THE JOSHUA SYSTEM

Vol. 1
GENERAL

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Approved by

H. C. Honeck, Research Manager
Computer Applications Division

NOTICE

This document contains descriptions of the JOSHUA Operating System and Applications Modules. The document is prepared as a set of loose-leaf notebooks which will be periodically updated. All of the material is preliminary and should not be referenced.

November 1969

E. I. DU PONT DE NEMOURS & COMPANY
SAVANNAH RIVER LABORATORY
AIKEN, S. C. 29801

CONTRACT AT(07-2)-1 WITH THE
UNITED STATES ATOMIC ENERGY COMMISSION

DISTRIBUTION OF THIS DOCUMENT UNLIMITED
DISTRIBUTION OF THIS DOCUMENT UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

GENERAL

Part A Introduction

Part B Documentation

Part C Data Set Abstracts

Part D Physics Module Abstracts

Part Z Reprints of Publications

JOSHUA

DPSTM-500

Title Part A - Introduction

Vol. I	Issue A.O.1
Rev.	Date 10/69

Contents

A.1 - Introduction to the JOSHUA System.

JOSHUA
DPSTM-500

Title

I	A.O.2
Rev.	Date 10/69

(Blank)

INTRODUCTION TO THE JOSHUA SYSTEM

Work Done By

H. C. Honeck

J. E. Suich

Report Written By

H. C. Honeck

J. E. Suich

Page A.1.2	Vol. 1	Title INTRODUCTION TO THE JOSHUA SYSTEM	JOSHUA DPSTM-500
Date 1-69	Rev.		
(Blank)			

CONTENTS

	<u>Page</u>
1. Introduction	A.1.5
2. Concepts	A.1.7
2.1 Stand-alone Programs	A.1.7
2.2 Linked Programs	A.1.10
2.3 Modular Systems	A.1.15
2.4 The JOSHUA System	A.1.19
2.5 Influence on Savannah River	A.1.21
3. Operating System	A.1.25
3.1 Elements of the System	A.1.25
3.2 Facilities	A.1.28
3.3 Data Management	A.1.28
3.4 User Access	A.1.35

Page A.1.4	Vol. 1	Title INTRODUCTION TO THE JOSHUA SYSTEM	JOSHUA DPSTM-500
Date 1-69	Rev.		
(Blank)			

1. Introduction

The third generation of computers now in use at most nuclear reactor design installations offers new possibilities to the developers and users of reactor computing codes. In particular, high-speed central processing units are now commonplace; almost all systems have large quantities of direct access storage, and many new devices are available to improve the accessibility of the computing system to the reactor designer - such as alphanumeric display terminals and cathode ray tubes with full graphic capability. The trend in reactor computing systems^(1,2) has been towards utilization of these features of third-generation computers. At the Savannah River Laboratory, the JOSHUA system is being developed to capitalize on the experience gained with earlier systems, and to include a remote terminal capability.

JOSHUA is an operating system which provides the data handling services required to accomplish extensive, iterative scientific calculations on a large semi-permanent data base. JOSHUA provides standard, "catalogued procedures" for solving routine problems, and allows easy redefinition of these procedures for experimental computation. The user is permitted to intervene between steps of a computation, display results, and possibly redirect the procedure. The use of individual display terminals for problem setup and output display is central to the design philosophy of JOSHUA, although duplicate facilities are provided for batch mode use where practicable.

2. Concepts

JOSHUA is not a radically new and untried concept. Rather, it is the natural extension of scientific computing techniques now being used throughout the country. To place JOSHUA in the proper perspective, we will try to show how and why the concept evolved.

2.1 Stand-alone Programs

The earliest and still-used method of performing a scientific calculation using a computer is illustrated in Figure 1. The important point to understand in this illustration is how information flows between the various components. Information (input data) flows from the user to the computer via the medium of input cards. The computer program performs operations on this input data and the results are transmitted to the user via printed output sheets.

A nuclear reactor cannot be designed by running a single computer program. The actual design procedure involves many such calculations run in series and in parallel. This is illustrated in Figure 2. The designer runs one or more calculations using Program A. The results of these runs are digested by the designer, runs are repeated where required, and the results are then merged to form input data to run Program B. Similar steps are then used to run and rerun subsequent programs until a reactor design is obtained.

There are several key points to recognize:

1. The designer is in complete control of the process. He decides what calculations are to be performed and in what order. He decides whether a calculation was successful or whether more are required. He adds experience and judgment to the calculations to obtain his results.
2. The designer spends much of his time in routine manipulation of data. He must extract pertinent results from the output sheets, correlate them, and prepare input for subsequent calculations.
3. A simple three-step calculation as shown in Figure 2 might take as many days to complete. Time is required to write input sheets and get cards punched. Several hours elapse after a run is submitted before the output is available. If input errors were made they must be corrected and the calculation repeated.

Figure 1

Figure 2

Page A.1.10	Vol. 1	Title INTRODUCTION TO THE JOSHUA SYSTEM	JOSHUA
Date 1-69	Rev.		DPSTM-500

2.2 Linked Programs

In the early 1960's computer hardware and software had progressed to the point where it was possible to link programs together. This is illustrated in Figure 3. Programs A and B are executed in sequence in a single computer run. Part of the output data from Program A was placed on a data file in the computer to be read by Program B. In the earliest versions the data file was a magnetic tape. Later versions use disks or drums. In order to develop linked programs two things are necessary.

1. The computer software had to be available.
2. The results of one calculation (Program A) require no user inspection prior to being used as input to a later calculation (Program B); that is, this part of the design process could be fully automated.

There are several advantages to this type of operation:

1. The input data was simplified, common input to Program A and B need only be given once as opposed to twice if they were stand-alone programs.
2. The output data was simplified and automatically correlated the results of the two programs.
3. Two steps were performed in one run on the computer. The computer time used was not reduced but the time for the user to get his results was reduced by nearly a factor of two.

Figure 3

Figure 3 illustrates the linking of two codes. It is quite possible to link many programs together to further reduce the user's work and allow him to get his results quicker. The HAMMER program is a linkage of 6 existing programs plus 2 new programs to simplify the input data and summarize the output data. Other laboratories have developed similar programs of the complexity of HAMMER.

How far can the development of linked programs be pursued? Figure 4 is an illustration of how a linked program of about the complexity of HAMMER would look. There are several problems to recognize:

1. Linked programs were not designed as such but grew from existing programs. Thus the connecting data files were constructed to simply contain the output from one program and make it available to later programs. These program-oriented data files were not easily accessible to, or readable by, the user.
2. The tight inter-connections between the programs and the data files means that changes in any one element require changes in many elements. It is difficult to upgrade or refine a program.
3. The user cannot easily run only one program in the linked program. This requires that stand-alone versions of each program are also available to the user.
4. The user appears only at the input and output to the linked program. This wastes much computer time. Errors discovered in later calculational steps may require repeating the entire sequence of calculations because intermediate results necessary to restart were not saved. The user has lost control over the calculation since he cannot conveniently alter the course of the calculation. If the results of early steps were physically meaningless (but yet cause no program errors), the calculation proceeds and the later calculational steps are wasted.
5. The entire system becomes so complex that neither the user nor the program designer understands the program. The system is a black box.

Figure 4

Page A.1.14	Vol. 1	Title INTRODUCTION TO THE JOSHUA SYSTEM	JOSHUA DPSTM-500
Date 1-69	Rev.		
<div>(Blank)</div>			

JOSHUA DPSTM - 500	TITLE	INTRODUCTION TO THE JOSHUA SYSTEM	Vol.	Page
			1	A.1.15
			Rev.	Date
				1-69

2.3 Modular Systems

About four years ago, in 1964, it became obvious that linked programs had reached their limit and that a different approach was required. Two independent groups began development of modular systems. It is interesting to note that the applications which motivated these developments were different. Argonne National Laboratory felt that their modular system (called ARC) was the only way to calculate fast reactor accidents which involve strong coupling between neutronics, thermal-hydraulics, reactor geometrical configuration changes, stress analysis, energy release, etc., all as a function of time. KAPL designed their modular system (called NOVA) to simplify the design of high performance power reactors and be able to automatically calculate all neutronic, thermal-hydraulic, and control requirements as a function of the operating time of the reactor. The common element in both systems is that many calculational steps are required, that these steps are strongly interdependent, and that these steps must be repeated many times before the desired results are obtained.

In principle, modular systems could have been developed with magnetic tape-oriented computers. In practice, modular systems had to await the development of more efficient computers. In 1964 these "third-generation computers" became available along with the software to use them. The main features of these new machines (IBM 360, CDC-6600, UNIVAC-1107) were their increased speed and their use of large, fast access magnetic disk storage. Increased speed meant more complex calculations could be done in the same time. Disk storage devices meant a different approach to the data files which linked programs together.

A modular system is illustrated in Figure 5. The calculational programs are called "modules". The collection of data files is called the "data pool". Interposed between is a computer program called the "data manager". As shown, the figure is more illustrative of the NOVA system than of the ARC system, since ARC has a relatively simple data manager and rigid data pool construction.

How does such a system work? One of the modules is the control program (called a "path"). Assume that the path is Module A. It reads input data prepared by the user and transmits its output to the data pool. The path next executes another module which gets its input data from input cards and from the data pool. The results are placed in the data pool. Thus, modules are executed in the sequence dictated by the path and each module receives its input data from cards and/or the data pool and transmits its results to the data pool and/or a printer.

Figure 5

COMPUTER

How does this differ from the linked programs? At first glance, all that has been done is to collect the calculational programs on one side and call them modules, collect the data files on the other side and call them a data pool, and show a data manager neatly connecting them instead of a confusing set of lines. The important feature is that in Figure 5 it is shown as an integrated system and must be designed as a system, not as a set of independent programs coupled together in a haphazard fashion as the need for linking arises. The data files in a modular system are quite different from those in a linked program. In a linked program the data files are merely intermediate storage devices intended to pass data from program to program. The data files were designed to hold the data the program produced. In the modular system, the data files are first designed to contain the important data needed both by the designer and by modules in later calculational steps. Modules are then designed to take data from one or more data files and produce other data files.

The advantages of a modular reactor computing system are two-fold. First, it is extremely advantageous in the program development stage to be able to work with an independent computational module since the programmer who is developing the module need not be concerned with the programming ground rules or the structure of the other modules in the system. This greatly facilitates conversion of modules developed off-site and allows several programmers to work independently on the same system without causing repercussions throughout the entire system when programming errors are discovered and corrected.

In the second place, the modular structure of the reactor computing system leads to great flexibility in its use, since any module can be called in at any stage of the calculation as long as the appropriate interface data sets have been created on the direct access storage devices. This allows the same modules to be used, for example, for studies of infinite lattice reactor physics parameters, and for the averaging of cross sections for an iterative depletion calculation. Furthermore, in a system which provides the user with an interactive access device (such as a cathode ray tube), the path of flow of the computation through the various modules can be altered dynamically to reflect the immediate interest and/or judgment of the reactor designer without requiring a precompiled path to be located in core. This greatly enhances the investigation of exceptional situations which were not foreseen at the time the reactor computing system was generated.

Page A.1.18	Vol. 1	Title INTRODUCTION TO THE JOSHUA SYSTEM	JOSHUA DPSTM-500
Date 1-69	Rev.		

There are several disadvantages in the way that current modular systems (ARC and NOVA) have been implemented, but these will not be elaborated here. In concept, there is one major objection - the user still appears only at the start and end of the calculation.

2.4 The JOSHUA System

In 1967 it became apparent that it was impractical to extend linked programs such as HAMMER and that highly complex burnup and safety studies would require a modular system like ARC or NOVA. At this point in time, graphic display terminals had been used for several years at other installations and had demonstrated their ability to provide communication between the computer and the user. The use of these devices as an active part of ARC or NOVA was not anticipated so that these systems were not designed to use them. Experience using ARC and NOVA had demonstrated the advantages of modular systems, but, at the same time, had pointed out several areas where they might be better implemented. Consequently, it appeared to be the right time to develop another modular system better suited to the needs of Savannah River and which avoided the major shortcomings of ARC and NOVA. The development of JOSHUA system started in early 1968. An illustration is shown in Figure 6.

Two changes from the previous figure should be noted. First, the user can communicate with the computer using graphic display terminals. (Of course, the usual "batch" mode operation is also available.) Second, all the functions relating to the flow of data and control of the job are placed in the JOSHUA operating system.

Under JOSHUA, the user is in direct contact with the computer when he uses one of the terminals. A typical session might go as follows:

1. The user signs on and requests JOSHUA to perform a certain calculation.
2. Assisted by JOSHUA, the user enters his input data from the terminal. JOSHUA stores this data on one of the files (say, File 1).
3. JOSHUA then calls in the desired module (say, Module A).
4. Module A takes the data from File 1, performs the calculation, and stores its results on another file (say, File 3). In the course of doing the calculation, Module A may have called in one or more other modules needed to perform parts of the calculation.

Figure 6

5. When Module A has finished its calculation and stored its answers on File 3, control returns to JOSHUA which informs the user that his results are ready.
6. The user can then ask JOSHUA to display on the terminal, selected results. If these are not satisfactory he can revise the input data still stored in File 1 and repeat the calculation.
7. When the results are satisfactory the user may wish to perform another calculation using the results just obtained and still stored in File 3.

There are several important points to observe in this simple example.

1. The user can place his input directly into the computer, obtain immediate verification of its accuracy, and immediately correct it if necessary.
2. Each calculational step is under the control of the user, yet well defined sequences of steps can be fully automated and prepared as a module.
3. The results of a calculation are immediately available to the user and to later calculational steps.

2.5 Influence on Savannah River

As the JOSHUA System continues to develop it will influence many scientific computing activities at Savannah River. This influence will be felt in the areas of:

1. Code Development
2. Standardization
3. Problem-Solving Time
4. Information Retrieval
5. Computer Requirements

Code development is both easier and more difficult under JOSHUA than for stand-alone codes. At first it is more difficult since much of the total system must be developed before any codes can be run. Many data files must also be specified in a manner that avoids early obsolescence.

Code development under JOSHUA should be much easier than before. Modules interact only through well-defined data files. They can be written independently of one another and quickly modified or upgraded as needed. Complex codes can be conveniently generated from existing modules.

