LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

ORNL/TM-8736
Distribution Category UC-77

Contract No. W-7405-eng-26

Engineering Physics Division

IMPLEMENTING A MODULAR SYSTEM OF COMPUTER CODES

D. R. Vondy
T. B. Fowler

Date Published: July 1983

Research sponsored by
U.S. DOE Office of
Converter Reactor Deployment

OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37830
operated by
UNION CARBIDE CORPORATION
for the
DEPARTMENT OF ENERGY

TABLE OF CONTENTS

ABSTRACT

A modular computation system has been developed for nuclear reactor core analysis. The codes can be applied repeatedly in blocks without extensive user input data, as needed for reactor history calculations. The primary control options over the calculational paths and task assignments within the codes are blocked separately from other instructions, admitting ready access by user input instruction or directions from automated procedures and promoting flexible and diverse applications at minimum application cost. Data interfacing is done under formal specifications with data files manipulated by an informed manager. This report emphasizes the system aspects and the development of useful capability, hopefully informative and useful to anyone developing a modular code system of much sophistication. Overall, this report in a general way summarizes the many factors and difficulties that are faced in making reactor core calculations, based on the experience of the authors. It provides the background on which work on HTGR reactor physics is being carried out.

ORNL/TM-8736
Distribution Category UC-77

INTERNAL DISTRIBUTION

1. L. S. Abbott
2. R. G. Alsmiller
3. D. E. Bartine
4. B. L. Broadhead
5. D. G. Cacuci
6. R. L. Childs
7. M. E. Emmett
8. G. F. Flanagan
9. T. B. Fowler
10. U. Gat
11. N. M. Greene
12. R. A. Lillie
13. R. E. Maerker

14 18. F. C. Maienschein

19. J. H. Marable
20. B. F. Maskewitz
21. R. W. Peelle
22. L. M. Petrie
23. J. P. Renier
24. W. A. Rhoades
25. D. L. Selby
26. C. O. Slater

27 31. D. R. Vondy

32. C. R. Weisbin
33. B. A. Worley
34. P. W. Dickson, Jr. (Consultant)
35. H. J. C. Kouts (Consultant)
36. W. B. Loewenstein (Consultant)
37. R. Wilson (Consultant)

38 39. Central Research Library

40. Laboratory Records, RC
41. Laboratory Records
42. Y-12 Technical Library
    Document Reference Section
43. ORNL Patent Office
44. EPIC

EXTERNAL DISTRIBUTION

45-46. Director, Division of HTR Development, DOE, Washington, D.C. 20545

47. Office of Assistant Manager for Energy Research & Development, DOE-ORO, Attention: S. W. Ahrends
48. Director, Office of Converter Reactor Deployment, NE-15, DOE, Washington, D.C. 20545
49. G. A. Newby, Director, Division of HTR Development, NE-15, DOE, Washington, D.C. 20545

GA Technologies, P.O. Box 81608, San Diego, CA 92138

50. A. J. Neylan
51. R. K. Lane
52. A. M. Baxter

General Electric, P.O. Box 508, Sunnyvale, CA 94086

53. G. R. Pflasterer
54. C. L. Cohen

55 188. Given distribution as shown in TID-4500, Distribution Category UC-77 - Gas Cooled Reactor Technology

INTRODUCTION

Our capability for performing analysis of a nuclear reactor core is nearly wholly contained in computer codes due to the complexity of the problems. The neutron flux, eigenvalue problem must usually be solved as a part of any problem and this requires a computer calculation. Thus our analysis capability is limited to available programmed, validated, and qualified capability. The more global a problem to be solved, the more sophisticated the computational requirements. For example, a reactor history calculation requires coupled neutronics, exposure calculations, plus the capability to realistically and reliably model fueling, reactivity compensation, and control. The utility of a stand-alone neutronics code without coupled capability for performing exposure (fuel burnup) calculations is severely limited in reactor core performance analysis.

With sophistication in computer code development and implementation has come modularization at the code level (as opposed to the routine level). The primary objectives of modularization are to automate the calculation that involves the use of more than one code and the coupling of the results obtained by one for use by another, to provide for flexibility of the calculational path, and to parallel procedures to make readily available alternative capability for physical or phenomenal modeling. The individual calculational procedures are blocked into distinct code packages that perform specific tasks on demand. A system of codes that may be applied one after the other as ordered by a user in one computer run (with simple system control instructions) is an advance over a group of stand-alone codes. Here, however, interest is in an organized set of codes that generally do not obtain data from the user input stream and thus can in principle be used in any order over and over. In a sophisticated computation system the precise order of use of the codes for a specific problem depends on the calculated results and hence cannot be determined in advance. There are major differences between a stand-alone code and a full member of a sophisticated modular system. Certain requirements for functioning in a specific computation system have to be satisfied to admit membership to a code.

The purpose of this discussion is to make available our experience to others faced with building a modular code system. We do note that at least locally the system of codes we have implemented is in routine use on nearly a daily basis in several projects. Wide use at other installations, as is common of some codes developed, has at least not yet followed. There were some surprises regarding where major difficulties arise. We think we learned some interesting things and this information may prove to be useful to others. Considerable effort in such a project can go into decision making, and this process is apt to get out of hand when a large number of individuals contribute and are involved.

The more complicated the system and the more elaborate the rules, the harder it is to add capability and the higher the cost and the longer it takes, and possibly reliability may suffer. If too simple, the resulting capability is inadequate and there will not be sufficient flexibility to satisfy the needs. Any computation system will fall between those extremes, a good one being the direct result of balancing the various considerations.

Our experience has been with the development and use of the BOLD VENTURE code system[1,2] that contains the VENTURE and VALE diffusion theory neutronics codes.

Capability is indicated in Fig. 1 showing neutronics and related exposure, fuel management, limited thermal hydraulics, perturbation, and sensitivity capability. These are coded in Fortran, except for the resident driver, with machine language minimized. We had found that adequate capability cannot practically be packaged in a single code in the development of the CITATION code[3] oriented to core history calculations. We were certainly influenced by the information available about the Argonne Reactor Calculation development at ANL.[4] Indeed we took advantage of the experience and viewpoints of a number of individuals involved in an interinstallation cooperative physics methods effort under U.S. DOE auspices.[5,6]

We note that technological advances in computing over the last twenty years have only modestly affected what we do: we now solve much larger and more complicated problems, but the analysis challenges remain much the same. Code development has only increased in difficulty due to increase in the complexity without significant technological relief. Only a few years ago we would not have expected to ever work on a modular code system.[6] Certainly much effort along this line is suspect of producing little useful product. The importance we assign to the subject area speaks for itself. Our inability to make much use of the products of others in this area should also speak for itself. When a large community is involved, the impact is nearly overwhelming.[7]

Refer to Fig. 1 for the primary contents of this computation system discussed below.

Driver resident machine language code for bringing code modules into memory and transferring execution to them, but does not exercise primary control regarding the calculational path.

At the start, the Driver moves a block of input data from the user input stream to the device read by the limited number of codes that process user input data, brings the identified Control Module into memory and transfers execution to it. A string of code names is returned to the Driver which it brings into memory in order successively. An error condition is checked by the Driver after each code has been executed and any failure condition passed to the control module for wrap up discontinues the calculational path. Successful completion of the assigned tasks also tells the Driver to bring the Control Module back in for further instructions. The Driver also keeps a block of user input data (that required by one special processor) ready on the device from which this data is read by the codes in advance knowing which modules need input data. (Those codes that read input data are named in the input data and assigned a common number, the codes not being full members of the computation system, except for the Input Processor.) The Driver identifies certain error conditions such as the lack of input data or in some cases its improper arrangement. The lines of user input data are listed on an output device for documentation. After completion of a calculation the Driver will reaccess the Control Module to start another calculation.

Control Module Exercises control over the calculational path based on simple user instructions for successive accesses, has the capability for looping over blocks of codes, and applying initialization and wrap up procedures.

