International Atomic Energy Agency

INDC(CCP)-344
Distr.: G

INDC INTERNATIONAL NUCLEAR DATA COMMITTEE

GRUKON - A PACKAGE OF APPLIED COMPUTER PROGRAMS SYSTEM INPUT AND OPERATING PROCEDURES OF FUNCTIONAL MODULES

V.V. Sinitsa and A.A. Rineiskij
Institute of Physics and Power Engineering
Obninsk, The Russian Federation

Translated by Dr. A. Lorenz for the IAEA

April 1993

IAEA NUCLEAR DATA SECTION, WAGRAMERSTRASSE 5, A-1400 VIENNA

INDC(CCP)-344
Distr.: G

GRUKON - A PACKAGE OF APPLIED COMPUTER PROGRAMS SYSTEM INPUT AND OPERATING PROCEDURES OF FUNCTIONAL MODULES

V.V. Sinitsa and A.A. Rineiskij
Institute of Physics and Power Engineering
Obninsk, The Russian Federation

Translated by Dr. A. Lorenz for the IAEA

April 1993

Abstract

This manual describes a software package for the production of multigroup neutron cross-sections from evaluated nuclear data files. It presents the information necessary for the implementation of the program's modules in the framework of the execution of the program, including: operating procedures of the program, the data input, the macrocommand language, the assignment of the system's procedures. This report also presents the methodology used in the coding of the individual modules: the rules, the syntax, the method of procedures. The report also presents an example of the application of the data processing module.

Reproduced by the IAEA in Austria
April 1993

93-01214

TABLE OF CONTENT

Introduction

The software package for the production of multigroup data "GRUKON" was conceived toward the end of the 1970's as part of the SOKRATOR [1] project whose overall objective was to create an automated system of computer programs for the production of multigroup data for the calculation of nuclear reactors and radiation shielding. This system was to replace the existing set of individual computer programs, which were used in a semi-automated manner to calculate multigroup data using a variety of representations of evaluated neutron data in the form of resonance parameters [2], averaged resonance parameters [3], detailed cross-section dependences [4] and angular [5] and energy [6] distribution of secondary neutrons.

The need for a replacement of the available programs was dictated by a widespread increase in the sophistication of evaluated neutron data files which required a correspondingly high level automated computational tools. In order to satisfy the requirements of multigroup system developers and evaluated data processors, it was necessary to create an extensive system of computer programs to support this effort. After considering the existing requirements, it was decided to replace the extensively used existing programs [7], which had inflexible interfaces and were incompatible with the dynamic character of the data, and develop a new set of computer programs to form an integrated software package with a standardized input [8].

In context of the nature of the problem, the following features of this software package were deemed important: flexibility in the computational control, dynamic adjustment of the data, independence of each program component and potential increase in the functional capabilities of the program. These features constituted the principle guidelines in the development of the programs.

The first stage in the development of this package of applied programs (PAP-GRUKON) consisted in the creation of a specialized systematic input, the development of a macrocommand language, and the adoption of existing algorithms. This stage in the development of the program was concluded with the running of the first version of this program on the BESM-6 computer [9,10]. Although the capabilities of this first version were limited to the production of the simplest type of data, such as group-averaged cross-sections, it was enough to demonstrate the efficiency of the system and justify the precepts used in its development.

The development of the subsequent three versions of this program [11-14], was concentrated on satisfying the more basic requirements of the data processing system. The capabilities of these versions consisted of the calculation of self-shielding factors, subgroup parameters of the resonance structure of cross-sections, and probability transfer matrices. Starting with the 4th version, two independent versions of the PAP-GRUKON program were developed: one for the BESM-6 computer and another for the ES computer (the adaptation of the program package to the ES computer and the extension of its capability to process angular and energy distribution data were written by A.A. Rineiskij).

As the capabilities of this software package grew, so did the number of specialists interested in its development. This was evidently due to the fact that the functions of this package were easily applied to related problems requiring the generation of group data (resulting from the analysis of experimentally measured cross-section dependences such as transmission functions etc.), as well as to the subsequent steps required in the production of evaluated data files and in the preparation of macroscopic nuclear data for specific component materials. It was therefore anticipated that the number of programmers who would be interested to take part in the development of the individual program modules would grow.

The documentation of the GRUKON program that have been released so far have been addresses primarily to the users of this program, namely those involved in the development of multigroup data libraries. This report comprises the first systematic description of the problems which arise in the design of individual modules of the GRUKON package and illustrates its implementation.

1. GENERAL DESCRIPTION OF THE PROGRAM

1.1 Principal Objective of the Program

The package of applied programs, PAP-GRUKON, is designed for the automated calculation of multigroup microscopic neutron data using computer libraries of evaluated data. The PAP-GRUKON package is designed to be used for the solution of problems directly or indirectly applicable to the production of nuclear data - ranging from the analysis of experimentally measured transmission functions to the generation of evaluated nuclear data file.

The basic input required to run this version of the GRUKON program consists of evaluated neutron data files MF= 1,2,3,4,5 written in ENDF/B format. These are:

File MF=1 - special neutron reaction quantities,

File MF=2 - resolved and unresolved resonance parameters,

File MF=3 - tabulated energy dependent cross-sections in the non-resonance region and the resonance background cross-sections,

File MF=4 - parameters of angular distributions of secondary neutrons,

File MF=5 - parameters of energy distributions of secondary neutrons.

The basic output generated by the program consists of:

- group-averaged cross-sections,
- resonance self-shielding factors,
- the temperature dependence of these factors,
- sub-group parameters normalized to the average cross-section including the temperature independent part,
- probability matrices and angular components of the inter-group transfers for elastic and inelastic scattering and (n,2n) and (n,3n) reactions,
- energy production cross-sections.

Additional quantities can be obtained using an expanded output:

- transmission functions and self-indicators measured under conditions of good geometry,

- sensitivity coefficients of the cross-section functions to the average resonance parameters,
- detailed cross-section dependence for given target temperatures.

All of the output data can be printed in the form of annotated tables. Detailed cross-section dependences can be represented in the MF=3 file format of the ENDF/B library as well as graphically.

1.2. Operating Procedure.

The flow diagram of the GRUKON program is shown in Fig.1.1. The main elements of the software package are:

- control procedures and linkage procedures (system input),
- set of functional modules (functional input)
- macrocommand language.

The data used by the program in the solution of a problem consist of basic nuclear cross-section data, so-called "processed" data (PD), and auxiliary data which define the nature of the problem, so-called "control" data (UD); these are stored in the working file of the program, the so-called "library of standard parameters", or BSP library.

A run of this program begins with the reading in of the information defining the problem to be solved, by the so-called "processing program", written in the macrocommand language. The processing program (PP) generates various commands, such as: "read the data from punched cards and store them in the BSP library" (for the input of UD data), or "transfer data from the evaluated data library and store the results in the BSP library", or "output the data to the printer", etc... (See Chapter 2 for the description of the system's commands).

