JAERI 1274

JAERI 1274

Modular Programming Method at JAERI

February 1982

日 本 原 子 力 研 究 所

Japan Atomic Energy Research Institute

JAERIレポートは，日本原子力研究所が研究成果編集委員会の審査を経て不定期に公刊している研究報告書です。

入手の問合わせは，日本原子力研究所技術情報部情報資料課（〒319-11茨城県那珂郡東海村）あて，お申しこしください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11 茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

JAERI reports are reviewed by the Board of Editors and issued irregularly.

Inquiries about availability of the reports should be addressed to Information Section, Division of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

編集兼発行　日本原子力研究所
印　　刷　いばらき印刷㈱

JAERI 1274 Errata

Page	Line	Printed	To be corrected
ii	footnote	内藤俶孝, 浅井　清	内藤俶孝, 茅野政道, 浅井　清
3	7	1869	1969
4	Fig. 2.1	CPU time in year	CPU time in year(hours)
"	12	experiences	experience
9	27	couping	coping
11	Fig. 2.4	EXECUTER	EXECUTOR
"	23	usuable	usable
19	35	notrious	notorious
20	Fig. 3.4	M200　2	M200 x 2
22	8	quantative	quantitative
23	Table 3.3, line 17	a alternative	an alternative
24	Table 3.3, line 4	logcial	logical
"	" line 11	implementor for	implementor of
"	" line 27	execution of	execution,

Modular Programming Method at JAERI

Edited by
Kiyoshi Asai and Satoru Katsuragi†

Computing Center, Tokai Research Establishment
Japan Atomic Energy Research Institute
Tokai-mura, Naka-gun, Ibaraki-ken

Received September 3, 1981

Abstract

In this report the histories, concepts and a method for the construction and maintenance of nuclear code systems of Japan Atomic Energy Research Institute (JAERI) are presented. The method is mainly consisted of novel computer features. The development process of the features and experiences with them which required many man-months and efforts of scientists and engineers of JAERI and a computer manufacturer are also described.

One of the features is a file handling program named datapool. The program is being used in code systems which are under development at JAERI.

The others are computer features such as dynamic linking, reentrant coding of Fortran programs, interactive programming facility, document editor, quick system output viewer and editor, flexible man-machine interactive Fortran executor, and selective use of time-sharing or batch oriented computer in an interactive programming environment.

In 1980 JAERI has replaced its two old computer systems by three FACOM M-200 computer systems and they have such features as mentioned above.

Since 1981 most code systems, or even big single codes can be changed to modular code systems even if the developers or users of the systems will not recognize the fact that they are using modular code systems.

The purpose of this report is to describe our methodology of modular programming from aspects of computer features and some of their applications to nuclear codes to get sympathetic understanding of it from persons of organizations who are concerned with the effective use of computers, especially, in nuclear research fields.

Keywords: Modular, Code System, Computer, Performance, Nuclear Code, Multics, Methodology.

† Chairman, Modular Programming Subcommittee of Nuclear Code Committee, JAERI
(members: Takanori Shimooke, Atsuo Kohsaka, Taketoshi Arai, Tatsuoki Takeda, Toshihide Tsunematsu, Kinji Koyama, Nobuaki Ohnishi, Yoshitaka Naito, Masamichi Chino, and Kiyoshi Asai)

原研におけるモジュラ・プログラミングの方法

日本原子力研究所東海研究所計算センター

浅井　清・桂木　学†（共編）

1981年9月3日受理

要　　旨

本報告は日本原子力研究所におけるコード・システム構築，維持の歴史，考え方，および方法について述べた．その方法は主として計算機の機能から成っている．それら機能の開発には原研の研究者，技術者，および計算機メーカの技術者の多大の人手を要したが，その開発過程についても述べている．それらの機能のひとつはデータプールと呼ばれるファイル取扱いのプログラムで，これは現在，原研で開発中のコード・システムで使用されている．その他は，プログラムの動的結合，主記憶内プログラム共用可能，会話型プログラミング，記録清書，システム出力表示，実行速度制御可能 Fortran，タイムシェアリングあるいはバッチ処理用計算機の選択的使用などの計算機機能である．最近，原研に設置された計算機はこれら諸機能を装備しているので，原子力コードの開発者や利用者が特に意識しなくても，既存の大型コードをモジュラ・コード・システムに変更することができる．

本報告は，原研におけるモジュラ・プログラミングの方法論と適用例を記述し，それによって，原研で行った方法について，原子力分野における計算機の有効利用に関心のある人々の理解を得ることを目的としている．

---

† 日本原子力研究所原子力コード研究委員会，総合化専門部会長，
（専門委員；下桶敬則，鴻坂厚夫，荒井長利，竹田辰興，常松俊秀，小山謹二，大西信秋，内藤俶孝，浅井　清）

Contents

目　　次

1. Introduction

In this report we will give a brief description on the histories, concepts and a method for the construction and maintenance of nuclear code systems of Japan Atomic Energy Research Institute (JAERI). The development process of the method and experiences with it in the actual applications which required many man-months and efforts of scientists and engineers of JAERI and a computer manufacturer are also described.

In the decade of 1960s, the matured research fields in reactor physics and increasing demand for design calculations on nuclear reactors induced scientists and engineers to an attempt to automate the calculations using computers.

Around midst of the decade the attempt began to have a shape in code systems. A computation system is called a code system when it has a unified, orderly set of computer programs and data libraries. If the code system is designed so as to be able to change easily its component, i.e., a program or a library, the system and the component are called a modular code system and a module, respectively.

The first of the code system in the nuclear research and application fields would be the NOVA code system of Knolls Atomic Power Laboratory (KAPL). This system was implemented on a second generation computer and was used for the design of naval reactors[1]. Later adopting operational experiences, the system seemed to have grown up as a general purpose framework DATATRAN for modular code systems[2].

In the subsequent years similar code systems were developed at Argonne National Laboratory (ANL, 1967) and Savannah River Laboratory (SRL, 1968) using the third generation computer.

The code system made at ANL is called ARC (Argonne Reactor Computation system)[3),4] and it allows to transfer data between modules using files and/or main storage. The data must be in prefixed forms and, because of this constraint, every existing (sub) program will be forced to be rewritten before it is used as a module in the system.

The code system of SRL is called JOSHUA[5),6] and its all modules except a few were made anew so that the system reflected the state-of-the-art of physical and computational methods of the time. Data transfer between modules are done by using files. The input/output operations from/to files are executed by pseudo-Fortran statements unique to the system. It was able to unify the input/output operations, because almost all modules were made anew when the system was constructed. This standardized data format made it easy to provide the system with general purpose utilities for data such as editor, graphics, etc. and as the results the system became very useful for its users. The standardization of data format, however, seems to become an obstacle for casual users of the system, because they must change their computer programs to obey the standardization.

In Japan, during the same period, a similar attempt was done at Hitachi Works, Ltd. using its second generation computer HITAC5020[7].

At JAERI, in 1971, we organized a subcommittee for modular code system in Japan Nuclear Code Committee to develop a modular code system for analysis and design calculation of nuclear reactors. The Japan Nuclear Code Committee is a council committee to the Director of Tokai Establishment, JAERI. Using an IBM370 computer, the subcommittee had designed a framework of modular code system and developed several routines to control modules.

During the same period some code systems in nuclear research fields were planned and

constructed at JAERI.

Introductory papers of these systems were submitted to the conference in modular code systems held on December 1970 at the Computer Program Library, Ispra, Italy[8]. Brief descriptions of these code systems will be given later in Chapter 2, Section 2.2.

After construction a prototype code system, the subcommittee resolved itself in 1975 and established a new subcommittee for unification of nuclear codes. The purpose of the new one was to set up practical methods for code unification by 1980. Noticing a fact that it takes many man-months to construct and maintain a code system, the subcommittee had decided to pursue computer features which would replace the man-months. Activities and results of this subcommittee will be sketched in Chapter 3, Section 3.3.

As the results, we have obtained novel computer features which have provided us with a present method for modular code systems. In Chapter 3, Section 3.4, a comparison between our present method and the conventional ones is given. Our present method has been applied to several code systems of JAERI. Since recently our method became available, several scientists of JAERI were requested by the subcommittee to use it for constructions of their code systems. In Chapter 4, two of them are described briefly to show how the method has been applied. The Section 4.1 and 4.2 are written by K. Tuchihashi and K. Muramatsu of JAERI, respectively. Both of them applied our method to their code systems accepting the subcommittee's request. In Chapter 5, the concluding remarks are given with some findings obtained by the use of the method. In the Appendix a brief description of the structure and characteristics of our datapool is given. This is due to the reason that a datapool constitutes a basis of any modular code system and the modularity of the code system depends on the structure and characteristics of the datapool which is being used by the code system. This datapool was designed by K. Asai, M. Tomiyama of JAERI, M. Yoshimori of IBM, Japan Ltd., Y. Takigawa of Fujitsu, Ltd. et al. After discussions in meetings of the subcommittee, it was approved by the members of the subcommittee and constructed by M. Tomiyama et al.

The other parts of the report are written and compiled by K. Asai and S. Katsuragi who are the member and the chairman of the subcommittee, respectively.

2. A Short History of JAERI's Approach to Code Systems

2.1 Environments of Computing Facilities

(1) The period from 1956 to 1963 may be called a dawn of computational era at JAERI. During this period we at first had an IBM602A relay type calculator and then an IBM650 electric computer of a drum storage with 2000 words capacity.

During the next period from 1964 to 1869, JAERI introduced an IBM7044 electronic computer with 32768 words core storage. This computer might be called as a member of the class of second generation computer. The IBM7044 had no multiprogramming feature. In the period our computer was used with one shift operation mainly because of budgetary reason. Although the number of jobs increased every year and at the end of the period we were forced to wait one week even for a job which required only an elapsed time of three minutes. Only magnetic tapes and cards were used for computer files. The idea of the modular code system did not come to our mind at all.

In 1970 the IBM7044 was replaced by a Japan-made computer FACOM230-60, which was called as F60 shortly. The F60, having one central processing unit, 128K (1K = 1024) words core storage, was used with two shift operation. In the first shift operation in day time the F60 was operated by operators and in the second shift it was operated by volunteer users who wanted to use the computer for their own jobs.

The F60 had the multiprogramming feature and the operating system was similar to that of the IBM360 computer. The F60 with 8 base registers, 256K directly addressable words, a third generation operating system Monitor V, might be called the second and a half generation computer.

At JAERI, in this era, most computer programs (codes) were developed and used by individual researchers and engineers. Every individual made codes for his own use. Even the codes introduced from foreign countries were used in this way.

(2) In 1971 JAERI's F60 computer system was enhanced to two systems of F60's. Each of the systems equipped with two central processing units and 256K words storage. At the same time numbers and amount of disk units increased. The operation of the computer systems had been carried out by the computer manufacturer based on a contract, at first with two shifts, and a few years later, with three shifts.

The increase of computing power, storage capacity and amount of disk file space have encouraged the scientists and engineers of JAERI with the possibility of construction of modular code system. The computing power of every computer systems installed at JAERI is shown in **Fig. 2.1**.

The experienced persons of JAERI were naturally stimulated by various domestic and foreign projects of the construction of modular code systems. Reflecting these circumstances, modular code systems at JAERI met the stage of materialization in this period.

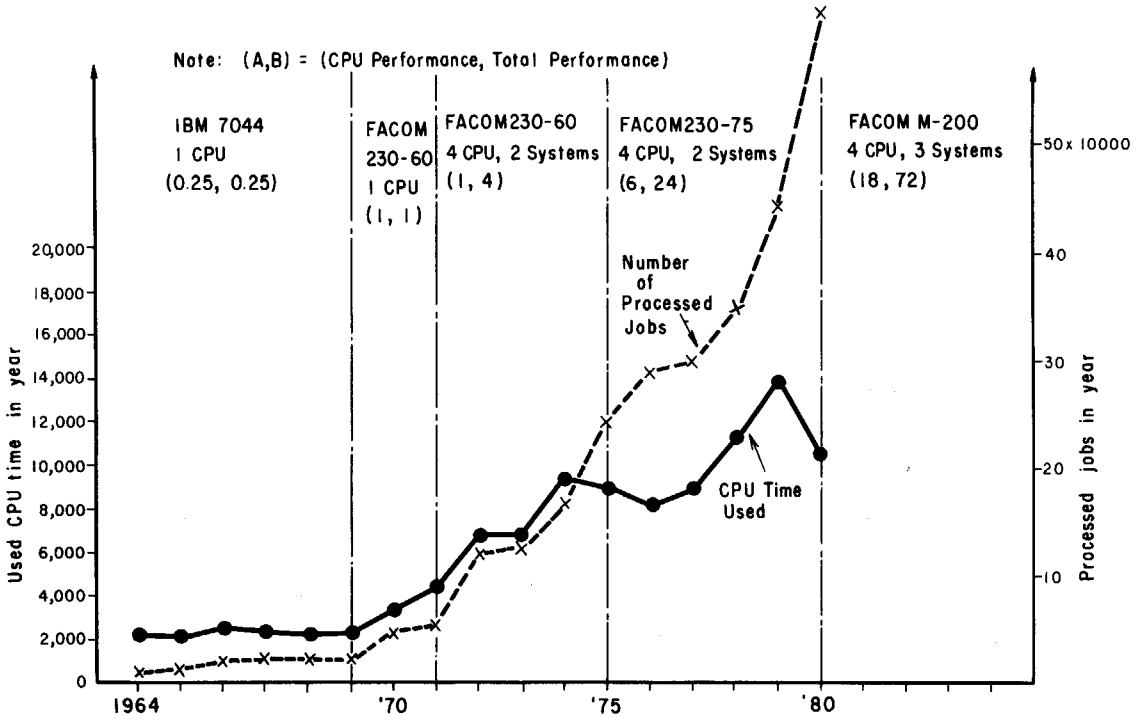


Fig. 2.1 Transition of Computing Power at JAERI

2.2 JAERI Code Systems: Old Ones – First Trials

In the beginning of the last decade several code systems had appeared in the fields of nuclear research and engineering[1)-8)]. Among them there were famous code systems such as JOSHUA, ARC, etc. Trials and successes of the code systems accomplished by preceding organizations were great spurs to us.

As mentioned in Chapter 1, in 1971 the Japan Nuclear Code Committee decided to establish a new subcommittee to develop a methodology to promote systematic constructions of modular code systems.

Around 1971 three code systems had been developed at JAERI. The names of these code systems were DOYC, JCOMPACT, and JFRIC. The code systems were all different in methods adopted for realization of their purposes. In the following section we will give brief descriptions of these code systems because the experiences with the systems has led us to our current attitude toward code systems.

2.2.1 DOYC Code System

DOYC was a code system designed and made by H. Kuroi, K. Koyama et al. for analysis of fast reactor physics. The word DOYC is an abbreviation of a phrase "Do it On Your Choice"[9)]. It was at first implemented on the FACOM230-60 computer. It uses the conventional overlay technique for amalgamation of modules. So that most modules of this system are subroutine-type subprograms. Therefore the construction of the system does not require any additional extension of the computer feature. It is a modular code system in the sense that it has a datapool common to modules and that its modules are replaceable by other modules of similar functions. The datapool of DOYC is dependent on a direct access technique. Data are stored and retrieved by a non-hierarchical name. By referencing the inner table the datapool routine converts the name to an address pointer of records of a disk

file. It is a very time consuming and error-inducing task to add or delete a subroutine or a program to or from a complex overlay code system which is implemented on a second generation computer. To elude this difficulty, DOYC has many built-in dummy subroutines and when a new subroutine is required, its name is selected from one of the pre-determined dummy subroutine names. Thus the user has no need to reconstruct the overlay structure of the system.

Most modules are stored in a disk or tape in relocatable binary form and linked together by the linkage editor before execution. When both module names on the system input file and the user's module library file are the same, the module on the system input file is used. Thus user is able to make a slight modification of the system by using his own subprograms.

The DOYC code system is especially used for analysis of experimental results of the Fast Critical Assembly installed at JAERI.

The designers and users of DOYC have pointed out following merits and demerits on the system:

1) Ease of Use

Input/output data handling and manipulation of job control language which had been normally necessary for linking independent programs were removed and the description of input data is made simple.

2) Saving and/or Waste of Computation Time

Using the DOYC system waste time of computation is saved, since the simplified input specification reduced most of errors due to incorrect specifications. However, it sometimes occurred for DOYC users to carry out meaningless and erroneous calculations, because the system has no functions to accept an interrupt by users and display the input informations before the calculations.

2.2.2 JCOMPACT Code System

During the period from 1971 to 1972 the code system JCOMPACT was developed by a JAERI scientist T. Nishida and an engineer M. Tomiyama, assisted by system engineers of a computer manufacturer[10]. It was a modular code system in the sense that it was equipped with a datapool common to all modules, a path driver, a checker which had ability to select and drive modules in a module library according to user's specification, and an interruption handler in the execution time of the system.

The designer of this code system had placed much emphasis on the modular use of each module included in the system. As the result, most efforts were devoted for construction of the datapool, path checker, and rewriting of input/output statements of each module. A module was essentially a subprogram or a set or subprograms stored in relocatable binary form. Users of this code system was able to specify a path and modules in a form

$$\text{PATH} = (P_1\ (N_{11}, N_{12}, N_{13}), \ldots\ldots, P_m\ (N_{m1}, N_{m2}, N_{m3})),$$

where P_i was a module name, N_{i1}, N_{i2} and N_{i3} were option numbers for card input, datapool input and datapool output of module i, respectively.

The path checker checked whether the specified path was already registered in the code system. If the path was already registered the control was transferred to a module representing the path and the Fortran main program in executable binary form began to execute other modules along with the path. If the checker found that the path was not in the registered path table, it checked the logical validity of the specification and when it was correct, the control was transferred to a job step of library editor. The library editor created a Fortran source program in which the specified module names appeared in Fortran CALL statements.

In the subsequent two job steps, the created Fortran main program was compiled and

link-edited. If no error was found by this time, the system proceeded to the next job step and in that job step, the new module containing the specified path began to execute the other modules.

Data in the datapool were identified by a triplet (i, j, k) and accessed by the system's data handler using the triplet as an identifier. A file allocated on a disk was used for the datapool. The data handler stored and retrieved the data in the datapool by a random access technique. The identifier served as a key to the data.

This system used three types of data structure for transferring data between a module and the datapool.

The type-1 data were fundamental ones for the system and stored in COMMON areas of a module when the module was loaded in the main storage. The calling sequence to read the data into COMMON areas was automatically generated and executed by the system. It was a rather difficult task to re-adjust the type-1 data of the system in case of addition or deletion of modules.

The type-2 data were input, output and/or temporary work data of a module and the module was responsible for storing and retrieving the data into or from the datapool.

The type-3 data were same as the type-2's in their use except that they were accessed in blocks. For a second generation computer, it was not able to store in main storage a large amount of data corresponding to a physical variable, say sigma (i, j, k), where i, j, or k's value ranged from unity to one hundred. In this case a technique is often used to divide the area for data into two parts, one is in main storage as an working space, and the other is in the file as the entire data space. The type-3 data was provided for this purpose. The space for sigma (100, 100, 100) was reserved in the datapool and if sigma (100, 50, 10) was declared in a module, the space for sigma in the datapool was segmented in 1 x 2 x 10 blocks. One block was the unit to read or write the data.

One of the most desirable features of a code system will be an interruption handler. By this function the user of the code system can display intermediate results or can change a part of the path. The code system JCOMPACT realized this function by combining the code system routines such as data handler, path checker, etc. with computer system's monitor commands WTO (write to operator) and WTOR (write to operator with reply). WTO command in a module writes a specified message to an operator console. WTOR command does the same function as WTO and accepts an input from the operator. Thus the WTOR command can display a part of computational results and accept new input from the operator console. The reader will probably recognize at once that the commands and an operator console are not a sufficient tool for input/output operation for the user of code system.

This was also true to the JCOMPACT, but the reader should note that the code system was constructed tentatively to seek for every possibility of modular code system. Even at this moment we have no easy way to implement a flexible interruption handler. We will see later in Chapter 3 a computer software which will serve as an interruption handler better than WTOR.

This system was built tentatively to see the problems on construction of code systems. It is not used now but it offered useful findings of difficulties and necessary computer features on construction of code systems.