Increased standardization of both data and methods will occur under JOSHUA. Standardization of methods is desirable both to simplify the execution of a complex procedure and to insure that everyone executes the procedure in the same way. When such standardization is desired, the procedure is programmed as a module which controls the sequence and operation of the other modules. HAMMER is an example of a standard method.

Standardization of data is accomplished through the use of permanent data files available to all users. Two types of data can be considered; basic physical data, and problem description data. The desirability of standardizing basic physical data (nuclear cross sections, heat transfer coefficients, chemistry data, etc.) is obvious. Problem description data includes material and dimensional specification of a fuel assembly, description of a reactor charge, etc. If descriptions of these and similar items are stored in the computer, everyone can use the same descriptions for their calculations and have descriptions which can be conveniently modified for other applications.

The use of terminals to set up calculations, execute them, and inspect results will greatly reduce the amount of user time required to solve a problem. Design problems that require many individual computer runs can be completed in minutes or hours while days or weeks might be needed to run the calculation individually. The possibility of interdisciplinary teams design is opened up. A group of designers, each representing a different discipline, could effectively use JOSHUA to produce a complete design in a matter of days. Potential accident situations could be quickly analysed. AEC demands for "instant answers" could be satisfied.

Selected results of each calculation are permanently stored in the computer and are available for user inspection or later calculations. Rapid and convenient access to this information allows all users to better utilize their results as well as the results of others.

At the present time results are buried in stacks of computer output. Summary results are collected and put into reports. Frequently it is easier to rerun a calculation than locate the old results. When these results are available from the computer on demand it will be much easier to locate results and prepare new summaries.

It is clear that the size of the computer determines the amount of calculation that can be done. Decreasing the problem-solving time allows the user to solve more problems in a given time

period provided computer time is available. Since, at the present time, our computer is working nearly full time, we cannot solve many more problems than we are solving now. JOSHUA will not increase our computing capacity; in fact, it will somewhat increase our computing overhead.

This overhead is expected to be small and should be more than offset by better utilization of the codes. There will be less duplication of runs since results are stored and made available to all users. Fewer calculations are required since the user has immediate access to his results and can judge whether more calculations are necessary before proceeding to the next step of the calculation. The user will have a variety of calculational procedures available to him and can select either "fast and sloppy" or "slow and precise" procedures, depending on the problem at hand.

Page A.1.24	Vol. 1	Title INTRODUCTION TO THE JOSHUA SYSTEM	JOSHUA DPSTM-500
Date 1-69	Rev.		
(Blank)			

3. The Operating System

We next turn our attention to the operating system shown in Figure 6. This system must provide a variety of facilities to the user of the system. In the next two sections we outline the various elements of the system and the required facilities. The third section briefly describes data management, and the last section describes user access to the system.

3.1 Elements of the System

Figure 6 is an over-simplified diagram of a reactor computing system. A more realistic diagram is shown in Figure 7 and it is general enough to describe all existing reactor computing systems. The large square block represents the central processing unit and fast core storage. The bottom-most box labeled "OS" indicates that a portion of core storage must be devoted to an operating system which controls the computer system for the individual tasks above it. The box labeled "batch" indicates the possibility of multiprogramming (or multiprocessing) a background stream of jobs outside the reactor computing system (for example, FORTRAN compiles, non-reactor computations, etc.). The larger box above, subdivided by dashed lines, represents the reactor computing system itself. The top-most of these boxes labeled "JØS" is the collection of programming aides or control programs that must be provided to interface the reactor computing system with the operating system. The box labeled "Path" is meant to indicate that a program may reside in core to control the flow of computations between the various possible modules. Finally, the modules themselves also must be brought into core for execution.

To the right of the diagram are indicated the various devices involved in the flow of data in and out of the computer and between the various segments of core. The two diagrams at the bottom indicate that the batch mode area utilizes the usual off-line printer and card reader of a stacked job processor. The top two figures are schematics for graphic display devices and remote keyboards which show the possibility of allowing a reactor designer to gain easy on-line access to the reactor computing system, and to make use of the graphic capabilities of the new terminals in setting up complex input data. All of the communication between the job control path and module areas is via data and programs which reside on direct access storage. It is this extensive use of direct access storage which has illuminated the limited nature of facilities available in the standard reactor physics programming languages.

FIG. 7

TYPICAL REACTOR COMPUTING SYSTEM

Other elements of the JOSHUA System which will be discussed in more detail in later sections are:

- 1) Data - individual EBCDIC/binary values.
- 2) Data Record - collection of data.
- 3) Record Name - a string of names separated by periods.
- 4) Named Data Record - data record associated with a record name.
- 5) Data Set Name - first name of a record name string; i.e., a record name is a qualified data set name.
- 6) Data Set - collection of all named data records with the same data set name.
- 7) Catalogue - collection of defined record names organized into a hierarchical tree structure of data set name qualifiers, with pointers to associated named data records.
- 8) Data Descriptor - name and type of variable (datum).
- 9) I/Ø List - nest(s) of "implied DO loop(s)" of variable name(s).
- 10) Format - display overlay and data field descriptor(s).
- 11) Record Name Type - record name with variable qualifiers delimited by '?'.
?
- 12) Record Descriptors - I/Ø list, associated data descriptors and format(s).
- 13) Template - record descriptors associated with a record name type.
- 14) Glossary - collection of all defined variable names, with pointers to the associated template(s).
- 15) Monitor - JOSHUA Operating System, which provides data handling services to users.
- 16) Procedure Step - manipulation of named data record(s) by user or a module.
- 17) Procedure - collection of procedure step(s) required to perform a task.

3.2 Facilities

The JØS must provide many facilities both to the user and the programmer. These facilities are:

- 1) Named data records can be read, written, or deleted from FORTRAN coded modules in unformatted form.
- 2) Modules may be called into execution from FORTRAN coded modules; control is returned to the calling module upon completion of execution of the lower level module. Modules are passed only on "job" name.
- 3) Records in formatted form may be displayed, modified, created, deleted, by the user at the terminal or in the batch.
- 4) Catalogues of record names may be displayed by the user at the terminal.
- 5) Modules may be called into execution by the user at the terminal.
- 6) Procedures to be followed by the user at the terminal automatically introduce appropriate procedure steps.
- 7) A tutorial mode of terminal operation allows users to interrogate the Glossary.
- 8) Records may be referred to at the terminal by name or by logical relationships satisfied by data.
- 9) Data created by each user is catalogued separately from the standard system data base. A user may duplicate standard names without overwriting the standard data for other users.

3.3 Data Management

A large reactor computing system with many modules (all interfacing through direct access storage) creates, and requires the management of, a very large amount of data. Both the reactor designer and the program developer are faced with the formidable problem of keeping track of all of this data and referring to it repeatedly. The first activity demands that the data be referred to by the user in his natural language, and the second activity demands that the data be handled efficiently by the computing system.

JOSHUA DPSTM - 500	INTRODUCTION TO THE JOSHUA SYSTEM	Vol.	Page A.1.29
		Rev.	Date 1-69

Figures 8 and 9 address themselves to the problem of handling large quantities of data in a natural way. Figure 8 is a diagram of a typical multigroup cross section library, showing that the data are structured in the user's mind in a hierarchical fashion, that is, as a tree structure. For example, it is most natural for the user to indicate that he is interested in the value (in the multigroup library for use with the HAMMER system) for ^{238}U of the cross section for the $n,2n$ reaction. He cannot, without a great deal of research, refer to this data as, for example, FORTRAN record 1057. Similarly, the library of results of infinite lattice calculations contains data which the user thinks of in a hierarchical structure. That is, he would, for example, like to determine the value (in the lattice library for user Badge No. 3726) in the liquid metal-fueled reactor infinite lattice, of the output data for few-group cross sections.

TYPICAL MULTIGROUP CROSS SECTION LIBRARY

FIG. 9

TYPICAL INFINITE LATTICE DATA LIBRARY

In order to achieve efficient transmission of the data involved, a great many device-dependent factors must be considered. Figure 10 lists some typical auxiliary storage devices with their characteristics. The 729 tape is an IBM first/second generation device while the 2400 tape drive is a drive utilized with the third-generation System/360. There are two disk types available from IBM; the 2311 and the 2314, the only difference between the two being their storage capacity and the transfer rate. There are two drums available from IBM; the 2303 and the 2301, again, the only difference between them being in capacity and transfer rate. It is clear from the listed transfer rates and access times that there is a hierarchy of devices in terms of their speed. However, the arrangement given is in hierarchical order on transfer rate while a different order is required for access time. Since the efficient use of these devices demands that both the transfer rate and the access time be considered, it is obvious that one does not necessarily obtain more efficient data transfer by simply moving to the faster transfer rate devices. The third column of numbers labeled "Block", is the size of data which must be transferred in a block to cause the time spent simply transferring data to equal the time spent gaining access to the data, i.e., a block size which achieves 50% efficiency when gauged by the over-all transmission rate relative to the instantaneous transfer rate. These are widely different for the various number of devices. The final column labeled "Buffer" indicates the amount of core storage which the programmer should set aside to accept data being transferred into fast memory if he wishes to achieve complete overlap of data transmission with computation requiring 100 Model 65 operations per word of data transferred. Again, there is a large difference between the various devices. It is therefore necessary in handling the data to take cognizance of the various auxiliary storage devices which are available to the system.

FIG. 10

<u>DEVICE</u>	<u>TRANSFER</u>	<u>ACCESS</u>	<u>BLOCK</u>	<u>BUFFER**</u>
TAPE (729)	.4 msec/wd	10 msec	25 wds	-
TAPE (2400)	.1	10	100	150
DISK (2311)	.025	max 150	6000*	1000
		avg 75	3000*	500
		min 25	1000	250
DRUM (2303)	.0125	max 18	1370*	110
		avg 9	670	50
		min 0	-	-

*Exceeds track capacity

**To overlap 100 operations/wd

It seems clear that the burden of efficient use of the direct access storage should be placed on the subroutines which actually accomplish the transmission of data. Since fairly large tree structures will be involved in the location of data, the data manager may require a substantial amount of core space to avoid constant accesses to the DA space for the purpose of retrieving sections of the name tree lists. Since the data will be used in several different ways, that is, retrieved by name for specific application from given load modules, versus being retrieved by characteristic for display to the user at a graphic terminal, undoubtedly the full panoply of information retrieval techniques must be utilized here and this implies the creation and maintenance of more than one list structure; for example, the direct structure as well as an inverted structure and a key word vocabulary are probably all necessary.

Additionally, it will be necessary to develop utility programs to delete, update and rearrange the data on disk since the efficiency of access to the data will depend on the proximity of logically related data on the direct access storage device, and yet the data may not be created in that fashion.

A mature reactor computing system will have generated a good deal of data - undoubtedly more than can be accommodated by the on-line direct access storage space. It will thus be necessary to push down data collections from the faster access devices to the slower, ultimately to a tape library which can be filed away for possible future reference. This activity should go on in the background of the other data management activity. That is to say, storage space management should be done dynamically within the system to prevent inordinate delays while systems utilities are used to clean up disk space. Conversely it will be necessary to retrieve data from the slower access devices to the faster access devices, perhaps ultimately to work space on a drum, in order to make most efficient use of the hierarchy of direct access spaces available. To achieve this "pop up" of data, it is necessary to have some look-ahead. This is to say, it is necessary to know what data are likely to be required by a program then in core. The named data set access method provides for this very nicely. Once a program begins to work in a given area of the tree structure, it is most likely to continue to work in contiguous areas. This logical relationship between the data and its probable use should also provide for look-ahead buffering without requiring an explicit statement from the programmer.

3.4 User Access

For various reasons, existing reactor computing systems do not provide for user access to the system other than as sequential batch mode operation. The use of graphics is a possibility for the display of data sets for verification by the user, for the description of the current contents of data sets for user choice, for monitoring information for long problems, and for the use of a light pen for data revision during execution. These functions are, of course, the ones that first come to mind, and the ones which have been considered in the design of the JOSHUA system. Allowing the user access to the reactor computing system via an interactive mode terminal has many advantages. In the first place, the user gets "instantaneous" turnaround because it appears to him that the computing machine is working only on his job, and displaying results on his terminal as soon as they become available. This, however, is probably not the overriding advantage of the remote interactive terminal. The user, if he knows that his data is stored in readily retrievable form, under a name which he can easily remember and request, is not likely to demand the very large amount of hard copy, or printed paper, which he now receives from the batch mode system. A fair amount of time is lost in actually trying to locate data on the quantity of paper which the typical user saves. Secondly, the problem of preparing input for other reactor codes is greatly simplified by being able to call up the results of previous calculations, and manipulate these data in the machine, on line, into a form suitable for use by other modules. Thirdly, the preparation of input is vastly simplified by allowing the user to communicate with the machine in his natural language for describing the problem at hand. In most cases there will reside, in the data libraries, a very similar problem which the user can modify to suit his current interest. In the case of preparing, for example, a description of the reactor composition areas for a two-dimensional flux calculation, the reactor user of a stand-alone code such as PDQ must today spend a great deal of time (on the order of days) laying out his grid, assigning composition numbers to the grid points, obtaining a keypunched deck of this data and verifying the output flux map. All of this would be eliminated by simply flashing a previously utilized reactor grid on the screen of a cathode ray tube and allowing the user, with the light pen, to adjust the reactor face map to his current problem.

A second possibility which suggests itself is that of a reactor design team sitting down together at a graphic display to accomplish the total reactor design simultaneously, rather than successively running their aspects of the design independently. For example, a reactor physicist, a mechanical engineer, and an operations planning analyst might meet at the graphic display

device and begin running a design problem, each contributing his own specific knowledge during the flow of the computation through the various load modules. By pointing out immediately that a given design which is emerging from the calculation is defective in some respects with which the individual scientist is familiar, the very long iterative process of designing a reactor charge acceptable to all disciplines could be eliminated; in the space of an afternoon, perhaps, the work which currently occupies months might be accomplished. The simple fact that such devices have not been available before has prevented users from seeing their over-riding importance to the designer. Such devices are now in use routinely in other engineering areas such as automotive design and electrical engineering simulation studies, and there is no technical reason why these devices should not be similarly used in reactor design.

