NUMBER OF ROUTINE TO BE TIMED      (INPUT)
```

78 252

****

SUBROUTINE TKANDC (NN,L,MM)

SUBROUTINES TKANDC OBTAINS THERMAL CONDUCTIVITY AND SPECIFIC HEAT CAPACITY FOR EACH CONDUCTOR NODE FROM THE TABULAR MATERIAL PROPERTY DATA.

TKANDC IS CALLED BY SUBROUTINES SINITL AND COND.

NN = NUMBER OF HEAT CONDUCTOR NODES (INPUT)
L = HEAT CONDUCTOR FILE INDEX (INPUT)
NN = HEAT CONDUCTOR GEOMETRY FILE INDEX (INPUT)

****

FUNCTION TRIPDT (TT)

SUBROUTINE TRIPDT PREDICTS TRIP ACTUATION TIMES TO FORCE THE EXPLICIT TIME STEP CONTROL OPTION TO CONSIDER TRIP TIMES AS PART OF THE TIME STEP SELECTION LOGIC.

TRIPDT IS CALLED BY SUBROUTINE TSED.

TT = PREVIOUS TIME STEP SIZE (SEC) (INPUT)

****

SUBROUTINE TRAN(IUNIT)

SUBROUTINE TRAN DIRECTS THE PROGRAM FLOW FOR THE RETRAN AND RESTRT TRANSIENT CALCULATIONS.

TRAN IS CALLED BY SUBROUTINE RETRAN.

IUNIT = TAPE UNIT NUMBER UPON WHICH DATA IS WRITTEN (INPUT)

****

FUNCTION TRIP (N,DELT)

FUNCTION TRIP SCANS ALL TRIPS AND ACTIVATES OR DEACTIVATES TRIPS ONCE THE SETPOINTS ARE REACHED.

TRIP IS CALLED BY SUBROUTINES TRAN, STSTAT, CHKV, FILL, MIXQ, CONDHT, PUMPS, POWRT, SCRM AND TAPEBC.

N = TRIP TYPE (INPUT)
DELT = ELAPSED TIME SINCE ACTIVATION OF TRIP (OUTPUT)
TRIP = .TRUE. IF TRIP IS ACTUATED (OUTPUT)

8 253

```
**********************************************************************

   SUBROUTINE TRNSPT (II,J,H,X,DT,IBH1)

SUBROUTINE TRNSPT COMPUTES  MESH ENTHALPIES FOR THE TEMPERATURE
TRANSPORT MODEL.

TRNSPT IS CALLED BY SUBROUTINES BAL, JUNHP AND RESOPT.

II = VOLUME FILE INDEX OF THE TRANSPORT VOLUME                 (INPUT)
J = JUNCTION FILE INDEX OF THE DOWNSTREAM JUNCTION.            (INPUT)
H = JUNCTION ENTHALPY                                  (INPUT/OUTPUT)
X = JUNCTION QUALITY                                          (OUTPUT)
DT = RETRAN TIME STEP                                          (INPUT)
IBH1 = 1 INTEGRATE MASS FLOWS AROUND THE TRANSPORT VOLUME
     = 2 ESTIMATE JUNCTION ENTHALPY FOR NEXT TIME STEP
     = 3 SHIFT MESH ENTHALPIES ACCORDING TO MASS FLOWS IN TO
       THE TRANSPORT VOLUME                                    (INPUT)

**********************************************************************

   SUBROUTINE TRPSUM

SUBROUTINE TRPSUM PROVIDES A TRIP ACTUATION HISTORY SUMMARY AS A
FUNCTION OF TIME.

TRPSUM IS CALLED BY SUBROUTINE RETRAN.

**********************************************************************

   FUNCTION TSED (MAJCON,MINCON,DMPCON,PLTCON,DTSTEP,IBRNCH)

FUNCTION TSED IS USED ONLY IF THE EXPLICIT TIME STEP CONTROL
OPTION IS USED. (NCHK=2)  TSED SETS A LIMIT FOR THE MAXIMUM
TIME STEP SIZE THAT CAN BE TAKEN RESTRAINED BY EDIT TIMES,
TRIP TIMES AND CHANGING OF TIME STEP DATA CARDS  (IBRNCH=1),
AND SET CONTROL EDIT FLAGS(IBRNCH=2).

TSED IS CALLED BY SUBROUTINES TRAN.

MAJCON = MAJOR EDIT FLAG                                      (OUTPUT)
MINCON = MINOR EDIT FLAG                                      (OUTPUT)
DMPCON = ERROR EDIT FLAG                                      (OUTPUT)
PLTCON = DATA RECORD DUMP FLAG                                (OUTPUT)
DTSTEP = PREDICTED TIME STEP SIZE                             (OUTPUT)
IBRNCH = 1 CALCULATE MAXIMUM TIME STEP SIZE, 2 SET EDIT
         FLAGS                                                (INPUT)
TSED = MAXIMUM TIME STEP SIZE ALLOWED                         (OUTPUT)

**********************************************************************

   SUBROUTINE TSTMOD (IDXCRD,LTST)
```

SUBROUTINE TSTMOD READS AND CHECKS THE TIME STEPS ALGORITHMS CONSTANTS DATA CARD (030001) AND/OR THE DETAILED EDIT CARD (030002).

TSTMOD IS CALLED BY SUBROUTINES INTSTP.

IDXCRD = INDEX OF THE ARRAY CONTAINING INPUT CARD DATA (INPUT)
LTST = .TRUE. IF TIME STEP CARDS ARE SUPPLIED ON A RESTART RUN (INPUT)

****

FUNCTION TSTP (MAJCON,MINCON,DMPCON,PLTCON)

FUNCTION TSTP IS THE STANDARD TIME STEP CONTROL ROUTINE. TSTP SELECTS THE RETRAN TIME STEPS AND SETS EDIT FLAGS.

TSTP IS CALLED BY SUBROUTINES TRAN.

MAJCON = MAJOR EDIT FLAG (OUTPUT)
MINCON = MINOR EDIT FLAG (OUTPUT)
DMPCON = ERROR EDIT FLAG (OUTPUT)
PLTCON = WRITE DATA TAPE FLAG (OUTPUT)
TSTP = RETRAN TIME STEP (OUTPUT)

****

SUBROUTINE VAPOR1 (LV,LE,LS)

SUBROUTINE VAPOR1 SETS UP SLOPE AND INTERCEPT FOR CALCULATION OF PARTIAL DENSITIES OF GAS BUBBLES AND MIXTURE USED IN SUBROUTINE BUBB.

VAPOR1 IS CALLED BY SUBROUTINES OVRLYP, DELHP, STATPH, BUBINT, AND BAL.

LV = BEGINNING FILE INDEX OF THE VOLUME FILE
LE = INDEX OF LAST SET IN THE VOLUME FILE
LS = SET SIZE OF THE VOLUME FILE

****

SUBROUTINE VELLIM (INPUT,OUTPUT,UPLIM,DWNLIM,GAIN,OFFSET)

SUBROUTINE VELLIM MODELS A SLEW RATE LIMITED AMPLIFIER.

VELLIM IS CALLED BY SUBROUTINE CONTRL.

INPUT = VALUE AT INPUT SIGNAL AT THE CURRENT TIME STEP (INPUT)
OUTPUT = VALUE OF THE OUTPUT SIGNAL AT THE CURRENT TIME STEP

8 255

```
          (OUTPUT)
UPLIM     = SLEW RATE IN POSITIVE DIRECTION                    (INPUT)
DWNLIM    = SLEW RATE IN NEGATIVE DIRECTION                    (INPUT)
GAIN      = GAIN OF THE AMPLIFIER                              (INPUT)
OFFSET    = OFFSET APPLIED AT THE OUTPUT OF THE AMPLIFIER      (INPUT)
```

****

FUNCTION VISC (N,T1,RHO1)

FUNCTION VISC COMPUTES VISCOSITY OF SATURATED LIQUID AND SATURATED AND SUPERHEATED VAPOR - 1967 STEAM TABLES.

VISC IS CALLED BY SUBROUTINES STPM, QDOT AND TURBCF.

```
N = 1, SATURATED LIQUID                          (INPUT)
    2, SATURATED AND SUPERHEATED VAPOR.
T1 = TEMPERATURE                                 (INPUT)
RHO1 = DENSITY                                   (INPUT)
VISC = VISCOSITY                                  (OUTPUT)
```

****

SUBROUTINE VDUDT (IDXV,ISZV)

SUBROUTINE VDUDT COMPUTES VOLUME DU/DT FOR STEADY STATE INITIALIZATION

