Distribution Category:
LMFBR—Physics: Base
Technology (UC-79d)

ANL-83-3

ANL--83-3

DE83 015740

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

THE UTILITY SUBROUTINE PACKAGE USED BY APPLIED PHYSICS DIVISION EXPORT CODES

by

C. H. Adams, K. L. Derstine,
H. Henryson II, R. P. Hosteny,*
and B. J. Toppel

Applied Physics Division

April 1983

*Reactor Analysis and Safety Division, ANL.

[illegible]

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

LIST OF FIGURES

LIST OF TABLES

THE UTILITY SUBROUTINE PACKAGE USED BY APPLIED PHYSICS DIVISION EXPORT CODES

by

C. H. Adams
K. L. Derstine
H. Henryson II
R. P. Hosteny
B. J. Toppel

ABSTRACT

This report describes the current state of the utility subroutine package used with codes being developed by the staff of the Applied Physics Division. The package provides a variety of useful functions for BCD input processing, dynamic core-storage allocation and management, binary I/O and data manipulation. The routines were written to conform to coding standards which facilitate the exchange of programs between different computers.

1. INTRODUCTION

The ARC System, a set of reactor physics analysis codes, was developed at Argonne during the 1960's for the laboratory's IBM computers. Fundamentally, it is a set of program modules which communicate using precisely defined data sets and which use a common set of utility subroutines, utility modules and coding conventions for dynamic storage allocation, file management and BCD input processing. Reference 1 describes the system characteristics and data set formats of the ARC System.

More recently reactor analysis codes developed for the Reactor Research and Technology (RRT) division of the Department of Energy have been subject to a set of standards defined by the Committee on Computer Code Coordination (CCCC). The CCCC standards define coding conventions and interface data sets which are intended to facilitate the exchange of computer codes between laboratories. Reference 2 describes the CCCC standards.

The codes currently being developed by Argonne's Applied Physics Division are written to CCCC specifications. They are "exportable" (i.e. designed to travel easily from one computer to another), they use CCCC data sets to communicate between code modules, and they use the few utility subroutines specified by CCCC for file management. However, they also use many of the ARC System features to which Argonne programmers and users have become accustomed.

The evolution from ARC System to CCCC environments has led to a proliferation of utility subroutines. In several instances there are multiple versions of the same routine: one for local ARC System use and others for applications on other machines at other installations.

This report and its likely subsequent revisions will document an "official" set of utility subroutines and modules that support local ARC System applications, the export of Argonne codes to other machines, and the implementation of codes written under CCCC standards at other laboratories. These routines have been tested on IBM, CDC and CRAY computers. This "official" set is available in object form for use on Argonne's IBM computers and in source form for stand-alone applications and for export. The source code will be distributed with reactor analysis codes sent to code centers.

The subroutines in the package provide a diverse set of services. Some of them can be grouped according to common functions:

1. There is a set of subroutines which satisfy the standard calling sequences defined by the Committee on Computer Code Coordination (CCCC). The functions performed by these routines are described in Reference 2; the programming details of our implementation of these standard calls are described in this report.

2. The BPOINTER package (Reference 1) for dynamic core-storage allocation was developed for ARC System codes and has been rewritten over the years into an exportable form. This report discusses the implementation for IBM, CDC and CRAY machines.

3. The FFORM routine reads free-format, BCD card image input.

4. SCAN and STUFF are BCD input preprocessors. This report describes the use of exportable and modular versions of these routines.

Subroutines in these categories are discussed in four separate chapters of this report. Not all the subroutines in the package are discussed in the text, and some that are discussed are referenced in more than one section. Appendix A is a collection of short writeups of all the routines.

Most are written in Fortran, with machine-dependent syntax set off by comment cards in a manner that permits the automated translation from one Fortran dialect to another. For situations where Fortran can not be used assembler routines appropriate to the host machine are provided.

2. MODULAR AND STAND-ALONE SYSTEMS

The reactor analysis production codes developed by the Applied Physics Division usually consist of a "driver" (or Standard Path in ARC System terminology) and a number of large, independent code blocks executed by the driver in some fixed sequence. Typically the set of code blocks includes several input processors, one or more computational modules and perhaps an output processor. They communicate with each other, and with the driver, by means of data sets; no problem data are passed in-core from one code block to another.

The production codes currently being developed are designed to run in two different environments: modular and stand-alone. In the modular format each code block, including the driver, is organized as a separate load module. Each load module contains versions of all the utility subroutines called from the module and is, in fact, an executable program. At Argonne, and at other IBM installations at which Argonne staff maintains codes, production codes are set up in the modular style.

In the stand-alone format the entire production code is a single load module. Usually the driver and the utility subroutines are contained in a base overlay; the code blocks executed by the driver are separate overlays. When production codes are organized for export (e.g. to the National Energy Software Center) they are set up in stand-alone style.

Figure 1 illustrates the structure of modular and stand-alone systems. The example is a code consisting of a driver and two code blocks, A and B. In a modular environment on IBM/370 systems execution is transferred to load modules A and B via the LINK macro.[1] In a stand-alone system A and B are primary overlays in a single load module.

In addition to sharing a common architecture, all Applied Physics production codes draw on a common set of utility routines which provide a wide variety of frequently used services and functions. Most of the utility routines exist as single Fortran versions, and that version is used in both modular and stand-alone environments. Some come in more than one version, however, usually for one of the following reasons:

1. A particular function may call for assembler language coding or for a system function peculiar to a particular computer or operating system,

2. Some of the routines must themselves be load modules in a modular environment.

This chapter describes the coding practices, program architecture and program libraries used by Applied Physics staff in production codes.

```
                     ___________________
                    |                   |  driver load
                    |      driver       |  module
                    |  (Standard Path)  |
                    |- - - - - - - - - -|
                    |   utilities for   |
                    |      driver       |
                    |___________________|
                      /
                     / via LINK
                    /   macro
  load module A    /                             load module B
    ______________/__                           ___________________
   |                 |                         |                   |
   |    code block   |                         |    code block     |
   |        A        |                         |        B          |
   |- - - - - - - - -|                         |- - - - - - - - - -|
   |  utilities for  |                         |  utilities for    |
   |  code block A   |                         |  code block B     |
   |_________________|                         |___________________|
```

(a) Modular structure

```
               _______________________________________
              |                                       |  main
              |                driver                 |  overlay
              |            (Standard Path)            |
              |- - - - - - - - - - - - - - - - - - - -|
              |             all utilities             |
              |- - - - - - - - - - - - - - - - - - - -|
              |                   |                   |
      primary |    code block     |    code block     |  primary
    overlay A |        A          |        B          |  overlay B
              |                   |                   |
              |___________________|___________________|
```

(b) Stand-alone structure

Figure 1. Alternative Structures for a Production Code

2.1 A CASE-STUDY INTRODUCTION

The traditional method of introduction to a new code or computer system is by example; a user or programmer starts from an input deck that is known to work. This is far more efficient than trying to use reference manuals to construct a job from scratch, and this is the approach we shall take to describe the programming practices used by Applied Physics Division staff in their production codes.

In this section we shall follow through the procedures for constructing both modular and stand-alone versions of an applications code package on the Argonne computers. The example should serve to illustrate the differences between stand-alone and modular systems and, perhaps more importantly, to provide models for programmers to use in setting up their own codes. The details of the features used in the following example are covered in depth in later sections of this report.

The example we will use is a very simple job that executes two Applied Physics Division production load modules (GNIP4C and HMG4C) and one special purpose load module (MYMOD). GNIP4C is a program that processes BCD data into binary files defining the geometry and atom number densities for a reactor model. HMG4C generates macroscopic cross sections from microscopic cross sections and number densities. Both GNIP4C and HMG4C are in public load module libraries available to anyone at Argonne. MYMOD is a very short module written expressly for this example; it reads a few data from a library file created by GNIP4C.

2.1.1 A Simple Standard Path

Standard Paths are the driver programs for particular calculations. In a modular system they are load modules which issue initialization calls to some of the utility routines and execute the applications load modules which perform all the computations. Figure 2 is a listing of the Standard Path for this example. The following points explain the important features of the Standard Path.

1. The common blocks /IOPUT/ and /STFARC/ are needed by certain of the utility routines. A complete list of the Applied Physics library of Fortran-callable utility routines, with instructions on their use, is given in Appendix A.

2. The logical unit numbers for the card-input and printer-output files are defined and stored in the common block /IOPUT/. Many of the current generation of codes under development offer the user two printer-output files, thus the two unit numbers NOUT and NOUT2. The purpose of this is to permit the user to route selected output to different output media (e.g. paper, microfiche, or a terminal file). NOUT2=0 implies there is no second printer-output file for the job.

3. The CCCC utility subroutine SEEK should be initialized in the Standard Path. SEEK manages the assignments of logical unit numbers for all files except the card-input and printer-output files. The names of the files are listed in the DSNAME array. The logical unit numbers of the files in this example are assigned in the order defined by the DSNAME array, starting with 11 (see Figure 4 for a listing of the DD cards for the execution step). SEEK is described in detail in the chapter on "CCCC Standard Subroutines" and in the writeups for SEEK, SEEKARC and SYS003 in Appendix A.

4. Subroutine SCAN ingests the entire BCD card input file (from logical unit NIN). The functions performed by SCAN are described in detail in the chapter on "BCD Input Conventions".

5. The call to STUFF processes all the BCD input under the input card "BLOCK=SAMPLE". STUFF creates individual BCD files from the input card-image data. The functions performed by STUFF are described in detail in the chapter on "BCD Input Conventions".

6. The load modules GNIP4C, HMG4C and MYMOD are executed through calls to the LINK routine.

```
      IMPLICIT REAL*8(A-H,O-Z)
C
C  COMMON BLOCKS REQUIRED BY THE UTILITY ROUTINES.
C
      COMMON /IOPUT/ NIN,NOUT,NOUT2
      COMMON /STFARC/ STFNAM,BLKNAM(50),IBLTAB(3,50),NBLOCK,NRET
C
C  DSNAME IS THE LIST OF NAMES OF THE FILES THAT THE JOB USES.
C
      DIMENSION DSNAME(9)
      DATA DSNAME / 6HISOTXS,6HCOMPXS,6HNDXSRF,6HZNATDN,6HSCR001,
     1 6HA.NIP3,6HGEODST,6HLABELS,1H$/
      DATA GNIP4C/6HGNIP4C/, HMG4C/5HHMG4C/, HMYMOD/5HMYMOD/
      DATA SAMPLE/6HSAMPLE/
C
C  INITIALIZE THE CARD INPUT FILE NO. (NIN), THE OUTPUT PRINT FILE
C  NOS. (NOUT AND NOUT2) AND THE SEEK TABLES.
C
      NIN=5
      NOUT=6
      NOUT2=0
      CALL SEEK(DSNAME,0,0,3)
C
C  SCAN/STUFF PREPROCESSING OF THE BCD INPUT.
C
      CALL SCAN
      STFNAM=SAMPLE
      CALL STUFF
      IF (NRET.LT.0) GO TO 20
C
C  EXECUTE THE CALCULATIONAL LOAD MODULES.
C
      CALL LINK(GNIP4C)
      CALL LINK(HMG4C)
      CALL LINK(HMYMOD)
   20 CONTINUE
      PRINT 500
  500 FORMAT(2X,3HEND)
      STOP
      END
```

Figure 2. A Sample Driver Module for IBM Machines Only

2.1.2 Calculational Modules

Calculational modules are load modules that are executed via calls to LINK in the Standard Path (e.g. GNIP4C, HMG4C and MYMOD in the sample Standard Path shown in Figure 2). Figure 3 shows the source code for a very simple calculational module. In general a calculational module can be any executable code; it does not have to use any of the utility routines described in this report. The following points explain the important features of the module listed in the Figure.

1. The module uses two utility routines, SEEK and REED, that require the /IOPUT/ common block. The variables in /IOPUT/ must be initialized at the beginning of each calculational module since that data has not been passed from the driver.

2. The logical unit number of the GEODST file is determined by a call to SEEK. The assignments of the logical unit numbers are made in the Standard Path only.

3. Binary files are written and read through the subroutines RITE and REED (see Appendix A).

4. Control is returned to the Standard Path via the Fortran RETURN.

```
      COMMON /IOPUT/ NIN,NOUT,NOUT2
      DIMENSION IBUFF(27)
      DATA GEODST/6HGEODST/
C
C  INITIALIZE THE OUTPUT PRINT FILE NOS.
C
      NOUT=6
      NOUT2=0
C
C  DETERMINE THE LOGICAL UNIT NO. FOR THE GEODST FILE, READ THE
C  SECOND RECORD, AND REWIND THE FILE.
C
      CALL SEEK(GEODST,1,NGEOD,0)
      CALL REED(NGEOD,2,IBUFF,27,0)
      CALL REED(NGEOD,0,IBUFF,0,0)
      PRINT 500,(IBUFF(I),I=1,27)
  500 FORMAT(2X,27I3)
      RETURN
      END
```

Figure 3. The Calculational Module MYMOD

2.1.3 Job Control Language

Figure 4 shows the Job Control Language required to execute this sample job in a compile-load-and-go mode. The first step in the job creates a load module library containing the calculational module MYMOD. The second step compiles and executes the Standard Path. The following points are important:

1. It is essential that when both the Standard Path and the calculational modules are created the linkage editor has access to the automatic call library C116.CCCC.SYSLIB. This library contains all the utility routines described in this report as well as versions of the Fortran I/O routines which use a special (and required) version of the IBCOM I/O module called ARCIBCOM. Even if a calculational module uses none of the utility routines it must be linked with

   ```
   PRELIB='C116.CCCC.SYSLIB'
   ```

   in order to make sure it uses the proper version of IBCOM.

2. MYMOD and the Standard Path are passed to the GO step in a temporary partitioned data set (&MODLIB) in this example.

3. The library C116.CCCC.MODLIB must be provided. It contains several utility modules that are needed for execution as well as the production modules GNIP4C and HMG4C.

4. Logical unit 9 is a scratch file required by SCAN to store a copy of the input-card-image file.

5. Logical unit 10, the second printer-output file, is dummied out in this example

6. Logical units 11 through 18 are the files listed in the DSNAME array in Figure 2

7. For an explanation of the input to the job see the discussion in the chapter "BCD Input Conventions".

```
//*   CREATE MODULE LIBRARY, MYMOD.
// EXEC FTXCEP,OPTIONS='XREF',PRELIB='C116.CCCC.SYSLIB'
//FTX.SYSIN DD *
          include here the MYMOD source
         code in Figure 3
/*
//EDT.SYSLMOD DD DSN=&MODLIB(MYMOD),DISP=(NEW,PASS),UNIT=SASCR,
//     SPACE=(TRK,(5,2,1)),DCB=BLKSIZE=6144
/*
//*   SECOND STEP. COMPILE AND EXECUTE STANDARD PATH.
// EXEC FTXCLG,OPTIONS='XREF',PRELIB='C116.CCCC.SYSLIB'
//FTX.SYSIN DD *
          include here the Standard Path
         source code in Figure 2
/*
//GO.STEPLIB DD DSN=&MODLIB,DISP=SHR
//            DD DSN=C116.CCCC.MODLIB,DISP=SHR
//GO.FT09F001 DD UNIT=SASCR,SPACE=(CYL,(1,1)),
//     DCB=(RECFM=VBS,LRECL=84,BLKSIZE=3064)
//GO.FT10F001 DD DUMMY
//GO.FT11F001 DD DSN=C116.Bnnnnn.VARIJOB.ISOTXS,DISP=SHR
//GO.FT12F001 DD DSN=&COMPXS,UNIT=SASCR,
//     SPACE=(TRK,(10,5)),DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//GO.FT13F001 DD DSN=&NDXSRF,UNIT=SASCR,
//     SPACE=(TRK,(2,1)),DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//GO.FT14F001 DD DSN=&ZNATDN,UNIT=SASCR,
//     SPACE=(TRK,(10,5)),DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//GO.FT15F001 DD DSN=&SCR001,UNIT=SASCR,
//     SPACE=(TRK,(10,5)),DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//GO.FT16F001 DD DSN=&ANIP3,UNIT=SASCR,
//     SPACE=(TRK,(10,5)),DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//GO.FT17F001 DD DSN=&GEODST,UNIT=SASCR,
//     SPACE=(TRK,(10,5)),DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//GO.FT18F001 DD DSN=&LABELS,UNIT=SASCR,
//     SPACE=(TRK,(2,1)),DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//GO.SYSIN DD *
BLOCK=OLD
DATASET=ISOTXS
BLOCK=SAMPLE
UNFORM=A.NIP3
02    0  1  500  0  500
03    10
04    3  2
06    REG  0.0  1.0  2
14    COMP  PU239 1.0
15    COMP  REG
/*
```

Figure 4. JCL for Compile-Load-and-Go Execution of the Example

2.1.4 Stand-Alone Form

Programs that are intended for release to code centers for general distribution are written so that they can be executed either in modular or stand-alone form. The example shown in Figures 2 and 3 requires only modest changes to run in a stand-alone mode. Figure 5 shows the source code for the Standard Path and MYMOD in stand-alone form. The important differences between the source code for the modular system and Figure 5 are:

1. The source code for the Standard Path and the calculational module MYMOD can be compiled at the same time for the stand-alone code.
2. The calculational modules (GNIP4C, HMG4C and MYMOD) are now subroutines and are executed via Fortran CALLs in the stand-alone code.
3. MYMOD requires a SUBROUTINE declaration in the stand-alone code.
4. NOUT and NOUT2 do not have to be reset in MYMOD in the stand-alone version since there is communication with the Standard Path via the /IOPUT/ common block.

Figure 6 shows the JCL required to execute the example in stand-alone form. The important differences between the modular system (Figure 4) and the stand-alone (Figure 6) are:

1. Several utility routines are pulled explicitly from the library C116.CCCC.SEGLIB instead of the automatic call library C116.CCCC.SYSLIB. The differences between the versions of these utilities in the two libraries are explained in the section in this chapter entitled "Applied Physics Utility Subroutine Libraries".
2. The object code for GNIP4C and HMG4C in the stand-alone version is pulled from private libraries (SEGLIB1 and SEGLIB2), and the entire package is link edited at one time.
3. GNIP4C, HMG4C and MYMOD are primary overlays. GNIP4C is further structured using secondary overlays. Linkage editor input for Applied Physics Division production codes is available at Argonne from the library C116.CCCC.OVERLAY.

```
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON /IOPUT/ NIN,NOUT,NOUT2
      COMMON /STFARC/ STFNAM,BLKNAM(50),IBLTAB(3,50),NBLOCK,NRET
      DIMENSION DSNAME(9)
      DATA DSNAME / 6HISOTXS,6HCOMPXS,6HNDXSRF,6HZNATDN,6HSCR001,
     1 6HA.NIP3,6HGEODST,6HLABELS,1H$/
      DATA SAMPLE/6HSAMPLE/
      NIN=5
      NOUT=6
      NOUT2=0
      CALL SEEK(DSNAME,0,0,3)
      CALL SCAN
      STFNAM=SAMPLE
      CALL STUFF
      IF (NRET.LT.0) GO TO 20
C
C  EXECUTE THE CALCULATIONAL LOAD MODULES.
C
      CALL GNIP4C
      CALL HMG4C
      CALL MYMOD
   20 CONTINUE
      PRINT 500
  500 FORMAT(2X,3HEND)
      STOP
      END
C
C  MYMOD IN SUBROUTINE FORM
C
      SUBROUTINE MYMOD
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON /IOPUT/ NIN,NOUT,NOUT2
      DIMENSION IBUFF(27)
      DATA GEODST/6HGEODST/
      CALL SEEK(GEODST,1,NGEOD,0)
      CALL REED(NGEOD,2,IBUFF,27,0)
      CALL REED(NGEOD,0,IBUFF,0,0)
      PRINT 500,(IBUFF(I),I=1,27)
  500 FORMAT(2X,27I3)
      RETURN
      END
```

Figure 5. Source Code for Stand-Alone Version of the Example

```
// EXEC FTXCEG,OPTIONS='XREF',PRELIB='C116.CCCC.SYSLIB',
//      EDTOPTS='LIST,MAP,OVLY',LSIZE='(200K,40K)'
//FTX.SYSIN DD *

    include here the source code shown in Figure 5

/*
//EDT.SEGLIB1 DD DSN=C116.Bnnnnn.SEGLIB,DISP=SHR
//EDT.SEGLIB2 DD DSN=C116.Bmmmmm.HMG4C.SEGLIB,DISP=SHR
//EDT.CCCCLIB DD DSN=C116.CCCC.SEGLIB,DISP=SHR
//EDT.SYSIN DD *
 ENTRY MAIN
 INCLUDE CCCCLIB(ARCBCD,LINES,REED,SEEK,TIMER)
 OVERLAY LEVEL1
 INCLUDE SEGLIB1(G4C10A)
 OVERLAY LEVEL2
 INCLUDE SEGLIB1(G4C11A,G4C11B,G4C11C)
 OVERLAY LEVEL2
 INCLUDE SEGLIB1(G4C12A,G4C12B,G4C12C)
 OVERLAY LEVEL2
 INCLUDE SEGLIB1(G4C13A,G4C13B,G4C13C,G4C13E)
 OVERLAY LEVEL2
 INCLUDE SEGLIB1(G4C14A,G4C14C)
 OVERLAY LEVEL2
 INCLUDE SEGLIB1(G4C15A,G4C15B)
 OVERLAY LEVEL2
 INCLUDE SEGLIB1(G4C16A)
 OVERLAY LEVEL2
 INCLUDE SEGLIB1(G4C17A,G4C17B)
 OVERLAY LEVEL1
 INCLUDE SEGLIB2(HMG00,HMG10,HMG20,HMG21,HMG22,HMG23,HMG30,HMG40)
 OVERLAY LEVEL1
 INSERT MYMOD
/*
//GO.STEPLIB DD DSN=C116.CCCC.MODLIB,DISP=SHR
//GO.FT09F001 DD UNIT=SASCR,SPACE=(CYL,(1,1)),
//     DCB=(RECFM=VBS,LRECL=84,BLKSIZE=3064)
//GO.FT10F001 DD DUMMY
//GO.FT11F001 DD DSN=C116.Bnnnnn.VARIJOB.ISOTXS,DISP=SHR
//GO.FT12F001 DD DSN=&COMPXS,UNIT=SASCR,

                 the remainder of the execution step
                 is as is shown in Figure 4
```

Figure 6. JCL for Stand-Alone Execution of the Example

2.2 APPLIED PHYSICS UTILITY SUBROUTINE LIBRARIES

A set of libraries exists at Argonne which contain the utility routines in all the forms that are required to support modular and stand-alone systems.

2.2.1 C116.CCCC.MASTER

Most of the source code for the utility subroutine package is contained in the LIBRARIAN file "C116.CCCC.MASTER". LIBRARIAN is a data storage and editing system designed specifically for developing and maintaining source code.[3]

Appendix A of this report consists of brief writeups of all members of this file. These writeups describe functions, calling sequences and applications. Table 1 lists the contents of C116.CCCC.MASTER, the system application (stand-alone or modular) for each member, and the language each member is written in.

TABLE 1

Contents of C116.CCCC.MASTER Library

Member	System Usage:		Language:	
Name	Stand-Alone	Modular	Fortran	Assembler
ABEND	X	X		X (IBM)
ARCBCD	X	X	X	
CRED	X	X	X	X (CD76)
DOPC	X	X	X	
DRED	X	X	X	
ECMV	X	X	X	
ERROR	X	X	X	
FEQUAT	X	X	X	
FFORM	X	X	X	
FLTSET	X	X	X	
IEQUAT	X	X	X	
IGTLCM	X	X	X	
INTSET	X	X	X	
IN2LIT	X	X	X	
LINES	X	X	X	
LINESARC		X	X	
LRED	X	X	X	
MYLCM	X	X		X (IBM)
POINTR	X	X	X	
REED	X		X	
REEDSIO	X	X	X	
SCANARC		X	X	
SECNDARC		X	X	
SECOND	X	X		X (IBM)
SEEK	X	X	X	
SEEKARC		X	X	
SEKPHL	X	X	X	
SIO	X	X		X (IBM)
SIOSUB	X	X		X (IBM)
SNIFF		X	X	
SPACE	X	X	X	
SQUEZE	X	X	X	
SRLAB	X	X	X	
STUFFARC		X	X	
SYS001		X	X	
SYS002		X	X	
SYS003		X	X	
SYS004		X	X	
SYS005		X	X	
TIMER	X	X	X	
TIME1	X	X		X (IBM)

2.2.2 C116.CCCC.SYSLIB

The object code routines in this file are in the form required for the modular system used at Argonne. If the user specifies

```
PRELIB='C116.CCCC.SYSLIB'
```

on the EXEC card of the linkage editor step, the linkage editor automatically includes appropriate versions of required utility routines when it creates application load modules. Tables 2 and 3 contain lists of the contents of the library C116.CCCC.SYSLIB as of April 25, 1983.

As Table 2 shows, most of the members of the library are compiled directly from source code in C116.CCCC.MASTER. Excepted from this rule are several members (see table 3) which have been copied from the ARC System library C116.ARC.SYSLIB. The source code for these ARC System routines have not been included in C116.CCCC.MASTER because they provide functions which are unique to the ARC System environment at Argonne and which are avoided in export version of codes.

The third column in Table 2 indicates which of the routines are "module interface subroutines." These routines are short subroutines which transfer execution temporarily to a utility load module from the module library C116.CCCC.MODLIB. Why some utility routines must themselves be load modules is discussed in the next section.

The last column shows aliases - other entry points - for some of the members of C116.CCCC.SYSLIB.

The writeups in Appendix A describe the functions of each of the members listed in Table 2.