The burden of controlling the interactive terminal and initiating the dynamic path rests with the Job Control Manager (see Figure 7). The primary tasks of the Job Control Manager should be the interrogation of the input data sets for previously-run problems similar in nature to the problem at hand, the provision for user-modification of that input data subset (thereby creating a new subset), the initiation of a path which will execute the desired procedures in the reactor computing system, and the possibility of dynamically defining new procedures for the reactor computing system. It would also be very useful for the reactor physicist investigating unusual situations to be able to retrieve the source module corresponding to some standard load module, modify a few FORTRAN statements, (thus creating a new module), and have that module compiled and executed dynamically under the control of his interactive terminal. The extreme flexibility of such procedures would greatly enhance the amount of work done by the reactor physicist by making it easy for him to utilize the full facilities of OS/360.

If the user is to have access to the data, the system must handle both the unformatted data sets stored on the disk and the formatted data supplied by, or presented to, the user. In a batch mode system this facility is supplied by the Fortran `FORMAT` statements. An analogous concept for a graphic terminal oriented system is the "template". A template can be defined for each data record type and the template information itself can be stored as part of the data base. A template would consist of the `I/O` list for the record type, the data descriptors (name and type of the variables), and formats (display overlay and data field descriptors). If a user at a terminal requests the system to display a named data record, the system would first examine the name to determine the record type, obtain the

JOSHUA DPSTM - 500	Title INTRODUCTION TO THE JOSHUA SYSTEM	Vol. 1	Page A.1.37
		Rev.	Date 1-69

appropriate template, and display the requested data in the format specified by the template.

Templates could be prepared as short Fortran subroutines containing the read/write statements and formats. It is more efficient, however, to store the template information as a data record and have the system interpret this information to read or display data. The template information can be prepared from input cards but it would be highly desirable to be able to create templates directly from a terminal.

The basic facilities which the system must supply the terminal user are the ability to create, modify, and display named data records, and the ability to execute a module. With these facilities available the user can prepare (by creating or modifying) the input data records required by a module, execute the module, and display the output data records. Each of these individual operations is called a "step", and the collection of steps is called a "procedure". A document could be prepared which would instruct the user as to what steps were required to run a module. It would be more desirable to have this information contained in the system so the user would not have to consult manuals. This information is called a "catalogued procedure" and is stored as part of the data base. In the simplest cases when the user calls a catalogued procedure, the system automatically guides the user through the input preparation, executes the modules, and displays the output. In more complicated cases, however, procedures may involve execution of several modules, perhaps even iteratively. Thus, a procedure is equivalent to a program written in a problem-oriented language defined by the functions of the module and terminal commands.

Page A.1.38	Vol. 1	Title INTRODUCTION TO THE JOSHUA SYSTEM	JOSHUA
Date 1-69	Rev.		DPSTM-500

REFERENCES

- (1) Argonne National Laboratory, AEC Research and Development Report (ANL-7332), The Argonne Reactor Computation (ARC) System, November 1967.
- (2) Kopp, H. J. and D. S. Selengut, Knolls Atomic Power Laboratory, Report CNM-R-2, Proceedings of the International Conference on The Utilization of Research Reactors and Reactor Mathematics and Computation, Mexico, D. F., May 1967. DATATRAN - A Data-Handling Computer Language for a Large Modular Reactor Design System, pp 1460-1472.

JOSHUA

DPSTM-500

Title

Part B - Documentation

Vol.

I

Page

B.O.1

Rev.

Date

10/69

Contents

B.1 - Documentation Procedures

JOSHUA
DPSTM-500

Title

Vol.
I

Page
B.O.2

Rev.

Date
10/69

(Blank)

JOSHUA

DPSTM-500

Title Documentation Procedures

Vol.

I

Page

B.1.1

Rev.

Date

10/69

Work Done By

H. C. Honeck

Report Written By

H. C. Honeck

Contents

	<u>Page</u>
1. Introduction	B.1.3
2. Modules	B.1.4
3. Subroutines	B.1.6
4. Data Sets	B.1.7
Appendix A: The MEMØ Program	B.1.8

1. Introduction

Proper documentation is essential for a large complex code system. Documentation serves several purposes; it provides communication between the many people developing the system, it tells users what is available and how to use it, it facilitates later changes and improvements to the system, and it describes the theories and techniques employed so that others may use them.

All JOSHUA documents will be unclassified. Care should be taken to keep classified data out of modules and module descriptions. This should present no problem for module descriptions and data set specifications. Listings of data contained in data sets will have to be carefully edited to prevent security violations. Until such procedures are worked out, no classified information should be placed in data sets.

We might classify documentation as either "working" or "historical". The documents described here are working documents. They are informal and subject to constant revision. The organization of these working documents was chosen for its convenience to the people who must prepare the documents. A different organization is required for the formal "historical" report. It is expected that working documents will be prepared as modules are written and modified frequently. When the modules have been operating successfully for some period of time, and when no further revisions are contemplated for the near future, it would be appropriate to prepare a formal report on a group of modules which are related or frequently used together.

The documentation for JOSHUA will be kept in a set of loose-leaf notebooks. The titles of the first four notebooks are:

- Volume 1 - General Information
- Volume 2 - Users' Guide
- Volume 3 - Data Set Specifications
- Volume 4 - Physics Modules

Other volumes will be added as required. Each volume is separated in parts labeled A, B, C, etc. Within each part, documents are numbered sequentially. Pages are numbered sequentially within a document. For example, 4.B.2.26 refers to page 26 of the second document in Volume 4, Part B. This numbering system should facilitate the updating of the documentation.

2. Modules

Documentation should be a habit, not a chore. It is certainly less of a chore if each worker reports only on the work he actually did. Thus, we recognize that several people are involved in the preparation of a module. First, there is the originator who specifies the problem. Second, an analyst prepares the logical and numerical restatement of the problem. Next, a programmer prepares the computer code. Finally, all three work together to use the code and verify that the stated problem has been solved. The organization of a document reflects this subdivision of effort.

A module description consists of three separate sections:

<u>Section Title</u>	<u>Prepared By</u>
1. Abstract	Analyst
2. a. Theory	Originator
b. Program Logic	Analyst
c. Coding Details	Programmer
3. User Information	Originator/Analyst

The abstract should be no longer than two single-spaced pages (one page, both sides). All abstracts will be collected in Volume 1 of the JOSHUA Manual. The format for the abstract is:

Module Name:

Authors: (include all contributors)

Purpose: (brief description of the theory and use of the module)

Status: (experimental, production, etc.)

Modules Used: (list of names of other modules or standard JOSHUA subprograms called by this module)

Data Sets: (list of data sets used and generated by this module)

Restrictions: (note any limitations or restrictions on the use of this module)

The second section will appear in the appropriate volume (i.e., Physics Modules) and may be subdivided into three main sub-sections; theory, program logic, and coding details. While this organization might be convenient for some modules, a different organization might be better for other modules. Any organization is satisfactory if it facilitates the preparation and updating of the document and contains (where required) the following type of information.

Theory

- Statement of the problem to be solved.
- Method of solution.
- Equations and physical approximations.
- References

Program Logic

- Numerical methods and approximations.
- Logical organization of the program.
- Flow diagrams where needed.
- References

Coding Details

- Symbol correspondence.
- Data organization and restrictions.
- Flow diagrams where needed.
- Special coding techniques.
- Code listing.
- References to standard subroutines.

The third section, user information, will be placed in Volume 2 along with information about the operating system. This section should cover the following topics:

- Input description.
- Output description.
- Interaction with other modules.
- Error messages.
- Suggestions for proper use.
- Sample problem.
- Execution information.

The first page of each section should contain a table of contents for the section, and the credits in the form -

Work Done By

H. C. Honeck
C. E. Bailey

Report Written By

H. C. Honeck

3. Subroutines

Frequently, subroutines (or other major blocks of code) will be used in several modules. These should be documented in a format similar to that for a module.

4. Data Sets

An abstract of each data set should be prepared for inclusion in Volume 1, Part C. This abstract should be no longer than two single-spaced pages and contain a description of the purpose of the data set and the classes of data which it can contain.

Descriptions of the data sets are contained in Volume 3 of the JOSHUA Manual. A data set description is given in five sections:

1. Introduction

- Purpose and Usage.
- Relationship With Other Data Sets and Modules.
- Structure of the Data Set (over-all).
- Any Theory or Explanations Required.

2. Nomenclature

- Symbols
- Definitions
- Types (e.g., REAL*4)

3. Record Descriptions

- Names
- Lists

4. Templates

- Record Type
- Sample Display

5. Data

- Abstracts of Data Record Names Currently Available.
- Abstracts of Available Data.

Sections 2 and 3 are produced from punched cards by a computer program (see Appendix A). Section 4 can be produced from print-outs of sample records or photographs of displays. Section 5 is produced by computer programs which prepare selected abstracts of the actual data stored in the data set.

Appendix A: The MEMØ Program

The MEMØ Program produces Sections 2 (Nomenclature) and 3 (Record Descriptions) of a data set description from card input data. There are two reasons for recording these descriptions on punched cards. When a data set is being developed the format changes frequently until a satisfactory format is obtained. Punched cards provide a convenient means of effecting these changes and printing updated documents for use by programmers. The second reason is the advantage of having complete data set descriptions in a computer-readable form. These descriptions could (at a later stage of development of the operating system) be stored along with the data and be used for tutorial and information retrieval purposes.

MEMØ is a stand-alone program operating only in the batch mode. Data set descriptions are punched on cards according to the formats given in the following table. Synonym and Format cards are intended for later use and will be ignored by MEMØ. MEMØ prints the cards in a format which can be used directly in the data set document.

Description of Input Card Deck for Each Data Set

GENERAL READ STATEMENT

```
      READ(5,1) T,C,(A(I),I=1,9)
1  FORMAT(A1,4X,A1,8A8,A2)
```

T (COL.1) A TEST CHARACTER FOR CARD TYPE.

- B - BEGIN DATA SETCARD. DATA SETNAME IS IN COLS. 7-14 AND STORED IN A(1). DATE (XX/XX/XX) IS IN COLS. 15-22 AND STORED IN A(2). SHORTHAND NOTATION (NAME.NAME. ETC →X) IS IN COLS. 23-72 AND STORED IN A(3)-A(9).
- D - SYMBOL DEFINITION CARD. SYMBOL IS IN COLS. 7-14 AND STORED IN A(1). DEFINITION IS IN COLS. 15-70 AND STORED IN A(2)-A(8). VARIABLE TYPE IS IN COLS. 71-72 AND STORED IN A(9). THE TYPE IS WRITTEN AS 'R4', 'R8', 'I4', 'C8', ETC, WHERE R=REAL, I=INTEGER, C=CHARACTER.
- S - SYNONYM CARD. MAIN SYMBOL IS IN COLS. 7-14 AND STORED IN A(1). SYNONYMS ARE SEPARATED BY COMMAS AND ARE IN COLS. 15-72 AND STORED IN A(2)-A(9).
- H - HEADING CARD. THIS CARD STARTS THE DEFINITION OF A RECORD AND IS FOLLOWED BY N(=NAME), L(=LAST), AND F(=FORMAT) CARDS. THE TITLE OF THE RECORD IS IN COLS. 7-72 AND STORED IN A(1)-A(9).
- N - NAME CARD. THE RECORD NAME IS IN COLS. 7-72 AND STORED IN A(1)-A(9).
- L - LIST CARD. THE RECORD LIST IS IN COLS. 7-72 AND STORED IN A(1)-A(9).
- F - FORMAT CARD.
- C - COMMENT CARD. COMMENTS ARE IN COLS. 7-72 AND STORED IN A(1)-A(9).
- P - PRINT CONTROL CARD TO SUPPLY TITLES FOR GROUPING RECORDS OR DEFINITIONS. TITLE IS IN COLS. 7-72 AND STORED IN A(1)-A(9).
- E - END CARD.

C (COL.6) A CONTINUATION TEST. ANY OF THE ABOVE CARD TYPES CAN BE CONTINUED WITH A NON-BLANK PUNCH IN COL. 6 ON THE CARD (OR CARDS) FOLLOWING. COLS. 1-5 ARE IGNORED AND THE CONTINUATION IS IN COLS. 7-72 AND STORED IN A(1)-A(9).

DECK ORDER -

1. B CARD NAMING DATA SET
2. NOMENCLATURE CARDS TYPE D,P,C,S
3. E CARD
4. RECORD DEFINITION CARDS TYPE H,N,L,F,P,C
5. E CARD

JOSHUA
DPSTM-500

Title

Vol.
I

Page
B.1.10

Rev.

Date
10/69

(Blank)

JOSHUA

DPSTM-500

Title

Part C - Data Set Abstracts

Vol.

I

Page

C.O.1

Rev.

Date

10/69

Contents

Page

C.1	-	ENDFB	C.1.1
C.2	-	FINEGRP	C.2.1
C.3	-	MULTIGRP	C.3.1
C.4	-	FEWGRP	C.4.1
C.5	-	ASSEMBLY	C.5.1
C.6	-	RILIB	C.6.1
C.7	-	LATTICE	C.7.1

Data Set Name: ENDFB

Author: D. R. Finch

Purpose or Use:

The ENDFB data set provides isotopic cross sections in the detailed pointwise representation of the Evaluated Nuclear Data File.(1) This data may be utilized by processing modules to produce cross sections in any multigroup or special representation desired by the user. Data contained in the ENDFB data set consists of the standard ENDF/B set distributed by the National Neutron Cross Section Center plus locally-evaluated cross sections put into the ENDF/B format. Various versions within the data set enable new evaluated cross sections to be introduced while retaining current evaluations intact.

Code modules are available to transfer data from ENDF/B library tape to the ENDFB data set, and to average ENDFB data into the multigroup representation of the FINEGRP data set.

Types of Data Stored:

The types of data stored in the ENDFB data set are the following:

1. Documentation of sources of cross sections.
2. Smooth cross section.
3. Resonance parameters for single-level or multilevel representations.
4. Secondary angular distributions.
5. Secondary energy distributions.
6. Secondary angle-energy correlations.
7. Thermal scattering law data in tabulated $\delta(\alpha, \beta)$ form.
8. Fission product yield and decay parameters.
9. Delayed neutron yield per fission and energy spectrum.
10. Total neutron yield per fission and energy spectrum.

* Honeck, H. C., ENDF/B, Specifications for an Evaluated Nuclear Data File for Reactor Applications, BNL 50066 (1966), Revised July 1967.