```
IDXV = INDEX OF A SCRATCH ARRAY USED TO STORE VOLUME VARIABLES
       FOR STEADY STATE INITIALIZATION.                  (INPUT)
ISZV = SET SIZE OF THE VOLUME SCRATCH ARRAY.             (INPUT)
```

****

SUBROUTINE WPACK

SUBROUTINE WPACK COMPUTES READJUSTED FLOWS AT THE TIME OF WATER PACKING.

WPAC IS CALLED BY SUBROUTINE BAL.

1758 256

X. SUBROUTINE CALL CHARTS

Figures X.1-1 through X.1-6 present the call charts for the RETRAN code package. These call charts show the calls from each routine in the RETRAN code package with the exception of calls to INP package routines, the FTB package routines, error routines and other miscellaneous subroutines that would lend little insight to the overall flow of the code. These calls were omitted in order to keep the charts readable without sacrificing details of the logical organization. These charts show logical organization but not the structure of the loaded code which is covered in Sections II through V.

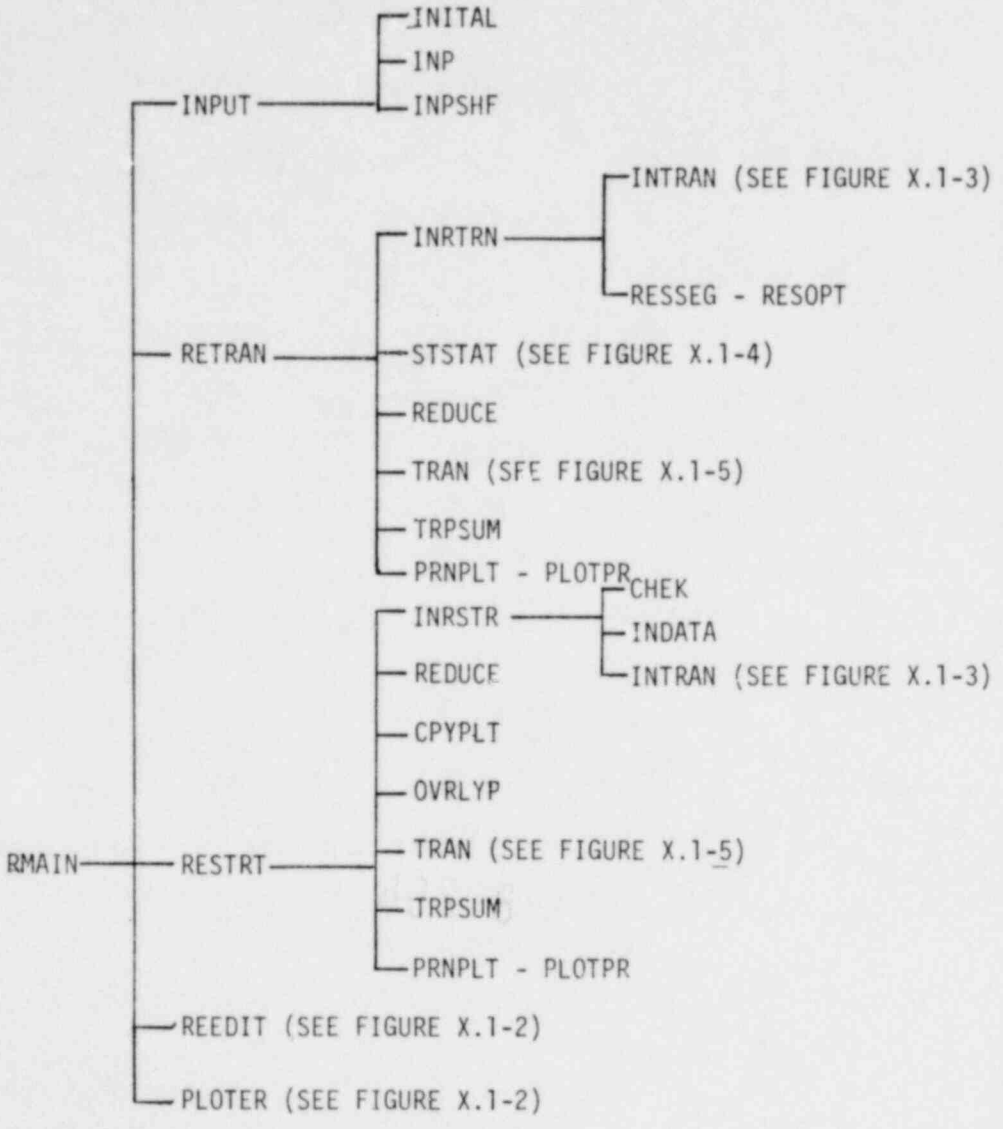

Figure X.1-1 Subroutine Calls from RMAIN

8 259

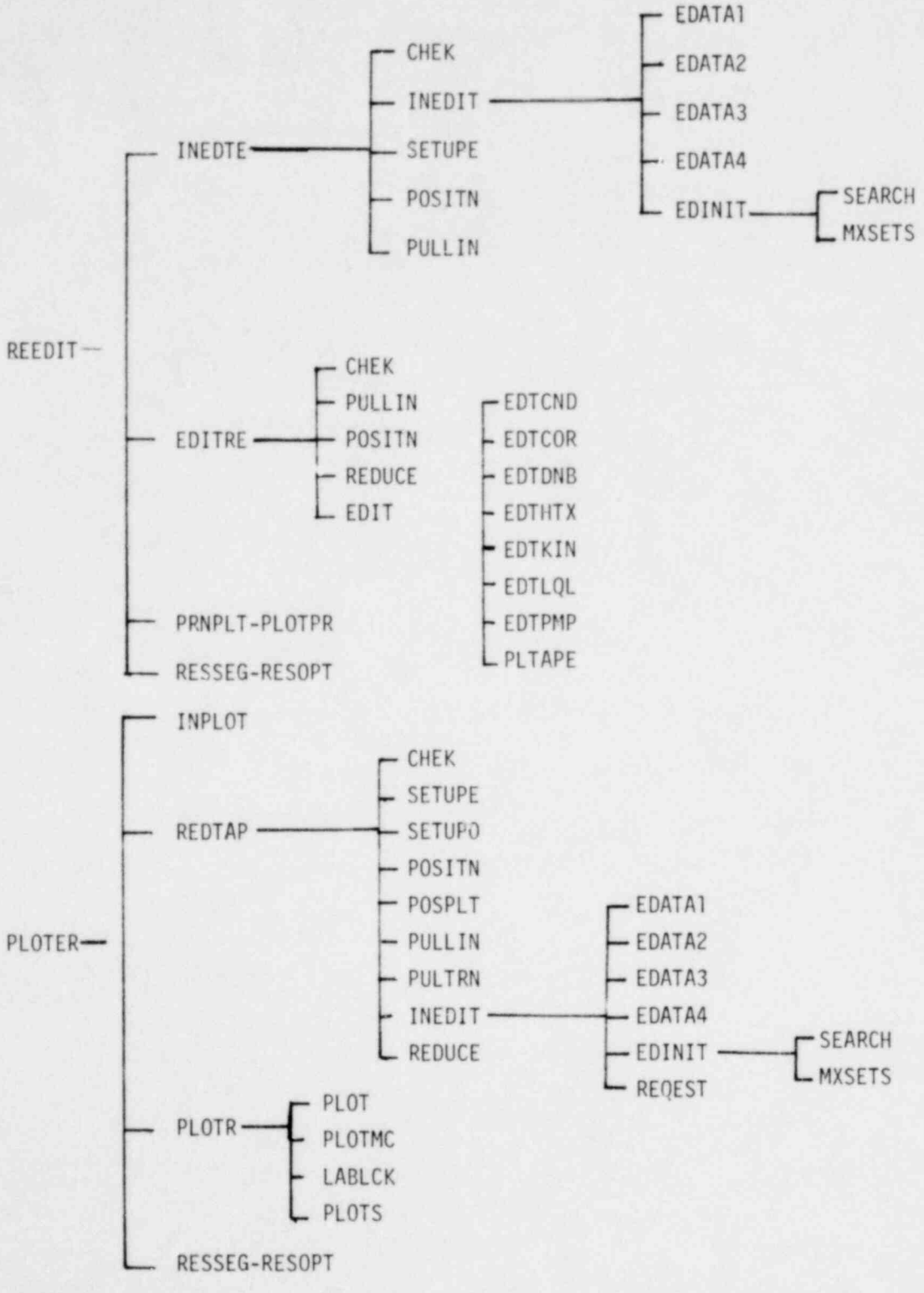

Figure X.1-2 Subroutine Calls from REEDIT and PLOTER

8 260

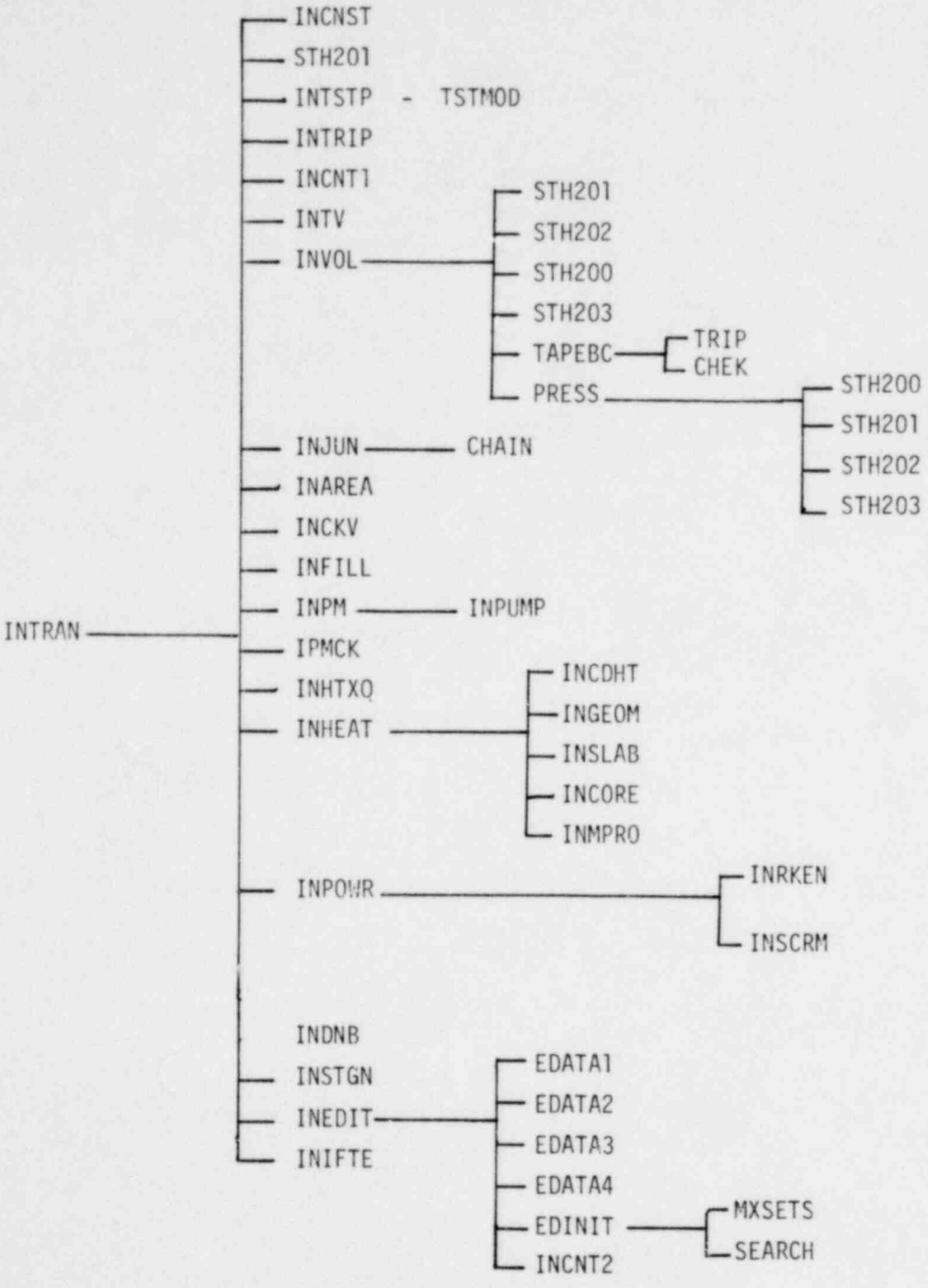

Figure X.1-3 Subroutine Calls From INTRAN

8 261

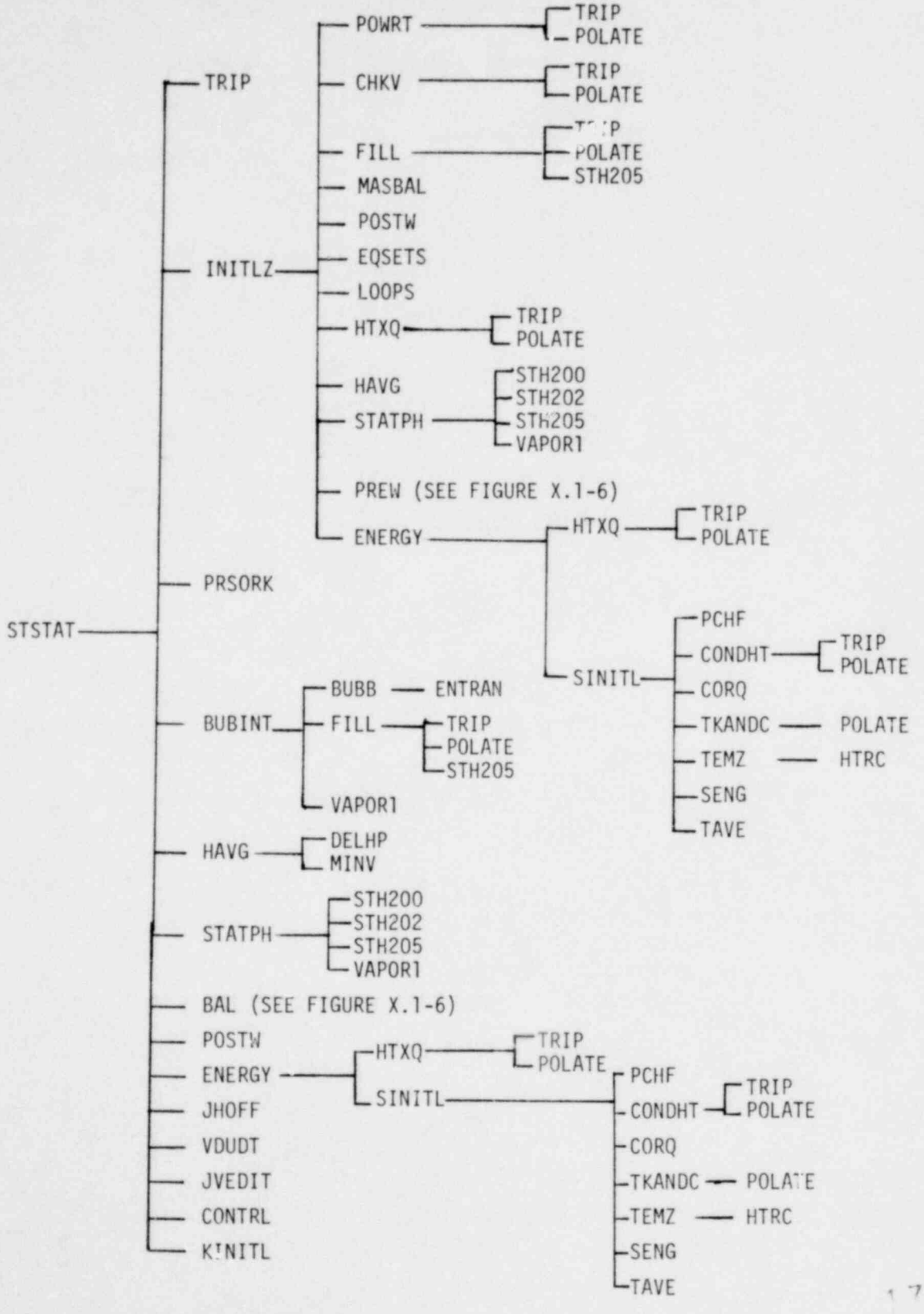

Figure X.1-4 Subroutine Calls from STSTAT

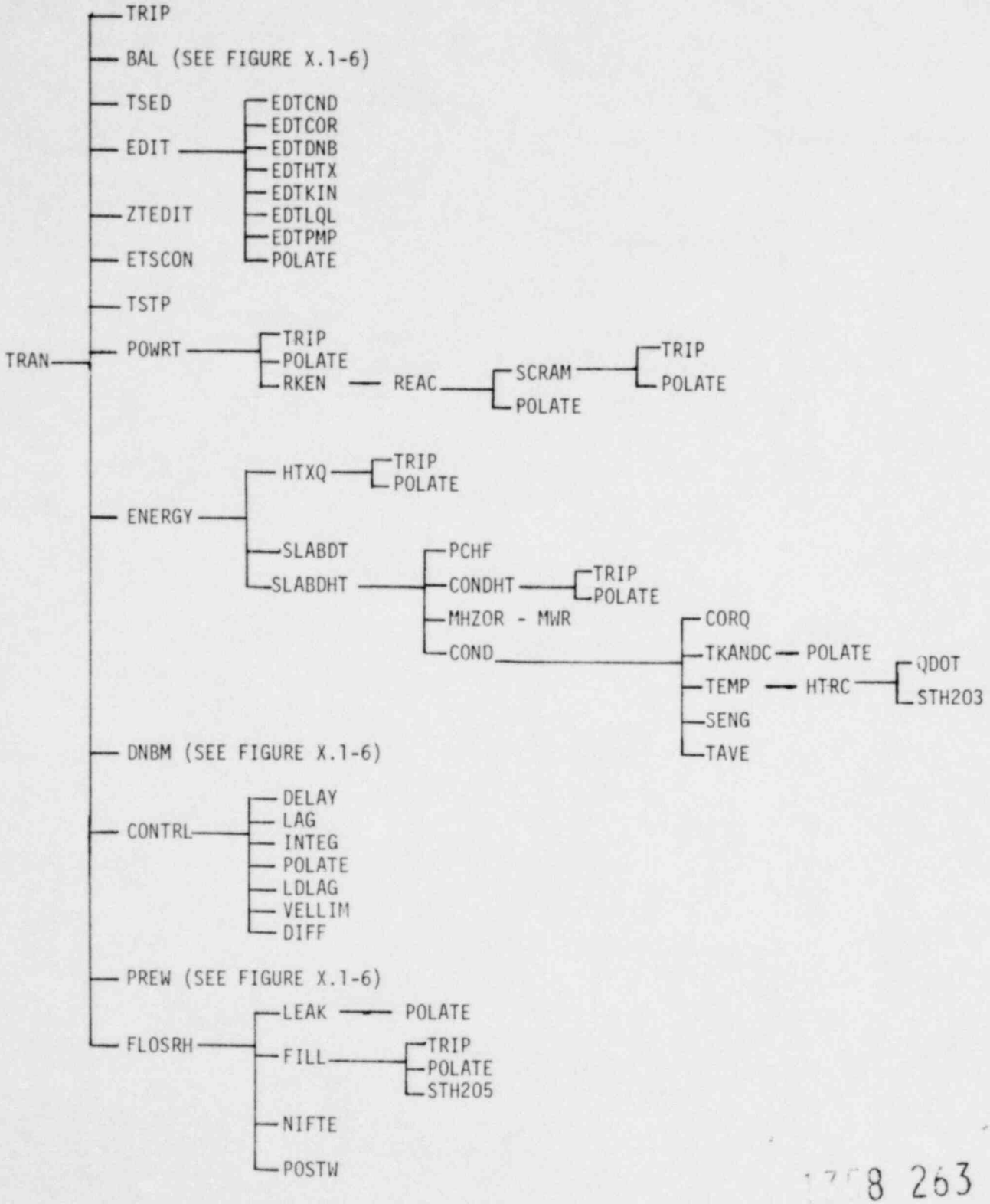

Figure X.1-5 Subroutine Calls from TRAN

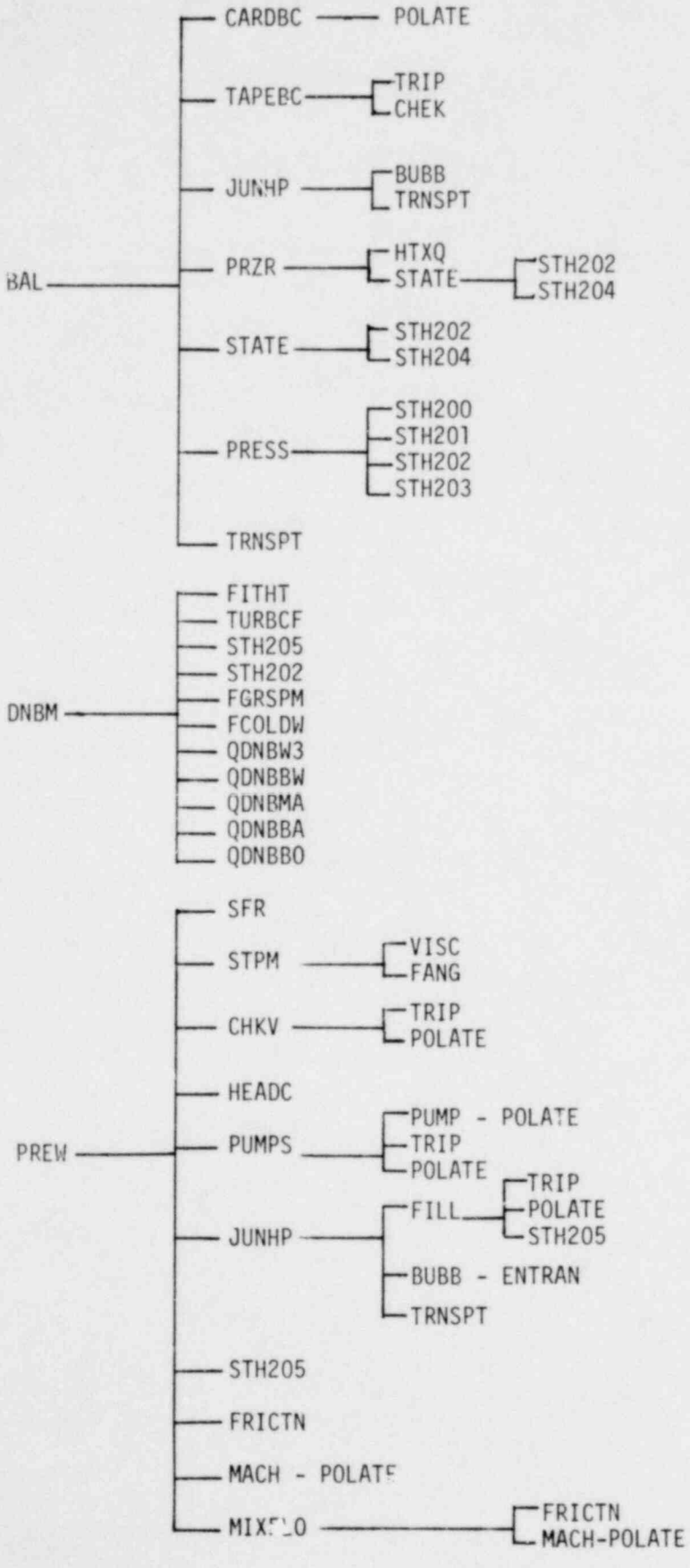

Figure X.1-6 Subroutine Calls from BAL, DNBM and PREW

XI. REFERENCES

I-1 "WREM: Water Reactor Evaluation model", Revision 1, NUREG-75/056, May 1975.

II.1-1 Fortran Extended Version 4 Reference Manual, Control Data Corp, #60305600, Revision E(5-10-74).

II.1-2 IBM OS FORTRAN IV (H-Extended) Compiler Programmer's Guide Order No. SC28-6852-2.

II.1-3 NOS/BE VERSION 1 REFERENCE MANUAL, Control Data Corporation, #60493800, Revision E (6-1-78).

II.1-4 LOADER REFERENCE MANUAL, Control Data Corporation, Publication No. 60344200, Revision G (3-75).

II.1-5 CYBER LOADER VERSION 1 REFERENCE MANUAL, Control Data Corporation, Publication Number 60429800, Revision F (April 15, 1978)

II.1-6 IBM OS Linkage Editor and Loader, Order Number GC28-6538-10.

VII.1-1 INEL Environmental Subroutine Manual, 1972.

VII.3-1 C. A. Mayer et al, "Thermodynamic and Transport Properties of Steam", The American Society of Mechanical Engineers, 1967.

VII.3-2 E. Schmidt, Properities of Water and Steam in SI Units, Springer-Verlag New York Inc., 1969.

VII.3-3 K. V. Moore, ASTEM -- A Collection of Fortran Subroutines to Evaluate the 1967 ASME Equations of STATE for Water/Steam and Derivatives of These Equations, ANCR-1026 (October 1971).

VII.4-1 RELAP4/MOD5 Volume II Program Implementation, ANCR-NUREG-1335 (Sept 1976), p. 237.

VII.4-2 Programming CALCOMP Electromechanical Plotters, California Computer Products, Inc, August 1974.

VIII.1-1 UPDATE REFERENCE MANUAL, Control Data Corporation, No. 60342500, Revision D(5-74).

178 267

APPENDIX A. RETRAN
FILE DIRECTORY

Appendix A

RETRAN FILE DIRECTORY

The RETRAN file directory is presented in this Appendix. Examples of file construction, including accessing the files and adding new variables to these files, are given in Section II.2.

```
*****************************************************************
*                      RETRAN FILE DIRECTORY                    *
*                                                               *
*   ALL STORAGE REQUIREMENTS FOR DATA ARRAYS ARE ASSIGNED AT    *
*   EXECUTION TIME, IN THE APPROPRIATE INPUT ROUTINES. ALL      *
*   STORAGE IS ASSIGNED WITHIN A SINGLE ARRAY. A UNIQUE         *
*   DESCRIPTION IS MAINTAINED FOR EACH FILE THROUGH USE OF THE  *
*   FOUR CONTROL ARRAYS DESCRIBED BELOW. SOME FILES MIGHT NOT   *
*   BE DEFINED DURING EXECUTION OF A GIVEN PROBLEM IF THE DATA  *
*   NORMALLY FOUND IN THE FILE IS NOT REQUIRED FOR PROBLEM      *
*   EXECUTION.                                                  *
*                                                               *
*            FILID (II) = UNIQUE FLOATING POINT FILE ID NUMBER  *
*            FILIDX(II) = UNIQUE INTEGER INDEX WHICH SPECIFIES  *
*                         THE ORIGIN OF THE FILE                *
*            FILSIZ(II) = AN INTEGER WORD SPECIFYING THE TOTAL  *
*                         LENGTH OF A GIVEN FILE                *
*            SETSIZ(II) = AN INTEGER WORD SPECIFYING THE SIZE   *
*                         OF A SUBSET OF DATA WITHIN A FILE.    *
*                         AS AN EXAMPLE, THE DATA FOR A GIVEN   *
*                         VOLUME FORMS A SET AND THE NUMBER OF  *
*                         SETS WOULD BE EQUAL TO THE NUMBER OF  *
*                         VOLUMES DEFINED. CONSEQUENTLY, THE    *
*                         FILSIZ MUST BE AN INTEGER MULTIPLE    *
*                         OF THE SETSIZ                         *
*                  II   = A FIXED INDEX ASSIGNED TO EACH FILE   *
*                                                               *
*   THE FILE DEFINITION AND STRUCTURE IS SHOWN BELOW. ALL       *
*   FILES LISTED, ARE PACKED AT THE PREFERED END OF CORE.       *
*                                                               *
*****************************************************************
```

```
**********************************************************************

*** FILID(01) = CARDS - CONTAINS CARD INPUT DATA PROCESSED BY
                THE INP PACKAGE. ONCE THE INPUT DATA IS PROCESSED,
                THE DATA IS WRITTEN TO FORTRAN UNIT 2
                FOR USE IN MULTIPLE CASE RUNS AND DATA
                TAPE HEADER RECORDS

**********************************************************************
```

```
**********************************************************************

*** FILID(02) = PRBDIM - PROBLEM DIMENSIONS
*** FILE 2 IS RESERVED IN SUBROUTINE INPUT

 WORD
 (01) = LDMP   = RESTART, PLOT TAPE CONTROL FLAG
 (02) = NEDI   = NUMBER OF MINOR EDIT PARAMETERS
 (03) = NTC    = TIME STEP CARD COUNT
 (04) = NTRP   = TRIP CARD COUNT
 (05) = NVOL   = NUMBER OF VOLUMES
 (06) = NBUB   = BUBBLE DATA CARD COUNT
 (07) = NTDV   = NUMBER OF TIME DEPENDENT VOLUMES
 (08) = NJUN   = NUMBER OF JUNCTIONS
 (09) = NPMPC  = NUMBER OF PUMPS
 (10) = NCKV   = CHECK VALVE CARD COUNT
 (11) = NLK    = NUMBER OF LEAK SETS
 (12) = NFLL   = NUMBER OF FILL SETS
 (13) = NSLB   = NUMBER OF HEAT CONDUCTORS
 (14) = NGOM   = NUMBER OF HEAT CONDUCTOR GEOMETRIES
 (15) = NMAT   = NUMBER OF HEAT CONDUCTOR MATERIALS
 (16) = NCOR   = NUMBER OF CORE SECTIONS
 (17) = NHTX   = NUMBER OF HEAT EXCHANGER SETS
 (18) = NTMM   = TWO STREAM MOMENTUM MIXING OPTION - JET PUMP
                 =0 NOT USED, >0 OPTION USED
 (19) = NODEI  = POWER CALCULATION TYPE
 (20) = MWREAC= METAL WATER CALCULATION FLAG
                 =0 NO M-W REAC, >0 M-W REAC CALC
 (21) = NLVC   = NUMBER OF VOLUMES TO BE SUMMED IN EQUIVALENT
                 LIQUID LEVEL CALCULATION
 (22) = MTDV   = NUMBER OF TIME DEPENDENT VOLUME BOUNDARY
                 CONDITIONS TO BE RETRIEVED FROM A DATA TAPE
 (23) = ISFLAG= FLAG TO TURN ON SLIP CALCULATION
 (24) = NCHT   = NUMBER OF CONDENSING STEAM HT CORRELATION DESCRIPTIONS
 (25) = JSST   = FLAG TO TURN OFF STEADY STATE INITIALIZATION, 1 FOR NO
                 INITIALIZATION, 0 FOR INITIALIZATION
 (26) = IPRZR  = NON-EQUILIBRIUM OPTION FLAG
 (27) = ITRNS  = TRANSPORT DELAY OPTION FLAG
 (28) = IDNBC  = AUXILIARY DNB OPTION FLAG
 (29) = ICF    = CONTROL SYSTEM OPTION FLAG

**********************************************************************
```

178 272

```
**********************************************************************

*** FILID(03) = EDVLOC - CONTAINS MINOR EDIT HEADINGS, EDIT
                VARIABLE ADDRESSES AND VALUES FOR UP TO 50
                TIME STEPS

*** EDVLOC CONTAINS 55 SETS, EACH SET IS (NEDI + 1) WORDS LONG
    AND CONTAINS INFO FOR EACH MINOR EDIT VARIABLE PLUS
    THE ELAPSED TIME

*** FILE 3 IS RESERVED IN SUBROUTINE EDINIT

 WORD
 (01) = MINOR EDIT VARIABLE AND REGION NUMBER
 (02) = 1ST HALF OF THE TITLE LINE          (A8)
 (03) = 2ND HALF OF THE TITLE LINE (A4)
 (04) = UNITS LINE OF TITLE (A8)
 (05) = CORE LOCATION USED TO FETCH VALUE OF EDIT VARIABLE
 (06) = VALUES OF MINOR EDIT VARIABLES FOR 50 TIME STEPS
 .
 .
 (55)

**********************************************************************
```

1778 273

```
**********************************************************************

*** FILID(04) = TSTEP  - TIME STEP INPUT DATA
*** FILE TSTEP HAS NTC DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 4 IS RESERVED IN SUBROUTINE INTSTP

 WORD
 (01) = NMIN = NUMBER OF TIME STEPS PER MINOR EDIT AND PLOT TAPE
               EDIT
 (02) = NMAJ = NUMBER OF MINOR EDITS PER MAJOR EDIT
 (03) = NDMP =  NUMBER OF MAJOR EDITS PER TIMING EDIT
 (04) = NCHK = TIME STEP CONTROL OPTION- 0,BLANK OR NEG=YES,POS=NO
 (05) = DELTM= MAXIMUM TIME STEP SIZE
 (06) = DTMIN= MINIMUM ALLOWABLE TIME STEP SIZE
 (07) = TLAST= END OF INTERVAL
 (08) = EPSMAX= MAXIMUM ERROR TOLERANCE IN FLOW SOLUTION
 (09) = EPSMIN= MINIMUM ERROR TOLERANCE IN FLOW SOLUTION
 (10) = EPST  = ERROR TOLERANCE IN HEAT TRANSFER SOLUTION
 (11) = EPSCVL= ERROR TOLERANCE IN CAUSAL VOLUME TREATMENT
 (12) = FLGI  = MINIMUM CHANGE IN RELATIVE PRESSURE THAT ACTIVATES
                THE IMPLICIT ITERATIVE SOLUTION
 (13) = EPSI  = CONVERGENCE CRITERA IN PRESSURE FOR THE IMPLICIT SOLUTION
 (14) = IEVERY = DETAILED EDIT CONTROL

**********************************************************************
```

```
**********************************************************************

*** FILID(05) = VOLUME - VOLUME QUANTITIES
*** FILE VOLUME HAS NVOL DATA SETS. A DATA SET IS SHOWN BELOW
*** FILE 5 IS RESERVED IN SUBROUTINE INVOL