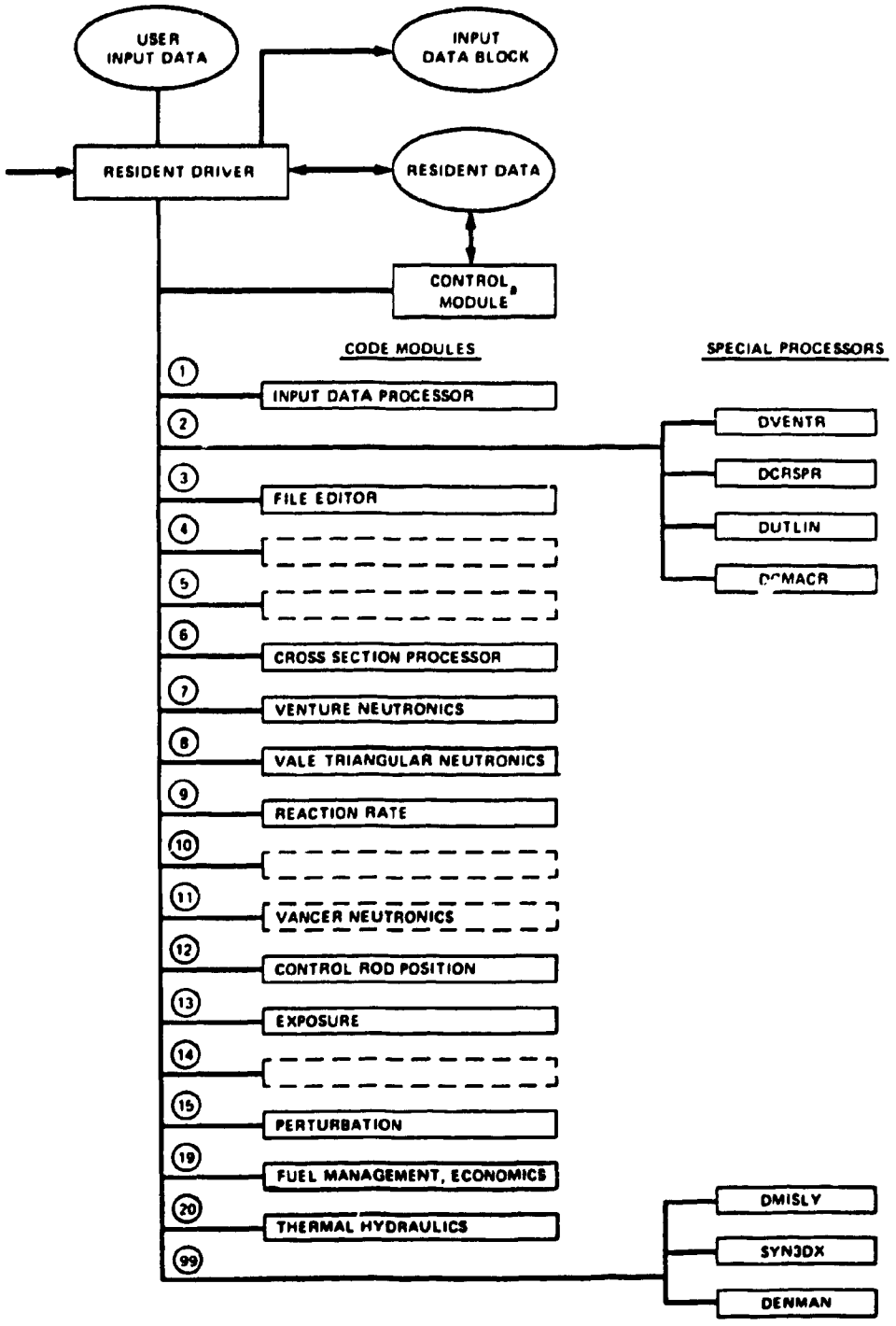

Fig. 1. Components of the Computation System.

The intent was to develop special Control Modules to serve different roles, but the effort on this system has produced only a single Control Module that allows user prescription of a set calculational path. The original plans remain sound and possibly future effort will extend the capability.

Member Code Modules	codes that do not read user input data and thus may be used on demand.
Special Processors	codes that serve special roles and always use data from the user input data stream.

Special needs in a project are often satisfied by adding codes to the system. In this way capability is readily added to perform auxiliary calculations at the end of solving a problem.

Input Data Processor[5]	free form format user input data processing, one-to-one interface data file generation.
File Editor	file listing capability and file copying into a form readable by the Input Processor (binary to formatted).

A typical calculational path would include initial data processing, then successive uses of a neutronics code and the exposure code to follow a reactor history of operation.

THE COMMITMENT AND MANAGEMENT

Beyond any doubt the decision to go modular is a major one requiring a commitment by management, developers and programmers to be successful. Key decisions have to be made from the start and essential requirements satisfied as work progresses. Ignoring these requirements, deferring decisions, or the lack of cooperation toward the objective will compromise the product. A most useful product is the direct consequence of effective management of a development project. Effort on a volunteer basis or without designated individual responsibilities is to be avoided. An honest commitment is required by those involved and others impacted by the product. It is not practical for an individual developer to take this route. Below some development group size the approach is impractical. Still, what others have done may be adopted to make the approach practical with a limited investment.

THE COST

Certain initial support effort is essential that requires more than just casual programming. This cost should be recognized and provision made for the dedication of the necessary effort. The separation of user input processing from primary coding, data management rules, formal file specifications, etc. add development cost. An indirect cost is the deferred return from current effort that should be minimized but must be recognized and

adequately allowed. There are special considerations when the effort is funded by an outside agency, impacting the effort and the decision making, and naturally affecting the product.

The initial cost depends to a large extent on how much use can be made of available capability. We expended at least three man years. This included the work done on interface specifications and free form input data processing, and coding the driver and data management and data handling procedures for opening, reading, and writing files.[2] Expect to consume one man year in support effort at the very least for a new system plus any computer system work required.

Likely most developers of modular systems would like to change things as development progresses, but active use may well prevent any disruptive change. A continuity of use must be maintained with an active system, a commitment required when the computation system becomes actively in routine use by others than the developers.

EFFECTING ANALYSIS CAPABILITY

Decisions must be made that may seriously compromise the ability to perform reliable analysis as a direct consequence of modularization. Methods are chosen for economic and reliability reasons, and interfaces are drawn effecting separation for effective computation reasons. Such action must be taken. But decisions that lead to severe modeling compromises or that eliminate practical calculational methods should be avoided. Compromise should be done for good reason, the consequences recognized, and the effects be subjected to later review to confirm that the choice was sound. Of special concern are the implications of interface data file specifications: if a calculation is to be done using specific data from a file, the modeling is limited to whatever is allowed by the data. Limiting cross section data to isotropic scattering prevents extension to treat anisotropic scattering, as an obvious example. A geometric mesh description that allows only separable orthogonal coordinates for parallel rows of mesh points in two- and three-dimensions is inadequate for other descriptions. Of particular concern are the differences in viewpoints of experts having different experience and backgrounds and the difficulty of accommodating these without compromising the ability to produce a useful product.

Note should be made of the need for effective use of the computer and its environment, including support software. In this project we did call on operating system experts for help when needed, but only a minimum. Help is needed and should be utilized directly to support such a project.

SPECIAL REQUIREMENTS TO GET INTO OPERATION

What is readily available with a local computer operating system and its supporting capability may not be adequate to satisfy the needs. The ability to drive a calculation through a series of code accesses must be satisfied. In our case, a Driver residing in

memory performs key functions using operating system features and capability, while codes are accessed under the direction of a Control Module that is not normally resident. To carry out a sophisticated calculation involving choices of calculational paths, the associated logic and decision making must be located somewhere, and this programmed procedure is, in a sense, a Control Module. A sophisticated computation system may have more than one Control Module, perhaps one for automating the solution of each major type of global problem or even parts thereof. One of these controllers will admit direct user specification of a set calculational path, including a loop structure and possibly double loops, and it must decide to change the calculational path based on the progress of a calculation.