JOSHUA
DPSTM-500

Title

Vol.
I

Page
C.1.2

Rev.

Date
10/69

(Blank)

Data Set Name: FINEGRP

Authors: H. C. Honeck, D. R. Finch

Purpose or Use:

The FINEGRP data set contains multigroup cross sections in a detailed isotopic representation with energy spectrum and temperature dependence taken into account. This data set is designed to store cross sections with as high a degree of detail in multigroup structure and energy spectrum as is desired by the user. Various versions of this data set correspond to different multigroup structures and energy spectrums.

Within a version of the FINEGRP data set the data is ordered by 1) isotope, 2) data type, 3) reaction, and 4) temperature. This structure is designed to provide the data for a single isotope in the detail required by the user.

Code modules are available to produce the multigroup cross sections in FINEGRP form from the ENDFB data set, and to take FINEGRP cross sections and reduce them to the representation of the MULTIGRP data set.

Types of Data Stored:

Cross sections for the following types of reactions are stored in the FINEGRP data set:

1. Smooth cross sections for all reactions.
2. Single-level Breit-Wigner resolved and unresolved resonance parameters.
3. Secondary angular distributions.
4. Secondary energy transfer matrices.
5. Secondary angle-energy correlation matrices.
6. Delayed neutron data.
7. Fission product yield and decay data.

JOSHUA

DPSTM-500

Title

Vol. I

Page C.2.2

Rev.

Date 10/69

(Blank)

Data Set Name: MULTIGRP

Authors: H. C. Honeck, D. R. Finch

Purpose or Use:

The MULTIGRP data set provides spectrum-averaged isotopic cross sections in a multigroup format easily utilized by cell spectrum codes. The number of energy groups within a version of the MULTIGRP data set is generally as few as possible to provide adequate resolution of the cell energy spectrum and cross sections. Standardized versions of the MULTIGRP data set are available for general usage and correspond to energy spectrums that represent reasonable approximations to the exact spectrum for a particular reactor concept.

Within a version of the MULTIGRP data set the data is ordered by 1) isotope, 2) temperature, 3) energy, and 4) data type. This ordering corresponds to the order in which data is utilized by most cell spectrum codes.

Code modules are available to produce multigroup cross sections in MULTIGRP form by combining a detailed Legendre spectrum with cross sections from the FINEGRP data set. Several cell spectrum codes will interface to utilize the cross sections in the MULTIGRP data set.

Types of Data Stored:

The types of data stored with MULTIGRP data set are the following:

1. Smooth cross sections.
2. Total and delayed fission spectrums.
3. Secondary energy transfer matrices.
4. Delayed neutron production data.

JOSHUA

DPSTM-500

Title

Vol. I	Page C.3.2
Rev.	Date 10/69

(Blank)

Data Set Name: FEWGRP

Author: J. W. Stewart, H. C. Honeck

Purpose or Use:

The FEWGRP data set provides an interface between cell or super-cell calculational modules and reactor calculational modules (e.g., static, kinetic, burnup). The cross sections in the FEWGRP data set are generally for group structures numbering less than ten groups. FEWGRP cross sections are generated by convoluting the MULTIGRP cross sections with a multigroup cell spectrum and averaging over the cell compositions.

Each version of the FEWGRP data set corresponds to a fixed energy group structure. Within a version the cross sections are ordered by 1) lattice, 2) assembly within the lattice, and 3) data type.

Types of Data Stored:

The data provided in the FEWGRP data set are the following:

1. Macroscopic cross sections by energy group.
2. Microscopic cross sections by reaction type, isotope and energy group.
3. Delayed neutron parameters by delayed family and energy group.
4. Anisotropic scattering kernels by energy group and Legendre order.

JOSHUA
DPSTM-500

Title

Vol. I	Page C.4.2
Rev.	Date 10/69

(Blank)

Data Set Name: ASSEMBLY

Author: H. C. Honeck, C. E. Bailey

Purpose or Use:

This data set contains the geometric and material descriptions of fuel, targets, and control assemblies and their associated moderator, that is, everything that is within the boundary of a cell. Data are catalogued by cell. The cells are later put together by RAHAB to form a lattice.

Types of Data Stored:

Four records are usually required to describe a cell:

1. A list of the materials used, their temperatures and densities. Several built-in density formulas permit calculation of density as a function of temperature and atom fractions.
2. A list of concentrations of isotopes in the materials. These concentrations can be specified in units of atoms/barn-cm., atom fraction, weight fraction, gm/ft., gm/cc., ppm, etc; that is, in whatever units are natural to the user.
3. Geometric description of an inner sub-assembly. Allowed geometry types are annular (with or without ribs) or cluster.
4. Geometric description of an outer sub-assembly. Allowed geometry types are hexagonal and square (not yet implemented).

The above geometric descriptions are the engineering descriptions as opposed to the mathematical descriptions used in the calculations. RAHAB prepares the mathematical description at execution time by compressing or subdividing (radially and azimuthally) the regions in the engineering description.

JOSHUA

DPSTM-500

Title

Vol.
I

Page
C.5.2

Rev.

Date
10/69

(Blank)

JOSHUA

DPSTM-500

Title Data Set Abstract - RILIB

Vol. I	Page C.6.1
Rev.	Date 10/69

Data Set Name: RILIB

Authors: D. A. Sharp - H. C. Honeck

Purpose or Use:

This data set contains the results of resonance integral calculations for cells catalogued on the ASSEMBLY data set. It is assumed that resonance rates can be calculated for each cell independently (in an infinite uniform lattice).

Types of Data Stored:

The data are resonance capture and fission rates, and the self-shielding factors based on a unit slowing-down density in the cell. The data for each cell are tabulated by region, isotope, and resonance. Unresolved rates (captures/unit lethargy) are given at a set of linearly-connected points (rate, lethargy). Concentrations and temperatures are included in the data so that simple formulas can be used to correct the rates for minor variations in these quantities.

JOSHUA DPSTM-500	Title	Vol. I	Page C.6.2
		Rev.	Date 10/69

(Blank)

Data Set Name: LATTICE

Author: H. C. Honeck

Purpose or Use:

This data set contains the results of lattice calculations. Because of the large amount of data produced by a module such as RAHAB, considerable care is required to retain only the minimum essential for later use.

Type of Data Stored:

The data is catalogued by lattice. For each lattice there are six major groups of data:

1. Zero-buckling flux by region and group.
2. Slowing-down density by cell and group.
3. Legendre fluxes ($P_0 - P_3$) by group for the homogenized lattice at a variety of bucklings.
4. Few-group flux by cell, region, and group.
5. Integral parameters and other summary characteristics of the lattice.
6. Fission power by cell and sub-region.

Additional Comments:

Items 1 - 3 can be used to prepare new few-group cross sections without repeating the RAHAB calculation. Item 3 can be used to weight the data on FINEGRP and prepare a compressed MULTIGRP data set. Item 4 can be used to further compress the FEWGRP data set or prepare lattice average few-group sets from the cell averaged set. Item 6 is intended for input to engineering modules.

JOSHUA
DPSTM-500

Title

Vol.
I

Page
C.7.2

Rev.

Date
10/69

(Blank)

Contents

Page

D.1	-	ETØJ	D.1.1
D.2	-	JMPG1	D.2.1
D.3	-	CREØL	D.3.1
D.4	-	RAHAB	D.4.1
D.5	-	BLEAK	D.5.1
D.6	-	PIDLE	D.6.1
D.7	-	SARC	D.7.1
D.8	-	DAMMET	D.8.1
D.9	-	ØNEDX	D.9.1
D.10	-	BURP	D.10.1
D.11	-	WIGLE	D.11.1
D.12	-	JGAUGE	D.12.1

JOSHUA
DPSTM-500

Title

Vol. I

Issue D.O.2

Rev.

Date
10/69

(Blank)

Module Name: ETØJ (ENDF/B TØ JØSHUA)

Author: D. R. Finch

Purpose:

The ETØJ module averages isotopic pointwise neutron cross section information from the ENDFB data set into isotopic multigroup neutron cross sections in the FINEGRP data set. Averaging is performed over any arbitrarily defined energy group structure and weighting spectrum, or over any energy group structure and weighting spectrum stored in the FINEGRP data set. Various input options are available for specifying weighting spectrums.

ETØJ processes any information in the ENDF/B format into multigroup cross sections consistent with the definitions for the FINEGRP data set. Output from the module will be all or part of the following depending on information given in the ENDFB data set.

- Smooth cross sections for all pertinent reactions.
- Resolved and unresolved resonance parameters in the single level Breit-Wigner representation with or without interference scattering times.
- Secondary angular distributions up to P_5 Legendre order for elastic and inelastic scattering and for (n,2n) and (n,3n) reaction.
- Secondary energy transfer matrices of P_5 Legendre order for inelastic scattering and for $^0(n,2n)$ and (n,3n) reactions.
- Secondary angle-energy correlation matrices up to P_5 Legendre order for elastic scattering.
- Energy spectrum of prompt fission neutrons.
- Neutron yield per fission for prompt fission.
- Fission product yields.
- Fission product decay constants for all decay modes.
- Energy spectrum of delayed fission neutrons for each delayed neutron family.
- Neutron yield per fission for each delayed neutron family.
- Decay constant for each delayed neutron family.

JOSHUA

DPSTM-500

Title Physics Module Abstract - ETØJ (ENDF/B
TØ JØSHUA)

Vol. I	Page D.1.2
Rev.	Date 10/69

Status:

Experimental

Modules Used:

None

Data Sets:

1. INPUT.ETØJ.STANDARD.FØRM
2. ENDFB
3. FINEGRØUP

Restrictions:

Limitations built into the module are the following:

Maximum number of multigroups = 1000

Maximum number of temperatures = 20

Highest angular dependence = P_5 Legendre order

Maximum number of delayed neutron families = 20

Module Name: JMPG1

Author: H. C. Honeck

Purpose:

The module creates a FINEGRP library from HAMMER cross section card decks. A slight modification to the energy group structure in group 54 is made to eliminate the overlap between the MUFT and THERMOS groups. Elastic P_0 and P_1 transfer matrices based on the MUFT σ_{s0} and σ_{s1} are computed¹ and merged with the thermal scattering matrices.

Status:

Production

Modules Used:

None

Data Sets:

System Input File, FINEGRP

Restrictions:

Can only be used with HAMMER cross section decks and group structure.

JOSHUA

DPSTM-500

Title

Vol. I	Page D.2.2
Rev.	Date 10/69

(Blank)

Module Name: CREØLAuthor: H. C. Honeck, R. A. PriestPurpose:

CREØL Compresses and REOrders the FINEGRP Library (data set) and prepares a MULTIGRP data set. The FINEGRP data set contains high resolution multigroup cross section data ordered by isotope and reaction type. It is frequently desirable to compress this group structure to fewer groups and create a MULTIGRP data set ordered by isotope and group, that is, an order best suited for calculational modules such as RAHAB. Since RAHAB can trade off groups for regions, the compressed data set will permit more regions to be used. The weighting spectrum is obtained from a library of spectra produced by B_1 calculations of homogenized lattices. CREØL also combines reaction types (such as elastic and inelastic into scattering) to further compact the data.

Status:

Production

Modules Used:

None

Data Sets:

INPUT, FINEGRP, MULTIGRP, LATTICE

Restrictions:

1. 100 groups on FINEGRP and MULTIGRP
2. P_3 weighting spectrum

JOSHUA
DPSTM-500

Title

Vol.
I

Page
D.3.2

Rev.

Date
10/69

(Blank)

Module Name: RAHAB

Author: H. C. Honeck, C. E. Bailey, D. A. Sharp, W. H. Hightower,
R. A. Priest, C. E. Parker, C. E. Henderson

Purpose:

RAHAB is the control module and series of sub-modules in the JOSHUA physics sub-system concerned with the calculation of lattice fine-structure. RAHAB performs the same function in reactor design as the HAMMER code, and uses the same basic theory (integral transport theory). The main features of RAHAB are:

- Single multigroup structure incorporating the thermal, epithermal, and fast energy regions. The number of groups is variable from 1 - 100.
- Full two-dimensional geometry capability including full or subdivided annular and hexagonal regions, clusters, multicell (mixed) lattices.
- Multigroup Monte Carlo option (not yet completed)
- Buckling, time eigenvalue, or concentration (not yet completed) searches
- Simplified input in engineering units and terms

RAHAB takes most of its input from catalogued cell descriptions on the ASSEMBLY data set and cross sections from the MULTIGRP data set. The output fluxes and integral parameters are placed on the LATTICE data set. Few group cross sections are placed on the FEWGRP data set.

Status:

Experimental

Modules Used:

RASP, RAGE, RARES, RABINX, RATMAT, RABIT, RABLIN, RABBLE, RABED

Data Sets:

INPUT, FINEGRP, MULTIGRP, ASSEMBLY, RILIB, LATTICE, FEWGRP,
RAHAB (temporary storage)

Restrictions:

1. Up to 100 groups
2. Up to 100 sub-regions
(groups and sub-regions are traded-off so that it may not be possible to run a full 100-100 problem)
3. Up to 10 group output and edits.

Module Name: BLEAK

Authors: H. C. Honeck, R. A. Priest

Purpose:

BLEAK is a control module which allows the B₁ LEAKage calculational modules, RABLIN and RABBLE, in RAHAB to be used as a stand-alone calculational procedure. The module performs a B₁ leakage calculation for a homogeneous medium. Four search options are allowed:

- k_{eff} calculation
- buckling search
- time eigenvalue search
- material search (not yet completed)

Few-group material cross sections can be edited.

Resonance integral data for cells will be used in the B₁ calculation if the data is on the RILIB data set.

Status:

Experimental

Modules Used:

RABLIN, RABBLE

Data Sets:

INPUT, MULTIGRP, RILIB, LATTICE, FEWGRP

Restrictions:

1. P₃ fluxes always computed.
2. Up to P₃ scattering matrices can be used.
3. Up to 100 groups input from MULTIGRP.
4. Up to 10 groups output to FEWGRP.

JOSHUA

DPSTM-500

Title

Vol.
I

Page
D.5.2

Rev.

Date
10/69

(Blank)

Module Name: PIDLE

Author: H. C. Honeck, R. A. Priest

Purpose:

PIDLE is used to check thermal scattering kernels on the FINEGRP data set by performing a Proper Integration of the Diffusion Length Equation. The inverse diffusion length is computed as a function of absorption and the results analyzed by least-squares fitting of various order to obtain the diffusion and cooling coefficients.