  WORD          VARIABLE
  (01) = IBUB = BUBBLE DATA INDEX
  (02) = IREAD= VOLUME DATA RETRIEVAL INDEX
  (03) = P    = VOLUME PRESSURE
  (04) = TEMP = VOLUME TEMPERATURE
  (05) = HUM   = VOLUME HUMIDITY
  (06) = V    = VOLUME (FT3)
  (07) = ZVOL = VOLUME HEIGHT
  (08) = ZM   = MIXTURE LEVEL
  (09) = JTPMV= TWO PHASE FRICTION INDEX
  (10) = FLOWA =VOLUME FLOW AREA
  (11) = DIAMV =EQUIVALENT DIAMETER OF FLOW AREA
  (12) = ELEV  =ELEVATION AT THE BOTTOM OF THE VOLUME
  (13) = SATP  =SATURATION PRESSURE
  (14) = SATT  =SATURATION TEMPERATURE
  (15) = SATVF =SATURATION SPECIFIC VOLUME OF LIQUID
  (16) = SATVG =SATURATION SPECIFIC VOLUME OF GAS
  (17) = SATUF =SATURATED LIQUID SPECIFIC INTERNAL ENERGY
  (18) = SATUG =SATURATED GAS SPECIFIC INTERNAL ENERGY
  (19) = SATHF =SPECIFIC ENTHALPY OF SATURATED LIQUID
  (20) = SATHG =SPECIFIC ENTHALPY OF SATURATED GAS
  (21) = VL    =SPECIFIC LIQUID VOLUME
  (22) = VS    =SPECIFIC GAS VOLUME
  (23) = UW    =INTERNAL SPECIFIC ENERGY
  (24) = HW    =VOLUME SPECIFIC ENTHALPY
  (25) = GASH  =GAS SPECIFIC ENTHALPY
  (26) = LIQH  =LIQUID SPECIFIC ENTHALPY
  (27) = ZL    =LIQUID LEVEL
  (28) = AVEX  =AVERAGE QUALITY
  (29) = ZMO   =MIXTURE LEVEL FOR PREVIOUS TIME STEP
  (30) = GASM  =GAS MASS (INCLUDES BUBBLE MASS)
  (31) = FMASS =TOTAL WATER MASS
  (32) = ARMASS=AIR MASS
  (33) = BUBM  =BUBBLE MASS
  (34) = LIQM  =LIQUID MASS
  (35) = MIXV  =MIXTURE VOLUME
  (36) = GASV  =GAS VOLUME
  (37) = LIQV  =LIQUID VOLUME
  (38) = AVED  =AVERAGE DENSITY
  (39) = U     =TOTAL ENERGY
  (40) = MIXQ  =MIXTURE QUALITY
  (41) = A     =VOLUME AREA
  (42) = FLOWL =VOLUME FLOW PATH LENGTH
  (43) = FLOWRA=1/(144*GC*FLOWA**2)
  (44) = WVBAR =AVERAGE VOLUME FLOW (LBM/SEC)
  (45) = WVBARO=PREVIOUS TIME STEP VALUE OF WVBAR
  (46) = SPVZ  =VOLUME AVERAGE SPECIFIC VOLUME (FT3/LBM)
  (47) = SPVZO =PREVIOUS TIME STEP VALUE OF SPVZ
```

178 275

(48) = AINERV=HALF-VOLUME INERTIA (L/2*A*144*GC)
(49) = CMAS =CENTER OF MASS HEIGHT
(50) = FANING=FANNING FRICTION FACTOR
(51) = ELOD =(L/D)/(144*GC*A**2)
(52) = TPMV =TWO-PHASE MULTIPLIER FOR FANNING FRICTION TERMS
(53) = CMACHV=MACH NUMBER SQUARED FOR AVERAGE VOLUME FLOW
(54) = IFAN =MEMORY INDEX FOR FANNING FRICTION CALCULATION
(55) = JVISC =MEMORY INDEX FOR VISCOSITY CALCULATION
(56) = TLIQ =LIQUID TEMPERATURE (TWO TEMPERATURE MODEL)
(57) = SOUND2=ISENTROPIC SONIC VELOCITY SQUARED, DP/DR;HHO
(58) = VSTIDX=MEMORY INDEX FOR VOLUME STATE PROPERTIES
(59) = SATIDX=MEMORY INDEX FOR SATURATED STATE PROPERTIES
(60) = PHASE =PHASE OF WATER IN VOLUME
(61) = IIN =JUNCTION INLET NUMBER (NEW)
(62) = IOUT =JUNCTION OUTLET NUMBER (NEW)
(63) = CSUBP =SPECIFC HEAT CAPACITY AT CONSTANT PRESSURE
(64) = CSUBPF=CSUBP FOR SATURATED WATER
(65) = CSUBPG=CSUBP FOR SATURATED STEAM
(66) = IQIN =HEAT CALCULATION INDEX
(67) = A1 =SLOPE OF GAS BUBBLE DENSITY
(68) = B =INTERCEPT OF GAS BUBBLE DENSITY
(69) = E =SLOPE OF MIXTURE DENSITY
(70) = F =INTERCEPT OF MIXTURE DENSITY
(71) = RGBT =BUBBLE FRACTION AT MIXTURE SURFACE
(72) = DPDU =DP/DU AT CONSTANT M, MA
(73) = DPDMF =DP/DMF AT CONSTANT U, MA
(74) = DPDMA =DP/DMA AT CONSTANT U, MF
(75) = FU =DU/DT = ENERGY EQUATION
(76) = FM =DM/DT = MASS EQUATION
(77) = FMA =DMA/DT = AIR MASS EQUATION
(78) = WQ =POWER INTO COOLANT
(79) = QSPL =VOLUME HEAT SOURCE
(80) = ITWOT = INITIAL VOLUME NUMBER TREATED BY TWO TEMP. MODEL
(81) = IVFROM = VOLUME FROM WHICH TWO TEMP. MODEL IS PROPOGATED
(82) = OLVOLN = OLD VOLUME NUMBER
(83) = GFLAG = FLAG FOR PRESENCE OF GAS OR AIR IN THE VOLUME
(84) = VGASV = VOLUME GAS VELOCITY
(85) = VSLPVO = VOLUME SLIP VELOCITY
(86) = JTPR = PRESSURE TABLE INDEX FOR 2-PHASE MULTIPLIER CALC.
(87) = JTX1 = FIRST QUALITY TABLE INDEX FOR 2-PHASE MULTIPLIER CALC.
(88) = JTX2 = SECOND QUALITY TABLE INDEX FOR 2-PHASE MULTIPLIER CALC.
(89) = IVEDIT = NUMBER OF TIMES VOLUME HAS BEEN WATER PACK
(90) = OLDTEM = OLD VOLUME TEMPERATURE
(91) = OLDP = OLD VOLUME PRESSURE
(92) = XUO = OLD VOLUME ENERGY
(93) = XMFO = OLD VOLUME FLUID MASS
(94) = XMAO = OLD VOLUME AIR MASS
(95) = BUBMO = OLD VOLUME BUBBLE MASS
(96) = ICVOL = CAUSAL VOLUME INDICATOR,YES=1
(97) = XGAMA1= CREDIT IN ENERG FOR NONCAUSAL VOLUME
(98) = XGAMA2= CREDIT IN FLUID MASS FOR NONCAUSAL VOLUME
(99) = XGAMA3= CREDIT IN AIR MASS FOR NOCAUSAL VOLUME
(100) = TLIQO = OLD LIQUID TEMPERATURE (TWO TEMPERATURE MODEL)
(101) = VOIDV = VOLUME VOID FRACTION

8 276

```
(102) = WGV    = VOLUME GAS FLOW
(103) = WLV    = VOLUME LIQUID FLOW
(104) = VSLPV  = VOLUME SLIP VELOCITY
(105) = QVBAR  = AVERAGE VOLUMETRIC FLOW
(106) = QVBARO= OLD AVERAGE VOLUMETRIC FLOW
(107) = SPVZC  = SPECIFIC VOLUME AT VOLUME GEOMETRIC CENTRE
(108) = IFR    = FLOW REGIME INDEX
(109) = INEQ   = NON-EQUILIBRIUM VOLUME NUMBER
(110) = VR     = RAINOUT VELOCITY (NON-EQUILIBRIUM VOLUME)
(111) = IPTN   = PIPE TRANSPORT VOLUME NUMBER

*****************************************************************
```

```
**********************************************************************

*** FILID(06) = DJUNCT - JUNCTION QUANTITIES
*** FILE DJUNCT HAS NJUN DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 6 IS RESERVED IN SUBROUTINE INJUN

 WORD
 (01) = IW1    = VOLUME NUMBER AT JUNCTION INLET
 (02) = IW2    = VOLUME NUMBER AT JUNCTION OUTLET
 (03) = IPUMP  = PUMP CURVE NUMBER
 (04) = IVALVE = VALVE NUMBER
 (05) = WP     = JUNCTION WEIGHT FLOW
 (06) = AJUN   = JUNCTION FLOW AREA
 (07) = ZJUN   = JUNCTION ELEVATION
 (08) = INERTA = JUNCTION INERTIA
 (09) = FJUNF  = SPECIFIC ENERGY LOSS COEFFICIENT FOR FORWARD FLOW
 (10) = FJUNR  = SPECIFIC ENERGY LOSS COEFFICIENT FOR REVERSE FLOW
 (11) = JVERTL = VERTICAL JUNCTION CONTROL INDEX
 (12) = JCHOKE = JUNCTION CHOKING INDEX
 (13) = JCALCI = INITIAL CONDITION CALCULATION TYPE CONTROL INDEX
 (14) = MVMIX  = MOMENTUM MIXING INDEX
 (15) = DIAMJ  = JUNCTION DIAMETER
 (16) = CONCO  = CONTRACTION COEFFICIENT
 (17) = ICHOKE = LIQUID PHASE CHOKING CONDITION (0=NO CHOKING)
 (18) = IHQCOR = ENTHALPY TRANSPORT INDEX
 (19) = DELA   = PRESSURE DIFFERENTIAL - ACCELERATION
 (20) = DELE   = PRESSURE DIFFERENTIAL - ELEVATION
 (21) = DELF   = PRESSURE DIFFERENTIAL - FRICTION
 (22) = XP     = JUNCTION QUALITY
 (23) = HP     = JUNCTION FLUID ENTHALPY
 (24) = ITP    = CHOKING TYPE INDICATOR
 (25) = PUMPP  = PUMP HEAD
 (26) = FMFRAC = MASS FRACTION OF WATER IN JUNCTION FLOW
 (27) = DELP   = PRESSURE DIFFERENTIAL - VOLUME TO VOLUME
 (28) = SPVJ   = JUNCTION SPECIFIC VOLUME
 (29) = AJUNT  = TIME DEPENDENT JUNCTION FLOW AREA
 (30) = KCHOKE = CHOKING INDICATOR (0=UNCHOKED)
 (31) = CHKVK  = CHECK VALVE FRICTION FACTOR
 (32) = IJ     = MIXING JUNCTION NUMBER
 (33) = IK     = MIXING VOLUME NUMBER
 (34) = PSTAR1 = UNMIXED PRESSURE AT JUNCTION OUTLET
 (35) = PSTAR2 = UNMIXED PRESSURE AT JUNCTION INLET
 (36) = VGASJ1 = JUNCTION GAS VELOCITY
 (37) = JVLPJO = PREVIOUS TIME STEP JUNCTION SLIP VELOCITY
 (38) = SPVST1 = SPECIFIC VOLUME AT JUNCTION OUTLET
 (39) = SPVST2 = SPECIFIC VOLUME AT JUNCTION INLET
 (40) = BWSQR1 = COEFFICIENT OF WJ**2 FROM MACH
 (41) = BWSQR2 = FANNING FRICTION COEFFICIENT OF W**2
 (42) = AVEDJ  = AVERAGE JUNCTION DENSITY
 (43) =        =
 (44) =        =
 (45) = HKIN   = JUNCTION KINETIC ENERGY
 (46) = IVAP   = JUNCTION VAPOR PRESSURE INDEX
 (47) = SPVJO  = PREVIOUS TIME STEP JUNCTION SPECIFIC VOLUME
```

```
(48) = RESDK   = RESIDUAL JUNCTION FRICTION FACTOR
(49) = JPEDIT  = NUMBER OF TIMES JUNCTION J HAS BEEN MODIFIED
                 (WATER PACKING)
(50) = TPMJ    = 2-PHASE MULTIPLIER FOR FORM LOSS
(51) = CMACH1  = JUNCTION MACH NUMBER
(52) = KMACH   = MACH TABLE MEMORY INDEX
(53) = HEADR1  = HEAD TERM FOR JUNCTION INLET
(54) = HEADR2  = HEAD TERM FOR JUNCTION OUTLET
(55) = IH      = ENTH. INDEX FOR MOODY CRITICAL FLOW TABLES
(56) = IP      = PRES. INDEX FOR MOODY CRITICAL FLOW TABLES
(57) = IHH     = ENTH. INDEX FOR HENRY CRITICAL FLOW TABLES
(58) = IPH     = PRES. INDEX FOR HENRY CRITICAL FLOW TABLES
(59) = IHHE    = ENTH. INDEX FOR EXTENDED HENRY CRIT. FLOW TABLES
(60) = IPHE    = PRES. INDEX FOR EXTENDED HENRY CRIT. FLOW TABLES
(61) = WJSUM   = UNUSED
(62) = HSPIN   = EQUIVALENT JUNCTION INLET ENTHALPY
(63) = HSPOUT  = EQUIVALENT JUNCTION OUTLET ENTHALPY
(64) = IW1N    = NEW VOLUME NUMBER AT JUNCTION INLET
(65) = IW2N    = NEW VOLUME NUMBER AT JUNCTION OUTLET
(66) = IPUMPN  = NEW PUMP CURVE NUMBER
(67) = OLJUNN  = OLD JUNCTION NUMBER
(68) = HOLDA   = OLD JUNCTION ENTHALPY SAVE ARRAY
(69) = ESUBKO  = OLD ESUBK
(70) = FSUBK   = FRICTION TERM K/RHO FOR MOMENTUM EQUATION
(71) = ESUBK   = DW/DT MOMENTUM EQUATION
(72) = CMACHJ  = MACH NUMBER SQUARED AT JUNCTION
(73) = PUMPK   = PUMP FRICTION FACTOR
(74) = WAVG    = TIME AVERAGED FLOW
(75) = DW1     = FLOW CHANGE (PREVIOUS TIME STEP)
(76) = DW2     = PROJECTED MAXIMUM FLOW CHANGE
(77) = DW      = ABS ( DW1 - DW2 )
(78) = DWOLD   = DW (PREVIOUS TIME STEP)
(79) = WOLD    = OLD FLOW
(80) = HPL     = JUNCTION LIQUID ENTHALPY
(81) = HPG     = JUNCTION GAS ENTHALPY
(82) = SPVJL   = JUNCTION LIQUID SPECIFIC VOLUME
(83) = SPVJG   = JUNCTION GAS SPECIFIC VOLUME
(84) = VOID    = JUNCTION VOID FRACTION
(85) = VSLPJ   = JUNCTION SLIP VELOCITY
(86) = WLJ     = JUNCTION LIQUID FLOW
(87) = WGJ     = JUNCTION GAS FLOW
(88) = ISP     = SPRAY JUNCTION FLAG
```

****

8 279

```
**********************************************************************

*** FILID(07) = SYSTEM - OVERAL SYSTEM QUANTITIES
*** FILE 7 IS RESERVED IN SUBROUTIN INCNST

 WORD
 (01) = NOGO   = ERROR FLAG
 (02) = OMEGA  = IMPLICIT-EXPLICIT NUMBERS CONSTANT,I.D#FULLY IMPLICIT
 (03) = POWRI  = INITIAL POWER LEVEL
 (04) = AE     = ENERGY ADDED
 (05) = AMASS  = MASS ADDED
 (06) = BMASSW = MASS BALANCE OF WATER
 (07) = BMASSA = MASS BALANCE OF AIR
 (08) = DMASSW = MASS OF WATER LEAKED
 (09) = DMASSA = MASS OF AIR LEAKED
 (10) = EB     = ENERGY BALANCE
 (11) = FE     = ENERGY IN FUEL
 (12) = HE     = ENERGY EXTRACTED BY HEAT EXCHANGER
 (13) = PNORM  = NORMALIZED REACTOR POWER
 (14) = QLOSS  = TOTAL RATE OF HEAT REMOVAL
 (15) = UFILL  = ENERGY FROM FILLS
 (16) = ULOSS  = ENERGY LEAKED
 (17) = TIMEX  = PROBLEM TIME
 (18) = POWER  = SYSTEM POWER
 (19) = PNUCL  = INITIAL NUCLEAR POWER LEVEL
 (20) = PTHERM = INITIAL THERMAL POWER LEVEL
 (21) = MAXNOD = MAXIMUM NUMBER OF NODES SPECIFIED FOR ANY CONDU.
 (22) = PMPPOM = TOTAL HEAT ADDITION TO THE SYSTEM FROM THE PUMPS
 (23) = TIMEX1 = TIMEX ONE TIME STEP BACK
 (24) = TIMEX2 = TIMEX TWO TIME STEPS BACK
 (25) = LCOUNT = MAX. NUMBER OF STEADY STATE ITERATIONS
 (26) = ACEPSI = ACCELERATION PSI CONVERGENCE CRITERIA
 (27) = HEPSII = VOLUME ENERGY BALANCE CONVERGENCE CRITERIA
 (28) = EPSIMI = MASS BALANCE CONVERGENCE CRITERIA
 (29) = ICEQST = FLOW NETWORK NUMBER TO WHICH THE CORE SECTIONS
                 BELONG

**********************************************************************
```

8 280

```
**********************************************************************

*** FILID(08) = TRIP - TRIP VARIABLES
*** FILE TRIP HAS NTRP DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 8 IS RESERVED IN SUBROUTINE INTRIP

 WORD
 (01) = IDTRP = INDEX FOR TRIP TYPE
 (02) = IDSIG = INPUT SIGNAL INDEX
 (03) = IX1   = SIGNAL 1 INDEX (VOLUME OR JUNCTION NUMBER)
 (04) = IX2   = SIGNAL 2 INDEX (NONZERO FOR DIFFERENCES  )
 (05) = SETPT = SIGNAL SETPOINT, TYPE ACCORDING TO IDSIG
 (06) = DELAY = TRIP DELAY AFTER SETPOINT IS PASSED
 (07) = TSIG  = TIME TO REACH SETPOINT
 (08) = FLOG  = TRUE MEANS SETPOINT REACHED
 (09) = ABX   = PREVIOUS TIME STEP VALUE OF TRIP VARIABLE

**********************************************************************
```

8 281

```
**********************************************************************

*** FILID(09) = STEAMT - FILE CONTAINING THE WATER THERMODYNAMIC
                 PROPERTY TABLE
*** FILE 9 IS RESERVED IN SUBROUTINE STH20I

 WORD
 (01) = THERMO = THERMODYNAMIC PROPERTY TABLES

**********************************************************************
```

178 282

```
**********************************************************************

*** FILID(10) = EDTFIL - RETRAN  DOCUMENTATION AND PROBLEM TITLE.
*** FILE 10 IS RESERVED IN SUBROUTINE INPUT

 WORD
 (01) =          = HEADER LABEL
 (02) =          =
 (03) =          =
 (04) =          =
 (05) =          = DOCUMENTATION HEADER
 (06) =          =
 (07) =          =
 (08) =          =
 (09) =          =
 (10) =          =
 (11) =          =
 (12) =          =
 (13) =          =
 (14) =          =
 (15) =          =
 (16) =          =
 (17) =          =
 (18) =          =
 (19) =          = DATE
 (20) =          = PROBLEM TITLE
 (21) =          =
 (22) =          =
 (23) =          =
 (24) =          =
 (25) =          =
 (26) =          =
 (27) =          =
 (28) =          =
 (29) =          =
                   WORDS 30 AND ON WILL NOT BE PRESENT IF A
                   DATA TAPE IS NOT GENERATED
 (30) =          = TRAILER LABEL
 (31) =          =
 (32) =          =
 (33) =          =
 (34) =          = 8HDATA REC
 (35) =          = DATA RECORD NUMBER
 (36) =          = LENGTH OF DATA RECORD
 (37) =          = TAPE LABEL
 (38) =          =
 (39) =          = TAPE VSN
 (40) =          = JULIAN CREATION DATE
   .
   .
   .               REEDIT  MAY REQUIRE MULTIPLE PAIRS OF
                   VSN AND CREATION DATE

**********************************************************************
```

```
**********************************************************************

*** FILID(11) = BUBBLE - CONTAINS BUBBLE SET DATA
*** FILE BUBBLE HAS NBUB=1 DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 11 IS RESERVED IN SUBROUTINE INBUBL

    FIRST SET CONTAINS ALPH = 0. AND VBUB = 0.