TABLE 2

Contents of C116.CCCC.SYSLIB

C116.CCCC.SYSLIB Member	Corresponding C116.CCCC.MASTER Source Member	Module Interface Subroutine	Aliases
ABEND	ABEND		ABSTOP, TRACER
CRED	CRED		CRIT
DOPC	DOPC		DOPCD, DOPCO
DRED	DRED		DRIT
ECMV	ECMV		
ERROR	ERROR		
FEQUAT	FEQUAT		
FFORM	FFORM		
FLTSET	FLTSET		
IEQUAT	IEQUAT		
IGTLCM	IGTLCM		IGTSCM, IGTXCM, FRELCM, FRESCM, FREXCM, LOCFWD
INTSET	INTSET		
IN2LIT	IN2LIT		
LINES	LINESARC	yes	LINES2
LRED	LRED		LRIT
MYLCM	MYLCM		LOCF
POINTR	POINTR		DUMP
REED	REEDSIO		RITE
SCAN	SCANARC	yes	
SECOND	SECNDARC	yes	
SEEK	SEEKARC	yes	
SEKBCD	SEKBCD		
SEKPHL	SEKPHL		
SIO	SIO	yes	RECFM, SIOTRC
SNIFF	SNIFF	yes	
SPACE	SPACE		
SQUEZE	SQUEZE		GOWEST
SRLAB	SRLAB		
STUFF	STUFFARC	yes	
TIMER	TIMER		
TIME1	TIME1		CLOCK, DATE1, JOBID

TABLE 3

Contents of C116.CCCC.SYSLIB (from C116.ARC.SYSLIB)

C116.CCCC.SYSLIB Member	Module Interface Subroutine	Aliases
AVAIL		
DELETE		
FILEID		
IBCOM	yes	IBCOM#, DIOCS#, LDFIO#, IN#, OUT#, WAIT#, FRDNL#, FWRNL#, ERRSET, ERRSAV, ERRSTR, ERRMON, ERRTRA
LINK		
LOAD		
OPENDS		
TWAIT		
USERID		

These members were copied directly from C116.ARC.SYSLIB. The source code is not available from C116.CCCC.MASTER.

2.2.3 C116.CCCC.MODLIB

In a modular environment most of the utility routines described in this report are entirely contained within individual applications load modules. A few of the utility routines, however, must be in the form of separate load modules. Table 4 is a list of these utility modules.

The fundamental reasons for making a utility routine a load module are:

1. To save core storage.

2. The routine's function may require that there be only one copy of the routine available to all applications modules.

An example of the former is SYS002, which includes the BCD input preprocessors SCAN and STUFF; it is usually called only from drivers, and to include it in the driver throughout the execution of a job wastes space that could be used by applications load modules. An example of the latter is SYS004, which handles pagination for the output print file; pages could not be numbered consecutively if each load module used its own, independent version of the routine. A second example of the latter is SEEK, a subroutine which keeps a table indicating which files exist (i.e. have been written) and which do not exist. The SEEK table is initialized by the driver module and is updated and consulted, through subsequent calls to SEEK, by all the other load modules. It is essential, therefore, that there be only one table; individual load modules in a modular system cannot each have a copy of SEEK with its own table.

The top diagram in Figure 7 shows how SEEK is used in a modular system. The driver and each individual application load module contain a "module interface subroutine" version of SEEK whose sole function is to LINK to a utility load module, SYS003. SYS003 contains the SEEK subroutine that performs bookkeeping on the SEEK table. The version of SEEK in C116.CCCC.SYSLIB is the module interface subroutine. The lower diagram shows subroutine SEEK in a stand-alone code; in this case there is only one version of the routine present.

Programmers working with the libraries described in this section need not be familiar with the utility modules in C116.CCCC.MODLIB. All of them are accessed through module interface subroutines in C116.CCCC.SYSLIB that are automatically included in a load module when

```
PRELIB='C116.CCCC.SYSLIB'
```

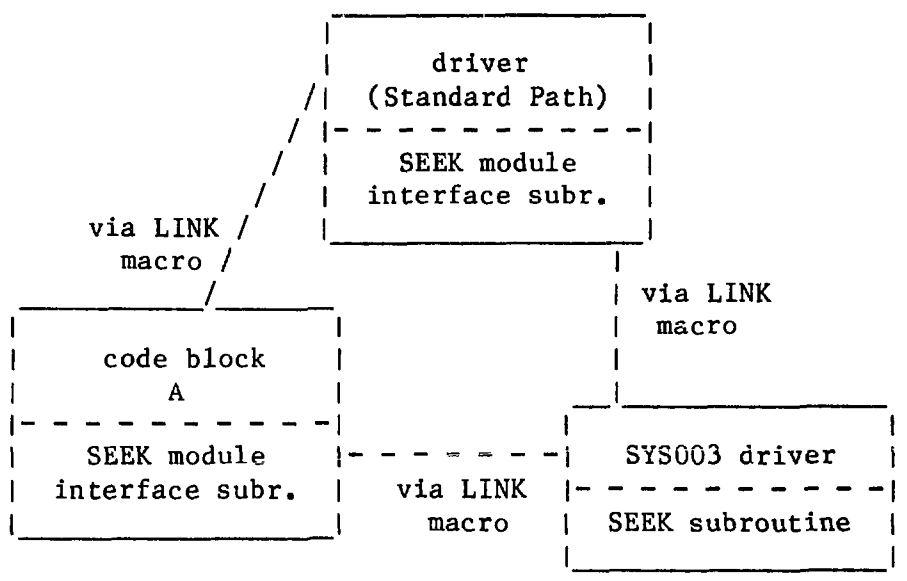
is specified on the EXEC card of the linkage editor step. C116.CCCC.SYSLIB utility routines which access utility modules are indicated in Table 2 in the column headed "Module Interface Subroutine."

TABLE 4

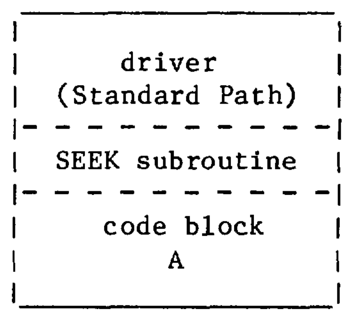
Utility Modules in C116.CCCC.MODLIB

C116.CCCC.MODLIB Member	REUSable	Aliases	Members of C116.CCCC.SEGLIB INCLUDEd
ARCIBCOM	yes		+
SIOSUB	yes		SIOSUB
SNIFF			SNIFF
SYS001	yes		SYS001
SYS002			SYS002, ARCBCD
SYS003	yes		SYS003, SEEK
SYS004	yes		SYS004, LINES
SYS005	yes		SYS005, SECOND

+This member was copied directly from C116.ARC.MODLIB.



(a) SEEK in a modular system. SEEK is accessed from both the driver and code block A.



(b) SEEK in a stand-alone system

Figure 7. Organization of the Utility Routine SEEK

Most ARC System modules exist in-core only while they are in execution and are not saved when execution is transferred to another module. The ARC System REUS capability (see Reference 1) has been used to keep specific utility modules in-core throughout the execution of a job. Table 4 indicates which utility modules are REUSable, that is, which are kept in-core for the entire job.

The REUSable modules ARCIBCOM and SIOSUB are automatically loaded on the first call to the system I/O routine IBCOM#. SYS001, SYS003, SYS004 and SYS005 are automatically loaded on the SEEK initialization call.

Where appropriate, individual writeups in Appendix A show linkage editor input for utility modules. Many of the linkage editor inputs for utility and production load modules are available from the file C116.CCCC.OVERLAY.

The one module, ARCIBCOM, that cannot be created from source code in C116.CCCC.MASTER was copied from C116.ARC.MODLIB.

2.2.4 C116.CCCC.SEGLIB

The library C116.CCCC.SEGLIB serves one or both of two applications:

1. To contain object code required in the creation of utility and applications load modules.

2. To make available in stand-alone subroutine form some of the utility routines represented by module interface subroutines in C116.CCCC.SYSLIB. A programmer would use the stand-alone form of a subroutine when running a code at Argonne in a stand-alone environment.

Table 5 lists the members of C116.CCCC.SEGLIB, lists the corresponding C116.CCCC.MASTER source members, and indicates which of the applications each member serves. Ordinarily programmers working in the Argonne modular environment will not have any use for this library; they will be accessing all utility routines through members of C116.CCCC.SYSLIB.

In C116.CCCC.SYSLIB SCAN, STUFF, LINES, SECOND and SEEK are module interface subroutines; they only provide a link to utility load modules. In C116.CCCC.SEGLIB ARCBCD (which contains SCAN and STUFF), LINES, SECOND and SEEK are the utility routines which do the actual work. It is the SEGLIB versions of these routines which are included in the utility modules or in stand-alone codes.

In C116.CCCC.SYSLIB REED provides an asynchronous I/O option (at the cost of requiring extra baggage in the form of the assembler routines SIO and SIOSUB). The version of REED in C116.CCCC.SEGLIB is simpler and is probably adequate for most applications. It is also more easily exported because of the absence of assembler code.

TABLE 5

Object Code Segments in C116.CCCC.SEGLIB

C116.CCCC.SEGLIB Member	Corresponding C116.CCCC.MASTER Source Member	Required for Load Module	Stand-Alone Subroutine
ARCBCD	ARCBCD	yes (SYS002)	yes
LINES	LINES	yes (SYS004)	yes
REED	REED	no	yes
SECOND	SECOND	yes (SYS005)	yes
SEEK	SEEK	yes (SYS003)	yes
SIOSUB	SIOSUB	yes (SIOSUB)	
SNIFF	SNIFF	yes (SNIFF)	yes
SYS001	SYS001	yes (SYS001)	
SYS002	SYS002	yes (SYS002)	
SYS003	SYS003	yes (SYS003)	
SYS004	SYS004	yes (SYS004)	
SYS005	SYS005	yes (SYS005)	

2.3 SPECIAL FILE AND COMMON BLOCK NAMES

Within the system described in this report there are two file names that a programmer should avoid. One is "ARC", which is the name of a file onto which SCAN (see the chapter on "BCD Input Conventions") spools the BCD input. The other is "$", which is a symbol used in the SEEK initialization procedure to signal the end of the file name list (see the writeup for SEEK in Appendix A).

Many of the utility routines reference data in labeled common blocks, and in coding programmers must avoid labels that conflict. One or more of the routines described in Appendix A use the following labeled common blocks:

/ALLOCS/
/BFLAGS/
/CRALOC/
/INITIO/
/ECMLOC/
/ECMRF1/
/ECMRF2/
/IOPUT/
/LCMSIZ/
/LOCATE/
/PTERR/
/PTITLE/
/RNDMIO/
/SAVMEM/
/STFARC/
/TABLES/

2.4 MACHINE-DEPENDENT CODING

In organizing the source library (C116.CCCC.MASTER) of utility routines we made a special effort to generalize the coding to run on any machine. Most of the source code is written in standard Fortran IV (i.e. Fortran '66) and has been compiled successfully by a number of compilers. This section discusses approaches taken in those few situations where it is impossible to avoid machine-dependent code. To date we have considered IBM System/370, CDC Scope 2.1, CRAY-1 under CTSS and CRAY-1 under COS 1.10 (CFT 1.09) systems.

2.4.1 Fortran Coding

Even when a routine is entirely coded in Fortran it may contain machine-dependent code. For example, subprogram ENTRY statements may have arguments in IBM Fortran but not in CDC Fortran.

We have used the device of surrounding machine-dependent Fortran with pairs of special "keyword" comment cards. The coding between a keyword pair is selectively activated or deactivated by a simple preprocessing Fortran program which places a blank or the letter C in column 1 of the bracketed card images. Table 6 indicates the set of keywords which must be activated for export to the large scale scientific computers which have been considered to date.

Keyword syntax permits three basic forms:

1. simple keywords - "CDC*" OR "CIBM"

2. compound keywords - "CDC*-OVL"

3. negated simple or compound keywords - "COVL-" or "CDC*-OVL-"

Simple keywords consist of at most four non-blank alphanumeric characters, the first character of which must be the letter "C". Compound keywords must have the initial character "C" in the second keyword replaced by a hyphen; this hyphen must be positioned in column 5 of the card image. A compound keyword is activated when its constituent keywords are both activated. A hyphen suffix with simple or compound keywords toggles the activity state of the corresponding keyword.

The keywords fall into three general categories:

1. Keywords delineating machine architecture features which can be considered independent of computer manufacturer (CSW/CLW, C1LV/C2LV).

2. Keywords delineating compiler and loader features unique to classes of computer manufacturers (CIBM, CDC*, CRAY, CD76, CENT, CTSS, CCOS, COVL, CSEG).

3. Keywords distinguishing stand-alone and modular environments (CANL,CSA).

The CSW/CLW keywords delineate coding peculiar to short-word and long-word machines respectively. A short word machine may require double precision arithmetic to obtain sufficient numerical accuracy, whereas single precision arithmetic is sufficient on long-word machines. Typical code delineated here might be precision specification statements and precision-dependent function calls. The CSW/CLW keywords are mutually exclusive.

The precision specification for Hollerith variables is usually treated with CSW/CLW keywords, although it would not be impossible for the conscientious programmer to treat Hollerith data via the category (2) keywords. The latter practice would provide the option of exercising either short-word or long-word arithmetic on the same machine.

The C1LV/C2LV keywords delineate coding peculiar to one-level and two-level storage hierarchy machines. The IBM and CRAY machines are one-level machines; they have a single level of fast core memory. The CDC 7600, on the other hand, has two levels of memory, a small core memory (SCM) and a large, relatively slower memory (LCM). The C1LV/C2LV keywords are mutually exclusive.

TABLE 6

Keyword Correspondence for Code Export

Keyword	IBM 370	CDC 7600	CRAY	CDC STAR	UNIVAC
CSW	X				X
CLW		X	X	X	
C1LV	X		X	X	X
C2LV		X			
CENT	X		X	X	X
CIBM	X				X
CDC*		X		X	
CRAY			X		
CD76		X			
CUNI					X
CANL					
CLBL		(X)			
CSA	(X)	X	X	X	X
COVL		(X)	(X)		
CSEG		(X)	(X)		
CTSS			(X)		
CCOS			(X)		

The distinction between one-level and two-level machines is kept independent of computer manufacturer, largely to permit us to exercise both one-level and two-level logic on either type of machine.

The CIBM/CDC*/CRAY/CUNI keywords are mutually exclusive. Used with the COVL keyword they delineate miscellaneous compiler differences in the implementation of overlay calling sequences and system functions including END-OF-FILE tests and memory address requests.

The CENT keyword addresses the different compiler implementations of ENTRY points. It is used to delineate the ENTRY point argument list.

Two keywords, CANL and CLBL, are used to demarcate coding to be activated at two specific installations, Argonne (CANL) and Lawrence Berkeley Laboratory (CLBL). These keywords are used mostly to take advantage of special features of the local operating system. In addition, CANLs bracket features peculiar to the modular system in use at Argonne; CSAs bracket features peculiar to stand-alone program architecture.

The keyword CDEC is treated differently than the remaining keywords. If the CDEC keyword is active, then card-images with the characters "CDECK" in columns 1-5 are changed to "*DECK" in columns 1-5. This feature is primarily useful for users of the CDC UPDATE utility. The CDECK cards are always tabulated by the preprocessor program and thereby provide an index of their associated code blocks or subroutines.

The COVL keyword is intended to bracket card images that are dependent on whether or not overlay processing has been requested. Concurrent with the activation of cards bracketed by COVL keywords, a search for card-images with OVERLAY or CALL OVERLAY text is initiated for the purpose of replacing the existing overlay file name in these card images with the user-supplied non-blank file name provided with the control input for the preprocessor program. The replacement is performed whether or not the COVL keyword is active. The search is bypassed when the user-supplied name is blank.

The CSEG keyword is used to bracket coding that provides the functional equivalent of indirect addressing of segment names. Certain segmented loader inplementations do not permit segment names to be passed as a subroutine argument.

Keywords CTSS and CCOS are used to bracket coding intended for the CRAY Timesharing (Operating) System and the standard CRAY Operating System, respectively.

Table 7 summarizes the intended applications for the present list of keywords. Examples of keyword use are found in Figures 8 and 9 and in the example given in the chapter on FFORM. The list of keywords is necessarily open-ended to permit expansion to new machines as they arise.

The implementation of the preprocessor program has purposely been kept simple. It requires control input (see Fig. 10) which consists of a list of simple keywords to be activated, the logical unit numbers of the input and output BCD files, an optional overlay file name to be inserted in OVERLAY directives, and a character code conversion sentinel (0/1/2) which requests: no conversion, translation of EBCDIC (029) to BCD (026), or vice versa.

TABLE 7

Keyword Usage Summary

Keyword	Usage
CSW/	Short-word/Long-word machine
CLW	1. precision specifications for floating point numbers
	2. precision of function references
	3. precision specifications for Hollerith variables
	4. word length indicator (MULT=1 or MULT=2)
C1LV/	Coding unique to one- or two-level memory machines
C2LV	1. calls to CRED,CRIT,ECMV,ECZERO,LRED and LRIT (optional)
CIBM/	Machine Manufacturer and Compiler Features
CDC*/	1. END-OF-FILE tests
CRAY/	2. overlay calling sequences
CUNI	3. ENCODE/DECODE calls
	4. FORMAT statements with the O or Z format codes
	5. random access I/O
CD76	CDC 7600
	1. LEVEL specifications for LCM variables
	2. calls to READEC and WRITEC
	3. ECM container reference point initialization (JLOC=0)
CENT	Entry points
	1. arguments of entry points
CANL	Local Argonne usage
	1. modular system features, modules are executed by "CALL LINK"
	2. special features of the Argonne operating system
CLBL	Local Lawrence Berkeley Laboratory usage
	1. special features of the Berkeley operating system
CSA	Stand-alone usage
	1. invocation of modules via subroutine "CALLs"
COVL	Overlay usage
	1. CALL OVERLAY statements
	2. OVERLAY directives (CDC*)
	3. dynamic replacement of overlay file names (in 1. and 2. above) by the user-supplied name
CSEG	Segmented loader usage
	1. simulation of indirect calls to overlay segments
	4. RETURN statement in main overlays
CTSS/	Coding unique to CRAY systems with the CTSS operating system
CCOS	1. dynamic memory allocation
	2. LOCF/LOC function - machine address of a variable

```
      SUBROUTINE XAMPL (A, B, C, N, LLOCA)
CSW
      DOUBLE PRECISION A,B,C,STRING
CSW
      DIMENSION A(N), B(N), C(N), STRING(5)
      DATA STRING/ 6HTHIS I, 6HS A HO, 6HLLERIT, 6HH STRI, 6HNG.   /
C
C     INITIALIZE WORD LENGTH INDICATOR
CSW
      MULT=2
CSW
CLW
C     MULT=1
CLW
      RETURN
C************************************************************************
      ENTRY POINT1
CENT
     1             (A, B, C, N, LLOCA)
CENT
C************************************************************************
      DO 10 I=1,N
CSW
      A(I)=DCOS(B(I)+C(I))
CSW
CLW
C     A(I)=COS(B(I)+C(I))
CLW
   10 CONTINUE
C
C     COPY A TO ECM BEGINNING AT ECM LOCATION LLOCA.
C2LV
C     CALL CRIT(A, LLOCA, N*MULT, IER)
C2LV
      RETURN
      END
```

Figure 8. An Example of Keyword Usage

```
CDECK SAMPLE
CDC*-OVL
C     OVERLAY(OVLNAM,1,0)
C     PROGRAM SAMPLE
CDC*-OVL
CDC*-OVL-
      SUBROUTINE SAMPLE
CDC*-OVL-
      .
      .
      .
COVL
C     CALL OVERLAY(6HOVLNAM,1,1,0)
COVL
COVL-
      CALL OVL11
COVL-
      .
      .
      .
CDC*-OVL-
      RETURN
CDC*-OVL-
      END
```

Figure 9. An Example Using Keyword COVL

```
CARD 1      N2926,NIN,NOUT,OVNAME  (3I3,A6)
CARD 2     ((WORDIN(I,J),I=1,4),J=1,N) (20A4)


     N2926       0/1/2.  NO CHARACTER CONVERSION/029 TO 026
                 (EBCDIC TO BCD)/026 TO 029.
     NIN         INPUT LOGICAL UNIT NUMBER.
     NOUT        OUTPUT LOGICAL UNIT NUMBER.
     OVNAME      6-CHARACTER OVERLAY NAME.
     WORDIN(.,J) THE J-TH 4-CHARACTER KEYWORD.  EACH KEYWORD MUST
                 START WITH THE LETTER C AND MUST BE LEFT
                 JUSTIFIED IN THE FIELD.
     N           THE NUMBER OF KEYWORDS TO BE ACTIVATED.  THE
                 CODE DETERMINES THIS NUMBER BY COMPARING EACH
                 4A1 FIELD WITH AN INTERNAL LIST OF ALLOWED
                 KEYWORDS.  BLANK KEYWORDS ARE IGNORED.  KEYWORDS
                 THAT ARE NOT LISTED IN THE INPUT ARE DEACTIVATED.
```

Figure 10. Input Description for the Preprocessor CONVTCD

2.4.2 Assembler Coding

There are some functions which require assembler coding. In the library C116.CCCC.MASTER the members ABEND, MYLCM, SECOND, SIO, SIOSUB and TIME1 are all IBM assembler code. Member CRED includes appropriately bracketed Compass assembler coding for the subroutines WRITEC and READEC.

2.4.3 Generic Overlay Number Assignments

The overlay numbering assignments tabulated in Table 8 have been established to expedite the export of large scale Applied Physics production codes that employ many common utility modules (overlays). Adherence to the suggested numbering convention combined with the appropriate use of the COVL keyword cards within user supplied source code blocks should fully automate the code export process.

TABLE 8

Generic Overlay Number Assignments

Overlay Number (Decimal)	Overlay Number (Octal)	
1	1	SCAN
2	2	STUFF
3	3	GNIP4C
4	4	HMG4C
5	5	MODCXS
6	6	SRCH4C
7	7	CSE010 +
8	10	LASIP3 +
9	11	SUMMARY +
10	12	
11	13	
12	14	
13	15	
14	16	
15	17	UDOIT6
16	20	UDOIT5
17	21	UDOIT4
18	22	UDOIT3
19	23	UDOIT2
20	24	UDOIT1
	NEUTRONICS BLOCKS	
21	25	BCDINP
22	26	BININP
23	27	SSINIT
24	30	SSTATE
25	31	DNHSST
26	32	
27	33	
28	34	
29	35	DSSTOU
30	36	
	SUPER CODE BLOCKS (VARI3D, REBUS3, ETC.)	
31	37	
.	.	
.	.	
.	.	

+Export of these code blocks is not supported.

3. CCCC STANDARD SUBROUTINES

Reference 2 describes a set of subroutine calls defined by the Committee on Computer Code Coordination (CCCC) which standardizes data management in order to facilitate the exchange of programs between different computers and laboratories. Only the calling sequences and functions are standardized; the actual coding of each routine is left to individual installations.

The set of routines developed at ANL are designed to operate on machines with either one level of memory (e.g. IBM and Cray computers) or two levels (e.g. the CDC 7600). The machine-dependent coding has been kept to a minumum. Not only does this approach make code export easier, it also permits the testing of a two-level data-management strategy on a one-level machine.

The calling sequences and functions are defined fully in Reference 2 and briefly in Appendix A of this report. This section goes into some of the coding details for the versions of the CCCC subroutines included in the utility subroutine package.

3.1 SEEK

In the ANL interpretation of the CCCC standards all data sets except the output print file and input card image file are given names and version numbers. Some file formats (e.g. those containing isotopic neutron cross sections or the neutron flux distributions) are defined by the CCCC, but others (e.g. the file used by Applied Physics codes to store macroscopic cross sections) are code-dependent. Subroutine SEEK provides the connection between file names and file reference numbers, even for scratch files. SEEK is very similar to the ARC System routine SNIFF.[1]

The Fortran source code for SEEK is contained in member SEEK of the LIBRARIAN source file C116.CCCC.MASTER; the corresponding object code is in member SEEK of the C116.CCCC.SEGLIB library. In a modular environment the function of SEEK is provided by the utility load module SYS003 in C116.CCCC.MODLIB. SYS003 is link edited by combining members SYS003 and SEEK from C116.CCCC.SEGLIB. In a modular environment the version of SEEK which is built into applications load modules is a "module interface subroutine"; its sole function is to transfer execution to the load module SYS003. Source code for the module interface subroutine version of SEEK is in member SEEKARC of C116.CCCC.MASTER; the corresponding object code is member SEEK of C116.CCCC.SYSLIB. For further details see the writeups for SEEK and SEEKARC in Appendix A.

SEEK must create and maintain a table (the "SEEK table") that associates each unique file name and version number pair with a "file reference number". The SEEK table must also tell whether a file "exists" (i.e. has had something written into it) or not. The method of initializing the SEEK table is entirely up to the individual installation. The ANL version of SEEK permits two different methods for initialization. Both are described in the writeup of SEEK in Appendix A. One is the same as the procedure required by SNIFF, the other is more flexible. SEEK must be initialized before any files (binary or BCD) are read.

A distinction must be made between the "file reference number" used in the arguments of CCCC routines and the "logical unit number" that a programmer codes into a Fortran I/O statement. In the Los Alamos implementation of the CCCC standards the two are not the same. The programmer need not be concerned with the difference when dealing with binary files since all I/O is performed through calls to CCCC routines; applications programs should contain no Fortran I/O statements for binary files. It is a common ANL practice, however, to employ a number of BCD input files and to manage them with SEEK. This means that ANL coding contains calls to SEEK which reference file reference numbers as well as Fortran I/O READs and WRITEs which reference logical unit numbers. The correspondence between the two numbers is managed by means of a subroutine, SEKPHL, which is described later.

Because we employ subroutine SEEK with BCD and random access files in addition to sequential access files, it is instructive to review the following guidelines to avoid potential portability problems.

1. A call to SEEK with the proper read/write mode flag must be issued prior to the first read or write to a data set and prior to the first read or write to a data set that has been rewound. This practice is necessary for compatibility with implementations that dynamically assign file reference numbers upon each call to SEEK and dynamically release file reference numbers after a data set rewind command is received. A call to the appropriate routine, REED or RITE, with a record number of zero rewinds the data set.

2. The logical unit number for BCD data sets must be obtained by calling subroutine SEKPHL following the call to SEEK . SEKPHL returns the logical unit number corresponding to the file reference numbers returned by SEEK. SEKPHL must also be used to rewind and close BCD files. Figure 11 illustrates the usage of SEKPHL. Subroutine SEKBCD combines the calls to SEEK and to SEKPHL and is an alternative to calling SEEK and SEKPHL separately. Figure 12 illustrates the usage of SEKBCD. The calling sequences for SEKPHL and SEKBCD are found in Appendix A.