Status:

Production

Modules:

None

Data Sets:

INPUT, FINEGRP, ØUTPUT

Restrictions:

1. Up to P_3 scattering matrices
2. Up to 100 groups.

JOSHUA

DPSTM-500

Title

(

Vol.
I

Page
D.6.2

Rev.

Date
10/69

(Blank)

Module Name: SARC

Author: D. A. Sharp

Purpose:

This module will provide a way to compute effective resonance integrals in individual cells without the necessity of executing the complete RAHAB system.

The input to the module includes the detailed geometrical and material characteristics of the cell. Prior to the calculation, a search is made to determine if results already exist for the particular problem. If this is the case, execution is bypassed. If not, execution proceeds, with the cell regions being subdivided and/or homogenized, based on input options. Conversion of dimensions and isotopic concentrations into consistent units occurs automatically, along with homogenization of ribs and cladding. Resonance integrals are subsequently computed on a regionwise basis. The user may optionally specify a particular lethargy mesh structure and/or a collision probability interpolation procedure for use in the computation.

Status:

Experimental

Modules Used:

RASP, RARES

Data Sets:

ASSEMBLY, FINEGRP, RILIB, RAHAB (temporary storage)

Additional Output:

1. A cell description, including geometrical and material character items.
2. Regionwise resonance absorption and fission rates permit total source for each resonance of each isotope.
3. Regionwise concentrations and temperatures for each isotope having a resonance structure.
4. Cell-averaged resonance integrals, shielding factors, and effective cross sections for each resonance of each isotope.
5. Cell-averaged total resonance integrals and shielding factors for each isotope.

JOSHUA
DPSTM-500

Title

Vol.
I

Page
D.7.2

Rev.

Date
10/69

(Blank)

Module Name: DAMMET (Ddelete, Alter Mode, and Merge Tapes)

Author: D. R. Finch

Purpose:

The DAMMET module is a modified version of a code of the same name distributed by the National Neutron Cross Section Center for use in handling ENDF/B data. This module is used in JOSHUA to transfer data from ENDF/B library tapes to the ENDFB data set.

Status:

Production

Modules Used:

None

Data Sets

ENDFB

JOSHUA

DPSTM-500

Title

Vol.
I

Page
D.8.2

Rev.

Date
10/69

(Blank)

Module Name: ØNEDX

Author: J. W. Stewart

Purpose:

This module obtains numerical solutions for each of the following types of neutron diffusion problems by use of the central-difference approximation of the gradient operator:

- (a) eigenvalue-eigenvector problem (k_{eff} calculation),
- (b) external neutron source problem,
- (c) Green's function flux mode generation.

The module admits one-dimensional geometries in slab, cylindrical, or spherical geometry. Either the direct or adjoint solution may be obtained. Dynamic variable dimensioning allows a variable number of energy groups, materials, mesh points. Allowable boundary conditions include zero flux or current on external surfaces. A point relaxation inner iteration is used with flux relaxation and/or extrapolation.

Status:

Experimental

Modules Used:

None

Data Sets:

INPUT.1DX.JØB.?JØB
INPUT.1DX.CASE.?CASE
INPUT.1DX.FLUX.?NAME
INPUT.1DX.FLUX.?NAME
FEWGRP.?VERSN.?LATT.?ASSY.MACRØ.?NCASE

ØUTPUT.1DX.CASE.?CASE.FLUX
ØUTPUT.1DX.CASE.?CASE.FLUXPLØT
ØUTPUT.1DX.CASE.?CASE.MØDE.?NØ
ØUTPUT.1DX.CASE.?CASE.MODEPLØT.?NØ

Restrictions:

Core required is approximately 210K bytes including JØS requirement.

JOSHUA
DPSTM-500

Title

Vol.

I

Page

D.9.2

Rev.

Date

10/69

(Blank)

Module Name: BURP

Authors: C. E. Bailey
J. W. Stewart

Purpose:

This module provides the capability of predicting the time-response of an SRP reactor subjected to an external perturbation. A lumped parameter heat transfer/hydraulics model is coupled to a point kinetic neutronics model*. Coolant, fuel, and moderator temperatures and reactor power are calculated as a function of time. The engineering and neutronic kinetic equations are integrated over time by the Runge-Kutta-Gill method.

Status:

Production

Modules Used:

None

Data Sets:

INPUT.BURP.JØB.?JØB
INPUT.BURP.CASE.?CASE.RECØRD1
INPUT.BURP.CASE.?CASE.RECØRD2

ØUTPUT.TABLE.BURP.CASE.?CASE
ØUTPUT.PLOT.BURP.CASE.?CASE

Restrictions:

Maximum number of cases per job is 49. Core requirement is approximately 240K bytes including the JØS requirement.

* C. E. Bailey, DP-1099, to be published.

JOSHUA
DPSTM-500

Title

Vol. I	Page D.10.2
Rev.	Date 10/69

(Blank)

Module Name: WIGLE

Author: J. W. Stewart

Purpose:

This module represents a modification of the program WIGLE-40*. The one-dimensional space-time solution of neutron diffusion equation is obtained for two energy groups in slab geometry. Zero, one, or six delayed neutron families are allowed. A relatively generalized form of perturbation to the two-group diffusion parameters is allowed. A simple power feedback to the thermal removal cross-section is included. Dynamic variable dimensioning is used.

Status:

Experimental

Modules Used:

No other modules are called, however, interfacing with ØNEDX is provided, by option, for the definition of initial flux distributions.

Data Sets:

INPUT.WIGLE.?JØB
INPUT.WIGLE.FLUX.?NAME
ØUTPUT.1DX.CASE.?CASE.FLUX
FEWGRP.?VERSN.?LATT.?ASSY.MACRØ.?NCASE

ØUTPUT.WIGLE.?JØB.FLUX.?NTSTEP
ØUTPUT.WIGLE.?JØB.PØWER.?NTSTEP
ØUTPUT.WIGLE.?JØB.FINALPAR

Restrictions:

Core required is about 210K bytes including JØS requirements.

- * M. E. Radd, Wigle-40, A Two-Group, Time-Dependent Diffusion Theory Program for the IBM 7040 Computer, IDO-17125, December 1965.

JOSHUA
DPSTM-500

Title

Vol. I	Page D.11.2
Rev.	Date 10/69

(Blank)

Module Name: JGAUGE

Authors: P. L. Ames
J. W. Stewart

Purpose:

This module is a modification* of the GAUGE program**. The two-dimensional, multigroup neutron diffusion (eigenvalue-eigenvector) equation is solved in hexagonal geometry. Modifications to GAUGE include geometry specification, few-group neutron cross-section specification, and power and average cross-section editing by reactor "ring".

Status:

Experimental

Modules Used:

None

Data Sets:

INPUT.JGAUGE.JØB.?JØB
INPUT.JGAUGE.CASE.?CASE
INPUT.JGAUGE.GEØM.?NAME
FEWGRP.?VERSN.?LATT.?ASSY.MACRØ.?NCASE

ØUTPUT.JGAUGE.CASE.?CASE.
JGAUGE.XSECTION.CASE.?CASE.RING.?IRING

Restrictions:

Core requirement is approximately 210K bytes including JØS requirements. This module is stored as MEMBER=JGAUGE on PDS=SRL.JØHN3865.

* P. L. Ames, DPST-69-493, "GAUGE" Is Good For You.

** M. R. Wagner, GA-8307, "GAUGE, A Two-Dimensional Few-Group Neutron Diffusion-Depletion Program for a Uniform Triangular Mesh."

JOSHUA
DPSTM-500

Title

Vol.
I

Page
D.12.2

Rev.

Date
10/69

(Blank)

JOSHUA

DPSTM-500

Title Part Z - Reprints of Publications

Vol.
I

Page
Z.O.1

Rev.

Date
10/69

Contents

- Z.1 - JOSHUA - A Reactor Physics Computational System (DP-MS-68-94).
- Z.2 - Data Management for Multi-access Computational Systems (DP-MS-68-92).
- Z.3 - Cross Sections in the JOSHUA System (DP-MS-69-74).

JOSHUA

DPSTM-500

Title

Vol. I	Page Z.O.2
Rev.	Date 10/69

(Blank)

JOSHUA - A REACTOR PHYSICS COMPUTATIONAL SYSTEM*

H. C. Honeck, J. E. Suich,
J. C. Jensen, C. E. Bailey, and J. W. Stewart

Savannah River Laboratory
E. I. du Pont de Nemours & Company
Aiken, South Carolina 29801

Contract AT(07-2)-1 with The
United States Atomic Energy Commission

March 5, 1969

Presented at the Conference on The Effective
Use of Computers in the Nuclear Industry,
April 21-23, 1969, Knoxville, Tennessee.

* The information contained in this article
was developed during the course of work
under Contract AT(07-2)-1 with the U. S.
Atomic Energy Commission.

H. C. Honeck, J. E. Suich,
J. C. Jensen, C. E. Bailey, and J. W. Stewart

Savannah River Laboratory
E. I. du Pont de Nemours & Company
Aiken, South Carolina 29801

1. INTRODUCTION

The third generation of computers at most nuclear reactor design installations offers new possibilities to the developers and users of reactor computing codes. In particular, high-speed central processing units are commonplace; almost all systems have large quantities of direct access storage, and many new devices are available to improve the accessibility of the computing system to the reactor designer - such as alphameric display terminals and cathode ray tubes with full graphic capability. The trend in reactor computing systems^{1,2} has been toward utilization of these features of third-generation computers. At the Savannah River Laboratory, the JOSHUA system is being developed to capitalize on the experience gained with earlier systems, and to include a remote terminal capability.

This paper is a brief introduction to the JOSHUA system. The remainder of this section describes the major considerations in designing such a system and the organization needed to create the system. The operating system is described in the following section, and the last section outlines the scientific areas where the system will be first applied.

1.1 Elements of the System

JOSHUA is an operating system which provides the data handling services to accomplish extensive, iterative scientific calculations on a large semipermanent data base. JOSHUA provides standard, "cataloged procedures" for solving routine problems, and allows easy redefinition of these procedures for experimental computation. The user is permitted to intervene between steps of a computation, display results, and possibly redirect the procedure. Individual display terminals for problem setup and output display are central to the design philosophy of JOSHUA, although duplicate facilities are provided for batch mode use where practicable.

The computational system is based on a library of "computational modules" arranged in a variety of ways to suit the application, and a base of "named data records" of significant computed results and measured values created and displayed by either the user or modules.

A "named data record" is the smallest collection of logically related information that the user considers a coherent whole. The group of data elements is assigned a unique, qualified name in the natural language of the user's discipline. The logical relationship between records, suggested by the common name qualifiers, is accounted for in the system by cataloging the records as a tree structure. A limited, generally useful set of records is maintained as a protected, read-only "standard" data base. Data created by each active user are cataloged separately. A user may duplicate standard names without overwriting the standard data for other users. Also included in the system is a description of record types so that, upon reference to a record by the name, the data elements can be identified in the display.

A "computational module" is a logically complete, dynamically executable computer procedure, which receives input from named data records, and places results in other named data records. Since named data records are stored in the system until deleted by the user, they represent information of lasting value (significance) - they are not "scratch," or intermediate computed results. Thus the concept of a logically complete module is equivalent to the concept of a procedure which produces new information of lasting value from old information also of lasting value. Modularity in the conventional sense is achieved by restricting data references to previously defined named data record types.

A "cataloged procedure" is an automated sequence of user data manipulation and module execution that satisfies a particular application. In the simplest case, this will be no more than the creation of input named data records, execution of a single module, and display of output records. In more complicated cases, however, procedures may involve execution of several modules, perhaps even iteratively. Thus, a procedure is equivalent to a program written in a problem-oriented language defined by the functions of the module and terminal command.

1.2 Organization

The JOSHUA Operating System could be used in many areas of scientific computing at Savannah River. Extensive planning and organization is necessary before embarking on a large program to develop applications modules. This section describes this organization and the plans for the first two years of the program.

If the modules and data sets (subsystem) for a specific area of scientific computing (e.g., reactor physics) do not interact with those of other subsystems, they can be developed independently of the other subsystems. If, on the other hand, several subsystems interact, they must be designed to be compatible. Thus, it is necessary to plan the overall system at an early stage and then to develop individual subsystems. Further, it is essential to organize the personnel involved both in the planning activities and in the code development work to establish lines of communications and areas of responsibility. This organization is illustrated in Fig. 1.

The JOSHUA Task Force, organized in September 1968, consists of representatives from each Division at Savannah River which will utilize the JOSHUA system. Their initial assignment was to:

- survey all scientific computing at Savannah River
- project future computing requirements
- identify all subsystems and the extent of their interaction
- assign priorities to subsystems and individual modules of subsystems
- develop a schedule to implement each subsystem and the overall system.

This phase of the planning was completed in November 1968.

One of the first results of the Task Force activities was the identification of the following major subsystems which had a high priority for inclusion in the first phase of JOSHUA applications development.

- Reactor Physics

Fig. 1 Organization of the JOSHUA Project

- Reactor Engineering
- Reactor Kinetics
- Plant Simulation and Forecasting

It was evident that the Reactor Physics, Reactor Engineering, and Reactor Kinetics subsystems interact so strongly that they must be designed concurrently. The Plant Simulation and Forecasting subsystem interacts with the Reactor Physics subsystem, but the interaction is weak enough that it can be neglected at the present time. The two subsystems will be tied together later. Other areas such as Chemistry, Chemical Separations, Materials, and Business Data Processing interact only weakly with each other and with the four major subsystems. These were assigned low priorities for inclusion in the first phase of JOSHUA and will not be discussed further here. Areas of highest priority within each subsystem were also identified, and a development schedule was laid out.

The actual implementation of several subsystems began in December 1968 as the joint responsibility of the sponsoring Division (e.g., Theoretical Physics Division, Reactor Engineering Division, etc.) and the Computer Applications Division. The sponsoring Division provides complete specifications for the subsystem; the Computer Applications Division develops the modules and data sets from these specifications and ensures that data sets common to several subsystems are compatible.