After the command "end of PP" has been read in, the control program executes the PP program. Some of the commands, which do not require input from the BSP library (as in the case of the editing of the catalog) are executed by the control procedures.

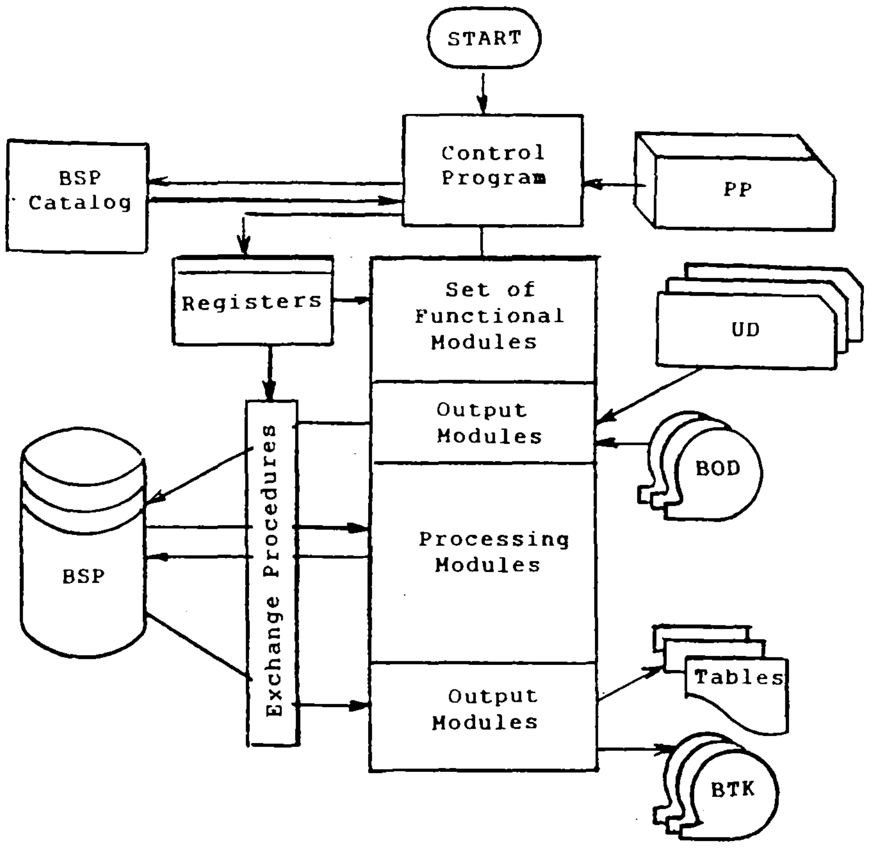

Fig. 1.1 Overall Schematic of the GRUKON Program

PP	- Processing program
UD	- Control data
BOD	- Evaluated data library
BTK	- Multi-group data library
BSP	- Standard parameters library

If a command requires the input from the BSP library (as in the case of input, processing or output data), the control program performs the following tasks: (a) prepares the information consisting of the formats of the input PD and UD data and the address allocation of the processed data in the BSP library, (b) stores the processed information in a special part of the memory (the so-called "system register"), and finally (c) transfers the control to the main part of the functional module.

To summarize, the functions performed by the functional module (excluding modules which do not exercise any control of the data and input-output modules) consists of the following steps:

- retrieval of the control data from the BSP library,
- retrieval of the processed data from the BSP library,
- execution of the data processing,
- storage of the results in the BSP library, and
- return control.

The exchange of data between the BSP library and the functional modules is performed with the help of special system procedures, consisting of linkage mechanisms which are controlled and implemented by the system's registers.

After a module has executed a given function, the controlling (main) program determines the size of the output on the basis of the value of the system's register, and enters it in a catalog. The main program then proceeds to execute the next commands until all of the data processing functions have been executed.

1.3. Standard Representation of the Data.

All data stored in the BSP library, PD as well as UD data, have the same structure, so called "standard representation" (SP), which has the following format:

```
        <SP> ::= <HEADING> , IM(LI), RM(LR)
        <HEADING> ::=  <NAME>, MAT, MZA,LI,LR
```

where

IM(LI)	is a set of integers of length LI,
RM(LR)	is a set of real numbers of length LR,
<NAME>	is a two word hollerith label defining the system's data name (see below),
MAT	is a data identification label, normally corresponding to the material number used in the evaluated data library,
MZA	is the charge-mass (ZA) label which identifies the nuclide: MZA=Z*1000+A (where Z is the charge and A is the mass).

The length of the "heading" is fixed to a maximum of 6 words. With the use of the SP format, it is possible to represent data in a uniform manner, especially in those cases when it is convenient and justified, as in operations involving reading in, copying or retrieving data. Generally speaking, however, SP formats must be different from one another. Their differences are dependent on the structure of the IM and RM data. The formats of these data depend on the type and function of the data, and are identified in the system by unique system names which are used to help in their recognition. In the present version of the program, the system has 9 assigned names (and consequently 6 formats) for the PD type data, and approximately 50 names for UD type data.

The following rules must be observed in assigning formats to IM and RM data: all descriptive information must precede the basic information so as to be able to determine the nature of the subsequent information when these data sets are read. The separation of the information into two separate sets, namely the integers IM and the numerical variables RM, does not contradict any rule inasmuch as the integers are used as flags, indexes, rule numbers, etc..., and are definitive in nature. Although the separation of the data according to type is not obligatory, it has been introduced here so as to have the option to print the content of the SP utility module. If such a separation leads to undesirable limitations, it can be omitted, by entering the integer type information in the RM data memory block. The only disadvantage of this approach would be that, when executing operation *OUTPU, the integer data would be printed in tables having the same format as the numeric variable data.

The following guidelines were adopted in the assignment of names in the GRUKON system.

- A system name can be composed of not more than 6 characters consisting of latin alphabet letters and symbols * / and -.
- The system name must start with the symbol * (and in the case of PD data it must end with the characters IM).
- The following names have been assigned to processed data:

* R *	- resolved resonance parameters,
* U *	- averaged resonance parameters,
* S *	- tabular representation of energy or temperature dependence of the cross-section,
* A *	- secondary neutron angular distribution parameters,
* E *	- secondary neutron energy distribution parameters,
* F *	- designator of the type of cross-section (for group as well as point-wise data, for the unresolved resonance region),

* P *	- subgroup parameters of the resonance structure (for group as well as pointwise data, for the unresolved resonance region),
* D *	- detailed energy dependence of neutron distributions and their angular moments,
* M *	- inter-group transfer matrix.