 WORD
 (01) =  ALPH  = BUBBLE GRADIENT PARAMETER
 (02) =  VBUB  = BUBBLE VELOCITY

**********************************************************************
```

```
**********************************************************************

*** FILID(12) = VALVES - CONTAINS VALVE DATA
*** FILE VALVES HAS NCKV DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 12 IS RESERVED IN SUBROUTINE INCKV

 WORD
 (01) = ITCV   = TYPE OF CHECK VALVE
 (02) = IACV   = INDEX FOR AREA VS TIME OR ANGLE TABLE
 (03) = PCV    = BACK PRESSURE TO CLOSE CHECK VALVE
 (04) = CV1    = FORWARD FLOW FRICTION FACTOR OR DOOR AREA TIMES
                 MOMENT OF INERTIA
 (05) = CV2    = REVERSE FLOW FRICTION FACTOR, VALVE OPEN
                 OR DOOR MOMENT OF INERTIA
 (06) = CV3    = REVERSE FLOW FRICTION FACTOR, VALVE CLOSED
                 OR DOOR DAMPING CONSTANT
 (07) = THDOT  = DOOR ANGLE VELOCITY
 (08) = THETA  = DOOR ANGLE
 (09) = OPEN   = .TRUE. MEANS CHECK VALVE IN OPEN POSITION

**********************************************************************
```

```
**********************************************************************

*** FILID(13) = TDAREA - CONTAINS TIME DEP. AREA DATA
*** FILE TDAREA HAS NLK DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 13 IS RESERVED IN SUBROUTINE INAREA

 WORD
 (01) = NAREA = NUMBER OF NORMALIZED AREA VS TIME DATA POINTS
                OR NORMALIZED AREA VS ANGLE DATA POINTS
 (02) = IAREA = CURRENT POSITION IN AREA VS TIME TABLE
 (03) = IDXARE= INDEX OF FILE CONTAINING NORM. AREA TABLE

*** FILE AREATL IS A SUBFILE OF TDAREA, CONTAINS AREA TABLE
 WORD
 (01) = TAREA(1) = TIME OR ANGLE
 (02) = TAREA(1) = AREA NORMALIZED TO FULL OPEN
  .
  .
  .
 (2*NAREA-1) = TAREA(NAREA) = TIME OR ANGLE
 (2*NAREA)   = TAREA(NAREA) =  AREA NORMALIZED TO FULL OPEN

**********************************************************************
```

1758 286

```
**********************************************************************

*** FILID(14) = FILLER - CONTAINS FILL DATA
*** FILE FILLER HAS NFLL DATA STES.  A D1TA SET IS SHOWN HELOW
*** FILE 14 IS RESERVED IN SUBROUTINE INFILL

 WORD
 (01) = NFILL  = NUMBER OF PAIRS IN FILL TABLE
 (02) = ITFILL = TRIP NUMBER CONTROLLING FILL
 (03) = JX     = INDEPENDENT VARIABLE TYPE(-1=DIFF PRES,0=TIME,1=VOL PRES)
 (04) = JY     = DEPENDENT VARIABLE TYPE (0=LB/SEC,1=GALL/MIN)
 (05) = IFILL  = CURRENT TABLE POSITION
 (06) = IDXFLT = INDEX OF FILE CONTAINING FILL TABLE
 (07) = PFILLO = PREVIOUS TIME STEP FILL PRESSURE
                 IF JX.GE.0,CONSTANT FILL PRESSURE IF JX.LT.0
 (08) = IDXENT = INDEX OF ENTHALPY SUBFILE
 (09) = STHIDX = MEMORY INDEX FOR FILL STATE PROPERTIES
 (10) = IDXPRS = INDEX OF PRESSURE SUBFILE - DEFINED ONLY IF JX .GE. 0

*** FILID FILTBL IS A SUBFILE OF FILLER, CONTAINS FILL TABLE
 WORD
 (01)       = FILTBL(1)      = TIME OR PRESSURE
 (02)       = FILTBL(1)      = FLOW
  .
  .
  .
 (2*NFILL-1)= FILTBL(NFILL) = TIME OR PRESSURE
 (2*NFILL)  = FILTBL(NFILL) = FLOW


*** ENTHALPY CORRESPONDENCE S BFILE
    (EACH ITEM CORRESPONDS TO W VS T OR P PAIR IN FILTBL)
 WORD
 (01) = FILENT(1) = FILL ENTHALPY
 (02) = FILENT(2) =
  .
  .
  .
 (NFILL) = FILENT(NFILL) = FILL ENTHALPY

*** PRESSURE CORRESPONDENCE SUBFILE
    (EACH ITEM CORRESPONDS TO W VS T OR P PAIR IN F'LTBL)
 WORD
 (01) = FILPRS(1) = FILL PRESSURE
 (02) = FILPRS(2) =
  C       *** NOTE - FOR JX = -1 THIS TABLE IS NOT DEFINDED
  .
  .
 (NFILL) = FILPRE(NFILL) = FILL PRESSURE
**********************************************************************
```

178 287

```
**********************************************************************

*** FILID(15) = PMPDSC - PUMP DESCRIPTION AND STOP DATA
*** FILE PMPDSC HAS NPMP DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 15 IS RESERVED IN SUBROUTINE INPUMS

 WORD
 (01) = IPC     = PUMP CURVE SET INDICATOR
 (02) = ITPUMP  = TRIP NUMBER TO SHUT OFF PUMP
 (03) = IRP     = PUMP REVERSE INDICATOR (1 = REVERSE ALLOWED)
 (04) = IPM     = INDEX FOR TWO-PHASE OPTION (0=NO - 1=YES)
 (05) = IMT     = INDEX FOR PUMP MOTOR TORQUE CURVE
 (06) = POMGAR  = RATED PUMP SPEED
 (07) = PSRAT   = RATIO OF INITIAL TO RATED SPEED
 (08) = PFLOWR  = RATED PUMP FLOW
 (09) = PHEADR  = RATED PUMP HEAD
 (10) = PTORKR  = RATED PUMP TORQUE
 (11) = PINRTA  = PUMP MOMENT OF INERTIA
 (12) = VRHOI   = RATED OR INITIAL DENSITY
 (13) = TORKMR  = RATED PUMP MOTOR TORQUE
 (14) = TORKF1  = TORKF1 THROUGH TORKF4
 (15) = TORKF2  = ARE COEFFICIENTS FOR
 (16) = TORKF3  = FRICTIONAL TORQUE AS A CUBIC
 (17) = TORKF4  = FUNCTION OF THE SPEED RATIO.
 (18) = CAVCON  = PUMP STOP ON ELAPSED TIME
 (19) = FPUMP   = PUMP STOP ON MAXIMUM FORWARD SPEED
 (20) = SPUMP   = PUMP STOP ON MAXIMUM REVERSE SPEED
 (21) = TORKFR  = FRICTIONAL TORQUE AT RATED SPEED
 (22) = TORQUE  = NORMALIZED PUMP TORQUE
 (23) = IMZ     = MEMORY INDEX FOR MOTOR TORQUE CURVE TABLE
 (24) = IVPUMP  = PUMP VOLUME NUMBER
 (25) = JDISRG  = PUMP DISCHARGE JUNCTION NUMBER
 (26) = JSUCTN  = PUMP SUCTION JUNCTION NUMBER
 (27) = JPHD    = MEMORY INDEX FOR HEAD CURVE TABLE
 (28) = JPHD4   = MEMORY INDEX FOR SINGLE-2 PHASE HEAD CURVE TABLE
 (29) = JPHM    = MEMORY INDEX FOR PUMP HEAD MULT. CURVE TABLE
 (30) = JPTK    = MEMORY INDEX FOR TORQUE CURVE TABLE
 (31) = JPTK4   = MEMORY INDEX FOR SINGLE-2 PHASE TORQ CURVE TABLE
 (32) = JPTM    = MEMORY INDEX FOR PUMP TORQUE MULT. CURVE TABLE
 (33) = POMGA   = ACTUAL PUMP ANGULAR VELOCITY
 (34) = OLPMPN  = OLD PUMP NUMBER
 (35) = IVPMPO  = OLD PUMP VOLUME NUMBER
 (36) = PMTORQ  = NORMALIZED PUMP MOTOR TORQUE
 (37) = PFTORQ  = NORMALIZED PUMP FRICTIONAL TORQUE
 (38) = PMPOWR  = POWER TO COOLANT DUE TO IRRECOVERABLE LOSSES
 (39) = IPEQST  = FLOW NETWORK NUMBER TO WHICH THE PUMP BELONGS

**********************************************************************
```

```
*****************************************************************

*** FILID(16) = PMTORK-CONTAINS INDEXES OF PUMP MOTOR TORK CURVES
*** FILE 16 IS RESERVED IN SUBROUTINE INPUMS

 WORD
 (01) = IDXPM1 = INDEX OF PUMP MOTOR TORQUE CURVE 1
 (02) = IDXPM2 = INDEX OF PUMP MOTOR TORQUE CURVE 2

*** FILID - PTORK1 IS A SUBFILE OF PMTORK, CONTAINS PUMP MOTOR
            TORQUE CURVE 1
 WORD
 (01) = NTM01 = NUMBER OF POINTS IN PUMP MOTOR TORQUE CURVE 1
 (02) = PTM0(1) = PUMP SPEED
 (03) = PTM0(1) = MOTOR TORQUE
  .
  .
  .                  .
 (2*NTM01)    =  PTM0(NTM01) = PUMP SPEED
 (2XNTM01+1) =  PTM0(NTM01) = MOTOR TORQUE

*** FILID - PTORK2 IS A SUBFILE OF PMTORK, CONTAINS PUMP MOTOR
            TORQUE CURVE 2
 WORD
 (01) = NTM02 = NUMBER OF POINTS IN PUMP MOTOR TORQUE CURVE 2
 (02) = PTM0(1) = PUMP SPEED
 (03) = PTM0(1) = MOTOR TORQUE
  .
  .
  .
 (2*NTM02)    = PTM0(NTM02) = PUMP SPEED
 (2*NTM02+1) = PTM0(NTM02) = MOTOR TORQUE

*****************************************************************
```

178 289

```
*********************************************************************

*** FILID(17) = PMPDEG - PUMP HEAD AND TORQUE MULTIPLIER
*** DATA FOR DEGRADATION CURVES
*** FILE 17 IS RESERVED IN SUBROUTINE INPUMS

 WORD
 (01) = ICURVE = INDEX OF FILE CONTAINING PUMP DEGRADATION CURVE
 (02) = IDXPHM = INDEX OF FILE CONTAINING HEAD MULTIPLIER CURVE
 (03) = IDXPTM = INDEX OF FILE CONTAINING TORQUE MULTIPLIER CURVE
  .
  .
  .
*** FILE PHDMLT IS A SUBFILE OF PMPDEG
 (01) = NPHM   = NUMBER OF DATA POINTS IN THE PUMP HEAD MULT. CURVE
 (02) = PHDM   = PUMP HEAD MULTIPLIER CURVE TABLE
*** FILE PTRMLT IS A SUBFILE OF PMPDEG
 (01) = NPTM   = NUMBER OF DATA POINTS IN THE PUMP TORQ. MULT CURVE
 (02) = PTRM   = PUMP TORQUE MULTIPLIER CURVE TABLE

*********************************************************************
```

8 290

```
**********************************************************************

*** FILID(18) = TIMFIL - CONTAINS SUBROUTINE TIMING INFORMATION
*** FILE 18 IS RESERVED IN SUBROUTINE TIMINT
 WORD
 (01) =          = ELAPSED CPU TIME FOR SUBROUTINE
 (02) =          = NUMBER OF CALLS FOR SUBROUTINES.
  .
  .
  .
 (2*N-1)         = ELAPSED CPU TIME FOR SUBROUTINE
 (2*N)           = NUMBER OF CALLS FOR SUBROUTINE.

**********************************************************************
```

8 291

```
*********************************************************************

*** FILID(19) = PPHDTQ - CONTAINS INDEXES OF EACH CURVE SET FILE
*** FILE 19 IS RESERVED IN SUBROUTINE INPUMP

 WORD
 (01) = IDXP1 = INDEX OF FILE CRVSET-1
 (02) = IDXP2 = INDEX OF FILE CRVSET-2
 (03) = IDXP3 = INDEX OF FILE CRVSET-3
 (04) = IDXP4 = INDEX OF FILE CRVSET-4

*** FILE CRVSET IS A SUBFILE OF PPHDTQ.  CRVSET CONTAINS BOTH
    HEAD AND TORQUE CHARACTERISTIC CURVES FOR ONE SET
 WORD
 (01) = IDXCRV(1) = INDEX OF HEAD CURVE TABLE - TYPE 1
 (02) = IDXCRV(2) =
 (03) = IDXCRV(3) =
 (04) = IDXCRV(4) =
 (05) = IDXCRV(5) =
 (06) = IDXCRV(6) =
 (07) = IDXCRV(7) =
 (08) = IDXCRV(8) = INDEX OF HEAD CURVE TABLE - TYPE 8
 (09) = IDXCRV(9) = INDEX OF TORQUE CURVE TABLE - TYPE 1
 (10) = IDXCRV(10)
 (11) = IDXCRV(11) =
 (12) = IDXCRV(12) =
 (13) = IDXCRV(13) =
 (14) = IDXCRV(14) =
 (15) = IDXCRV(15) =
 (16) = IDXCRV(16) = INDEX OF TORQUE CURVE TABLE - TYPE 8
*** PUMP HEAD AND TORQUE SUBFILES
 WORD
 (01) = NPCR   = NUMBER OF POINTS IN THE TABLE FOR THIS CURVE
 (02) = PHEAD(1) OR PTORK(1)
 (03) = PHEAD(2) OR PTORK(2)
 (04) = PHEAD(3) OR PTORK(3)
 (05) = PHEAD(4) OR PTORK(4)
   .
   .
   .
   .
   .
   .
*********************************************************************
```

8 292

```
**********************************************************************

*** FILID(20) = SLABHT - HEAT CONDUCTOR DESCRIPTIONS AND DATA
*** FILE SLABHT HAS NSLB DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 20 IS RESERVED IN SUBROUTINE INSLAB

 WORD
 (01) = IVSL    = INDEX NO. OF VOLUME AT LEFT SURFACE OF COND (NEW)
 (02) = IVSR    = INDEX NO. OF VOLUME AT RIGHT SURFACE OF COND(NEW)
 (03) = IGOM    = GEOMETRY INDEX
 (04) = ISB     = STACK INDICATOR
 (05) = IMCL    = LEFT SURFACE INDICATOR FOR HEAT TRAN. COR.
 (06) = IMCR    = RIGHT SURFACE INDICATOR FOR HEAT TRAN. COR.
 (07) = AHTL    = HEAT TRAN AREA AT LEFT CONDUCTOR SURFACE
 (08) = AHTR    = HEAT TRAN AREA AT RIGHT CONDUCTOR SURFACE
 (09) = VOLS    = VOLUME OF HEAT SLAB
 (10) = HDML    = HYDRAULIC DIAMETER OF VOLUME ON LEFT OF COND
 (11) = HDMR    = HYDRAULIC DIAMETER OF VOLUME ON RIGHT OF COND
 (12) = DHEL    = HEATED EQUIVALENT DIAMETER ON LEFT OF COND
 (13) = DHER    = HEATED EQUIVALENT DIAMETER ON RIGHT OF COND
 (14) = CHNL    = CHANNEL LENGTH ON LEFT OF COND
 (15) = CHNR    = CHANNEL LENGTH ON RIGHT OF COND
 (16) = SE      = STORED ENERGY IN CONDUCTOR
 (17) = IHXQF   = FLAG TO INDICATE USE OF HEAT CONDUCTOR AS A HEAT
                  EXCHANGER = 1
 (18) = FCHL    = CRITICAL HEAT FLUX AT LEFT COND SURFACE
 (19) = FCHR    = CRITICAL HEAT FLUX AT RIGHT COND SURFACE
 (20) = HTCL    = HEAT TRANSFER COEFFICIENT AT LEFT COND SURF
 (21) = HTCR    = HEAT TRANSFER COEFFICIENT AT RIGHT COND SURF
 (22) = PHIL    = HEAT FLUX AT LEFT COND SURFACE
 (23) = PHIR    = HEAT FLUX AT RIGHT COND SURFACE
 (24) = SLEN    = EQUIVALENT LENGTH OF HEAT COND
 (25) = WQCL    = HEAT TRANS RATE TO FLUID AT LEFT COND SURFACE
 (26) = WQCR    = HEAT TRANS RATE TO FLUID AT RIGHT COND SURFACE
 (27) = IBCL    = LEFT BOUNDARY CONDITION INDICATOR
 (28) = IBCR    = RIGHT BOUNDARY CONDITION INDICATOR
 (29) = IHTL    = HEAT TRANS MODE AT LEFT COND SURFACE
 (30) = IHTR    = HEAT TRANS MODE AT RIGHT COND SURFACE
 (31) = ISCO    = CORE NUMBER
 (32) = TL      = LEFT SINK TEMPERATURE, F
 (33) = TR      = RIGHT SINK TEMPERATURE, F
 (34) = AZL     = AZL THROUGH HZR ARE COEFFICIENTS FOR CHF
 (35) = AZ1L    = CORRELATIONS (LEFT SIDE)
 (36) = BZL     =   .
 (37) = CZL     =   .
 (38) = EZL     =   .
 (39) = EZ1L    =   .
 (40) = FZL     =   .
 (41) = HZL     =   .
 (42) = WED     = WETTED EQUIVALENT DIAMETER, IN
 (43) = IDXTP   = INDEX OF FILE CONTAINING TP FOR EACH NODE
 (44) = IDXAP   = INDEX OF FILE CONTAINING AP FOR EACH NODE
 (45) = IDXTPC  = INDEX OF FILE CONTAINING ITPC FOR EACH REGION
 (46) = IDXTPK  = INDEX OF FILE CONTAINING ITPK FOR EACH REGION
```

```
(47) = IDXTPX = INDEX OF FILE CONTAINING TPX FOR EACH REGION
(48) = OLSLBN = OLD CONDUCTOR NUMBER
(49) = IVSLOL = OLD VOLUME NUMBER (LEFT SIDE)
(50) = IVSROL = OLD VOLUME NUMBER (RIGHT SIDE)
(51) = AZR    = AZR THROUGH HZR COEFICIENTS FOR CHF CORRELATIONS
(52) = AZ1R   = FOR RIGHT SIDE
(53) = BZR    =   .
(54) = EZR    =   .
(55) = EZ1R   =   .
(56) = FZR    =   .
(57) = FZR    =   .
(58) = HZR    =   .
(59) =        =
(60) =        =
(61) = PHOL   = HEAT FLUX AT LEFT COND SURFACE (OLD TIME STEP)
(62) = PHOR   = HEAT FLUX AT RIGHT COND SURFACE (OLD TIME STEP)
(63) = GL     = LEFT COOLANT FLOW RATE
(64) = GR     = RIGHT COOLANT FLOW RATE
(65) = PHIRO  = OLD RIGHT SURFACE HEAT FLUX
(66) = DPHIR  = FIRST NUMERICAL DERIVATIVE OF RIGHT SURFACE
                HEAT FLUX
(67) = CREDL  = CAUSAL CONDUCTOR CREDIT (LEFT SIDE)
(68) = CREDR  = CAUSAL CONDUCTOR CREDIT (RIGHT SIDE
(69) = ICC    = CAUSAL CONDUCTOR COUNTER
(70) = PREQST = FLOW NETWORK NUMBER FOR SECONDARY SIDE OF THE
                STEAM GENERATOR ( >0 ONLY IF IHXQF>0 )
```

*** FILE SLBTP IS A SUBFILE OF SLABHT. SLBTP CONTAINS VARIABLE TP FOR EACH NODE IN A SLAB. INDEX OF FILE SLBTP IS WORD (43) OF FILID(20)

```
WORD
(01) = NNODE = NUMBER OF TEMPERATURE NODES IN THE HEAT SLAB
(02) = TP = NODE 1 TEMPERATURE
(03) = TP = NODE 2 TEMPERATURE
  .
  .
  .
(XX+1) = TP = NODE XX TEMPERATURE
(XX) = TP = NODE XX TEMPERATURE
```

*** FILE SLBAP IS A SUBFILE OF SLABHT. SLBAP CONTAINS VARIABLE AP FOR EACH NODE IN A HEAT CONDUCTOR. INDEX OF FILE SLBAP IS WORD (44) OF FILID(20)

```
WORD
(01) = AP = -2*SURFACE WEIGHT*THERMAL CONDUCTIVITY (FOR NODE 1)
(02) = AP = -2*SURFACE WEIGHT*THERMAL CONDUCTIVITY (FOR NODE 2)
  .
  .
  .
(XX) =AP = -2*SURFACE WEIGHT*THERMAL CONDUCTIVITY (FOR NODE XX)
```

8 294

*** FILE SLITPC IS A SUBFILE OF SLABHT. SLRAP CONTAINS VARIABLE
ITPC FOR EACH REGION OF A HEAT CONDUCTOR. INDEX OF FILE SLITPC
IS WORD(45) OF FILID(20)

WORD
(01) = ITPC = CURRENT POSITION IN TPC TABLE FOR REGION 1
(02) = ITPC = CURRENT POSITION IN TPC TABLE FOR REGION 2
.
.
.
(XX) = ITPC = CURRENT POSITION IN TPC TABLE FOR REGION XX

*** FILE SLITPK IS A SUBFILE OF SLABHT. SLITPK CONTAINS VARIABLE
ITPK FOR EACH REGION OF A HEAT CONDUCTOR. INDEX OF FILE SLITPK
IS WORD(46) OF FILID(20)

WORD
(01) = ITPK = CURRENT POSITION IN TPK TABLE FOR REGION 1
(02) = ITPK = CURRENT POSITION IN TPK TABLE FOR REGION 2
.
.
.
(XX) = = CURRENT POSITION IN TPK TABLE FOR REGION XX

*** FILE SLITPX IS A SUBFILE OF SLABHT. SLIPTX CONTAINS VARIABLE
ITPX FOR EACH REGION OF A HEAT CONDUCTOR. INDEX OF FILE SLITPX
IS WORD(47) OF FILID(20)

WORD
(01) = ITPX = CURRENT POSITION IN TPX TABLE FOR REGION 1
(02) = ITPX = CURRENT POSITION IN TPX TABLE FOR REGION 2
.
.
.
(XX) = ITPX = CURRENT POSITION IN TPX TABLE FOR REGION XX

****

8 295