3. A set of fifteen generic file names (RNDM01-RNDM15) have been reserved for random access I/O applications. In order to maintain portability, calls to SEEK for random access data sets are embedded within our version of DOPC and DRED/DRIT. Consequently, SEEK calls for random access files are not otherwise necessary and should never be coded by the programmer.

4. Successive calls to SEEK (with different read/write mode flags) without intervening rewinds must be avoided. Such situations may arise when SEEK is called in a read mode solely to determine file existence. If the file exists, but the programmer does not intend to read the file, then REED (for binary files) or SEKPHL (for BCD files) should be called

to rewind it. Later, if the file is actually to be read, SEEK must be called again.

The version of SEEK in the utility package performs no finalizing or wrap-up function (NOP=2). The other operations specified by the CCCC standard are all implemented (NOP = 0, 1, 3, 4, 5).

```
      DATA BCDNAM/6HINPUT /, IVER/1/, MODE0/0/, MODE1/1/
C
C     OBTAIN DATA SET REFERENCE NUMBER VIA SEEK
C     OBTAIN DATA SET LOGICAL UNIT NUMBER VIA SEKPHL
C
      CALL SEEK ( BCDNAM, IVER, NREF, MODE0 )
      IF( NREF.LE.0 ) GO TO 2
      CALL SEKPHL ( NREF, LUN, MODE0 )
      IF( LUN .LE.0 ) GO TO 1
C
C     READ FORMATTED DATA FROM LOGICAL UNIT NUMBER LUN
C
      READ (LUN,10) DATA
   10 FORMAT( A4 )
C
C     REWIND DATA SET
C
    1 CONTINUE
      CALL SEKPHL ( NREF, LUN, MODE1 )
    2 CONTINUE
```

Figure 11. SEKPHL Usage

```
      DATA BCDNAM/6HINPUT /, IVER/1/, MODE0/0/, MODE1/1/
C
C     OBTAIN THE DATA SET REFERENCE NUMBER AND DATA SET LOGICAL
C     UNIT NUMBER VIA SEKBCD
C
      CALL SEKBCD ( BCDNAM, IVER, LUN, MODE0, NREF)
      IF( NREF.LE.0 ) GO TO 2
      IF( LUN .LE.0 ) GO TO 1
C
C     READ FORMATTED DATA FROM LOGICAL UNIT NUMBER LUN
C
      READ (LUN,10) DATA
   10 FORMAT( A4 )
C
C     REWIND DATA SET
C
    1 CONTINUE
      CALL SEKBCD ( BCDNAM, IVER, LUN, MODE1, NREF)
    2 CONTINUE
```

Figure 12. SEKBCD Usage

3.2 SNIFF

SNIFF is not a CCCC subroutine; it is the ARC System equivalent of SEEK.[1] Because of the local requirement that we be able to mix ARC System and CCCC modules in the same job it is necessary that there be some way to direct CALLs to both SEEK and SNIFF to the same utility load module.

The version of SNIFF in this utility routine package simply executes the SEEK utility load module SYS003. It will return file reference numbers prior to reads or writes, it will execute the "CHANGE" option, and it will set and erase existance flags. It will not execute the SEEK table initialization call; that must be done through a direct call to SEEK.

The source code for SNIFF is in member SNIFF of C116.CCCC.MASTER. The object code is in both C116.CCCC.SYSLIB and C116.CCCC.SEGLIB. SNIFF is available as a utility load module in C116.CCCC.MODLIB.

3.3 TIMER

TIMER is a subroutine that provides the programmer access to such quantities as the elapsed time, the current date and the wall clock time. All of these quantities must come from somewhere in the operating system, and so the coding of TIMER is very machine-dependent.

The source code in member TIMER of C116.CCCC.MASTER is designed for both local and export applications. Calls to special system routines are demarcated by the special comment cards CIBM, CDC*, CRAY, CANL and CLBL as described in the section in the previous chapter on "Machine-Dependent Coding." The options available in these two versions are listed in Table 9 .

The IBM export version of TIMER requires the assembler routines DATE1, JOBID, CLOCK, TIME1 and SECOND contained in members TIME and SECOND of C116.CCCC.MASTER. The CDC export version requires the generally available system routines SECOND and DATE. The CRAY export version calls SECOND, DATE and CLOCK.

At Argonne TIMER calls the system routines TLEFT, TWAIT and USERID in addition to the routines called in the IBM export version. At Lawrence Berkeley Laboratory TIMER calls the local routines STATUS, JOBCARD and HOUR in addition to the CDC system routines SECOND and DATE.

The object code for TIMER is in C116.CCCC.SYSLIB.

TABLE 9

Options Available in Various Versions of TIMER

Sen-tinel	Parameters Returned	Versions: ANL, SYSLIB	IBM, EXPORT	CDC, EXPORT	CRAY, EXPORT	LBL, LOCAL
-1	return all+	yes	yes	yes	yes	yes
0	initialize	yes	yes	yes	yes	yes
1	elapsed CP time *	yes	yes	yes	yes	yes
2	time left *	yes				yes
3	elapsed PP time *	yes				yes
4	date (A8)	yes	yes	yes	yes	yes
5	user ID	yes				yes
6	user charge no.	yes				yes
7	user job name	yes	yes			yes
8	wall clock (A6)	yes	yes		yes	yes
10	wall clock (A8)	yes	yes			

* time in seconds, elapsed time is time since last call to TIMER with a sentinel of 0

+ "all" means all parameters available for a particular version of TIMER

3.4 REED/RITE

The CCCC standards require that all binary I/O operations be executed through calls to standard subroutines, not through Fortran I/O coded into applications programs. This practice permits individual installations to take advantage of locally available, efficient access methods without recoding programs; all that is needed is a local set of CCCC standard I/O routines.

REED and RITE are the CCCC routines specified for binary sequential data transfer between fast core (FCM) and external data files (disk). The calling sequence for REED is:

```
CALL REED(NREF,IREC,ARRAY(I),NWDS,MODE)
```

This call transfers NWDS single-precision words from record number IREC of the sequential file with file reference number NREF to the FCM locations starting at the address of ARRAY(I). A similar call to RITE performs the inverse operation. MODE is a sentinel that permits a programmer to code buffered I/O. When MODE=0 I/O operations are completed before the return from REED/RITE. When MODE=1 I/O operations are not necessarily completed before the return to the calling program; a subsequent call with MODE=2 is required to complete the outstanding I/O operation.

Figure 13 shows examples of code that use REED/RITE both in the normal mode (MODE=0) and with buffering (MODE=1).

```
      DIMENSION A(NWDS), B(NWDS)
      DATA IO/0/
C
C  LOOP OVER NRECS RECORDS IN FILE LUNA.  READ EACH RECORD.
C
      DO 10 IREC=1,NRECS
      CALL REED(LUNA,IREC,A,NWDS,IO)
C
C  COMPUTE B FROM A.  WRITE RECORD IREC.
C
      CALL CALC(A,B,NWDS)
      CALL RITE(LUNB,IREC,B,NWDS,IO)
   10 CONTINUE
```

(a) Conventional sequential use of REED/RITE.

```
      DIMENSION A(NWDS,2), B(NWDS,2)
      DATA I1/1/, I2/2/, IBUF1/1/, IBUF2/2/
C
C  LOOP OVER NRECS RECORDS IN FILE LUNA.  FINISH READING RECORD
C  IREC BEFORE STARTING TO READ IREC+1.
C
      CALL REED(LUNA,I1,A(1,IBUF1),NWDS,I1)
      DO 10 IREC=1,NRECS
      CALL REED(LUNA,IREC,A(1,IBUF1),NWDS,I2)
      IF( IREC.LT.NRECS ) CALL REED(LUNA,IREC+1,A(1,IBUF2),NWDS,I1)
C
C  COMPUTE B FROM A.  FINISH WRITING RECORD IREC-1 OF FILE LUNB
C  BEFORE STARTING TO WRITE IREC.  ROTATE BUFFERS.
C
      CALL CALC(A(1,IBUF1),B(1,IBUF1),NWDS)
      IF( IREC.GT.1 ) CALL RITE(LUNB,IREC-1,B(1,IBUF2),NWDS,I2)
      CALL RITE(LUNB,IREC,B(1,IBUF1),NWDS,I1)
      I=IBUF2
      IBUF2=IBUF1
      IBUF1=I
   10 CONTINUE
```

(b) Use of REED/RITE with buffering.

Figure 13. REED/RITE Usage

3.4.1 The Export REED/RITE, Sequential I/O Only

The export version of REED/RITE is a relatively simple routine that uses ordinary Fortran I/O. Though the records of a sequential file are necessarily written consecutively, the CCCC standards provide that they may be accessed in a random fashion. Therefore, the export REED/RITE keeps track of the current record number on all files, and uses REWINDs and dummy READs, if necessary, to position a file before reading the next record requested. Calls with MODE=1 are treated the same as MODE=0; calls with MODE=2 are ignored.

The source code for this export version of REED/RITE is contained in member REED of the C116.CCCC.MASTER library. The corresponding object code is member REED of C116.CCCC.SEGLIB.

3.4.2 REED/RITE with SIO Asynchronous Option

The source code in member REEDSIO of C116.CCCC.MASTER is the production version of REED/RITE that is used at Argonne. The code is intended for IBM implementation, since special access methods are used which are programmed in IBM assembler language. In addition to providing the standard sequential I/O capability of the Fortran language, this version of REED/RITE provides an asynchronous, random access I/O capability. The SIO program is used to obtain this capability. In short, SIO uses IBM BSAM macro instructions, along with internal tables and absolute track addressing, to process the I/O requests. SIO was originally written for the OS/MVT operating system as a more efficient and more convenient alternative to IBM Fortran Direct Access. The current version of the routine runs under both OS/MVT and OS/MVS. The record format for SIO files must have the undefined attribute (RECFM=U). Since each logical record requires at least one track of direct access storage, the use of the SIO access capability for short record transfers is not efficient.

Within the REED/RITE subroutine, the RECFM parameter of a file's JCL is interrogated by a call to the subroutine RECFM. If the file has an undefined attribute (RECFM=U) the code will use SIO access methods. Any other record format (e.g. VBS, VS, FB, etc.) is processed by standard Fortran sequential I/O. To perform the SIO data transfers, a subroutine SIO is invoked. This routine in turn invokes the subtask SIOSUB which actually performs the I/O operations. The MODE parameter which is passed to REED/RITE is used to determine whether transfer is returned to the calling routine before the I/O operation is complete. This facility, therefore, provides the user with the ability to overlap I/O and CPU operations or I/O operations on one file with those on another. The second example in Figure 13 shows an application of REED/RITE which takes advantage of the SIO access method.

The object code for this version of REED/RITE is in member REED of C116.CCCC.SYSLIB. The assembler language routines required by this subroutine (RECFM, SIO, SIOTRC) are in member SIO of C116.CCCC.SYSLIB. The module SIOSUB which is invoked to perform the asynchronous, random access I/O is in member

SIOSUB of C116.CCCC.MODLIB. The assembler source code for RECFM, SIO and SIOTRC are in member SIOASM of C116.CCCC.MASTER. The Fortran source code for this version of REED/RITE is in member REEDSIO of C116.CCCC.MASTER. The assembler source code for SIOSUB is in member SIOSUB of C116.CCCC.MASTER.

3.5 DOPC AND DRED/DRIT

The CCCC standards require that all random access I/O operations be channeled through calls to standard subroutines, not through Fortran I/O coded into applications programs. Also specified is the fact that such data should be transferred between external data files (disk) and extended core memory (ECM).

The calling sequence for DRED is:

```
CALL DRED( NREF, IREC, LOCBFU, NWDS, MODE )
```

This call transfers NWDS single-precision words from record number IREC of the the random access file with file reference number NREF to the ECM locations starting LOCBFU words from the (user) ECM reference address. A similar call to DRIT performs the inverse operations. MODE is a sentinel that permits the programmer to code asynchronous I/O. When MODE=0 operations are completed before return from DRED/DRIT. When MODE=1 I/O operations are not necessarily completed before return from DRED/DRIT; a subsequent call with MODE=2 completes the outstanding I/O operation.

Prior to calling DRED/DRIT the random access I/O implementation must be initialized by several DOPC calls. The five DOPC calling options are illustrated in Figure 14 and are summarized below:

1. (IOP=0) Initialize DOPC and, in one-level implementations, supply a pseudo ECM reference location.

2. (IOP=1) Supply file characteristics for reference number NREF.

3. (IOP=2) Conclude the definition of the file group, NGREF. NGREF includes all files defined with IOP=1 calls since either the last IOP=2 call or the original IOP=0 call.

4. (IOP=3) Delete file group NGREF and its constituent files.

5. (IOP=4) Finalize DOPC at the conclusion of the program module. All file groups are deleted.

Upon completing DOPC initialization options 1,2 and 3 (above), DRED/DRIT may be used as illustrated in Figure 15.

```
      COMMON /ARRAY/ BLK(1)
      DIMENSION NREF(3), NGREF(2), MXBLEN(3), MXBLKS(3), LENFIL(3)
      DATA I0/0/, I1/1/, I2/2/, I3/3/, I4/4/, IDUM/0/
      DATA NREF/11,12,13/, NGREF/1,2/
C
C     INITIALIZE DOPC
C     ASSUME MXBLEN, MXBLKS AND LENFIL ARRAYS HAVE BEEN INITIALIZED
C
      CALL DOPC(I0, IDUM, IDUM, NERR, BLK, IDUM, IDUM, IDUM)
C
C     DEFINE FILE GROUP 1 CONSTITUENTS
C
      CALL DOPC(I1, NREF(1), IDUM, NERR, DUM, MXLEN(1), MXBLOK(1),
     1              LENFIL(1) )
      CALL DOPC(I1, NREF(2), IDUM, NERR, DUM, MXLEN(2), MXBLOK(2),
     1              LENFIL(2) )
      CALL DOPC(I2, IDUM, NGREF(1), NERR, DUM, IDUM, IDUM, IDUM )
C
C     DEFINE FILE GROUP 2 CONSTITUENTS
C
      CALL DOPC(I1, NREF(3), IDUM, NERR, DUM, MXLEN(3), MXBLOK(3),
     1              LENFIL(3) )
      CALL DOPC(I2, IDUM, NGREF(2), NERR, DUM, IDUM, IDUM, IDUM )
      .
      .
      .
C
C     DELETE A FILE GROUP
C
      CALL DOPC(I3, IDUM, NGREF(1), IER, DUM, IDUM, IDUM, IDUM )
C
C     FINALIZE DOPC (I.E. DELETE ALL FILE GROUPS)
C
      CALL DOPC(I4, IDUM, IDUM, IER, DUM, IDUM, IDUM, IDUM )
```

Figure 14. DOPC Usage

```
      SUBROUTINE DMSMPL (LA, LB, LOCA, LOCB, NREFA, NREFB, NWDS)
      COMMON /ARRAY/ BLK(1)
C
C     LOOP OVER NRECS RECORDS IN FILE NREF. READ EACH RECORD.
C     LOCA AND LOCB ARE STARTING LOCATIONS OF ECM ARRAYS A AND B
C     OF LENGTH NWDS.  LA AND LB ARE THE STARTING LOCATIONS OF THE
C     FCM BUFFER ARRAYS IN WHICH CALCULATIONS ARE PERFORMED ON 2-LEVEL
C     MACHINES.  ON 1-LEVEL MACHINES LA AND LB DIRECTLY ADDRESS THE
C     PSEUDO-ECM ARRAYS.
C
C1LV
      LA=LOCA
      LB=LOCB
C1LV
      DO 10 IREC=1,NRECS
      CALL DRED ( NREFA, IREC, LOCA, NWDS, IO )
C2LV
C     CALL CRED ( BLK(LA), LOCA, NWDS, IER )
C2LV
C
C     COMPUTE B FROM A, THEN WRITE ARRAY B TO FILE NREFB
C
      CALL CALC( BLK(LA), BLK(LB), NWDS )
C2LV
C     CALL CRIT( BLK(LB), LOCB, NWDS, IER )
C2LV
      CALL DRIT ( NREFB, IREC, LOCB, NWDS, IO)
   10 CONTINUE
      RETURN
      END
```

Figure 15. DRED/DRIT and CRED/CRIT Usage

3.5.1 General Implementation Considerations

The connection between a random access file reference number NREF and its corresponding logical unit number is established in the ANL implementation by calling subroutine SEEK during the processing of each DOPC (IOP=1) call. The call to SEEK uses the generic random access file name RNDMnn which by convention corresponds to the random access file reference number NREF=nn. Currently NREF must satisfy 0<NREF<16. Codes which use this version of DOPC and DRED/DRIT need only supply in the SEEK initialization call those generic file names used by the applications code.

The DOPC initialization call establishes the pseudo ECM reference point for DRED and DRIT calls on one level machines. Although never explicitly specified in the CCCC standards, this pseudo ECM reference point initialization must also apply to CRED and CRIT usage. Consequently, all calls to DRED, DRIT, CRED and CRIT on one-level machines must be preceded by a DOPC initialization call. By definition in the CCCC standard, the ECM reference location on two-level machines is the first word of ECM (e.g. LCM on the CDC 7600). It should be emphasized that the ECM reference point is just that. It does not necessarily specify the starting location of ECM.

Except for the implementations on the CDC and CRAY computers, DRED/DRIT call REED/RITE to perform the random access I/O operations. Consequently, the implementation of DRED/DRIT on IBM 370 systems is simply the REED/RITE implementation discussed earlier in this section.

3.5.2 Implementation Considerations on the CDC 7600

Random access I/O on the CDC 7600 is implemented using the routines OPENMS, CLOSMS, READMS and WRITMS found in the Fortran utility library. Auxiliary storage equal in length to the number of records in the file must be supplied during the OPENMS call for each file. An auxiliary ECM container named XCM is allocated directly from DOPC by calling entry point IGTXCM in the IGTLCM dynamic storage allocation subroutine package. Consequently, DOPC and DRED/DRIT depend only on IGTXCM for dynamic storage allocation.

The subscript index limitation of 131071 words imposed on CDC 7600 LCM arrays is effectively raised to 393213 by employing two routines DRED1 and DRED2 each of which addresses a successively higher block of 131070 words of ECM. The circumvention is accomplished by passing the initial address of the next adjacent block of 131070 words of ECM to the appropriate routine, DRED1 or DRED2.

3.5.3 Implementation Considerations for the CRAY-1

The CRAY-1 implementation of DOPC and DRED/DRIT employs the random access I/O capabilities provided by the Fortran '77 standard which is implemented in the standard CRAY-1 (CFT 1.09) compiler.

3.6 CRED/CRIT

CRED and CRIT are the CCCC routines specified for data string transfer between ECM and FCM. The calling sequence for CRED is:

```
CALL CRED ( FCM(I), LECM, NWDS, IER )
```

This call transfers NWDS single precision words starting from ECM location LECM to the FCM locations starting at the address of FCM(I). IER is an error sentinel. A similar call to CRIT performs the inverse operation. The example in Figure 15 illustrates the use of CRED/CRIT.

The implementation of CRED/CRIT on the CDC 7600 employs the COMPASS assembly language routines WRITEC and READEC to perform the actual data transfers between ECM and FCM. The three arguments in the calling sequence for WRITEC are identical in type to the first three arguments in the CRIT calling sequence (e.g. CALL WRITEC (FCM, LECM-1, NWDS)).

The second parameter LECM in CRIT denotes an ECM location relative to the ECM reference array, while the second parameter supplied in the WRITEC call denotes the corresponding ECM address (e.g. LECM-1). On one-level implementations CRED and CRIT simply transfer data between FCM and pseudo ECM locations both of which reside in the same memory level. The transfers are performed via standard Fortran assignment statements.

As noted in the DOPC and DRED/DRIT section, CRED/CRIT are interlocked with DOPC to ensure that the ECM container reference address pointers have been initialized by DOPC.

The Fortran source code for CRED/CRIT is in member CRED of C116.CCCC.MASTER.

3.7 ECMV

ECMV is the CCCC routine specified for transfering data strings between locations in ECM on two-level machines (e.g. CDC 7600). Transfers are performed using CRED and CRIT which route data through a 64-word FCM buffer array local to ECMV. This approach circumvents the compiler restriction limiting array sizes to 131071 words on the CDC 7600. The calling sequence for ECMV is:

```
CALL ECMV ( LECM1, LECM2, NWDS )
```

This call transfers NWDS single precision words starting from ECM location LECM2 to the ECM location starting at LECM1.

3.8 LRED/LRIT

LRED and LRIT are the CCCC routines specified for executing binary sequential I/O operations that transfer data between external data files (disk) and ECM. The calling sequence for LRED is:

```
  CALL LRED ( NREF, IREC, ECM(I), NWDS, MODE )
```

This call transfers NWDS single-precision words from record number IREC of the sequential file with reference number NREF to the ECM locations starting at the address of ECM(I). A similar call to LRIT performs the inverse operations. MODE is a sentinel that permits programmers to code buffered I/O as explained in the REED/RITE description in a previous section. The only difference between LRED/LRIT and the export version of REED/RITE is the memory level within which the data array to be transferred is located.

Because LRED/LRIT and REED/RITE can access the same file, a common initialization subroutine (ZEROIO) is used.

The Fortran source code for LRED/LRIT is located in member LRED of C116.CCCC.MASTER.

4. BCD INPUT CONVENTIONS

The input to major production codes usually consists of both BCD card input and disk files. The disk files are usually libraries (e.g. cross section data) or restart files and are created by other jobs. The BCD card input is prepared by the user and, for complicated models, may run to thousands of cards containing a variety of types of data.

Applied Physics production codes use input preprocessing routines to break up a single, BCD card input file into several smaller BCD disk files. These, in turn, are then read by different applications load modules. These preprocessing routines also provide a convenient method of executing a number of similar calculations in a single job with a minimum of input. This chapter discusses input conventions and input preprocessing - first from the standpoint of the user, and then from the standpoint of the programmer.

We will use the term "ARC System" to identify these input conventions since they were largely set for the original ARC System.[1] They have evolved with the new generation of CCCC[2] codes, however, and are completely exportable.

4.1 USER CONSIDERATIONS

In the ARC System input convention the input BCD card images are grouped into "BLOCKs", and the card images within each BLOCK are grouped into "DATASETs". BLOCKs and DATASETs are identified by cards containing one of the following phrases:

```
BLOCK=blknam
DATASET=dsname
SUBLOCK=dsname
UNFORM=dsname
NOSORT=dsname
MODIFY=dsname
REMOVE=dsname
```

The words to the left of the "=" sign are "keywords"; the words to the right of the "=" sign are BLOCK or DATASET names. Keywords must start in column 1 of the card, and there can be no imbedded blanks.

Both binary and BCD files are given names and, following the CCCC conventions,[2] version numbers. Binary files include CCCC standard interface files, code dependent interface files for passing data between load modules, and scratch files used only within particular load modules. BCD files for ARC System codes are usually given names starting with "A."; for example A.NIP3 is the BCD file which defines the neutronics input geometry and isotopic number density data for most of our new codes.

The number of versions permitted for a particular file is established by individual programs. In references in the BCD card input to one of several versions of a file, the version number must follow the file name and be separated from it by a comma. Taking examples from the list of keyword phrases above:

```
DATASET=A.SAMPLE,2
REMOVE=RTFLUX,1
```

A version number of unity is implied when no version number is given. Thus, the second example above could have been written:

```
REMOVE=RTFLUX
```

Most files for most programs are permitted only one version, and so version numbers are required only occasionally.

4.1.1 BLOCK=

The BCD input stream is divided into BLOCKs, and each BLOCK starts with a card containing

```
BLOCK=blknam     .
```

"blknam" may be the word "OLD" or a name appropriate to the particular code being executed. The special case of BLOCK=OLD is discussed in a later section. A BLOCK ends at the last card before the next BLOCK= card or at the end of the card input file, whichever is encountered first.

As far as the user is concerned, the phrase BLOCK=blknam causes the execution of a particular sequence of load modules. If the phrase occurs twice in the input stream the sequence is executed twice. Only data contained in the first BLOCK are available to the program during the first execution. Data in the second BLOCK modify or replace the first BLOCK data for the second execution.

The name of a BLOCK is a convention set by the individual applications program. It may, for example, be the name of the program itself.

BLOCKs with the same name are processed in the order in which they appear in the input. BLOCKs with different names may be placed in any arbitrary order; it will make no difference to the execution of the program.

4.1.2 DATASET=, UNFORM=, SUBLOCK=, NOSORT=

The data within each BLOCK are subdivided into DATASETs, and each DATASET starts with a card containing one of the phrases:

```
DATASET=dsname
SUBLOCK=dsname
UNFORM=dsname
NOSORT=dsname
```

A DATASET ends at the last card before the next keyword or at the end of the card input file, whichever comes first. The order of dissimilarly named DATASETs within a single BLOCK makes no difference to the execution of the program.

DATASETs designated DATASET=, SUBLOCK= or UNFORM= are expected to contain cards on which the first two columns contain either

1. a positive, 2-digit, nonzero "card type number," or

2. blanks, zeros or non-numeric characters.

The type numbers (01, 02, ... 99) are used in ARC System input to identify the type of data on each card. For example, in the BCD input file named A.NIP3 mesh data are supplied on "type 09 cards" (cards that have "09" punched in columns 1-2).

Before individual load modules read cards in a particular DATASET specified by DATASET=, SUBLOCK= or UNFORM=, the cards with card type numbers are rearranged in order of ascending card type number. When more than one card of a particular card type is present the relative order of those similarly numbered cards is unchanged.

At the same time as numbered cards are reordered, unnumbered cards (those with blanks, zeros or non-numeric characters in cols. 1-2) are repositioned after all numbered cards. Some users use unnumbered cards as "comment cards" to annotate their decks of numbered cards; before the data are read by applications load modules the unnumbered comment cards are swept to the back of the DATASET where they may not be seen by the load module. A listing of the input deck before sorting is printed on the user's output medium so that the comments are available for documentation.

DATASETs designated NOSORT= are not reordered in any way. NOSORT DATASETs are normally used for data required by a load module which was written at another installation but which was incorporated as a load module in an Applied Physics production code.