Any set of PD data is designated by the name *DATA. Control data names are normally designated using the format *I/P-1, where I corresponds to the type of input cross-section data, O to the type of output data, and P is a mnemonic variable which corresponds to a descriptive parameter in the UD file:

C	- concentration,
D	- dilution cross-section,
E	- approximation or interpolation accuracy,
I	- method of interpolation,
G	- group boundary,
T	- medium temperature.

For instance:

*S/C-S	- designation of UD data for the multiplication of tabulated cross-sections with given concentrations,
*S/T-S	- designation of UD data for the adjustment of cross-sections to a given temperature, etc...

When data processing is performed on any type of data, or if input or output data have a non-standard format (i.e. non-GRUKON format), one can use the abbreviations of the English word which designates the type of processing or nature of the input or output data, instead of the parameter names. For instance:

*ENDF	- the name of input or output UD data in ENDF format
*EXTRA	- the name of UD data to be retrieved according to a given criterion (e.g. a given material number MZA), etc...

If certain data are not to be processed, these data are described by a single name. For instance:

*INPUT - the name of an "empty file" of UD data to be entered from punched cards in the internal BSP format.

*TRANS - the name of a file to be copied into the BSP library (e.g., transfer of data).

In the case of "empty files", it is not necessary to enter UD headings in the BSP library, it is enough to have these names stored in the BSP catalog.

The list of control data names used in the system are given in Section 1.5, together with the functions of the corresponding modules.

1.4. Library of Standard Parameters.

All data to be processed, UD as well as PD data, are entered into the BSP library using input modules; after processing, the data are stored in the BSP library. The location of the data in the BSP library is controlled by PP commands originated by the processing program. Questions related to the rational utilization of the resources available to the BSP library are described in more details in subsequent sections of this report which analyze the individual PP commands. The general structure of the BSP library and the data cataloging procedures will also be described at that point. The BSP library is made up of internal BSP memory sectors, each of which is characterized by a sector number, consisting of its address (i.e., the number of the first word), and the dimension of the sector (i.e., the number of allocated words). In order to describe the configuration of the BSP library in the BESM-6 computer, it is necessary to indicate in the "job description" (problem documentation) where each sector is going to be located. Within the BSP library it is possible to use MB elements, system or working MD elements, as well as individual MD and ML elements. For instance, the entry in the problem documentation requesting a specific amount of BSP library storage space, reads

```
sector 51 (2C)
sector 52 (2C)
sector 50 (200-3P)
```

requests the creation of a BSP library consisting of three sectors, two of which (designated 51 and 52) to be located on the system's disc, and one (number 53) on the individual ML #200.

For EC computers, the configuration of the BSP library, is defined in the main program of the software package (see Section 3.2). BSP sectors correspond to direct access files designated by the operator DEFINE; the numeration of the sectors corresponds to the numeration of the files.

Data stored in the BSP library are registered in the library's catalog. The BSP catalog is located in the MR block region COMMON/CATAL/MR(8),MC(6,84). It consists of 84 records, of 6 words each.

The format of a catalog record is as follows:

<record> : : = <name> ,N,M,L

where <name> is a three-word Hollerith label, the first two of which (only one for the BESM-6 computer) combine to form the system's data name,

N - number of the BSP sector (corresponding to the file number in the EC computer, and to the number of the facility in the BESM-6 computer),

M - number of the word which signals the beginning of the data array,

L - the number words taken up by the data array.

The catalog records are numbered from 1 to 84; the first 64 records are reserved for the registration of the data; the remaining 20 records are used to store the names of the processing functions without parameters. In the current version of the program there are eight such processing functions which are located in the following records of the catalog:

65	*INPUT	69	*SELEC
66	*OUTPU	70	*TABLE
67	*TRANS	71	*CONTE
68	*TRANC	72	*CONDE

(the description of these functions is given in Section 1.5)

Initially the first 64 catalog records are empty (contain zeros), subsequently they contain the above-stated names.

With the name *INPUT listed in the catalog it is possible to call upon the input module to transfer UD data from punched cards to the BSP library for their eventual processing, and at the same time register their names in the BSP catalog under the name DATA.

If any of the UD data entered are in the ENDF format, they are first translated into the SP format by the format translation module. After all of the data have been entered, the pertinent module is called upon to perform the required processing steps on the entered UD data array. After each processing step, the name of the resulting quantity is entered in the BSP catalog. The results of this operation can be retrieved using the command *TABLE.

At any point during the processing operation, it is possible to copy the data to the BSP segment designed for long-term storage and at the same time print the pertinent catalog record together with its address. The catalogued data can be accessed at a subsequent stage of their processing by entering their catalog name and address using the special catalog editing command (see Section 2.2). In this manner, the correspondence of the catalog content with the status of the BSP library can be maintained. Although the same system names are used in the BSP library and the catalog, their information content differs depending on whether they are UD or PD data.

In the catalog, the largest information content is carried by the UD names which are used to determine the name of the module that is required for the execution of the processing. PD names in the catalog are rarely used (such as in the use of *ENDF, *SELECT and *EXTRA in the processing, or for the specification of output data). In general, these names can take any form other than a blank (these ,however, can in some cases be interpreted as a signal requiring a specific response, see Section 2.3). In the BSP library, on the other hand, the information content is carried primarily by the PD names; they are used for searching procedures and data identification, while the use of UD names are needed very rarely, except in modules where external UD data are foreseen to be used (at the present time this option is used only in the module *F/G-F for *S/G-F data).

The option to use the same data to perform different functions improves the flexibility of the command language. This can be illustrated by the following examples.

The first example refers to the processing functions *PLUS, *MINUS, *MULTI and *DIVID. In order to execute these operations (addition, deduction, multiplication, division) on two given operands which have the same structure, it is necessary to assign the name of the processing function to the first operand and define it as a UD datum so that it can be retrieved by the processing module.

The second example has to do with the adjustment of data parameters. In this type of operation, the UD and PD data usually interchange their roles: the name of the adjustment operation is assigned to the PD data, and the UD data is processed accordingly. At the present time this operation is applied only to the quantification of data (see Section 2.5).

1.5. Input/Output Procedures

At the present time, the input procedure consists of a group of seven module.
Data input/output.

*INPUT	- input of data from punched cards in internal format into the BSP library.
*IN	- input of data stored in sequentially accessed files into the BSP library.
*ENDF	- input of data from files 1-5 of the evaluated data library in ENDF/B format with translation into the SP format.
*OUTPU	- output of data in internal format from the BSP library to the ATPU[1]
*OUT	- output of data from the BSP library in the form of sequential accessible files.

[1]Translator's Note: no definition of ATPU is provided.