Not too long after our system came into extensive use locally, the complaint was received that only 500 code accesses were allowed in one run! This number had to be increased to satisfy application needs. In no simple way could the equivalent calculation have been coded outside of a modular system.

There is always some concern that when the needs are satisfied with routines coded in machine language, they cannot be exported nor easily converted locally to a different type of computer. We have indeed experienced this block that may require special computer operating system changes if our computation system is to be made operational on some computer types. Such considerations may well affect what is done, but they should not be allowed to seriously compromise the effort of establishing what capability is really needed locally. First identify the important requirements for current and projected applications and then see that these are satisfied without submitting to the whims of the computer manufacturer and its systems people or your own regarding what has been made available. These specialists may have to help, of course, and system changes should be kept to a minimum so that conversion to another system is not too difficult. Lacking such help or available local system expertise, success of such a project is compromised.

MINIMAL USEFUL CAPABILITY

There is always a strong incentive and there possibly may be pressure to get minimal useful capability operational. Satisfying this objective can be an important signal to others that the effort is viable. In our case the minimal useful capability consists of:

Driver
Control Module
User Input Data Processor
Cross Section Processor
Neutronics Code

and the following had to be implemented:

Job Control Instructions
Data Management System
Interface Data Requirements Satisfied
Scratch Data Requirements Satisfied

Note the significant amount of effort involved! This evidently major chore and associated long time delay for effecting capability is a serious block to modular code system development. Unusual circumstances including no direct reactor development project responsibility allowed us to proceed with this project in the early 1970's. At the very least, use should be made of a product from another installation. For the system to be very useful, more than one code is required, and effective user input data processing is also necessary. Special data is always needed in active development, so both a temporary and a mature scheme are usually needed for generating an interface data file.

QUALITY ASSURANCE ASPECTS

To effect a high level of quality assurance in the application of computer codes requires that very specific action be taken. The procedures must be qualified, for example. Here we are concerned with such aspects as providing an edit trace that documents what calculation procedures were used and what results were stored on data files and what data were used. Let us say that the neutron flux file is generated on logical unit 12 by code X, on logical unit 14 by code Y, and is read from logical unit 15 by code Z, unless told otherwise. Simple reliability arguments lead at least to the defaulting of a common file to the same logical unit in the member code modules. What if data can be read from a file and used as if it were the flux, but it is not the neutron flux? Data need identification, and certain testing must be done if proper data communication is to be effected. We don't rely on a read failure to indicate that the wrong data file is being read.

The documentation of the use of calculational procedures is not simple. A code generation date and version number can be edited for a reliable reference if the user does not replace a code with his own special version, an action that should be discouraged by good practice and code/system management, as well as by making such action difficult. The only accurate documentation of user input for a calculation is a direct listing of what was supplied, relatively easy to do at single input data processing point.

DRAWING INTERFACE LINES

The task of establishing interface lines should not be taken lightly. Basically the anticipated calculational tasks need to be carefully grouped and reasonably blocked for individual code modules. Perhaps the larger the effort the more practical are smaller code blocks, the optimum for one programmer being one code (at one time). The smaller the task assignment block size the more flexible the collection, but the more severe the data interfacing requirements. If macroscopic cross sections are generated outside of the neutronics code, they are data to be interfaced that would not be required if generated in the neutronics code. Evidently similar tasks should be carried out in a single code rather than used to justify the proliferation of modules. We would concentrate diffusion theory neutronics capability in a single code up to the point where additional capability would become uneconomical to incorporate, as due to code complexity.

A working interface diagram covering the full capability is useful to support decision making. Each code module should have written specifications that are revised as the development continues to account for changes and to provide reference information. Developers have come to appreciate that a complete set of specifications is crucial documentation, although objectives may change in the process of code development, requiring revision of the specifications to keep them up to date.

PRIMARY DATA SOURCES AND PROCESSING

Some data such as nuclear cross sections involve too many numbers to allow simple user input and so must be supplied as an external data file. Often several codes obtain data from the same data file, causing serious impact if the specifications for this data are changed. The requirements for adequacy and flexibility of the data are crucial as is the fixing of the format early on. Thus one or more data libraries become a key data source supplied externally. What is the most practical to generate is seldom what is the most needed for a calculation, so library data processing is a common necessity to select, combine, and restructure it. In our case a nuclide-ordered microscopic cross section data file is recast into an energy-group ordered data file for efficient subsequent use, and this processing capability had to be implemented early in the project.

A geometric description of a reference reactor core takes on the importance of a primary data source. Reducing the amount of simplification and approximation in its representation is an important objective. (Projected requirements need man-machine interaction with display to generate such data.)

There is interest in some circles in assigning data to levels of hierarchy. Certainly use could be made of such a scheme for any complicated calculational system. It is not clear what level of effort is justified to support and maintain such, certainly little at our place. The effort that would be invested to just understand the use of an available scheme could not be justified by us to place it into operation.

DATA FILE MANAGEMENT

Data management means different things to different people. A primitive form of data management is a user telling a code that certain data are on logical unit 5 instead of the default unit 4, and to put results that are generated on unit 10 instead of unit 5. There is a certain definiteness of logical unit numbers (required in current Fortran) that seems to be obscured when dealing directly with data set names.

A neutronics code generates a region average, group dependent neutron flux interface data file. A simple exposure calculation uses this data. For simple analysis a single file is needed with the latest data. It is generated by one code and used by another. A more sophisticated calculation might do the exposure calculation using flux estimates for the beginning and end of an exposure period. Then two files are required. If a sensitivity analysis is to be done, then each of the flux files used may be needed later, so several files

are needed, preferably stacked on an input/output device. These files contain different data but have identical formats. Data file management provides a mechanism for controlling, identifying, and using such files.

A data file manager can associate a formal file name with a logical unit so that this number is not fixed in a code. Then logical input/output units are used as needed for such data, the automated association eliminating the frustrating and error prone assignment with user input data. When a file is to be written, the manager tells the code what unit to write on, and when a file is to be read the manager says where it is. Association of version numbers or other mechanisms admit a hierarchy, age or importance association that is often handled automatically.

We have learned that no one else understands how our data file management scheme (manager) works in detail, and in some cases even we have been surprised. Rather provincial rules have been adopted as the implementation proceeded regarding file version numbers. When two or more files have the same name they are assigned different version numbers, the highest version number represents the latest data, and that file is generally always used, unless there are special user instructions. Only the file version numbers in the data file management tables are used, the actual numbers written on the files being generally unique but not necessarily the same as the ones in the table. Default procedures cause a new file to be written over the highest version old one that exists, overridden by user instructions, with certain notable exceptions. The file proliferation tendency of a code system must be kept under control. We do this by usually writing over old files. Such action may impact the understanding of the information that is made available to the user.

The scheme of assigning a version number to a file, indexing this up as new versions are added, and recognizing the highest version as the working version to be used or replaced is a nice default procedure. Consider the nuclide density file. Normally only one version exists, and it always contains the latest data. If the neutronics code performs a criticality search, it generates a new nuclide density file so that the original data is preserved in the process of iteration to a solution. (In retrospect it might have been better to store the original data in a scratch data file, but how the demand for scratch files grows!) This is done by writing over an old file (if it exists) as the default procedure, limiting to two the usual number of nuclide density files existing. What if original nuclide densities supplied in an interface data file are to be saved? There are simple ways to do this in this system including making one or two copies of the original file with the initialization procedures. Then the original file has a lower version number than the active ones. (Another way is to set up a stack of files for recovery and use these to initiate.) At any point along a calculational path that a copy of the nuclide densities is desired, as to allow auxiliary calculations or to provide restart capability, the nuclide density file version numbers may be indexed up (tell the exposure code to write a new version number rather than write over the old one). For control rod positioning and fuel management, reference data is kept by copying a nuclide density file into another file with a different identifying name.