```
**********************************************************************

*** FILID(21) = SLABGM - VARIABLES DESCRIBING CONDUCTOR GEOMETRY
*** FILE SLABGM HAS NGOM DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 21 IS RESERVED IN SUBROUTINE INGEOM

 WORD
 (01) = IDXR   = INDEX OF FILE CONTAINING GEOMETRY REGION VARIABLES
 (02) = IG     = GEOMETRY TYPE, 1-RECTANGULAR, 2-CYLINDRICAL
 (03) = NR     = NUMBER OF REGIONS
 (05) = ASUR   = RIGHT COND SURFACE AREA PER UNIT HEIGHT
 (04) = ASUL   = LEFT COND SURFACE AREA PER UNIT HEIGHT
 (06) = NSI    = NODE AT RIGHT SURFACE
 (07) = IDXS   = INDEX OF FILE CONTAINING SURFACE AREA WEIGHT FOR
                 THIS CONDUCTOR GEOMETRY
 (08) = IDXVP  = INDEX OF FILE CONTAINING NODAL VOLUMES FOR THIS
                 CONDUCTOR GEOMETRY


*** FILE GEOMR IS A SUBFILE FROM SLABGM.  GEOMR CONTAINS VARIABLES
    ASSOCIATED WITH EACH GEOMETRY REGION.  THE INDEX OF FILE GEOMR
    IS THE FIRST WORD OF FILID(21)
 WORD
 (01) = IKC  = MATERIAL INDEX
 (02) = POFR = FRACTION OF POWER IN REGION OF CORE HEAT COND
 (03) = NREG = VOLUME OF REGION PER FOOT OF LENGTH
 (04) = VRI  = VOLUME RATIO AT INTERFACE
 (05) = XDCR = COLD RADIUS OR DISTANCE TO RIGHT SURFACE OF REGION
 (06) = IGAP = GAP INDICATOR
 (07) = NI   = NODE AT INTERFACE

*** FILE NODEVP IS A SUBFILE FROM SLABGM.  NODEVP CONTAINS REGION
    VOLUMES FOR EACH COND.  THE INDEX FOR THIS FILE IS WORD (08)
    OF FILID(21)
 WORD
 (01) = VP = VOLUME OF REGION 1
 (02) = VP = VOLUME OF REGION 2
  .   =
  .   =
  .   =
 (XX) = VP = VOLUME OF THE LAST REGION XX


*** FILE NODES IS A SUBFILE OF SLABGM.  NODES CONTAINS SURFACE
    AREA WEIGHT, A/2DX, FOR ONE GEOMETRY.  THE INDEX FOR THIS
    FILE IS WORD(07) OF FILID(21)
 WORD
 (01) = S = SURFACE AREA WEIGHT OF REGION 1
 (02) = S = SURFACE AREA WEIGHT OF REGION 2
  .
  .
  .
 (XX) = S = SURFACE AREA WEIGHT OF THE LAST REGION XX
```

8 296

```
**********************************************************************

*** FILID(22) = SLABCM - CORE VOLUME AND METAL WATER REACTION DATA
*** FILE SLABCM HAS NCOR DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 22 IS RESERVED IN SUBROUTINE INCORE

 WORD
 (01) = ISLB    = OLD CONDUCTOR NUMBER
 (02) = CLTI    = INITIAL CLAD THICKNESS
 (03) = QFRAC   = FRACTION OF POWER IN CORE SECTION
 (04) = FQ      = POWER GENERATION IN CORE HEAT CONDUCTOR
 (05) = TM      = AVERAGE TEMPERATURE OF CORE HEAT CONDUCTOR
 (06) = TS      = SURFACE TEMPERATURE OF CORE HEAT CONDUCTOR
 (07) = CTR     = DEPTH OF MW REAC PENETRATION AT END OF TIME STEP
 (08) = CTRL    = DEPTH OF MW REAC PENETRATION AT STRT OF TIME STEP
 (09) = RRO     = ORIGINAL FUEL PIN RADIUS
 (10) = ISLBNW  = NEW CONDUCTOR NUMBER
 (11) = QMWR    = HEAT GENERATED BY M-W REACTION (BTU/HR)
 (12) = CTRIN   = INTERNAL DEPTH OF M-W REACTION
 (13) = MQ      = MODERATOR HEATING RATE  (BTU/HR)

**********************************************************************
```

8 297

```
**********************************************************************

*** FILID(23) = XNIFTE = SCRATCH
                SPACE FOR THE COEFFICIENT ARRAYS USED IN NIFTE
                (JJ IS THE NUMBER OF CONTROL VOLUMES, NVOL)
*** FILE 23 IS RESERVED IN SUBROUTINE INIFTE

 WORD
 (01)      = ALPHA(1,1)
   .
   .
 (JJ)      = ALPHA(JJ,1)
 (JJ+1)    = ALPHA(1,2)
   .
   .
 (2*JJ)    = ALPHA(JJ,2)
 (2*JJ+1)  = ALPHA(1,3)
   .
   .
 (3*JJ)    = ALPHA(JJ,3)
 (3*JJ+1)  = BETA(1,1)
   .
   .
 (6*JJ)    = BETA(JJ,3)
 (6*JJ+1)  = XGAMA(1,1)
   .
   .
 (9*JJ)    = XGAMA(JJ,3)
 (9*JJ+1)  = XLAMB(1,1)
   .
   .
 (12*JJ)   = XLAMB(JJ,3)
```

178 298

```
**********************************************************************

*** FILID(24) = THRCND - CONTAINS NMAT TABLES OF THERMAL
                 CONDUCTIVITY DATA
*** FILE 24 IS RESERVED IN SUBROUTINE INMPRO

 WORD
 (01) = IDXK1 = INDEX OF TABLE FOR MATERIAL 1
 (02) = IDXK2 = INDEX OF TABLE FOR MATERIAL 2
  .
  .
  .
 (IDXK1) = NKP = NUMBER OF POINTS IN TABLE 1
 (IDXK1+1)= TPK(1) = TEMPERATURE
 (IDXK1+2)= TPK(2) = THERMAL CONDUCTIVITY
  .
  .
  .
 (IDXK)    = NKP = NUMBER OF POINTS IN TABLE NMAT
 (IDXK+1) = TPK(1) = TEMPERATURE
 (IDXK+2) = TPK(2) = THERMAL CONDUCTIVITY
  .
  .
  .

**********************************************************************
```

8 299

```
**********************************************************************

*** FILID(25) = VHTCAP - CONTAINS NMAT TABLES OF VOLUMETRIC
                 HEAT CAPACITY DATA
*** FILE 25 IS RESERVED IN SUBROUTINE INMPRO

 WORD
 (01) = IDXC1 = INDEX OF TABLE FOR MATERIAL 1
 (02) = IDXC2 = INDEX OF TABLE FOR MATERIAL 2
  .
  .
  .
 (NMAT) =IDXC  = INDEX OF TABLE FOR MATERIAL NMAT
 (IDXC1)  = NCP = NUMBER OF POINTS IN TABLE 1
 (IDXC1+1)= TPC(1) = TEMPERATURE
 (IDXC1+2)= TPC(2) = VOLUMETRIC HEAT CAPACITY
  .
  .
  .
 (IDXC1+2*NCP-1)= TPC (2*NCP-1) = TEMPERATURE
 (IDXC1+2*NCP) = TPC(2*NCP)   = VOLUMETRIC HEAT CAPACITY
 (IDXC1+2*NCP+1) = TPCI(1) = INTEGRAL OF TCP
  .
  .
  .
 (IDXC1+3*NCP) = TPCI(NCP) - INTEGRAL OF TCP
  .
  .
  .
 (IDXC)  = NCP   = NUMBER OF POINTS IN TABLE NMAT
 (IDXC+1)= TPC(1) = TEMPERATURE
 (IDXC+2)= TPC(2) = VOLUMETRIC HEAT CAPACITY
  .
  .
  .
 (IDXC+2*NCP-1)= TPC(2*NCP-1)= TEMPERATURE
 (IDXC+2*NCP)  = TPC(2*NCP)  = VOLUMETRIC HEAT CAPACITY
 (IDXC+2*NCP+1)= TPCI(1) = INTEGRAL OF TCP
  .
  .
  .
 (IDXC+3*NCP) = TPCI(NCP) = INTEGRAL OF TPC

**********************************************************************
```

8 300

```
**********************************************************************

*** FILID(26) = EXPCOF - CONTAINS NMAT TABLES OF THERMAL
                         EXPANSION COEFICIENTS
*** FILE 26 IS RESERVED IN SUBROUTINE INMPRO

 WORD
 (01) = IDXX1 = INDEX OF TABLE FOR MATERIAL 1
 (02) = IDXX2 = INDEX OF TABLE FOR MATERIAL 2
  .
  .
  .
 (NMAT)= IDXX  = INDEX OF TABLE FOR MATERIAL NMAT
 (IDXX1)  = NXP = NUMBER OF POINTS IN TABLE 1
 (IDXX1+1)= TPX(1) = TEMPERATURE
 (IDXX1+2)= TPX(2) = LINEAR EXPANSION COEFICIENT
  .
  .
  .
(IDXX) = NXP  = NUMBER OF POINTS IN TABLE NMAT
 (IDXX+1)= TPX(1)= TEMPERATURE
 (IDXX+2)= TPX(2)= LINEAR EXPANSION COEFICIENT
  .
  .
  .

**********************************************************************
```

8 301

```
**********************************************************************

*** FILID(27) = CONKIN - CONTAINS KINETICS CONSTANTS DATA
*** FILE CONKIN IS A RESERVE FILE WITH STARTING INDEX IDXKIN
*** FILE 27 IS RESERVED IN SUBROUTINE INRKEN

 WORD
 (01) = KODEL = KINETICS CALCULATION OPTION FLAG
 (02) = KMUL  = MULTIPLYING FACTOR FOR DECAY ENERGY RELEASE RATES
 (03) = BOVL  = BETA / MEAN NEUTRON LIFETIME
 (04) = RHOIN = INITIAL REACTIVIY
 (05) = UDUF  = U-238 ATOMS CONSUMED PER U-235 ATOMS FISSIONED
 (06) = PROMPT = FRAC OF FISSION POWER RELEASED AT TIME OF FISSION
 (07) = LAMBDA = FRACTION OF SURFACE HEAT PRODUCING SUBCOOLED
                 VAPOR BUBBLES
 (08) = TAU   = VAPOR BUBBLE LIFETIME
 (09) = DT    = IREKIN TIME INTERVAL
 (10) = W1    = INVERSE PERIOD AT PREVIOUS TIME STEP
 (11) =        =
 (12) = PEROID = REACTOR PERIOD
 (13) = PREAC = TOTAL REACTIVITY
 (14) = SLOPE = RATE OF CHANGE OF REACTIVITY
 (15) = SS    = TIME DEPENDENT SOURCE
 (16) = SUM   = DELAYED NEUTRON SOURCE
 (17) = T     = IREKIN TIME STEP SIZE
 (18) = W     = AVERAGE RECIPROCAL PERIOD
 (19) = W3    = INSTANTANEOUS RECIPROCAL PERIOD
 (20) = KK    = RUNGA KUTTA STAGE
 (21) = PHI1  = OLD EMITTER CONCENTRATION
   .
   .
   .
   .
 (41) = PHI1 = OLD EMITTER CONCENTRATION
 (42) = PHI2 = NEW EMITTER CONCENTRATION
   .
   .
 (62) = PHI2 = NEW EMITTER CONCENTRATION
 (63) = PPOW = PROMPT POWER
 (64) = DPOW = DELAYED POWER
 (65) = EGAM(1) = YIELD FRACTIONS OF GAMMA NEUTRON EMITTERS
  .
  .
  .
 (75) = EGAM(11)= YIELD FRACTIONS OF GAMMA NEUTRON EMITTERS
 (76) = EGAM(12)= EQUIVALENT YIELD FRACTIONS FOR U-239 + NP-239
 (77) = EGAM(13)= EQUIVALENT YIELD FRACTIONS FOR U-239 + NP-239
 (78) = EGAM(14)= EQUIVALENT YIELD FRACTION FOR FISSION POWER
 (79) = DLAMDA  = DECAY CONSTANTS FOR SIX DELAYED NEUTRON GROUPS
 (85) = AJOVRA  = YIELD FRACTIONS FOR SIX DELAYED NEUTRON GROUPS

**********************************************************************
```

```
**********************************************************************

*** FILID(28) = SCRAMT - CONTAINS REACTIVITY TABLE QUANTITIES
*** FILE 28 IS RESERVED IN SUBROUTINE INSCRM

 WORD
 (01) = RC      = CONTROL REACTIVITY
 (02) = RCCAL   = CALCULATED CONTROL REACTIVITY AT THE START
 (03) = NSC     = NUMBER OF SCRAM CURVES
 (04) = NSCR    = NUMBER OF SCRAM DATA POINTS FOR TABLE 1
 (05) = ITSCRM = TRIP NUMBER FOR SCRAM 1
 (06) = ISCR    = CURRENT TABLE 1 POSITION
 (07) = IDXTBL = INDEX OF FILE CONTAINING TABLE 1
 (08) = NSCR    = NUMBER OF SCRAM DATA POINTS FOR TABLE 2
 (09) = ITSCRM = TRIP NUMBER FOR SCRAM 2
 (10) = ISCR    = CURRENT TABLE 2 POSITION
 (11) = IDXTBL = INDEX OF FILE CONTAINING TABLE 2
  .
  .
  .
 (XX) = NSCR    = NUMBER OF SCRAM DATA POINTS FOR TABLE NSC
 (XY) = ITSCRM = TRIP NUMBER FOR SCRAM NSC
 (YY) = ISCR    = CURRENT POSITION IN TABLE NSC
 (YX) = IDXTBL = INDEX OF FILE CONTAINING TABLE NSC

*** FILID REATBL IS A SET OF SUBFILES OF SCRAMT, EACH FILE
    CONTAINS A REACTIVITY TABLE
 WORD
 (01) = TSCR = TIME
 (02) = TSCR = REACTIVITY OR NORMALIZED POWER
  .
  .
  .
 (2*NSCR-1) = TSCR = TIME
 (2*NSCR)   = TSCR = REACTIVITY OR NORMALIZED POWER

**********************************************************************
```

8 303

```
**********************************************************************

*** FILID(29) = FEDBAK - CONTAINS DENSITY REACTIVITY FEEDBACK DATA
*** FILE 29 IS RESERVED IN SUBROUTINE INREAC

 WORD
 (01) = RHOCAL = INPUT REACTIVITY
 (02) = NVOID  = NUMBER OF POINTS IN VOIDRO TABLE
 (03) = IDXDNT = INDEX OF FILE CONTAINING VOIDRO TABLE
 (04) = RW     = COOLANT REACTIVITY
 (05) = RWCAL  = CALCULATED COOLANT REACTIVITY AT START
 (06) = RV     = VOID REACTIVITY
 (07) = RVCAL  = CALCULATED VOID REACTIVITY AT START
 (08) = NDOP   = NUMBER OF POINTS IN DOPRO TABLE
 (09) = IDXDPT = INDEX OF FILE CONTAINING DOPRO TABLE
 (10) = RF     = FUEL REACTIVITY
 (11) = RFCAL  = CALCULATED FUEL REACTIVITY AT START
 (12) = RD     = DOPPLER REACTIVITY
 (13) = RDCAL  = CALCULATED DOPPLER REACTIVITY AT START
 (14) = IDXMHT = INDEX OF MODERATOR HEATING WEIGHTING FACTOR
                 SUBFILE
 (15) = NMODHT = NUMBER OF MODERATOR HEATING WEIGHTING FACTOR
                 DATA POINTS

*** FILID DENTBL IS A SUBFILE OF FEDBAK, CONTAINS DENSITY
    REACTIVITY TABLE
 (01)      = VOIDRO(1)    = DENSITY NORMALIZED TO BEGINNING VALUE
 (02)      = VOIDRO(1)    = REACTIVITY
  .
  .
  .
 (2*NDEN-1)= VOIDRO(NDEN)= DENSITY NORMALIZED TO BEGINNING VALUE
 (2*NDEN)  = VOIDRO(NDEN)= REACTIVITY

*** FILID DOPTBL IS A SUBFILE OF FEDBAK, CONTAINS DOPPLER TABLE
 WORD
 (01)      = DOPRO(1)     = TEMPERATURE
 (02)      = DOPRO(1)     = REACTIVITY
  .
  .
  .
 (2*NDOP-1)= DOPRO(NDOP) = TEMPERATURE
 (2*NDOP)  = DOPRO(NDOP) = REACTIVITY

*** FILID AMODHT IS A SUBFILE OF FEDBAK, CONTAINS MODERATOR
    HEATING WEIGHTING FACTOR TABLE.

 WORD
 (01)          = WEIGHT(1) = FLUID DENSITY
 (02)          = WEIGHT(1) = WEIGHTING FACTOR
   .
   .
   .
 (2*NMODHT-1) = WEIGHT(NMODHT) = FLUID DENSITY
```

8 504

(2*NMODHT) = WEIGHT(NMODHT) = WEIGHTING FACTOR

****

5

8 30

```
**********************************************************************

*** FILID(30) - CONTAINS CONDENSING HEAT TRANSFER DATA
*** FILE 30 IS RESERVED IN SUBROUTINE INCOHT

 WORD
 (01) = IDXCHT = INDEX OF THE DATA SET CONTAINING CONDENSING
                 HEAT TRANSFER DATA
 (01) = MODE   = CONDENSING HEAT TRANSFER CORRELATION TYPE
 (02) = UCHMUL = UCHIDA CORRELATION MULTIPLIER
 (02) = HCONST = USER SUPPLIED CONSTANT HEAT TRANSFER COEFFICIENT
 (03) = IUCHID = TABLE POSITION INDICATOR
 (04) = IBLO   = TRIP ID USED TO INITIATE BLOWDOWN
 (05) = BLOTIM = ESTIMATED LENGTH OF BLOWDOWN PHASE
 (06) = HMAX   = MAX. PRE-BLOWDOWN HEAT TRANSFER COEFFICIENT
 (07) = HMIN   = MIN. PRE-BLOWDOWN HEAT TRANSFER COEFFICIENT
 (08) = TRNCON = TRANSITION DELAY CONSTANT

**********************************************************************
```

```
**********************************************************************

*** FILID(31) = REACT - CONTAINS REACTIVITY COEFICIENT DATA
*** FILE REACT HAS NCOR DATA SETS.  A DATA SET IS SHOWN BELOW
*** FILE 31 IS RESERVED IN SUBROUTINE INREAC

 WORD
 (01) = VOIDWT = DENSITY WEIGHTING FACTOR
 (02) = DOPWT  = FUEL TEMPERATURE WEIGHTING FACTOR
 (03) = ALPHTM = FUEL TEMPERATURE COEFICIENT
 (04) = ALPHTW = WATER TEMPERATURE COEFICIENT
 (05) = V1     = INITIAL  COOLANT SPECIFIC VOLUME
 (06) = IDOP   = CURRENT POSITION IN DOPRO TABLE
 (07) = IVOID  = CURRENT POSITION IN VOIDRO TABLE
 (08) = VW     = SUBCOOLED VAPOR VOLUME FRACTION
 (09) = QPMOD  = CORE SECTION MODERATOR PROMPT POWER FRACTION
 (10) = QDMOD  = CORE SECTION MODERATOR DELAYED POWER FRACTION
 (11) = IMHT   = MEMORY INDEX FOR MODERATOR HEATING WEIGHTING
                 FACTOR TABLE

**********************************************************************
```

178 307

```
**********************************************************************

*** FILID(32) = ALIQLV - CONTAINS LIQUID LEVEL CALCULATION
                 VARIABLES
*** FILE 32 IS RESERVED IN SUBROUTINE INLVC

 WORD
 (01) = ZLVC    = EQUIVALENT LIQUID LEVEL
 (02) = ITLVC   = CURRENT TABLE POSITION
 (03) = NTLVC   = NUMBER OF PAIRS IN VZLVC
 (04) = IDXLVC  = INDEX OF FILE CONTAINING IVLC
 (05) = IDXTLC  = INDEX OF FILE CONTAINING VZLVC
 (06) = LVCVOL  = 1ST VOLUME IN SUM,FOR EDIT ONLY


*** FILE AILVC IS A SUBFILE OF ALIQLV, CONTAINS IVLC
 WORD
 (01) = ILVC = VOLUMES TO BE SUMMED (NEW)
 (02) = ILVC =
  .
  .
  .
 (NLVC)= ILVC =