*ENDF	- output of data of the type *S* from the BSP library in the format of file 3 of the ENDF/B library.
*TABLE	- output of data of the type *S* *F* *P* and *M* in the form of annotated tables.
*GRAF	- output of data of the type *S* to the plotter (valid only for the ES computer)

BSP library maintenance

*TRANS	- copy data to a designated location within the BSP library.
*TRANC	- copy data that are not registered and merge with data stored in a given location in the BSP library.
*SELEC	- select data according to their names and copy them to a designated location within the BSP library.
*EXTRA	- select data according to their identification number (i.e., MAT number), charge-mass number MZA, or type of interaction MT, and copy them to a designated location in the BSP library.
*ORDER	- sort data according to their increasing identification number.
*CONDE	- delete blanks in the data stored on the BSP library.
*CONTE	- output and print the names and addresses of data located in a given location in the BSP library.

Data editing

*REMAT	- change the identifying MAT number.
*RENUM	- change the numbering of group intervals for *F* data.
*ABAND	- enter nuclear concentrations in *R* and *U* data and multiply by the nuclear concentrations of the *S* data.
*DISTU	- merge perturbation coefficients with the unresolved resonance parameters (for the calculation of sensitivity coefficients).

Adjustment of the control data

*QUANT	- merge information on energy interval boundaries and information on reaction type (deduced from the data on cross-section type *R*, *U*, and *S*) with the *S/C-S and *U/D-F data.

Computation modules

*R/T-S	- reestablish the cross-section energy dependence on the basis of resonance parameters using Breit-Wigner, Adler-Adler and Reich-Moore formalisms.
*S/T-S	- Doppler broadening of resonance cross-sections given in the form of detailed dependence.
*S/I-S	- change of the energy interpolation method for the detailed representation of the cross-section with a check on the interpolation accuracy of the new (normally linear-linear) interpolation method.
*S/E-S	- delete of redundant energy reference values for given interpolation accuracies in the cross-section tables .
*S/C-S	- average detailed cross-section sets at commonly chosen reference energies; cross-sections for the same reactions, weighted by the concentration.
*S/A-S	- average and calculate detailed cross-section dependences for different reactions (for example the calculation of total cross-section as the sum of partial cross-sections).
*S/G-F	- calculate group parameters of cross-sections on the basis of their detailed energy dependence.
*U/D-F	- derivee point-wise dependence of cross-section attribute values in the unresolved resonance region from averaged resonance parameter values.
*F/G-F	- group-average cross-section attributes on the basis of their energy dependence.
*F/C-F	- fold in cross-section attributes, that is, derive summed cross-section parameters on the basis of given parameters for the cross-section components.
*F/E-P	- calculate sub-group parameters from cross-section attributes such as the cross-section moment, with the guarantee that the required approximation accuracy of the parameter dependence on the dilution cross-section parameters is satisfied.
*P/D-F	- calculate the cross-section attributes on the basis of sub-group parameters.
*AE/-D	- calculate detailed energy distribution of secondary neutrons, and their angular momenta on the basis of angular and energy distribution parameters.

*S/D-M	- calculate inter-group transfer matrices on the basis of the detailed dependence of the cross-section and the distribution of secondary neutrons.
*M/C-M	- multiply inter-group transfer matrices.
*KERMA	- calculate energy production cross-sections (in the *S* format) on the basis of the detailed cross-section energy dependence and the angular and energy distribution of secondary neutrons.

Data repackaging modules

*S/-S	- unfold tabulated cross-sections for a few types of interactions, separate for each type.
*F/-S	- translate averaged cross-section format *F* to *S*.
*F/-F	- create table of combined cross-section attributes from individual attribute data tables.

Modules for the calculation of final results

*SOS	- Derive a function from a detailed cross-section energy dependence and storage it in the *S* format.
*FOF	- Derive special cross-section attributes, such as self-shielding factors, Doppler broadening, shielding of cross-sections, and store them in *F* format.

*PLUS, *MINUS, *MULTI, *DIVID - execute arithmetic operations (addition, subtraction, multiplication and division) in *S* and *F* formats.

2. THE MACROCOMMAND LANGUAGE

The execution of the GRUKON program consists of two parts. The first part, concerned with the data processing (PP) program is made up of the computational system and is written in macrocommand language. The writing of the data processing program (PP) presupposes a knowledge of the characteristics of each data processing step, familiarity with the data structures, and an ability to handle them in a rational manner using external resources. This macrocommand language is concise and is not evident to the layman.

The second part is concerned with the internal manipulation of the (UD) data by the functional modules. It consists of information (such as energy range, temperature regime, energy group boundaries, etc...) that has a simple structure and is easily understood by the user. It is also possible to annotate these data so as to make them easily understood.

The first part is more "conservative" [1]. It is possible to construct a few typical data processing programs which would satisfy most users, and which would only require for the user to assimilate the program and its associated control UD data. This offers the possibility to assign to the programers, or to a small number of qualified users, the task to develop the data processing (PP) programs, who are in any case responsible for the development of the entire software package. In keeping with the objective of this report, this chapter deals with questions concerning the development of the first part of the task, namely the structure of the macrocommand statements used in the (PP) program, the distribution of the available resources of the BSP library, the formulation of the data processing scheme and its realization in the form of the data processing program.

2.1. Macrocommand Language Structure

The PP macrocommands have the following general structure:

<COMMAND>	::= I, J, K <NAME> , N, M, L, where I,J,K are command indexes, given in base ten integers;
<NAME>	label consisting of up to three Hollerith words;
N, M, L	are address variables, given in base ten integers (but in octal notation in the BESM-6 computer), used in the location specification in the memory of the BSP library:

N - number of the BSP segment,
M - address of the first word of a field in that segment,
L - length of the field (L>0), or address of the first word following a field (L<0).

The quantities M and L are expressed in the following notation: $i_1K + i_2$ where i_1 and i_2 are integers (e.g. 10K + 1012). This

[1]Translator's Note: the same word is used in the Russian text.

form of notation is convenient in describing BSP library locations: "K" represents the page in the BSP library, i_1 gives the number of filled pages, i_2 gives the number of words entered in the last unfilled page. In the case of the ES computer, "K" is usually taken to be equal to the length of the disc track given in terms of words, $K=1823_{10}$ (see the description of L R E and C in Section 3.2). In the BESM-6 computer, K is the length of the ML zone, $K=2000_8$. As an example, 10K+1012 in the BESM computer is interpreted as the number $(10_8 * 2000_8) + 1012_8 = 21012_8$; and in the ES computer, as the number (10*1823) + 1012 = 19242.