The point to be made here is that data file management is a relatively complicated subject. Careful study of the needs that can be anticipated may influence the rules. Certainly a set of rules should be drawn up that are explicit to the extent of being informative

as well as permitting new implementations to be in conformance, and these should be reviewed and kept up to date. Prepare to defend yourself when informing programmers about changes in the rules.

The data file manager can play initialization and wrap up roles. For example, we supply data files that are made known to the manager, although such is always at some burden to the user. Computer operating system routines are then used to generate one or more copies of any of these initial files desired before the calculation starts. At wrap up, selected files are stacked on one input/output device for saving and selective recovery is then done if desired later, also using operating system routines.

What has not been done is to work out an effective procedure for wrap up on an abort condition (as an alternative to normal wrap up). Clearly special procedures are in order with some selectivity regarding the storing away of intermediate results. If a significant development effort went into a new scheme, the approach of using Fortran logical input/output unit numbers at the file management level would likely be changed. An alternative could be quite attractive. A mechanism would be needed to retire files when no longer needed, saving them for future recovery under some form of identification, possibly by stacking and/or data set identification, and reassigning logical units or reusing the space. Effective automation of such a procedure might limit its applicability. For example, the existence of more than two versions of the same file could be used to cause the oldest to be stored away (copied with system routines) with override and recovery procedures possible to implement. The utility of such a scheme may well depend on how well it is tailored to a specific application, the amount of its use affected by both the available flexibility and its complexity.

DATA COMMUNICATION AND PHYSICAL FILES

A rather frustrating aspect of coupling individual codes is the inability to communicate any data from one to another within the memory under simple operating system rules. How needed is the capability to simply declare that an array of data is resident in memory.

In this system there is a data array in memory allocated in the Driver that is communicated to the Control Module under a weird and non-standard entry to a subroutine. Within any code using the packaged data handling routines, capability is available to default one or more scratch data files to be resident in memory if a procedure for doing this is programmed in the code and space allows; in this way the amount of data transfer is held reasonably low, allowing the solving of small problems at an acceptable penalty. The table of data file management information is read upon entry to each member code and written on an external device prior to exit, thereby always communicating any changes.

The interface data files are stored on disk, except that the use of a mass storage device is practical for long term storage, as of a cross section file, and some use is yet made of tapes for very large files. Files are all closed upon exit from a code and must be reopened

prior to use in another code. There is no simple allowance for interfacing files to be structured for direct access by record in the rules we operate under, so the interface files are all written sequentially.

Locally the virtual memory mode of computer operation is in vogue, the system being able to move data in and out of memory much more efficiently than can the programmer under the Fortran compatible rules. Although the virtual memory feature does not seem to gain anything significant for a modular computation system, it certainly is of importance in implementing capability. Although the need for data transfer from the disk within the program instructions is not eliminated, a simpler scheme may be the direct consequence of an apparently large memory, and much larger problems can of course be solved without programmed data transfer.

INTERFACE DATA FILE SPECIFICATIONS

Hard and fast and formal interface data file specifications are necessary. Typically these are drawn up to satisfy data needs blocked into practical packages considering generation aspects. Major difficulties include early disagreement about details, lack of data format and content flexibility to admit future changes, lack of clarity regarding contents, confusing and impacting complexity for flexibility, and resistance by the programmer to conform. An unimplemented data file specification remains an untested unproven unreliable quantity. Proof testing involves implementation of the generation and independent implementation of the use with resolution of difficulties.

In our case, a national effort went into such specifications under federal support that had limited success, as evidenced by local changes and extensions, and the lack of extensive use and conformity within the funded community. Of course the steady decrease in support funding of basic reactor physics methods development confuses the matter.

A major chore of drawing up interface data file specifications is allocating data into distinct blocks. Not only must the subject be viewed from the receiving point to define what is needed, but also from the delivery end to identify what can be packaged. Obvious objectives include simplicity to facilitate implementation, sophistication to admit complicated modeling, flexibility to cover most any situation, limitation to avoid uncontrolled proliferation, completeness to avoid revision and adequacy to satisfy the needs.

It is difficult to avoid data redundancy, especially as enhancements are added. Any computation system in which a user must describe the same thing more than once for a problem description is clearly deficient, and the results are subject to gross discrepancy. Extensive use must be made of the basic data files with provision to override where desirable. One thing that can be done to reduce future redundancy is to define a global problem description file with the details to evolve with development effort. We found a need for such a file.

The view that each datum in a data file must be formally defined in a specification is simply provincial. Certainly the format of a file must be described and the user of data

must know what the generator put in a file. Thus if a code squirrels away some data in a file for subsequent use by this code, there is little interest in and likely no concern about the data outside of it. Early specifications are not likely to be complete, inviting the provision for extension. (Revising codes to satisfy data file format changes is one big headache, however.) At least undefined flags should be included that allow the communication of special information and indeed the future addition of records of data at the end beyond data initially defined. "The number of groups is made negative to indicate . . . " is a description that means lack of foresight in the initial specifications.

Code and code system documentation that does not cover interface data file specifications is inferior and incomplete. An example interface data file specification is shown in Fig. 2. Note the association of a name with the file as well as simple user identification. Also within the specifications is a note that explains an extension beyond a simple orthogonal coordinate system expanding the original coverage.

INTERFACE DATA FILE WRITING

We initially took the approach that no interface data file should be written in a code module unless so specified by the user. We still believe that the writing of data files should be kept down through user control. When, however, certain data are the primary product of a code module, the associated data file should be written as the default. For example, our exposure code that treats burnup normally rewrites the nuclide density file with data for the end of exposure (but not after any specified shutdown period) as the default. With sophistication comes less certainty regarding what constitutes the primary results.

As an example, a neutronics code may be used to solve the importance problem for a specific adjoint source. Evidently the "flux" solution is the primary result and this file should be written as a default. If, however, the higher harmonic problem was solved instead, likely the value of k_1 is the primary result and use may not be made of the flux solution. On the other hand, for reactor core history calculations the zone average multigroup flux solution is of primary interest and the associated interface file should be written. If a thermal hydraulics calculation is to be done, then the pointwise power density file is a result of much interest. Note the difficulties. Evidently the user must exercise some control over interface file writing while flexible and effective default procedures should be implemented. A sophisticated scheme could assign many of the interface data file option selections to a Control Module which then would communicate case dependent requirements to the codes. Our Control Module trips certain file writing options based on the calculational path, but not enough control is automated.

SCRATCH DATA FILES

A difficulty is associated with storing large amounts of data on auxiliary storage units, at least on some computers. For large neutronics problems, data must be transferred to

```
C*****************************************************************************
C                        REVISED 07/23/75 - REVISED 04/01/80                  -
C                                                                             -
CP          PWDINT                                                            -
C                                                                             -
CR          POWER DENSITY BY INTERVAL                                         -
C                                                                             -
C*****************************************************************************

C-----------------------------------------------------------------------------
CP          FILE IDENTIFICATION                                               -
C                                                                             -
CL    HNAME,(HUSE(I),I=1,2),IVERS                                             -
C                                                                             -
CW    1+3*MULT=NUMBER OF WORDS                                                -
C                                                                             -
CD    HNAME             HOLLERITH FILE NAME - PWDINT - (A6)                   -
CD    HUSE(I)           HOLLERITH USER IDENTIFICATION (A6)                    -
CD    IVERS             FILE VERSION NUMBER                                   -
CD    MULT              DOUBLE PRECISION PARAMETER                            -
CD                       1- A6 WORD IS SINGLE WORD                             -
CD                       2- A6 WORD IS DOUBLE PRECISION WORD                   -
C                                                                             -
C-----------------------------------------------------------------------------


C-----------------------------------------------------------------------------
CP          SPECIFICATIONS (1D RECORD)                                        -
C                                                                             -
CL    TIME, POWER, VOL, IM, JM, KM, NCY, NBLK                                 *-
C                                                                             -
CW    8=NUMBER OF WORDS                                                       -
C                                                                             -
CD    TIME              REFERENCE REAL TIME, DAYS                             -
CD    POWER             POWER LEVEL FOR ACTUAL NEUTRONICS PROBLEM,            -
CD                       WATTS THERMAL                                         -
CD    VOL               VOLUME OVER WHICH POWER WAS DETERMINED,CC             -
CD    IM                NUMBER OF FIRST DIMENSION FINE INTERVALS              -
CD    JM                NUMBER OF SECOND DIMENSION FINE INTERVALS             -
C                                                                             -
CN          FOR SPECIAL TRIANGULAR GEOMETRIES, IM IS THE NUMBER OF            *-
CN          TRIANGLES ON EACH PLANE AND JM IS NOT DEFINED (SET TO 1)          *-
C                                                                             -
CD    KM                NUMBER OF THIRD DIMENSION FINE INTERVALS              -
CD    NCY               REFERENCE COUNT (CYCLE NUMBER)                        -
CD    NBLK              =1                                                    *-
C                                                                             -
C-----------------------------------------------------------------------------


C-----------------------------------------------------------------------------
CP                      POWER DENSITY VALUES (2D RECORD)                      -
C                                                                             -
CL    ((PWR(I,J),I=1,IM),J=1,JM)-----NOTE STRUCTURE BELOW-----                -
C                                                                             -
CW    IM*JM=NUMBER OF WORDS                                                   -
C                                                                             -
CS    DO 1 K=1,KM                                                             -
CS  1 READ(N) *LIST AS ABOVE*                                                 -
C                                                                             -
CD    PWR(I,J)          POWER DENSITY BY INTERVAL, WATTS/CC                   -
C                                                                             -
C-----------------------------------------------------------------------------


C*****************************************************************************
C                                                                             -
CEOP        PWDINT                                                            -
C                                                                             -
C*****************************************************************************
```