*** FILE AVZLVC IS A SUBFILE OF ALIQLV, CONTAINS VZLVC
 WORD
 (01)       = VZLVC = HEIGHT(1)
 (02)       = VZLVC = VOLUME(1)
  .
  .
  .
 (2*NTLVC-1)= VZLVC = HEIGHT(NTLVC)
 (2*NTLVC)  = VZLVC = VOLUME(NTLVC)

**********************************************************************
```

178 308

```
*********************************************************************

*** FILID(33) = TVOLUM - TIME DEPENDENT VOLUME QUANTITIES
*** FILE 33 IS RESERVED IN SUBROUTINE INTV

 WORD
 (01) = TIMTBL(1) = TIME
 (02) = PTABL(1)  = PRESSURE
 (03) = TTABL(1)  = TEMPERATURE
 (04) = XTABL(1)  = QUALITY OF MIXTURE
 (05) = ZTABL(1)  = MIXTURE LEVEL
 (NTDV) = IIR = CURRENT TABLE INDEX OF LAST TIME DEPENDENT VOLUME
 (NTDV+1) = IDXTVT = INDEX OF FILE CONTAINING T. D. VOLUME 1 TABLE
   .
   .
   .
 (2*NTDV) = IDXTVT = INDEX OF FILE CONTAINING LAST T. D. V. TABLE

*** FILE TDVTBL IS A SUBFILE OF TVOLUM, CONTAINS TIME DEPENDENT
    VOLUME TABLE FOR VOLUME XX. THERE ARE NTDV TDVTBL FILES
 WORD
 (01) = IRIN = NUMBER OF DATA POINTS FOR THIS SET
 (02) = TIMTBL(1) = TIME
 (03) = PTABL(1)  = PRESSURE
 (04) = TTABL(1) = TEMPERATURE
 (05) = XTABL(1) = QUALITY OF MIXTURE
 (06) = ZTABL(1) = MIXTURE LEVEL
   .
   .
   .
 (5*IRIN-3) = TIMTBL(IRIN)
 (5*IRIN-2) = PTABL(IRIN)
 (5*IRIN-1) = TTABL(IRIN)
 (5*IRIN  ) = XTABL(IRIN)
 (5*IRIN+1) = ZTABL(IRIN)

*********************************************************************
```

1258 509

```
*****************************************************************

*** FILID(34) = SLBSTK - CONTAINS SLAB STACK INDICATORS
*** FILE 34 IS RESERVED IN SUBROUTINE INSLAB

 WORD
 (01) = ISHD(1) = NUMBER OF DIMENSIONS FOR HEAT TRANSFER .STACK 1)
 (02) = ISSB(1) = INDEX OF SLAB AT BOTTOM OF AXIAL STACK (STACK 1)
 (03) = ISST(1) = INDEX OF SLAB AT TOP    OF AXIAL STACK (STACK 1)
 (04) = ISHD(2) = NUMBER OF DIMENSIONS FOR HEAT TRANSFER (STACK 2)
 (05) = ISSB(2) = INDEX OF SLAB AT BOTTOM OF AXIAL STACK (STACK 2)
 (06) = ISST(2) = INDEX OF SLAB AT TOP    OF AXIAL STACK (STACK 2)
  .
  .
  .
 (XX) = ISHD(XZ/3) -
 (XY) = ISSB(XZ/3) -
 (XZ) = ISST(XZ/3) -

*****************************************************************
```

8 10

```
**********************************************************************

*** FILID(35) = TSPCON - CONTAINS TIME STEP CONSTANT FACTORS
*** FILE 35 IS RESERVED IN SUBROUTINE TSTMOD

 WORD
 (01) = FC3 = FC3 ... FC7 = TIME STEP CONSTANT FACTORS
 (02) = FC4 =
 (03) = FC5 =
 (04) = FC6 =
 (05) = FC7 =
 (06) = FC8 =
 (07) = FC9 =
 (08) = DTNEXT = DECREMENTED TIME USED TO GIVE EVEN TIME NTTS BOUNDARIES
 (09) = IDMP   = NUMBER OF COMPLETE EDITS SINCE LAST TIMING EDIT
 (10) = IMAJ   = NUMBER OF BRIEF EDITS SINCE LAST COMPLETE EDIT
 (11) = IMIN   = NUMBER OF TIME STEP SINCE LAST BRIEF EDIT
 (12) = IPLT   = NUMBER OF TIME STEPS SINCE LAST PLOT SAVE
 (13) = NSET   = CURRENT INTERVAL INDEX
 (14) = NSTEP  = CURRENT TIME STEP NUMBER
 (15) = NTSET  = NUMBER OF TIME STEP SETS
 (16) = NTTSP  = TOTAL NUMBER OF TIME STEPS USED
 (17) = ENDSTP = .TRUE. INDICATES LAST TIME STEP INTERVAL REDUCED
 (18) = REDUC  = .TRUE. INDICATES PREVIOUS TIME STEP REDUCED
 (19) = IDXITS = INDEX OF FILE CONTAINING ITSPL
 (20) = IDXKNT = INDEX OF FILE CONTAINING KNTDT
 (21) = DTOLDS = PREVIOUS TIME STEP UPON RETURN TO TSTP,CURRENT
                 DT UNTIL RETURN
 (22) = OLDHTC = OLD HEAT TRANSFER TIME CREDIT
 (23) = HTMIN  = HEAT TRANSFER TIME CONSTANT

 (24) = HTT    = HEAT TRANSFER TIME STEP
 (25) = HTTO   = OLD HEAT TRANSFER TIME STEP
 (26) = EXPCVG = EXPLICIT ITERATION CONVERGENCE FLAG LOGICAL
 (27) = IMPCVG = IMPLICIT ITERATION CONVERGENCE FLAG LOGICAL
 (28) = HTFLG  = HEAT TRANSFER CALCULATION FLAG    LOGICAL
 (29) = DTMAX  = MAXIMUM ALLOWABLE TIME STEP SIZE FOR A GIVEN TIME POINT


*** FILE FITSPL IS A SUBFILE OF TSPCON, INDEX OF FILE FITSPL IS
    WORD (19) OF FILID(35)
 WORD
 (01)  = ITSPL = COUNT FOR VOLUME OR JUNCTION CONTROLLING TIME STP
  .
  .
  .
 (NMAX) = ITSPL = (NMAX = MAX0(NJUN,NVOL))


*** FILE FKNTDT IS A SUBFILE OF TSPCON, INDEX OF FILE FKNTDT IS
    WORD(20) OF FILID(35)
 WORD
 (01) = KNTDT = NUMBER OF TIME STEP SELECTION CRITERIA
  .
```

8 311

.
.
(10) = KNTDT =

****

8 312

```
**********************************************************************

*** FILID(37) = HEAT EXCHANGER QUANTITIES
*** FILE 37 IS RESERVED IN SUBROUTINE INHTXQ

 WORD
 (01) = INDHTX = INDEX FOR TIME-HEAT REMOVAL RATE TABLE
 (02) = IHTX   = HEAT EXCHANGER TYPE  0= FLOW AND TEMPERATURE
 (03) = ITHTXQ = TRIP NUMBER CONTROLLING HEAT EXCHANGER
 (04) = JVOL   = VOLUME NUMBER (NEW)
 (05) = THQ    = HEAT EXCHANGER PRIMARY SIDE TEMPERATURE
 (06) = HTQ    = HEAT REMOVAL RATE
 (07) = TSEC   = HEAT EXCHANGE SECONDARY SIDE TEMPERATURE
 (08) = HTXCO  = HEAT EXCHANGER HEAT TRANSFER COEFFICIENT
                 DEPENDENT  N = NUMBER OF TIME-POWER POINTS
 (09) = IHQ    = LAST POSITION IN HTXTBL
 (10) = IHTYPE = FLAG FOR TYPE OF NON-CONDUCTING HEAT EXCHANGER
 (11) = QHTR   = PRESSURIZER HEATER POWER
 (12) = QTAU   = PRESSURIZER HEATER DECAY CONSTANT
 (13) = QPREV  = PRESSURIZER HEATER POWER (PREVIOUS TIME STEP)
 (14) = QUAP   = PEAK POWER BEFORE PRESSURIZER HEATER TRIPS OFF
 (15) = TIMOFF = TIME PRESSURIZER HEATER TRIPS OFF
 (16) = HTXCOI = INITIAL HEAT TRANSFER COEFICIENT
 (17) = POWERF = INITIAL FRACTION OF POWER TO BE REMOVED
 (18) = IHEQST = FLOW NETWORK NUMBER TO WHICH THE HEAT EXCHANGER
                 BELONGS

*** FILID HTXTBL IS A SUBFILE OF HEATEX,CONTAINS TIME-HEAT REMOVAL
*** RATE
 (01)      = HTXTBL = TIME
 (02)      = HTXTBL = NORMALIZED POWER
  .
  .
  .
 (IHTX-1) = HTXTBL = TIME
 (IHTX)   = HTXTBL = NORMALIZED POWER

**********************************************************************
```

178 313

```
**********************************************************************

*** FILID(38) - WORK - SCRATCH WORK AREA FOR TRANSIENT LINK
                  THE FILSIZ & SETSIZ FOR FILID 38 ARE SET TO
                  THE TOTAL LENGTH OF SCRATCH SPACE AVAILABLE
                  BEGINING AT WORKA(1)
*** FILE 38 IS RESERVED IN SUBROUTINE INIFTE

 WORD
 (01) = IDXWKA= INDEX OF FILE CONTAINING WORKA OR ALL
                AVAILABLE SCRATCH SPACE
                ALSO THE INDEX OF THE C ARRAY USED IN NIFTE
 (02) = IDXG  = INDEX OF FILE CONTAINING G ARRAY USED IN NIFTE
 (03) = IDXBA = INDEX OF THE BA ARRAY USED IN NIFTE
 (04) = IDXXAL= INDEX OF FILE CONTAINING XALPH ARRAY USED IN NIFTE
 (05) = IDXGAM= INDEX OF FILE CONTAINING XGAM ARRAY USED IN NIFTE
 (06) = IDXOLW= INDEX OF FILE CONTAINING OLDWP ARRAY USED IN GAUSS


*** FILE WORKA IS A SUBFILE OF WORK. WORKA PROVIDES
    A UNIVERSAL WORK ARRAY UTILZED BY SEVERAL ROUTINES
    IN RETRAN.  THE INDEX FOR THIS FILE IS WORD(01) FROM
    FILID(38).
    (XX IS NJUN*NJUN AND YY IS NJUN)
 WORD
 (01) = C(1) OR WORKA(1)
 (02) = C(2) OR WORKA(2)
  .
  .
 (XX) = C(XX) OR WORKA(XX)
*** FILE G IS A SUBFILE OF WORK. G PROVIDES STORAGE UTILIZED
    IN NIFTE. THE INDEX OF THIS FILE IS WORD 2 FROM FILE 38.
 WORD
 (01) = G(1) OR WORKA(XX+1)
 (02) = G(2)
  .
  .
 (YY) = G(YY) OR WORKA(XX+YY)
*** FILE BA IS A SUBFILE OF WORK. BA PROVIDES SCRATCH
    STORAGE UTILIZED IN NIFTE.  THE INDEX FOR THIS FILE IS
    WORD 3 FROM FILE 38.
 WORD
 (01) = BA(1) OR WORKA(XX+YY+1)
 (02) = BA(2)
  .
  .
 (YY) = BA(YY) OR WORKA(XX+2*YY)
*** FILE XALPH IS A SUBFILE OF WORK. XALPH PROVIDES STORAGE
    UTILIZED IN NIFTE. THE INDEX OF THIS FILE IS WORD 4 FROM
    FILE 38.
 WORD
 (01) = XALPH(1) OR WORKA(XX+2*YY+1)
 (02) = XALPH(2)
  .
```

```
  .
 (YY) = XALPH(YY) OR WORKA(XX+3*YY)
*** FILE XGAM IS A SUBFILE OF WORK. XGAM PROVIDES STORAGE
    UTILIZED IN NIFTE. THE INDEX OF THIS FILE IS WORD 5 FROM
    FILE 38.
 WORD
 (01) = XGAM(1) OR WORKA(XX+3*YY+1)
 (02) = XGAM(2)
  .
  .
 (YY) = XGAM(YY) OR WORKA(XX+4*YY)
*** FILE OLDWP IS A SUBFILE OF WORK. OLDWP PROVIDES STORAGE USED IN
    GAUSS. THE INDEX OF THIS FILE IS WORD 6 FROM FILE 38
 WORD
 (01) = OLDWP(1) OR WORKA(XX+4*YY+1)
 (02) = OLDWP(2)
  .
  .
  .
 (YY) = OLDWP(YY) OR WORKA(XX+5*YY)

   NOTE *****  WORKA MAY BE LONGER THAN XX+4*YY

*****************************************************************
```

8 315

```
**********************************************************************

*** FILID(39) = TRIPID - CONTAINS VARIABLES ASSOCIATED WITH EACH
                         IDTRP.  FILE TRIPID HAS A DATA SET FOR
                         EACH IDTRP.  A DATA SET IS SHOWN BELOW
*** FILE 39 IS RESERVED IN SUBROUTINE INTRIP

 WORD
 (01) = OLDTRP = OLD TRIP ID
 (02) = TRPT   = TIME TO REACH SETPOINT + DELAY TIME
 (03) = FLAG   = TRUE MEANS SETPOINT REACHED AND DELAY TIME HAS
                 EXPIRED
 (04) =          UNUSED AT PRESENT TIME
 (05) = TRPT   = TIME TO REACH SETPOINT + DELAY TIME FOR RESET
                 TRIP
 (06) = FLAG   = TRUE MEANS SETPOINT REACHED AND DELAY TIME HAS
                 EXPIRED FOR RESET TRIP

**********************************************************************
```

178 316

```
**********************************************************************

*** FILID(40) = RANGER - CONTAINS VARIABLES FOR JUNCTION ORDERING
*** FILE 40 IS RESERVED IN SUBROUTINE INIFTE

 WORD
 (01) = MPP      = NUMBER OF FIRST CRITICAL JUNCTION
 (02) = MS       = NUMBER OF CHAINS
 (03) = MCTR     = LENGTH OF JUNCTION CONNECTION ARRAY IWF
 (04) = NQ       = LENGTH OF NON-CHAIN PORTION OF CONNECTION MATRIX
 (05) = NTRI     = NUMBER OF CHAIN JUNCTIONS
 (06) = NX       = NUMBER OF NON-CRITICAL JUNCTIONS
 (07) = IDXIWF   = INDEX FOR FILE CONTAINING IWF ARRAY
 (08) = IDXNP    = INDEX FOR FILE CONTAINING CHAIN LENGTH ARRAY
 (09) = IXNPSI   = INDEX OF FILE CONTAINING NPSI ARRAY
 (10) = IDXNUM   = INDEX OF FILE CONTAINING NUMAX ARRAY.

*** FILE FIWF IS A SUBFILE FROM RANGER. FIWF CONTAINS
    JUNCTION CONNECTION ARRAY IWF . THE INDEX FOR THIS
    FILE IS WORD (07) OF FILID(40)
 WORD
 (01) = IWF = POSITION OF FIRST NON-ZERO ELEMENT
 (02) = IWF = POSITION OF SECOND NON-ZERO ELEMENT
  .
  .
 (XX) = IWF = POSITION OF LAST NON-ZERO ELEMENT


*** FILE FND IS A SUBFILE OF RANGER. FNP CONTAINS ARRAY
    NP. THE INDEX FOR THIS FILE IS WORD(08) OF FILID(40)
 WORD
 (01) = NP = LENGTH OF FIRST CHAIN
 (02) = NP = LENGTH OF SECOND CHAIN
  .
  .
 (XX) = NP = LENGTH OF FINAL CHAIN


*** FILE FNPSI IS A SUBFILE FROM RANGER. FNPSI CONTAINS
    ORDER OF JUNCTION SOLUTION. THE INDEX FOR THIS FILE
    IS WORD(09) OF FILID(40)
 WORD
 (01) = FIRST JUNCTION SET
 (02) = SECONDJUNCTION SET

  .
  .
 (XX) = NPSI = LAST JUNCTION SET


*** FILE FNUMAX IS A SUBFILE OF RANGER. FNUMAX CONTAINS
    THE NUMAX ARRAY. THE INDEX FOR THIS FILE IS WORD(10)
    FROM FILID(40)
 WORD
```

(01) = NUMAX = NUMBER OF NONZERO ELEMENTS IN FIRST ROW OF MATRIX
(02) = NUMAX = NUMBER OF NONZERO ELEMENTS IN SECOND ROW OF MATRIX
•
•
(XX) = NUMAX = NUMBER OF NONZERO ELEMENTS IN LAST ROW OF MATRIX

****

```
**********************************************************************

*** FILID(41) - CONTAINS INDEXES OF FILES

 WORD
 (01) = IDXQN = INDEX OF FILE CONTAINING VARIABLE QN
 (02) = IDXTK = INDEX OF FILE CONTAINING VARIABLE TK
 (03) = IDXCR = INDEX OF FILE CONTAINING VARIABLE CR

*** FILE SLBCR IS A SUBFILE OF FILID(41). SLBCR CONTAINS VARIABLE
    CR FOR EACH NODE IN A HEAT CONDUCTOR.
    SUBFILES OF FILID(41)
 WORD
 (01) = CR = VOLUMETRIC HEAT CAPACITY AT NODE 1
 (02) = CR = VOLUMETRIC HEAT CAPACITY AT NODE 2
  .
  .
  .
 (XX) = CR = VOLUMETRIC HEAT CAPACITY AT NODE XX


*** FILE SLBTK IS A SUBFILE OF FILID(41). SLBTK CONTAINS VARIABLE
    TK FOR EACH NODE IN A HEAT CONDUCTOR.
    SUBFILES OF FILID(41)
 WORD
 (01) = TK = THERMAL CONDUCTIVITY AT NODE 1
 (02) = TK = THERMAL CONDUCTIVITY AT NODE 2
  .
  .
  .
 (XX) = TK = THERMALL CONDUCTIVITY AT NODE XX

**********************************************************************
```

8 319

```
**********************************************************************

*** FILID(42) = TIME DEPENDENT DATA RETREIVED FROM TAPE
*** FILE 42 IS RESERVED IN SUBROUTINE INTV

 WORD
 (01) = NDREC  = CURRENT DATA RECORD NUMBER
 (02) = NVOLO  = NUMBER OF VOLUMES ON RETREIVED TAPE
 (03) = NVOLS  = SETSIZ OF RETREIVED VOLUME DATA RECORD
 (04) = NJUNO  = NUMBER OF JUNCTIONS ON RETREIVED TAPE
 (05) = NJUNS  = SETSIZ OF RETREIVED JUNCTION DATA RECORD
 (06) = TIMNEW = TIME OF CURRENT RETREIVED RECORD
 (07) = TIMOLD = TIME OF PREVIOUSLY RETREIVED RECORD
 (08) = POWNEW = NORMALIZED POWER OF CURRENT RETREIVED RECORD
 (09) = POWOLD = NORMALIZED POWER OF PREVIOUSLY RETREIVED RECORD
 (10) = IDXVTB = INDEX OF TIME DEPENDENT VOLUME DATA SUBFILE
 (11) = IDXVSN = INDEX OF SUBFILE CONTAINING VSN AND CREATION DATE
 (12) = NTVOL  = NUMBER OF TAPE VOLUMES WITH DATA TO RETREIVE
 (13) = NTVOLX = COUNTER ON CURRENT LOCATION IN VSN SUBFILE
 (14) = LABEL1 = LABEL OF TAPE FROM WHICH TO RETREIVE DATA.
 (15) = LABEL2 = TWO WORDS
 (16) = IDXVLC = INDEX OF VOLUME TO TIME DEPENDENT VOLUME DATA SET
                 CORRESPONDENCE SUBFILE
 (17) = IVLOL  = OFFSET USED TO PICK VOLUME NUMBER FROM ABREVIATE
                 VOLUME FILE
 (18) = IPRES  = OFFSET USED TO PICK PRESSURE FROM AN ABREVIATED
                 VOLUME SET
 (19) = ITEMP  = OFFSET USED TO PICK TEMP. FROM AN ABREVIATED
                 VOLUME SET
 (20) = IQUAL  = OFFSET USED TO PICK QUALITY FROM AN ABREVIATED
                 VOLUME SET
 (21) = IZMIX  = OFFSET USED TO PICK MIXTURE LEVEL FROM AN
                 ABREVIATED VOLUME SET
 (22) = ITIMX  = OFFSET USED TO PICK TIMEX FROM ABREVIATED SYSTEM
                 FILE
 (23) = IPOWN  = OFFSET USED TO PICK PNORM FROM ABREVIATED SYSTEM
                 FILE

*** SUBFILE CONTAINING VOLUME DATA FOR N VOLUMES
*** (THERE ARE MTDV SETS OF DATA)

 WORD
 (01) = IVOLN  = VOLUME NUMBER OF VOLUME DATA RETREIVED
 (02) = PNEW   = CURRENT PRESSURE
 (03) = POLD   = PREVIOUS PRESSURE
 (04) = TNEW   = CURRENT TEMPERATURE
 (05) = TOLD   = PREVIOUS TEMPERATURE
 (06) = XNEW   = CURRENT QUALITY
 (07) = XOLD   = PREVIOUS QUALITY
 (08) = ZNEW   = CURRENT MIXTURE LEVEL
 (09) = ZOLD   = PREVIOUS MIXTURE LEVEL

*** SUBFILE CONTAINING TAPE VSN AND CREATION DAT
```

8 320

```
 WORD
 (01) = IVSN    = VOLUME SERIAL NUMBER OF TAPE
 (02) = IDATE   = JULEAN CREATION DATE OF TAPE
                  FOR NTVOL PAIRS


*** SUBFILE CONTAINING INDECES OF SETS IN THE VOLUME FILE.FOR VOLUMES
    USING THE CORRESPONDING TIME DEPENDENT VOLUME DATA SET FROM THE ABOVE
    SUBFILE
 WORD
 (01) = IDXVST(1) = INDEX OF VOLUME DATA SET USING THE FIRST SET
                    OF TIME DEPENDENT VOLUME DATA
 (02) = IDXVST(2) = INDEX OF VOLUME DATA SET USING THE SECOND SET
                    OF TIME DEPENDENT VOLUME DATA
  .
  .
  .
  .                      .
 (N ) = IDXVST(MTDV)          .