The notation used to write macrocommands permits the use of abbreviations as well as blanks. Depending on the type and function of the command, it can have the following form:

```
I, J, K    <NAME>     (if I is omitted, M and L are deleted)
I, J, K               (if the name is omitted, N,M,L)
 , , ,     <<NAME>    (if I, J and K are omitted, N, M, and L
                       are deleted).
```

Even if I is very short, it is better to use the notation I,,, to avoid any misunderstanding. Addresses can be given in modified notation; thus the notations for the following three examples are equivalent:

```
10K + 1012 = 10K1012 = 19242
0K + 723 = 0K723 = 723
17K + 0 = 17K 0 = 17K
```

The constant L can have two interpretations, depending on whether it is positive or negative. If L<0 it is accepted by the system without any change; if a minus sign "-" precedes L (e.g. -10K+1012, or -723, or -17K), then its notation is changed to |L|-M (in accordance with the interpretation given above).

On punched cards, the commands are entered starting in the first column (in the BESM-6 computer, starting in the second column), and can take as many as 72 columns. Only one command can be entered on one card, and text for a given command cannot be continued on another. If an error is detected in any one of the seven parts of a command (each separated by a comma) as they are

entered, the program returns the following error message:
"*** error in n" (where n is the number of the part). A gap in the entry is interpreted as a 0 or a blank, depending on whether the gap occurred in a number or a name.

2.2. Types of Commands

Data processing commands.

Data processing commands are the most fundamental and most numerous of all. The basic elements of data processing commands are the following:

I	- number of the catalog record which contains the registration of the input data to be processed;
J	- number of the catalog record which contains the control data ;
K	- number of the catalog record which contains the processed data;
<NAME>	- three Hollerith words, the first two of which (the first only in the case of the BESM-6 computer) represent(s) the name assigned by the system, and the remaining word(s) are arbitrary;
N	- number of the BSP sector that will be used to accommodate the processed data;
M	- the address of the word in the sector that defines the beginning of the processed data field;
±L	- the maximum number of words that can be assigned to output data (+), or the uppermost address that defines the beginning of the BSP sector which cannot be used (-).

Of all of the data processing commands, the data input/output commands are the most numerous. Depending on whether the type of data stored in the BSP library are input or output data, the commands take the following form:

```
, J,K, <NAME> ,N,M,L     for input data, and
  I,J                    for output data.
```

Address Allocation Commands.

It is possible to do without the address part of the data processing commands (i.e., omit N, M, L) if the output data are stored in the "main segment" of the BSP library.

In the ES computer, the system assigns segment 1 as the main segment of the BSP library, in the BESM-6 computer it is segment 11, (MB). The length of this segment is unlimited (the operational system will indicate if there is not enough space available). If for any reason there is a need to assign a different segment to function as the main segment, it is necessary to use the command ,,, FIELD, N, M, L where N, M and L are the segment parameters; if M is omitted, it is assumed to be equal to Φ, and L to be unlimited.

Catalog Editing Commands

Let us assume that certain data are stored in a known location of the BSP library, but they are not registered in the catalog and consequently not accessible (such a situation arises when data are calculated and saved). In order to register such data in the catalog, the following command must be used:

```
,, K, <NAME> ,N,M,L.
```

As a result of this command, the name and address of the data specified in the command will be registered in the Kth record of the catalog.

In another example, there may be a need to change the parameter of certain data registered in the Ith record of the catalog. This can be done by executing the command:

```
I,,K, <NAME> ,N,M,L
```

As a result of this command, the information stored in the Ith record will be moved to the Kth record, with the exception of those parameters specified in the command which will be set equal to those parameters that were entered with the command. For instance, changing the name of certain data, but at the same time preserving their address, can be done by giving the command I,,K, <NAME> where <NAME> is the new assigned name of the data.

With the help of this method, it is possible to address various modules using the same UD data (if the structure of UD data corresponds to that specified in these modules, or if there is a provision to convert the structure of UD data from one module to another, as it is implemented for instance in *F/G-F with respect to the parameters *S/G-F). The content of the catalog can be queried with the help of this command[3], where I is the number of the record which defines the beginning of the desired catalog content. All catalog records, starting with the Ith record to the 64th record inclusive, which contain information are printed out.

Control Commands

This group consists of commands which affect the sequential execution of the data processing programs. The general format of this set of commands is:

```
, , K₁,    (((
. . .   . . . .
. . .   . . . .
I₂, J₂, K₂,)))
```

where K_1, I_2, J_2, K_2 ≥ 0

With the help of this command, it is possible to execute all commands specified within the parentheses K_1 number of times. If one or the other of the repeated commands, which has the form such as I,J,K <NAME>, is encountered by the program during the first cycle of this procedure, then the counter of the indexes I,J and K will be advanced to I_2, J_2 and K_2 after each cycle of the procedure, respectively:

$$I=I+I_2,\ J=J+J_2,\ K=K+K_2,\ \text{<NAME>}$$

With the help of this cyclical command, it is possible to formulate long chains of data processing functions in a concise manner. For instance instead of having to write

```
10, 1, 11, *DATA*
11, 2, 12, *DATA*     the same          ,, 6, (((
12, 3, 13, *DATA*    command can        10,1,11*DATA*
13, 4, 14, *DATA*   be written as       I,I,I, )))
14, 5, 15, *DATA*
15, 6, 16, *DATA*
```

[3] Translator's note: the command is omitted in the original text.

The use of the assigned name DATA, as is shown in this example, is permissible inasmuch as the names of the processed data are not used by the system. The command which indicates the end of the data processing program is: ,,, END.

2.3. Rules of Omission

Some of the elements of the data processing commands (e.g., the name of the data, the segment number N, the address of the first word M, the total number of words) are assigned by the system and can therefore be omitted in the input of the command. In addition to the convenience that the commands are shorter, this rule of omission provides a more significant advantage, particularly in the assignment of the M and L parameters.

First, the system possesses the required information to optimize the representation of the dynamic character of the data in accordance with the current status of the BSP library, which permits a more economical usage of the internal memory.

Second, by omitting the entry of the M and L parameters, the correspondence of the catalog content with the BSP library status is guaranteed and precludes accidental omission of data in the data processing operations. The method used to determine a missing parameter depends on its type. There is, however, one rule that is valid for all parameters without exception: if the value of a parameter of a given command is neither a zero or a blank, it is accepted by the system as such (except for identical operations from the point of view of the system). In the following list, the parameters that are singled out are presumed to be omitted in the command:

<NAME>	- the omission of the name is allowed only if the Kth record, used to enter the command parameters, already has a name entered. The absence of a name in the command is interpreted by the system as an instruction to store the results of the data processing behind the data that are entered in that record and to retain the previously assigned name (in this case, the parameters M and L must also be absent).
N	- the number of the segment is taken from the Kth record. If the record is empty, N is assumed to be

equal to the number of the main segment of the BSP library.

Case 1. Both the name and the segment number N are given. M is taken to be equal to the first free word in the segment N.

Case 2. The name is given, the number N is omitted. If the record is empty, M is taken to be equal to the number of the first word in the main segment. If the record contains recorded data, the value for M is taken from the record.