2. OPERATING SYSTEM

A brief description of the JOSHUA Operating System (JOS) is given in this section. The emphasis is on facilities provided to the user by JOS and how these facilities are used. How the JOS actually provides these facilities is described in a paper³ to be given at this conference.

A simplified diagram of a computing system is shown in Fig. 2. The large block represents the central processing units and fast core storage. "ØS" indicates that a portion of the core storage must be devoted to an operating system which controls the computer system for the individual tasks above it. "Batch" indicates the possibility of multiprogramming (or multiprocessing) a background stream of jobs outside the reactor computing system (for example, FORTRAN compiles, nonreactor computations, etc.). Programming aids or control programs that must be provided to interface the reactor computing system with the operating system are subdivided by dashed lines. Separate systems are provided for terminal operation and job execution. "Path" indicates that a program may reside in core to control the flow of computations between the various possible modules. Finally, the modules themselves must be brought into core for execution.

Fig. 2 Simplified Computing System

To the right of the diagram are the various devices involved in the flow of data in and out of the computer and between the various segments of core. The two diagrams at the bottom indicate that the batch mode area utilizes the usual off-line printer and card reader of a stacked job processor. The display terminals and remote keyboards show the possibility of allowing a reactor designer to gain easy on-line access to the reactor computing system, and to make use of the graphic capabilities of the new terminals in setting up complex input data. All communication between the job control path and module is via data and programs which reside on a direct access storage device (DASD).

Before discussing how these devices are used, it is necessary to describe how data are handled by the JOS.

5

2.1 Named Data

Fig. 3 illustrates the problem of handling large quantities of data in a natural way. The figure shows a typical multigroup data set structured (as in the user's mind) in a hierarchical fashion, that is, as a tree structure. For example, the user may indicate his interest in a data record containing values of the fission cross section for ^{238}U in the HAMMER multigroup data set. He cannot, without considerable research, refer to this data record as, for example, record 53 on track 108 of the disk. He can, however, easily refer to this data record with the name

MULTIGRP.HAMMER.U238.XSECTION.FISSION

if facilities are provided for using names of this kind in FORTRAN programs, and relating the names with locations on a direct access device. These facilities are provided in JOS by a precompiler and a data manager. Similar facilities are provided in the NOVA system.⁴

Fig. 3 Typical Multigroup Data Set Structure

A data record referred to by name is called a named data record and is defined by a record name and a list. The list defines the structure of a record and is similar to the usual FORTRAN I/O list, except that all indexing must be shown explicitly and all nonconstant index limits must appear at the beginning of the list. For example, if the user wants to write the record named above from the array, $\text{XF}(N), N=1, \text{NEG}$, a suitable list would be:

$\text{NEG}, (\text{XF}(N), N=1, \text{NEG})$

The FORTRAN IV language does not provide any direct facility for handling named data records. It is possible to provide FORTRAN-callable subroutines with the record name and list as arguments. This becomes awkward, however, if arbitrary length names and lists are to be allowed. Instead it was decided to extend the FORTRAN language and use a precompiler to interpret the new statements. The basic extension is the addition of a new read/write statement. The normal unformatted read/write statement has the form:

$\text{READ}(n, \text{ERR}=m) \text{ list}$

where n is a unit number and m is an error return statement number. The new read statement replaces the unit number with the record name and signals that it is a special JOSHUA statement with a J in column 1:

6

```
J      READ(record name,ERR=m) list
```

For the example above the read statement would be:

```
J      READ('MULTIGRP'.'HAMMER'.'U238'.'XSECTION'.'FISSION',  
1      ERR=m) NEG,(XF(N),N=1,NEG)
```

The use of ERR is optional. The manner in which the precompiler handles these statements and how the data manager relates names to direct access storage locations is discussed later. Before doing that a more complete description of record name construction is required.

The elements of the record name (e.g., MULTIGRP,FISSION) are called qualifiers. A qualifier may be:

- a literal string of 8 or less characters
- an integer
- a FORTRAN variable whose value is either an integer or an 8-character literal string.

For example, the name used in a JOSHUA statement might be:

```
'MULTIGRP'.3726.DISØ(N)'.XSECTION'.RTYPE
```

The quotes around MULTIGRP and XSECTION mean "the literal value of." The integer 3726 refers to a particular version (3726 might be the user's badge number) of the multigroup data set. DISØ(N) is the Nth location in the REAL*8 array named DISØ and contains an 8-character isotope name. RTYPE is a REAL*8 variable containing the reaction type name (e.g., FISSION,CAPTURE).

It is clear that the same record name and list structure can be used for all isotopes and all cross section reaction types. Thus, we can write a general name:

```
'MULTIGRP'.'?VERSN'.'?DISØ'.'XSECTION'.'?RTYPE'
```

Qualifiers enclosed in question marks are called variable qualifiers and a record name containing variable qualifiers is called a record name type. The combination of a record name type and a list defines a named data record type. The example above indicates that the same name and list structure are used for all reaction types, all isotopes, and all versions of the multigroup data set. Obviously, at execution time the variable qualifiers must be replaced with literals, integers, or variables having specific values which refer to a unique named data record.

The first qualifier in a record name is called the data set name (e.g., 'MULTIGRP'). The data set name must not be a variable qualifier and must start with an alphabetic character. The data set name provides the user with a natural major subdivision of the data base. The data set name is used to allocate direct access storage space.

The hierarchical tree structure of data record names illustrated in Fig. 3 provides a convenient and powerful method for maintaining catalogs of record names and pointers. A FORTRAN IV subroutine provides the facility for maintaining and searching catalogs of record names and associated data pointers. These catalogs are themselves part of the data base and are stored on disk. The catalog search subroutine is called from the user module each time a record is to be read or written. The appropriate statements are automatically inserted into the program by the precompiler. 7

2.2 Precompiler

Let us assume that a module has been programmed using JOSHUA read/write statements. The source deck is precompiled to replace JOSHUA statements with a set of FORTRAN IV statements. The resulting source deck can then be compiled with the system's FORTRAN IV compiler. The intermediate deck is not punched and the programmer always works with the original source deck.

The precompiler is written in FORTRAN IV and will process four types of JOSHUA statements (each with a J in column 1).

- module statement
- read/write statements
- shorthand statements
- variable typing statements

The module statement arranges for communication of control parameters between the module and the batch executor. The read/write statements have been previously defined. The shorthand statements allow the programmer to predefine a frequently used string of qualifiers and use an abbreviated notation thereafter. In the previous example, if many references are made to the same version of the multigroup data set, it would be convenient to include the following statement near the start of the program.

```
J      'MULTIGRP'. 'HAMMER' -> V
```

(The arrow is punched as a combination of the - and > characters). A read statement given later in the program could then be written:

```
J      READ(V. 'U238'. 'XSECTION'. 'FISSION')
1      NEG, (XF(N), N=1, NEG)
```

The action of the precompiler is to move the character string defined on the left of the arrow into the portion of the name given by V so that the qualifiers used in the shorthand statement need not be fully defined at the time the shorthand statement is given. The shorthand statement is used only to abbreviate the first part of the name. The number of qualifiers is arbitrary. The shorthand symbol (e.g., V) may be any 1-6 character alphameric string and will not conflict with any FORTRAN variable. Another example is:

```
J      'MULTIGRP'. 'HAMMER'. DISØ(I). 'XSECTION' -> XF
-----
DØ 10 I=1, ISØ
J      READ(XF. 'FISSION') NEG, (XF(N, I), N=1, NEG)
10 CØNTINUE
```

The typing statement defines the type (e.g. REAL*8, etc.) of each non-4-byte FORTRAN variable used in a read write statement. Indexing of arrays is not required. For example:

```
J    REAL*8 A,B,C
J    INTEGER*2 I,J
```

These statements are used in addition to the usual FORTRAN typing statements. A later version of the precompiler will use only the usual FORTRAN typing statements and the special typing statements will not be required.

The shorthand and type statements are used by the precompiler to interpret the read/write statements and will not be present in the intermediate source deck. The precompiler replaces each read/write statement with a group of FORTRAN IV statements. These statements:

- define qualifiers as 8-character strings
- compute the length of the list
- call the data management subroutine which locates the record name in its catalogs and passes back the associated pointers to the data
- issue a direct access read/write statement using the pointers and list (This statement is a non-ASA standard FORTRAN statement but is a standard IBM 360 FORTRAN IV statement.).
- transfer to error statements if required.

2.3 Data Management

The last two sections have outlined how data records are constructed and named, and how a module can read or write records from direct access storage. If the computer were dedicated to a single user, little more data management capability would be required. However, the computer will be used concurrently for many jobs to achieve reasonable efficiency. At any one time, one or all of the following jobs may be in process:

- manipulation (displaying, modifying, creating) of the data sets from several terminals
- execution of modules initiated from terminals
- execution of modules initiated from the batch
- execution of non-JOSHUA batch jobs.

A multilevel, multiuser access method is used to maintain the integrity of the data base during these concurrent operations. Briefly, three levels of data sets are used:

Job	- temporary data produced as output from the execution of a module. These data can be read or written by all modules comprising the user's job. At the completion of a job, these data can (at the user's option) be selectively merged with the user data sets.

User - permanently stored data for a single user which can be read by the user's modules. These data can be written only from a terminal or the batch input.

9

Standard - permanently stored data which can be shared (read) by all users. These data can be written only by JOS programs.

Any read operation searches all three levels in the above order, and transmits the data found at the lowest level. For example, a module might read data from a standard data set (because no data with the same name were found in the job or user data sets), modify the data, and write the new data with the same name. The new data are written on the job data set, and all future references to that name will retrieve the data from the job data set (since it is searched first). The original data still remain in the standard data set, but it appears to the module that the data have been overwritten. If the module deletes the data, those data in the job data set are deleted, and the standard data are again available to the module.

2.4 Terminal Facilities

The JOS provides several facilities for manipulation of the data base and execution of modules, for both terminal and batch operation. Because the facilities for batch operation simulate those for terminal operation, only the terminal facilities Templating, Data Access, Module Execution, and Cataloged Procedure will be discussed here.

The data stored on a DASD are unformatted, that is, in binary form. Templates format the data so they may be displayed (or printed) in an easily readable form. A template is analogous to a FORTRAN FORMAT statement. A template is defined for each named data record type, and the information contained in the template is stored as a data set. Because there will be many data records of the same type, little additional storage is required for templates.

Six instructions allow the user to access his data from a terminal: LIST, DISPLAY, COPY, CREATE, MODIFY, and DELETE. LIST allows the user to look at the catalogs to determine what records exist in each data set. In using these commands the user gives a record name to be read, written, or deleted. Normally, reading follows the same hierarchical search through the job, user, and standard data sets, as if the request came from a module. The user can force a search only of his user data set by prefixing the record name with his user number. Similarly, the search of the standard data sets can be forced by prefixing the record name with the qualifier 'STD'. All writes refer to the user data sets.

After the input data are set up, the JOS can be commanded to execute a given module. The JOS enters the module into the OS batch queue for execution. Following execution the user can display the output records produced by the module. These records will be in the job data set.

Associated with each module is a series of defined input records and output records. A cataloged procedure assists the user in executing a module. The cataloged procedure is stored as a systems data set and contains the names of the input records, the module name, and the names of the output records. When the user executes a procedure, the JOS automatically displays the proper blank records to be created (or existing records to be modified), assists the user in setting up variable qualifiers in the names, and executes the module. A second procedure automatically displays the output records.

3. APPLICATIONS

10

The schedule for the development of major application subsystems was given in the Introduction, Section 1. A brief discussion of each subsystem is given here.

In developing each subsystem, the decision to convert an existing program or develop a new program depends on the following factors.

- Is an existing code available?
- Is adequate documentation available?
- Will the theory and techniques in the existing code be adequate or will improvements be required during the next few years?
- Are better techniques (theoretical, numerical, or programming) now available than are used in an existing code?
- Would an existing code become a module, part of a module, or several modules?
- How much recoding is required to convert an existing code to run under the JOSHUA system?
- If future improvements to an existing code are anticipated, will they be easier to make if a new code is written now?
- Will a complete recoding yield substantial reductions in computer time?

In nearly every case considered, the decision has been made to develop a new module and utilize the latest theoretical, numerical, and programming techniques.

3.1, Reactor Physics

The minimum reactor physics subsystem at Savannah River requires four groups of modules and data sets:

- Cross section data sets and manipulative modules.
- Lattice physics modules.
- Reactor modules.
- Burnup modules.

Several cross section data sets are planned. The first, ENDF/B, contains point cross section data. The second, a fine-group data set, is derived from ENDF/B by simple weighted averaging and is designed to efficiently contain 100 to 1000 group sets. The third data set, a multigroup data set, is derived from the fine-group set by averaging with computed flux distributions. The multigroup data set is designed to efficiently contain 20 to 100 group sets and will be the primary input for lattice physics modules. Finally, a few-group (1 to 20 groups) data set will contain the cross sections produced by the lattice physics modules and supply the reactor, burnup, and kinetics modules.

The lattice physics modules will be based on collision probability methods (e.g., HAMMER) and the Monte Carlo method. A completely new HAMMER system being developed will feature full two-dimensional geometry (clusters, mixed lattices, exact cell boundaries), flexible energy group structures (1 to 100 groups), and improved leakage calculations. An equivalent multigroup Monte Carlo module is also being developed.

11

The first reactor modules will be two-dimensional diffusion theory modules based on PDQ and/or GAUGE. The source-sink code, HERESY-II, will also be used. Two- and three-dimensional synthesis modules will be included later.

Burnup modules are being developed for both simple lattice burnup and for complete reactor life history calculations.

4.2 Reactor Engineering

The basic requirement of the reactor engineering subsystem is to provide for rapid and efficient design of fuel assemblies and reactor charges and specification of their operating characteristics and limits. Three groups of modules and data sets, closely paralleling the reactor physics modules, are required:

- Basic engineering data sets (e.g., defining thermal conductivities, specific heats) and manipulative modules.
- Fuel element design modules.
- Reactor charge design modules.

The basic engineering data sets will relieve the user from providing information repetitively, and ensure that all calculations are made on a consistent basis.

The fuel element design and reactor charge design modules will be used to calculate the temperature distributions in the various components of the reactor system. Ultimately, modules will be created to provide solutions to the heat diffusion and heat transfer equations in one, two, and three dimensions.