2.2.3 JFRIC Code System

JFRIC code system was designed and made by S. Katsuragi and an engineer of IBM Japan M. Yoshimori to integrate many stand-alone codes and data for fast reactor physics. Its ultimate purpose was to construct a fast reactor integrated computation system to serve

as a substitute for something like a handbook of reactor physics[11]. In other words, if JFRIC system could be successfully developed, it would be composed of files of various kinds of data and libraries of computation schemes executing calculations with these data. Any user of this system would use and combine any codes and data to perform a series of calculations meeting his requirements.

The designer of the system observed that the essential drawback of existing codes, to be applied for such a flexible system, was an extreme inter-relationship between data and computation schemes or a complete dependence of data on computation schemes. If data were always stored into standard data storage instead of arbitrary, selfish storaging, the independence of data from computation schemes could be kept and the construction of the integrated computation system would be easy. The complex inter-relationship between data and computation schemes also seemed as a major obstacle for linking a computer code to any code system or for replacement of any code with other code.

By this observation an effort was put on the development of a methodology to separate data from computation schemes and to control data by a general common program. To combine a proper set of data with any codes, JFRIC uses a certain kinds of modular binding among them. The scheme of JFRIC code system is shown as **Fig. 2.2**.

The system is consisted of three major functional modules, i.e., a path driver, control modules and calculation modules.

The path driver plays a main role of the system. It determines the path and combines control modules, which calls a calculation module to execute a part of the whole physical problem, and performs data management. That is, it defines the size of required main storage, allocates the main storage, retrieves and sets scalar and array variables for the calculation module into allocated storage prior to the calculation, and files specific data into storage devices attaching appropriate indices after calculation. Termination of whole calculation or restart of calculation is also controled by the path driver. The control variables to data management in the path driver are supplied by control modules.

Usually one control module is provided for one calculation module and it is actually

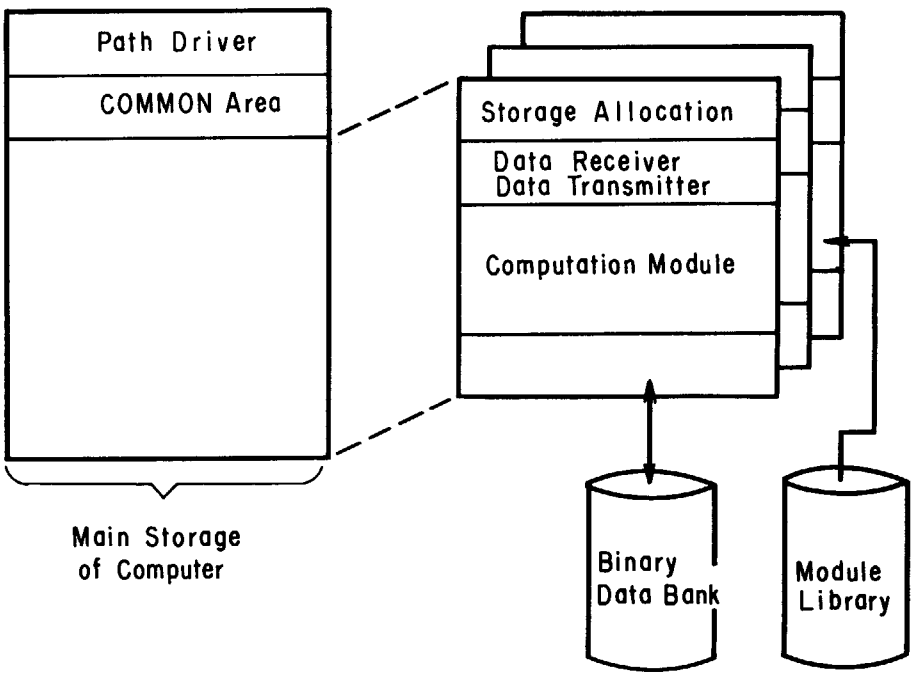


Fig. 2.2 Structure of JFRIC Code System

an interface between the path driver as data handler and the calculation module performing only physics calculations, on the one hand, and it performs the function to link the calculation module in relocatable form on the other hand; that is, it defines absolute main storage location of variables and entry points of various subroutines in the path driver, which are called by the calculation module to be executed. The control module loads the calculation module and executes calculation.

The calculation modules are modified from usual program so that all I/O statements except those to external output devices are removed, and almost all declaration statements are also removed. All the sequential or BCD input data necessary to the whole calculation are collected altogether and read by path driver just after the start. The input data are filed according to appropriate indices so as to be used properly by the calculation modules. The data handling mentioned above is indispensable to assure flexibility for selecting any calculation of data stored into the main storage.

The name of calculation module included in the system is listed in the two dimensional table. The calculation path or the order of calculation module is specified by an input of two dimensional table, an element of which is processed by the path driver to find the next calculation module.

The fundamental structure was tested by sample paths and sample calculation modules and proved workable. The framework can be used in the IBM360 or 370 type computers.

As for the merits and demerits of the framework, the designer has pointed out the followings.

1) Flexible Path Formation

The user can define, in advance, the path along with his requirement unless it is prohibited by physics. The next calculation module can be selected according to the result of calculation.

2) Saving of Main Storage

The saving of main storage becomes possible by linking of calculation modules at the execution time and by the separate handling of all the data from calculation modules.

3) Difficulty of Separating Variables from Calculation Scheme

Since all the data should be handled by the path driver separately from the calculation schemes, it is necessary to convert usual codes into modules and data acceptable to the system. Every code is designed based on a different design philosophy and a different data treatment from each other. Hence, converting data set in each code into a form acceptable to the system seems to be a very troublesome task. If data handled by the path driver is restricted to input

Table 2.1 Our Efforts for Old Code Systems

Code System	Application Area	Participated Staffs	Invested Man-Month	
			Development	Revision & Maintenance
DOYC	Experimental analysis for Fast Critical Assembly	Scientists 3 Programmers 4	90	144
JCOMPACT	Framework for code system and diffusion, burnup calculation for light water reactor	Scientist 1 Programmers 3	36	–
JFRIC	Framework for code system for fast breeder reactor calculation	Scientist 1 Programmer 1	15	–

data, library data, and data of large amount, this difficulty will be reduced to a large extent. The improvement of the system framework, however, was not performed, because the then computer FACOM230-75 was not suited to such trial.

Our efforts for these code systems in terms of invested manpower are summarized in **Table 2.1**.

2.2.4 Interactive Fortran Processor

In this section several sentences which explain a finding about an experimental timesharing system and the necessity of computer utility are cited from the excellent review paper by F.J. Corbato and V.A. Vyssotsky[12]. The cited sentences are enclosed by quotation marks. Our approach to TSS-Fortran would be best explained by the philosophy of the famous Multics project. Multics (Multiplexed Information and Computing Service) is a comprehensive, general purpose programming system which has been developed by Massachusetts Institute of Technology, General Electric Company and Bell Telephone Laboratories, Inc. as a joint research project. The design philosophy of "Multics was to create a computing system which was capable of meeting almost all of the present and near-future requirements of a large computer utility." At the time when the system was designed, around 1965, most computers were used only for batch processing and the timesharing use of them was in a stage of development. The most eminent characteristic of the Multics system was to allow many users flexible man-machine interactions. For the purpose the system was provided with many novel features in its hardware and software. It is, however, not our aim to go into details about the system and later in Chapter 3 we will describe very briefly on some of features which were relevant to our TSS-Fortran.

For JAERI people the interactive use of computer means the timesharing use of large scale computers. It was said that "the impetus for timesharing first arose from professional programmers because of their frustration in debugging programs at batch processing installations." During the period of 1966–1969, this was very true for scientists, engineers and professional programmers of JAERI because they had to wait, couping with each other for limited computer resources, one week to get their jobs of three minutes' elapsed time processed by the computer.

People joined in the Project MAC[13], a foregoing project of the Multics, found a new evolution in the interactive use of computer.

"The Multics system was expected to run continuously 7 days a week, 24 hours a day in a way similar to telephone or power systems." This requirement came from over two year experience of a preceding experimental timesharing system named Project MAC. "The experience suggested that continuous operation in a utility-like manner, such as a telephone or power system with flexible remote access, encourages users to view a system as a thinking tool in their daily intellectual work." This was a very new finding in that time.

If a timesharing system leads users to view the system as a thinking tool, what type of man-machine interaction is most suitable for the purpose? The question is the starting point of our TSS-Fortran processor. To answer the question we firstly characterized actions of human being and computer, secondly devised desirable functions of the processor to complement the human characteristics with corresponding software names for the functions, and thirdly listed necessary computer features to support the functions as shown in **Fig. 2.3**.

In a word, we thought that a computer system should be subject to motions of human beings and a language processor should be developed to support this idea if the system would play a role of thinking tool.

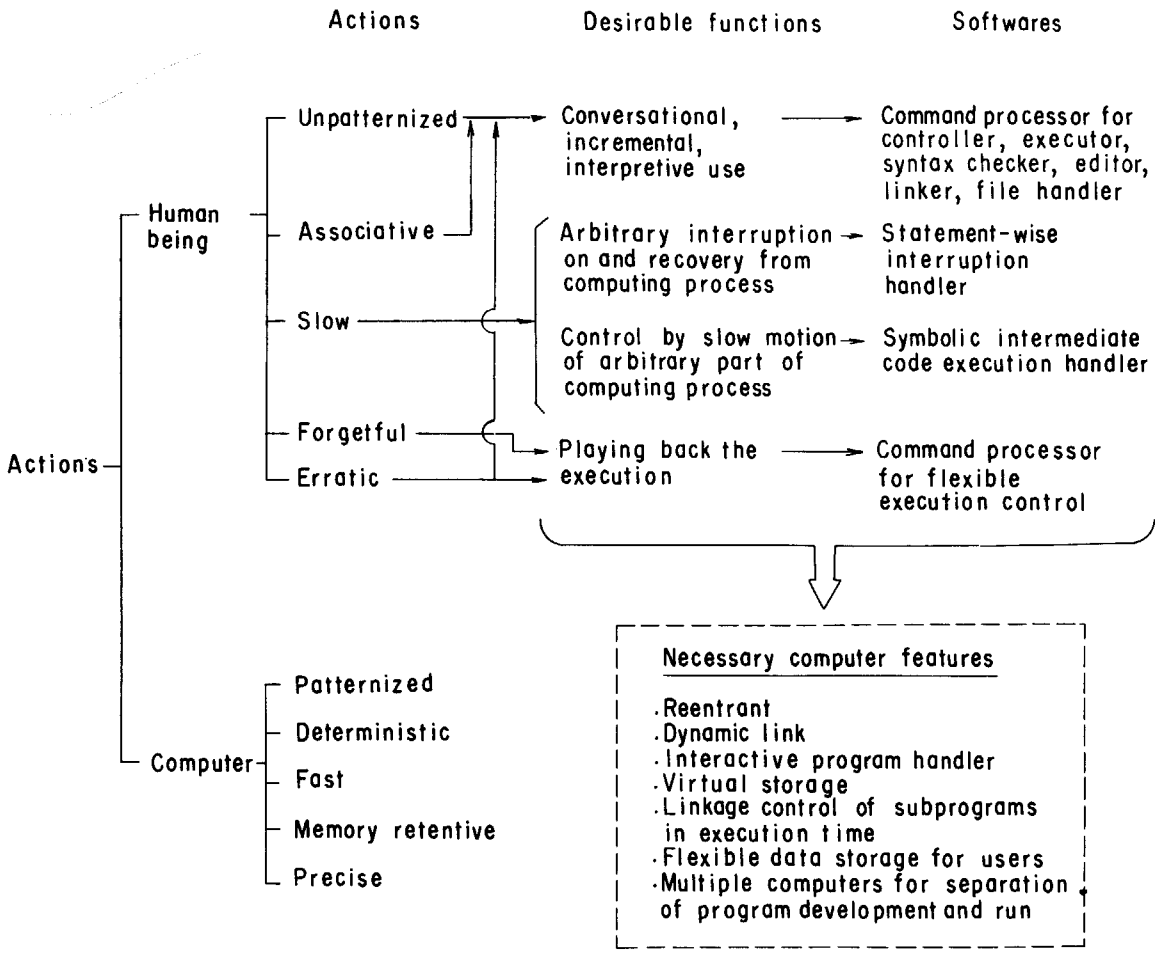


Fig. 2.3 Human Action, Corresponding Computer Use, Softwares, Computer Features

Since most computer programs in the nuclear research fields are written in Fortran, we decided to develop a timesharing Fortran, plus a command processor which included all of the functions described in **Fig. 2.3**.

The decision was not absurd because in 1972 Japan-made computers which enabled to realize our TSS-Fortran had appeared in the market. They, HITAC 8700 and 8800, had versatile features such as tightly coupled multiple central processing units up to four in total, virtual storage, optional machine interruption for subroutine calling, ring level protection mechanism in hardware, and multi-virtual storage, reentrant programming, dynamic linking of programs, subprograms including Fortran COMMON areas, one job control language for batch and timesharing processing, tree-structured, hierarchical naming of files in software. It was evident that these computer systems were under the influence of the Multics concepts.

We anticipated that this type of computers would prevail in the future.

Therefore we determined to construct our interactive Fortran, plus a command processor on the basis of these new features, although it was not clear at the time that we could procure computers with such features.

We thought that a large scale timesharing system should be used not as a game playing toy or a text editing machine, but as a one from which the user could obtain full computing resources as if he was the only user of the system. This idea was a very new one and it took a long time to be accepted in our environment.

In 1972, we fixed the specifications of our processor and began to construct one of its parts. The skelton of the processor was roughly shown in **Fig. 2.4**.

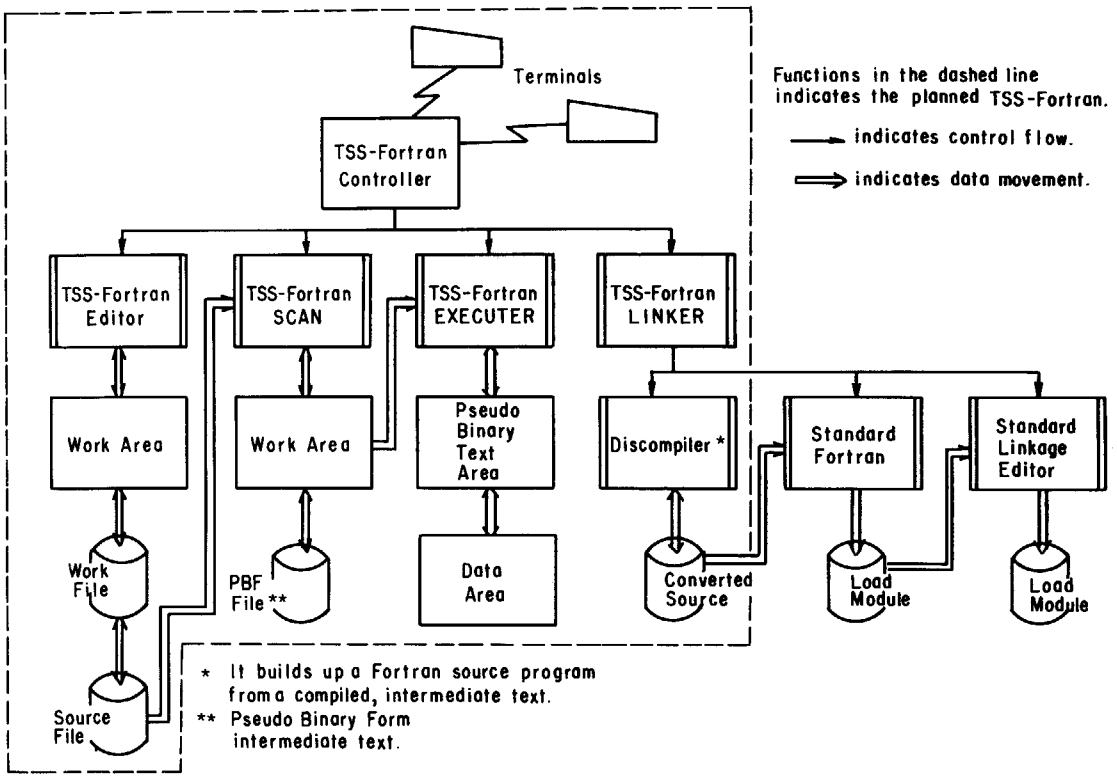


Fig. 2.4 Skeleton of Our TSS-Fortran

For better understanding of our processor, the reader is suggested to imagine the video recording and display system in mind. The expected functions of our processor were forward and backward search of point where the user wished to see the system status, slow and fast execution speeds, address stops and statement by statement execution, splitting the display screen into slow and fast parts, or current, past or future status parts, arbitrary interruption of current status for editing, compiling of source programs or replacing linked modules.

The reason why the computer features listed in **Fig. 2.3** are needed to realize these functions requires rather lengthy technical explanations and is not given here because it is not the main theme of this report. However we will later, in Chapter 3, present our new method for modular programming and where the reader can infer the necessity of the features. Two software engineers of JAERI Computing Center engaged in the development, one was as a full time and another a part time worker.

In 1975, two FACOM230-60 computer systems were replaced by two FACOM230-75's. The HITAC8700/8800 were powerful candidates, but we could not afford them because of their high prices. In this period the manufacturer of FACOM230-75 computer had decided to equip it with reentrant and dynamic link features for its newly coming third generation computer. By the decision there remained a possibility that a user of the computer manufacturer can construct such interactive processor as described here on the future machine. After the replacement we continued the development of our TSS-Fortran intermittently for one year, but as the FACOM230-75 computer had none of features described above, we felt it very difficult to realize the functions initially specified for our TSS-Fortran. Thus we abandoned to continue our effort to develop the processor. By this time a component of the processor was developed and became usuable. It was a Fortran syntax checker named SCAN and was written in a system writing language GPL[14]. Later this syntax checker was rewritten in Fortran and applied to some other needs[15]. A preprocessor for datapool soft-

ware described in the appendix of this report and in a report[16] are examples of the applications.

After five years since then we replaced the two FACOM230-75 systems by three loosely coupled FACOM M-200 computer systems.

Using our concepts the computer manufacturer has developed a new conversational Fortran, plus command processor for the M-200 computer[17]. Its external specifications closely resemble with our abandoned one's. It is able to operate on a display terminal with high, medium, slow, or statement by statement execution speed, splitting the display screen for statements, data and commands.

The computer manufacturer has also presented independently of our idea of TSS-Fortran a software for subsystem control which allows the user to transfer control from a Fortran program activated in a timesharing environment to editor or other subsystems and then returning control to the Fortran program[18].

(Kiyoshi ASAI, Computing Center and Satoru KATSURAGI, Div. of Reactor Safety)

3. Our Present Method for Code Systems

3.1 Background to Support Code Systems

The concept of modular code system at JAERI was and has been different from those of other research organizations in the sense that it has been deemed as an integrated unification of codes and data libraries on physical problems, computer softwares and hardwares to help users who want to use them. The backgrounds which support a big code system will be listed in items as in **Fig. 3.1**.