Applications programs reading data specified by DATASET= or SUBLOCK= expect the data cards to be formatted; the cards will be read using formatted, Fortran I/O. Programs reading data specified by UNFORM= will process unformatted data (in cols. 3-72; cols. 1-2 are still reserved for card type numbers). The conventions for unformatted data are given in the section on the free-format routine FFORM. The user should be aware that some applications programs may not permit free-format input. The format rules for NOSORT DATASETs depend on the individual load modules which read them.

When BLOCK=blknam appears twice in the input - specifying two executions of the same sequence of load modules - DATASETS in the first BLOCK are automatically preserved for the second execution unless the user deliberatly redefines a DATASET in the second BLOCK. For example,

```
BLOCK=TEST
DATASET=A.SAMPLE
01    data ...
02    .........
07    .........
BLOCK=TEST
DATASET=A.SAMPLE
01    new data
02    .........
06    .........
```

The first DATASET is entirely replaced by the second before the second execution. A later section discusses how one can make selective changes to DATASETs.

4.1.3 BLOCK=OLD

The special BLOCK "OLD" permits the user to tell a program which files already exist on disk and are being input to the calculation. Input disk files must be listed under BLOCK=OLD in the following manner:

```
BLOCK=OLD
DATASET=dsname
DATASET=dsname
    etc.
```

BLOCK=OLD may be placed anywhere in the BCD card input file.

These BLOCK=OLD files are usually binary library or restart files. Occasionally it may be convenient to create and save a BCD file in one job and then pass it to a second job on disk rather than in the BCD card input file. In such a situation the DATASET name should appear under BLOCK=OLD in the second job and not in any other BLOCK processed by the second job.

4.1.4 MODIFY=, REMOVE=, nn=DELETE

MODIFY=dsname permits the user to replace cards of a particular card type in an old DATASET without affecting the rest of the data. Type numbered cards following MODIFY=dsname replace the cards of that type (or those types) in a previously defined DATASET. For example, if a DATASET in one BLOCK contains seven type 09 cards and five new type 09 cards are provided in a second BLOCK under MODIFY=dsname, then the seven original cards are deleted and the five new cards substituted before the second execution.

It is not necessary for a DATASET to be defined in one BLOCK and MODIFYed in another. Some users like to define a reference DATASET with DATASET=dsname and then make changes in the same BLOCK before execution with MODIFY=dsname. Subsequent use of MODIFY will operate on the most recent version of the DATASET.

REMOVE=dsname deletes the entire DATASET. This option is frequently used with binary data sets to force applications load modules, for one reason or another, to rewrite a data set.

nn=DELETE, where nn is a card type number, after a MODIFY=dsname will cause all of the type nn cards to be deleted from the DATASET.

4.1.5 Sample Input

Figure 16 shows a BCD card input file for a fictitious program. The input is designed to exercise most of the options described above. Three input DATASETs are defined; they are two separate versions of a file named A.SAMPLE and one named A.XAMPLE. Below the listing of the input, Figure 16 shows the contents of each file after preprocessing and before the imaginary program is executed. There are two BLOCKs (i.e. two executions in the job).

MODIFY= is used in the first BLOCK to modify a DATASET defined in the same BLOCK (A.SAMPLE,2). It is used in the second BLOCK to modify a DATASET defined in the first BLOCK (A.SAMPLE,1).

Note that A.SAMPLE,1 and A.SAMPLE,2 are defined with DATASET= and UNFORM=; the cards are reordered according to card type with unnumbered cards placed last. A.XAMPLE is defined with NOSORT= and is unaffected by the preprocessing.

```
BLOCK=TEST
DATASET=A.SAMPLE
    A.SAMPLE,1
09  09 CARD
05  1ST 05 CARD
 5  2ND 05 CARD
    UNNUMBERED
07  07 CARD
5   3RD 05 CARD
UNFORM=A.SAMPLE,2
XX  A.SAMPLE
XX  VERSION 2
08  08 CARD
MODIFY=A.SAMPLE,2
08  REPLACE 08
BLOCK=TEST
MODIFY=A.SAMPLE,1
09=DELETE
05  REPLACE 05
REMOVE=A.SAMPLE,2
NOSORT=A.XAMPLE
    A.XAMPLE
    NOSORT
07  TYPE 07 CARD
    NO NUMBER
07  ANOTHER 07
```

To the left is a sample input file illustrating many of the input processing options. There are two BLOCKs and three DATASETs referenced.

Contents of each of the three DATASETs after the first BLOCK is processed.

```
    A.SAMPLE             A.SAMPLE
   version 1            version 2           A.XAMPLE

 05  1ST 05 CARD     08  REPLACE 08     not defined in
  5  2ND 05 CARD     XX  A.SAMPLE       the first BLOCK
 5   3RD 05 CARD     XX  VERSION 2
 07  07 CARD
 09  09 CARD
     A.SAMPLE,1
     UNNUMBERED
```

Contents of each of the three DATASETs after the second BLOCK is processed.

```
 05  REPLACE 05      not defined in          A.XAMPLE
 07  07 CARD         the second BLOCK        NOSORT
     A.SAMPLE,1                         07   TYPE 07 CARD
     UNNUMBERED                              NO NUMBER
                                        07   ANOTHER 07
```

Figure 16. Illustration of Input Conventions

4.1.6 Output from BCD Input Card Preprocessors

The BCD input preprocessing routines normally produce two kinds of edits. At the beginning of the job the user's input is listed on both the regular and auxiliary output print files by the routine SCAN. In addition, all data sets processed under each BLOCK=blknam are edited by the routine STUFF. Users have control over the STUFF edits for each BLOCK through an integer sentinel, n, that can be added to the BLOCK= card:

```
BLOCK=blknam,n
```

```
n = 0, edits given on both regular and
       auxiliary output files (default).
  = 1, edits on regular output file only.
  = 2, edits on auxiliary output file only.
  = 3, no edits for the current BLOCK.
```

4.2 GENERAL PHILOSOPHY ON INPUT DATA

A number of principles have guided the design of BCD card input for ARC System codes. We have continued to observe them in the development of CCCC codes.

1. Data that are not essential to the problem should not be required in the BCD card input. In particular, no redundant data should be required.

2. Card input files should be easy to create and to modify.

3. Whenever possible, labels and names should be used instead of numbers for descriptive data.

The BCD input conventions defined in the previous section support these principles.

The convention of numbered cards containing very specific types of data helps to eliminate nonessential and redundant data. The preprocessing routines pass to the applications programs the number of cards of each type contained in a particular DATASET. The user never has to tell a code how many data of a particular type are input; the code can figure it out by itself. Default values can be provided not only when a particular datum is missing, but also when whole card types are missing.

About forty different card types are defined for the geometry and isotope number density file A.NIP3. Rarely are more than a dozen used for a particular job. Instead of user supplied sentinels, the presence or absence of particular card types signals options. Explicit sentinels would be redundant.

Numbered cards and the free-format option make it relatively easy to create and modify DATASETs. Long strings of input data and tables are convenient to the programmer but not to the user. ARC System input has always tended towards requiring only a few pieces of data per card, with the format of the card designed for the convenience of the user. In some cases cards of a particular type may be shuffled without affecting the definition of a problem. In other cases the order of cards has significance; the data on one card may overlay, in some way, data defined on a previous card. Modifications to input can frequently be made simply by adding or changing the order of cards; no changes to existing cards are required.

If nothing else, the use of labels instead of numbers for input quantities makes the BCD card input file more readable to users. In the A.NIP3 DATASET compositions and geometric regions are given labels, and isotopes are referred to by name.

4.3 PROGRAMMING CONSIDERATIONS

The two subroutines which preprocess the BCD card input file are SCAN and STUFF. This section discusses their use in a program.

4.3.1 SCAN and STUFF Input Preprocessors

SCAN must be called before any BCD input is read and before the first call to STUFF; it is called only once in a job. The initialization call to SEEK must precede the call to SCAN. SCAN reads the entire BCD card input file from logical unit NIN (NIN is the first variable in the labeled common block /IOPUT/) and copies it to another file which is either the file named ARC or, if ARC is not in the SEEK tables, logical unit 9. In the process it sets up a table of pointers to the beginning of each BLOCK. The call to SCAN also processes the data in BLOCK=OLD if it is present in the input file.

All BLOCKs other than BLOCK=OLD are processed by calls to STUFF. Before each call to STUFF the variable STFNAM (the first variable in the labeled common block /STFARC/) must be set equal to the name of the BLOCK to be processed. STUFF returns a flag (NRET in /STFARC/) which permits the program to test for end of input. STUFF writes, or rewrites, each BCD disk file referenced under the particular BLOCK=STFNAM according to the instructions in the input (DATASET=, MODIFY=, etc.). It is STUFF that reorders, replaces and deletes numbered cards. Since the STUFF processing follows the processing of BLOCK=OLD by SCAN, a new DATASET input on cards would destroy the data already in an existing file of the same name referenced under BLOCK=OLD.

Figure 17 shows a simple driver that uses SCAN and STUFF to preprocess BCD card input. In fact, this driver could be used with the input shown in Figure 16 since the BLOCK and DATASET names are consistent. The driver starts by setting card and printer file numbers and by initializing SEEK, TIMER and LINES. Following the single call to SCAN it goes into a loop containing a call to STUFF and an execution of a program (PROG). Execution terminates when the last BLOCK=TEST has been processed. Figure 17 is a simplified, but otherwise typical, driver; a calculation is performed for each BLOCK in the input.

Additional details about SCAN and STUFF, and their use, may be found in the writeups of ARCBCD, SCANARC, STUFFARC, SYS001 and SYS002 in Appendix A.

The source code for SCAN and STUFF is in member ARCBCD of C116.CCCC.MASTER; the object code is in member ARCBCD of C116.CCCC.SEGLIB. In a modular environment the module interface subroutines are in members SCAN and STUFF of C116.CCCC.SYSLIB.

```
CSW
      IMPLICIT REAL*8(A-H,O-Z)
CSW
      COMMON /PTITLE /TITLE(66), TIME(10), HNAME(4), KOUT, KOUT2, NTITLE
      COMMON / IOPUT / NIN, NOUT, NOUT2
      COMMON / STFARC / STFNAM, BLKNAM(50), IBLTAB(3,50), NBLOCK, NRET
      DIMENSION DSNAME(6)
      DATA DSNAME / 8HA.SAMPLE, 8HA.SAMPLE, 8HA.XAMPLE, 6HRTFLUX,
     1 6HISOTXS, 1H$ /
      DATA BLOCK/4HTEST/, BLANK/6H      /
      DATA IM1/-1/, I0/0/, I1/1/, I3/3/, I4/4/, I11/11/
C
      NIN=5
      NOUT=6
      KOUT=NOUT
      NOUT2=I0
      KOUT2=NOUT2
      NTITLE=0
      CALL FLTSET(TITLE,BLANK,I11)
      CALL FLTSET(HNAME,BLANK,I4)
C
      N=0
      CALL SEEK(DSNAME,I1,N,I3)
      CALL TIMER(I0,TIME)
      CALL TIMER(IM1,TIME)
      CALL LINES(I0,I1)
      CALL SCAN
C
      STFNAM=BLOCK
   10 CONTINUE
      CALL STUFF
      IF( NRET.LE.0 ) GO TO 20
      CALL PROG
      GO TO 10
C
   20 CONTINUE
      RETURN
      END
```

Figure 17. An Example of the Use of SCAN and STUFF

4.3.2 Setting Logical Unit Numbers for BCD Files Via SEKPHL

The ANL convention of using more than one BCD file is not covered in the CCCC rules, and it caused problems when codes were exported to Los Alamos. It is not necessarily true that the file reference number for a BCD file returned by SEEK is identical to the logical unit number used in Fortran I/O coding. The relationship between the two numbers is always hidden behind subroutine calls (SEEK, REED, etc.) for binary files.

Because of the potential for a problem a subroutine, SEKPHL, is used to return a logical unit number given a SEEK file reference number. SEKPHL is also used to rewind (and close if required) BCD files, a function REED and RITE perform for binary files.

The calling sequence for SEKPHL is defined in Appendix A. An example of its use with SEEK was given in the chapter entitled 'CCCC Standard Subroutines". The Argonne export SEKPHL simply sets the logical unit equal to the file reference number obtained from a call to SEEK. Other installations (e.g. LANL) may use a different correspondence. Figure 11 illustrates SEKPHL usage.

4.3.3 Reading BCD Files Preprocessed by STUFF

As it writes or rewrites a BCD card image file STUFF inserts a few formatted records at the beginning which contain information about the structure of the file. These lead records may be read:

```
      READ(M,99)ANAME,MAXTYP,NONUM,NOFORM,(N(I),I=1,MAXTYP)
   99 FORMAT(A8,3I5/(16I5))
```

M	file logical unit number
ANAME	file name
MAXTYP	highest card type number in file
NONUM	number of unnumbered cards
NOFORM	0/1, cards are to be read formatted/free-format
N(I)	number of type I cards in the file

The logical unit number, M, should be obtained through calls to SEEK and SEKPHL:

```
      CALL SEEK(ANAME,IVER,I,0)
      CALL SEKPHL(I,M,0)
```

IVER	file version number
I	file reference number

There are always at least two of these lead records in a BCD file written by STUFF; Fortran I/O expects a second record even if MAXTYP=0. Indeed, for NOSORT DATASETs MAXTYP is zero.

The NOFORM sentinel is 0 for files designated DATASET= or SUBLOCK=; it is 1 for files designated UNFORM=.

These lead records permit applications modules reading BCD files to make decisions based on the presence or absence of particular card types. The user's input card images follow the lead records and can be read as if they were cards - one 80-column card per record.

5. BPOINTER, A DYNAMIC STORAGE ALLOCATION PROGRAM

Small, single-purpose codes can afford to be programmed with fixed-dimension arrays for storage. Such a practice imposes rigid limits on the size of a problem a code can solve, but those limits can be made large or can be changed by rewriting the DIMENSION statements.

In the case of large production codes the problem size limitations imposed by fixed-dimension arrays may be intolerable. Running small problems on codes designed for large ones can be needlessly expensive. Code changes may be awkward and, from a quality assurance standpoint, risky. Most large codes, therefore, use some sort of dynamic storage allocation system to manage the core storage of data during execution. Core storage is reserved for a particular dimensioned array only during the time the corresponding data are required to be in-core; at other times the space is made available for the storage of other data.

The ARC System dynamic storage allocation routines are contained in the BPOINTER package.[1] BPOINTER is a collection of subprograms which was developed to alleviate bookkeeping chores associated with the use of dynamic storage allocation techniques. These chores are separated into two functional categories:

1. The highly machine-dependent functions of obtaining/releasing large blocks of workspace called "containers" from/to the operating system.

2. The largely machine-independent bookkeeping functions associated with managing array allocations within a given container

Category 1 tasks are performed by the IGTLCM package, a self-contained set of subroutines that may be used independently of BPOINTER. Consequently, in situations requiring only the IGTLCM functions (e.g. dynamic container allocation for codes imported by Argonne), inclusion of the BPOINTER routines is unnecessary.

5.1 USER CONSIDERATIONS

The user needs to know nothing about the BPOINTER routines themselves, only that they require one or two large blocks of workspace called "containers" for data storage during the execution of a job. The container sizes are usually input quantities, and the choice of sizes is entirely code and problem dependent.

The first container, the FCM (fast-core memory) container, is always in fast memory. Fast memory is called SCM (small-core memory) on the CDC 7600 computer. The second, the ECM (extended-core memory) container, is in LCM (large-core memory) on CDC 7600 machines and in fast memory on other, one-level computers. The pseudo-ECM container used on one-level machines unifies the code from an applications and programming standpoint.

The terms SCM and LCM are usually used only in the context of CDC 7600 systems. We will occasionally use them interchangeably with their generic counterparts FCM and ECM, respectively.

In IBM/370 systems the user must be sure that the job or step REGION size is large enough to accomodate the program, file buffers and containers. BPOINTER will print an error message if the requested containers are too large for the REGION.

On the CDC 7600 system the user should request field lengths only large enough to execute the code without the BPOINTER containers. BPOINTER requests additional memory for the containers (i.e. increases the field length during execution) and returns an error flag if that space cannot be allocated.

On the CRAY system the user should request a field length in the JOB control statement large enough to accomodate the program, file buffers and containers. BPOINTER will print an error message if the requested containers cannot be allocated. Most programs that use BPOINTER return the field lengths to their original values upon normal termination.

5.2 PROGRAMMING CONSIDERATIONS

Programs which use the BPOINTER capability tend to be structured in subroutine form. A control routine is used to define one or two blocks of storage (the containers) and to make the appropriate calls to BPOINTER to control the allocation of space within these containers. Calls to calculational subroutines transmit pointers corresponding to array locations through the calling sequences. All BPOINTER capabilities are accessed through an appropriate call to an entry point, subroutine or function subprogram. The following capabilities are available in the BPOINTER system:

1. Storage of data in and retrieval of data from the container array via user defined variable arrays.

2. Purge of variable arrays stored in the container array.

3. "Cleanup" of the container array when more storage is required (to avoid fragmentation).

4. Redefinition of array sizes without loss of data already stored in the array.

5. Dump of selected integer, floating point or Hollerith arrays in an appropriate format.

6. Trace edits of BPOINTER activities.

7. Status reports of the BPOINTER tables.

Detailed program documentation including flow charts, common block information and subprogram descriptions is available in Reference 1. A shorter, functional writeup is included in Appendix A of this report (member POINTR) and gives calling sequences for the BPOINTER routines. This section is intended to provide a brief description of how the program package operates.

The short example shown in Figure 18 illustrates the structure of a program using the BPOINTER package. This example demonstrates the manner in which a container is allocated, pointers defined and used, and the container released.

The letters M and B are used as mnemonics within BPOINTER to designate routines which operate on the FCM and ECM containers, respectively. Thus PUTM allocates an array in the FCM container while PUTB allocates an array which must be referenced on a CDC 7600 as either a LEVEL 2 or a LEVEL 3 array. According to CCCC conventions,[2] arrays allocated in ECM are referenced through the standard subroutines CRED/CRIT and DRED/DRIT in exportable source code intended for two-level computers.

On IBM equipment without HIARCHY support (e.g. the 370/195) the two containers are both in fast core. The distinctions noted above between the two dynamic containers are important on the CDC 7600 where the containers are addressed quite differently and on IBM equipment with HIARCHY support where access to the LCM container (HIARCHY 1, subpool 1) is significantly slower than access to the MAIN core container (HIARCHY 0, subpool 2).

In the example all dynamically allocated FCM arrays are addressed relative to the labeled common block /ARRAY/ which contains a single array element, BLK(1). In the short-word version of the code the element must be declared REAL*8. In the two-level (CDC 7600) version of BPOINTER the ECM container is addressed relative to the first word of LCM. The pseudo ECM container on IBM equipment is a second container which may be given a HIARCHY 1 location but is addressed in precisely the same manner as the first (FCM) container. The one word assigned to the container by the applications program provides a reference address. At execution time the function routines IGTLCM and IGTSCM are used to obtain the addresses of core which are available to the program for the allocation of data arrays.

A few codes at the same time use BPOINTER and directly address ECM on two-level machines. In these programs the "LEVEL 2" BPOINTER reference common block must start at the first word of LCM. BPOINTER calculates address offsets based on that assumption. Most codes currently under development do not address ECM directly; they employ CRED and CRIT to transfer blocks of data between the two levels of memory.

Occasionally we find it convenient to exercise the two-level implementation on a one-level machine. In such cases it is necessary to precede the BPOINTER initialization call by the DOPC initialization call so that the user reference address of the BPOINTER ECM container is initialized prior to the first call to CRED/CRIT (BPOINTER employs CRED/CRIT in its two-level implementation). A discussion of IGTLCM/IGTSCM and the associated assembler routines that allocate these blocks of core follows.

```
CSW
      IMPLICIT REAL*8(A-H,O-Z)
      REAL*4 BLK4
CSW
      COMMON/ARRAY/BLK(1)
      COMMON/IOPUT/NIN,NOUT,NOUT2
      DIMENSION BLK4(1)
      EQUIVALENCE (BLK(1),BLK4(1))
      DATA FLUX/4HFLUX/, POWER/5HPOWER/, MAXSIZ/10000/, NG/27/,
     1 I4/4/, I8/8/, IO/0/
      NOUT=6
C  ALLOCATE CONTAINER WITH MAXSIZ WORDS OF FCM AND NO ECM.
      CALL BULK(IO)
      CALL POINTR(BLK,MAXSIZ,IO)
C  ALLOCATE SPACE FOR ARRAYS POWER AND FLUX.  DETERMINE THE
C  POINTER FOR SUB-ARRAY CURRENT WHICH FOLLOWS THE FIRST NG
C  SINGLE-PRECISION WORDS FOR THE ARRAY FLUX.  THEN CHECK FOR
C  A BPOINTER ERROR.
      CALL PUTM(POWER,NG,I8,IPOWR)
      CALL PUTM(FLUX,2*NG,I4,IFLUX)
      ICURNT=IPT2(IFLUX,NG,IO)
      IF( IPTERR(DUMMY).LE.0 ) GO TO 10
      PRINT 500
  500 FORMAT(15H0BPOINTER ERROR)
      STOP
   10 CONTINUE
C  CALL SUBROUTINE INIT TO USE THESE ARRAYS.  THEN FREE THE
C  CONTAINER AND STOP.
      CALL INIT(BLK(IFLUX),BLK(IPOWR),BLK4(ICURNT),NG)
      CALL FREE
      STOP
      END
C
C-------------------------------------------------------------
C
      SUBROUTINE INIT(PHI,POWER,CURENT,NG)
CSW
      REAL*8 POWER
CSW
      DIMENSION PHI(1), POWER(1), CURENT(1)
      DO 10 I=1,NG
      PHI(I)=1.0
      POWER(I)=3.1E+06
      CURENT(I)=.333
   10 CONTINUE
      RETURN
      END
```

Figure 18. A BPOINTER Example

5.2.1 IGTLCM/IGTSCM/IGTXCM

Function IGTLCM and its associated entry points IGTSCM and IGTXCM manage the allocation of the ECM, FCM and XCM containers, respectively. The FCM container always resides in fast memory (e.g. the FCM storage pool). The ECM container and the auxiliary ECM container named XCM both reside in the same storage pool. On single-level machines they reside in the FCM storage pool along with the FCM container; on two-level machines like the CDC 7600 they reside in LCM. If the C1LV language flag is activated in a CDC 7600 implementation then the ECM and XCM containers will be allocated in the FCM storage pool along with the FCM container.

IGTLCM, IGTSCM and IGTXCM route all memory allocation requests through function subroutine JGT which calls the appropriate assembler, Fortran or system routines. The calling sequence for IGTLCM is:

```
LOCECM = IGTLCM( NWORDS )
```

This call returns the (REAL*4) word address of the requested block of NWORDS which constitutes the ECM container. A similar call to IGTSCM or IGTXCM allocates the appropriate container. The function value of -1 is returned if the container allocation fails. Subroutine FRELCM with associated entry points FRESCM and FREXCM release the corresponding containers. The example in Figure 19 illustrates the use of the IGTLCM package. The function LOCFWD provides the (REAL*4) word address of the reference variable used to address the container.

```
      COMMON /REFFCM/ BLK(1)
C
C     ALLOCATE FCM CONTAINER
C
      LOCFCM = IGTSCM( NWORDS )
      IF ( LOCFCM .EQ. -1 ) GO TO 10
C
C     DETERMINE WORD OFFSET OF CONTAINER FROM REFERENCE ARRAY BLK(1)
C
      LFCREF = LOCFCM - LOCFWD ( BLK(1) )
C
C     INITIALIZE CONTAINER
C
      DO 1 I=1,NWRDS
      BLK(LFCREF+I)=0.0
    1 CONTINUE
      .
      .
      .
C
C     FREE CONTAINER
C
      CALL FRESCM
      .
      o
      .
C
C     ERROR EXIT
C
   10 CONTINUE
      .
      .
```

Figure 19. IGTSCM/FRESCM/LOCFWD Example

5.2.2 IBM Allocation

The assembler routine MYLCM with entry point MYSCM, FREELC and FREESC (called by JGT or FRELCM) uses the standard IBM macro instructions GETMAIN and FREEMAIN to allocate and free consecutive words of core which are available to the program. The designations subpool 1 and 2 are assigned to the ECM and FCM containers, respectively. Since allocations are performed in units of 256 (eight byte) words, it is most efficient to request blocks of core in such multiples.