Fig. 2. Example of Interface Data File Specification.

and from disks during the iterative solution process. These must be randomly accessed by logical record for transfer to be efficient, and careful blocking and separation effected between devices. Thus we have some applications where certain scratch data file requirements are severe. These files have been assigned set logical numbers not used for interface data files, and space allocation is done with parameters supplied in the job control instructions through a catalogued procedure as a user convenience. Thus the associations of logical units for interface data files and scratch files are separated. (In some systems, unused logical units are used for scratch locally and then closed after use, causing the association of file with logical unit to depend on the calculational path.)

USER INPUT DATA AND PROCESSING

There are serious difficulties in the area of user input data. Here only a few aspects will be addressed. Codes written even in adjoining offices require quite different instructions and often use individual input data formatting. Putting codes together in a computation system results in inconsistencies difficult to explain, incompatibilities hard to eliminate, redundancies that do not go away, etc. Data formatting differences thwart the efficient supply of data by a user.

The following describe action we took:

1. The global problem description (calculational path through the code modules for example) should certainly be separate from subproblem descriptions.

2. All of the primary options for each code and key data requirements were collected and made readily controllable by the user (or by a Control Module) by locating them in a special interface data file. This important step of compacting the primary control data is very important to simplify the changing of instructions between code accesses, yet it is a step that has often not been taken. There is of course some difficulty dividing the data between that usually unchanged and that made readily accessible. Certainly there are limitations. Still the instructions to the individual codes must often be changed between accesses to allow global problems to be solved. An example is the use of a neutronics code to obtain the regular and adjoint solutions one access and then an importance solution the next. Clearly the user should not have to supply two or more copies of a complete neutronics problem description. Special edits may be desired, special interface data files read or written, or different termination criteria applied for one or more access than in subsequent ones. A criticality search may be done initially after refueling and then not later, or a different type of search then done to model control rod positioning.

   So our system member codes each access a special record of instructions from a file of control information to initiate action and establish task assignment with major option control. Information about the control data file 'CONTRL' is shown in Fig. 3.

```
C*******************************************************************-
C                        REVISED 04/01/90                           -
C                                                                   -
CF          CONTRL                                                  -
C                                                                   -
CP          BOLD VENTURE SYSTEM CONTROL                             -
C                                                                   -
C*******************************************************************-

C-------------------------------------------------------------------
CS          FILE STRUCTURE                                          -
CS                                                                  -
CS       0V FILE IDENTIFICATION                                     -
CS                                                                  -
CS          RECORD NAME    CODE BLOCK (MODULE)            REQUIREMENT  -
CS          -----------  ------------------------------  ------------ -
CS       1D   PROINS     (PROBLEM INFORMATION)           ALWAYS FIRST -
CS       1D   DVPINS     (MODULE DRIVING INFORMATION)    ALWAYS       -
CS       1D   XCPINS     CROSS SECTION PROCESSOR (CIGAR) AS NEEDED    -
CS       1D   DTNINS     NEUTRONICS (VENTURE)            AS NEEDED    -
CS       1D   RRTINS     REACTION RATE (RATEF)           AS NEEDED    -
CS       1D   RODINS     CONTROL ROD POSITIONING (ROD'MOD) AS NEEDED   -
CS       1D   EXPINS     EXPOSURE (BURNER)               AS NEEDED    -
CS       1D   PERINS     PERTURBATION (PERTUBAT)         AS NEEDED    -
CS       1D   REFINS     FUEL MANAGEMENT                 AS NEEDED    -
CS       1D   RRLINS     THERMAL HYDRAULICS              AS NEEDED    -
CS       1D   BLANK      (CLOSURE)                       ALWAYS LAST  -
CS                                                                  -
C-------------------------------------------------------------------

C-------------------------------------------------------------------
CP          FILE IDENTIFICATION                                     -
C                                                                   -
CL     HNAME,(HUSE(I),I=1,2),IVERS                                  -
C                                                                   -
CW          3*MULT + 1                                              -
C                                                                   -
CD     HNAME          FILE NAME (A6) 'CONTRL'                      -
CD     HUSE(I)        USER IDENTIFICATION (A6)                     -
CD     IVERS          FILE VERSION NUMBER                          -
C                                                                   -
CW     MULT           1 FOR LONG WORD, 2 FOR SHORT WORD MACHINES   -
C                                                                   -
C-------------------------------------------------------------------

C-------------------------------------------------------------------
CP          PROBLEM INFORMATION                                     -
C                                                                   -
CL     PROINS, (XX(I),I=1,100), (IX(I),I=1,100)                     -
C                                                                   -
CW          101*MULT + 100                                          -
C                                                                   -
CD     PROINS         PROBLEM INFORMATION DATA IDENTIFIER (6HPROINS)-
C                                                                   -
CD     XX(1-24)       PROBLEM OR CASE TITLE (A6)                   -
C                                                                   -
CD     XX(25)         USER LABEL (JOBNAME) FOR INTERFACE DATA FILES (A6) -
C                                                                   -
CD     XX(26)         USER LABEL (DATE) FOR INTERFACE DATA FILES (A6) -
C                                                                   -
CD     XX(27-100)     RESERVED                                     -
C                                                                   -
CD     IX(1-100)      RESERVED                                     -
C                                                                   -
C-------------------------------------------------------------------
```

Fig. 3. Information About the Code Module Control Data File.

3. A free form input data processor written at LANL was adapted for the generation of interface data files directly from the user input data stream. Free form input data with significant blank separators is deemed to be better than fixed form. A major difficulty with it, however, is the fact that the length of each array of data must be prespecified locally. The ability to selectively revise an existing data file sounded more important than was found to be the case in practice. More often than not the revision procedures were found to fail or perhaps were not well understood. One-to-one input of data files does suffer from a major limitation: more friendly user input requirements would allow deriving one or more files of data using special processing procedures, but we have not been able to invest effort in this area and still rely on special fixed form format user input processors. The use of certain special input processors is preferred to one-to-one interface file writing in some cases even though the data have a fixed form format. Extensive use of terminals for input to codes, however, also favors free form formatting. One advantage of the free form format is the ability to debug the most data in one pass (with modest limitations).