At first members of the Modular Code Subcommittee were unconscious of the fact that the every item referred to in **Fig. 3.1**, especially of a fact that the "Extensive effort for development and maintenance", is essential to construct and maintain a comprehensive modular code system. Or it might be said that we neglected consciously the items on con-

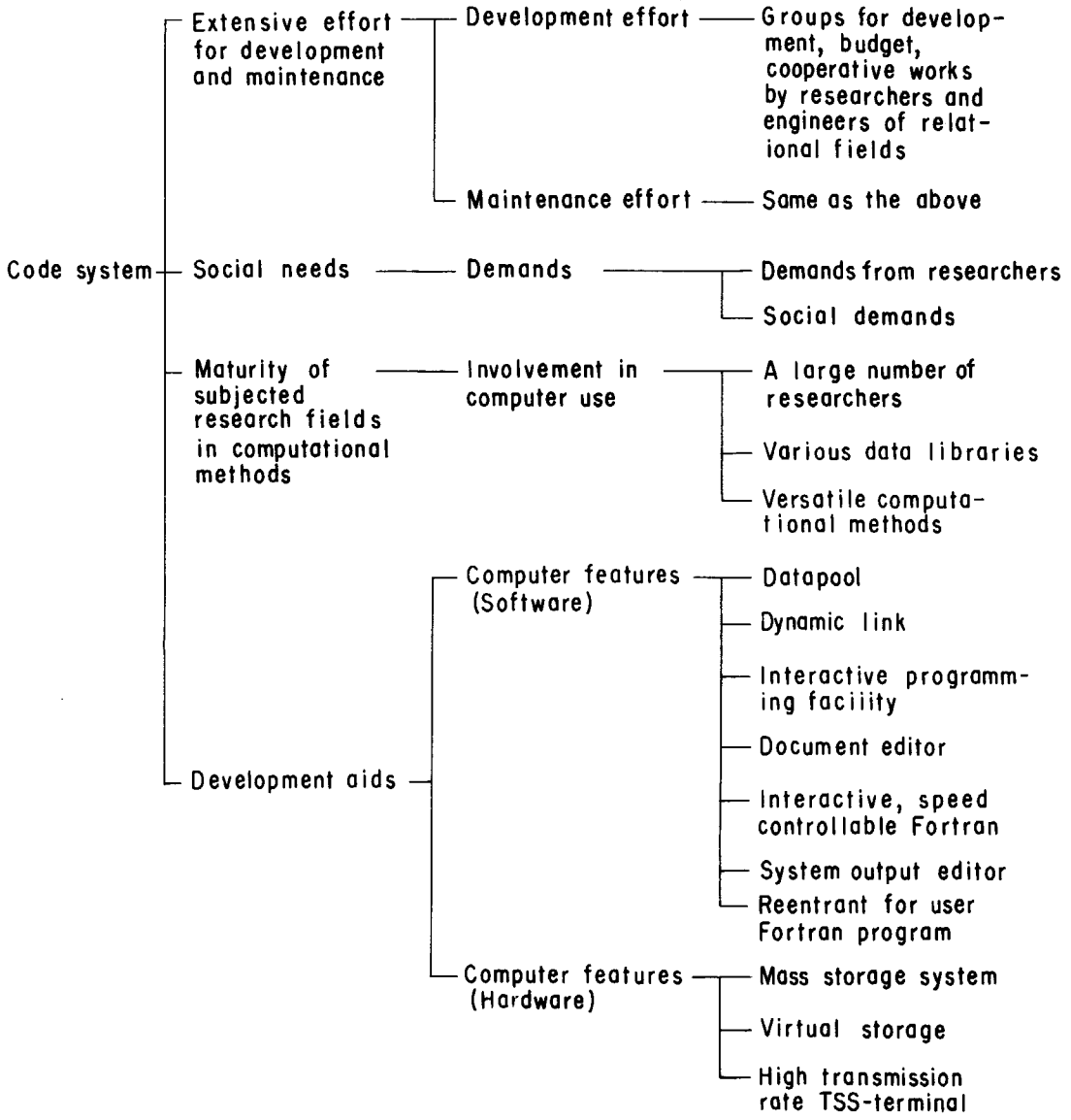


Fig. 3.1 Background for Code Systems

struction and maintenance of a code system by reasons that the items should be taken into account after the construction of a modular code system, or it was too early to consider all aspects of modular code system. This attitude should not be blamable in that time because we were, even a computer manufacturer biggest in the world, optimistic in our productivity on computer programs[20]. As time went on, however, it gradually became clear for us that the existence and survival of a code system depend on the realization of every item listed in **Fig. 3.1**. This is a conclusion that we have reached by looking on successes and failures of preceding several projects for modular code systems in the world including Japan. This is the reason why modular code systems in organizations of fundamental researches cannot survive continually. Our JAERI is not an exception. Thus we must investigate in more detail the works normally needed to construct and maintain a code system to replace the works by computer features. We think the computer features and software tools described in Section 2.2.4 are the solutions to this problem and in this chapter we will explain them by comparing our present method with the old, conventional ones.

3.2 Reflections on Our First Trials

In Section 2.2 we have sketched out our first trials on modular code systems. Reflections on these trials were the starting point of our second trial. Our experiences in the first trials taught us that we need a big amount of man-months to develop and maintain a so called modular code system. We were forced to recognize the fact that we need to have permanent groups continually, if we wish to make survive a big modular code system. At JAERI it is our custom to promote any software project with a very limited number of staffs. Hence we

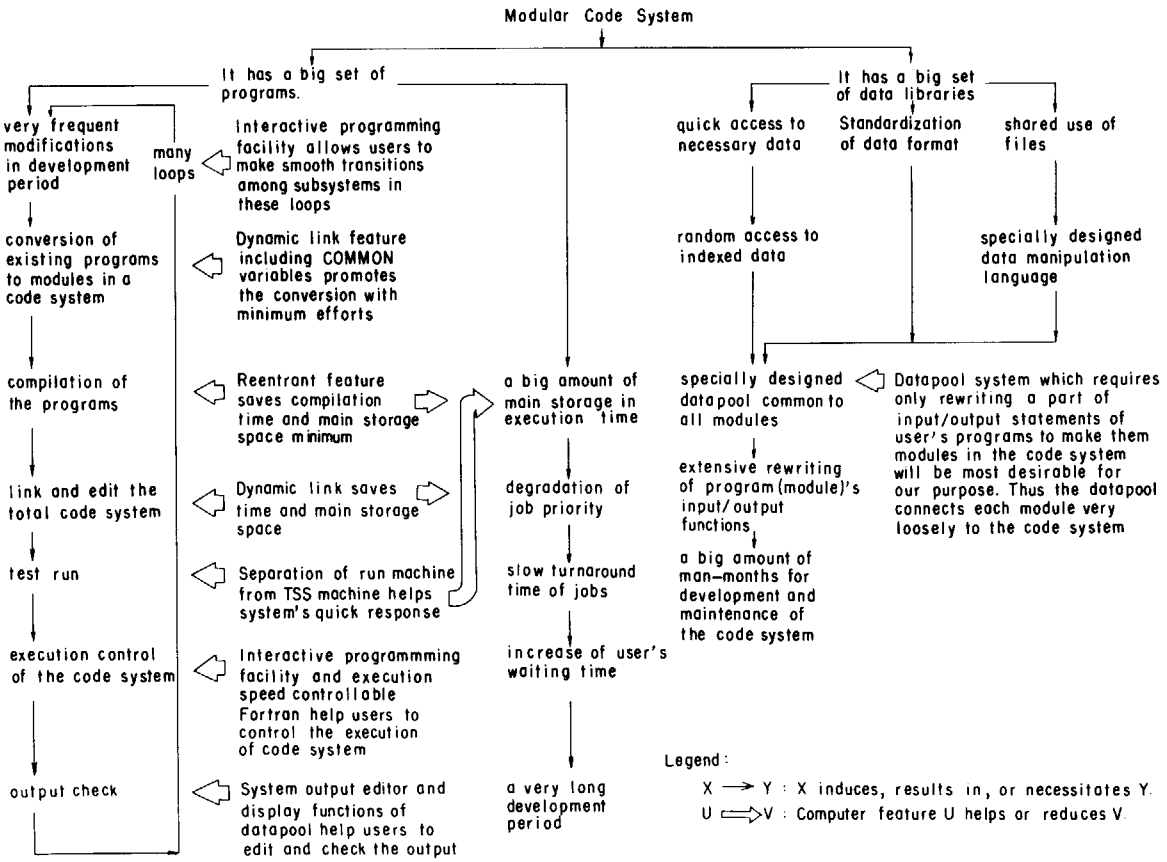


Fig. 3.2 Relationship between Required Tasks and Computer Features

cannot invest much manpower for a software project. This custom is an obstacle to the construction of big code systems. The experiences in first trials, however, suggested us that we might construct and maintain a big modular code system without spending much manpower, if we could make use of advanced computer features. In **Fig. 3.2** are shown the relations between tasks required for construction of a modular code system and the advanced computer features estimated by a rough analysis based on our experiences. In the period of the first trials we had none of the features. The analysis had led us to the development of computer features for code systems described in the following Section 3.3.

3.3 Development of Computer Features for Code System

In 1975 the Modular Code Subcommittee was reorganized as Modular Programming Subcommittee. It was expected to engage in following two activities:

(1) to assist scientists and engineers who have needs to construct code systems,

(2) to develop software tools which support the modular programming methods and to promote easy implementation of modular code systems at JAERI.

As for the results of activity stated in the item (1), we will see some examples later, in the Chapter 4. Before determining types of software tools stated in the item (2), it was necessary for the subcommittee to select what structure of code system would be suitable for scientists and engineers of JAERI. After discussions in its meetings, we shaped a concept of ideal modular code system. There might be several types of structure for code systems. We have characterized the merits and requirements of our code systems as described in the following **Table 3.1**.

Table 3.1 Merits and Requirements on Code System

Merits		Requirements
1. Simple use	1.1 Reduction of user works consumed in editing of input data preparation, output tabulation and graphical presentation.	1.1.1 Output data of a program should be accepted automatically or semi-automatically by other programs which use the data.
	1.2 Both batch and interactive accesses to codes or data files.	1.2.1 Provision for file manipulation, including display and tabulation of input/output data by timesharing terminals should be offered to users.
2. Compatibility with other computers	2.1 Independence of a program or a module from a code system or a computer which supports the program.	2.1.1 A program or a module and its data of a code system should be dependent only on input/output methods of the code system. The structure of a code system may be dependent on a computer.
	2.2 Separation of data from a program.	2.2.1 Data of common use should be independent of a program and must be subject to easy operations such as display, tabulation, and graphical output. This means that some files of the code system must be structured to contain data which have catalogued names.

3.3.1 Desirable Characteristics of Code System

In addition to the requirements stated in **Table 3.1**, desirable characteristics of code system are as follows:

i) Open ended system – Users must be free to add a new program (module) to or delete an existing program (module) from the code system.

ii) Labour and cost saving system – Simple labour which a computer can do must be computerized. Already existing programs should be fully utilized in the code system with minimum modifications.

iii) Easily maintenable and changeable system – A modification or change of a program must be done only in the scope of the program itself and should not give a considerable influence on the code system.

iv) Easily usable system – The paths of computation should be controlled by a simple, plain language.

According to the desirable characteristics of the code system it was decided that the system and its modules should not be combined tightly with each other so as not to be influenced by a simple modification, addition or deletion of modules on the system. Considering these requirements we determined to promote developments of following softwares.

a) Construction of Datapool

Construction of a JOSHUA database-like subsystem was planned. The subsystem would create and support hierarchical tree structured files. The users of the code system would be recommended to use the files to transfer, accept, retrieve, and display data of modules. This datapool was not the same one as described in Section 2.2.1 and 2.2.2, but it was a general purpose utility subsystem.

b) Provision of Dynamic Link Feature

Every module in the module library should be loaded into the main storage of the computer and should be linked to the caller module dynamically when it is called at the execution. These are because that the amount of modules of a code system will become giant, and that it is very difficult to provide a storage space for total modules at one time. In the conventional methods this difficulty is avoided by overlay features or by a very restricted type of dynamic linking. In the restricted type of use the user is forced to change his source programs to a considerable degree. However we thought that the user who wished to use the feature should not be imposed on a tedious and time consuming translation work to change his program to fit into the scheme of the feature. The user's program should be used without any modifications. Later the reader will see that this is accomplished on our computer systems.

c) Construction of Precompiler

The user who wishes to use his program in the code system with the datapool should not be imposed on a time consuming and error inducing work to tailor the input/output procedures of his program to fit it into the frame of the datapool system. It was decided to construct a precompiler which translates a source program with datapool input/output statements into a normal Fortran program.

d) Construction of System Output Editor

A code system will produce a considerable amount of system output, i.e., output produced by Fortran statements WRITE (6, n) I/O-list. The output file for this statement is not under the control of the datapool subsystem and there was a need to construct a system output editor specially designed to retrieve, display, edit and print the selected output.

The output editor was designed to be operated using a timesharing terminal. As for the terminal device, the Tektronix T-4014 was selected because of a capability to display a full

page space same as the usual line printer and high transmission rate of 9600 bits per second.

e) Construction of Simple Path Controller

It was decided to construct a simple path controller. As the language for the path control we chose the Fortran language because of its common use and compatibility. The controller was determined to be an interactive one. This was because firstly it was desirable for the user to control the paths easily and secondly it was deemed necessary to control the execution by watching the computational results of some modules. As the controller it was planned to use our timesharing processor which was previously mentioned in Section 2.2.4 and was being developed at that time.

3.3.2 Schedule

In 1975 the subcommittee planned to accomplish the framework of its code system in five years, and set the schedule. Some of items described in the previous section have been accomplished and are currently used, some of other items had been accomplished, used for a few years and then became obsolete or were replaced by more powerful alternatives, and remainder were abandoned as not attainable.

We will describe briefly about the results of them.

1) First Year

a) Precompiler

A precompiler to process the CJ-statements, i.e., statements for the datapool I/O operations was implemented. It was a modified version of Fortran syntax checker already developed for the TSS-Fortran processor which was described briefly in Section 2.2.4. At first the precompiler was written in a system writing language GPL[14], a modified version of the famous PL360[21]. Now the precompiler has been completely rewirtten in Fortran.

b) System Output Editor

A system output editor was designed to assist users of code system to retrieve, display and edit the normal system output. It was called as SYSOUT and intended to use the Tektronix T-4014 storage tube type terminal in timesharing environment. The device has capacity to display 132 characters in a line, 64 lines in the screen. Its maximum data transmission rate is 9600 bits per seconds.

The system output editor gave sequential page numbers to the output in the system output file. The user used the number as an access key to the output. The page number was displayed in the lowest line in the screen with normal output over the page number. Backing or forwarding was done by −n or +n command. Extracting, merging, printing or transferring pages was executed by a simple command. Following specifications proposed by us, the computer manufacturer made the processor.

It was used for some years by several users, but the circle of users did not expand so much. It was because that we could not provide users with enough disk storage to store their output, congestion of timesharing use and 2400 bits per second transmission rate made the speed of display slow, limited number of terminals restricted users to use freely, and average amount of output was not small, and so on.

A few years later, the computer manufacturer delivered a new system output editor. It was a refresh type terminal directly connected to a multiplexer channel of host computer. It can display a page in one second[22].

c) Pseudo Dynamic Link Feature

As was mentioned in Section 2.2.4, the FACOM230-75 computer could not afford the so called dynamic link feature. This was mainly because of its hardware architecture. Accepting our request for availability of dynamic link feature, software engineers of the

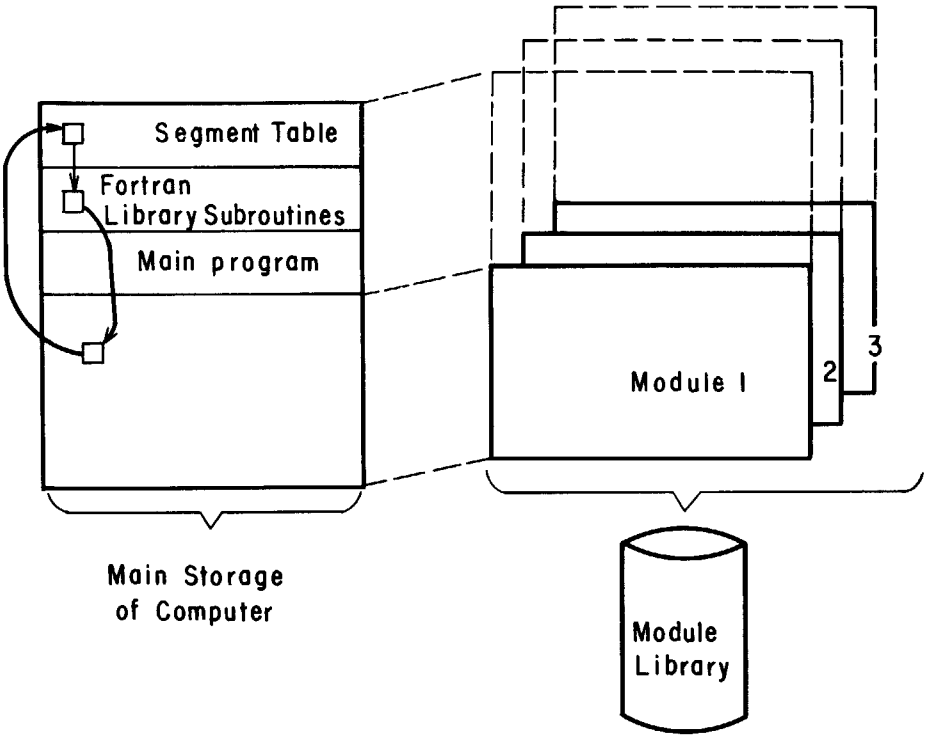


Fig. 3.3 Pseudo Dynamic Linking for FACOM230-75 Computer

computer manufacturer squeezed out a tricky technique to facilitate a very simple dynamic linking. The method is illustrated in **Fig. 3.3**.

The method was simple extension of overlay structure. A module was an executable binary form subroutine or a set of subroutines and the subroutines had no external calls except the Fortran library subroutines. The module was link-edited by a linkage editor specially designed for this purpose. Each module was assigned a same address, as is shown in **Fig. 3.3**. By this reason the main program, or a path driver in the sense of code system, could call only one module at a time.

If a module called a Fortran library subroutine, the control was transferred to the corresponding location. In the location the exact address of the library subroutine had been registered by the linkage editor. The transfer of the control is shown by arrows in the **Fig. 3.3**.

The reader will find nothing new ideas in this method. The point was that the procedures to make and execute modules were automated by the computer manufacturer's standard software. Actually it was able to use the method by a mere specification in the job control card.

The software was delivered in due date but we had a very few chances to use it. It had a restriction that the user could call only a module at one time and could call no other module from a called module. On the contrary to our anticipation, this was a severe constraints on our applications.

2) Second Year

a) Datapool

In the second year a first version of the datapool was constructed and became available. The first version had not had a command processor for timesharing use. In addition to it, in the first version all directories of nodes of a hierarchical, tree-structured file had been resident in main storage. This forced the user of the datapool to consume a big amount of main storage, to endure a low job priority because of its storage requirement. The defect

was corrected in the fourth version.

3) Third Year

a) Command Processor of Datapool for Timesharing Use

In the third year a command processor of datapool in a timesharing environment was developed. The functions of the processor are

i) to display attributes of data,

ii) to display data literally or graphically,

iii) to add or delete data,

iv) to compare data in the datapool files.

b) Document Editor

In constructing a framework of code system it was felt necessary to provide developers and users with a some sort of document editor. It was planned to make an editor which had capability of the famous ROFF, plus newly devised functions for screen formatting, file handling, etc. for documents written in English[23]. Its character editing functions were based on a concept of the so called full screen option presented by the computer manufacturer and other's were based on miscellaneous ideas collected in our institute. Our timesharing terminal devices at that time had no full screen option and we were afraid that the inconvenience in use would result from the concept and lack of the functions on devices.

After the construction of the editor and in initial test phase, this anxiety became actual, and we were forced to revise the editor. In some part this was done in next year.

4) Forth Year

At first, before the first year, it had been planned to construct a path control processor which was to accept a command like language in batch and timesharing environments. We had a plan to use our TSS-Fortran for this, but as was mentioned in Section 2.2.4, the development of TSS-Fortran was abandoned. We chose the standard Fortran language as its alternative.

a) Revision of Datapool

In this year some users began to use the datapool software. They found several errors and inconveniences in it. The fatal one was the memory usage of the datapool. The datapool was designed to use a rather big amount of main storage. The designer of the datapool neglected the problem because he implicitly expected a computer with virtual storage and dynamic link feature. These features are essential not only for code systems but also for big utility programs such as the datapool to use them with small size of real storage. Unfortunately the computer on which the datapool was implemented had no such features. This defect of the datapool was amended by separating subroutines and some extensions were added soon but the happening and initial bugs made the datapool notrious for a time. The current datapool is a revised version[15] of this and it is recognized as useful.

b) Revision of Document Editor

In a tentative use of the document editor, there had been found many clumsy and inconvenient specifications and functions. They were modified in this year. The editor was written in a system writing language named SPL, developed by the computer manufacturer. At first the language compiler and hence the editor itself seemed easy to be implemented on our new computers. After an investigation, however, it was found unsuitable to rewrite the processor of the editor on new computers, and its maintenance was abandoned.

Two years later the computer manufacturer has made a new document editor picking up some functions of the abandoned one and adding new functions such as graphical representations of mathematical formulas, greek letters, etc.[24].