Case 3. The name is omitted, the number N is given. Number M is taken to be equal to the number of the first free word in the N segment, the name is taken from the Kth record in the catalog.

Case 4. Both name and the N number are omitted. The results are stored behind the data registered in the Kth record of the catalog.

L - If the kth record is empty, L is taken to be equal to the length of the memory section of the segment located after the word whose address is given by M. If the record contains data, L is first taken to be equal to the length of these data; thereafter, the catalog is checked whether there are data in the N segment (in the order of their distribution), if that is the case, then the value of L is increased according to the remaining length of the segment.

2.4. Sequence of Data Processing Steps

The address part of the catalog record (e.g. parameters N, M and L) define a given section of the BSP library which, generally speaking, can accommodate not one, but a number of PD as well as UD data formats. The total number of formatted data, stored one behind the other in a single segment of the BSP library, and registered under one name in the catalog, will be referred in this report as a "data cluster"

A data cluster can be entered from punched cards using the *INPUT operation. Data obtained after the *ENDF operation, which converts files of evaluated data into a standard format, are also registered as data clusters. It is also possible to consolidate data into clusters in data processing operations; the command language provides for such a possibility (see the input of the name in the "rules of omission" section above).

Let us stop here and analyze the question how data clusters are interpreted by the data processing modules, and the manner in which they aide in the execution of data processing commands.

The programming rules used in the creation of functional modules (see Section 4) foresee the cyclical input of PD and UD data. Each functional module executes a "series" of data processing steps using data contained in the clusters. At the present time, two separate schemes are used in the organization of cycles.

The first and most widely used scheme is implemented at the beginning of each processing step, when one set of PD and UD data are required to be retrieved from the series at a time in order to produce a single set of PD data.

Letting I(i) be the ith set of input PD data, P(j) be the jth set of UD input data, and O(i,j) be the set of PD output data which were derived from the I(i) and P(j) input data, then the sequential processing of a data cluster can be expressed in the form of the following cycle:

$$((I(i)/P(j) \rightarrow O(i,j) \qquad i=1,N \qquad j=1,M$$

where N is the number of PD input data, and M the number of UD input data. The resultant O(i,j) set of data will be stored in the BSP library in the form of a data cluster in the order of their sequential input. In practice, it is customary in sequential calculations to organize the input data in the form of clusters.

The following rules are observed in the processing of data clusters. If a given cluster I contains data that are not foreseen as input in the called functional module, or if the input data parameters do not correspond to the UD parameter values (e.g., the energy range of the given PD data, or lies outside of the energy range boundaries of the UD data), then such PD data are left out. If a cluster does not contain data types that are to be processed, then the entry in the catalog will consist of the registration of a set of output data having a length L=0 (i.e., it will exist formally in the catalog only, but will not exist in the BSP library).

In practice, the use of UD data clusters in serial calculations is related only to the data quantification procedure (see next Section): in order to subdivide the energy range of a given data set into smaller intervals, and to perform independent calculations in each of them, it is convenient to store the information pertaining to these intervals (which may be quite numerous) in the form of UD data, rather than PD data. The number of UD data in a cluster will be equal to the number of energy intervals. As a result, the output will consist of a data cluster for each interval. A cluster of UD data (that underwent the *S/C-S and *U/D-F conversion) is generated with the *QUANT operator. Other such cases where UD data clusters are used occur primarily in the generation of evaluated data.

At the present time, the second procedure is used only in four modules: *S/C-S, *F/C-F, *M/C-M and *F/-F. These modules are used primarily to merge data. (The term "merge", used here, encompasses any data operation in which more than one set of input data results in one set of output data, independent of the data conversion algorithm). Since this operation is based on a cyclical operation processing input data within the conversion algorithm, the series of operations is performed with one UD data cluster at a time. The number of output PD data is determined by the number of UD data in a cluster. (The number of output data can, however, be smaller if part of the UD data in the input cluster are not given in terms of parameters).

The use of data clusters and serial processing of data makes the whole data processing program independent of the number of attributes (such as the number of levels of the system, the type of interactions, the number of energy intervals for which the data are given) and of the qualitative characteristics of the evaluated data structure (such as the presence of unresolved resonance regions). Because of this characteristic, this processing program, although designed for general applications, is also applicable to the solution of more specific problems.

2.5. Data Processing Procedures

In the development of this data processing program, the calculational operations were postulated on the basis of a

standardized representation of the data structure which are generated at each step of the data processing operations.
In the selection of individual data processing modules, and in the determination of the sequence in which they are called during data processing operations, it has proven to be convenient to represent the relative position of the standard quantities along the energy axis in the form of a data structure.

Let us look at a typical standard data structure of the ENDF/B library (see Fig.2.1). Upon inspection it can be seen that it has the following features:

- a preponderance of S-, R- and U-data (that is, a presence of a few similar representations in the same as well as in different energy regions);
- a total or partial overlapping of R- and S-data;
- a total or partial overlapping of U- or S-data;
- a total overlapping of R-data;
- a total overlapping of U-data;
- a total or partial overlapping of S-data;
- an absence of any overlapping of R- and U-data.

Note that in the case of partial overlapping, the overlapping can be eliminated by subdividing the sets of data into smaller energy groups defined by the group boundaries of the other data sets (shown in Fig. 2.1 by the dashed lines). Partial overlapping constructions can be avoided by executing the commands: *S/C-S and *U/D-F[4].

Let us define data that completely overlap as "quantiles" of data, and the operations involved in their processing the "quantilization" of data. It is assumed from now on that data can either overlap entirely or not al all. It is thus possible

[4]Actually, the need for quantification arises because of the variety of ways which can be used to merge detailed information entered with the *R* and *S* parameters and the probabilities given by parameters *U* in the calculation of non-linear functions. The detailed information is merged at the *S* level, using the command *S/C-S, and the probability at the *F* level using the command *F/C-F. The quantification operations are used to separate the detailed data representations from the probability representation.