4. The input data stream supplied by the user is edited for documentation.

5. A computer system difficulty forced us to copy each block of input data from the input stream to a special unit prior to its use. Each distinct block of input data is terminated with a simple "END" message in columns 1-3 on a terminating card, a technique that proved to be quite practical. We chose to implement look-ahead capability to effect early termination if a user input data block is not available. Most user input data in this system must be properly ordered, and this is not judged to be a severe application problem, although some ordering reference could be useful.

6. Mixed fixed form and free form formatted user input data is only better than mixing two different types of free form data with different rules. Enhancing the capability of a free form user data processor seems to be a never ending topic; not only must new capability be added but schemes must also be implemented to do clever things to aid the user.

A further comment is offered about unfriendly interface data files. A neutronics code needs a detailed geometric description while a fuel manager must have volume information, and calculating this from a general geometric description is not very practical every time the data is needed. The user should not have to supply both the geometry description and the zone volumes. A most suitable input data processor would derive information from data supplied. Since a general processor won't have built in data association capability, its utility is not entirely adequate. Another example is the association between nuclide reference names, locations in the reactor core, and cross section data. Using the simple order number in the cross section set as a primary reference has long been used successfully, but this is rather prone to error. Extension from this to the use of absolute names really complicates matters. Further, the required associations are not simple to construct without the use of an informed input data processor.

ERROR, FAILURE HANDLING

This subject is quite complicated, although it is more a consequence of solving global problems than of using a modular code system. We attempt to identify whether or not a code module has successfully carried out its assigned task. Task failure is cause for an abort with wrap up. Failures include programmed tests such as lack of convergence of an iterative neutronics problem and certain unacceptable results. Operating system tests that cause abort, as of overflow, divide by zero and inadequate storage, have not yet been brought under control; we still deem such to be a failure and cause to abort, but would like to have control over wrap up.

FILE OPENING, DATA TRANSFER, EDITING

This subject area would seem to be separable from the modular system aspects. However the same physical space may have to be used for scratch data by two or more codes leading to some complications. The use of a set of standardized data file management and data transfer procedures is desirable in the codes, but this certainly is more practical for a new code than for one being rewritten.

We would comment that flexible editing capability is attractive. This does not mean that editing capability is not needed within the individual code modules, but only that it may be minimized with general system capability. For example, given sufficient reporting of other results, a space, energy pointwise neutron flux edit is seldom needed. When it is needed, a file editor may be quite adequate for the purpose, eliminating the need to program this in the neutronics code, even though the flux interface file would have to be written, and this may not be necessary otherwise. Local editing of data will generally be attractive when the data are derived and not simply defined for interfacing.

Quite generally we like to see an edit trace of information along the calculational path from each code, information that often for one reason or another proves to be quite useful. We once attempted to suppress the iterative history of solving multidimensional problems, but won't likely do this again.

The capability to generate a hard copy of any of the interface data files in a form permitting reconstruction by an input processor is of much utility, binary to formatted and back.

This discussion would be incomplete without addressing the subject of data transfer, primarily done between memory and disk, although more permanent storage is also involved for some files. Under the USDOE interinstallation cooperative physics code effort, REED and RITE statements supplanted the READ and WRITE Fortran statements in the codes. Data are transferred by block (string or array) to and from binary files. So the location of the first word in the string, the length, the unit number and a transfer mode flag are presented, along with the file record number that requires accounting within the coding. Thus both or either sequential and direct access by record data transfer techniques are supported. Committing to the use of service routine for data

transfer is important. Limiting the capability to what is practical is only reasonable. For example, we consider only fixed length records in the direct access files.

You find rather provincial techniques used for data transfer. We experienced so much trouble with sequential file backspacing (tapes) that we do not use it. We like to explicitly place an end of file mark when a file has been written, apparently now done automatically by most systems. The environment affects what is done. Locally we could not communicate direct access files from one code to another (because the system opening procedure default wrote over the file), and so we do not use this attractive capability. Basically in engineering oriented coding effort, a set of rules and restrictions to operate under is found that works, and they become rather absolute without concern about the reasons. Note that we still have to work around what are viewed at least as peculiarities of the Fortran compiler, as when optimization changes the intent of coded procedure.

The file READ and WRITE instructions are done in service routines, as is file opening. Basic capability is relatively easy to implement. Three important extensions are attractive that are not so easy to implement. The operating system input/output procedures may be used directly, supplanting the operating system Fortran procedures. The subject is beyond this discussion, although the authors would note rather poor experience in this area. Prompt and efficient data transfer are desirable that may not be easily achieved, especially under a multitasking system that breaks long logical records into small physical records. Serious delay time is associated with not completing the transfer of all of a logical record once started. The fastest transfer minimizes rotational delay and read head positioning where applicable, and involves overlapping of actions. Any changing of the rules for using the operating system procedures is stressful to say the least. Any significant impact on the user job control instructions from the use of special input/output procedures should be avoided, as with the use of catalogued control instructions.

When a hierarchy of data storage is available, special action is necessary for efficient use. The slow extended core is a third level of storage presenting a challenge. To what extent simple procedures can be used effectively for multilevel data transfer seems to be only limited by the ingenuity of the developers, although some of the early computer operating systems effectively prevented efficient computer use. Here the view is expressed that the code programmers should not be burdened with special accounting or association requirements when these are easily buried within the service routines.

The third extension we are quite pleased with is the ability to default data files to reside in memory. This is done within the file opening procedures, but only by coded instruction. Our capability in this area is limited to scratch files within a code. Thus the programmer is burdened with generating the necessary instructions, accounting for available space, making explicit decisions. A more global approach to memory and disk space allocation is needed with more automation of the choices within a presented hierarchy of preference. Sequential files present a challenge because the record lengths must be stored, and storage for such information is different from that for the data. Of special concern is the reading of less than all of a record of data, requiring special positioning action unless

such is disallowed (impacting the coding rules). Defaulting small data files to memory is necessary for solving small problems efficiently on many computers. This capability significantly reduces the burden of programming parallel data handling and calculational procedures required to efficiently treat a wide band of problem size. Indeed this capability may change the whole approach taken to data handling, especially since the coding must often admit solving the largest conceivable problem.

CALCULATION SUMMARY

A summary of a calculation is deemed to be an essential edit. We chose to edit key information from the control module plus a line or two from each computational (but not data processing) module as a summary that typically is one or at most three pages long. What constitutes a reasonable summary of the history of a calculation is, however, rather obscure. Over-emphasis will likely be placed on certain results, on computation aspects or perhaps the calculational path. A compact summary is possible if certain editing is avoided that might seem to be desirable. Special considerations include the need for such information to be returned to a remote terminal separately from the primary edit, but it also should be included with the primary edit for documentation and to serve as a directory. A computation system that does not generate a condensed summary of the calculation with key results is deficient. Although it may be practical to generate a file of data for direct reporting, that is not the primary objective. Tables for reports should be generated separately where deemed useful.

A sample of a calculation summary is presented in Fig. 4. In this calculation following a reactor history the neutron flux solution is obtained at selected points in time, exposure is calculated, a preselected control rod positioning schedule is applied, and the core is refueled. Prior to refueling, the dominant harmonic neutronics solution is obtained. The ability to normally apply a neutronics code one way and yet to assign it a special task is demonstrated. Examination of the data shows that a net generation of fissile material occurs, so the core is that of a breeder reactor.