5) Fifth Year

In the fourth year it had been decided to replace our old computers by new ones. The

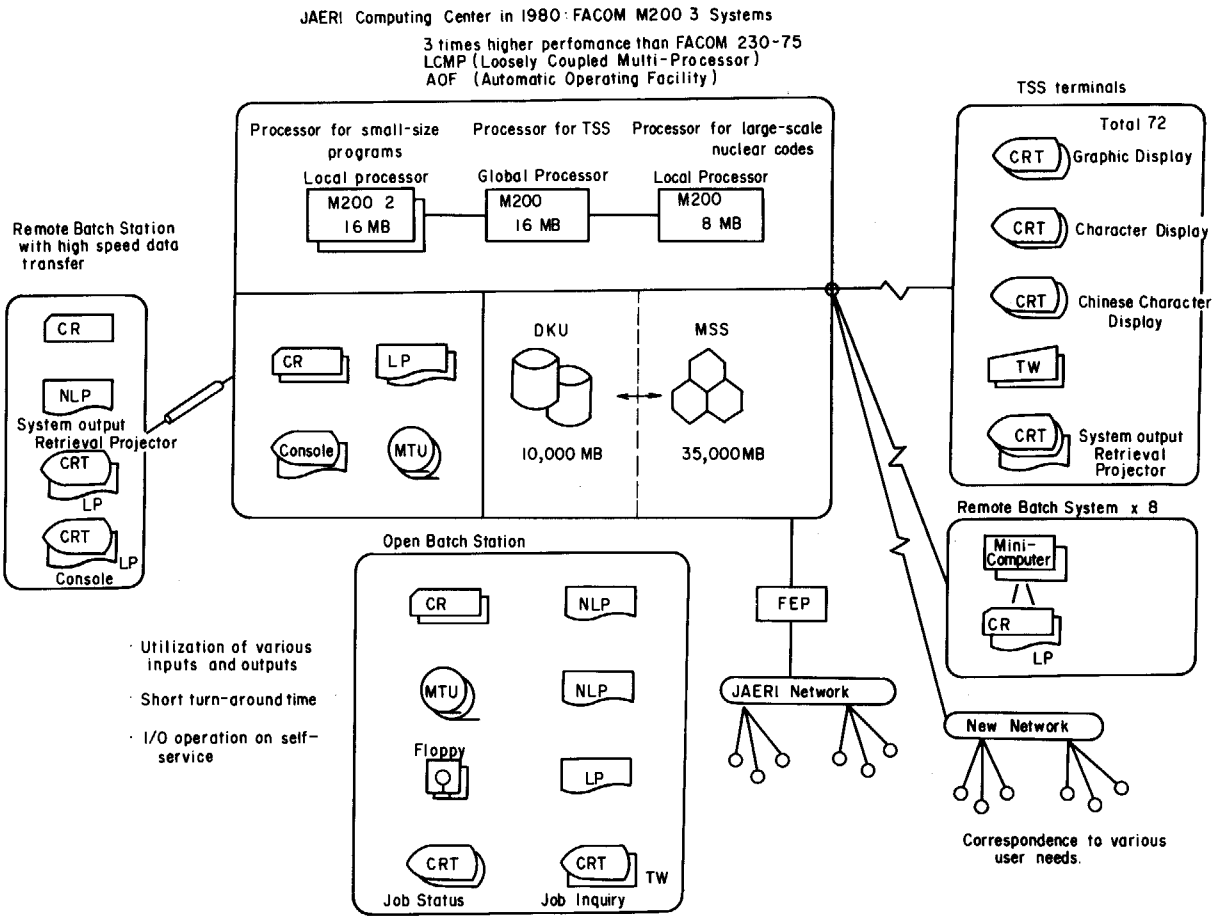


Fig. 3.4 Computer Configuration at JAERI in 1980.

new ones were installed and became operational in the fifth year.

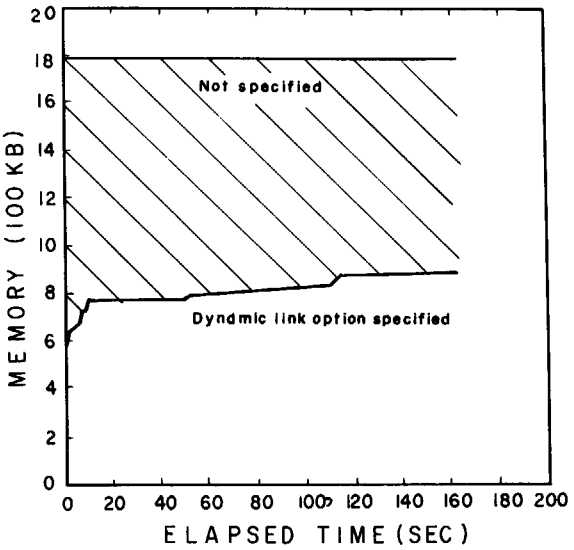
The new computers, three FACOM M-200 systems with four central processing units in total are currently operated by LCMP, i.e., loosely coupled multiprocessor mode as shown in **Fig. 3.4**. The configuration was determined from viewpoint of easiness to develop computer programs and quickness to run large-scale nuclear codes. Each central processing unit has 10 MIPS, i.e., 10 million instructions per second performance. The computer system is compatible with IBM370 computers in source program and load module forms.

By the introduction of these systems many long desired softwares such as dynamic link[25], reentrant[25], system output editor, document editor, interactive programming facility[18], execution speed controllable Fortran[17], and hardware features such as virtual storage, mass storage system, high speed display terminal became available.

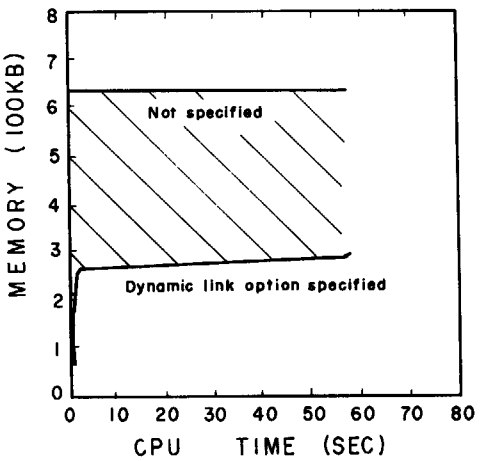
It might be said that the above mentioned softwares are produced from stimuli given by our concept on code systems. Especially the fact that the dynamic link feature can link even Fortran COMMON areas dynamically is very important because without this we are forced to rewrite every program extensively to fit it into a code system.

Before and after the delivery of these features two software tools have been developed by two contract based visiting staffs from the computer manufacturer. One is a Fortran source program analyzer ANALYSIS which reads source programs and produces tree structured caller-called subprograms relations, categorized listings of variables, etc. The other is a measurement tool DYNALEAT which, reading a source program, inserts CALL statements into every main and subprogram to call a data logging subroutine[26]. The logging subroutine writes on a datapool file the time and storage use of the caller (sub)program. By an execution

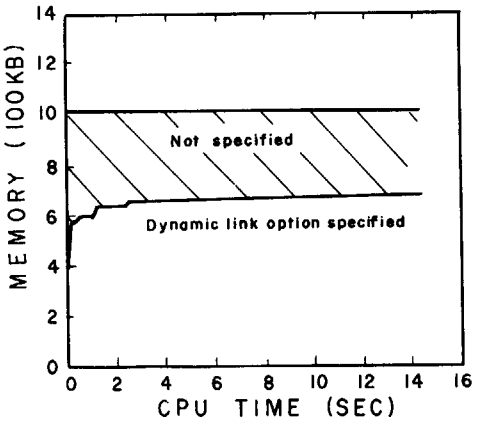
of the program with a sample input data the accumulated data on storage use is displayed using timesharing commands of the datapool. The examples are shown in **Fig. 3.5**. In the **Fig. 3.5**, every shaded area indicates main storage which is saved due to the dynamic link option.



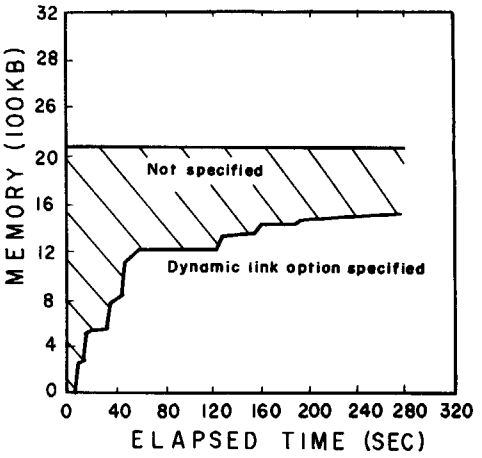
CITATION: Three dimensional diffusion code, ORNL-TM-2496, 3793,



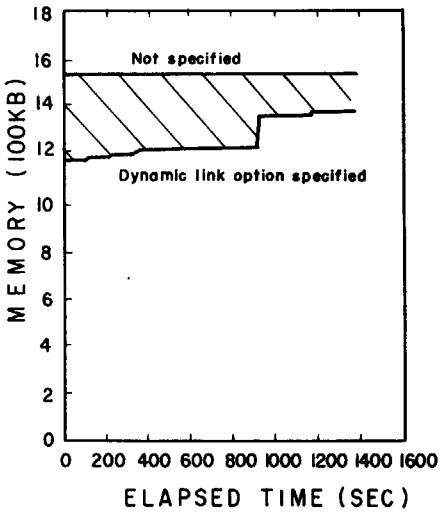
MORSE-CG: Monte Carlo radiation transport code, ORNL-4972,



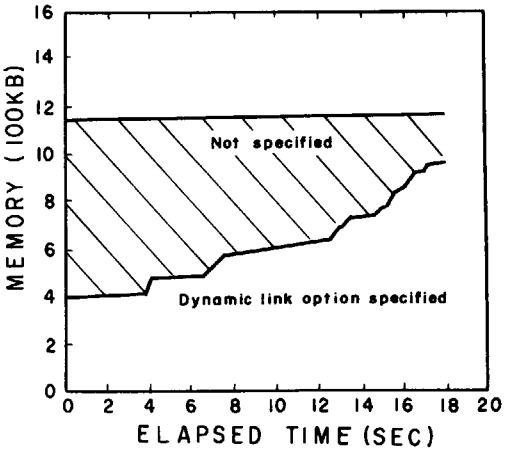
EQUCIR: Free-boundary toroidal MHD equilibrium code, JAERI-M 9127.



VENTURE: Three dimensional neutronics code, ORNL-TM-5062, 5065,



ANISN: One dimensional discrete ordinates transport code, ORNL-RSIC-CCC82,



ALARM-P1: Blowdown analysis code for PWR, JAERI-M 8004,

Fig. 3.5 An Example of Storage Saving by Dynamic Link

3.4 Accomplishments by Our Development

3.4.1 Advantages of Our Method over Conventional Ones

The softwares listed in **Table 3.2** are the accomplishments obtained by our development. On the basis of the fact that they are available now, we can compare our method for code systems with the old, conventional ones.

The comparison of our method with conventional ones is shown in **Table 3.3**. In the table we have made comparison with respect to qualitative characteristics of the methods. The reader, however, will recognize the quantative differences of the methods. In our present method developers and users of a code system can enjoy reduction and saving of tedious works and computer resources to a considerable extent when they are performing the development and run of a code system. Furthermore we can construct a code system with high modularity as is shown in **Fig. 3.6**. In Chapter 4 we will see that this structure is realized in SPLPLOT2 system. Thus we may expect that a considerable amount of works consumed in the item "Extensive effort for development and maintenance" listed in **Fig. 3.1** can be replaced by the item "Development aids."

Table 3.2 Softwares for Code Systems

		Software	Invested Man-Month	
			JAERI	Manufacturer
1.	Dynamic Link	F75-DYNA*1	2	2
		Standard*2	10	59
2.	Reentrant Programming	Standard*2	1	50
3.	Interactive Subsystem Control	TSS-Fortran*3	2	–
		IPF*2	–	6
4.	Interactive Fortran	TSS-Fortran*3	40	–
		DOCK*2	–	180
5.	Document Editor	K-System*1	10	30
		ATF*2	–	70
6.	System Output Editor	SYSOUT*1	2	12
		SORP*2	–	120
7.	Data Storage and File Handling	DATAPOOL*2	50	–
8.	Fortran Source Program Analyzer	ANALYSIS*2	5	–
9.	Dynamic Link Effect Analyzer	DYNALEAT*2	2	–
	Total		124	529

Note: *1: Now obsolete,
*2: Current alternative in use,
*3: Abandoned,
Standard: Standard product of manufacturer.
Manufacturer means the computer manufacturer, Fujitsu, Ltd.

Table 3.3 Softwares for Code Systems

Item	Conventional Methods for Modular Code Systems	Our Method for Modular Code Systems	Computer Features for Our Method
Differences	○ Imcompetent computer features forced users to consume a long time and expenses for the development of a code system. Furthermore modules (subroutines or programs) in the code system are changed so often as not to be available to other applications.	○ Using new features of operating system and language processor, no specific restrictions are put on modules in code systems. Minimum modifications are required to fit a subroutine or a program into code systems. Only modification required is for input/output statements to read or write data in datapool files.	
Gradual Development of Code System	○ Data are transferred between modules by an unified method unique to the code system, and it is needed to rewrite a considerable part of a subroutine or a program when it is to be used as a module in the code system. For this reason it is very difficult to construct and maintain modular code systems.	○ Existing programs and subroutines are used mainly as modules in the code systems. Data are transferred between modules by methods which have been used in the original subroutines or programs, i.e., via subroutine parameters, variables in common areas, or files. The datapool files are sometimes used for this purpose.	Dynamic linking of program module
	○ Users of conventional modular code systems are prohibited easy use of their non-standard modules and therefore the modular code systems are forced to equip many modules to provide versatile, flexible or a alternative use of modules in the code system.	○ Methods of transferring data between modules are not unified in a code system. The user must pay attention to the method of data transfer of his non-standard module when he wants to use it in the code system.	
		○ Informations such as physical meanings, methods of data transfer, etc. of a module are maintained by the code system and users of the system can retrieve the informations using the information retrieval facility of the datapool. By these informations users can make an interface routine between his non-standard module and the code system.	Datapool

Table 3.3 (Cont.)

Item	Conventional Methods for Modular Code Systems	Our Method for Modular Code Systems	Computer Features for Our Method
Module Construction	○ Existing subroutines or programs must be rewritten to make them modules in a code system. Each module must obey the logcial structure of the code system. Since a unique and fixed method is usually adopted for data transfer between modules, it costs big expenses, consumes a long time to make use of existing subroutines and programs as modules in a code system.	○ It has no need to rewrite existing subroutines or programs to fit them into a code system except input/output statements for transferring data via datapool. Sometimes an interface routine is added to a subroutine or to a program to adjust common variables or subroutine parameters when the subroutine or program is adopted as a module in a code system.	
Flow Control of Program Modules	○ It is often done to develop a simple language and its processor to control the flow of modules in a code system. Hence for the implementor for a code system, it needs time and expenses to develop the language processor and for users of the code system, it needs time to learn the language specification.	○ Fortran language is used for module control. Since the flow control program is a Fortran written user program or a standard program of the code system, it is processed in TSS or BATCH mode operation.	
		○ The user of a code system can make his own flow control programs and can register them in the code system.	
		○ Interface routines to adjust information transfer between modules are mainly called by the flow control program, so that there is no need to change modules or user's own programs.	
		○ The interactive programming facility of our computer software admits the users to branch the flow control to TSS command mode. So the user can retrieve information of modules, can display and edit data for modules rather easily.	Simultaneous use of TSS and BATCH mode
		○ In the interactive use of a code system, the user can submit a giant module of big memory and long execution time to a subsystem (a computer system dedicated solely for giant jobs). After execution of the user can accept the calculated results via user's files and he can also wait the execution of the giant job using other subsystem for TSS or BATCH processing.	Simultaneous use of TSS and BATCH mode and JES3like control of distributed computer systems

Table 3.3 (Cont.)

Item	Conventional Methods for Modular Code Systems	Our Method for Modular Code Systems	Computer Features for Our Method
Linking and Execution of Program Module	○ All modules which are referred to are linked before execution. By this fact users must provide a big amount of memory to run jobs.	○ All modules are linked dynamically when they are actually required in the execution.	Dynamic linking of program module
	○ In addition to the above linking method, users can use a dynamic link feature in a restricted manner, i.e., dynamic linking of modules and information transfer via subroutine parameters or files. Users and the code system implementor are forced to consume time and expenses to rewrite subroutines and programs because the above dynamic linking method does not allow the link of common areas.	○ Common area of subroutines are linked dynamically and there may be few need to rewrite subroutines or programs to fit them into a code system.	
	○ Usually a flow control program executes modules by activating several batch job steps.	○ The flow control program calls dynamically the program editor, compiler, modules of the code system. When a module is needed to execute on a high speed computer, by entering the TSS command mode and submitting a job, the user can send it to the high speed computer in the system and he can get the calculated results in files. Owing to these features the user is able to make input to the code system by trial and error method and as the results he can use the computers and code system as if a simulation device.	The same feature as the above and JES3 like control of distributed computer systems
Use of Data Module	○ Data are transferred via subroutine parameters or files. Users must use files of the code system specified for this purpose. These files are often called the datapool, databank, or database of the code system.	○ Data are transferred via subroutine parameters, common areas, or files. The files may or may not be datapool files.	Dynamic linking of program module
	○ The databank system has facilities for display, modification, addition, and deletion, etc.	○ The datapool system has a set of commands and the command processor for display, addition and deletion of data. Each data may have comments which explain the physical attributes of data. The user can display, modify, add, the delete the comments.	Datapool
	○ Users can tailor data in the databank using a language and its processor equipped with the databank.	○ Users must provide programs to tailor their data in their files to make the data usable in the code system.	
		○ The datapool system has a facility of information retrieval for data, files or modules.	Datapool

Table 3.3 (Cont.)

Item	Conventional Methods for Modular Code Systems	Our Method for Modular Code Systems	Computer Features for Our Method
		○ The datapool can store with data the programming attributes in input/output list, a character string form of the input/output list, and comments for each input/output statement to datapool files. Using these informations the user can modify and produce new data in interactive programming mode.	Datapool and simultaneous use of TSS and BATCH mode
		○ Users can share one copy of every procedure in the code system because in our computer system users are able to specify their Fortran programs as reentrant.	Reentrant feature of user program
		○ By virtue of virtual storage machine, the interactive procedures in the code system are processed quickly and effectively.	Virtual storage
Registration of Modules		○ There are two types of registration in our code system. The standard module is a module which is tested and recognized as valid by users groups. It is ready to use in the code system with data for information retrieval. The non-standard module is a module which is under test. It is registered in the code system with its data for information retrieval. The user can use these modules or his own module in his responsibility if they are non-standard ones.	Datapool
Development and Run Aids		○ Development tools for program structure, dynamic link effect, document editor, quick display and editing of output will accelerate the development and use of code system.	Program analysis tools, Document editor, System output editor
		○ The execution speed controllable Fortran will encourage users in employing the highly interactive use of code system.	Execution speed controllable Fortran

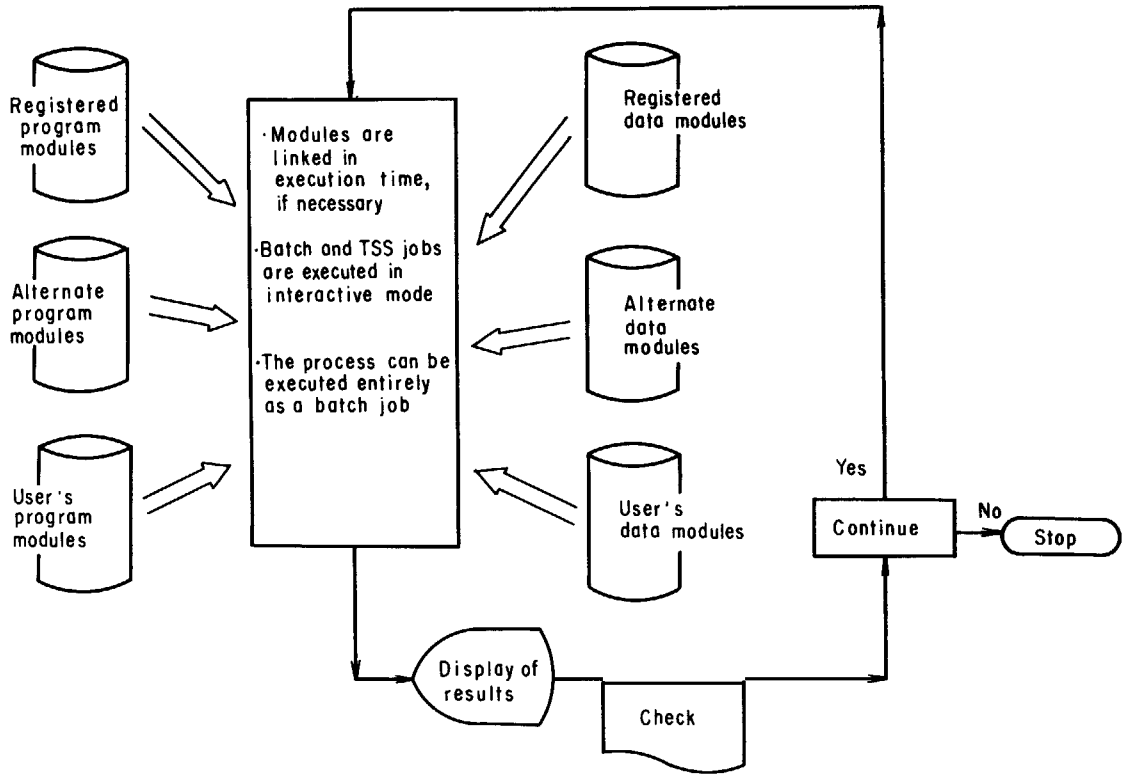


Fig. 3.6 An Ideal Structure of Code System

3.4.2 Obtained Findings

We have now some experiences with the softwares listed in **Table 3.2.** It will be very valuable for us and other research organizations to summarize them briefly. Since it is needless to say about the merits of the softwares, we make some mention of points which should be improved further or taken care of for new development.