**********************************************************************

        .
```

8 321

```
**********************************************************************

*** FILID(43) - PLTREC - DATA RECORD DESCRIPTIONS

 WORD
 (01) = NOFILS = NUMBER OF ABREVIATED FILES DUMPED
 (02) = IDXADD = INDEX OF FILE CONTAINING ADDRESSES
 (03) = NSET     NUMBER OF VARIABLES REQUIRED TO DESCRIBE EACH
                 FILE INCLUDED IN A DATA RECORD. THE VARIABLES
                 ARE DEFINED AS FOLLOW.
 (04) = NFIL   = RELOCATABLE FILE ID. IF A SUBFILE, THE VALUE
                 IS MINUS ONE TIMES THE MAJOR FILE ID POINTING
                 IS MINUS ONE TIMES THE MAJOR FILE ID. INDEX IN
                 MAJOR FILE POINTS TO SUBFILE AS DISCUSSED BELOW.
 (05) = LFIL   = ABREVIATED FILE SET SIZE
 (06) = NSET   = NUMBER OF SETS IN FILE
 07) = MFIL    = FLAG FOR FILE WITH ASSOCIATED SUBFILE TO BE
                 WRITTEN TO DATA TAPE.
                    = 0 MAJOR FILE WITH NO SUBFILE TO BE WRITTEN
                    > 0 IF MAJ FILE, NUMBER OF SUBFILES TO BE
                        WRITTEN
                 IF A SUBFILE (DETERMINED BY SIGN OF NFIL), THE
                 BOTTOM HALF OF THE WORD CONTAINS THE OFFSET FROM
                 THE BEGINING OF THE ABBREVIATED MAJOR FILE TO
                 THE INDEX POINTING TO THE SUBFILE. THE TOP
                 HALF WORD CONTAINS THE OFFSET TO THE INDEX FOR
                 THE FULLY DESCRIBED MAJOR FILE.
  .
  .
  .
  .            = FOR NOFILS QUADRUPLETS
  .
  .
  .
*** ADDRESS IS A SUBFILE OF FILE PLTREC AND CONSISTS OF NOFIL
    GROUPS OF DATA. A TYPICAL GROUP IS DESCRIBED BELOW.
 WORD
 (01) = LFIL(1) = THE NUMBER OF EDIT ADDRESSES FOR FILE ID NFIL(1)
 (02) =         = ADDRESS OF 1ST EDIT VARIABLE FROM FILE NFIL(1)
  .
  .
  .
  .
  .
  .
 (LFIL(1)+1) =  = ADDRESS OF LAST EDIT
  .
  .
  .
  .
  .
  . AND SO ON FOR NOFIL GROUPS (IF A FILE SPECIFIED IN NFIL ARRAY
  . DOES NOT EXIST FOR A GIVEN PROBLEM,THE GROUP CONSISTS OF
  . ONE WORD = 0 )
```

8 322

```
**********************************************************************

*** FILID(44) = PLOTD - PLOTTING INFORMATION

 WORD
 (01) = NDSET  = NUMBER OF DATA SETS FOR PLOTTING
 (02) = FRAMES = NUMBER OF PLOT FRAMES REQUESTED
 (03) = NPLOTC = NUMBER OF PLOT CURVES
 (04) = NPLOTD = NUMBER OF PLOT CURVES WITH COMBINATIONS
 (05) = NPEDIT = EDIT FLAG - LE.0 NO EDIT - GT.0 EDIT DATA PLOTTED
 (06) = IDXNDS = INDEX OF SUBFILE FOR DATA SET DESCRIPTIONS
 (07) = IDXAXS = INDEX OF SUBFILE FOR PLOT AXIS DESCRIPTIONS
 (08) = IDXPLC = INDEX OF SUBFILE FOR PLOT CURVES
 (09) = IDXPLD = INDEX OF SUBFILE FOR PLOT CURVES WITH COMBINATION
 (10) = IDSET  = NUMBER OF CURRENT DATA SET FROM WHICH DATA IS
                 BEING RETREIVED
 (11) = FTBID  = FTB FILE ID OF FIRST PLOT DATASET
*** FILE DATSET IS A SUBFILE CONTAINING DATA SET INFORMATION
***
 WORD
 (01) = NVSN   = NUMBER OF VSN'S FOR THIS DATA SET
 (02) = NPSIZE = SETSIZE OF FILE
 (03) = NPSETS = NUMBER OF SETS IN FILE
 (04) = LABL1  = ALPHANUMERIC LABEL FOR TAPE, UP TO 17 CHARACTERS
 (05) = LABL2  = IN TWO WORDS
 (06) = IDSDEN = RECORDING DENSITY OF DATA SET
 (07) = IVSN   = VOLUME SERIAL NUMBER OF FIRST VOLUME OF SET
 (08) = ICREAT = CREATION DATA OF FIRST TAPE VOLUME OF SET
 .
 .
 (2*NVSN+2) = IVSN = VOLUME SERIAL NUMBER OF LAST VOLUME OF SET
 (*NVSN+3) = ICREAT = CREATION DATE OF LAST TAPE VOLUME OF SET
*** FILE AXIS IS A SUBFILE CONTAINING PLOT AXIS INFORMATION
***
*** X AXIS DATA
***
 WORD
 (01) = XVAR   = INDEPENDENT VARIABLE
 (02) = XREG   = DEPENDENT VARIABLE REQUEST FLAG
 (03) = XLINOG = LINEAR OR LOGARITHMIC SCALE ON X AXIS,DEFAULT LIN
 (04) = XLENG  = LENGTH OF X AXIS
 (05) = XMIN   = MINIMUM X
 (06) = XMAX   = MAXIMUM X
 (07) = XLABL1 = X AXIS LABEL   4A8
 (08) = XLABL2 =      .
 (09) = XLABL3 =      .
 (10) = XLABL4 =      .
 (11) = NAXIS  = NUMBER OF Y-AXES FOR FRAME
***
*** Y AXIS DATA   (INDEX = IDXAXS + 11*NFRAMES)
***
 WORD
 (01) = YLINOG = LINEAR OR LOGARITHMIC SCALE ON Y AXIS
 (02) = YLENG  = LENGTH OF Y AXIS
```

```
 (03) = YMIN   = MINIMUM Y
 (04) = YMAX   = MAXIMUM Y
 (05) = YLAB1  = Y AXIS LABEL  4A8
 (06) = YLAB2  =    .
 (07) = YLAB3  =    .
 (08) = YLAB4  =    .
 (09) = KFRAME = FRAME NUMBER FOR Y-AXIS
 (10) = NCURVS = NUMBER OF CURVES TO USE Y AXIS SCALING
*** FILE PLOTC IS A SUBFILE CONTAINING PLOT CURVE INFORMATION
***
 WORD
 (01) = YVARC  = DEPENDENT VARIABLE
 (02) = IYREGC = DEPENDENT VARIABLE REGION
 (03) = IDSETC = DATA SET NUMBER
 (04) = IDXYAX = INDEX OF Y-AXIS SUBFILE
 (05) = ICARDC = CARD NUMBER, USED FOR COMBINATION PLOTS
 (06) = YSCTRN = DEPENDENT VARIABLE TRANSLATION SCALING FACTOR
 (07) = YSCMAG = TRUE IF MAGNIFICATION,  FALSE IF TRANSLATION
 (08) = XSCTRN = INDEPENDENT VARIABLE TRANSLATION SCALING FACTOR
 (09) = XSCMAG = TRUE IF MAGNIFICATION,  FALSE IF TRANSLATION
*** FILE PLOTD IS A SUBFILE CONTAINING PLOT CURVE WITH COMBINATION
*** INFORMATION
***
 WORD
 (01) = ICPLOT = FRAME NUMBER FOR THIS COMBINATION PLOT (XXY)
 (02) = NOP    = NUMBER OF OPERATIONS FOR COMBINATION CURVE
 (03) = IDCOMB = INDEX OF COMBINATION SUBFILE
*** FILE PLOTCS IS A SUBFILE CONTAINING INDICES OF PLOT CURVES
***DEFINED IN PLOTC, TO BE COMBINED.
 WORD
 (01) = IDXCRV = INDEX OF PLOTC SUBFILE
 (02) = IOP    = COMBINATION CURVE OPERATION ( + - * / )
 (03) = IDXCRV
 (04) = IOP                  AND SO ON FOR NOP OPERATIONS
 (05) = IDXCRV
   .
   .
   .

*****************************************************************
```

8 324

```
**********************************************************************

*** FILID(47) - CONTAINS PROBLEM DIMENSIONS FOR DNB CALCULATIONS,
                HOT CHANNEL FACTORS, LINEAR HEAT GENERATION RATES, DNBR
                VALUES, AND THE INFORMATION OBTAINED IN CORE CALCULATIONS.
*** FILE 47 IS RESERVED IN SUBROUTINE INDNB

 WORD
 (01) = LWR     = REACTOR TYPE
 (02) = ICW     = FLAG FOR COLD WALL
 (03) = NUH     = FLAG FOR NON-UNIFORM HEAT FLUX
 (04) = ICHF    = FLAG FOR CHF CORRELATION
 (05) = ISPGR   = GRID SPACER INDICATOR INDICATOR
 (06) = NOA     = NUMBER OF AXIAL POWER PROFILE DATA SETS
 (07) = N1      = NUMBER OF CORE VOLUMES
 (08) = FQENG   = ENG. HEAT FLUX FACTOR
 (09) = FRN     = NUC. RADIAL HEAT FLUX FACTOR
 (10) = FQUNC   = HEAT FLUX UNCERTAINTY FACTOR
 (11) = FDHCOR  = ADJUSTMENT FACTOR ON HOT BUNDLE MASS FLUX
 (12) = FDELH   = CHANNEL ENTHALPY ADJUSTMENT FACTOR
 (13) = ZMIN    = MIN. HEIGHT OF FUEL FOR DNB CALC.(FT)
 (14) = ZMAX    = MAX. HEIGHT OF FUEL FOR DNB CALC.(FT)
 (15) = PITCH   = ROD-ROD PITCH(FT)
 (16) = RODIA   = ROD DIAMETER(FT)
 (17) = EQDIA   = EQUIVALENT DIA. BASED ON WETTED PERIMETER(FT)
 (18) = TDC     = THERMAL DIFFUSIVITY PARAMETER
 (19) = FRPOW   = FRACTION OF POWER GENERATED IN FUEL ROD
 (20) = RGRID   = R GRID FACTOR
 (21) = HEATL   = HEATED CHANNEL LENGTH
 (22) = NDELT   = NO. OF DNB CALC. BETWEEN RETRAN TIME STEP
 (23) = HIDIA   = HYDRAULIC DIA BASED ON HEATED PERIMETER
 (24) = JLEVEL  = NODAL POINT NUMBERED FROM BOTTOM OF CORE
 (25) = COTHT   = TOTAL CORE HEIGHT
 (26) = TOTA    = TOTAL HEAT TRANSFER AREA
 (27) = QBAR    = AVERAGE HEAT FLUX IN CORE
 (28) = QTZBAR  = AVERAGE HEAT FLUX IN CORE AT TIME ZERO
 (29) = HGGRCO  = LINEAR HEAT GENERATION RATE IN CORE
 (30) = DNBRCO  = DNB RATIO OF AVERAGE CORE
 (31) = HGRMNZ  = LINEAR HEAT GENERATION RATE AT ZMIN
 (32) = DNBRMZ  = DNB RATIO AT ZMIN
 (33) = HGRHOT  = LINEAR HEAT GENERATION RATE AT HOT SPOT
 (34) = DNBHRT  = DNB RATIO AT HOT SPOT
 (35) = ZMDNB   = HEIGHT OF MIN DNBR
 (36) = HGRMD   = LINEAR HEAT GENERATION RATE AT ZMDNB
 (37) = DNBRM   = MIN DNBR VALUE
 (38) = TIMDNB  = TIME OF DNB CALCULATION
 (39) = ZHOT    = HEIGHT OF HOT SPOT
 (40) = CORHP1  = CORE INLET ENTHALPY
 (41) = CORHP2  = CORE OUTLET ENTHALPY
 (42) = CORFL1  = CORE INLET FLOW RATE
 (43) = CORFL2  = CORE OUTLET FLOW RATE
 (44) = NEG     = NO. OF NODAL POINTS FOR ENERGY CALC. IN CORE
 (45) = NDNB    = NO. OF NODAL POINTS FOR DNB CALC.
 (46) = IDXPRO  = SUBFILE INDEX FOR POWER PROFILE TABLE
```

(47) = IDXNOD = SUBFILE INDEX FOR DNB CALC.
(48) = IDXDRA = SUBFILE INDEX FOR DNBR

****

8 326

```
**********************************************************************

*** FILID(48) - CONTAINS THE INFORMATION PERTAINING TO THE AVERAGE CORE
                VOLUMES FOR DNB MODEL
*** FILE 48 IS RESERVED IN SUBROUTINE INDNB

 WORD
 (01) = COREPR = CORE INLET VOLUME PRESSURE
 (02) = CORESP = CORE INLET SPECIFIC VOLUME
 (03) = HTA    = HEAT TRANSFER AREA FOR A CONDUCTOR IN CORE
 (04) = COHTF  = HEAT FLUX FOR CONDUCTOR IN CORE
 (05) = COVDHT = INLET CORE VOLUME HEIGHT
 (06) = COREX  = INLET CORE VOLUME QUALITY
 (07) = SOL    = COEFFICIENTS FOR HEAT FLUX AS FUNC. OF POSITION
 (08) = CONODE = NODAL POSITION IN CORE VOLUMES
 (09) = QCRIT  = DNB FLUX FOR AVG. CORE
 (10) = COTEMP = CORE FLUID TEMPERATURE
 (11) = NVLDNB = CORE VOLUME NUMBER

**********************************************************************
```

8 327

```
**********************************************************************

*** FILID(49) - CONTAINS THE INFORMATION PERTAINING TO THE ENERGY
                CALCULATIONS FOR THE DNB MODEL
*** FILE 49 IS RESERVED IN SUBROUTINE INDNB

 WORD
 (01) = XQUAL   = QUALITY AT NODAL POINT FOR ENERGY CALC.
 (02) = ZENG    = NODAL POINT HEIGHT FOR ENERGY CALC.
 (03) = PRSRE   = PRESSURE AT NODAL POINT FOR ENERGY CALC.
 (04) = GFLO    = MASS FLUX AT NODAL POINT FOR ENERGY CALC.
 (05) = QQENG   = HEAT FLUX AT NODAL POINT FOR ENERGY CALC.
 (06) = RHO     = DENSITY AT NODAL POINT FOR ENERGY CALC.
 (07) = HENG    = ENTHALPY AT NODAL POINT FOR ENERGY CALC.
 (08) = GDNB    = LOCAL MASS FLUX CORRECTED FOR ENERGY CALC.
 (09) = XTEN    = QUALITY AT NODAL POINT
 (10) = TTEN    = TEMPERATURE AT NODAL POINT
 (11) = SPA     = SPECIFIC VOLUME AT NODAL POINT
 (12) = FZN     = AXIAL HOT CHANNEL FACTOR
 (13) = HF      = ENTHALPY OF SATURATED LIQUID
 (14) = HFG     = LATENT HEAT
 (15) = QTZFZ   = HEAT FLUX AS FUNCTION OF HEIGHT AT TIME ZERO

**********************************************************************
```

```
**********************************************************************

*** FILID(50) - CONTAINS THE INFORMATION PERTAINING TO DNB CALCULATIONS
*** FILE 50 IS RESERVED IN SUBROUTINE INDNB

 WORD
 (01) = QQ      = HEAT FLUX FOR DNB CALC.
 (02) = PRS     = LOCAL PRESSURE
 (03) = DENS    = DENSITY OF FLUID AT LOCAL POINT
 (04) = XQ      = QUALITY AT LOCAL POINT
 (05) = GT      = MASS FLUX AT LOCAL POINT
 (06) = HDNB    = ENTHALPY AT LOCAL POINT
 (07) = HFDNB   = ENTHALPY OF SATURATED LIQUID AT LOCAL POINT
 (08) = ZDNB    = NODAL POINT FOR DNB CALC.
 (09) = HFGDNB  = LATENT HEAT FOR DNB CALC.
 (10) = BB      = HEAT FLUX CALCULATED FOR NON-UNIFORM FACTOR CALC.
 (11) = XM      = HEAT FLUX / LENGTH   FOR NON-UNIFORM FACIOR CALC.

**********************************************************************
```

8 529

****

*** FILID(51) - NON EQUILIBRIUM VOLUME PRESSURIZER QUANTITIES.
*** FILE 51 IS RESERVED IN SUBROUTINE INVOL

WORD
(01) = NELM = NON-EQ. LIQUID REGION TOTAL MASS
(02) = NELU = NON-EQ. LIQUID REGION SPECIFIC ENERGY
(03) = NELE = NON-EQ. LIQUID REGION TOTAL ENERGY
(04) = NELH = NON-EQ. LIQUID REGION AVERAGE ENTHALPY
(05) = NELLM = NON-EQ. LIQUID REGION LIQUID MASS
(06) = NELVM = NON-EQ. LIQUID REGION VAPOR MASS
(07) = NELX = NON-EQ. LIQUID REGION QUALITY
(08) = NELLV = NON-EQ. LIQUID REGION LIQUID SPECIFIC VOLUME
(09) = NELVV = NON-EQ. LIQUID REGION VAPOR SPECIFIC VOLUME
(10) = NELV = NON-EQ. LIQUID REGION SPECIFIC VOLUME
(11) = NELLH = NON-EQ. LIQUID REGION SATURATED LIQUID ENTHALPY
(12) = NELVH = NON-EQ. LIQUID REGION SATURATED VAPOR ENTHALPY
(13) = NELVOL = NON-EQ. LIQUID REGION VOLUME
(14) = VIDXL = NON-EQ. LIQUID REGION STATE PROP. MEMORY INDEX
(15) = SIDXL = NON-EQ. LIQUID REGION SAT. STATE PROP. MEMORY IND
(16) = NELT = NON-EQ. LIQUID REGION TEMPREATURE
(17) = NELP = NON-EQ. LIQUID REGION PRESSURE
(18) = NEVM = NON-EQ. VAPOR REGION MASS
(19) = NEVU = NON-EQ. VAPOR REGION SPECIFIC ENERGY
(20) = NEVE = NON-EQ. VAPOR REGION TOTAL ENERGY
(21) = NEVH = NON-EQ. VAPOR REGION AVERAGE ENTHALPY
(22) = NEVLM = NON-EQ. VAPOR REGION LIQUID MASS
(23) = NEVVM = NON-EQ. VAPOR REGION VAPOR MASS
(24) = NEVX = NON-EQ. VAPOR REGION QUALITY
(25) = NEVLV = NON-EQ. VAPOR REGION LIQUID SPECIFIC VOLUME
(26) = NEVVV = NON-EQ. VAPOR REGION VAPOR SPECIFIC VOLUME
(27) = NEVV = NON-EQ. VAPOR REGION SPECIFIC VOLUME
(28) = NEVVOL = NON-EQ. VAPOR REGION VOLUME
(29) = VIDXV = NON-EQ. VAPOR REGION STATE PROP. MEMORY INDEX
(30) = SIDXV = NON-EQ. VAPOR REGION SAT. STATE PROP.MEMORY INDEX
(31) = NEVT = NON-EQ. VAPOR REGION TEMPERATURE
(32) = NEVP = NON-EQ. VAPOR REGION PRESSURE
(33) = IXP = FLAG FOR SPECIAL TREATMENT OF SPRAY
(34) = IPHSEL = PHASE UNDEX FOR LIQUID REGION
(35) = IPHSEV = PHASE INDEX FOR VAPOR REGION
(36) = IPHSE = PHASE INDEX AFTER SYSTEM HAS GONE CRITICAL
(37) = NELV2 = SPECIFIC VOLUME LIQUID REGION TWO TIME STEPS BACK
(38) = NELV1 = SPECIFIC VOLUME OF LIQUID REGION FROM PREVIOUS TIME STEP
(39) = CONM = MASS RAINED FROM VAPOR REGION TO LIQUID
(40) = FLAM = LIQUID MASS FLASHED FROM LIQUID TO VAPOR REGION
(41) = CONE = ENERGY ADDED TO LIQUID REG. FROM MASS RAINED OUT
(42) = FLAE = ENERGY ADDED TO VAPOR REGION FROM FLASHING
(43) = NELMO = NON-EQ. LIQUID REGION MASS (PREVIOUS TIME STEP)
(44) = NELEO = NON-EQ. LIQUID REGION TOTAL ENERGY (PREVIOUS TIME STEP)
(45) = NEVMO = NON-EQ. VAPOR REGION MASS (PREVIOUS TIME STEP)
(46) = NEVEO = NON-EQ. VAPOR REGION TOTAL ENERGY

8 330

(PREVIOUS TIME STEP)
(47) = VOLOLD = OLD LIQUID REGION VOLUME
(48) = NEVOLD = OLD VAPOR REGION VOLUME

****

8 531