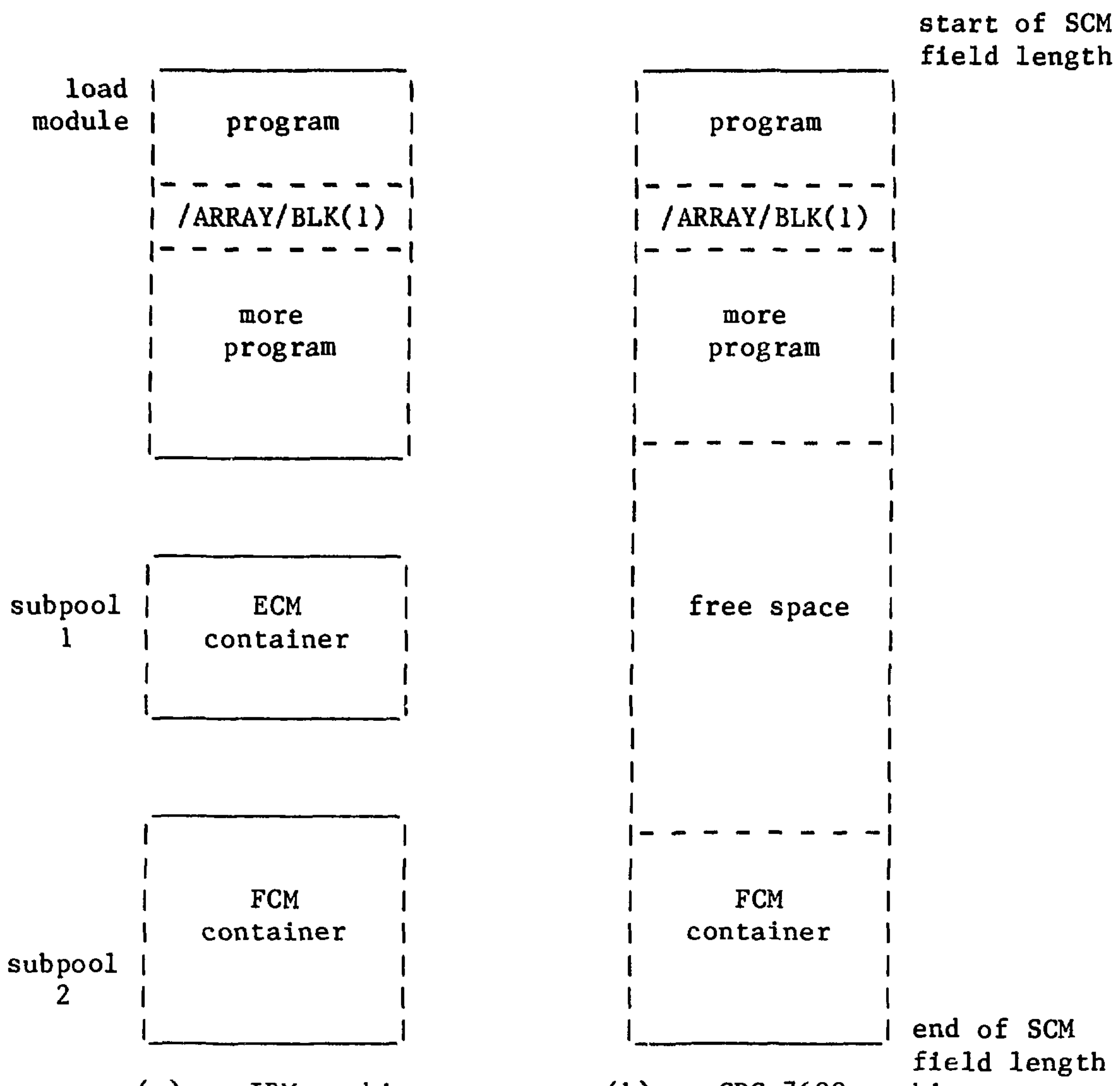
Figure 20 shows a schematic diagram of a program and SCM container.

5.2.3 CDC Allocation

The COMPASS assembler routine JGTSCM with entry point JGTLCM (called by JGT) uses the standard CDC macro instruction MEMORY to determine and to change the job's SCM and LCM field lengths.

The FCM container is placed at the end of the user's SCM field length, as shown in Figure 20. The ECM container is placed at the end of the user's LCM field. The last word of each container is four words short of the user's SCM or LCM field length; this is done to avoid I/O problems in systems that attempt to read ahead.

BPOINTER releases containers when they are no longer needed by returning field lengths to their original values.



(a) on IBM machines (b) on CDC 7600 machines

Figure 20. Fast-Core Allocation on IBM and CDC 7600 Machines

5.2.4 CRAY Allocation (CTSS)

Two subroutines LASTMEM and MEMORY from the CRAY Time Sharing System (CTSS) Fortran Library at Los Alamos National Laboratory are called by JGT to determine and change a job's field length, respectively. JGT establishes the user program length (i.e. the high limit of user code, JCHLM) by an initial call to LASTMEM. Each time a new container is requested JGT allocates space in one of two ways:

1. If another container has been previously allocated, and there is enough free space between it and the program, the new container is established in the free space. The field length is not changed.

2. If adequate free space is not available MEMORY is called to increase the field length, and the new container is placed such that the address of its last word is the new value of JCHLM.

Figure 21 shows a schematic diagram of fast core of a CRAY machine containing a program and two containers.

JGT reduces the field length by an appropriate amount when the container ending at address JCHLM is released.

```
                      start of FCM
       _____________  field length
      |             |
      |   program   |
      |             |
      |- - - - - - -|
      | /ARRAY/BLK(1) |
      |- - - - - - -|
      |             |
      |    more     |
      |   program   |
      |             |
      |             |
      |- - - - - - -|
      |             |
      |             |
      |     FCM     |
      |  container  |
      |             |
      |             |
      |- - - - - - -|
      |             |
      |             |
      |             |
      |     ECM     |
      |  container  |
      |             |
      |             |
JCHLM |- - - - - - -|
      |             |
      |system tables|
      | and buffers |
      |             |
      |_____________|
```

Figure 21. Fast-Core Allocation on a CRAY Machine

5.2.5 CRAY Allocation (COS)

Dynamic memory allocation on machines using the standard CRAY Operating System (COS) is implemented in a manner that is functionally equivalent to the CTSS implementation. CTSS subroutines MEMORY (2 arguments) and LASTMEM are simulated on COS installations by the Fortran subroutine MEMGET and its entry point LASTMEM, respectively. A blank common array of length 1 must be located as follows in order for it to provide a reference point (JCHLM) for the dynamic memory allocation:

1. Non-overlayed COS systems - place it after all object code (the CFT compiler does this by default).

2. Segmented loading on COS system - assign it to a second memory level above all overlays.

3. COS overlay loading types 1 or 2 - assign it (via the SBCA overlay directive) to a specified address larger than any address used in the overlay structure (this number is installation dependent and must be determined upon completion of loading.

Subroutine MEMGET calls the COS system routine MEMORY (5 arguments) which issues calls to the CAL assembler MEMORY macro to increment or decrement JCHLM, the length of the user code area. A corresponding field length change occurs simultaneously.

6. FFORM, A FREE-FORMAT INPUT ROUTINE

Subroutine FFORM enables programmers to implement free-format input in a relatively simple manner. The design of the routine and its usage were dictated by the following goals and constraints:

1. The syntax had to be consistent with ARC System data set descriptions. ARC System input data sets consist of sets of type-numbered cards with an arbitrary number of cards allowed of each type. Individual cards contain a limited number of data, and the data may be a mix of fixed point numbers, floating point numbers and literals.

2. The applications should not be restricted to ARC System input.

3. Implementation should not force an abrupt conversion from formatted input to free-format, if only in order not to obsolete existing, formatted input decks. This argues for a scheme that works in parallel with conventional, formatted Fortran READ's. Free-format input can then be coded into programs without disturbing the Fortran READ's already there.

4. All coding must be exportable. This requirement precludes the use of IBM list-directed READ's. It encourages the development of a Fortran callable subroutine.

5. Whatever syntax is available must be consistent with CCCC (Committee on Computer Code Coordination) standards.

The convenience of free-format input is not without some cost. On a formatted card a blank field is interpreted as a zero. Users cannot leave a blank on a free-format card, however, and expect the input processor to interpret it as a blank or a zero (zeroes and blanks must be specified explicitly). Users who are new to free-format input processors will find they make this mistake a few times as they adapt to the system.

6.1 FFORM Calling Sequence

CALL FFORM(DATA,NDATA,MAXDTA,NFL,IFLAG,NOUT)

DATA	an array (REAL*8 on IBM machines) into which FFORM is to place the data found on a card.
NDATA	the number of words of data encountered by FFORM on the card.
MAXDTA	the length of the array DATA (REAL*8 words on IBM machines). If NDATA is greater than MAXDTA the extra words are not returned. (input)
NFL	the logical unit number of the BCD file to be read. (input)
IFLAG	error flag returned by FFORM. 0, an end of file was encountered. 1, a card was read and data transferred successfully. -1, an input error (an unrecognized format) was encountered.

NOUT the logical unit number for the output error messages. If NOUT=0 no messages are printed. (input)

Each call to FFORM causes a single card image record (the next record) to be read from logical unit NFL. FFORM scans the first 72 columns of the card image, interprets the data and translates the BCD representation into the appropriate variable type. The user must provide an array, DATA in the calling sequence shown above, into which the data on the card are to be placed. If a card contains more data than there is space for (i.e. if, on return, NDATA is greater than MAXDTA) only MAXDTA words are transfered to the DATA array. Hollerith data are stored six characters to the word.

On IBM machines the word length difference between INTEGER*4 and REAL*8 variables causes a small problem. If the Nth datum on an input card is an integer, FFORM places it in the first half (the first 4 bytes) of DATA(N). In the routine that calls FFORM, the array DATA must be equivalenced with an INTEGER*4 array dimensioned 2*MAXDTA when a mix of integer and floating point data is to be read. On CDC machines integer, floating point and literal data are all stored in single words.

6.2 FFORM Usage

Figure 22 is an example of how FFORM can be used in parallel with formatted Fortran READ's. This style of coding has been used in most of our current input processors. The example shows one way to cope with INTEGER*4 data on IBM machines. A REAL*8 one-dimensional array DATA is equivalenced with an INTEGER*4, two-dimensional array IDATA. On IBM systems IDATA is dimensioned (2,MAXDTA); on CDC systems IDATA is dimensioned (1,MAXDTA). In the previous section it was noted that on IBM machines FFORM places INTEGER*4 words in the first half of the appropriate 8-byte element of the DATA array. With DATA and IDATA dimensioned as shown in Figure 22 an integer which is the Nth datum on an input card returns from FFORM in IDATA(1,N) on both IBM and CDC machines.

In order that ARC System codes can identify the input style chosen by the user a new data set designation, UNFORM=dataset, has been added to the STUFF input preprocessor, and an additional parameter has been added to the first record of BCD data sets produced by STUFF. The lead records of BCD data sets produced by STUFF must now be read:

```
      READ(M,99)ANAME,J,K,NOFORM,(NI),I=1,J)
   99 FORMAT(A8,3I5/(16I5))
```

(see Reference 1, pg. 53 for the old format). NOFORM=0 if the data set is in the formatted style (DATASET=dataset or SUBLOCK=dataset), NOFORM=1 for free-format datasets (UNFORM=dataset). When MODIFY=dataset is used the modifications must be in the same style as the original data set. NOSORT=dataset situations are usually special cases and, as such, are exempt from ARC System conventions. Stuff always sets NDFORM=0 for NOSORT data sets; the programmer may choose to ignore the NOFORM parameter if his application permits. Note that some data sets may be supplied in formatted style (DATASET=) and others in free-format style (UNIFORM=) within the same job.

```
CSW
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION DATA(12),IDATA(2,12)
CSW
CLW
C     DIMENSION DATA(12),IDATA(1,12)
CLW
      DIMENSION FLPT(4), INT(4), HOL(4)
      EQUIVALENCE (DATA(1),IDATA(1,1))
      DATA BLANK/6H      /
      MAXDTA=12
C
C  FORMATTED READ.
C
      IF( NOFORM.EQ.1 ) GO TO 20
   12 FORMAT(2X,4(F6.1,I6,A6))
CIBM
      READ (NIN,12,END=100) (FLPT(I),INT(I),HOL(I),I=1,4)
      GO TO 30
CIBM
CDC*
C     READ (NIN,12) (FLPT(I),INT(I),HOL(I),I=1,4)
C     IF( EOF(NIN) ) 100,30
CDC*
CRAY
C     READ (NIN,12) (FLPT(I),INT(I),HOL(I),I=1,4)
C     IF( IEOF(NIN) ) 100,30
CRAY
C  UNFORMATTED READ.
C
   20 CONTINUE
      DO 22 I=1,10,3
      DATA(I)=0.0
      IDATA(1,I+1)=0
      DATA(I+2)=BLANK
   22 CONTINUE
      CALL FFORM(DATA,NDATA,MAXDTA,NIN,IFLAG,NOUT)
      DO 24 I=1,4
      J=3*(I-1)+1
      FLPT(I)=DATA(J)
      INT(I)=IDATA(1,J+1)
      HOL(I)=DATA(J+2)
   24 CONTINUE
      IF( IFLAG.EQ.0 ) GO TO 100
   30 CONTINUE
  100 CONTINUE
```

Figure 22. An Example of FFORM Usage

6.3 FFORM SYNTAX

The rules for writing free-format card images are given below. The first five must be observed in any application of FFORM; the sixth applies to ARC System input only.

6.3.1 Delimiters

Data (integers, floating point numbers and Hollerith words) must be separated either by blanks or by combinations of one or more of the four special delimiters:

,	comma
(	left parenthesis
)	right parenthesis
/	slash

6.3.2 Data Forms

Integer and real numbers must be written according to the usual Fortran rules and may not have imbedded blanks. Hollerith data can be supplied in any of the following three ways:

1. A string of letters and numbers, beginning with a letter, with no imbedded blanks.

   e.g. U238 PU239

2. A string of symbols surrounded by asterisks or apostrophes. In the current version of FFORM for CDC machines only the asterisk can be used to set off Hollerith data; the apostrophe has not been implemented.

   e.g. *NA 23* 'REG1'

3. A string preceded by the Hollerith prefix nH, where n is an integer constant.

   e.g. 3H016

On IBM machines an asterisk may be part of a Hollerith string only when that string is surrounded by apostrophes (e.g. 'X*Y') or defined by the nH convention (e.g. 3HX*Y). Similarly an apostrophe may be a part of a Hollerith string only when that string is surrounded by asterisks (e.g. *ED'S*) or defined by the nH convention (e.g. 4HED'S). On CDC machines (where FFORM currently does not recognize apostrophes) an asterisk may be part of a Hollerith string only when that string is defined by the nH convention.

When a single asterisk (or apostrophe) is encountered the remaining data on the card are treated as Hollerith data. This does not apply, of course, to an asterisk that is clearly a part of a Hollerith string.

When FFORM passes the data it has read to the calling program, it has stored Hollerith data six characters to the word. Therefore, if the input description calls for one or more separate Hollerith words, each word must be six characters or less.

6.3.3 Implied Blanks and Zeroes

Pairs of commas, slashes, or left and right parentheses in consecutive columns of the card image will be interpreted as integer zeroes. Pairs of asterisks (or apostrophes) in consecutive columns of the card image imply Hollerith blanks.

```
e.g.    ,, = () = // = 0
        ** = '' = 1H
```

6.3.4 nR, the Repeat Option

nR causes the previous datum to be repeated n-1 times. n is an integer constant. When several pieces of data are enclosed by slashes or parentheses and are followed by a repeat instruction, the entire string of data will be repeated. Repeats can be nested by the use of slashes and parentheses, but each pair of symbols (// or ()) can be used only once per nest. This limits the depth of the nest to two levels.

```
e.g.    1.0,3R/2.0,1/2R = 1.0 1.0 1.0 2.0 1 2.0 1
        /(WORD 2R) 3R/ 4R = WORD 24R
```

6.3.5 $, End of Card

All data including and following a $ will be ignored. This will permit the user to include comments on a card. The symbol $ between asterisks (or apostrophes) or somewhere in an nH field is not affected.

6.3.6 UNFORM and Card Type Numbers

ARC System BCD data sets that are input in free-format form must be specified by "UNFORM=dataset" instead of the usual "DATASET=dataset". Card type numbers must continue to appear in columns 1-2, but all subsequent data can be punched without regard to field definitions.

```
e.g.    UNFORM=A.NIP3
        01    title
        02    0 1 0 7R
             etc.
```

Title cards in ARC System BCD data sets (the type 01 cards for several data sets) will continue to be read by ARC System input processors in the formatted mode.

7. REFERENCES

1. L. C. Just, H. Henryson II, A. S. Kennedy, S. D. Sparck, B. J. Toppel and P. M. Walker, "The System Aspects and Interface Data Sets of the Argonne Reactor Computation (ARC) System," ANL-7711, Argonne National Laboratory (1971).

2. R. Douglas O'Dell, "Standard Interface Files and Procedures for Reactor Physics Codes, Version IV," LA-6941-MS, Los Alamos Scientific Laboratory (1977).

3. LIBRARIAN Software Product, Applied Data Research, Inc., Princeton, New Jersey.

Appendix A

UTILITY SUBROUTINE DESCRIPTIONS

```
SOURCE MEMBER NAME.  ABEND
==================

USER ENTRY POINTS.  ABEND, ABSTOP, TRACER
=================

FUNCTION.  HANDLE ABNORMAL EXITS ON IBM MACHINES.  ABEND AND ABSTOP
========   LINK TO THE ABEND MACRO.  ABEND TRIGGERS A DUMP, ABSTOP
           DOES NOT.  TRACER IS A TRACEBACK ROUTINE.

           THESE ROUTINES ARE FORTRAN CALLABLE.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==========================================

      CALL ABEND
      CALL ABSTOP
      CALL TRACER(SUBNAM,ISN)

           SUBNAM  NAME OF PREVIOUS SUBROUTINE IN TRACEBACK.
           ISN     ISN IN PREVIOUS SUBROUTINE FROM WHICH CURRENT
                   SUBROUTINE WAS CALLED.

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.  NONE
================================================================

COMMON BLOCKS.  NONE
=============

LANGUAGE.  IBM ASSEMBLER
========

FILES REFERENCED.  NONE
================

ACCOMPANYING SUBPROGRAMS.  ABEND, TRACER
========================

LIBRARY, LINK EDIT INPUT.  C116.CCCC.SYSLIB(ABEND)
========================

RELATED MEMBERS.  ERROR
===============

REFERENCES.  NONE
==========
```

SOURCE MEMBER NAME. ARCBCD
================

USER ENTRY POINTS. SCAN, STUFF
================

FUNCTION. SCAN READS THE ENTIRE CARD-IMAGE INPUT FILE (LOGICAL UNIT NUMBER NIN) AND SPOOLS THE CARD IMAGES ONTO A FILE NAMED ARC (IF ARC IS NOT A FILE NAME IN THE SEEK TABLE, THE DEFAULT LOGICAL UNIT NUMBER IS 9). AT THE SAME TIME SCAN LOOKS FOR THE BLOCK KEYWORD (BLOCK=) AND DEFINES THE VARIABLES BLKNAM,IBLTAB AND NBLOCK IN /STFARC/.

STUFF PROCESSES THE INPUT DATA ASSOCIATED WITH THE NEXT UNPROCESSED DATA BLOCK NAMED STFNAM (STFNAM IS A VARIABLE IN /STFARC/). STUFF CREATES OR MODIFIES BCD DATASETS AND SETS THE VARIABLE NRET IN /STFARC/.

```
NRET = 1, THE NFXT DATA BLOCK NAMED STFNAM WAS PROCESSED.
      -3, THERE IS NO DATA BLOCK NAMED STFNAM IN THE INPUT.
      -5, ALL DATA BLOCKS NAMED STFNAM WERE PROCESSED PRIOR
          TO THIS CALL TO STUFF.
```

THESE VERSIONS OF STUFF AND SCAN SHOULD BE USED IN A STANDALONE ENVIRONMENT AND FOR EXPORT. THE INTERFACE SUBROUTINE VERSIONS (SEE SOURCE MEMBERS STUFFARC AND SCANARC) SHOULD BE USED IN A MODULAR ENVIRONMENT. IN A MODULAR ENVIRONMENT THE FUNCTIONS OF SCAN AND STUFF ARE PERFORMED BY THE MODULE SYS002.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL SCAN
      CALL STUFF
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
===

NOUT	LOGICAL UNIT NUMBER OF OUTPUT PRINT FILE.
NOUT2	LOGICAL UNIT NUMBER OF AUXILIARY OUTPUT PRINT FILE.
NIN	INPUT CARD FILE LOGICAL UNIT NUMBER. APPLIES FOR CALLS TO SCAN ONLY.
STFNAM	NAME OF THE DATA BLOCK TO BE PROCESSED BY STUFF. APPLIES FOR CALLS TO STUFF ONLY.

COMMON BLOCKS.
============

```
      COMMON/STFARC/STFNAM,BLKNAM(50),IBLTAB(3,50),NBLOCK,NRET
      COMMON/IOPUT/NIN,NOUT,NOUT2
```

(continued)

LANGUAGE. FORTRAN
========

FILES REFERENCED. INPUT CARD FILE, SPOOLED INPUT FILE (THE FILE NAMED ARC, OR LUN 9), OUTPUT FILES NOUT AND NOUT2. VARIOUS BCD DATA SETS.
================

ACCOMPANYING SUBPROGRAMS. SCAN, STUFF, STUFF1, CODECD
========================

LIBRARY. C116.CCCC.SEGLIB(ARCBCD)
=======

RELATED MEMBERS. STUFFARC, SCANARC, SYS001, SYS002
===============

REFERENCES. L.C. JUST, H. HENRYSON, II, A.S. KENNEDY, S.D. SPARCK, B.J. TOPPEL, AND P.M. WALKER, THE SYSTEM ASPECTS AND INTERFACE DATA SETS OF THE ARGONNE REACTOR COMPUTATION (ARC) SYSTEM, ANL-7711, ARGONNE NATIONAL LABORATORY (1971).
==========

SOURCE MEMBER NAME. CRED
==================

USER ENTRY POINTS. CRED, CRIT
=================

FUNCTION. CRED/CRIT HANDLES DATA TRANSFER BETWEEN EXTENDED CORE (ECM) AND FAST CORE (FCM) ACCORDING TO CCCC SPECIFICATIONS. CRED TRANSFERS DATA FROM ECM TO FCM. CRIT TRANSFERS DATA FROM FCM TO ECM.
========

CRED/CRIT IS INTERLOCKED WITH DOPC TO ENSURE THAT COMMON BLOCK /ECMLOC/ HAS BEEN INITIALIZED BY SUBROUTINE DOPC PRIOR TO THE FIRST CRED/CRIT CALL.

ON SHORT-WORD MACHINES (E.G. IBM) THE DATA TRANSFER STARTING LOCATION (LCM) MUST BE IN UNITS OF SHORT (REAL*4) WORDS. (E.G. OBTAINED THROUGH A CALL TO IPT2 OR ITS EQUIVALENT IN BPOINTER APPLICATIONS).

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===

```
      CALL CRED(SCM,LCM,NSWDS,NERR)
      CALL CRIT(SCM,LCM,NSWDS,NERR)
```

SCM	STARTING FCM ADDRESS FOR DATA TRANSFER
LCM	STARTING ECM LOCATION FOR DATA TRANSFER
NSWDS	NO. OF SINGLE-PRECISION WORDS TO BE TRANSFERRED
NERR	ERROR FLAG

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
===

NOUT	OUTPUT PRINT FILE LOGICAL UNIT NUMBER
/ECMLOC/	ENTIRE ECMLOC COMMON BLOCK MUST BE INITIALIZED BY A CALL TO DOPC PRIOR TO FIRST CRED/CRIT CALL ON ONE-LEVEL MACHINE IMPLEMENTATIONS.

COMMON BLOCKS.
==============

```
      COMMON/ECMLOC/MCHLEV,NDXECM,LCMUSR,LCMSYS,LOCKEC
      COMMON/ECMRF/BLK(1)
      COMMON/IOPUT/NIN,NOUT,NOUT2
```

LANGUAGE. FORTRAN
=========

(continued)

FILES REFERENCED. NONE
===============

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(CRED)
========================

RELATED MEMBERS. DOPC
===============

REFERENCES. R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
========== FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
LOS ALAMOS NATIONAL LABORATORY (1977).

SOURCE MEMBER NAME. DOPC
=================

USER ENTRY POINTS. DOPC, DOPCO, DOPCD
=================

FUNCTION. DOPC DEFINES AND RELEASES RANDOM ACCESS DISK FILES WHICH
======== ARE USED FOR MULTILEVEL DATA TRANSFERS ACCORDING TO CCCC SPECIFICATIONS. DOPC ALSO INITIALIZES THE PSEUDO-ECM CONTAINER REFERENCE POINTERS IN COMMON BLOCK /ECMLOC/ WHICH ARE USED BY DOPC, DRED/DRIT AND CRED/CRIT. DOPC MUST BE CALLED PRIOR TO DRED/DRIT AND CRED/CRIT CALLS. THIS ROUTINE INITIALIZES PARAMETERS WHICH ARE NEEDED FOR RESERVING SPACE ON DISK AND IN CORE.

THE CDC VERSION OF DOPC CALLS IGTXCM TO ALLOCATE AN AUXILIARY ECM CONTAINER NAMED XCM. WITHIN XCM IS ALLOCATED DIRECTORIES FOR EACH RANDOM ACCESS I/O FILE.

ENTRY POINTS DOPCO AND DOPCD PROVIDE THE /ECMLOC/ INITIALIZATION INTERLOCK FOR CRED/CRIT AND DRED/DRIT, RESPECTIVELY.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL DOPC(IOP,NREF,NGREF,IERRCS,ARRAY,MXLEN,MXBLOK,LENFIL)
```

IOP	0/1/2/3/4. INITIALIZE DOPC/DEFINE FILE/ FILE GROUP COMPLETE/DELETE GROUP/WRAP-UP DOPC
NREF	FILE REFERENCE NO. (IOP=1)
NGREF	FILE GROUP NO. (IOP=2,3)
IERRCS	ERROR FLAG
ARRAY	STARTING ADDRESS OF ECM CONTAINER ON ONE-LEVEL MACHINES (IOP=0)
MXLEN	MAX. BLOCK LENGTH IN SINGLE-PRECISION WORDS (IOP=1)
MXBLOK	MAX. NO. OF BLOCKS (IOP=1)
LENFIL	TOTAL FILE LENGTH. LENFIL=0 IMPLIES MXBLOK BLOCKS OF MXLEN WORDS EACH

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
==

NOUT	OUTPUT PRINT FILE LOGICAL UNIT NUMBER

(continued)

COMMON BLOCKS.
============

```
      COMMON/ECMLOC/MCHLEV,NDXECM,LCMUSR,LCMSYS,LOCKEC
      COMMON/ECMRF1/BLK(1)
      COMMON/ECMRF2/BLKECS(1)
      COMMON/IOPUT/NIN,NOUT,NOUT2
      COMMON/RNDMIO/IDXDF(99),IDXGRP(25),LENIDX(25),LOCIDX(25),
     X LOPENF(25),LSTLOC,MAXLOC,MAXRDM,MAXREF,NRNDOM,LOCBAS
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(DOPC)
========================

RELATED MEMBERS. IGTXCM, DRED, CRED
===============

REFERENCES. R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
========== FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
            LOS ALAMOS NATIONAL LABORATORY (1977).

SOURCE MEMBER NAME. DRED
==================

USER ENTRY POINTS. DRED, DRIT
=================

FUNCTION. DRED/DRIT HANDLES RANDOM ACCESS I/O BETWEEN ECM AND DISK
========= ACCORDING TO CCCC SPECIFICATIONS. DRED READS DATA FROM
DISK. DRIT WRITES DATA TO DISK.