1) Dynamic link – Load module using this feature requires a half more disk file area. This is because each compiled program contains a set of tables to keep unresolved external calls. This is necessary but the fact often surprises novice users of the feature. A BALR i.e., branch and link instruction is used for connecting a module dynamically. Once the module is connected, i.e., linked to the caller, it is not able to separate the linking. Linked modules begin to remain in the job's virtual space. However if we wish to remove this defect, the system overhead, i.e., interruption handling time will increase. The problem needs more investigation.

2) Reentrant – This saves a storage space by sharing common program (procedure) parts in the storage with other jobs. Currently we have no common utility or code system which is accessed by many users at the same time. It is expected that this feature is required in the near future.

3) Interactive subsystem control (IPF) – This is a very useful tool and will become more if it is permitted to make dormant the timesharing session for a while without logging off.

4) Interactive Fortran (DOCK) – The current version is made by modifying a Fortran 77 compiler. Although it requires a big amount of storage, it seems a powerful tool for debug and control of the execution of a code system. We need, however, more experiences before to say something about this.

5) Document editor (ATF) – This is an editor for documents written in English with

Greek letters, mathematical expressions, etc. The current version is not able to be called by a user's program. If it is able to be, it will be very useful.

6) System output editor (SORP) – The terminal devices for this editor are more used as usual timesharing terminals than as their original use. This is because the demand for TSS terminals is strong and a user cannot occupy the SORP terminal for a long time. The merit of printed output would not be overcomed unless we can provide each user with this type of terminal device.

7) Data handling and filing system (Datapool) – This may be called a general purpose data storing and retrieving system for Fortran programs. Because of its generality, it is a rather big software for casual users for small code systems. It also not suitable to use in a code with heavy and fast sequential I/O demands. It will require improvements especially for its graphic functions.

8) Fortran source program analyzer (ANALYSIS) – The user is enjoying the versatile options and neat output of the analyzer.

9) Analyzer for dynamic link effects (DYNALEAT) – This is very useful for analyzing the effect of the dynamic link feature on storage usage. Its output is stored in a datapool file, but the graphical output of the datapool is not elaborated for easy looking and should be changed so as to be seen more clearly.

(Kiyoshi ASAI, Computing Center)

4. Applications

As is mentioned in Chapter 1, accepting the request of the subcommittee, in the last one year several code systems have been developed[27),28)] or being developed[29),30)] using our present method. Some of them are utilizing almost all of the features listed in **Table 3.2**, and others are using a part of them. In this chapter we will show two examples of them. In Section 4.1, K. Tuchihashi describes the application of our datapool to a code system SRAC and the experiences with it. In Section 4.2, K. Muramatsu describes the application of the interactive subsystem control IPF and the dynamic link feature to a code system SPLPACK.

4.1 Application of Datapool to Data Storage of SRAC Code System

4.1.1 Introduction

The SRAC code system has been developed as a neutronic calculation subsystem of the JAERI thermal reactor standard code system (**Fig. 4.1**). A stand-alone version of the SRAC code system recently succeeded[31)] in interpreting the several experimental critical masses of TCA (a critical facility of light water power reactor), DCA (a critical facility of ATR, heavy water moderate light water cooled pressure tube type reactor) and core patterns for VHTR simulated by SHE (a graphite moderate semi-homogeneous critical facility).

In the SRAC code system PDS (partitioned data set) files of binary mode have been used for data storage. A PDS file has a simple structure so that a member containing one dimensional binary array can be accessed by a member name composed of eight characters. An assembler language subroutine allows read/write from/into PDS files by Fortran programs. No further utility is required because the utility for partitioned data set files installed in the

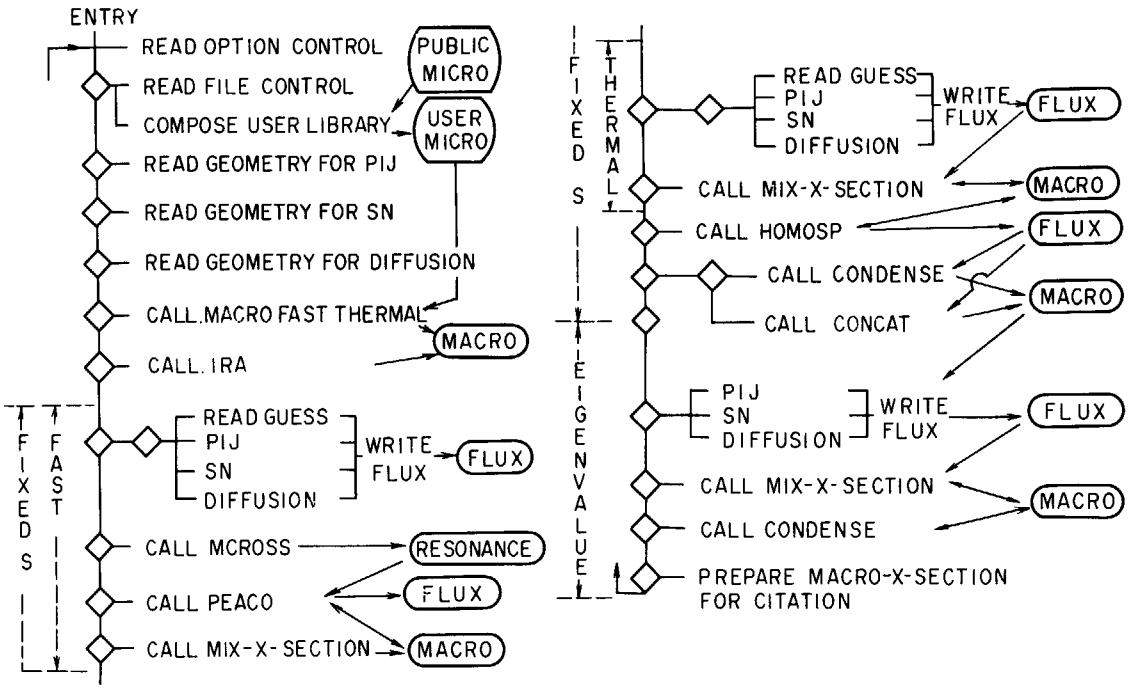


Fig. 4.1 SRAC Flow Diagram

present operating system permits the timesharing commands such as 'allocate', 'copy', 'list directory', 'delete', and 'condense'.

Although the use of the JAERI datapool had been intended since the draft stage of the SRAC, we had to refrain its application until its performance was verified to be complete because the merge of two under-developing software, we anticipated, brought more difficulty in debugging.

Since the function of JAERI datapool has been fixed[15], and the conversion of pre-processor from FACOM230-75 to M-200 computer has been finished and also the performance of SRAC has been proved good, it is the time, we think, to convert the data storage method of SRAC.

It has been foreseen that the use of PDS files encounters difficulties in controlling the variety of data by such a simple structure where subdivision of eight character field of a member name has been used to express a series of physical quantities. Now the hierarchical data structure by concatenation of node names allowed in the datapool releases us from elaborating tasks to compress several items into a member name.

In this section the experience gained through the conversion will be discussed.

4.1.2 Data Files Used in SRAC

The information transferred by catalogued disk files falls into four categories;

1) The basic nuclear libraries for general usage

Data are compiled on the finest energy group structure over whole range of temperature for all possible nuclei. In this category we have three files, a) fast neutron library, b) thermal neutron library, c) resonance neutron library.

2) The secondary nuclear libraries for the particular users

To save the computer time each user can construct his own libraries of the same format and structure as the basic ones on less fine energy group structure, on certain range of temperature, for selected nuclei. These files are used throughout a series of calculations.

3) The case dependent information

In a case we can perform a cell and a core calculation. Usually successive several cases prepare the smeared cross sections of different cells and a core calculation is performed in the final case. Three files are used to store and transfer the information betwen cases; a) macroscopic cross section file, b) flux file, c) effective microscopic cross section file.

4.1.3 Hierarchical Structure of Data Files

Our process was initiated by deciding the hierarchy of nodes in each file. The preliminary work had been done which resulted in the structure of member names of PDS files. For example, a member 'MU08W002' in the basic thermal library implies a matrix (by M) of uranium 238 (by U08) in the compound Uo2 (by W), not being specialized by the cell calculation (by 00) for temperature of 350 degree (K) (by 2). We started by exchanging the symbols appearing in a member name into a self-explanatory character string which described their physical meanings. To the member in the above example the node name of 'U-238. U02. MATRIX. T2' was supposed. The index of the cell calculation history was not considered for the simplicity at the initial stage. The subdivision of nodes in the fast neutron library was taken to clarify the kind of reaction. A node of 'MATRIX' which had been packed into a one dimensional array to avoid too many members in a PDS file was split into several arrays under the node name of 'CAPT', 'N-N', 'ELPO', etc. (**Fig. 4.2**).

Fast neutron library (basic, user's)

```
'FASTLIB' 150*(energy structure weight)
NUCLIDE          .'CONT' 20 (control of nuclide)
                 .'MATRIX'        .'CAPT'    74 (capture X-section)
                                  .'FISS'    74 (fission X-section)
                                  .'FNU'     74 (fission yield)
                                  .'ESPC'    74 (fission neutron spectrum)
                                  .'TR'      74 (transport X-section)
                                  .'WEIGHT'  74 (weights)
                                  .'ELAS'    74 (elastic X-section)
                                  .'N-N'     300 (inelastic matrix)
                                  .'N2N'     300 (N2N matrix)
                                  .'ELPO'    8000 (P0 matrix)
                                  .'ELPI'    8000 (P1 matrix)
                 .'FTAB'          .'TR'      1800 (f-table for transport)
                                  .'CAPT'    1800 (f-table for capture)
                                  .'FISS'    1800 (f-table for fission)
                                  .'ELAS'    1800 (f-table for elastic)
                                  .'ELAR'    1800 (f-table for elastic removal)
                                  .'WEIGHT'  1800 (f-table for weights)
                 .'RESP'          .'CONT'    50 (control for resonance)
                                  .'S-WAVE'  1500 (s-wave parameters)
                                  .'P-WAVE'  1000 (p-wave parameters)
                                  .'SMOOTH'  .'CAPT' 5000
                                             .'FISS' 5000
                                             .'ELAS' 5000
```

* Numbers following node name denote the maximum record size in words

Thermal neutron library (basic, user's)

```
'THERMAL'        .TEMP 100 (energy structure, weights)
NUCLIDE          .'CONT' 30 (material name control)
                 .'MATRIX'        .TEMP      2600
                 .'FTAB'          .TEMP      1600
```

Resonance neutron library (basic, user's)

```
NUCLIDE          .'CONT'          .TEMP      50 (control)
                 .'CAPT'          .TEMP      20000 (capture X-section)
                 .'FISS'          .TEMP      20000 (fission X-section)
                 .'ELAS'          .TEMP      20000 (elastic X-section)
```

Macroscopic cross section file

```
'CONT'           .ERANGE          .FINE      220 (energy structrue, weights)
MTNAME           .ERANGE          .FINE      .'00'          .L    4000
CASENAME         .ERANGE          .FINE      .XREG-NUM .'TR'  4000
```

Fig. 4.2 File Structure of SRAC

Flux file

```
MTNAME      .ERANGE     .FINE     .'00'          .L    110
CASENAME    .ERANGE     .'VOL'    20 (Volumes of R-regions)
CASENAME    .ERANGE     .FINE     .'00'          .L    2200
CASENAME    .ERANGE     .FINE     .XREG-NUM      .L    110
```

Effective microscopic cross section file

```
'CONT' 50 (energy structure, weights)
NUCLIDE     .MTNAME     .XREG-NUM 500 (. BRUNUP-STEP)
```

Physical meanning of node name

'FASTLIB'	String enclosed by ' is used as a constant
NUCLIDE	Alphabetic chemical symbol followed by mass number and specialized by compound tag as PU239, U-235, U-238W, H-001H, etc
ERANGE	Neutron energy range expressed as 'FAST' or 'THERMAL' or 'ALL'
FINE	Energy group structure defined as 'FINE' or 'COARSE'
MITNAME	Mixture name specified by user as 'FUEL01'
L	Spherical harmonics indicator; 'P0' and 'P1' are self-explanatory 'TR' denotes P0 components after transport correction
CASENAME	Alphabetic Case label specified by user
XREG-NUM	Two digits as '01', '12' to specify the cross section smearing region
TEMP	Temperature tag as '1', '2' - - - - -, '9', 'A'
BURNUP-STEP	Future use for burn-up step tag as 'B0', 'B1', - - - - -, 'B9'

List of catalogued DASD

```
FAST LIBRARY (PUBLIC)
THERMAL LIBRARY (PUBLIC)
RESONANCE CROSS SECTION (PUBLIC)
FAST LIBRARY (USER)
THERMAL LIBRARY (USER)
RESONANCE CROSS SECTION (USER)
MACRO CROSS SECTION (USER)
FLUX (USER)
EFFECTIVE MICRO CROSS SECTION (USER)
```

Fig. 4.2 (Cont.)

4.1.4 Conversion of Program and Data

The next work for conversion was to translate the whole read/write statements in the SRAC for the PDS files into CJ-statements for the preprocessor of datapool. The possible reduction of core storage requirement due to the subdivision of node was not considered. Instead of conversion of library generation programs, the ad hoc programs were used to convert the data from the PDS files into the datapool files.

Now we shall discuss the result of conversion using an example in which a) condense and selection of data from the basic libraries into the user's ones, and b) a typical cell calcula-

tion for a light water lattice were done.

In **Table 4.1** the computer resources are compared between the usages of the PDS files and the datapool system for the above example.

We can find no effective reduction of CPU time, large scale reduction in the elapsed time due to the removal of OPEN and CLOSE processes for each read/write operation in the PDS file, intolerable increase of I/O access count which nearly reaches the limit of the normal run, and increase of core storage requirement such as degrades the priority by one rank. The number of I/O access might be reduced by order of 20–30% by enlarging the logical record length which is now set to 300 words throughout all files, but this problem derives mainly from the subdivision of node which increases I/O accesses. This fact shows us that the design of node should be decided not only on physical clearness but also on the I/O frequency. The increase of core storage requirement comes from the inevitable addition of the managing software of the datapool system instead of the simple read/write subroutine for the PDS file.

In **Table 4.2** the requirements for disk file's are compared. We find certain increase of storage in any file which might be diminished by the use of optimized record lengths.

We have not yet discussed about the effective microscopic cross section file which is intended to transmit the cell averaged few group microscopic cross sections to the other modules. The utilization of this file has not realized until a plan is materialized to extend the SRAC to include the cell burn-up calculation where the key (node name) has to include the information to discriminate the nuclide, the location of nuclide (region number or mixture, cell), burn-up history and decay chain scheme. Such a complicated labelling will be accomplished only by the datapool system.

Table 4.1 Computer Resources of PDS Files

	PDS File	Datapool
CPU Time	90 sec.	85 sec.
I/O Access	3236 times	19592 times
Core Storage	740 KB	948 KB

1 KB = 1024 Bytes

Table 4.2 Disk File Requirements

File Name	PDS	Datapool
Basic fast library	68 Trk	72 Trk
Basic thermal libr.	265	294
Basic resonance libr.	270	350
User test libr.	21	39
User thermal libr.	70	86
Macro X-section libr.	17	23
Flux	2	16
Micro effective libr.	–	–

1 Trk = 19 KB

4.1.5 Conclusion

The conversion of data storage system from the PDS to the datapool system has been completed for the present version of the SRAC code system. The improvement in performance from the view point of computer resources and disk file requirements have not been done yet but we can except the application of the datapool system in the complicated data arrangements in a core burn-up calculation under planning.

(Keiichiro TUCHIHASHI, Div. of Reactor Engineering)

4.2 Development of the SPLPACK Data Plotting System for Transient Analysis Codes and Transient Experiments

4.2.1 Introduction

A number of computer codes have been developed in the field of reactor safety analysis research for the purpose of predicting the behavior of reactor or a component of reactor during accident conditions. When we use this kind of transient analysis codes, we usually display the calculated results in two dimensional graphs of time versus variables. Drawing graphs is, actually, an indispensable process to get quick and clear understanding of the results. Therefore we can expect that good graphics softwares may improve the efficiency of code development activities.

SPLPACK[32] is a general tool, written mostly in FORTRAN-IV, applicable to plotting the results of transient analysis codes and/or system transient experiments. Two major benefits of its utilization are as follows:

1) We can save the cost of plotter program development.

2) We can easily compare results of one code with those of the other codes or with experimental data.

The latter is very important when we evaluate the performance of a new analysis code by comparison with the other codes or experiments.

In the SPLPACK system, plotting is performed by a program SPLPLOT, which is independent of user's analysis codes. The wide applicability of the SPLPLOT program was realised by the utilization of data base with a standardized format as the interface between user's analysis codes and this plotter program.

The first version of this system, SPLPACK-1, has been incorporated in six computer codes developed at JAERI for the LWR safety analysis, and has been effectively used in the development of these codes as described in Section 4.2.6. From the experience of these applications, following two improvements have appeared to be most desirable.

1) Extension of data processing function of the plotter program

Data processing is required for many purposes. For an example, when we compare calculated results with experimental data, this comparison requires various kinds of data transformation, because measured physical quantities are usually not identical to the calculated variables. Another purpose is the application to a systematic usage of computer codes. When we use calculated results of one code as input to another, some data processing may become necessary.

These two kinds of data processing is expected to be effectively performed by a coupled system of data plotting and data processing functions.

2) Improvement of interactive usage option

For obtaining high quality figures as quickly as possible and for effective usage of data processing functions, improvement of the interactive usage option of the plotter program is desirable.

Above two improvements have been successfully realized in the second version of the system, SPLPACK-2, by the use of new programming techniques, the dynamic link and the IPF (Interactive Programming Facility), supported by the FACOM M-200 computer system.

The variety of data processing functions which can be implemented in one program is limited by available core memory size and this limit is relatively serious when the program is operated on the TSS (timesharing system). This problem has been solved by the use of the dynamic link function. The interactive usage option was enhanced by the dynamic link and IPF as in Section 4.2.7.

This chapter describes the features of SPLPACK system and the status of its application as well as the new programming techniques utilized in the second version.

4.2.2 Outline of the SPLPACK System

SPLPACK system was designed according to the following principles.

1) Plotter program must be independent of user's analysis codes. Therefore the calculated results should be transmitted to the plotter program via a data base of a standardized format.

2) The format of the data base must be sufficiently generalized.

3) Tools must be provided to minimize program modification of the user's code for outputting calculated results.

4) The usage of the plotter program must be as easy as possible.

The following sections describe how these principles have been materialized in the SPLPACK system.