```
**********************************************************************

*** FILID(52) - PIPE TRANSPORT QUANTITIES
*** FILE 52 IS RESERVED IN SUBROUTINE INVOL

 WORD
 (01) = WPTO2  = SUM OF THE FLOWS OUT OF THE VOLUME
 (02) = WPTO1  = SUM OF THE FLOWS OUT OF THE VOLUME
                 (PREVIOUS TIME STEP)
 (03) = WPTI2  = SUM OF THE FLOWS INTO VOLUME
 (04) = WPTI1  = SUM OF THE FLOWS INTO THE VOLUME
                 (PREVIOUS TIME STEP)
 (05) = MIN    = INTEGRATED MASS INTO THE VOLUME OVER TIME STEP
 (06) = EIN    = INTEGRATED ENERGY INTO THE VOLUME OVER TIME STEP
 (07) = MOUT   = INTEGRATED MASS OUT OF THE VOLUME OVER TIME STEP
 (08) = EOUT   = INTEGRATED ENERGY OUT OF VOLUME OVER TIME STEP
 (09) = ACEIN  = ACCUMULATED ENERGY IN THE VOLUME
 (10) = ACMIN  = ACCUMULATED MASS IN THE VOLUME
 (11) = ACEOUT = ACCUMULATED ENERGY OUT OF THE VOLUME
 (12) = ACMOUT = ACCUMULATED MASS OUT OF THE VOLUME
 (13) = ACMINO = ACCUMULATED MASS IN THE VOLUME (OLD TIME STEP)
 (14) = ACEINO = ACCUMULATED ENERGY IN TEH VOLUME (OLD TIME STEP)
 (15) = ACMOTO = ACCUMULATED MASS OUT OF THE VOLUME
                 (OLD TIME STEP)
 (16) = ACEOTO = ACCUMULATED ENERGY OUT OF THE VOLUME
                 (OLD TIME STEP)
 (17) = IDXMSH = INDEX OF THE SUBFILE CONTAING MESH ENTHALPIES
 (18) = MESH   = NUMBER OF MESH POINTS IN THE VOLUME
 (19) = ITVOLD = OLD VOLUME NUMBER FOR MINOR EDIT REGION CHECKS

**********************************************************************
```

```
**********************************************************************

*** FILID(53) = CONTROL - CONTAINS PARAMETERS DESCRIBING EACH
                CONTROL BLOCK.  EACH SET IS IN ORDER OF INCREASING VALL
                OF IDC (CONTROL BLOCK ID).
*** FILE CONTRCL CONTAINS (IDMAX-IDMIN+1) SETS
*** FILE 53 IS RESERVED IN SUBROUTINE INCNT1

 WORD
 (01) = ITYPE= BLOCK TYPE
 (02) = INC1 = INTEGER CONTROL BLOCK PARAMETER 1, USUALLY USED TO
               SPECIFY IDC OF THE CONTROL BLOCK CONNECTED TO THE INPUT.
 (03) = INC2 = INTEGER CONTROL BLOCK PARAMETER 2
 (04) = CP1  = FLOATING POINT CONTROL BLOCK PARAMETER 1
 (05) = CP2  = FLOATING POINT CONTROL BLOCK PARAMETER 2
 (06) = CGAIN= CONTROL BLOCK GAIN
 (07) = COUT = CONTROL BLOCK OUTPUT
 (08) = CMIN = MINIMUM ALLOWED VALUE OF CONTROL BLOCK OUTPUT
 (09) = CMAX = MAXIMUM ALLOWED VALUE OF CONTROL BLOCK OUTPUT
 (10) = IDXEDT = MINOR EDIT REGION CHECK FOR CONTROL SYSTEM

**********************************************************************
```

8 333

```
**********************************************************************

*** FILID(54) = CSYSTEM - OVERALL CONTROL SYSTEM QUANTITIES
*** FILE 54 IS RESERVED IN SUBROUTINE INCNT1

 WORD
 (01) = NCI    = NO. OF CONTROL SYSTEM INPUT CARDS
 (02) = NCB    = NO. OF CONTROL BLOCK DESCRIPTION CARDS
 (03) = IDMIN  = A NEGATIVE NUMBER EQUAL TO MINIMUM VALUE OF IDC
 (04) = IDMAX  = A POSITIVE NUMBER EQUAL TO MAXIMUM VALUE OF IDC
 (05) = NCSTOR= NUMBER OF WORDS OF AUXILIARY STORAGE IN ARRAY
                 NAMED BOX
 (06) = INIT   = INITIALIZATION FLAG
 (07) = NTS    = NUMBER OF ACTUAL TIME STEPS
 (08) = ETIME  = ELAPSED PROBLEM TIME
 (09) = DT     = NEW TIME STEPS
*** THE REST OF THIS FILE CONSISTS OF TWO ARRAYS IDC AND BOX.
    IDC IS A CARD ORDERED ARRAY WHICH ALSO DETERMINES THE ORDER OF
    CALCULATION USED BY SUBROUTINE CONTRL
    BOX IS AN AUXILLIARY STORAGE ARRAY USED TO STORE THE STATE OF
    CONTROL BLOCKS WHICH HAVE MEMORY
 WORD
 (10)              = IDC(1)
 (11)              = IDC(2)
  .                    .
  .                    .
  .                    .
 (IDMAX-IDMIN+10)  = IDC(IDMAX-IDMIN+1)
 (IDMAX-IDMIN+11)  = BOX(1)
  .                    .
  .                    .
  .                    .
 (IDMAX-IDMIN+10+NCSTOR) = BOX(NCSTOR)

**********************************************************************
```

8 334

```
**********************************************************************

*** FILID(56) - CONTAINS ENERGY BALANCE INFORMATION DESCRIBING STEAM
                GENERATORS OR FEEDWATER SYSTEMS FOR STEADY STATE
                INITIALIZATION.  THIS FILE IS RELEASED AFTER STEADY STA
*** FILE 56 IS RESERVED IN SUBROUTINE INSTGN

 WORD
 (01) = NSTG    = NUMBER OF STEAM GENERATORS
 (01) = ISGNUM  = STEAM GENERATOR NUMBER
 (02) = JBIAS   = JUNCTION NUMBER FOR ENTHALPY BIAS
 (03) = JBAL    = OUTLET JUNCTION NUMBER FOR POWER REMOVAL SYSTEM
 (04) = POWF    = POWER FRACTION TO BE REMOVED BY THIS STEAM GEN.
 (05) = HTXSUM  = TOTAL POWER REMOVED BY THE STEAM GENERATOR
 (06) = ISEQST  = FLOW NETWORK NUMBER THIS STEAM GENERATOR IS TO
                  REMOVE POWER FROM

**********************************************************************
```

8 535

```
**********************************************************************

*** FILID(57) - CONTAINS TOTAL THERMAL POWER FOR EACH FLOW
                NETWORK

*** FILE 57 IS RESERVED IN SUBROUTINE INTILZ

 WORD
 (01) = NEQST  = NUMBER OF FLOW NETWORKS
 (02) = PTHRM  = NETWORK 1 THERMAL POWER
 (03) = PTHRMO = NETWORK 1 PREVIOUS TIME STEP THERMAL POWER
   .     .
   .     .
   .     .
        PTHRM  = NETWORK NEQST THERMAL POWER
        PTHRMO = NETWORK NEQST PREVIOUS TIME STEP THERMAL POWER
```

178 336

Appendix B

UPD PROGRAM DESCRIPTION

UPD is a program for editing and/or updating card files on an IBM computer. This Appendix provides instructions on the use of UPD along with sample input deck listings.

TABLE OF CONTENTS

8 339

1.0 INTRODUCTION

UPD is a program for editing and/or updating card files. Source decks for programs are the most common application for UPD, but its use is not limited to source cards. UPD can process any 80 column card image provided there is no $ character in Column 1 (could be modified if necessary) and that Columns 73-80 are reserved for a name and sequence field. This document describes the control cards available and their function. Sample input deck listings illustrate the use of the program.

2.0 CONTROL CARDS

The UPD program operates on a sequence of card images. Operations may include dividing the cards into decks, deleting certain ones, adding others, and replacement. These operations are under the control of input cards to the UPD program. Such control cards are identified with a $ or * character in Column 1. Cards with any other characters in Column 1 are assumed to be source cards. Cards having a * character in Column 1 are treated as control cards only if the next three characters are one of the permissible control identifiers.

UPD operates in a deck mode, a deck being a series of cards with identical character strings in Columns 73-76. Cards are ordered within decks by a sequence number in Columns 77-80. For programs with many subroutines, it is natural to use the subroutine name, or a derivative, in Columns 73-76. At the end of a run, UPD produces an index of the decks contained on the old master tape, the new master tape, and the auxiliary (temporary) file. The auxiliary file will normally be used as input to a compiler or an assembler and erased.

Source cards may either be input from the normal input channel or obtained from an old master program file. In either case, a new master program file may be created which will include the current control requests.

Two types of decks may reside on a master file. The first kind is the normal source deck for a subroutine, for example. These decks are accessed sequentially and control cards or replacement cards must be organized in the same sequence as the master file. The second type of deck is a COMDECK which is a group of cards which is (usually) common to multiple decks. These decks must reside at the beginning of the master program file and are accessed randomly during a UPD run. COMDECK's are identified by name and the contents of the deck inserted in the auxiliary file replacing the request control card. The request card is copied onto the new master program file.

2.1 $LOCATE

$LOCATE: Format (19A4, I4)

178 341

Examples: (numbers are column numbers)

```
           1   1   2
1      9   3   7   0
$LOCATE DECK
$LOCATE DECK PCH DEK
```

$LOCATE causes the old master file to be positioned at a deck boundary, completing a partially processed deck if necessary. The old master is then advanced and copied until the deck specified in Columns 9-12 is found. The located deck is marked partially processed and the auxiliary file writer turned on if a non-blank appears anywhere in Columns 13-16.

A second deck name in Columns 17-20 has the effect of copying all decks between and including the two deck names to the auxiliary file.

2.2 $REMOVE

$REMOVE: Format (19A4, I4)

Examples:

```
           1   1   2
1      9   3   7   0
$REMOVE DCKQ
$REMOVE DCKQDCKT
```

$REMOVE causes the old master file to be positioned at a deck boundary, completing a partially processed deck if necessary. The old master is then advanced and copied until the deck specified in Columns 9-12 is found. The old master is then advanced without copying until the deck specified in Columns 13-16 is passed. If the second deckfield is blank, only the deck specified in Columns 9-12 is removed. No removed deck can be put on the auxiliary file.

2.3 $NUMBER

$NUMBER: Format (3A4, 2I4, 14A4, I4)

78 542

Examples:

```
           1   1   2
1      9   3   7   0
$NUMBER DECK0050
$NUMBER DECK00500002
```

$NUMBER turns the sequencing feature on. Cards are given the identifier listed in Columns 9-12 and starting sequence number in Columns 13-16. In the examples, the next card placed on the new master file (and auxiliary file) will be numbered (Columns 73-80) DECK0050. Subsequent cards will have the sequence number (77-80) increased by the increment specified in Columns 17-20 of the $NUMBER card. A blank or zero increment will default to a value of 0010. The renumbering feature is turned off when processing a new deck is begun.

NOTE: A deck can be renamed by renumbering.

2.4 $INSERT

$INSERT: Format (19A4, I4)

Examples:

```
           1   1   2
1      9   3   7   0
$INSERT DECK
$INSERT  PUN
```

$INSERT causes the old master file to be positioned at a deck boundary, completing a partially processed deck if necessary. Cards are read from the input stream and placed on the new master file (and the auxiliary file if a non-blank appears anywhere in Columns 13-16). If a deck name has been put in Columns 9-12 of the $INSERT card, the insert operation will terminate when a non-blank deck name different from that on the $INSERT card is encountered. If Columns 9-12 of the $INSERT card have been left blank the insert operation will process all source cards until a control card other than a $NUMBER card is found.

2.5 $ENDRUN

$ENDRUN: Format (19A4, I4)

Examples:

```
           1   1   2
1      9   3   7   0
$ENDRUN DECK P
$ENDRUN
$
```

$ENDRUN (or a single $ character) causes the old master file to be positioned at a deck boundary, completing a partially processed deck if necessary. The old master is then advanced and copied until the deck specified in Columns 9-12 is found. The auxiliary writing control is then turned on if a non-blank appears anywhere in Columns 13-16, the specified deck is copied, and the update run is terminated. Blanks in Columns 9-12 will result in copying the remainder of the old master. An end-of-input condition results in a $ card being generated.

2.6 $DELETE

$DELETE: Format (3A4, 2I4, 14A4, I4)

Examples:

```
           1   1   2
1      9   3   7   0
$DELETE DECK00650250
$DELETE DECK0065
```

If Columns 9-12 are different from the current deck name, the effect is to issue a $LOCATE DECK XXX control card before finding the cards to be deleted.

$DELETE copies the old master file until the first sequence number (Columns 13-16) is reached. Cards are then deleted (inclusively) until the old master file is beyond the second sequence number (Columns 17-20). A blank or zero second sequence number will cause the specified card to be the only one deleted.

2.7 $CHANGE

$CHANGE: Format (3A4, I4, 16A4)

Examples:

```
           1   1
1      9   3   7
$CHANGE DECK0150 'ABC'CBA
$CHANGE DECK0220 /25/
```

$CHANGE will locate the card stipulated by DECK (Columns 9-13) and the card sequence number (Columns 13-16). On the $CHANGE card, the first non-blank character after Column 17 will be designated a delimiter. The program searches the input card for the second occurrence of the delimiter, counting the intervening characters. The program then searches the designated source card for the same sequence of characters found on the $CHANGE card between the first and second delimiters. When this is found, it is replaced with an identical number of characters which follow the second delimiter on the $CHANGE card. In the first example given, card number 0150 of the routine DECK would be located and the first occurrence of the characters ABC would be replaced by CBA. In this example single quotes are used as delimiters. In the second example slashes are the delimiters and card number 0220 will have the number 25 replaced by blanks. This is a convenient way to remove statement numbers.

2.8 $MODIFY

$MODIFY: Format (3A4, I4, 16A4)

Examples:

```
           1   1
1      9   3   7
$MODIFY DECK0990 'I'L
```

The $MODIFY card is similar to the $CHANGE card except that instead of only the first occurrence of the character sequence being replaced, every occurrence on the source card of the character sequence between the two delimiters will be replaced by the sequence following the second delimiter. Thus, in the example above, I in the selected source card will be replaced by an L whereas $CHANGE 'I'L would cause only the first I encountered (left-most) to be replaced.

2.9 *COMDECK

*COMDECK: Format (2A4, 1X, A3, 17A4)

Example:

```
1       9
*COMDECK DK01
```

A *COMDECK card identifies the cards following it as a COMDECK. The order of any COMDECKs in the input stream or on the old master file is not important, but these decks must be processed before any of the other decks. At least one blank character must separate the COMDECK character sequence and the deck name (DK01 in the example). The deck name is arbitrary and independent of the card identifiers in Columns 73-76. However, it is advantageous to have the same names for both identifiers. COMDECKs are updated by referencing the identifier in Columns 73-80. COMDECKs are called by referencing the deckname in a *CALL NAME card.

2.10 *END

*END: Format (A4)

Example:

```
1    5
*END
```

A *END card terminates processing of the COMDECK. All cards following a *COMDECK card are placed in the COMDECK and new master file until a *END card is encountered.

2.11 *CALL

*CALL: Format (A5, 1X, A2, 17A4, I4)

Example:

```
1    6
*CALL DK01
```

A *CALL card requests a COMDECK to be placed in the auxiliary file. The *CALL card resides on the old master file or is read in the input stream as a source card. The new master file includes the *CALL card. In the auxiliary file, the *CALL card is deleted and replaced with the contents of the requested COMDECK.

2.12 SOURCE CARD

SOURCE LANGUAGE CARD: Format (19A4, I4)

UPD will accept and process properly formatted source cards without requiring the presence of control cards. This is a normal situation when replacing or adding source cards. This mode of operation may be arbitrarily intermixed with the use of control cards.

If the deckfield (Columns 73-76) is non-blank and different from the current deck name, the effect is to issue a $LOCATE DECK XXX control card before proceeding. (Exception: In the $INSERTbbbb mode, the deck name is not checked.) The old master file is then copied until at the positioned at the source card having the same sequence number. The new source language card is inserted in the new master file and the corresponding card (if one exists) is deleted.

3.0 SAMPLE DECKS AND JCL

3.1 JCL Deck

A sample list of JCL cards sufficient to execute a UPD problem is given as follows:

```
//UPD          EXEC PGM=UPD,REGION=100K
//STEPLIB      DD   DSN=@1152.UPD7,DISP=(OLD,DELETE),VOL=REF=*.UPDL.DISK
//FT04F001     DD   DSN=@1152MEA.TOODEE2.SOURCE,UNIT=TAPE9,VOL=SER=901215,
//                  DISP=(OLD,KEEP),LABEL=(2,BLP,EXPDT=98000),
//                  DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//FT05F001     DD   DDNAME=SYSIN
//FT06F001     DD   SYSOUT=A
//FT07F001     DD   DSN=&&CARDS,UNIT=SYSDA,SPACE=(1600,(200,20)),
//                  DISP=(NEW,PASS),DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
//FT08F001     DD   DUMMY
//FT99F001     DD   UNIT=SYSDA,SPACE=(80,(2500,20),,CONTIG),
//                  DISP=(NEW,DELETE),DCB=(RECFM=F,BLKSIZE=80)
//SYSIN        DD   *
```

The STEPLIB card defines an executable load module and its location. The particular JCL options specified would be in accordance with the status of the load module.

The old master program library, if one exists, is defined by the FT04F001 card. This may be either disk or tape. The auxiliary file, if written, is defined by the FT07F001 card. This file is normally a temporary one to be passed to a compile step. The new master program is written onto the file defined by the FT08F001 card. In the example, no file is saved. The FT99F001 card defines the random access file for COMDECKs.

3.2 Sample UPD Decks

The examples of UPD decks illustrate the use of the control and source cards.

178 348

EXAMPLE 1: Make a master tape from cards read in the input stream. The program has three subroutines, MAIN, INPUT, and OUTPUT.

```
Col  1        9                                  73        80
     //SYSIN  DD
     $INSERT
     $NUMBER MAIN0010
             Source Cards for MAIN
     $NUMBER INPT0010
             Source Cards for INPUT
     $NUMBER OUTP0010
             Source cards for OUTPUT
/* or other JCL card.
```

The $INSERT operation specifies cards follow on the input stream and is necessary since no old master program library exists. Each $NUMBER card defines a new deck name and numbering sequence for the cards following them.

EXAMPLE 2: Replace cards INPT0350 with a new one, delete cards INPT0430 through INPT0510, insert INPT0615, and delete card INPT0700. Note that a card in the input stream with an identifier identical with one in the old master library is a replacement card. A card in the input stream which has no counterpart in the old master library is a new one to be inserted.

```
Col  1       8                                  73
     //SYSIN   DD   *
             X3 = X3 +1                         INPT0350
     $DELETE   INPT03400510
             X5 = X4 - 3                        INPT0615
     $DELETE   INPT0700
/* or other JCL card.
```

EXAMPLE 3: Create a master file with a COMDECK and main program. The COMDECK is named DK01, which is also used as the sequence identifier. The main program is labeled DK02. Note that $NUMBER cards are not required since the deck cards were previously sequenced. The *END card terminates the COMDECK.

```
Col  1                                          73
     //SYSIN   DD   *
     $INSERT
     *COMDECK DK01                              DK010000
              IMPLICIT REAL*8 (A-H,O-Z)         DK010010
     C        DECK TO TEST COMDECKS             DK010020
              COMMON/FTB/A(1),FIRST,LAST        DK010030
              LOGICAL FIRST                     DK010040
              INTEGER LAST                      DK010060
     C                                          DK010070

     *END     DK01
     $INSERT
     C        MAIN PROGRAM                      DK020010
     *CALL DK01                                 DK020020
              WRITE (6,101)                     DK020030
     101      FORMAT (1H1,5X,'TEST
              OF COMDECK')                      DK020040
              RETURN                            DK020050
              END                               DK020060
/* or other JCL card.
```

8 350