VIABLE CALCULATIONAL PATHS

In some calculational systems there may be a clean distinction between calculational paths that will generate viable results and those that will not. In our system the distinction is hazy, each code requiring only certain data to support a task assignment and avoid an abort due to a missing file. A user likely becomes accustomed to using but a few of the possible paths that he assumes are or he has proven to be viable. Thus the thermal hydraulics code will solve a problem given a geometric description and a power density distribution, whether these were both supplied as input or the heat source was calculated by a neutronics code prior to access of the thermal hydraulics code. Whether either was desired by the user is up to him. So the identification of a path that is not viable is left as a rather obscure but severe burden on the user, rather an undesirable state. An obvious

```
BOLD VENTURE VERS-0 RUN ON THE IBM-360/***. CONTROL MOD=CONTROL1, DATE=08-01-81, TIME=18.19.57, JOBNAME=TBFA     .

INITIALIZATION, REFERENCE - REMAINING I/O=  18.99, CPU MIN=  19.06

RUN TITLE AND CONTROL MODULE DATA
 SOLID SAMPLE PROBLEM.
2.0000     0     0     0     0     0     0     0     0 0 0  1  0  0  0
  1  2  2 12  7 19 13  2 12  7 13  2 12  7  9 15  2  7  2 12  7  2 12 19
  7


                    INITIAL I/O FILE MANAGEMENT TABLES
                  FILE NAME SUPPLIED BY
                  ---------------------                                  USER
FILE NUMBER       SEEK    USER-EXIST   USER-STACK      VERSION      WRITTEN   IDENTIFICATION
    10           CONTRL                                  1            1     TBFA      08-01-81


MODULES TO BE ACCESSED IN ORDER
     1     2     2    12     7    19    13     2    12     7    13     2    12     7     9    15     2     7     2    12
     7     2    12    19     7

READ DATA AND ACCESS MODULE  1    - REMAINING I/O=  18.97, CPU MIN=  19.06

READ DATA FOR A SPECIAL PROCESSOR - REMAINING I/O=  18.83, CPU MIN=  19.01

ACCESS PROCESSOR 'DVENTR'         - REMAINING I/O=  18.81, CPU MIN=  19.01

READ DATA FOR A SPECIAL PROCESSOR - REMAINING I/O=  18.77, CPU MIN=  18.99

ACCESS PROCESSOR 'ROUTIN'         - REMAINING I/O=  18.76, CPU MIN=  18.99

READ INPUT DATA ONLY (LOOK-AHEAD) - REMAINING I/O=  18.74, CPU MIN=  18.99

ACCESS MODULES 12  7 19 13  2     - REMAINING I/O=  18.72, CPU MIN=  18.99
     CYCLE, OPT, POWER HIST, CYCLE TIME, TIME IN CYCLE, FRAC.IN CYCLE=  1  0  1  0.0          0.0          0.0
     ITERATIONS, CONVERGENCE, SEARCH, PEAK POWER DENSITY, K =     32  -3.229E-05   0.0          6.83995E 02    0.9935811
     PRIMITIVE CONVERSION RATIO, ALSO FOR CRITICAL SYSTEM, FUEL CONSUMPTION(ATOMS/WATT-SEC) =   1.47192   1.45235   2.837325 10
     FUEL MANAGEMENT,TIME= 0.0        DAYS,CYCLE=    0,INITIAL CORE FISSILE INVENTORY(KG)  8.961172 03
     TIME AFTER EXPOSURE 295.500 DAYS, ESTIMATED K  0.985126, FISSILE INVENTORY 5.2331812 03 KG, CONVERSION RATIO(FB)    1.42573

ACCESS PROCESSOR 'ROUTIN'         - REMAINING I/O=  13.56, CPU MIN=  17.25

READ INPUT DATA ONLY (LOOK-AHEAD) - REMAINING I/O=  13.54, CPU MIN=  17.24

ACCESS MODULES 12  7 13  0  0     - REMAINING I/O=  13.50, CPU MIN=  17.24
     CYCLE, OPT, POWER HIST, CYCLE TIME, TIME IN CYCLE, FRAC.IN CYCLE=  1  0  1  0.0          2.555002 02  0.0
     ITERATIONS, CONVERGENCE, SEARCH, PEAK POWER DENSITY, K =     38  -1.869E-05   0.0          5.33619E 02    0.9898390
     PRIMITIVE CONVERSION RATIO, ALSO FOR CRITICAL SYSTEM, FUEL CONSUMPTION(ATOMS/WATT-SEC) =   1.46977   1.43931   2.875815 10
     TIME AFTER EXPOSURE  511.000 DAYS, ESTIMATED K  0.990895, FISSILE INVENTORY 5.5028522 03 KG, CONVERSION RATIO(FB)    1.41552

ACCESS PROCESSOR 'ROUTIN'         - REMAINING I/O=  12.83, CPU MIN=  15.87

READ INPUT DATA ONLY (LOOK-AHEAD) - REMAINING I/O=  12.8 , CPU MIN=  15.86

ACCESS MODULES 12  7  9 15  0     - REMAINING I/O=  12.39, CPU MIN=  15.86
     CYCLE, OPT, POWER HIST, CYCLE TIME, TIME IN CYCLE, FRAC.IN CYCLE=  1  0  1  0.0          5.111302 02  0.0
     ITERATIONS, CONVERGENCE, SEARCH, PEAK POWER DENSITY, K =     38   4.002E-05   0.0          5.072882 02    1.0188367
     PRIMITIVE CONVERSION RATIO, ALSO FOR CRITICAL SYSTEM, FUEL CONSUMPTION(ATOMS/WATT-SEC) =   1.42956   1.47280   2.942332 10
     ADJOINT - ITERATIONS, CONVERGENCE, K =     28  -1.125E-05   1.018*398
     FISSILE INVENTORY 5.50... KG, CONVERSION RATIO(FB)    1.41662, FISSILE CONSUMPTION RATE 1.1622812E-14 KG/W-SEC
     FWD ADJ K 1.018835 1.018836, LIFETIME 3.410002-07, P.D. ACCP.  1.837625-13, STP (K DELTA K)/(K DELTA S)  2.253940-01
```

Fig. 4. Sample of Calculation Summary.

```
ACCESS PROCESSOR 'DUTLIN'          - REMAINING I/O=   9.71, CPU MIN=  12.23

READ INPUT DATA ONLY (LOOK-AHEAD) - REMAINING I/O=   9.68, CPU MIN=  12.22

ACCESS MODULES  7  0  0  0  0      - REMAINING I/O=   9.68, CPU MIN=  12.22
     ADJOINT - ITERATIONS, CONVERGENCE, K -    49  5.91715D-06    1.0148364

ACCESS PROCESSOR 'DUTLIN'          - REMAINING I/O=   5.69, CPU MIN=   8.61

READ INPUT DATA ONLY (LOOK-AHEAD) - REMAINING I/O=   5.66, CPU MIN=   8.61

ACCESS MODULES 12  7  0  0  0      - REMAINING I/O=   5.66, CPU MIN=   8.62
     CYCLE, OPT, PT.IN HIST, CYCLE TIME, TIME IN CYCLE, FRAC.IN CYCLE=  1  1  1  0.0          5.110307 02  0.0
     ITERATIONS, CONVERGENCE, SEARCH, PEAK POWER DENSITY, K -    33   4.7726?D-06   0.0           5.09122E 02     0.9888956
     PRIMITIVE CONVERSION RATIO, ALSO FOR CRITICAL SYSTEM, FUEL CONSUMPTION(ATOMS/WATT-SEC) -   1.39758   1.09073   2.762795 10

ACCESS PROCESSOR 'DUTLIN'          - REMAINING I/O=   4.58, CPU MIN=   7.03

ACCESS MODULES 12 19  7  0  0      - REMAINING I/O=   4.57, CPU MIN=   7.03
     CYCLE, OPT, PT.IN HIST, CYCLE TIME, TIME IN CYCLE, FRAC.IN CYCLE=  1 -8  0  0.0          5.110307 02  0.0
    FUEL MANAGEMENT,TIME= 5.110307 02 DAYS,CYCLE=    0,FISSILE INVENTORY,BEGINNING,ADDED(KGM)  5.650133 03  6.193099 02  2.996042 03
     ITERATIONS, CONVERGENCE, SEARCH, PEAK POWER DENSITY, K -    25   4.2959?D-05   0.0           5.97836E 02     1.0201836
     PRIMITIVE CONVERSION RATIO, ALSO FOR CRITICAL SYSTEM, FUEL CONSUMPTION(ATOMS/WATT-SEC) -   1.38755   1.40635   2.907435 10

                              FINAL I/O FILE MANAGEMENT TABLES
                          FILE NAME SUPPLIED BY
                     ------------------------------                          TYPE
FILE NUMBER          SEEK    USER-EXIST    USER-STACK       VERSION    WRITTEN    IDENTIFICATION
    10           CONCEPL                                        1          1     CDCA      04-01-81
    11           GEOMDAT                                        1          1     CDCA      04-01-81
    12           RODDATA                                        1          1     CDCA      04-01-81
    13           REPORT                                         1          1     CDCA      04-01-81
    14           RESTRTL                                        1          1     CDCA      04-01-81
    15           SEQDATA                                        1          1     CDCA      04-01-81
    16           NDXSEQ                                         1          1     CDCA      04-01-81
    17           DUMPDUM                                        1          1     CDCA      04-01-81
    18           XMACRO                                         1          1     CDCA      04-01-81
    19           RSTRLNX                                        1          1     CDCA      04-01-81
    20           RSTRLNX                                        1          1     CDCA      04-01-81
    30           FMATDM                                         1          1     CDCA      04-01-81
    32           FIXSRC                                         1          1     CDCA      04-01-81
    33           ATFLNX                                         1          1     CDCA      04-01-81
    34           DEPLTD                                         1          1     CDCA      04-01-81
    35           ATFLNX                                         2          1     CDCA      04-01-81
    36           XMACRO                                         2          1     CDCA      04-01-81
    37           MASDAL                                         2          1     CDCA      04-01-81

WRAP-UP NOW COMPLETE. -- DRIVER TO SEEK NEW CASE - REMAINING I/O=   3.45, CPU MIN=   5.61 - TIME=19.28.58
```