The basic structure of the SPLPACK system is illustrated in **Fig. 4.3**. In this figure, data flows are shown by arrows. The calculated results of the user's code are edited and stored in a data base of a standardized format called the SPL format. A subroutine package SPLEDIT is provided for data editing. With the use of this package, the user can easily incorporate the SPLPACK system into his code. The program SPLPLOT retrieves data from the data base by SPLEDIT and draws graphs according to user's requests.

SPLPLOT is operational on both batch and timesharing systems. The input terminal for the requests may be a card reader or a keyboard, and the output terminal may be a XY-plotter, a laser printer, or a conversational graphics terminal such as the Tektronix terminal.

Brief descriptions of the SPL format, the SPLEDIT package, and the SPLPLOT program are given in Section 4.2.3, 4.2.4 and 4.2.5 respectively.

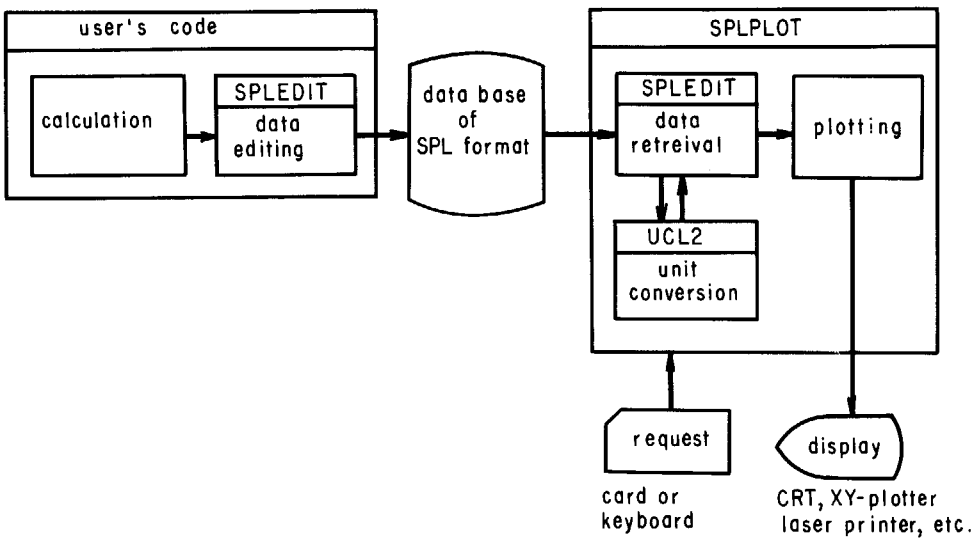


Fig. 4.3 Basic Structure of SPLPACK-1 System

4.2.3 Standard Format of Data Base

The structure of the SPL format is shown in **Fig. 4.4**. The data base is a sequential file in the binary mode, and has following four parts:

data base label part : data base name and format identifiers

comment part : general description of data in the data base

FILE NAME PART

Record 1 : Indicator of File Name Part (2)

MARK

MARK (2), A = '$STDDATA'

Record 2 : File Name and Format Description (25)

FILNM	CFMT	LFMT	DFMT	FNDMY

FILNH (2), A = file name
CFMT (1), A = comment format ID
LEMT (1), A = variable label format ID
DFMT (1), A = date format ID
FNDMY (20) = a dummy space

COMMENT PART

Record 3 : Indicator of Comment Part (2)

MARX

MARK (2), A = '$COMMENT'

Record 4 : Description of File (100)

PRGNM	DAY	TITLE	CMDMY

PRGNM (4), A = name of mother program
DAY (2), A = date of data set generation
TITLE (44), A = description of data set
CMDMY (50) = a dummy space

VARIABLE LABEL PART

Record 5 : Indicator of Variable Label Part (2)

MARK

MARK (2), A = '$LABEL

Record 6 : General Information (3)

USYS	IDBL	NVMAX

USYS (1), A = unit system ID
IDBL (1), 1 = data precision ID
(1 = single, 2 = double)
NVMAX (1), I= number of variables

Note :
All the records are written using Non-Formatted WRITE statements.

Record 7 : Label of Each Variable (50)
This Record is repeated NVMAX times.

VARID	CAPTN	UNTNM	UNID	IOIM	DMNM	JDBL	LAD	LDMY

VARID (2), A = variable name for identification
CAPTN (10), A = caption
UNTNM (10), A = physical unit
UNID (1), A = ID of physical dimension
NDIM (1), I = number of subscripts (NDIM<10)
IDIM (10), I = dimension of subscripted variable
DMNM (10), A = name of subscripts
JDBL (1), I = data precision ID
(1 = single, 2 = double)
LAD (1), I = location in the data record
LDMY (4) = a dummy space

Record 8 : Miscellaneous Information (20)

NEMAX	NDWRD	NSTP	STPID	INDVAR	LEDMY

NEMAX (1), I = total number of values in a data record
NDWRD (1), I = length of a data record in words
NSTP (1), I = number of data records in data set
STPID (1), A = name for counting the steps
(Default is 'STEP'.)
INDVAR (2), A = ID name of the independent variable
(Defauit is 'TIME'.)
LEDMY (14) = a dummy space

DATA PART

Record 9 : indicator of Data Part (2)

MARK

MARK (2), A = '$DATA'

Record 10 : Data of One Step (NDWRD)
If NSTP≠0 this record is repeated NST times.
If NSTP=0, it indicates that number of steps is not known.

DATA

DATA (NDWRD), R = values

Nomenciature :
'xxx' = a character string
(xxx) = length of an array in words
I, A, and R attached to array names indicate varible type.
I = integer
R = real number
A = character

Fig. 4.4 Structure of SPL Format

variable label part : description of each variable in the data base
data part : numerical data

The numerical data in the data part are the results of calculation or experiment. Since results of transient analysis codes are usually produced time step by time step, each record in this part contains the values of the variables per one time step. Type of the variable to be edited can be an array of up to 3 dimensions of a single or double precision real number. Integer and complex variables are not allowed.

4.2.4 SPLEDIT, a Subroutine Package for Data Base Management

A subroutine package SPLEDIT was prepared to assist users in editing, storing, and retrieving data of the SPL format. **Figure 4.5** shows how it is used in a transient analysis code. **Figure 4.5** is an example of FORTRAN subroutine that performs the data editing process of the user's code (see **Fig. 4.3**). In **Fig. 4.5**, variables that will be edited are TIME, X, DXDT, D2XDT2 in COMMON block/A/. Since all SPLEDIT subroutines have a letter '$' at the top of their names, they can be easily discriminated from the user's subroutines. The functions of the subroutines referred to here are:

– $SETWF specifies FORTRAN Unit number of the data base.

```
      SUBROUTINE EDIT
      COMMON /A/ TIME,DELT,X(2),DXDT(2),D2XDT2(2)
      COMMON /$WAREA/ MEMORY,IDUM,IA(5000)
      DATA ISTART /0/
      IF ( ISTART.NE.0 ) GO TO 10
C DATA SET INITIALIZATION
      ISTART = 1
      MEMORY = 5000
      CALL $SETWF (11,IER)
      CALL $FILNM ('TESTDATA',8)
      CALL $PGMNM ('TEST PROGRAM',12)
      CALL $TITST ('CALCULATION FOR TEST OF SPLPACK',31)
      CALL $PRCSN (1,1)
      CALL $UNSYS ('MKSA','MKSA')
      CALL $LBSX0 (TIME,'TIME ','TIME',' ',' ',1,1,
     - 'TIME',4)
      CALL $LBSX1 (X ,'L ','LNGT',' ',' ',1,1,
     - 'ELEVATION',9,2,'NODE')
      CALL $LBSX1 (DXDT,'V ','VELC',' ',' ',1,1,
     - 'VELOCITY',8,2,'NODE')
      CALL $LBSX1 (D2XDT2,'A ','ACCL',' ',' ',1,1,
     - 'ACCELERATION',12,2,'NODE')
      CALL $WLABL(1,IER)
C DATA SET INITIALIZATION END
   10 CONTINUE
      CALL $WDATA (IER)
      RETURN
      END
```

Fig. 4.5 An Example of Data Editing by SPLEDIT Package

- $FILNM registers file name.
- $PGMNM registers name of the user's code.
- $TITST registers the calculation title.
- $PRCSN specifies lengths of a word (precision) in the user's program and in the data base, respectively.
- $UNSYS specifies the unit systems used by the program and by the data base, respectively.
- $LBSX0, and $LBSX1 register variable labels. These routines also register addresses of the variables. Since a complete label of one variable is registered by one CALL $LBSX.. statement, it is easy to add or delete variables to be edited.
- $WLABL edits label information registered on the core memory by subroutines listed above, and writes them on the data base.
- $WDATA edits calculated results of the registered variables, and writes them on the data base.

In the data retrieval process shown in **Fig. 4.3**, SPLEDIT performs following two types of data transformations.

1) Rearrangement of data

Since data in the data base are arranged time-step-wise, they must be rearranged before plotting the history of one variable.

2) Unit conversion

Computer codes have been written in various unit systems. Therefore unit conversion is an inevitable process when a computer code utilizes data produced by the other codes. SPLEDIT automatically does unit conversion in data editing and data retrieval according to the user's specification. This conversion is performed with the use of the UCL2 subroutine package written by Abe[33]. This subroutine package has, among others, a function of

analyzing a character string of the given unit and generating a new character string of the unit in the required unit system as well as the unit conversion factor.

4.2.5 SPLPLOT, a Program for Drawing Graphs

SPLPLOT draws graphs of data in the SPL-formatted data bases. This program was designed on the basis of the LFTPLT7 program by Soda[34] and the ROSAS3 program by Sobajima[35], which will appear in Section 4.2.6 again.

The major differences between SPLPLOT and the other two programs are:

1) Applicability of SPLPLOT has been greatly extended by the use of the SPL format for data storage.

2) The unit conversion function was generalized by the use of the UCL2 package.

For the user's convenience, SPLPLOT has following functions:

- Unit of the variable can be automatically converted to the requested unit or unit in the requested unit system.
- Linear and logarithmic scaling can be used.
- Scale range can be automatically adjusted.
- Data from up to 10 data bases can be plotted in one figure.

```
column    ....+....1....+....2....+....
card 1     1
card 2
card 3
card 4     2
card 5
card 6    L           1
card 7
card 8
card 9    L           2
card 10    1
```

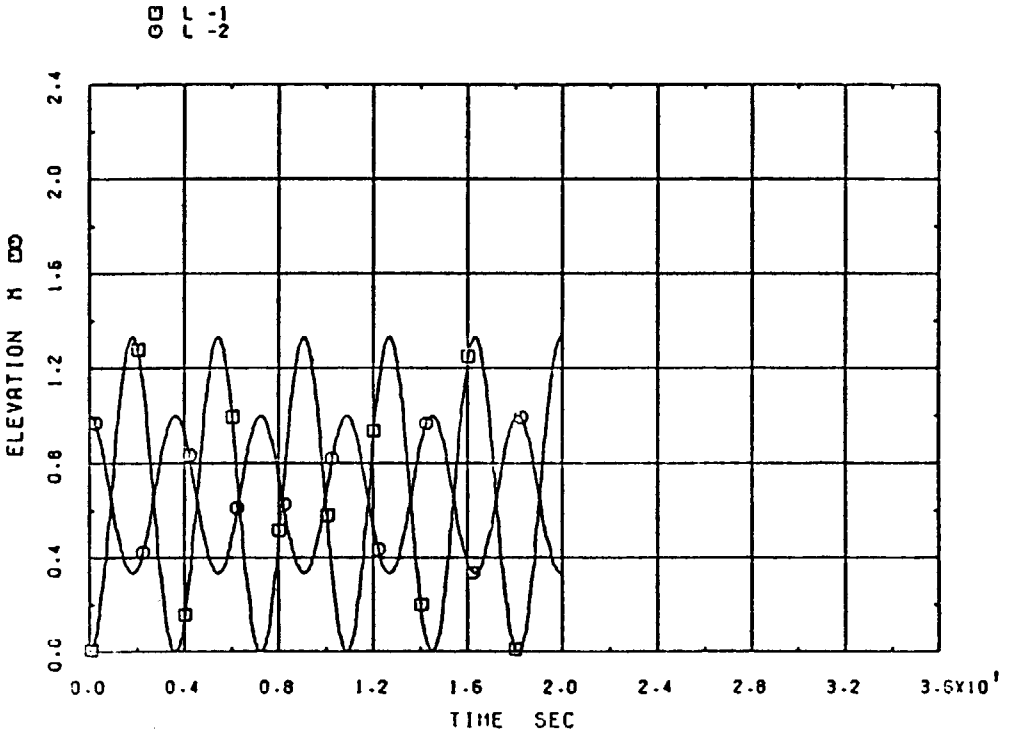


Fig. 4.6 An Example of Request Card and Plot Result

– Up to 10 lines can be plotted in one figure.
– Up to 4 scales can be drawn for Y axis. But only one scale is allowed for X axis.
– Values calculated from data in the data base can be plotted.

The last function, the calculation option, often becomes necessary when we compare results of analysis codes with experimental data. To use this function, the user must write a subroutine to perform the calculation and link it with the load module of SPLPLOT. Since modification of the original source program by individual users should be minimized, the name and arguments of this subroutine are limited to one fixed pattern. Therefore the complexity of the calculation is limited. Section 4.2.7 describes how this limitation has been removed in the second version.

The user is allowed to specify unit system to be used, figure title, axis captions, unit of each scale, figure size, etc., but since SPLPLOT is provided with various default options and the variable label contains information of units and captions, the input cards can be very simple. **Figure 4.6** shows a simple example of input cards together with plot results for it.

4.2.6 Present Status of the SPLPACK Application

SPLPACK has been applied to following six codes developed at JAERI. All of these codes were developed for the performance evaluation of the emergency core cooling system (ECCS) during a loss-of-coolant accident (LOCA) of a LWR. Their names and items that they analyze are:

ALARM-P1[36] ; thermohydraulics during a blowdown phase in a PWR,
ALARM-B2[37] ; thermohydraulics during a blowdown phase in a BWR,
HYDY-B1[38] ; thermohydraulics in a core during a blowdown phase in a BWR,
THYDE-B1[39] ; thermohydraulics during a small-break LOCA in a BWR,
THYDE-B-REFLOOD[39]; thermohydraulics during a reflooding phase within the core shroud of a BWR,
SCORCH-B2[40]; heat-up of a fuel bundle of a BWR.

The program modification of these codes for the incorporation of the SPLEDIT routines took about 2 man-days for each. Although this was the case of a programmer familiar with the SPLPACK system, a similar amount of man-days is expected for a person who starts with reading the manual.

For the performance verification of the above codes, it was desired to compare the calculated results with those of other codes or experimental data. To perform this comparison by SPLPLOT, adaptations were made to access the output data of

1) RELAP4 code[41] for the LWR LOCA analysis,
2) LOFT experiment[42] of PWR LOCA,
3) ROSA-III experiment[43] of BWR LOCA.

The adaptation to RELAP4 was made by providing a tool for conversion of a RELAP output tape into the SPL format. This program was written easily because the data arrangement in the data base was similar.

The output data of the LOFT and ROSA-III are stored, at JAERI, in the data bases of a format designed for the LFTPLT7 or ROSAS3 plot programs. Since the data arrangement in this format was not time-step-wise like the SPL format but variable by variable, subroutines for reading this format were directly incorporated in SPLPLOT.

Thus SPLPLOT can access seven codes and two LOCA experiments at present, and has contributed to the experimental verification of the above LOCA/ECCS analysis codes.

4.2.7 Scope of Future Development

The second version of the system, SPLPACK-2, is now under programming. This version will have various useful capabilities which are not provided in the present version. Some of the major enhancements to be achieved in this version are:

1) Extension of the data processing functions of SPLPLOT,

2) Improvement of an interactive usage option of SPLPLOT,

3) Extension of the data management subroutine package SPLEDIT by adding alternative options of data base formats.

The purpose of the first two improvements has been described in the introduction of this chapter. The third is for the convenience of various users who have different types of data bases. Improvements (1) and (2) are materialized by the use of new techniques, the dynamic link and the IPF, supported by the FACOM M-200 computer system. This section briefly describes the two techniques and how they are utilized in the SPLPLOT-2 program.

(1) Dynamic Link

The function of the dynamic link and the needs of it have been described in Subsection 3.3.1 (b). This function has been fully realized in the present FACOM M-200 system as mentioned in Subsection 3.3.2 (5).

Two options of dynamic link can be selected in the present system.

One is the standard option. This option does not need any modification of the user's code. Only the specifications of dynamic link option at the compilation and at the linkage editing are required of the user.

The other one is linking via an assembler subroutine DLINK. In this method, the part of core memory occupied by a referred module will be released after execution of the module, while the referred modules are left on the core memory in the standard option. Therefore this option is more effective in saving memory size. The second option has another merit. The name of the subroutine to be referred to can be defined during the execution. This is not allowed by a conventional FORTRAN. Next example may help reader's understanding of this function.

FORTRAN statements:

NAME = 'SUB1'

CALL DLINK (NAME, argument 1, argument 2, ...)

is equivalent to:

CALL SUB1 (argument 1, argument 2, ...).

By this method, it has become possible to construct a frame of a code system in which the user can refer his own module (subroutine) without any modification of the original source program of the caller routine.

(2) IPF

As mentioned in Subsection 3.3.2 and 3.3.3, IPF is a part of the operating system of the FACOM M-200 system. By the aid of this system, we can use, during execution of a program written in FORTRAN, COBOL, or PL/I, various useful service softwares developed for the conversational usage of TSS.

A function of IPF used in SPLPLOT-2 is the interruption into a FORTRAN program. By calling a simple interface subroutine, we can interrupt the execution of a program and execute various commands defined on the conversational mode of TSS. This function is used for dynamic allocation of data bases. We can interactively allocate and catalogue a new data base or access old data base that has not been allocated at the beginning of program execution.

This is not allowed by the conventional FORTRAN.

(3) Structure of SPLPLOT-2 and Application of the Dynamic Link and IPF to the Program

The structure of the SPLPLOT-2 program is illustrated in **Fig. 4.7**. This program is operated in an interactive mode. In this program, the subroutines that are kept on the core memory are only those for command interpretation, for control of dynamic link (DLINK) and for data base management (SPLEDIT/UCL2). Subprograms for plotting and data processing are kept on auxiliary memory as a load module library.

The commands given by the user is interpreted by a command analyzer routine according to the information from a command specification data file. The command analyzer determines what subroutine should be executed as well as the arguments required by the subroutine. This information will be passed to DLINK to execute the requested subroutine for plotting or data processing. The output data of these subroutines are stored in the data base of the newly defined format of random access type. This format is more suitable for storing intermediate products of data processing than the format described in Section 4.2.3.

The modules for data processing can be easily added by the user by adding load module of his subroutine in the library and modifying the command specification data file.

The command specification data file containes the necessary information for each command, that is, a name of the subroutine to be executed, the attributes of arguments of the subroutine, and the default values for the arguments. Since this file is written in the card image, it can be easily revised by the use of text editing facilities of TSS.

The load module library need not be a single data file but may be a set of prioritized files. Therefore the user can add his library without modifying the original library.

Thus it can be said that the extensibility of SPLPLOT-2 has been greatly improved by the use of the dynamic link method.

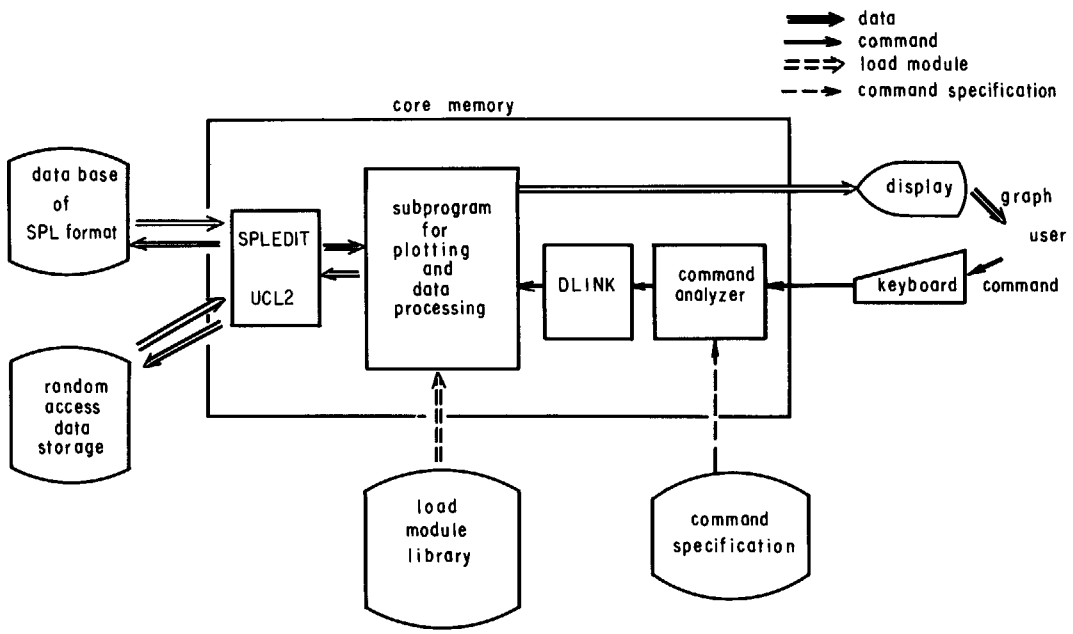


Fig. 4.7 Structure of SPLPLOT-2 Program

4.2.8 Conclusion

SPLPACK is a generally applicable data-plotting tool for transient analysis codes and transient experiments. This system has been applied for plotting of output data of seven computer codes and two experiments, and has contributed to experimental verification of these codes.