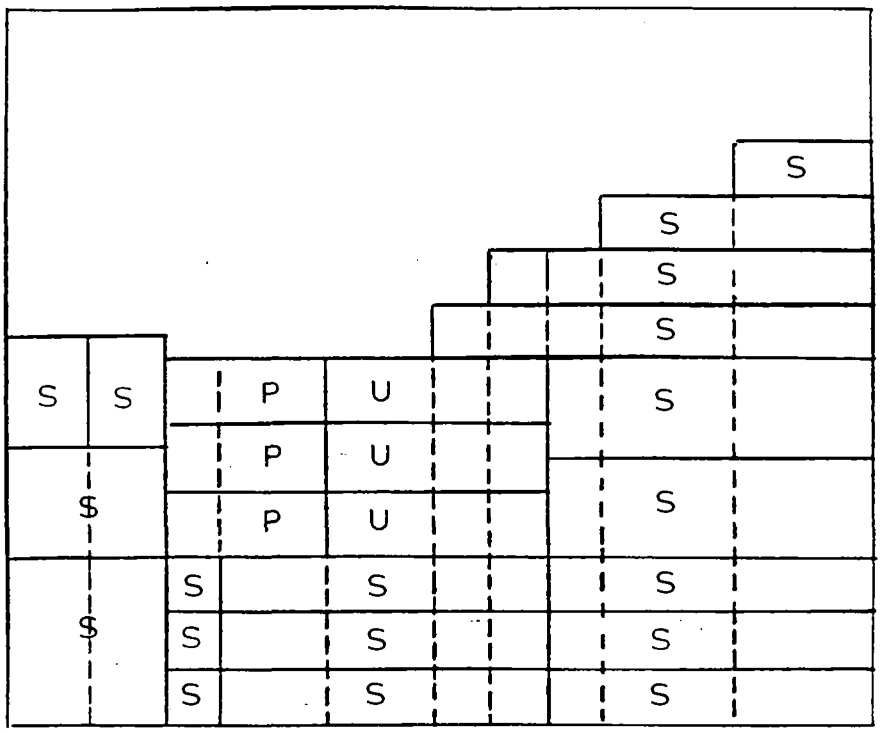

Fig. 2.1 Infrastructure of Cross-Section Data in the Evaluated Data Library

P - Resonance parameters
U - Average resonance parameters
S - Cross-sections

to identify two types of relationships of contiguous data sets: in one case, the data belong to the same "quantile" and the relationship is denoted by the symbol "X", and in the other, the data belong to different "quantiles" and the relationship is denoted by the symbol "+"; for example RxS or R+U. It is natural to represent most data as the result of a "multiplication by a whole number", or of an "exponentiation to a given power", depending on whether most of the data belong to different or the same quantile:

$$S + S + \dots + S = nS$$

$$R * R * \dots * R = R^n$$

As mentioned in the previous section of this report, as long as the quantitative characteristics of the infrastructure can be omitted in the development of the data processing program, the value of n is not important; what is important is the large number of data.

Using these data infrastructure designations, the information given in Fig. 2. can be represented by the following expression:

$$2S^{5} + 2(R^{3} * S^{3}) + U^{3} * S^{3} + U^{3} * S^{4} + U^{3} * S^{5} + S^{7} + S^{8} + S^{9}$$

or, taking into account the comment regarding the large number of data:

$$nS + n(R^{n} * S^{n}) + n(U^{n} * S^{n})$$

The possible procedures to calculate the cross-section attributes from the evaluated data S, R and U (in ascending order), in the framework of the GRUKON software package, are given below.

Procedure 1.

Calculation of group cross-section attributes F from separate S- , R- and U- data:

$$\begin{tikzcd}
S \arrow[r, "S/I-S"] & S \arrow[r, "S/T-S"] & S \arrow[r, "S/G-F"] & F \\
R \arrow[r, "R/T-S"] & S \arrow[r, "S/T-S"] & S \arrow[r, "S/G-F"] & F \\
U \arrow[r, "U/D-F"] & F \arrow[r, "F/G-F"] & F &
\end{tikzcd}$$

Procedure 2.

Calculation of F with consideration given to the large number of data:

$$\begin{tikzcd}
nS^{n} \arrow[r, "S/I-S"] & nS^{n} \arrow[r, "S/C-S"] & S \arrow[r, "S/T-S"] & S \arrow[r, "S/G-F"] & F & \\
nR^{n} \arrow[r, "R/T-S"] & nS^{n} \arrow[r, "S/C-S"] & S \arrow[r, "S/T-S"] & S \arrow[r, "S/G-F"] & F & \\
nU^{n} \arrow[r, "U/D-F^{5}"] & nF^{n} \arrow[r, "F/G-F"] & nF^{n} \arrow[r, "F/C-F"] & F \arrow[r, "F/G-F"] & F &
\end{tikzcd}$$

A secondary integration of *F/G-F is necessary for the elimination of the quantile boundaries in the data group structure.

Procedure 3.

For infrastructures which are made up of combinations of detailed data dependencies and resonance parameters, the calculation of the attributes can be calculated using the following scheme:

$$\begin{tikzcd}
nS^{n} + n(R^{n} * S^{n}) \arrow[r, bend left, "R/T-S"] \arrow[r, bend right, "S/I-S"'] & nS^{n} \arrow[r, "S/C-S"] & S^{n} \arrow[r, "S/T-S"] & S \arrow[r, "S/G-F"] & F
\end{tikzcd}$$

In this configuration, the R/T-S and S/I-S operations are executed using the same data cluster, whereby the R/T-S operation is executed on R-type data, and the S/I-S

[5]The parameters are quantified

is executed on S-type data; the results of these operations are merged to form a single cluster nS^{n}. This is done so that the S/I-S operation is not executed on S-type data that are derived from R data, since they are always produced in a linear form.

Procedure 4.

For those materials whose cross-sections include regions of unresolved resonances, the following procedure is used to process the data:

$$nS^{n} + n(R^{n} * S^{n}) + n(U^{n} * S^{n}) \longrightarrow \cdots$$

$$\begin{tikzcd}
 & & nS^{n} \arrow[r, "S/C-S\ (*)"] & nS \arrow[r, "S/T-S"] & nS \arrow[dr, dash, "S/G-F"] & & \\
\cdots \arrow[r, dash] & {} \arrow[ur, bend left, "R/T-S"] \arrow[ur, "S/I-S"'] \arrow[dr, "U/D-F\ (*)"'] & & & & {} \arrow[r] & \cdots \\
 & & nF^{n} \arrow[rrru, dash, "F/G-F"'] & & & &
\end{tikzcd}$$

$$\begin{tikzcd}
\cdots \arrow[r] & nF^{n} \arrow[r, "F/C-F"] & F \arrow[r, "F/G-F"] & F
\end{tikzcd}$$

In this configuration the data are split into two clusters, each of which is processed independently until the group attributes have been calculated. The combining of the detailed and probability information is executed independently.

Procedures 2, 3 and 4 take care of all cases that can be encountered in the ENDF/B evaluated data library; whereby procedure 3 can be applied to the case treated in procedure 2, and procedure 4 can be applied to the cases treated in procedures 2 and 3.

Procedure 4 can also be used in those cases that have not been foreseen in ENDF/B, but which are important when resolved and unresolved resonance regions are superimposed: $n(R^{n} * U^{n} * S^{n})$.

Procedure 5.