DRED/DRIT IS INTERLOCKED WITH DOPC TO ENSURE THAT COMMON BLOCK /ECMLOC/ HAS BEEN INITIALIZED BY SUBROUTINE DOPC PRIOR TO THE FIRST DRED/DRIT CALL.

ON SHORT-WORD MACHINES (E.G. IBM) THE DATA TRANSFER STARTING LOCATION (LOCBUF) MUST BE IN UNITS OF SHORT (REAL*4) WORDS. (E.G. AN ECM POINTER OBTAINED THROUGH A CALL TO IPT2 OR ITS EQUIVALENT IN BPOINTER APPLICATIONS).

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===

```
      CALL DRED(LUN,IREC,LOCBUF,NWDS,MODE)
      CALL DRIT(LUN,IREC,LOCBUF,NWDS,MODE)

           LUN     DISK FILE LIGICAL UNIT NUMBER
           IREC    FILE RECORD NUMBER
           LOCBUF  STARTING ECM LOCATION FOR DATA TRANSFER
           NWDS    NO. OF SINGLE-PRECISION WORDS TO BE TRANSFERRED
           MODE    0/1/2.  INITIATE NON-ASYNCHRONOUS TRANSFER/
                   INITIATE ASYNCHRONOUS TRANSFER/COMPLETE PREVIOUS
                   ASYNCHRONOUS TRANSFER
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
===

```
           NOUT    OUTPUT PRINT FILE LOGICAL UNIT NUMBER
          /ECMLOC/ ENTIRE ECMLOC COMMON BLOCK MUST BE INITIALIZED
                   BY A CALL TO DOPC PRIOR TO FIRST DRED/DRIT CALL
                   ON ONE-LEVEL MACHINE IMPLEMENTATIONS.
```

COMMON BLOCKS.
==============

```
      COMMON/ECMRF1/BLK(1)
      COMMON/ECMRF2/BLKECS(1)    (CDC ONLY)
      COMMON/ECMLOC/MCHLEV,NDEECM,LCMUSR,LCMSYS,LOCKEC
      COMMON/IOPUT/NIN,NOUT,NOUT2
      COMMON/RNDMIO/IDXDF(99),IDXGRP(25),LENIDX(25),LOCIDX(25),
     X LOPENF(25),LSTLOC,MAXLOC,MAXRDM,MAXREF,NRNDOM,LOCBAS
```

(continued)

LANGUAGE. FORTRAN
========

FILES REFERENCED. VARIOUS BINARY, RANDOM ACCESS DATA SETS
================

ACCOMPANYING SUBPROGRAMS. DRED1, DRED2 (CDC VERSION ONLY)
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(DRED)
========================

RELATED MEMBERS. DOPC
===============

REFERENCES. R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
========== FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
LOS ALAMOS NATIONAL LABORATORY (1977).

```
SOURCE MEMBER NAME.  ECMV
==================

USER ENTRY POINTS.  ECMV
=================

FUNCTION.  ECMV TRANSFERS DATA BETWEEN LOCATIONS IN EXTENDED CORE
========   (ECM) ACCORDING TO CCCC SPECIFICATIONS.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==========================================

      CALL ECMV(LCM1,LCM2,NSWDS)

           LCM1    ECM LOCATION TO WHICH DATA IS TO BE TRANSFERRED
           LCM2    ECM LOCATION FROM WHICH DATA IS TO BE TRANSFERRED
           NSWDS   NO. OF SINGLE-PRECISION WORDS

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.  NONE
================================================================

COMMON BLOCKS.  NONE
=============

LANGUAGE.  FORTRAN
========

FILES REFERENCED.  NONE
================

ACCOMPANYING SUBPROGRAMS.  ECMV
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(ECMV)
========================

RELATED MEMBERS.  CRED
===============

REFERENCES.  R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
==========   FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
             LOS ALAMOS NATIONAL LABORATORY (1977).
```

```
SOURCE MEMBER NAME.  ERROR
==================

USER ENTRY POINTS.  ERROR
=================

FUNCTION.  ERROR IS A UTILITY SUBROUTINE WHICH KEEPS STATISTICS ON
========   ERRORS DETECTED BY AN APPLICATION PROGRAM AND WHICH
           LOCALIZES THE HANDLING OF ABNORMAL TERMINATIONS.  THE
           ROUTINE SHOULD BE CALLED EVERY TIME AN ERROR IS DETECTED.
           WHEN ERROR IS CALLED WITH THE FIRST ARGUMENT = 5HFATAL
           A TABLE OF ENCOUNTERED ERRORS IS PRINTED, AND, IF ANY
           OF THE ERRORS WERE FATAL, THE JOB TERMINATES ON AN
           ABNORMAL EXIT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==========================================

      CALL ERROR(SUBNAM,IERR)

           SUBNAM  EITHER THE NAME OF A SUBROUTINE FROM WHICH AN
                   ERROR WAS DETECTED, OR 5HFATAL.  WHEN
                   SUBNAM.NE.(5HFATAL) THE ERROR IS NOTED BUT NO
                   OTHER ACTION IS TAKEN.  WHEN SUBNAM.EQ.(5HFATAL)
                   A LIST OF ERRORS IS PRINTED, AND, IF ANY FATAL
                   ERRORS HAVE BEEN ENCOUNTERED, THE JOB TERMINATES.
           IERR    ERROR NUMBER.  NEGATIVE NUMBERS INDICATE FATAL
                   ERRORS.  POSITIVE NUMBERS INDICATE NONFATAL
                   ERRORS (WARNINGS).

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
================================================================

           NOUT    OUTPUT PRINT FILE LOGICAL UNIT NUMBER
           NOUT2   AUXILIARY OUTPUT PRINT FILE LOGICAL UNIT NUMBER

COMMON BLOCKS.
=============

      COMMON/IOPUT/NIN.NOUT,NOUT2

LANGUAGE.  FORTRAN
========

FILES REFERENCED.  NONE
================
```

(continued)

ACCOMPANYING SUBPROGRAMS. NONE
=======================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(ERROR)
=======================

RELATED MEMBERS. NONE
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. FEQUAT
==================

USER ENTRY POINTS. FEQUAT
=================

FUNCTION. FEQUAT TRANSFERS ILEN FLOATING POINT FULL WORDS FROM XB TO XA.
========   ON IBM MACHINES XB AND XA ARE 8-BYTE VARIABLES.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL FEQUAT(XA,XB,ILEN)

           XA      OUTPUT ARRAY
           XB      INPUT ARRAY
           ILEN    NO. OF ELEMENTS IN XA
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(FEQUAT)
========================

RELATED MEMBERS. NONE
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. FFORM

USER ENTRY POINTS. FFORM

FUNCTION. FFORM READS THE NEXT CARD IMAGE IN A FREE-FORMAT BCD FILE, INTERPRETS THE TYPE OF DATA FOUND, AND RETURNS THE DATA TO THE CALLING PROGRAM.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.

```
      CALL FFORM(DATA,NWORD,MAXDTA,NFL,IFLAG,NOUT)
```

DATA	AN ARRAY (REAL*8 ON IBM MACHINES) INTO WHICH FFORM IS TO PLACE THE DATA FOUND ON A CARD
NWORD	THE NUMBER OF DATA ENCOUNTERED BY FFORM ON THE CARD
MAXDTA	THE LENGTH OF THE ARRAY DATA (REAL*8 ON IBM MACHINES)
NFL	THE LOGICAL UNIT NUMBER OF THE BCD FILE TO BE READ
IFLAG	0, AN END OF FILE WAS ENCOUNTERED, 1, A CARD WAS READ AND DATA TRANSFERRED SUCCESSFULLY, -1, AN INPUT ERROR (AN UNRECOGNIZED FORMAT) WAS ENCOUNTERED
NOUT	THE LOGICAL UNIT NUMBER FOR THE OUTPUT ERROR MESSAGES. IF NOUT = 0 NO MESSAGES ARE PRINTED.

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE

COMMON BLOCKS. NONE

LANGUAGE. FORTRAN

FILES REFERENCED. VARIOUS BCD DATASETS

ACCOMPANYING SUBPROGRAMS. FFORM1, FFORM2

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(FFORM)

RELATED MEMBERS. NONE

REFERENCES. NONE

SOURCE MEMBER NAME. FLTSET
==================

USER ENTRY POINTS. FLTSET
=================

FUNCTION. FLTSET SETS ALL THE ELEMENTS OF A LONG-WORD ARRAY (XA) EQUAL
========   TO A CONSTANT (XSET). ON IBM MACHINES XA AND XSET ARE
8-BYTE WORDS.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL FLTSET(XA,XSET,ILEN)

           XA      ARRAY TO BE INITIALIZED
           XSET    CONSTANT TO BE USED
           ILEN    NO. OF ELEMENTS IN XA
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(FLTSET)
========================

RELATED MEMBERS. NONE
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. IEQUAT
=================

USER ENTRY POINTS. IEQUAT
=================

FUNCTION. IEQUAT TRANSFERS ILEN INTEGER WORDS FROM IB TO IA.
========

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL IEQUAT(IA,IB,ILEN)

           IA      OUTPUT ARRAY
           IB      INPUT ARRAY
           ILEN    NO. OF ELEMENTS IN IA
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(IEQUAT)
========================

RELATED MEMBERS. NONE
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. IGTLCM

USER ENTRY POINTS. IGTLCM, IGTSCM, IGTXCM, FRELCM, FRESCM, FREXCM, LOCFWD

FUNCTION. IGTLCM AND IGTSCM ALLOCATE MEMORY IN BULK AND MAIN CORE, RESPECTIVELY. IGTXCM ALLOCATES AN AUXILIARY ECM CONTAINER NAMED XCM. FRELCM, FRESCM AND FREXCM RELEASE ALLOCATED SPACE.

ON IBM MACHINES THE STANDARD IBM MACRO INSTRUCTIONS GETMAIN AND FREEMAIN ARE USED TO ALLOCATE AND FREE CONSECUTIVE BLOCKS OF WORDS OF CORE. THE DESIGNATIONS SUBPOOL 1 AND 2 ARE ASSIGNED TO THE LCM AND SCM CONTAINERS, RESPECTIVELY. THE MACROS ARE EXECUTED THROUGH THE ASSEMBLER ROUTINES MYLCM, MYSCM, FREELC AND FREESC.

ON CDC MACHINES THE CONTAINERS ARE ESTABLISHED BY CALLS TO THE MEMORY MACRO WHICH INCREASES LCM AND/OR SCM FIELD LENGTH. THE MACRO IS EXECUTED THROUGH THE COMPASS ROUTINES JGTLCM AND JGTSCM. THE CONTAINER IS PLACED AT THE END OF THE NEWLY ACQUIRED SPACE. THERE IS A SMALL MARGIN BETWEEN THE CONTAINER AND THE END OF THE FIELD LENGTH FOR THOSE SYSTEMS ON WHICH SYSTEM IO ROUTINES MAY READ AHEAD.

ON THE CRAY MACHINE WITH THE CTSS OPERATING SYSTEM TWO LIBRARY SUBROUTINES LASTMEM AND MEMORY RESPECTIVELY DETERMINE AND CHANGE A JOB'S FIELD LENGTH. LASTMEM DETERMINES JCHLM THE HIGH LIMIT OF THE USER CODE AREA. SUBROUTINE MEMORY INCREMENTS AND DECREMENTS JCHLM CAUSING A CORRESPONDING CHANGE IN THE JOB FIELD LENGTH. THE RELEASE OF A CONTAINER WITH LAST WORD ADDRESS EQUAL TO JCHLM CAUSES JCHLM TO BE REDUCED. THE RELEASE OF OTHER CONTAINERS SIMPLY FREES THE SPACE FOR POTENTIAL USE IN THE NEXT IGTLCM CALL.

AT INSTALLATIONS WITH THE STANDARD CRAY OPERATING SYSTEM (COS), FORTRAN SUBROUTINE MEMGET WITH ENTRY POINT LASTMEM ARE FUNCTIONALLY EQUIVALENT TO THE CTSS ROUTINES MEMORY AND LASTMEM. MEMGET INVOKES THE COS LIBRARY ROUTINE MEMORY (5 ARGUMENTS) WHICH IN TURN INVOKES THE CAL ASSEMBLER MEMORY MACRO TO INCREMENT AND DECREMENT JCHLM.

(continued)

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
ILOCM=IGTSCM(NWDS)
ILOCB=IGTLCM(NWDS)
ILOCX=IGTXCM(NWDS)
CALL FRESCM
CALL FRELCM
CALL FREXCM
LOCWRD=LOCFWD(BLK(1))

     ILOCM   ADDRESS OF ALLOCATED MAIN MEMORY (BYTES).
     ILOCB   ADDRESS OF ALLOCATED BULK MEMORY (BYTES).
     ILOCX   ADDRESS OF ALLOCATED XCM CONTAINER (BYTES).
     BLK     VARIABLE FOR WHICH ADDRESS IS REQUIRED.
     LOCWRD  SHORT (REAL*4) WORD ADDRESS OF VARIABLE.
     NWDS    NO. OF 4-BYTE WORDS TO BE ALLOCATED OR FREED.
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
===

```
     NOUT    OUTPUT PRINT FILE LOGICAL UNIT NUMBER
```

COMMON BLOCKS.
=============

```
COMMON /IOPUT/ NIN,NOUT,NOUT2
COMMON /ALLOCS/ INI(3),NEW(3),NPT(3),MAXNU,MCHLEV,NBYTEW
COMMON /SAVMEM/ INIT,FLMAX,JCHLMO,JCHLM,LENFLD
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. JGTSCM (CDC), JGTLCM (CDC), MEMGET (CRAY),
FRELCM, JGT, LOCFWD, LASTMEM (CRAY)
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(IGTLCM)
========================

RELATED MEMBERS. POINTR, MYLCM, DOPC
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. INTSET
==================

USER ENTRY POINTS. INTSET
=================

FUNCTION. INTSET INITIALIZES AN INTEGER ARRAY (IA) TO THE VALUE OF AN
========   INPUT INTEGER CONSTANT (ISET). ON IBM MACHINES IA AND ISET
           ARE 4-BYTE WORDS.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL INTSET(IA,ISET,ILEN)

           IA      ARRAY TO BE INITIALIZED
           ISET    CONSTANT TO BE USED
           ILEN    NO. OF ELEMENTS IN IA
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(INTSET)
========================

RELATED MEMBERS. NONE
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. IN2LIT
==================

USER ENTRY POINTS. IN2LIT
=================

FUNCTION. IN2LIT CONVERTS AN INTEGER VARIABLE (INTEGER*4 ON IBM
======== MACHINES) INTO A 6-CHARACTER LITERAL, LEFT JUSTIFIED.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL IN2LIT(INT,WORD)

           INT     THE INPUT INTEGER
           WORD    THE OUTPUT LITERAL
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(IN2LIT)
========================

RELATED MEMBERS. NONE
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. LINES
==================

USER ENTRY POINTS. LINES, LINES2
=================

FUNCTION. LINES HANDLES PAGE EJECTS, HEADINGS, AND NUMBERING FOR THE
======== OUTPUT FILE NOUT. LINES2 DOES THE SAME FOR THE AUXILIARY
OUTPUT FILE NOUT2.

LINES KEEPS COUNT OF OUTPUT LINES PRINTED AND PAGE EJECTS IF THE NEXT BLOCK OF OUTPUT WILL NOT FIT ON THE CURRENT PAGE.

THIS VERSION OF LINES SHOULD BE USED IN A STANDALONE ENVIRONMENT AND FOR EXPORT. THE INTERFACE SUBROUTINE VERSION IN C116.CCCC.SYSLIB (SEE SOURCE MEMBER LINESARC) SHOULD BE USED IN A MODULAR ENVIRONMENT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===

```
      CALL LINES(NUMLIN,IFLAG1)
      CALL LINES2(NUMLIN,IFLAG1)

           NUMLIN  GT.0, THE NUMBER OF LINES ABOUT TO BE PRINTED
                   (ON LOGICAL UNIT NOUT FOR LINES, NOUT2 FOR LINES2).
                   EQ.0, SKIP TO THE TOP OF THE NEXT PAGE.
                   LT.0, THE LINE COUNT IS REDUCED.  CONSECUTIVE
                   CALLS TO LINES WITH FIRST POSITIVE, THEN NEGATIVE
                   LINE COUNTS CAN BE USED TO AVOID PRINTING A
                   TABLE HEADING, BUT NO DATA, AT THE BOTTOM OF A
                   PAGE.
           IFLAG1  RETURNED FLAG.  0/1, SAME PAGE/NEW PAGE.
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
===

```
           TITLE   ARRAY OF HOLERITH WORDS USED FOR PAGE HEADING
                   TITLE (A6).  TITLE IS PRINTED UP TO A MAXIMUM OF
                   6 LINES USING 11 WORDS PER LINE.
           TIME    SUBROUTINE TIMER VARIABLES (REAL*8 WORDS).
           HEAD    PAGE HEADING LEAD (PRINTED 4A6).
           NOUT    LOGICAL UNIT NUMBER OF OUTPUT PRINT FILE.
           NOUT2   LOGICAL UNIT NUMBER OF AUXILIARY OUTPUT PRINT FILE.
           NTITLE  NO. OF WORDS OF THE ARRAY TITLE TO BE PRINTED.
                   NTITLE IS LESS THAN OR EQUAL TO 66.
```

(continued)

COMMON BLOCKS.
=============

```
      COMMON /PTITLE/ TITLE(66),TIME(10),HEAD(4),NOUT,NOUT2,NTITLE
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. OUTPUT FILES NOUT AND NOUT2
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY. C116.CCCC.SEGLIB(LINES)
=======

RELATED MEMBERS. LINESARC, SYS004
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. LINESARC

USER ENTRY POINTS. LINES, LINES2

FUNCTION. THIS IS AN INTERFACE SUBROUTINE WHICH LINKS TO THE SYS004 MODULE TO EXECUTE THE LINES SUBROUTINE.

LINES HANDLES PAGE EJECTS, HEADINGS, AND NUMBERING FOR THE OUTPUT FILE NOUT. LINES2 DOES THE SAME FOR THE AUXILIARY OUTPUT FILE NOUT2.

LINES KEEPS COUNT OF OUTPUT LINES PRINTED AND PAGE EJECTS IF THE NEXT BLOCK OF OUTPUT WILL NOT FIT ON THE CURRENT PAGE.

THIS INTERFACE SUBROUTINE SHOULD BE USED IN A MODULAR ENVIRONMENT. THE VERSION IN C116.CCCC.SEGLIB (SEE SOURCE MEMBER LINES) SHOULD BE USED IN A STANDALONE ENVIRONMENT AND FOR EXPORT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.

```
      CALL LINES(NUMLIN,IFLAG1)
      CALL LINES2(NUMLIN,IFLAG1)
```

NUMLIN	GT.0, THE NUMBER OF LINES ABOUT TO BE PRINTED (ON LOGICAL UNIT NOUT FOR LINES, NOUT2 FOR LINES2). EQ.0, SKIP TO THE TOP OF THE NEXT PAGE. LT.0, THE LINE COUNT IS REDUCED. CONSECUTIVE CALLS TO LINES WITH FIRST POSITIVE, THEN NEGATIVE LINE COUNTS CAN BE USED TO AVOID PRINTING A TABLE HEADING, BUT NO DATA, AT THE BOTTOM OF A PAGE.
IFLAG1	RETURNED FLAG. 0/1, SAME PAGE/NEW PAGE.

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.

TITLE	ARRAY OF HOLERITH WORDS USED FOR PAGE HEADING TITLE (A6). TITLE IS PRINTED UP TO A MAXIMUM OF 6 LINES USING 11 WORDS PER LINE.
TIME	SUBROUTINE TIMER VARIABLES (REAL*8 WORDS).
HEAD	PAGE HEADING LEAD (PRINTED 4A6).
NOUT	LOGICAL UNIT NUMBER OF OUTPUT PRINT FILE.
NOUT2	LOGICAL UNIT NUMBER OF AUXILIARY OUTPUT PRINT FILE.
NTITLE	NO. OF WORDS OF THE ARRAY TITLE TO BE PRINTED. NTITLE IS LESS THAN OR EQUAL TO 66.

(continued)

COMMON BLOCKS.
=============

```
      COMMON /PTITLE/ TITLE(66),TIME(10),HEAD(4),NOUT,NOUT2,NTITLE
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. OUTPUT FILES NOUT AND NOUT2
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY. C116.CCCC.SYSLIB(LINES)
=======

RELATED MEMBERS. LINES, SYS004
===============

REFERENCES. NONE
==========

```
SOURCE MEMBER NAME.  LRED
==================

USER ENTRY POINTS.  LRED, LRIT
=================

FUNCTION.  LRED/LRIT HANDLES DATA TRANSFER BETWEEN EXTENDED CORE
=========  (ECM) AND SEQUENTIAL DATA SETS ACCORDING TO CCCC
           SPECIFICATIONS.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==========================================

      CALL LRED(LUN,IREC,B,NWDS,MODE)
      CALL LRIT(LUN,IREC,B,NWDS,MODE)

           LUN      FILE REFERENCE NO.
           IREC     RECORD NO.
           B        ECM ARRAY
           NWDS     NO. OF SINGLE-PRECISION WORDS TO BE TRANSFERRED
           MODE     0/1/2.  COMPLETE I/O BEFORE RETURN/START I/O BEFORE
                    RETURN/COMPLETE I/O STARTED BY PREVIOUS CALL

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
================================================================

           NOUT     LOGICAL UNIT NO. OF OUTPUT PRINT FILE

COMMON BLOCKS.
=============

      COMMON/IOPUT/NIN,NOUT,NOUT2
      COMMON/INITIO/ NREC(99)

LANGUAGE.  FORTRAN
========

FILES REFERENCED.  VARIOUS SEQUENTIAL DATA SETS
================

ACCOMPANYING SUBPROGRAMS.  NONE
========================

LIBRARY, LINK EDIT INPUT.  C116.CCCC.SYSLIB(LRED)
========================

RELATED MEMBERS.  REED
===============

REFERENCES.  R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
==========   FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
             LOS ALAMOS NATIONAL LABORATORY (1977).
```

SOURCE MEMBER NAME. MYLCM
==================

USER ENTRY POINTS. MYLCM, MYSCM, FREELC, FREESC, LOCF
=================

FUNCTION. MYLCM AND MYSCM ALLOCATE MEMORY IN BULK AND MAIN CORE,
========   RESPECTIVELY, IN IBM 370 SYSTEMS. FREELC AND FREESC
RELEASE ALLOCATED SPACE. LOCF RETURNS THE MACHINE
ADDRESS OF A VARIABLE.

THESE ROUTINES ARE FORTRAN CALLABLE.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
ILOCM=MYSCM(NWDS)
ILOCB=MYLCM(NWDS)
CALL FREESC(NWDS,ILOCM)
CALL FREELC(NWDS,ILOCB)
ILOC=LOCF(BLK)
```

ILOCM	ADDRESS OF ALLOCATED MAIN MEMORY (BYTES).
ILOCB	ADDRESS OF ALLOCATED BULK MEMORY (BYTES).
NWDS	NO. OF 4-BYTE WORDS TO BE ALLOCATED OR FREED.
BLK	VARIABLE FOR WHICH ADDRESS IS REQUIRED.
ILOC	ADDRESS OF VARIABLE (BYTES).

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
===

COMMON BLOCKS. NONE
=============

LANGUAGE. IBM ASSEMBLER
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. LOCF
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(MYLCM)
========================

RELATED MEMBERS. POINTR, IGTLCM
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. POINTR
==================

USER ENTRY POINTS. POINTR, PUTPNT, PUTBLK, BULK, FREE, WIPOUT, CLEAR,
================= GETPNT, GETN, DUMP, IGET, IPT2, PUTM, PUTB, IPTERR,
NNAMSF, ILAST, ILASTB, REDEF, REDEFM, REDEFB, PURGE,
PURGEB, STATUS, PRTI1, PRTI2, PRTR1, PRTA1, PRTR2,
PRTA2, PRTECM, ECZERO

FUNCTION. THESE ROUTINES, PLUS A FEW OTHERS LISTED UNDER RELATED
======== MEMBERS, COMPRISE THE BPOINTER PACKAGE USED FOR DYNAMIC CORE STORAGE MANAGEMENT. IN THE DESCRIPTION THAT FOLLOWS FCM REFERS TO FAST CORE AND ECM TO EXTENDED CORE (E.G. LCM ON A CDC 7600).

POINTR INITIALIZES THE PACKAGE AND ALLOCATES THE FCM DATA CONTAINER. BULK ALLOCATES THE ECM DATA CONTAINER AND MUST BE CALLED BEFORE POINTR. FREE RELEASES THE FCM AND ECM CONTAINERS.

PUTM AND PUTPNT RESERVE SPACE IN THE FCM CONTAINER FOR A PARTICULAR DATA ARRAY. PUTB AND PUTBLK PERFORM THE SAME FUNCTION IN ECM. WIPOUT RELEASES THE CONTAINER SPACE FOR A PARTICULAR ARRAY. CLEAR INITIALIZES AN ARRAY TO ZERO.

GETPNT AND IGET RETURN POINTERS TO PARTICULAR ARRAYS IN THE CONTAINERS. IPT2 RETURNS POINTERS TO SUBARRAYS WITHIN A PARTICULAR ARRAY. ILAST AND ILASTB RETURN THE FIRST AVAILABLE (I.E. NOT YET ALLOCATED TO AN ARRAY) LOCATION IN THE FCM AND ECM CONTAINER, RESPECTIVELY.

IPTERR RETURNS THE NUMBER OF ERRORS THAT HAVE OCCURED IN BPOINTER ROUTINES SINCE THE LAST CALL TO IPTERR. DUMP PROVIDES EDITS OF ARRAYS WHEN THE DUMP FLAG IS ON.

REDEF AND REDEFM REDEFINE THE FCM SPACE RESERVED FOR A PARTICULAR ARRAY. REDEFB PERFORMS THE SAME FUNCTION FOR ARRAYS IN ECM.

PURGE AND PURGEB ELIMINATE UNUSED SPACE BETWEEN ARRAYS (FCM AND ECM, RESPECTIVELY) BY TAMPING DOWN THE CONTAINERS. AFTER A PURGE THE NEW POINTERS MUST BE OBTAINED FROM CALLS TO GETPNT OR IGET.