The second version of this system, SPLPACK-2, is now under development. By improving the SPLPLOT program in data processing functions and in an interactive usage, this system is expected to become more effective not only for plotting but also for data analysis and data management.

Major conclusions obtained from the development work of this system are as follows:

1) The improvement of the interactive usage option and the data processing functions have been realized by the use of the dynamic link and the IPF of our computer system.

2) The dynamic link is effectively used for improving the extensibility of the SPLPLOT-2 program.

3) The wide applicability of the SPLPACK system is realized by a standardization of the data base format.

4) Unit conversion is an important process of data usage, and the incorporation of the UCL2 package enhanced the effectiveness of the SPLEDIT data editing subroutines.

(Ken MURAMATSU, Div. of Reactor Safety Evaluation)

5. Concluding Remarks

Reviewing the examples of applications in Chapter 4, we can point out some merits and demerits of the method presented in Chapter 3 as follows.

1) In the application of our present datapool to the SRAC code system some merits are observed;

(i) The datapool defines the file structure clearly and the structure prevents confusion about the contents in the file,

(ii) Since the file using our datapool keeps the input/output lists of corresponding Fortran statements, the data in the file may be handled independently of the program,

(iii) A program in a code system may or may not use datapool files, and this encourages us to construct a code system without hesitation, because we can use existing programs in the code system with minimum works.

On the other hand, following demerits are also observed;

(i) Refined data, nodes in a hierarchical structure of datapool file increases input/output operations,

(ii) For a random access file such as the datapool file, it is required to set maximum amount of space before its usage. This leads users to define unused, redundant amount of file space.

2) In the application of the interactive programming facility and dynamic link feature to SPLPLOT2 code system, following merits are observed;

(i) Interactive programming facility promotes to construct an interactive, flexible code system, because the facility allows users to allocate files, use timesharing commands in the execution time of the code system,

(ii) The dynamic link feature helps to save a main storage space in execution time and this is done with minimum changes of each (sub)program.

On the other hand following demerit is observed;

(i) File space for load modules is bigger than that of ordinary one by factor of one and a half.

3) In general it is the most important result of our present method that by using it a (modular) code system is constructed with the minimum amount of man-months.

About twelve years ago when the subcommittee for modular code system started we had thought that the goal of our activity would be to realize following schemes and functions for nuclear codes:

(1) standardization of nuclear codes and related data libraries,

(2) dynamic selection and linkage of codes at any arbitrary execution point,

(3) interruption of execution at arbitrary time and display of calculated results,

(4) dynamic modification and re-execution of codes,

(5) graphical and animated display of calculated results.

We have seen that our JAERI users are now provided with computer features which cover the items (2), (3), and (4). We have shown in this report some examples of their applications. The user is not forced to use all of these features but is recommended to use them selectively according to their own needs. In the application of the datapool to the SRAC code system, we can see the fact that the datapool provides users with the capability of displaying data interactively, and that it promotes standardization of data libraries. In the

application of the interactive subsystem control IPF and the dynamic link to the SPLPACK code system, we can see that the features provide users with the dynamic selection and linkage of codes at execution time, the interruption of execution at arbitrary time, the dynamic modifications and re-executions of codes. The draft of this report itself was written by the document editor ATF. However, as described in Section 3.4, these features need some extensions and improvements and some of the tasks are being done at both JAERI and the computer manufacturer. As for the item (1), our Establishment has organized a working committee to standardize data libraries and nuclear codes. The SRAC code system described in this report is one of the products of activities of the working committee.

Thus it can be said that the subcommittee for modular programming has devoted to construct a framework of code system while the working committee is making its contents.

The code systems described or cited in the Chapter 4 are not all of the systems in JAERI. Big code systems have been developed at JAERI without using the features explained in this report[44]. Our present method, however, will become common in the near future.

(Kiyoshi ASAI, Computing Center and Satoru KATSURAGI, Div. of Reactor Safety)

Acknowledgements

We, the members of the Modular Programming Subcommittee of Nuclear Code Committee, Japan Atomic Energy Research Institute wish to express our sincere thanks to following managers and engineers of the computer manufacturer, Fujitsu, Ltd. They recognized the necessity and value of the softwares proposed by us and realized them.

Akira Tabata, General Manager, Software Division, Computer Systems
Shigeru Suzuki, Ass. General Manager, Develop. Div., Computer Systems
Shozo Taguchi, Manager, Language Processor Development Department
Kuniaki Noguchi, Manager, Language Processor Section 2
Yoshiyuki Tanakura, same as above.
Toshinori Aso, same as above.
Masakazu Kobayashi, same as above.
Kazuhiko Takahashi, same as above.
Tomio Suzuki, Scientific Systems Section, Sci. & Energy Sys. Develop. Dept.
Yutaka Yamagishi, same as above.
Naoyuki Ishikawa, same as above.
Kenitsu Naraoka, Nuclear Energy Section, Sci. & Energy Sys. Develop. Dept.
Kazuhisa Kihara, same as above.
Yukio Narumi, same as above.
Tsuneyoshi Nishi, same as above.
Makoto Ikawa, Scientific Systems Section, Sci. & Energy Develop. Dept.

Lastly but not leastly we wish to express our thanks to Mr. T. Hirakawa, Chief and other staffs of Computing Center, JAERI for their continuous support to our works.

References

Note: Reference (19), an introductory book for Multics concept, is not cited in this report.

1) Reilly E.D., Jr. and Turner W.H.: ANL-7050, "The Automation of Reactor Design Calculations at the Knolls Atomic Power Laboratory", Proc. Conf. on Application of Computing Methods to Reactor Problems, Argonne, Illinois, May 17–19, pp. 251–263, (1965).
2) Kopp H.J.: KAPL-M-7372 (Rev. 1) "DATATRAN 2 User's Guide", (Aug. 1977).
3) Toppel B.J. et al.: ANL-7332, "The Argonne Reactor Computation System ARC", (1968).
4) Just L.C. et al.: "Recent Developments and Capabilities in the ARC System", Proc. Conf. on the Effective Use of Computers in the Nuclear Industry, Knoxville, Tennessee, April 21–23, pp. 337–343, (1969).
5) Honeck H.C., Suich, J.E. et al.: "JOSHUA-A Reactor Physics Computational System", ibid., pp. 324–336.
6) Honeck H.C.: DP-1380, "The JOSHUA System", (1975).
7) Watanabe T., Arai K. and Noda T.: "Hitachi Nuclear Codes Control System, NCCS", Proc. Conf. on the Effective Use of Computers in the Nuclear Industry, Knoxville, Tennessee, pp. 313–323, (1969).
8) "Modular Coding Systems for Reactor Calculations", Newsletter of the ENEA Computer Programme Library, No. 11, (March 1971).
9) "The System DOYC-1", ibid., pp. 75–77.
10) "Modular System JCOMPACT", ibid., pp. 77–79.
11) "JAERI Fast Reactor Integrated Computation JFRIC System", ibid., pp. 79–81.
12) Corbato F.J. and Vyssotsky, V.A.: "Introduction and Overview of the Multics System", Proc. Fall Joint Computer Conference, pp. 185–196, (1965).
13) Fano R.M.: "The MAC System: The Computer Utility Approach", IEEE Spectrum, **2**, Jan. pp. 56–64, (1965).
14) Asai K. and Tomiyama, M.: JAERI-M 4762, "GPL-Genken Programming Language", (March 1972) [in Japanese].
15) Tomiyama M., Takigawa Y., Yoshimori M., Ogitsu M. and Asai, K.: JAERI-M 8715, "Datapool; Its Concept and Facilities", (Jan. 1980) [in Japanese].
16) Tomiyama M. and Asai K.: JAERI-M 9719, "SCAN: Structure and Functions of Fortran Analyzer", (Aug. 1981) [in Japanese].
17) "FACOM OS IV/F4 DOCK/FORTRAN 77 Fortran Program Debugger for Display Terminal", 64SP3680, Fujitsu, Ltd., (June 1981) [in Japanese].
18) "FACOM OS IV/F4 IPF: Interactive Programming Facility – User's Manual", 64SP-3520-1, Fujitsu, Ltd., (July 1980) [in Japanese].
19) Ikeda K.: "Structure of Computer Utility – An Antomy of Multics", Shokodo Publ. Co., Tokyo, (June 1974) [in Japanese]
20) Brooks Jr., F.P.: "The Mythical Man-Month: Essays on Software Engineering", Addison-Wesley, Massachusetts.
21) Wirth, N.: "A Programming Language for the 360 Computers", Jour. Assoc. Comp. Machinery, **15**, No. 1, pp. 37–74, (1968).
22) "FACOM TSS SORP: System Output Retrieval Projector – User's Manual", 64SP-

2120-1, Fujitsu, Ltd., (July 1980) [in Japanese].
23) Nishi T., Yamagishi Y. and Asai K.: JAERI-M 9615, "K – System: A Text Editor and Formatter Program", (Aug. 1981) [in Japanese].
24) "FACOM OS IV/F4 ATF: Advanced Text Editor and Formatter – User's Manual", 70AR-8700-1, Fujitsu, Ltd., (May 1981) [in Japanese].
25) "FACOM OS IV/F4 User's Manual", 64SG-1002-1, (July 1981) [in Japanese].
26) Harada H., Kihara K. and Asai K.: JAERI-M 9650, "Dynamic Link: User's Manual", (Aug. 1981) [in Japanese].
27) Tuchihashi K.: "Application of Datapool to Data Storage of SRAC Code System", Chapter 3 in this report.
28) Muramatsu K.: "Development of SPLPACK Data Plotting System", Chapter 3 in this report.
29) Naito Y., Tsuruta S., Matsuura T. and Ouchi T.: JAERI-M 9396, "MGCL-Processor: A Computer Code System for Processing Multigroup Constants Library MGCL", (March 1981).
30) Naito Y. et al.: "Integral Experimental Data Bank System", to be published as JAERI-M report [in Japanese]
31) Akino F.: "Benchmark Tests of SRAC Code System", B26, Proc. of the Annual Meeting of Atomic Energy Society of Japan, (1981) [in Japanese].
32) Muramatsu K. et al.: "User's manual of SPLPACK", to be published as JAERI-M report [in Japanese].
33) Abe K.: JAERI-M 9592, "Manual on Unit Conversion Program Library UCL2", (1981) [in Japanese].
34) Soda K., Wada Y. and Otsubo N.: JAERI-M 7695, "LFTPLT7 – LOFT Plotter Program", (1978) [in Japanese].
35) Sobajima M., Osaki H. and Murata H.: JAERI-M 8499, "Instrumentation and Data Processing for ROSA-III Test", (1979) [in Japanese].
36) Akimoto M., Araya F., Sasaki S. and Sato K.: JAERI-M 8004, "ALARM-P1: A Computer Program for Pressurized Water Reactor Blowdown Analysis", (1978).
37) Akimoto M.: JAERI-M 6968, "ALARM-B1: A Computer Program for Boiling Water Reactor Blowdown Analysis", (1977).
38) Asahi Y.: JAERI-M 6539, "HYDY-B1 Code: Calculation Model for Core Thermal-Hydraulics during a Loss-of-Coolant Accident", (1976).
39) Muramatsu K.: JAERI-M 8119, "Computer Programs, THYDE-B1 for Analysis of Small Break LOCA of a BWR and THYDE-B-REFLOOD for Analysis of Reflood Phase", (1979) [in Japanese].
40) Abe K. and Sato K.: JAERI-M 6678, "SCORCH-B2: A Simulation Code of Reactor Core Heatup Version 2",(1976) [in Japanese].
41) Aerojet Nuclear Company: ANCR-NUREG 1335, "RELAP4/MOD5-A Computer Program for Transient Thermal-Hydraulic Analysis of Nuclear Reactors and Related Systems-User's Manual", (1976).
42) Reeder D.L.: NUREG/CR-0247, TREE-1208, "LOFT Systems and Test Description (5.5-ft Nuclear Core 1 LOCEs", (1978).
43) Anoda Y., Tasaka K., Suzuki M., Koizumi Y. and Shiba M.: JAERI-M 9243, "ROSA-III System Description", (1980).
44) Asaoka T.: "Present Status and Future Plans for Software Development in JAERI", Proc. on Workshop on NEA Data Bank Software, Argonne, Illinois, May 5–6, pp. 85–97, (1980).

Appendix ; An Outline of Datapool System*

1. Introduction

In this summary the authors describe a concept and facilities of a program named datapool.

Generally in scientific computations lengths of data, sequences of input/output operations from and/or to files depend on the input data and calculation processes of programs. When we use a data file it is often needed to scrutinize contents of Fortran programs and their input data which are relevant to the data file. This fact means that most data files have indivisible relations with programs which produced the data files. At JAERI, as in other institutions, many big programs with giant data are being used and the researchers have spent an amount of time to adjust data files and programs to tailor them into their needs.

We will show in the followings that simple extensions of ordinary Fortran input/output operations can resolve this difficulty. The simple extensions also promote the modularity and standardization of data files. The datapool system has been developed to support these extensions.

To meet this purpose, we have defined a characteristic of the datapool file as a hierarchical, tree structured naming, retention of data and their attributes, and retention of comments on the data. We also defined that utility programs for the datapool file processing should have functions of easy retrieval of data and attributes, display, modification, deletion, maintenance and management.

In addition to the above, we have put a restriction that processors of the datapool system should be entirely written in Fortran language.

The schematic view of the datapool system is shown in **Fig. 1A**.

2. Facilities of Datapool

2.1 Preprocessor

We have defined formats of input/output statements of the datapool file and have provided a preprocessor (block no. 1 **Fig. 1A**) which expands the input/output statements to ordinary Fortran statements. By this processor users can use the datapool file with the minimum efforts of translation of their programs.

The processor reads program source cards and when it finds statements with C and J characters in column 1 and 2 (henceforth we call the statements as CJ-statements), it expands the statements to ordinary Fortran statements.

An example of expansion is shown in **Fig. 2A**. As is shown in the example, the output statement is expanded to a Fortran direct access statement.

---

* Tomiyama M., Takigawa Y., Yoshimori M., Ogitsu M. and Asai K.: JAERI-M 8715, "Datapool; Its Concept and Facilities", (Jan. 1980).

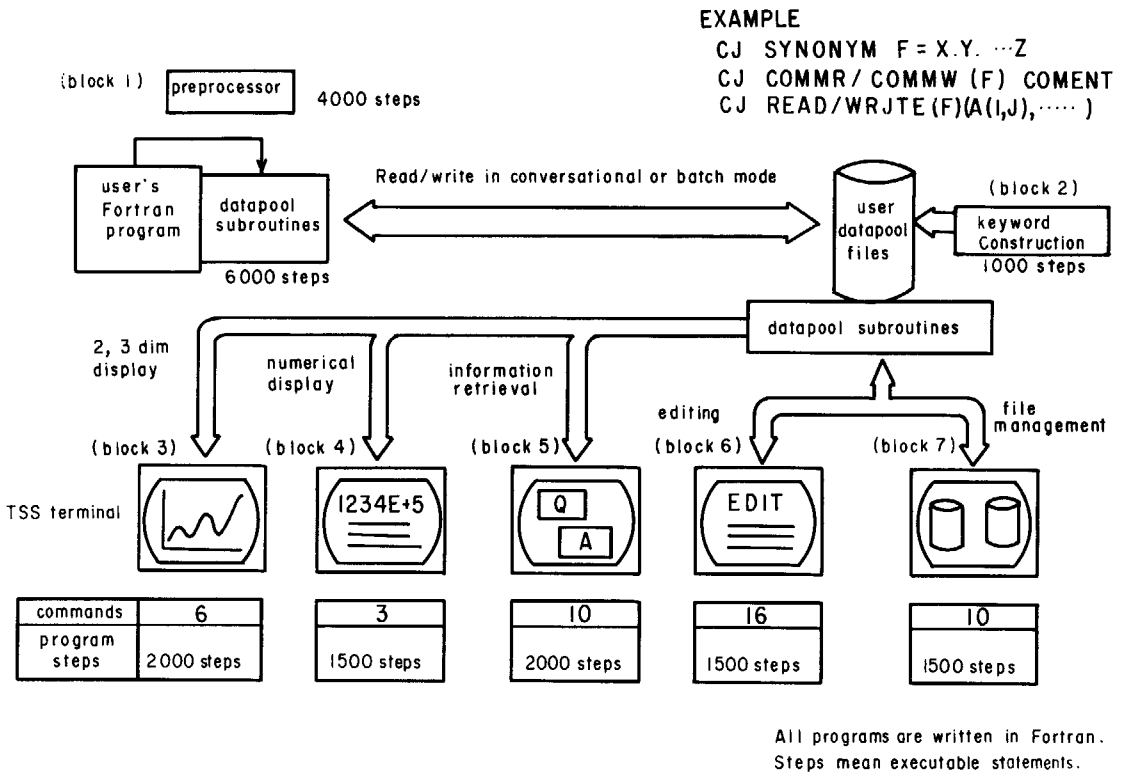


Fig. 1A Schematic View of Datapool System

```
      SUBROUTINE DPW(A,B,C,D,X,N)
      DOUBLE PRECISION A,B,C,D
      DIMENSION X(100)
CJ    WRITE(A.B.C.D.ERR=200) (X(I),I=1,N)
      RETURN
  200 STOP
      END
```

⇩

```
      SUBROUTINE DPW(A,B,C,D,X,N)
      DOUBLE PRECISION A,B,C,D
      DIMENSION X(100)
C=====================================================================
CJ    WRITE(A.B.C.D,ERR=200) (X(I),I=1,N)
C=====================================================================
      ILNGTH=(1*(N-1+1)+0)
      CALL QOPEN(4,2,ILNGTH,IUNIT,IRECRD,0,4,1,0,A,1,0,B,1,0,C,1,0,D,1,&
     1200)
      CALL QWRITE(IRETCD,0,12,'(X(I),I=1,N)',4,510,100,400,N,1,&200)
      WRITE(IUNIT,IRECRD,ERR=200)(X(I),I=1,N)
      CALL QCLOSE(IRETCD)
C=====================================================================
      RETURN
  200 STOP
      END
```

Fig. 2A Expansion of CJ-statement

2.2 Input/Output Access Method

The datapool processor accesses to the data by the following method.

(1) Datapool processor stops search of a data name when it encounters a data name consisting of blank characters, or when it reaches a data name at the end of the hierarchical, tree structured file. Using this feature users can access every data name of datapool files by a single input/output statement.

For example, the following statement

```
CJ   WRITE (A.B.C. . . . .X) (DATA (I), I = 1, N)
```

is equivalent to a statement

```
CJ   WRITE (A.B) (DATA (I), I = 1, N)
```

if all variables C, . . . ,X for data names contain only blank characters.

(2) Users can search data names vertically or horizontally along with the tree structure of a datapool file.

In the following example

```
CJ   POINT (A.B.C, MAXD, ND, DNAME, NRC)
CJ   NEXTV
CJ   READ (*, ERR = 123) U, V
CJ   NEXTH,
```

the POINT statement points a data name to initialize searching. In this case the data name is A.B.C. By the NEXTV statement the pointer moves to one more lower level and a first data name in that level, and the data name of this position is set in the array variable DNAME. By the READ (* . . . statement the user can read the data of this data name. The NEXTH statement moves the pointer to the right, horizontally, and the data name of that position is set in the variable DNAME.