Fig. 4. Continued.

resolution of the difficulty is not known. Certainly a new user can be expected to have some difficulty in this area. A constraining mechanism that would raise the level of reliability without undue limitation has not been identified.

Indeed some recent effort in the area of thermal hydraulics has gone in the direction of making the task assignment details dependent on the available data. Since the heat source information may come from more than one code and be in more than one form, a hierarchy is used. If a pointwise power density data file is available, it is used. (The highest version number is used and of course this will be current in an ongoing reactor history calculation only if it was recently written.) Lacking the availability of this file, a zone average power density data file is used, and the averaged data redistributed over the mesh points. The point here is that if at all possible a task should be carried out and user friendly procedures should do the best they can with the available information. Limiting what is done to only the most sophisticated procedure not only limits the utility of the procedure but also limits its ability to generate useful information, as may well be desired. The whole subject is rather complicated with decisions to be made in each specific instance. Application flexibility and utility, reliability, and documentation requirements bear consideration. Global problem solving capability impacts the requirements. The user is not going to check the results from each step before proceeding to the next (although a new global problem description certainly needs to be carefully checked out). Long calculations that require much computer time should be aborted upon failure, but there must be good cause for such abort.

We define a viable calculational path through the codes as a route for which there is full availability of needed data. (The results may or may not be of any use.) The failure to communicate current results is a cause for error not simply identified.

SOME COMPUTATIONAL DIFFICULTIES

A few significant challenges remain to be solved. The discussion below describes some of the difficulties.

1. Recently at 8:00 AM one Monday morning the computer operator called the first author to say that a job had been on the computer for about 10 hours and would likely need 10 hours more to complete. There was nothing he could do but to abort the job to make way for the day's work. There was nothing I could do to save what had been already done. Occasionally a very large neutronics problem is modeled to generate detailed results. (Note that how much relief comes from the nodal modeling approaches now popular in such a situation is not clear and likely depends on the situation.) There are times when at least a trivial amount of outside intervention is needed that is not available. We are forced to predescribe every calculation and would often in retrospect have done things differently. The effective application of a computer can be

achieved in some cases only by performing calculations in stages. This is due to the lack of adequate automated critical review of results, inadequate programmed decision making, and no capability for monitoring the results and intervention to change the course of a calculation. Hardware technological advance and software development have not brought full relief in this area.

2. What may seem to be but a trivial change in a calculational path can prove to be very hard to implement, especially if there are associated special data communication requirements. If data are to be obtained from an interface file that is not used by a module, the procedures for reading interface files must be incorporated; it is not simply reading a record from a file. If the data are not available in any file, it must be added to a file or a new one described, either of which impacts the formal specifications and the procedures when the data are available. If it is desired to perform a thermal hydraulics calculation after exposure has been calculated prior to a subsequent neutronics solution instead of after it, then the data requirements change somewhat and these may or may not be easily satisfied with or without additional programming effort and possibly alterations in the user instructions.

3. Similar to the above, what would seem to be but a small change in a modeling algorithm may not be easy to incorporate. Is this to replace or parallel the present procedure? Which is to be the default? Well, if the new one is, what happens to quality assurance and to continuing applications by unknown analysts? A step change in the results in time is seldom acceptable in any project. New data requirements may place a severe burden on implementation and may even be judged to be so severe as to make the implementation impractical.

4. Task assignments must be made prior to coding the procedures. Clearly the primary calculational paths must be predicted in advance. With limited foresight and but a modest investment to study the requirements, the end product will be imperfect at best. Study of the reasoning behind what others have done in this area and a view to the future regarding advanced technology, enhanced capability and the need for complexity and flexibility are highly recommended.

5. Redundant input data requirements lead to results of low reliability. Considerable effort can be justified to consolidate the data requirements. For example, when different geometric descriptions of a core are required for neutronics and thermal hydraulics problems, calculations will be done that are not consistent. When two code modules must each separately be provided overriding instructions or data to produce consistency in the treatment, inconsistency will occur. An example in this system is that each code must be provided both reference data and instructions to use the correlation on temperature. Typically two, three, or possibly four different places in the user input data stream are involved. There is always a need for the communication of key options

and associated data to the codes in a simple way. Burdening the analyst with the need to supply redundant data, special considerations to effect consistency, detracts -- often there are other aspects that should be receiving his primary attention. (Perhaps in some operations there is a large support staff that takes care of such matters.)

6. As an example of the lack of foresight, a block of interface data file names was defaulted [illegible]nager, and two of these were for a nuclide ordered cross section file. [illegible] codes use only a group ordered file generated infrequently from th[illegible]ontaining nuclide ordered data, we now in most runs have to override those inappropriate default assignments. Default procedures should be based on the need for simplicity in application with attention paid to eventual capability and use of some future complicated system.

7. The code members cannot be executed standalone, one reason being that they do not process input data directly. This impacts such aspects as testing. Special techniques are in order. For example, a special driver routine has been used as a substitute main routine to permit one of the codes to be executed standalone during development and permit extensive testing to be done easily.

8. Such a code system is applied by both experienced and naive analysts who have rather different requirements that should be satisfied, impacting the development.

9. The problems associated with there being the need for more files than allowed by the system and with limited flexibility in saving files are noted elsewhere.

FORTRAN CODING

Typical of scientific and engineering analysis methods implementation, the coding was done in Fortran.[a] Reasons for the use of Fortran include familiarity, continuity with the past, compatibility with other capability, and exportability. Also we desired to exploit the features of this programming language in spite of the difficulty of proving reliability of Fortran source coding.

Some experience with the use of Fortran and the use of good programming practices are essential for the economical generation of reliable codes. As a system of codes grows, the common practice of patching up poor coding could lead to excessive investment and unreliable capability. A large fraction of the coding effort must go into proof testing for validation.

Perhaps one of the more important things we have learned is that a simple structure of the logic is necessary. A visual inspection of a coded procedure should serve adequately as a basic proof of sound logic; otherwise, with a multiplicity of coded transfers, the logic becomes difficult to check. Generally too much reliance is placed on the compiler for eliminating errors and not enough use is made of the tools that have become available for reliability testing.

---

[a]Standard ANSI-X3.9, 1966, likely obsoleted by ANSI-X3.9, 1978.

ACKNOWLEDGEMENTS

We recognize the important contribution by L. M. Petrie. He wrote the resident Driver code, without which this effort could not have been seriously considered, contributing much to the success of the project. G. W. Cunningham was an important contributor in implementing this capability.