STATUS PRINTS THE CURRENT STATUS OF THE BPOINTER TABLES WHEN THE PRINT FLAG IS SET. NNAMSF RETURNS THE NUMBER OF NAMES IN THE BPOINTER TABLES. MAXWMF AND MAXWBF RETURN THE MAXIMUM FCM AND ECM SPACE USED.

(continued)

AFTER SPACE IS RELEASED (WIPOUT) OR ALTERED (REDEF) THE PROGRAMMER SHOULD CALL PURGE TO TAMP THE CONTAINER. BPOINTER IS PROGRAMMED TO DO AUTOMATIC (AND THEREFORE UNEXPECTED) PURGES WHEN REQUESTED SPACE IS UNAVAILABLE AS A CONTIGUOUS BLOCK BECAUSE OF CONTAINER FRAGMENTATION.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.

```
CALL POINTR(BLK,MAXSIZ,IPRINT)
CALL BULK(MAXBLK)
CALL FREE
CALL PUTPNT(ANAME,LEN,LWD)
CALL PUTM(ANAME,LEN,LWD,IPT)
CALL PUTBLK(ANAME,LEN,LWD)
CALL PUTB(ANAME,LEN,LWD,IPT)
CALL WIPOUT(ANAME)
CALL CLEAR(ANAME)
CALL GETPNT(ANAME,IPT)
IPT=IGET(ANAME)
JPT=IPT2(IPT,N4,N8)
NEXT=ILAST(DUMMY)
NEXT=ILASTB(DUMMY)
NERR=IPTERR(DUMMY)
NNAM=NNAMSF(DUMMY)
MAXWB=MAXWMF(DUMMY)
MAXWB=MAXWBF(DUMMY)
CALL DUMP(ANAME,IFORMT)
CALL REDEF(ANAME,LEN,LWD)
CALL REDEFM(ANAME,LEN,LWD,IPT)
CALL REDEFB(ANAME,LEN,LWD,IPT)
CALL PURGE(NEXT)
CALL PURGEB(NEXT)
CALL STATUS
```

BLK	LABELED COMMON VARIABLE USED AS A REFERENCE LOCATION FOR THE FCM CONTAINER. BLK IS A LONG WORD VARIABLE.
MAXSIZ	FCM CONTAINER SIZE (IN 8-BYTE WORDS ON SHORT-WORD MACHINES)
IPRINT	PRINT FLAG. 0/1/2/3 = NO PRINT/DUMPS ONLY/ TRACE ONLY/DUMPS AND TRACE
MAXBLK	ECM CONTAINER SIZE
ANAME	DATA ARRAY NAME
LEN	DATA ARRAY LENGTH
LWD	DATA ARRAY WORD LENGTH. 4/8 = SHORT WORD/LONG WORD
IPT	DATA ARRAY POINTER. ARRAY STARTS AT BLK(IPT)

(continued)

JPT	POINTER TO A SUBARRAY. SUBARRAY STARTS AT IBLK(JPT) WHERE IBLK IS A SHORT WORD VARIABLE EQUIVALENCED WITH BLK
N4	NO. OF SHORT WORD DATA PRECEDING SUBARRAY IN ARRAY
N8	NO. OF LONG WORD DATA PRECEDING SUBARRAY IN ARRAY
NEXT	FIRST AVAILABLE CONTAINER LOCATION POINTER
DUMMY	DUMMY VARIABLE
NNAM	NO. OF NAMES IN THE BPOINTER TABLE
MAXWM	MAXIMUM FCM CONTAINER USED
MAXWB	MAXIMUM ECM CONTAINER USED
IFORMT	DUMP FORMAT. 1/2/3 = INTEGER/REAL/BCD

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
==

NOUT	LOGICAL UNIT NO. OF OUTPUT PRINT FILE

COMMON BLOCKS.
=============

```
      COMMON/IOPUT/NIN,NOUT,NOUT2
      COMMON/LOCATE/LSTM,MAXM,LSTB,MAXB,INDXM,INDXB,IPRINT
      COMMON/TABLES/NAMLST(200),LENLST(200),IPTLST(200),LEN(200),
                   MLT(200),NNAMS
      COMMON/PTERR/NPTERR
      COMMON/LCMSIZ/ILOCM,ILOCB,MAXBLK,MAXFM,MAXFB,MULT,MAXNAM,
                   MULT3,JLOCM,JLOCB,MAXWM,MAXWB
      COMMON/BFLAGS/IFLAG(200)
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. POINTR, PUTPNT, BULK, FREE, WIPOUT, GETPNT, IGET, IPT2, PUTM, IPTERR, ILAST, REDEF, REDEFM, PURGE, STATUS, PRTI1, PRTI2, PRTR1, PRTR2, PRTECM, ECZERO
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(POINTR)
========================

RELATED MEMBERS. IGTLCM
===============

REFERENCES. L.C. JUST, H. HENRYSON, II, A.S. KENNEDY, S.D. SPARCK, B.J. TOPPEL, AND P.M. WALKER, THE SYSTEM ASPECTS AND INTERFACE DATA SETS OF THE ARGONNE REACTOR COMPUTATION (ARC) SYSTEM, ANL-7711, ARGONNE NATIONAL LABORATORY (1971).
==========

SOURCE MEMBER NAME. REED
=================

USER ENTRY POINTS. REED, RITE
=================

FUNCTION. REED/RITE HANDLES DATA TRANSFER BETWEEN FAST CORE
======== (FCM) AND SEQUENTIAL DATA SETS ACCORDING TO CCCC
          SPECIFICATIONS.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL REED(LUN,IREC,B,NWDS,MODE)
      CALL RITE(LUN,IREC,B,NWDS,MODE)

           LUN     FILE REFERENCE NO.
           IREC    RECORD NUMBER, =0 FOR REWIND
           B       FCM ARRAY
           NWDS    NO. OF SINGLE-PRECISION WORDS TO BE TRANSFERRED
           MODE    0/1/2.  COMPLETE I/O BEFORE RETURN/START I/O BEFORE
                   RETURN/COMPLETE I/O STARTED BY PREVIOUS CALL
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
===

```
           NOUT    LOGICAL UNIT NO. OF OUTPUT PRINT FILE
```

COMMON BLOCKS.
=============

```
      COMMON/IOPUT/NIN,NOUT,NOUT2
      COMMON/INITIO/NREC(99)
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. VARIOUS SEQUENTIAL DATA SETS
================

ACCOMPANYING SUBPROGRAMS. ZEROIO
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SEGLIB(REED)
========================

RELATED MEMBERS. LRED
===============

REFERENCES. R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
==========   FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
             LOS ALAMOS NATIONAL LABORATORY (1977).

SOURCE MEMBER NAME. REEDSIO

USER ENTRY POINTS. REED, RITE

FUNCTION.

REED/RITE HANDLES DATA TRANSFER BETWEEN FAST CORE (FCM) AND SEQUENTIAL DATA SETS ACCORDING TO CCCC SPECIFICATIONS.

THIS VERSION OF REED INCLUDES THE ANL SIO ACCESS OPTION FOR IBM MACHINES. SIO IS USED ONLY FOR THOSE FILES WITH RECFM=U.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.

```
      CALL REED(LUN,IREC,B,NWDS,MODE)
      CALL RITE(LUN,IREC,B,NWDS,MODE)
```

LUN	FILE REFERENCE NO.
IREC	RECORD NUMBER, =0 FOR REWIND
B	FCM ARRAY
NWDS	NO. OF SINGLE-PRECISION WORDS TO BE TRANSFERRED
MODE	0/1/2. COMPLETE I/O BEFORE RETURN/START I/O BEFORE RETURN/COMPLETE I/O STARTED BY PREVIOUS CALL

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.

NOUT	LOGICAL UNIT NO. OF OUTPUT PRINT FILE

COMMON BLOCKS.

```
      COMMON/IOPUT/NIN,NOUT,NOUT2
      COMMON/INITIO/NREC(99)
```

LANGUAGE. FORTRAN

FILES REFERENCED. VARIOUS SEQUENTIAL DATA SETS

ACCOMPANYING SUBPROGRAMS. SIOERR, SIOWU6, ZEROIO

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(REED)

(continued)

RELATED MEMBERS. LRED, SIOASM, SIOSUB
==============

REFERENCES. R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
==========   FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
             LOS ALAMOS NATIONAL LABORATORY (1977).

SOURCE MEMBER NAME. SCANARC
=================

USER ENTRY POINTS. SCAN
=================

FUNCTION. THIS IS AN INTERFACE SUBROUTINE WHICH LINKS TO THE
========= SYS001 AND SYS002 MODULES TO EXECUTE THE SCAN SUBROUTINE.

SCAN READS THE ENTIRE CARD-IMAGE INPUT FILE (LOGICAL UNIT NUMBER NIN) AND SPOOLS THE CARD IMAGES ONTO A FILE NAMED ARC (IF ARC IS NOT A FILE NAME IN THE SEEK TABLE, THE DEFAULT LOGICAL UNIT NUMBER IS 9). AT THE SAME TIME SCAN LOOKS FOR THE BLOCK KEYWORD (BLOCK=) AND DEFINES THE VARIABLES BLKNAM,IBLTAB AND NBLOCK IN /STFARC/.

THIS INTERFACE SUBROUTINE SHOULD BE USED IN A MODULAR ENVIRONMENT. THE VERSION IN C116.CCCC.SEGLIB(ARCBCD) (SEE SOURCE MEMBER ARCBCD) SHOULD BE USED IN A STANDALONE ENVIRONMENT AND FOR EXPORT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL SCAN
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
==

NOUT	LOGICAL UNIT NUMBER OF OUTPUT PRINT FILE.
NOUT2	LOGICAL UNIT NUMBER OF AUXILIARY OUTPUT PRINT FILE.
NIN	INPUT CARD FILE LOGICAL UNIT NUMBER. APPLIES FOR CALLS TO SCAN ONLY.

COMMON BLOCKS.
=============

```
      COMMON/STFARC/STFNAM,BLKNAM(50),IBLTAB(3,50),NBLOCK,NRET
      COMMON/IOPUT/NIN,NOUT,NOUT2
```

LANGUAGE. FORTRAN
=========

FILES REFERENCED. INPUT CARD FILE, SPOOLED INPUT FILE (THE FILE NAMED
================ ARC, OR LUN 9), OUTPUT FILES NOUT AND NOUT2.

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY. C116.CCCC.SYSLIB(SCAN)
=======

(continued)

RELATED MEMBERS. ARCBCD, SYS001, SYS002
==============

REFERENCES. L.C. JUST, H. HENRYSON, II, A.S. KENNEDY, S.D. SPARCK,
========== B.J. TOPPEL, AND P.M. WALKER, THE SYSTEM ASPECTS AND
INTERFACE DATA SETS OF THE ARGONNE REACTOR COMPUTATION
(ARC) SYSTEM, ANL-7711, ARGONNE NATIONAL LABORATORY
(1971).

SOURCE MEMBER NAME. SECNDARC
=================

USER ENTRY POINTS. SECOND
=================

FUNCTION. THIS IS AN INTERFACE SUBROUTINE WHICH LINKS TO THE
======== SYS005 MODULE TO EXECUTE THE SUBROUTINE SECOND.

SECOND RETURNS ELAPSED CP TIME.

THIS INTERFACE SUBROUTINE SHOULD BE USED IN A MODULAR ENVIRONMENT. SEE MEMBER SECOND FOR THE STAND-ALONE VERSION.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL SECOND(ELAPSE)

           ELAPSE  ELAPSED CP TIME (SEC)
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(SECOND)
========================

RELATED MEMBERS. SECOND, TIMER
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. SECOND
==================

USER ENTRY POINTS. SECOND
=================

FUNCTION. SECOND RETURNS ELAPSED CP TIME. THIS ROUTINE IS FORTRAN
======== CALLABLE.

THIS SUBROUTINE IS USED IN A STAND-ALONE ENVIRONMENT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===

```
      CALL SECOND(ELAPSE)

           ELAPSE  ELAPSED CP TIME (SEC)
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
===

COMMON BLOCKS. NONE
=============

LANGUAGE. IBM ASSEMBLER
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SEGLIB(SECOND)
========================

RELATED MEMBERS. SECNDARC, TIMER, SYS005
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. SEEK
==================

USER ENTRY POINTS. SEEK
=================

FUNCTION. SUBROUTINE SEEK PERFORMS FILE MANAGEMENT SERVICES
======== ACCORDING TO CCCC SPECIFICATIONS AS DOCUMENTED IN LA-6941-MS. FOR THE MOST PART IT IS CALLED FROM APPLICATIONS PROGRAMS TO ESTABLISH THE LOGICAL UNIT NUMBERS OF NAMED DATA SETS.

THIS VERSION OF SEEK SHOULD BE USED IN A STANDALONE ENVIRONMENT AND FOR EXPORT. THE INTERFACE SUBROUTINE VERSION IN C116.CCCC.SYSLIB (SEE SOURCE MEMBER SEEKARC) SHOULD BE USED IN A MODULAR ENVIRONMENT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL SEEK(FILE,IVER,NREF,ISEEK)

           FILE    THE HOLLERITH NAME OF A DATA SET
           IVER    DATA SET VERSION NUMBER
           NREF    DATA SET REFERENCE (LOGICAL UNIT) NUMBER
           ISEEK   0/1/2/3/4/5. PRIOR TO READING A FILE/PRIOR TO
                   WRITING A FILE/WRAP-UP/INITIALIZATION/
                   DELETE FILE/RETURN FILE NAME GIVEN NREF
```

INITIALIZATION OF SEEK (ISEEK=3) IS PERFORMED IN THE FOLLOWING MANNER. FILE IS AN ARRAY OF NAMES, AND SEEK ASSIGNS A LOGICAL UNIT NUMBER TO EACH NAME. THE UNIT NUMBERS ASSIGNED ARE THOSE INPUT IN THE ARRAY NREF - LOGICAL UNIT NUMBER NREF(I) IS ASSIGNED TO THE FILE NAME FILE(I). IF NREF(1) IS ZERO THEN LOGICAL UNIT NUMBERS ARE ASSIGNED SEQUENTIALLY STARTING AT FILE IOFF (SET TO 10) PLUS 1. A MAXIMUM OF 89 FILE NAMES IS PERMITTED, AND THE LIST IS TERMINATED BY A FILE NAME $. THE FIRST TIME A PARTICULAR NAME APPEARS IN THE FILE ARRAY IT IS ASSIGNED VERSION NUMBER 1. THE SECOND OCCURANCE IS VERSION 2, ETC.

IF FILE=6HCHANGE TWO FILE REFERENCE NUMBERS INPUT IN NREF AND ISEEK ARE INTERCHANGED WITH RESPECT TO THEIR NAMES AND VERSION NUMBERS.

```
RETURN ERROR CONDITIONS:
   NREF=0, SEEK TABLE NOT INITIALIZED OR FILE NAME NOT IN
           THE SEEK TABLE
       =-1, FILE HAS NOT BEEN INITIALIZED BY A CALL TO SEEK
           WITH ISEEK=1
```

(continued)

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
==

NOUT	LOGICAL UNIT NUMBER OF OUTPUT FILE

COMMON BLOCKS.
=============

```
      COMMON IOPUT/NIN,NOUT,NOUT2/
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SEGLIB(SEEK)
========================

RELATED MEMBERS. SEEKARC, SYS003
===============

REFERENCES. R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
========== FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
LOS ALAMOS NATIONAL LABORATORY (1977).

SOURCE MEMBER NAME. SEEKARC
=================

USER ENTRY POINTS. SEEK
=================

FUNCTION. THIS IS AN INTERFACE SUBROUTINE WHICH LINKS TO THE SYS003 MODULE TO EXECUTE THE SEEK SUBROUTINE.
========

SUBROUTINE SEEK PERFORMS FILE MANAGEMENT SERVICES ACCORDING TO CCCC SPECIFICATIONS AS DOCUMENTED IN LA-6941-MS. FOR THE MOST PART IT IS CALLED FROM APPLICATIONS PROGRAMS TO ESTABLISH THE LOGICAL UNIT NUMBERS OF NAMED DATA SETS.

THIS INTERFACE SUBROUTINE SHOULD BE USED IN A MODULAR ENVIRONMENT. THE VERSION IN C116.CCCC.SEGLIB (SEE SOURCE MEMBER SEEK) SHOULD BE USED IN A STANDALONE ENVIRONMENT AND FOR EXPORT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===

```
      CALL SEEK(FILE,IVER,NREF,ISEEK)
```

FILE	THE HOLLERITH NAME OF A DATA SET
IVER	DATA SET VERSION NUMBER
NREF	DATA SET REFERENCE (LOGICAL UNIT) NUMBER
ISEEK	0/1/2/3/4/5. PRIOR TO READING A FILE/PRIOR TO WRITING A FILE/WRAP-UP/INITIALIZATION/ DELETE FILE/RETURN FILE NAME GIVEN NREF

INITIALIZATION OF SEEK (ISEEK=3) IS PERFORMED IN THE FOLLOWING MANNER. FILE IS AN ARRAY OF NAMES, AND SEEK ASSIGNS A LOGICAL UNIT NUMBER TO EACH NAME. THE UNIT NUMBERS ASSIGNED ARE THOSE INPUT IN THE ARRAY NREF - LOGICAL UNIT NUMBER NREF(I) IS ASSIGNED TO THE FILE NAME FILE(I). IF NREF(1) IS ZERO THEN LOGICAL UNIT NUMBERS ARE ASSIGNED SEQUENTIALLY STARTING AT FILE IOFF (SET TO 10) PLUS 1. A MAXIMUM OF 89 FILE NAMES IS PERMITTED, AND THE LIST IS TERMINATED BY A FILE NAME $. THE FIRST TIME A PARTICULAR NAME APPEARS IN THE FILE ARRAY IT IS ASSIGNED VERSION NUMBER 1. THE SECOND OCCURANCE IS VERSION 2, ETC.

(continued)

```
            IF FILE=6HCHANGE TWO FILE REFERENCE NUMBERS INPUT IN
            NREF AND ISEEK ARE INTERCHANGED WITH RESPECT TO THEIR
            NAMES AND VERSION NUMBERS.

            RETURN ERROR CONDITIONS:
               NREF=0,  SEEK TABLE NOT INITIALIZED OR FILE NAME NOT IN
                        THE SEEK TABLE
                   =-1, FILE HAS NOT BEEN INITIALIZED BY A CALL TO SEEK
                        WITH ISEEK=1

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
=================================================================

            NOUT    LOGICAL UNIT NUMBER OF OUTPUT FILE

COMMON BLOCKS.
=============

      COMMON IOPUT/NIN,NOUT,NOUT2/

LANGUAGE.  FORTRAN
========

FILES REFERENCED.  NONE
================

ACCOMPANYING SUBPROGRAMS.  NONE
========================

LIBRARY, LINK EDIT INPUT.  C116.CCCC.SYSLIB(SEEK)
========================

RELATED MEMBERS.  SEEK, SYS003
===============

REFERENCES.  R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
==========   FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
             LOS ALAMOS NATIONAL LABORATORY (1977).
```

SOURCE MEMBER NAME. SEKBCD
==================

USER ENTRY POINTS. SEKBCD
=================

FUNCTION. SEKBCD IS AN INTERFACE BETWEEN THE FILE REFERENCE NUMBER
======== RETURNED BY SEEK FOR A BCD FILE AND FORTRAN READS AND WRITES FOR THAT FILE. THE USE OF THIS ROUTINE (OR SOMETHING LIKE IT) IS REQUIRED ON CERTAIN SYSTEMS (E.G. LANL) WHERE THE SEEK REFERENCE NUMBER AND THE LOGICAL UNIT NUMBER USED IN FORTRAN I/O ARE NOT THE SAME. SEKBCD IS ALSO USED TO REWIND BCD FILES.

SEKBCD COMBINES CALLS TO SEEK AND SEKPHL AND PROVIDES AN ALTERNATIVE TO USING SEKPHL DIRECTLY.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===

```
      CALL SEKBCD(FILE,IVER,LUN,MODE,NREF)

           FILE    THE HOLLERITH NAME OF A DATA SET
           IVER    DATA SET VERSION NUMBER
           LUN     LOGICAL UNIT NUMBER FOR FORTRAN I/O (RETURNED)
           MODE    0/1, GIVEN NREF RETURN LUN/REWIND FILE NREF
           NREF    FILE REFERENCE NUMBER FROM SEEK (RETURNED)
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
========

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(SEKBCD)
========================

RELATED MEMBERS. SEEK,SEKPHL
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. SEKPHL
==================

USER ENTRY POINTS. SEKPHL
=================

FUNCTION. SEKPHL IS AN INTERFACE BETWEEN THE FILE REFERENCE NUMBER RETURNED BY SEEK FOR A BCD FILE AND FORTRAN READS AND WRITES FOR THAT FILE. THE USE OF THIS ROUTINE (OR SOMETHING LIKE IT) IS REQUIRED ON CERTAIN SYSTEMS (E.G. LANL) WHERE THE SEEK REFERENCE NUMBER AND THE LOGICAL UNIT NUMBER USED IN FORTRAN I/O ARE NOT THE SAME. SEKPHL IS ALSO USED TO REWIND BCD FILES.
========

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL SEKPHL(NREF,LUN,MODE)

           NREF    FILE REFERENCE NUMBER FROM SEEK (INPUT)
           LUN     LOGICAL UNIT NUMBER FOR FORTRAN I/O (RETURNED)
           MODE    0/1, GIVEN NREF RETURN LUN/REWIND FILE NREF
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(SEKPHL)
========================

RELATED MEMBERS. SEEK
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. SIO
================

USER ENTRY POINTS. RECFM, SIO
================

FUNCTION. SIO IS A FORTRAN-CALLABLE PROGRAM PACKAGE WHICH PERFORMS UNBUFFERED, RANDOM ACCESS, ASYNCHRONOUS I/O FOR FILES ON DIRECT ACCESS DEVICES. SIO PROVIDES AN EFFICIENT ACCESS METHOD FOR SCRATCH FILES WHICH CONTAIN LARGE (.GT. 1 TRACK) LOGICAL RECORDS. RECFM RETURNS THE RECFM PARAMETER DEFINED ON A DD CARD.
========

THESE ROUTINES ARE FOR IBM MACHINES ONLY.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL RECFM(LUN,ITYPE)
      CALL SIO(IOC,LUN,IBUF,IREC,ILEN,IERCDE)
```

LUN	LOGICAL UNIT NO.
ITYPE	RETURNED RECFM FLAG. 1/2/3/4/5/6. U/VS OR VT/ VBS OR VBST/OTHER/NOT A DA DEVICE/NO DD CARD
IOC	1/2/3/4/5. READ/WRITE/WAIT FOR COMPLETION OF IO/ CLOSE FILE/UPDATE INDEX RECORD ON DISK
IBUF	STARTING LOCATION IN CORE FOR DATA TO BE TRANSFERRED
IREC	RECORD NO.
ILEN	NO. OF SINGLE-PRECISION (REAL*4) WORDS TO BE TRANSFERRED
IERCDE	ERROR FLAG

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
===

COMMON BLOCKS. NONE
=============

LANGUAGE. IBM ASSEMBLER
========

FILES REFERENCED. VARIOUS BINARY FILES
================

ACCOMPANYING SUBPROGRAMS. RECFM, SIO, SIOTRC
========================

(continued)

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(SIO)
========================

RELATED MEMBERS. REEDSIO, SIOSUB
===============

REFERENCES. NONE
==========

```
SOURCE MEMBER NAME.  SIOSUB
==================

USER ENTRY POINTS.  SIOSUB
=================

FUNCTION.  THE SUBTASK SIOSUB IS RESPONSIBLE FOR PERFORMING ALL I/O
========   OPERATIONS ON DATA SETS ACCESSED THROUGH SIO, AND DOING
           ALL THAT IS REQUIRED TO INTERFACE WITH THE BSAM SYSTEM
           ROUTINES WHICH ACTUALLY ISSUE THE EXCP'S.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===========================================

           SIOSUB IS CALLED BY THE SUBROUTINE SIO THROUGH AN ATTACH
           MACRO INSTRUCTION.  SIO THEN COMMUNICATES WITH SIOSUB
           THROUGH A COMMON AREA IN SIO AND FILE CONTROL BLOCKS
           (ONE PER OPEN FILE).