2.3 Retention of Input/Output List

Datapool processor stores the output list including attributes of variables appeared in the list into the datapool file (**Fig. 3A**). Usually it is stored automatically, but when the user wants to reduce number of directories in a file, he can omit the operation by specifying an appropriate value of the priority parameter in the corresponding WRITE statement.

When a new output list is written, no validity check of new and old one is done and the old list is always replaced by new one.

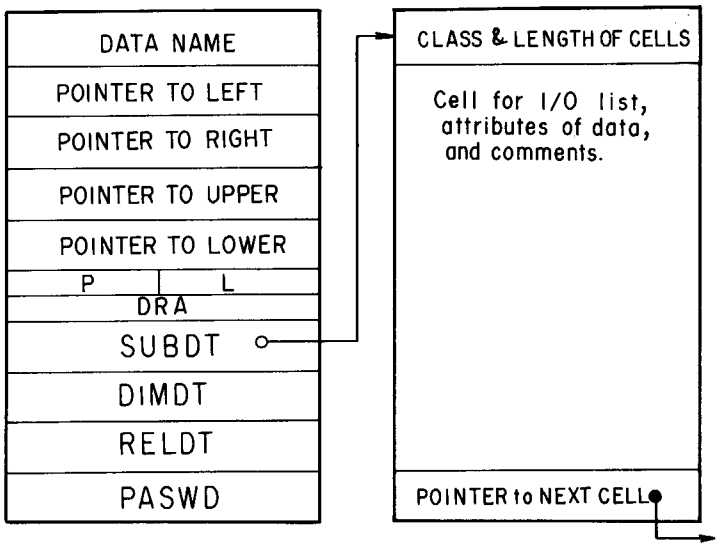


Fig. 3A Directory Cells for Data and Data Attributes

2.4 Comment

Users of the datapool can set comments on data in a batch or timesharing mode. As is shown in **Fig. 3A**, cells for comments are linked to the directory of the data. The comments are extensible or reducible in a unit of 20 words or 80 bytes and the user can edit the comments using a timesharing terminal. When the user has set the format of comments to a fixed form, he can construct a simple information retrieval system for the datapool files.

2.5 Information Retrieval

The datapool system has a simple information retrieval facility for data stored in the datapool files (block no. 5, **Fig. 1A**). The information retrieval facility consists of two programs, one is a program which makes keyword tables for retrieval and the other is an interactive program which retrieves informations in a timesharing mode.

The retrieval is induced by a logical expression which is made input from a TSS terminal. The operands of the expression consist of keywords or numbers representing keywords. An example is shown in **Fig. 4A**.

The number representing a keyword in the expression of **Fig. 4A** is assigned by the program for keyword table construction (block no. 2, **Fig. 1A**). This information retrieval system is for a considerably small set of keywords about up to 2000. The user can define big amount of data and data names irrelevant to the restriction on number of keywords.

```
KEY(P)
>PADE*APPROXIMANT +3
 ABUE      ACRS      ACRT      ACWF      ACWG
LIST(P)
>ACRS
**V:09,1975,46-50              C:ACRS,ICL1430,343,EBCDIC
T:A SUBROUTINE AND PROCEDURE FOR THE RAPID CALCULATION OF SIMPLE OFF-
  DIAGONAL RATIONAL APPROXIMANTS
A:P.R.G.MORRIS,D.E.ROBERTS
K:GENERAL,RATIONAL,APPROXIMANT,CHISHOLM,SIMPLE OFF-DIAGONAL,PRONG,
  PERTURBATION SERIES,PADE
AC:PROGRAM ACRS IS A FORTRAN VERSION OF SODS
LIST(P)
>#END
```

Fig. 4A Example of Information Retrieval

2.6 Graphical Display of Data

The user can display his data in datapool files on a graphical terminal (block no. 3, **Fig. 1A**). Any variables in the Fortran I/O list can be displayed in two dimensional or three dimensional mode. For display of an array variable, the user can specify starting, ending, and incremental values. He can also specify more than one y axes, labels and titles.

In case of two dimensional display, up to 2350 points can be specified for x and y axes, respectively. The Tektronix 4000 series terminals are assumed as output devices. An example is shown in **Fig. 5A**.

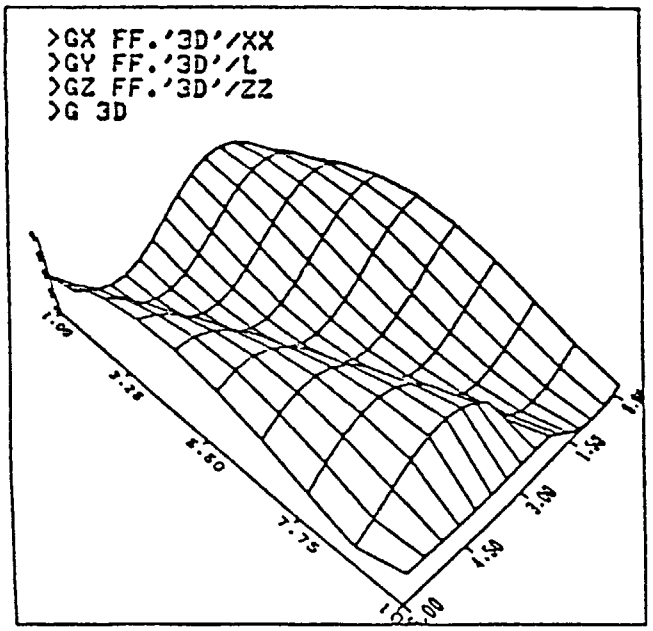


Fig. 5A Example of Graphical Display

2.7 Numerical Display of Data

The user can numerically display data stored in datapool files (block no. 4, **Fig. 1A**). As is the case of graphical display, it is possible to specify arbitrary variables in the Fortran I/O list corresponding to the data. The display operation is executed by interpreting the corresponding I/O list and attributes of variables which are written with the data. It is possible to display variables of integer, real, double precisioned real, and imaginary number of these types. The corresponding I/O list is displayed at the same time. An example is shown in **Fig. 6A**.

```
>XYA= 'XYALL'
>LISTD FF.XYA/XX
*** I/O LIST ***
Y2,N,(YY(I),I=1,N),I1,I2,(XX(J),J=I1,I2)
*** CONTROL VARIABLES ***
*** SYMBOLS ***
XX
*** DATA ***
 0.0             0.4000000E-01  0.8000000E-01
 0.3200000E+00   0.3600000E+00  0.4000000E+00
 0.6400000E+00   0.6800000E+00  0.7200000E+00
 0.9600000E+00   0.1000000E+01  0.1040000E+01
```

Fig. 6A Example of Numerical Display

2.8 Input, Display and Modification of Comments.

The user can store data with comments (block no. 6, **Fig. 1A**). Comments can be written into datapool files in the time when a user program is executed. The user can also input, display and modify comments in a conversational mode. An example is shown in **Fig. 7A**.

```
00010       PROCEDURE TESTPRO
00020       ATTACH F01,J9131.TESTPOOL,LRECL=80/U,MAXRCD=500
00030       ON ERROR STOP 'ERROR STOP'
00040       SYNONYM FILE='J9131'.'TESTPOOL'.'COMMENT'.#NN
00050       NN = 0
00060   100 NN = NN + 1
00070       CAREA 80,A(1)
00080       CW FILE
00090       ON (NN.LT.9) GOTO 100
00100       MANUAL  'RETURN TO MANUAL MODE'
00110       END

#.DPSRUN PROC=TESTPRO

>RUN TESTPRO
.START ***TESTPRO *** PROGRAM
.START TEXT MODE
#....*....1....*....2....*....3....*....4....*....5....*....6.
>TESTCOMMENT CREATED. DIMENSIONAL CELL #1
>SEND
.END TEXT MODE
```

Fig. 7A Example of Comment and Catalogued Procedure

2.9 File Management and Command Procedure

The user can change names of data, delete, move, copy, and compare data.

The datapool system has a simple command processor to enable fixed form operations easily (block no. 7, **Fig. 1A**). The commands are shown in **Table 1A**.

Table 1A TSS commands

Datapool facilities for TSS use

Commands

1. STOP command
2. PROCEDURE command
3. MANUAL command
4. GOTO command
5. END command
6. RUN command
7. ATTACH command
8. ALLOCATE command
9. SYNONYM command
10. Assignment statement
11. CATLIST command
12. LIST command
13. LISTD command
14. COMPARE command
15. MOVE command
16. COPY command
17. DELETE command
18. RENAME command
19. CHAP command
20. ON command (1)
21. ON command (2)
22. ON command (3)
23. COMMAREA command
24. DISPLAY command
25. CEDIT command
26. COMMR command
27. COMMW command
28. COMMWA command
29. COMMD command
30. Graphic initialization command
31. Graphic control commands
32. Graphic parameter setting commands

2.10 Maintenance of Datapool File

The datapool system provides following commands for maintenance of datapool files.

(1) RECOVER Command

In case of a user program abortion when it is writing data into a datapool file, the datapool file is not closed normally and the control table of the file remains in invalid status. For this file it is impossible to continue read or write operation. The RECOVER command remedies the control table of the file and makes the file reusable.

(2) CONDENSE Command

When data names are deleted, directories and areas for the table remain in the datapool file as garbages. The CONDENSE command clears them and make them reusable for the file.

(3) SORT Command

The hierarchical tree structure of a datapool file is made by the order the corresponding data have been stored. The SORT command reorder them in ascending or descending order.

3. Internal Structure of Datapool Processor

3.1 Preprocessor

The preprocessor detects CJ-statements, where C and J are punched in the 1-st and 2-nd columns, respectively in a user's Fortran source program, and expands the CJ-statements for datapool input/output operations to ordinary Fortran statements. The number of parameters in a subroutine call of the expanded statement depends on the length of data name.

The preprocessor itself has been coded also in Fortran (block no. 1, **Fig. 1A**). It has size of 4000 executable statements.

3.2 Datapool File

The datapool file is essentially a Fortran random access file. Hence the user can use arbitrary number of files in his program. Each datapool file is divided by an appropriate block size specified by the user and in its top portion it contains informations neccessary for the datapool file control.

There are two main informations for datapool file control, one is a DPCTL which describes attributes of the file, and the other is a set of directories which defines attributes of data. For each datapool file there exists one file control table DPCTL and it consists of 50 words (or 200 bytes). The contents of DPCTL is shown in **Table 2A**.

Table 2A DPCTL . . . Datapool Control Table

1.	'19'	18.		35.	POINTER FOR
2.	'31'	19.		36.	SHARABLE USERS
3.	DATA SET NAME	20.		37.	HASH TABLE SIZE
4.		21.		38.	PTR FOR HASH TABLE
5.		22.		39.	HASH CLUSTER SIZE
6.		23.	RECORD SIZE	40.	PTR FOR HASH CLUSTER
7.	CREATOR'S NAME	24.	MAX RECORDS	41.	MAX CASE NO.
8.		25.	USED RECORDS	42.	CURRENT CASE NO.
9.	CREATION DATE	26.	POINTER FOR	43.	POINTER FOR
10.		27.	UNUSED RECORDS	44.	CASE NO. ROOT
11.	EXPIRATION DATE	28.		45.	
12.		29.	MAX DIRECTORIES	46.	
13.	PASS WORD	30.	USED DIRECTORIES	47.	
14.		31.	POINTER FOR	48.	BYTE OR WORD
15.	COMPUTER NAME	32.	DIRECTORY ROOT	49.	
16.	DD SET DEVICE NAME	33.	POINTER FOR	50.	
17.		34.	UNUSED DIRECTORIES		

3.3 Directory

Each directory is a cell of 20 words (or 80 bytes) and contains data name, attributes, or comments. The user should specify a neccessary number of directories when he creates the file to store the informations mentioned above.

In **Fig. 3A**, each symbol represents the following meaning.

(1) P means a priority value specified by the user and the value is used to protect the data from destruction in writing or is used as a prohibition bit for display.

(2) L represents the length, i.e., number of blocks of the data. When a new data is written destroying the old one, the datapool processor compares the lengths of the new and old ones. If the old L is smaller than the new, the processor provides continuous blocks and the new data is written in the blocks. If the L value of the old is larger than the new one, the new data overrides the old data. In any case the value of L is updated.

(3) DRA is a starting address, i.e., a starting block number of data on a disk file.

(4) SUBDT is a pointer for the first subsidary cell which contains I/O list and/or comments.

(5) DIMDT is a pointer for one dimensional addressing of data. Each cell pointed by this or consequent pointer contains a subscript which is used to identify a one dimensionally subscripted data. By this function the user can make use of random and/or sequential access methods to a datapool file. An example of subscripted data is shown below.

```
      DO  10 I = J, K, M
CJ    WRITE (A.B. #I) I/O list
   10 CONTINUE
```

In the above example, data names A.B and A.B. #I point to different data respectively and any value of I is valid if it is a positive integer.

(6) RELDT is a pointer to a directory which has a relation with the directory containing this RELDT. Since the datapool file has a hierarchical tree structure, it provides no relation to connect nodes branching separately. RELDT is used to cure this defect.

(7) PASWD is a password for this data name.

3.4 Access to Data Name

It is desirable to make the directories resident in a main memory to get quick access to data, but in practical use it is often required to provide several ten thousands of directories. To solve this problem, the datapool processor has LRU (Least Recently Used) table, DT (Directory) table and DT buffer.

The LRU and DT tables consist of 30 cells respectively and are used by all datapool files in a user program. Each directory in the DT table has extra five pointers in addition to a copy of the directory of datapool file in a disk memory. The LRU table and DT table are shown in **Figs. 8A** and **9A**, respectively.

The additive five pointers of DT table point to cells in the DT table and by these pointers the processor retains local structures of the corresponding datapool file.

The DT buffer which is prepared for each datapool file according to a specification by the user as a preprocessor option contains one block of directory portion of the datapool file.

The i-th cell of LRU table corresponds to the i-th cell of DT table and these cells are managed by the LRU method.

The method of management is as follows.

(1) The Maximum reference number 30 is given to a node which is referenced most recently. Values of reference counters of other nodes are reduced by one. Counters with zero value remain unchanged.

(2) When a node is referenced which is nonexistent in the LRU table, a node of the minimum value of reference counter is deleted from the LRU and DT tables if the node is not updated while it is in the tables, or the corresponding dictionary of the datapool file is

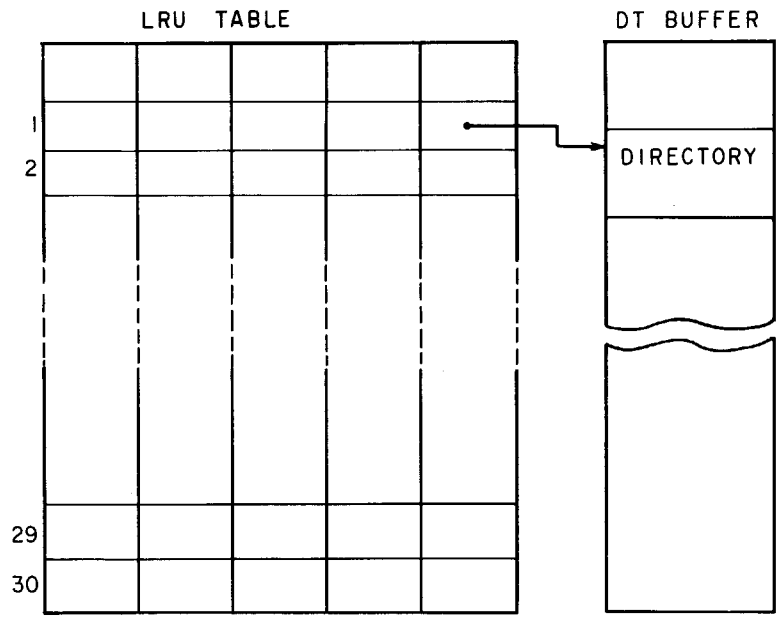


Fig. 8A LRU Table and DT Buffer

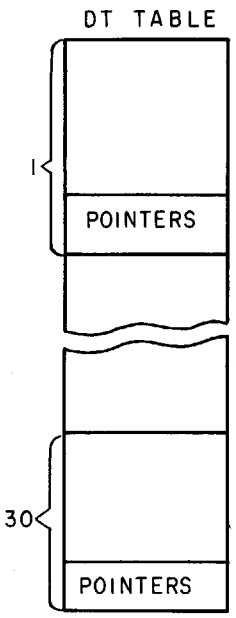


Fig. 9A DT Table

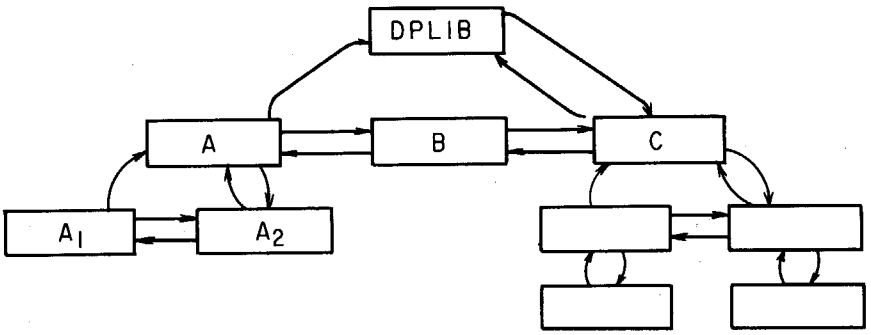


Fig. 10A Example of Datapool

renewed if it is updated. In this replacement algorithm, nodes A, C in **Fig. 10A** have a tendency to remain in the tables, but the nodes A_1, A_2, . . . , B have not. After the discussion with datapool users, we have adopted this method.

3.5 Access to Variable

In this section we will explain by a simple example an access method when both a data name and variables are specified.

Let us suppose that the data, with its data name D and I/O list ((A (I), B (I,J), I = 1, M), J = 1, N), are written in a datapool file.

The command

LISTD D/B, 1, M, 2

causes a display of contents of the variable B with incremental value of 2 to a TSS terminal. In this example 1 is starting, M is ending and 2 is incremental value, respectively.

Reading the data corresponding to ((A (I), B (I, J), I = 1, M), J = 1, N) into the work area and extracting data for ((B (I, J), I = 1, M), J = 1, N), the display routine gives the output to a terminal device. The display routine can treat data of integer, real, double precisioned real, quadruple precisioned real and imaginary numbers of the same types.

In case of graphical display, the numbers mentioned above are automatically converted

to single precisioned real numbers.

The analysis method of the I/O list is similar to that of Fortran compiler, but this is more complex because an interpretive preprocess is required before display.

3.6 Calculation of Logical Expression for Retrieval

(1) Syntax Check of Logical Expression

Checking an input character string from a TSS terminal, the retrieval routine constructs a stack SOP for input operators and a table SKW for keywords. The routine then checks the contents of SOP and SKW according to a syntax in **Table 3A** to test if the operators and keywords are in right order. The routine also checks balancing of left and right parentheses and their order. Each keyword of SKW is checked if it is already registered in the retrieval system. In **Table 3A** symbols *, +, <, or> mean logical product, logical sum, left boundary of input, and right boundary of input, respectively. The symbol X is for a wrong combination and 0 for a valid conbination.

(2) Extraction of Logical Expression

The retrieval routine interpretes the expression when no error is found in the above checks. The expression is executed according to specifications of in **Table 4A**. The symbol S, E, or T means a skipping of operation for left side operator, execution of operator, or a termination of the execution.

Table 3A Syntax Check Table

TBLS (7, 7)

left \ right	1 <	2 +	3 *	4 (	5)	6 >	7 keyword
1. <	X	X	X	O	X	X	O
2. +	X	X	X	O	X	X	O
3. *	X	X	X	O	X	X	O
4. (	X	X	X	O	X	X	O
5.)	X	O	O	X	O	O	X
6. >	X	X	X	X	X	X	X
7. keyword	X	O	O	X	O	O	X

Table 4A Execution Control Table

TBLP (6, 6)

left \ right	1 <	2 +	3 *	4 (	5)	6 >
1. <	X	S	S	S	X	O
2. +	X	E	S	S	E	E
3. *	X	E	E	S	E	E
4. (	X	S	S	S	O	X
5.)	X	X	X	X	X	X
6. >	X	X	X	X	X	X