The input information for the engineering modules in almost every application will include results from the reactor physics modules or the kinetics modules, or both. For example, results from the physics modules are necessary to calculate the steady-state temperature distributions, and from the kinetics modules to calculate operating limits.

Because of the strong dependence of the engineering modules on the physics and kinetics modules, the problems of fuel element and reactor charge design are inherently iterative, and suitable for modular computational techniques.

3.3 Reactor Kinetics

The ultimate objective of the kinetic subsystem effort is the capability to analyze the course and consequences of a variety of hypothetical reactor incidents. The kinetic subsystem interacts strongly with both the static reactor physics and static engineering subsystems for the definition of initial reactor conditions. The kinetic neutronics models will include multi-group and delayed neutron effects in multidimensional geometries. The kinetic engineering module will determine all pertinent state variables (e.g., temperature and density of coolant, moderator, and fuel; flow velocity of coolant and moderator; and coolant and moderator void fraction) as functions of space and time. Feedback to the neutronics module will be included via the updating of nuclear parameters (e.g., neutron cross sections and neutronic diffusion

coefficients) as functions of the engineering state variables. The neutronics module will, in turn, provide thermal power feedback information to the engineering module.

12

The kinetic neutronics module will provide several calculational models including a one-dimensional, finite-difference model similar to existing codes, and synthesis models utilizing either continuous or discontinuous spatial trial functions in multidimensional geometries.

The initial kinetic development effort will emphasize the generation of spatial trial functions and the numerical solution of the resulting set of ordinary coupled, linear differential equations for the time-varying model coefficients. Concurrently, an existing, one-dimensional, finite difference code will be modified to provide a standard of comparison for synthesized solutions. These neutronics models will be tested using simple feedback models (e.g., power feedback) but they will also provide for a more detailed feedback treatment. A parallel effort will be made to develop a kinetic engineering model which will later be coupled with the neutronics model.

Special problems, such as the calculation of spatial xenon oscillations will be treated as separate, specialized subsystems but will utilize existing, applicable program blocks (i.e., subroutines, modules, etc.) to treat feedback mechanisms, to define initial conditions, etc.

3.4 Plant Simulation and Forecasting

The Plant simulation and forecasting subsystem will be used to develop accurate time schedules for the use of the Plant facilities, record the impact of scheduled processes on inventories, and calculate the costs associated with Plant operation and the costs of programs and products.

The facility scheduling and inventory managing modules are being developed first; the cost modules will then be developed.

The Plant simulation and forecasting subsystem must be flexible because the users require a wide range of detail. The requirements include:

- Studies where only a facility schedule, with any necessary overlap in facility use, is desired.
- Long-range studies (10 - 20 years), where schedules for major facilities, inventory accounting for major items, and the associated costs are desired.
- Shorter-term studies (6 months - 2 years), where schedules for all facilities, inventory accounting for all items, and the associated costs are desired.

To provide this flexibility, the data base will include several descriptions of a given "campaign," where a "campaign" encompasses all Plant operations and inventories to produce a product or carry out a program.

These descriptions will permit any desired degree of detail to be specified and will permit one general set of scheduling, inventory managing, and costing modules to be used for all types of problems.

ACKNOWLEDGMENT

13

The JOSHUA capabilities now available have been programmed by R. L. Boyce and J. H. Maddox, R. A. Priest, W. H. Hightower, C. E. Parker, and A. H. Dobyns.

The information contained in this article was developed during the course of work under Contract AT(07-2)-1 with the U. S. Atomic Energy Commission.

REFERENCES

1. The Argonne Reactor Computation (ARC) System, USAEC Research and Development Report (ANL-7332), Argonne National Laboratory, November 1967.
2. E. E. Reilly, Jr. and W. H. Turner, "The Automation of Reactor Design Calculations at the Knolls Atomic Power Laboratory," Proceedings of the Conference on the Application of Computing Methods to Reactor Problems, ANL-7050, 1965.
3. J. E. Suich, J. C. Jensen, and H. C. Honeck, "Data Management for Multiaccess Computational Systems," Proceedings of the Conference on the Effective Use of Computers in the Nuclear Industry, Knoxville, Tenn., April 21-23, 1969.
4. H. J. Kopp and D. S. Selengut, "DATATRAN - A Data-Handling Computer Language for a Large Modular Reactor Design System," Proceedings of the International Conference on The Utilization of Research Reactors and Reactor Mathematics and Computation, Mexico, D.F., May 2-4, 1967, Vol. 3, pp. 1460-1472, Report CNM-R-2.

DATA MANAGEMENT FOR MULTIACCESS COMPUTATIONAL SYSTEMS*

J. E. Suich, J. C. Jensen, and H. C. Honeck

Savannah River Laboratory
E. I. du Pont de Nemours and Company
Aiken, South Carolina 29801

March 5, 1969

Presented at The
Conference on The Effective Use
of Computers in The Nuclear Industry
April 21-23, 1969
Knoxville, Tennessee

* The information contained in this article was developed during the course of work under Contract AT(07-2)-1 with the U. S. Atomic Energy Commission.

J. E. Suich, J. C. Jensen, and H. C. Honeck

1

Savannah River Laboratory
E. I. du Pont de Nemours and Company
Aiken, South Carolina 29801

1. THE JOSHUA CONCEPT

A companion paper¹ describes the development of the JOSHUA system at Savannah River Laboratory. Although the system is intended for application to problems in the management of a nuclear reactor facility, the system is quite general. Generality has been lost only in the sense that the design of the system was tailored to the requirements of the application area.

The JOSHUA system provides data and job management facilities for applications having the following characteristics:

- A large set of dynamically executable modules interfaced through direct access storage.
- An extensive data base of hierarchically ordered, unformatted records.
- A high ratio of records to record-types.
- A substantial amount of "standard," read-only, multiuser data.
- Significant quantities of semipermanent, read/write, single-user data.
- A minor amount of unchecked, temporary, job output data.
- A need for individual display terminal assistance in creating, modifying, and viewing data.
- A fairly long "think time" per display.
- A very long computer time per module execution request.

The machine system for which JOSHUA was designed has the following salient characteristics:

- (Conceptually) different processors for terminal and module execution.
- Attention-driven (interrupt) terminal control processor.
- Multiprogrammed, noninteractive, module execution.
- Processor to processor communication via shared direct access storage.

Such a system seems necessary to efficiently handle the assumed "think" and "compute" times postulated above. A degree of man-module interaction can be provided, when appropriate, by reducing a job to small, independent modules, and permitting terminal display and modification of results between each module. However, the addition of interactive modules, as required in graphics applications, will undoubtedly require a dedicated (multiprogrammed) attention-driven terminal processor (partition).

A large data base with many individually addressable records requires a cataloging scheme, which is conveniently referenced both by the user at a terminal, and by computational modules during execution. The user must be allowed a natural language naming scheme; the modules require a reference (record) numbering scheme for I/Ø access methods. 2

The heart of the JOSHUA system is the "named access method" which establishes, dynamically, the correspondence between record name and reference number(s).

A convenient way of representing a hierarchically organized data base is by concatenation of qualifying names,² e.g.,

MULTIGROUP.STANDARD.U238.XSECTION.FISSIØN

The collection of all such names into a general "tree" structure³ expresses quite nicely the logical order of the names (and data):

```
MULTIGROUP.STANDARD.CONTENTS
      .U238.CONTENTS
            .XSECTION.TOTAL
                  .FISSIØN
                  .ABS
                  .N2N
      .U235.CONTENTS
            :
      .HAMMER .CONTENTS
            :
```

The relationships between names in this tree reflect the most probable order of use of the data; that is, when the user or module refers to some given piece of data, he will very likely refer next to data named nearby in the tree.

Further, reference to a partial name string expresses interest in the associated subtree, and is a very convenient reference for terminal users in requesting multiple displays, and for modules to retrieve whatever data are available.

The data records themselves could be stored either in unformatted form (appropriate for FORTRAN list directed, direct access I/Ø), or formatted as appropriate for terminal display. Considering the fewer data accesses generated by terminal requests, the system is designed for unformatted storage. Displays for terminal users are formatted by maintaining a set of list descriptor records to control the conversion of each list type. Thus, computational modules, for the sake of efficiency, utilize normal I/Ø for data retrieval, and are simply provided cataloging services to replace the use of conventional record numbers.

Finally, the distinctions mentioned earlier between standard, multiuser data; semipermanent, single-user data; and temporary job results are accounted for by different procedures for allocation and management of direct access space for each data type.

2. DATA STRUCTURES

The management of data structures of the type described is not a new problem. The use of "list processors" to handle such structure has a venerable history;⁴ and more recently computer graphics systems⁵ have extended the simpler list processing techniques. 3

Consider the following simple tree:

A.B.D.
 .C.E.H
 .F.J
 .K
 .G

We may also represent this tree graphically:

Such a form is inconvenient in that any "node" (circle in the diagram) may have 0,1,2,...,n "sons" (lower level connected nodes). Since the tree must be capable of dynamic modification, keeping track of a variable number of sons will be wasteful of space (allowing for a very large number of sons) or of time (reordering a packed tree to admit or remove an additional son).

An alternative form of the general tree is the "binary tree," in which each node has only one son, and one "brother" (node to the right at the same level). In our example:

Here, a "father" points only to his first son and to his next brother. Thus the simplest computer structure of a node is:

Word	1	2
	name	
	3	4
	brother	son

in which we have allowed two full words for the name of a node, and one word each for pointers to the nodes representing next brothers and first son. The "elements" of node P will be referred to as name(P), brother(P), son(P). The "null" pointer, indicating termination of a branch, is designated $\emptyset$. Thus, e.g., son(P) = $\emptyset$ states that node P is a terminal node; i.e., that name(P) is the last qualifier of some name string.

In internal computer form, the "tree" becomes a list of chained nodes:

More complex structures have been developed to allow multiple associations between elements of the structure. For our purposes, however, the simple structure is quite sufficient, and can be handled efficiently without a generalized list processor. A useful feature of some list structures are "backward" pointers, in which each node contains pointers to the "father" node and older brother (node to the left at the same level). Such pointers allow one to traverse the tree in any direction, starting from an arbitrary node. For data cataloging, however, we expect only to traverse the tree in natural order. The only advantage of backward pointers is to avoid retracing the entire tree if a reference is similar to the prior reference. That is, a high proportion of the time, it should be possible to retrace the tree from the last node visited to a common node in the old and new name strings, and proceed forward from there. Keeping backward pointers, however, increases storage requirements by 50% (an additional two full words).

The same retracing advantages can be gained by maintaining a "trace" table of the nodes visited on each reference in an auxiliary stack - a much more economical solution.

In fact, the auxiliary stack is preferable because the complete tree structure cannot be core contained. Thus, all pointers in pages of the tree which have been overlaid would be lost without the auxiliary stack.

Another interesting feature of some list structures is a "circular list" or "ring" in which the last brother of a family points to the first brother, instead of utilizing the null pointer. This feature (which requires no additional storage) allows consecutive references to proceed without researching a set of brothers, by eliminating the right/left orientation of the simple binary tree. (One can search to the right without having to always locate the "first" son.) We have adopted this convention because of its utility.

Finally, a terminal node (one having no sons) must point to the data record number associated with the name string it terminates. This is done without use of additional storage by restricting node pointers to positive values, and utilizing a negative number as the null pointer. All pointers are relative to a base; the full word is interpreted as two half words in the form:

5

page 2

Trees are maintained as direct access data sets with 50 nodes per page (record). Up to 8 pages are in core at any time; each time a pointer is encountered with a nonzero base, a search of pages currently in-core is made. If it is necessary to bring in another page, that page which was least recently visited by a search is selected to be overlaid. If that page had any additions or deletions made to it, the copy on direct access space is rewritten before bringing in the new page. The statistics necessary for core management (counts of number of entries, number of additions, deletions, new page accesses, etc.) are available to the user, and can be used to determine the optimum use of in-core space (number and size of pages).

Consider a "complete t-ary tree" (each father has t sons, all terminal nodes are at level k). The total number of terminal nodes is $t^k \equiv m$. A search of such a tree would require, on the average, $t/2$ comparisons at each of k levels, or $\frac{k t}{2} = \frac{k m^{1/k}}{2}$ comparisons per search. Thus, a tree with $m = 32,768$ terminal nodes and $k = 5$ levels (typical of a large JOSHUA reactor physics data set) would require an average of 20 comparisons per retrieval for randomly selected names. Because of the large number of nodes, as many as 13 pages of the tree may need to be retrieved. However, retrieval of names in their logical (tree) order, should permit 25 to 50 searches without requiring a new page to be brought into core, at our present page size. Thus we estimate about 10% of direct access space is devoted to tree storage, 90% to data; about 95% of ordered searches will not require access for new tree pages, while random searches may require ~15 accesses for tree pages.

4. DATA MANIPULATION FACILITIES

Computational modules are written in FORTRAN (modified by a precompiler phase to provide read/write facilities). System modifications and improvements have been made transparent to the applications programmers (through the use of a precompiler). The operating group maintains a complete file of JOSHUA production application modules which can be recompiled as necessary.

Manipulation of data at display terminals requires formatting of the FORTRAN (binary) data records for visual display. For each type of record defined by an I/O list and name string structure, the applications programmer supplies to the system a "template" to control formatting.

- 1) Record Name: $\text{name}_1.\text{name}_2.\text{---}.\text{name}_n$ where name_1 is either a literal name (if uniquely defined for this record type), or a name embedded in "?" to signify a variable name (either computed or user supplied).
- 2) List: the I/O list used in FORTRAN read/write statements.
- 3) Variable Name/Type: for each variable in the list, a name (to be used later for tutorial purposes) and type specification (real, integer, character plus length, in bytes) is given. This relieves the restriction to full word variables.
- 4) Page Type Specification: for each implied DO loop, a page type is specified on which variables associated with that loop will be displayed.
- 5) Page Descriptions: for each page type mentioned in 4), a picture is created of the desired output format.

In the page descriptions, field specifications are given by inserting the symbol ">", followed by a string of A, I, E, F, or Z's to denote the field count (the > is not included in the count). A blank terminates the field description; decimal points are embedded in E and F strings where desired.

The page type images are analyzed for field locations corresponding to each loop initial variable; fields are analyzed for type and packed into two bytes in the form $[t, (w-1) + d*16]$ where t is a code corresponding to A, I, E, F, or Z, and w, d are the conventional integers in a FORTRAN format tw.d.