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.  NONE
=================================================================

COMMON BLOCKS.  NONE
=============

LANGUAGE.  IBM ASSEMBLER
========

FILES REFERENCED.  VARIOUS BINARY FILES
================

ACCOMPANYING SUBPROGRAMS.  NONE
========================

LIBRARY, LINK EDIT INPUT.  C116.CCCC.SEGLIB(SIOSUB)
========================   C116.CCCC.MODLIB(SIOSUB)

RELATED MEMBERS.  REEDSIO, SIOASM
===============

REFERENCES.  NONE
==========
```

SOURCE MEMBER NAME. SNIFF
==================

USER ENTRY POINTS. SNIFF
=================

FUNCTION. THIS IS A VERSION OF THE SNIFF ROUTINE THAT IS
========= USED IN THE ARC SYSTEM TO PROVIDE FILE MANAGEMENT SERVICES. ITS ONLY FUNCTION IS TO PROVIDE A LINK TO THE SEEK MODULE FOR THOSE OLDER (PRE-CCCC) APPLICATIONS MODULES WHICH WERE CODED WITH CALLS TO SNIFF.

THIS VERSION OF SNIFF COMES IN BOTH SUBROUTINE AND MODULAR FORM.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL LINK(SNIFF,HNAME,NREF,NOP)
      CALL SNIFF(HNAMF,NREF,NOP)

           SNIFF   5HSNIFF
           HNAME   FILE NAME
           NREF    FILE REFERENCE NO.
           NOP     0/1/2.  PRIOR TO READING A FILE/PRIOR TO WRITING
                   A FILE/DELETE A FILE

                   IF HNAME=6HCHANGE TWO FILE REFERENCE NUMBERS INPUT
                   IN NREF AND NOP ARE INTERCHANGED WITH RESPECT TO
                   THEIR NAMES.
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
=========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

(continued)

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(SNIFF)
====================== C116.CCCC.SEGLIB(SNIFF)

```
      //EDT.SYSLMOD DD DSN=C116.CCCC.MODLIB(SNIFF),DISP=(OLD,KEEP)
      //EDT.SEGLIB DD DSN=C116.CCCC.SEGLIB,DISP=SHR
      //EDT.SYSIN DD *
       ENTRY SNIFF
       INCLUDE SEGLIB(SNIFF)
```

RELATED MEMBERS. SEEK
================

REFERENCES. NONE
===========

SOURCE MEMBER NAME. SPACE
=================

USER ENTRY POINTS. SPACE
=================

FUNCTION. SPACE IS A UTILITY ROUTINE USED TO SKIP PAST CARD IMAGE
======== RECORDS IN A BCD DATA SET. THERE MUST BE SEPARATE CALLS FOR EACH BATCH OF CARDS WITH A PARTICULAR CARD TYPE NUMBER.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL SPACE(IRECNO,NUMCDS,ITYPNO,NWHERE)

           IRECNO  LOGICAL UNIT NO.
           NUMCDS  NO. OF CARDS TO BE SKIPPED
           ITYPNO  CARD TYPE NO. OF CARDS BEING SKIPPED
           NWHERE  RETURN ERROR FLAG.  1 = BAD CARD TYPE NO.
                   ENCOUNTERED
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
===

```
           NOUT    OUTPUT PRINT FILE LOGICAL UNIT NUMBER
           NOUT2   AUXILIARY OUTPUT PRINT FILE LOGICAL UNIT NUMBER
```

COMMON BLOCKS.
=============

```
      COMMON/IOPUT/NIN.NOUT,NOUT2
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. VARIOUS BCD DATASETS
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(SPACE)
========================

RELATED MEMBERS. NONE
===============

REFERENCES. NONE
==========

```
SOURCE MEMBER NAME.  SQUEZE
=================

USER ENTRY POINTS.  SQUEZE, GOWEST
=================

FUNCTION.  SQUEZE LEFT JUSTIFIES HOLERITH (A8) WORDS AND SQUEEZES OUT
========   IMBEDDED BLANKS.  GOWEST LEFT JUSTIFIES BUT RETAINS IMBEDDED
           BLANKS.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==========================================

      CALL SQUEZE(WORD,NUM)
      CALL GOWEST(WORD,NUM)

            WORD    AN ARRAY OF HOLERITH WORDS (A8)
            NUM     THE NUMBER OF HOLERITH WORDS (A8) IN THE WORD ARRAY

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.  NONE
================================================================

COMMON BLOCKS.  NONE
=============

LANGUAGE.  FORTRAN
========

FILES REFERENCED.  NONE
================

ACCOMPANYING SUBPROGRAMS.  NONE
========================

LIBRARY, LINK EDIT INPUT.  C116.CCCC.SYSLIB(SQUEZE)
========================

RELATED MEMBERS.  NONE
===============

REFERENCES.  NONE
==========
```

SOURCE MEMBER NAME. SRLAB
==================

USER ENTRY POINTS. SRLAB
=================

FUNCTION. SRLAB SEARCHES THE FIRST LEN ELEMENTS OF THE LIST HNAME2(OR MAX ELEMENTS IF MAX.LT.LEN) FOR THE LABEL HNAME1. IF HNAME1 IS FOUND ITS POSITION IS RETURNED IN IPOS. IF HNAME1 IS NOT FOUND AND LEN.LT.MAX HNAME1 IS ADDED TO THE LIST, ITS POSITION IS RETURNED IN IPOS AND LEN IS INCREMENTED BY ONE. IF HNAME1 IS NOT FOUND AND LEN.GE.MAX, IPOS IS SET TO ZERO.
========

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===

```
      CALL SRLAB(HNAME1,HNAME2,LEN,MAX,IPOS)
```

HNAME1	SEARCH LABEL
HNAME2	ARRAY TO BE SEARCHED
LEN	NO. OF ELEMENTS OF HNAME2 FILLED
MAX	MAX. NO. OF ELEMENTS OF HNAME2 PERMITTED
IPOS	POSITION OF HNAME1 IN HNAME2 ARRAY

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
===

COMMON BLOCKS. NONE
=============

LANGUAGE. FORTRAN
=========

FILES REFERENCED. NONE
=================

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(SRLAB)
========================

RELATED MEMBERS. NONE
================

REFERENCES. NONE
===========

SOURCE MEMBER NAME.   STUFFARC
=================

USER ENTRY POINTS.   STUFF
=================

FUNCTION.   THIS IS AN INTERFACE SUBROUTINE WHICH LINKS TO THE
========   SYS001 AND SYS002 MODULES TO EXECUTE THE STUFF SUBROUTINE.

STUFF PROCESSES THE INPUT DATA ASSOCIATED WITH THE NEXT UNPROCESSED DATA BLOCK NAMED STFNAM (STFNAM IS A VARIABLE IN /STFARC/). STUFF CREATES OR MODIFIES BCD DATASETS AND SETS THE VARIABLE NRET IN /STFARC/.

```
NRET = 1, THE NEXT DATA BLOCK NAMED STFNAM WAS PROCESSED.
      -3, THERE IS NO DATA BLOCK NAMED STFNAM IN THE INPUT.
      -5, ALL DATA BLOCKS NAMED STFNAM WERE PROCESSED PRIOR
          TO THIS CALL TO STUFF.
```

THIS INTERFACE SUBROUTINE SHOULD BE USED IN A MODULAR ENVIRONMENT. THE VERSION IN C116.CCCC.SEGLIB (SEE SOURCE MEMBER ARCBCD) SHOULD BE USED IN A STANDALONE ENVIRONMENT AND FOR EXPORT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      CALL STUFF
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.
===

NOUT	LOGICAL UNIT NUMBER OF OUTPUT PRINT FILE.
NOUT2	LOGICAL UNIT NUMBER OF AUXILIARY OUTPUT PRINT FILE.
STFNAM	NAME OF THE DATA BLOCK TO BE PROCESSED BY STUFF.

SPECIAL NOTE.   SCAN MUST BE CALLED BEFORE STUFF TO SET THE VARIABLES BLKNAM, IBLTAB AND NBLOCK IN /STFARC/.

COMMON BLOCKS.
=============

```
      COMMON/STFARC/STFNAM,BLKNAM(50),IBLTAB(3,50),NBLOCK,NRET
      COMMON/IOPUT/NIN,NOUT,NOUT2
```

LANGUAGE.   FORTRAN
========

(continued)

FILES REFERENCED. VARIOUS BCD FILES
================

ACCOMPANYING SUBPROGRAMS. STUFF1
=========================

LIBRARY. C116.CCCC.SYSLIB(STUFF)
=======

RELATED MEMBERS. CODECD, SCAN, SYS002
===============

REFERENCES. L.C. JUST, H. HENRYSON, II, A.S. KENNEDY, S.D. SPARCK,
==========   B.J. TOPPEL, AND P.M. WALKER, THE SYSTEM ASPECTS AND
INTERFACE DATA SETS OF THE ARGONNE REACTOR COMPUTATION
(ARC) SYSTEM, ANL-7711, ARGONNE NATIONAL LABORATORY
(1971).

```
SOURCE MEMBER NAME.  SYS001
==================

USER ENTRY POINTS.  SYS001
=================

FUNCTION.  SYS001 IS AN INTERMEDIATE STEP IN CALLS TO SCAN AND STUFF
========   IN A MODULAR ENVIRONMENT.  SYS001 IS A REUSABLE MODULE
           WHICH PROVIDES A PERMANENT, COMMON AREA OF STORAGE FOR THE
           SCAN/STUFF TABLES.

           SYS001 IS CALLED FROM APPLICATIONS PROGRAMS THROUGH THE
           INTERFACE SUBROUTINES SCAN AND STUFF.  IN TURN, SYS001
           CALLS THE NON-REUSABLE MODULE SYS002 WHICH EXECUTES THE SCAN
           AND STUFF SUBROUTINES.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==========================================

      CALL LINK(SYS001,SUBNAM,BLKNAM,NRET,NIN,NOUT,NOUT2)

           SYS001  6HSYS001
           SUBNAM  4HSCAN/5HSTUFF.  CALL TO SCAN/STUFF.
           BLKNAM  BLOCK NAME.  STUFF ONLY.
           NRET    1, BLOCK=STFNAM FOUND AND PROCESSED.
                   -3, BLOCK=STFNAM NOT FOUND.
                   -5, ALL BLOCK=STFNAM DATA HAS BEEN PROCESSED DURING
                       PREVIOUS CALLS TO STUFF.
           NIN     INPUT CARD FILE LOGICAL UNIT NUMBER.
           NOUT    LOGICAL UNIT NUMBER OF OUTPUT PRINT FILE.
           NOUT2   LOGICAL UNIT NUMBER OF AUXILIARY OUTPUT PRINT FILE.

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM.  NONE
================================================================

COMMON BLOCKS.
=============

      COMMON/STFARC/STFNAM,BLNAM(50),IBLTAB(3,50),NBLOCK,NRET
      COMMON/IOPUT/NIN,NOUT,NOUT2

LANGUAGE.  FORTRAN
========

FILES REFERENCED.  NONE
================

ACCOMPANYING SUBPROGRAMS.  NONE
========================
```

(continued)

LIBRARY, LINK EDIT INPUT. C116.CCCC.SEGLIB(SYS001)
========================

```
      //EDT.SYSLMOD DD DSN=C116.CCCC.MODLIB(SYS001),DISP=(OLD,KEEP)
      //EDT.SEGLIB DD DSN=C116.CCCC.SEGLIB,DISP=SHR
      //EDT.SYSIN DD *
       ENTRY SYS001
       INCLUDE SEGLIB(SYS001)
      /*
```

RELATED MEMBERS. SYS002, ARCBCD, SCANARC, STUFFARC
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. SYS002
=================

USER ENTRY POINTS. MAIN
================

FUNCTION. SYS002 IS THE MODULE DRIVER FOR SUBROUTINES SCAN AND STUFF IN A MODULAR ENVIRONMENT. SYS002 IS NORMALLY ACCESSED FROM THE REUSABLE MODULE SYS001.
========

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
===

```
      CALL LINK(SYS002,SUBNAM,STFNAM,BLNAM,IBLTAB,NBLOCK,NRET,
     X NIN,NOUT,NOUT2)
```

SYS002	6HSYS002
SUBNAM	4HSCAN/5HSTUFF. CALL SCAN/STUFF.
STFNAM	NAME OF THE DATA BLOCK TO BE PROCESSED BY STUFF.
BLNAM	BLOCK NAME ARRAY. DIMENSIONED BLNAM(50).
IBLTAB	BLOCK TABLE ARRAY. DIMENSIONED IBLTAB(3,50).
NBLOCK	NO. OF DATA BLOCKS. ESTABLISHED BY SCAN.
NRET	1, BLOCK=STFNAM FOUND AND PROCESSED. -3, BLOCK=STFNAM NOT FOUND. -5, ALL BLOCK=STFNAM DATA HAS BEEN PROCESSED DURING PREVIOUS CALLS TO STUFF.
NIN	INPUT CARD FILE LOGICAL UNIT NUMBER.
NOUT	LOGICAL UNIT NUMBER OF OUTPUT PRINT FILE.
NOUT2	LOGICAL UNIT NUMBER OF AUXILIARY OUTPUT PRINT FILE.

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS.
=============

```
      COMMON/STFARC/STFNAM,BLNAM(50),IBLTAB(3,50),NBLOCK,NRET
      COMMON/IOPUT/NIN,NOUT,NOUT2
```

LANGUAGE. FORTRAN
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. NONE
========================

(continued)

LIBRARY, LINK EDIT INPUT. C116.CCCC.SEGLIB(SYS002)
========================

```
      //EDT.SYSLMOD DD DSN=C116.CCCC.MODLIB(SYS002),DISP=(OLD,KEEP)
      //EDT.SEGLIB DD DSN=C116.CCCC.SEGLIB,DISP=SHR
      //EDT.SYSIN DD *
       ENTRY SYS002
       INCLUDE SEGLIB(SYS002,ARCBCD)
      /*
```

RELATED MEMBERS. SYS001, ARCBCD, SCANARC, STUFFARC
===============

REFERENCES. NONE
==========

SOURCE MEMBER NAME. SYS003

USER ENTRY POINTS. MAIN

FUNCTION.

SYS003 IS THE MODULE DRIVER FOR SUBROUTINE SEEK IN A MODULAR ENVIRONMENT. THE MODULE SYS003 MAY BE ACCESSED THROUGH A CALL LINK (SEE BELOW) OR BY USE OF THE INTERFACE SUBROUTINE SEEK (SEE SEEKARC).

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.

```
      CALL LINK(SYS003,FILE,IVER,NREF,ISEEK,NOUT,NOUT2)
```

SYS003	6HSYS003
FILE	THE HOLLERITH NAME OF A DATA SET
IVER	DATA SET VERSION NUMBER
NREF	DATA SET REFERENCE (LOGICAL UNIT) NUMBER
ISEEK	0/1/2/3/4/5. PRIOR TO READING A FILE/PRIOR TO WRITING A FILE/WRAP-UP/INITIALIZATION/ DELETE FILE/RETURN FILE NAME GIVEN NREF
NOUT	LOGICAL UNIT NO. OF OUTPUT PRINT FILE
NOUT2	LOGICAL UNIT NO. OF AUXILIARY OUTPUT PRINT FILE

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE

COMMON BLOCKS. NONE

LANGUAGE. FORTRAN

FILES REFERENCED. NONE

ACCOMPANYING SUBPROGRAMS. NONE

(continued)

LIBRARY, LINK EDIT INPUT. C116.CCCC.SEGLIB(SYS003)
========================

```
      //EDT.SYSLMOD DD DSN=C116.CCCC.MODLIB(SYS003),DISP=(OLD,KEEP)
      //EDT.SEGLIB DD DSN=C116.CCCC.SEGLIB,DISP=SHR
      //EDT.SYSIN DD *
       ENTRY SYS003
       INCLUDE SEGLIB(SYS003,SEEK)
      /*
```

RELATED MEMBERS. SEEK, SEEKARC
================

REFERENCES. NONE
===========

SOURCE MEMBER NAME. SYS004

USER ENTRY POINTS. MAIN

FUNCTION. SYS004 IS THE MODULE DRIVER FOR SUBROUTINE LINES IN A MODULAR ENVIRONMENT. THE MODULE SYS004 MAY BE ACCESSED THROUGH A CALL LINK (SEE BELOW) OR BY USE OF THE INTERFACE SUBROUTINE LINES (SEE LINESARC).

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.

```
      CALL LINK(SYS004,NUMLIN,IFLAG1,IENTRY,TITLE,NTITLE,TIME,HEAD,
     X NOUT,NOUT2)
```

SYS004	6HSYS004
NUMLIN	GT.0, THE NUMBER OF LINES ABOUT TO BE PRINTED (ON LOGICAL UNIT NOUT FOR LINES, NOUT2 FOR LINES2). EQ.0, SKIP TO THE TOP OF THE NEXT PAGE.
IFLAG1	RETURNED FLAG. 0/1, SAME PAGE/NEW PAGE.
IENTRY	1/2, CALL TO LINES/LINES2.
TITLE	ARRAY OF HOLERITH WORDS USED FOR PAGE HEADING TITLE (A6). TITLE IS PRINTED UP TO A MAXIMUM OF 6 LINES USING 11 WORDS PER LINE.
NTITLE	NO. OF WORDS OF THE ARRAY TITLE TO BE PRINTED. NTITLE IS LESS THAN OR EQUAL TO 66.
TIME	SUBROUTINE TIMER VARIABLES (REAL*8 WORDS).
HEAD	PAGE HEADING LEAD (PRINTED 4A6).
NOUT	LOGICAL UNIT NUMBER OF OUTPUT PRINT FILE.
NOUT2	LOGICAL UNIT NUMBER OF AUXILIARY OUTPUT PRINT FILE.

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE

COMMON BLOCKS.

```
      COMMON /PTITLE/ TITLE(66),TIME(10),HEAD(4),NOUT,NOUT2,NTITLE
```

LANGUAGE. FORTRAN

FILES REFERENCED. NONE

(continued)

ACCOMPANYING SUBPROGRAMS. NONE
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SEGLIB(SYS004)
========================

```
//EDT.SYSLMOD DD DSN=C116.CCCC.MODLIB(SYS004),DISP=(OLD,KEEP)
//EDT.SEGLIB DD DSN=C116.CCCC.SEGLIB,DISP=SHR
//EDT.SYSIN DD *
 ENTRY SYS004
 INCLUDE SEGLIB(SYS004,LINES)
/*
```

RELATED MEMBERS. LINESARC, LINES
================

REFERENCES. NONE
===========

SOURCE MEMBER NAME. SYS005

USER ENTRY POINTS. SYS005

FUNCTION. SYS005 IS THE MODULE DRIVER FOR SUBROUTINE SECOND IN A MODULAR ENVIRONMENT.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.

```
      CALL LINK(SYS005,ELAPSE)

           SYS005  6HSYS005
           ELAPSE  ELAPSED CP TIME (SEC)
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE

COMMON BLOCKS. NONE

LANGUAGE. FORTRAN

FILES REFERENCED. NONE

ACCOMPANYING SUBPROGRAMS. NONE

LIBRARY, LINK EDIT INPUT. C116.CCCC.SEGLIB(SYS005)

```
      //EDT.SYSLMOD DD DSN=C116.CCCC.MODLIB(SYS005),DISP=(OLD,KEEP)
      //EDT.SEGLIB DD DSN=C116.CCCC.SEGLIB,DISP=SHR
      //EDT.SYSIN DD *
       ENTRY SYS005
       INCLUDE SEGLIB(SYS005,SECOND)
      /*
```

RELATED MEMBERS. TIMER, SECOND, SECNDARC

REFERENCES. NONE

SOURCE MEMBER NAME. TIMER

USER ENTRY POINTS. TIMER

FUNCTION. TIMER IS A GENERAL TIMING ROUTINE SPECIFIED BY THE CCCC. THIS VERSION IS INTENDED FOR BOTH LOCAL AND EXPORT APPLICATIONS.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.

```
      CALL TIMER(I,T)

           I       .LT.0, RETURN ALL ITEMS IN THE T ARRAY
                    0, INITIALIZE
                    1, ELAPSED CP TIME
                    2, TIME LEFT
                    3, ELAPSED PP TIME
                    4, DATE, MM/DD/YY (A8)
                    5, USER ID (A6)
                    6, USER CHARGE NO. (A6)
                    7, USER JOB NAME (A8)
                    8, WALL CLOCK, HHMM.TT (A6)
                   10, WALL CLOCK, HH.MM.TT (A8)
           T        A FULL-WORD ARRAY OF 10 WORDS FOR THE
                    REQUESTED DATA

                    SEE THE SOURCE CODE FOR LISTS OF WHICH OPTIONS
                    ARE AVAILABLE ON DIFFERENT MACHINES
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE

COMMON BLOCKS. NONE

LANGUAGE. FORTRAN

FILES REFERENCED. NONE

ACCOMPANYING SUBPROGRAMS. NONE

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(TIMER)

(continued)

RELATED MEMBERS. TIME, SECOND, SECNDARC
===============

REFERENCES. R. D. O'DELL, STANDARD INTERFACE FILES AND PROCEDURES
========== FOR REACTOR PHYSICS CODES, VERSION IV, LA-6941-MS,
LOS ALAMOS NATIONAL LABORATORY (1977).

SOURCE MEMBER NAME. TIME1
==================

USER ENTRY POINTS. TIME1, CLOCK, DATE1, JOBID
=================

FUNCTION. TIME1 RETURNS THE CURRENT WALL CLOCK TIME IN THE
======== FORMAT HH.MM.SS. CLOCK RETURNS THE WALL CLOCK TIME IN
HUNDRETHS OF A SECOND. DATE1 RETURNS THE CURRENT DATE
IN THE FORMAT MM/DD/YY. JOBID RETURNS THE USER JOB NAME.

THESE ROUTINES ARE FORTRAN CALLABLE.

CALLING SEQUENCE, DEFINITIONS OF ARGUMENTS.
==

```
      WCLK1=TIME1(DUMMY)
      CALL CLOCK(WCLK2)
      DATE=DATE1(DUMMY)
      CALL JOBID(USER)

           WCLK1   WALL CLOCK IN A8 FORMAT, HH.MM.SS.
           DUMMY   DUMMY VARIABLE.
           WCLK2   WALL CLOCK AS FLOATING POINT VARIABLE.
           DATE    DATE IN A8 FORMAT, MM/DD/YY.
           USER    RETURNED JOB NAME (REAL*8).
```

VARIABLES IN COMMON WHICH MUST BE DEFINED IN THE CALLING PROGRAM. NONE
==

COMMON BLOCKS. NONE
=============

LANGUAGE. IBM ASSEMBLER
========

FILES REFERENCED. NONE
================

ACCOMPANYING SUBPROGRAMS. TIME, CLOCK, DATE, JOBID
========================

LIBRARY, LINK EDIT INPUT. C116.CCCC.SYSLIB(TIME1)
========================

RELATED MEMBERS. TIMER
===============

REFERENCES. NONE
==========

Distribution for ANL-83-3

Internal:

P. B. Abramson
C. H. Adams
P. I. Amundson
C. L. Beck
E. S. Beckjord
J. C. Beitel
S. K. Bhattacharyya
H. Bigelow
R. N. Blomquist
M. M. Bretscher
S. B. Brumbach
R. G. Bucher
J. E. Cahalan
S. G. Carpenter
B. R. Chandler
Y. I. Chang
P. J. Collins
R. J. Cornella
D. C. Cutforth
T. A. Daly
J. R. Deen
K. L. Derstine
D. R. Ferguson
K. E. Freese
E. K. Fujita
P. L. Garner
J. M. Gasidlo
E. M. Gelbard
G. L. Grasseschi
G. M. Greenman
H. Henryson
R. Hosteny
H. H. Hummel
R. N. Hwang
R. E. Kaiser
Kalinullah
H. Khalil
R. D. Lawrence
W. K. Lehto
R. M. Lell
L. G. LeSage
J. R. Liaw
M. J. Lineberry
D. J. Malloy
J. E. Matos
H. F. McFarlane
R. D. McKnight
D. Meneghetti
L. E. Meyer
F. Moszur
A. Olson
Y. Orechwa
E. M. Pennington
P. J. Persiani
P. A. Pizzica
R. B. Pond
J. R. Ross
R. R. Rudolph
G. K. Rusch
R. W. Schaefer
J. E. Schofield
D. Shaftman
D. M. Smith
K. S. Smith
J. L. Snelgrove
D. R. Snider
C. G. Stenberg
W. J. Sturm
S. F. Su
C. E. Till
B. J. Toppel
A. Travelli
R. B. Turski
A. J. Ulrich
R. Vilim
D. C. Wade
D. P. Weber
W. L. Woodruff
S. T. Yang
B. S. Yarlagadda
ANL Patent Dept.
ANL Contract File
ANL Libraries (2)
TIS Files (6)
AP Division Files (10)

External:

DOE-TIC, for distribution per UC-79d (123)
Manager, Chicago Operations Office, DOE
Director, Technology Management, DOE-CH
Deputy Asst. Secy., Breeder Reactor Programs, DOE-Wash. (2)
Applied Physics Division Review Committee:
  P. W. Dickson, Jr., Clinch River Breeder Reactor Project, Oak Ridge
  K. D. Lathrop, Los Alamos National Laboratory
  D. A. Meneley, Ontario Hydro
  J. E. Meyer, Massachusetts Inst. Technology
  R. Sher, Stanford U.
  D. B. Wehmeyer, Detroit Edison Co.
  A. E. Wilson, Idaho State U.

Other External:

H. Alter, Office of Breeder Technology, DOE
Advanced Reactor Library, Westinghouse Electric Co., Madison, Pa
M. Becker, Rensselaer Polytechnic Inst.
R. A. Bennett, Westinghouse Hanford Co.
F. W. Brinkley, Los Alamos National Laboratory
S. P. Congdon, General Electric Co.
C. Cowan, General Electric Co.
H. L. Dodds, Technology for Energy Corp.
R. Doncals, Westinghouse Electric Corp.
J. J. Dorning, University of Illinois
M. J. Driscoll, MIT
C. Durston, Combustion Engineering
R. Ehrlich, General Electric Co.
H. Farrar IV, Atomics International
Fast Breeder Dept. Library, General Electric Co.
G. F. Flanagan, Oak Ridge National Laboratory
N. M. Greene, Oak Ridge National Laboratory
D. R. Harris, Rensselaer Polytechnic Inst.
P. B. Hemmig, Reactor Development and Demonstration, DOE
A. F. Henry, MIT
J. Kallfelz, Georgia Institute of Technology
R. Karam, Georgia Institute of Technology
W. Y. Kato, Brookhaven National Laboratory
R. J. LaBauve, Los Alamos National Laboratory
J. Lewellen, Reactor Development and Demonstration, DOE
E. E. Lewis, Northwestern University
M. D. Libby, NUSCO
R. MacFarlane, Los Alamos National Laboratory
F. C. Maienschein, Oak Ridge National Laboratory
D. R. Marr, Los Alamos National Laboratory
D. R. Mathews, GA Technologies
D. R. McCoy, Los Alamos National Laboratory
H. A. Morowitz, Tarzana, CA
J. Naser, Electric Power Research Inst.
R. J. Neuhold, Reactor Development and Demonstration, DOE
R. D. O'Dell, Los Alamos National Laboratory
D. Okrent, University of California
K. Ott, Purdue University
O. Ozer, Electric Power Research Inst.
A. M. Perry, Oak Ridge National Laboratory
C. Porter, Wentinghouse Electric Corp.
J. Prabulos, Combustion Engineering
R. Protsik, General Electric Co.
A. B. Reynolds, University of Virginia
W. A. Rhoades, Oak Ridge National Laboratory
P. Rose, Brookhaven National Laboratory
D. H. Roy, Babcock and Wilcox Co.
R. Schenter, Westinghouse Hanford Co.
P. Soran, Los Alamos National Laboratory
E. R. Specht, Atomics International
S. Stewart, General Electric, Co.
L. E. Strawbridge, Westinghouse Electric Corp.

R. J. Tuttle, Atomics International
D. R. Vondy, Oak Ridge National Laboratory
J. E. Vossahlik, GRP Consulting, Inc.
C. Weisbin, Oak Ridge National Laboratory