Finally all strings of more than 2 repeated characters in the page image are packed into two bytes in the form $[(n-2),C]$ where n is the count (≤ 63) of characters C in the string. This page description appears to be at least as compact as standard FORTRAN formats for templates devised so far.

7

The display package moves data from the input stream as encountered into the next available field in an in-core image of the page type specified for its corresponding loop. If page overflow occurs, the page image is forced out of core, and the field into which data from that loop were first moved becomes the next available field. When the limit is exhausted for the innermost loop on a given page which has variable loop limits, the remainder of the page is blanked and forced out of core. Thus, for example, a simple nest with two loops on the same page might behave as follows:

DISPLAY (X(I),Y(I),(Z(I,J),J=1,I),I=1,20)

More complex displays are, of course, possible. All pages of a given type are presented in the order they were created before display of the next page type occurs. This is accomplished by developing a list of one-byte page record numbers by type (each page is a direct access record) in the following form:

number of page types
count of type 1 records
record number, first type 1 record
⋮
record number, last type 1 record
count of type 2 records
etc.

The one-byte word length restricts the total number of pages to 255, which is a reasonable limit for the present application.

The overall operation of the display and modify function is illustrated below:

Formatted DISPLAY and MODIFY

The display/modify scheme using "templates" obviates the need for a large library of FORTRAN subroutines to interpret individual record types. We have in essence replaced such a library by a generalized interpretive program to replace FORTRAN I/O.

5. DATA MANAGEMENT

While the above described data handling system represents an extensive amount of design and programming, the analysis is fairly straightforward. Consideration of the multiaccess features of the JOSHUA system poses a more formidable design problem.

Considering the data base as a whole, the following concurrent activities can be foreseen:

- A significant number (≥ 8) of active terminal requests to either DISPLAY, MODIFY, COPY, or CREATE records).
- A background batch activity by computational modules reading/writing records.
- System activity in updating standard records, allocating work space, deleting work space, etc.

Furthermore, the system design calls for at least conceptually different machines for terminal and batch activity. Consideration was given to use of OS/360 ENQ-DEQ macro's to ensure the integrity of shared files. How this facility could be implemented on independent processors was not known; furthermore, its use might be system and version dependent. Finally, it was considered extremely difficult to predict the result of the use of such a scheme on system response time. Thus, it was decided to look for a conceptually different scheme for controlling multiple accesses to the shared data base.

A scheme which allows for all concurrent activities except update of standard records is the following:

- Include all data of general utility which is fairly static in "standard," read-only data sets.

- Provide each active user with an individually maintained, "user" data set. This data set is read only to all users, and can be written into only from the terminal by its owner.
- Dynamically manage "job" data set space to allocate to each batch job a temporary data set assigned to the user submitting the job. On completion of the job, this data set becomes available for interrogation at the terminal, and can be selectively copied into a user data set and/or deleted in toto by the user.
- Restrict updating of standard records to the systems group, who will accomplish this during controlled periods.

9

These file protection schemes are accomplished by the search and retrieval routines, in a way which also provides a very nice "overwrite" capability for standard record modification. All write operations from a terminal (including copies from other data sets) automatically append the user ID number as name₁ of the record name. All writes from computational modules automatically append the job name as name₁ of the record name. This guarantees the integrity of all data sets. Reads, however, are now ambiguous if the initial data set name is not specified. The system removes this ambiguity by defining a hierarchy of data set names as follows:

1. job.name.---
2. user.name.---
3. [standard].name.---

Any read operation then searches for the named record first appending the "job" name, then the "user" name, and finally the name as given. A user at a terminal can override this hierarchy by simply giving a first name (data set name) equal to the current job name (a command, "set job," establishes this at any time), his user ID, or the name "STD" for standard. It is not expected that computational modules will alter the established hierarchy.

Thus if a user (or module) writes a record with the same name as a standard record, subsequent references by that user to that name automatically retrieve his "user" record - although the standard record has not been changed for other users of the system. A "delete" of this "standard" record by the user simply restores the existence of the original (true) standard record. Thus comparative studies can easily be accomplished without special concern for the integrity of the standard data set.

The management of "job" or work space is accomplished via a combination of batch job executor and the terminal monitor which provides data and job management services. The batch executor maintains a directory of work space. Each batch job specifies the module name(s) to be executed, and an estimate of job data set size required. If at least the estimated space is available in some contiguous block of free space, the job is assigned the first such block encountered. (The largest available block is not selected because this may compound the space "fragmentation" problem, and is less efficient.⁶)

Upon termination of the job, the batch executor marks the actual extent of the space used, and updates a catalog on the processor-to-processor shared direct access space, inserting the new job name as a data set name. Thus, the existence of the job data set is unknown to the user at the terminal until

the job has completed execution. At that point, the user is free to display, copy, modify, or delete records (or the entire data set). A delete of the data set returns the work space to the free list of the batch executor (simultaneous references to the same blocks by a terminal monitor "delete", and a batch executor "allocate" cause, at worst, a lack of recognition by the batch executor of the availability of space).

10

Thus, the space management scheme prevents simultaneous read/write operations to any data set, obviates the need for system dependent code, and places the primary burden of space maintenance on the user to whom the space is assigned. A method of automatically deleting old "job" data will also be obviously necessary. The apparently awkward situation of not being free to update standard records will probably prove to be a positive feature. The hierarchy of data sets allows any individual to effectively update standard records; when he has verified the validity of the new record, the systems group can actually change the standard record (after warning all users of the impending change, so that they may copy the old standard record into their user data set if they wish, thus insulating themselves!).

The efficiency of the t-ary tree search technique was mentioned earlier, and applies directly to batch computational module execution because only one task is utilizing the search routine. However, for the case of multiple terminal references though the same search routine, traces would be lost from the auxiliary stack owing to interleaved requests from the individual terminals. To obviate this problem, three traces per terminal can be maintained in in-core work space associated with each terminal. The 8 in-core tree pages are managed as described previously to reduce total space requirements. At worst, for logically ordered searches, each terminal reference, utilizing the saved trace, will generate one tree page access to restore an overlaid page. Experience with multiple terminal activity will be required to examine the net efficiency of dynamic page pool management versus buffer and trace space allocated per terminal.

The Savannah River computing system currently has two IBM 2314 direct access storage facilities available (with a third probably to be installed in the near future), and some 50-100 potential users of the JOSHUA system. For the immediate future, therefore, it seems probable that the system can operate as though all data sets were always on line. Data set maintenance is easy - a "copy" instruction specifying only the data set name produces a clean copy (deleted data records omitted with all space recovered) so that users can easily maintain their own space with standard utilities. It seems likely, however, that at some future time, space will become a problem (if not at Savannah River, then elsewhere with less generous space allocations). Under these conditions, dynamic pop-up/push-down of data to cheaper storage (tape?) will be necessary. Batch computational module execution does not seem a problem, in that if the user data set is not on-line, a simple pre/post-job step to read/write the tape can be initiated.

For terminal operations, however, pop-up cannot occur before user sign-on, and this will entail a delay of several minutes while the tape is located and mounted. Either time scheduled terminal sessions will be required, faster devices (data cells?) will be required, or the delay must be accepted. The user can, of course, be interrogating standard files, job files, etc., during this setup period. These problems will be tackled after sufficient experience with the on-line system has been gained to permit a guess at the correct answers.

ACKNOWLEDGMENT

The JOSHUA capabilities now available have been programmed by R. L. Boyce and J. H. Maddox.

11

The information contained in this article was developed during the course of work under Contract AT(07-2)-1 with the U. S. Atomic Energy Commission.

REFERENCES

1. H. C. Honeck, J. E. Suich, J. C. Jensen, C. E. Bailey, and J. W. Stewart, "JOSHUA - A Reactor Physics Computational System," Proceedings of the Conference on The Effective Use of Computers in the Nuclear Industry, Knoxville, Tenn., April 21-23, 1969.
2. H. J. Kopp and D. S. Selengut, "DATATRAN - A DATA-HANDLING COMPUTER LANGUAGE FOR A LARGE MODULAR REACTOR DESIGN SYSTEM," Proceedings of the International Conference on The Utilization of Research Reactors and Reactor Mathematics and Computation, Mexico City, D. F., May 2-4, 1967, Vol. 3, pp. 1460-1472, Report CNM-R-2.
3. D. E. Knuth, "The Art of Computer Programming," Fundamental Algorithms, Vol. 1, Chap. 2.3. (Addison Wesley Publishing Company, Reading, Mass., 1968.)
4. Ibid. Chap. 2.6.
5. J. C. Gray, "Compound Data Structure for Computer Aided Design; A Survey," Proceedings of 22nd National Conference, p 355-365, Thompson Book Company, Washington, D. C., 1967.
6. D. E. Knuth, op. cit., p 437.

CROSS SECTIONS IN THE JOSHUA SYSTEM*

D. R. Finch

Savannah River Laboratory
E. I. du Pont de Nemours & Co.
Aiken, South Carolina 29801

A Summary of a Paper for Presentation at the

Radiation Shielding Information Center and
ORNL Reactor Division Workshop Seminar
Oak Ridge National Laboratory

October 1-3, 1969

* *The information contained in this article was developed during the course of work under Contract AT(07-2)-1 with the U. S. Atomic Energy Commission.*

CROSS SECTIONS IN THE JOSHUA SYSTEM

D. R. Finch

Savannah River Laboratory
E. I. du Pont de Nemours & Co.
Aiken, South Carolina 29801

The JOSHUA system cross section libraries provide a source of data that may be easily utilized by either casual or critical users of cross sections. The structure of the libraries and the code modules forming the linkages between the libraries are particularly suited to the rapid evaluation and comparison of new cross section data; standardized libraries provide the data necessary for reactor design and shielding purposes. The libraries are designed to be general-purpose repositories of data for use by any type of computational module. The task of converting standard data forms to special forms for individual module usage is relegated to the computational module itself. New types of data may be added to the libraries with at most trivial structural changes in the current libraries.

Figure 1 shows the JOSHUA system cross section libraries and the computational modules linking them together. The libraries shown may be loosely thought of as falling into the following hierarchy:

- ENDF/B Library - contains pointwise data in the
ENDF/B format
- FINEGRP Library - contains isotopic multigroup data in
hundreds of groups
- MULTIGRP Library - contains isotopic multigroup data in
tens of groups
- FEWGROUP Library - contains cell-composition-averaged
multigroup data with less than ten
groups

Temperature and energy spectrum dependence are accounted for in all libraries.

Access to all cross section data is provided by the data manager section of the JOSHUA Operating System. All data resides on the direct access devices and is addressed by a name string that is, in general, descriptive of the data itself. For example, the total cross section for ^{239}Pu in version 5 of the FINEGRP Library might be addressed by the name string

FINEGRP.VERSION5.PU239.XSECTION.TOTAL

The qualifier VERSION5 in the name string specifies the group structure and energy spectrum over which the cross sections are averaged.

The code module ETØJ (ENDF/B TO JOSHUA) averages pointwise ENDF/B data over an arbitrary group structure and energy spectrum to produce multigroup cross sections in the FINEGRP Library. The current module is designed primarily for neutron cross sections; however, the methods are generally applicable to photon cross sections as far as they go. The spectrum used for averaging to the FINEGRP Library may be as gross or detailed as desired, the need for detail decreasing with the number of groups. ETØJ is restricted to a maximum of 1000 energy groups, 20 temperatures, and angular dependences up to P_5 .

The code module CREØL combines a detailed Legendre spectrum with the FINEGRP cross sections to produce group average cross sections in the MULTIGRP Library. The spectra used for this reduction to fewer groups may be the result of cell calculations using the FINEGRP Library cross sections. Spectra that have been used for this reduction have generally been infinite medium B_L spectra; however, there is no restriction as to the type of spectrum. The multigroup cross sections produced by CREØL are in a form most convenient for use by most cell spectrum codes.

FIG. 1 JOSHUA SYSTEM CROSS SECTION LIBRARIES

CREØL is restricted to a maximum of 100 energy groups, 20 temperatures, and angular dependence up to P_3 .

The FEWGROUPLibrary provides composition-dependent cross sections for reactor, kinetics, and depletion codes in either microscopic or macroscopic form. The link between the FINEGRP or MULTIGRP Libraries and the FEWGROUPLibrary may be any cell spectrum calculation interfaced to the libraries.

DISTRIBUTION

Copy No.

1. I. A. Hobbs, SROO
2. P. J. Hagelston
3. J. W. Croach, AED
4. F. E. Kruesi
5. B. C. Kline, SRP
6. R. G. Gambill, Jr.
7. R. G. Croom
8. W. B. Daspit
9. A. H. Peters
10. E. O. Tower
11. E. O. Kiger
12. D. A. Ward - C. J. Temple
13. J. E. Suich, SRL
14. H. C. Honeck
15. P. S. Tennant
16. L. M. Arnett - J. J. Bugg
17. R. L. Carson - W. R. Hartshorn
18. J. C. Jensen - R. L. Boyce
19. R. S. Wood - R. Wehler
20. C. E. Bailey - F. D. Knight
21. D. R. Finch - W. V. Baxter
22. W. H. Hightower - C. E. Henderson
23. J. H. Maddox - R. Wilcox
24. J. E. Bearden - R. C. Haywood
25. J. W. Stewart
26. B. P. Compton
27. W. E. Graves
28. D. A. Sharp
29. F. J. McCrosson
30. P. L. Ames
31. D. E. Hostetler
32. F. E. Warren
33. R. M. Wallace
34. J. A. Porter - M. L. Hyder
35. F. R. Field
36. D. H. Knoebel
37. I. M. Macafee
38. R. C. Thornberry
39. J. N. Wilson
40. J. Lewellyn (AEC)
41. B. Toppel (ANL)
42. Richard Ehrlich (KAPL)
43. D. Vondy (ORNL)
44. C. W. Craven, Jr. (ORNL)
45. Robert A. Dannels (WAPD)
46. R. L. Smith (SROO)
47. P. Michael (BNL)
48. D. Roy (B&W)
49. P. Aline (GE)
50. C. W. Lindenmeier (BMI)
51. G. H. Best (LASL)
52. H. Alter (AI)
53. B. Hutchins (GE)
- 54-99. TIS File
100. Vital Records File