In the case of linear attributes of averaged group cross-sections, the complex processing algorithm F/C-F can be omitted if the format conversion module F/-S is used, allowing for the conversion of average cross-section data from an F format to the S format. In the most general case, this procedure has the following configuration:

$$\begin{tikzcd}
n(R^{n} * U^{n} * S^{n}) \arrow[r, "R/T-S"] & n(U^{n} * S^{n}) \arrow[r, bend left, dash, "S/G-F"] \arrow[r, bend right, dash, "U/D-F"'] & {} \arrow[r] & \cdots
\end{tikzcd}$$

$$\begin{tikzcd}
\cdots \arrow[r] & nF^{n} \arrow[r, "F/-S"] & nS^{n} \arrow[r, "S/C-S"] & S \arrow[r, "S/G-F"] & {}
\end{tikzcd}$$

It is expedient to merge the results of the R/T-S operation with the input data cluster. Since the cross-sections are constructed at the group level, it is not necessary to use linear interpolation of the data.

2.6. Data Processing Program

Let us look at some of the methods used in the data processing program (PP) which were developed for the solution of various computational problems.

Assuming that the processing method has already been defined, the next step in the processing is the selection of the most optimal configuration of the BSP library. Normally this problem is solved in the following manner.

In the case of simple processing tasks, when the amount of data is not large, and the program flow does not have any branching (such as in the first case described in Section 2.4), it is possible to limit the required amount of memory to a single BSP library segment. For longer processing procedures having a linear dependence, it is more efficient to use an additional segment of the BSP library so as to be able to copy the data from one (segment) to the other, deleting the input data that were entered in the preceding step. This technique allows for a significant reduction in internal memory requirement, which is particularly important if the computation is performed on the BESM-6 computer.

In order to perform procedures with multiple branchings, as in the case of procedure 4 above, it is convenient to have two memory segments, in addition to the main one, so as to be able to exchange data between the main and the first working segments, and the main and second working segment alternatingly. Finally, if it becomes necessary to save the intervening data and to break up the computation into a number of stages, one could use one of the working segments as an archival memory segment (which allows for a long term storage of data), or use an additional, fourth, memory segment.

Concentrating on this last option, as one which is the more general, thus dedicating three BSP library memory segments, one would have: a main segment (designated as segment 1), and two working segments (designated 2 and 3). The data in the archival file, which will be used for subsequent access, will be stored in the internal format. Each category of information recorded in the catalog is assigned a specific number of records in the catalog. For example:

1 - 19	-	control data;
20 - 29	-	input data to be processed and UD data that have to be quantified;
30 - 59	-	intervening PD data;
60 - 63	-	processing results.

Using these conventions, the data processing program could have the following appearance:

PP Text	Comments
/__/65,1,*ENDF /__/,65,2,*R/T-S /__/,65,3,*S/I-S /__/,65,4,*S/T-S /__/,65,25,*S/C-S /__/,65,26,*U/D-F /__/,65,7,*S/G-F 7, ,8,*F/G-F /__/,65,9,*F/C-F /__/,65,10,*OUT /__/,65,11,*F/-F /__/,65,12,*MODF	Control data input is executed by the *INPUT operator which is always entered in the 65th record of the catalog. The UD data S/C-S and *U/D-F data require quantification. The input of the *F/G-F is replaced by the renamed *S/G-F data. Data are entered in the main segment.
/__/,1,20,*DATA*	Input of processed data from the library in ENDF/B format. This is carried out by the ENDF/B operator registered in the first record of the catalog. The name *DATA* indicates that there are many data types in the cluster (e.g., *R*, *U*, *S*, etc.).
20,21,*QUANT 25,21,5,*S/C-S 26,21,6,*U/D-F	Quantification of data. In order to execute this operation on the entered PD data the *QUANT designator is adopted, which gives the possibility to use them as UD data in the *S/C-S and *U/D-F conversions. This operation results in the formation of *S/C-S and *U/D-F UD clusters, which break up the entire energy region into non-overlapping intervals.
	Data processing
20,2,52,*S*,2	Calculation of the detailed energy dependance at zero degrees from resonance parameters. The results are entered into the 2nd segment of the BSP library.
20,3,52	Linear interpolation of point-wise cross-sections and background cross-sections and their addition to resonance cross-sections.
52,5,53,*S*,3	Addition of the resonance cross-sections to the background cross-sections and storage of the results in the 3rd BSP segment.

53,4,52,*S*	Adjustment of the detailed cross-section dependence to the given temperature, and their storage in the 2nd segment of the BSP library.
20,6,52	Calculation of the detailed dependence of the spin statistical factor of the cross-sections from averaged resonance parameters and their storage in the 2nd segment.
52,7,53,*F*	Calculation of the group-averaged values of the spin statistical factor from the detailed dependence of the cross-section, and their storage in the 3rd segment of the BSP library superseding earlier data.
52,8,53	Calculation of the group-averaged values of the cross-section spin statistical factor from the detailed dependence of the spin statistical factor, and their storage in the 3rd segment.
53,9,52,*F*	Merging of the spin statistical factors and their storage in the 2nd segment.
52,8,53,*F*	Second integration within each group in order to eliminate the quantile boundaries from the group structure and storage of the results in the 3rd segment.
53,10	Storage in the archive is executed using the operator *OUT; the UD data of this operator defines the parameters of the subsequently accessed file and the entry format.
53,11,62,*F*,2	Preparation of the data for output includes the selection of the spin statistical factors to match the set of output parameters (e.g., the reaction type, the temperature, the dilution cross-section) using the *F/-F operator.
62,12,63,*F*,3	Calculation of the required attributes (e.g., self-shielding factors of shielded cross-sections, effects of temperature increase).
63,70	Printing of the output data in the form of annotated tables is executed by the TABLE operator which is permanently entered in the 70th record of the catalog.
,,,END	End of the processing program.

3. SYSTEM INPUT`

Two groups of input procedures can be identified in the GRUKON program: the data processing control procedures, and the data linkage procedures (see Fig. 3.1).

The control procedures organize the calling of functional modules and the preparation of the data for their subsequent processing by data processing program.

The linkage procedures facilitate the exchange of data between the BSP library and the scratch file of the functional module.

The control procedures provide a pool of information to the functional modules, and the linkage procedures provide data from that pool of information to the functional module, at the same time guaranteeing its integrity. The interaction between the control and linkage procedures is realized by the system's registers; these consist of special regions of COMMON which hold information on the location of data required for the execution of the module's functions (conversion register), and on the exchange interface of data between the BSP library and the scratch file of the module (exchange register). The system has a diagnostics provision, included in procedure ERROW, which contains a list of errors which can be encountered in the data processing operations. The linkage procedures, together with the diagnostics procedure, make up a group of programs available to the programmer of functional modules; they are incorporated in the body of the module and constitute its system support.

3.1. The System's Registers

The exchange of information between the components of the input system and between the input system and the functional modules is realized by the system's registers located in specially assigned regions of COMMON.

The system has two registers.

The first is the conversion register located in the first eight words of the COMMON/CATAL/MR(512) region. It is used for the

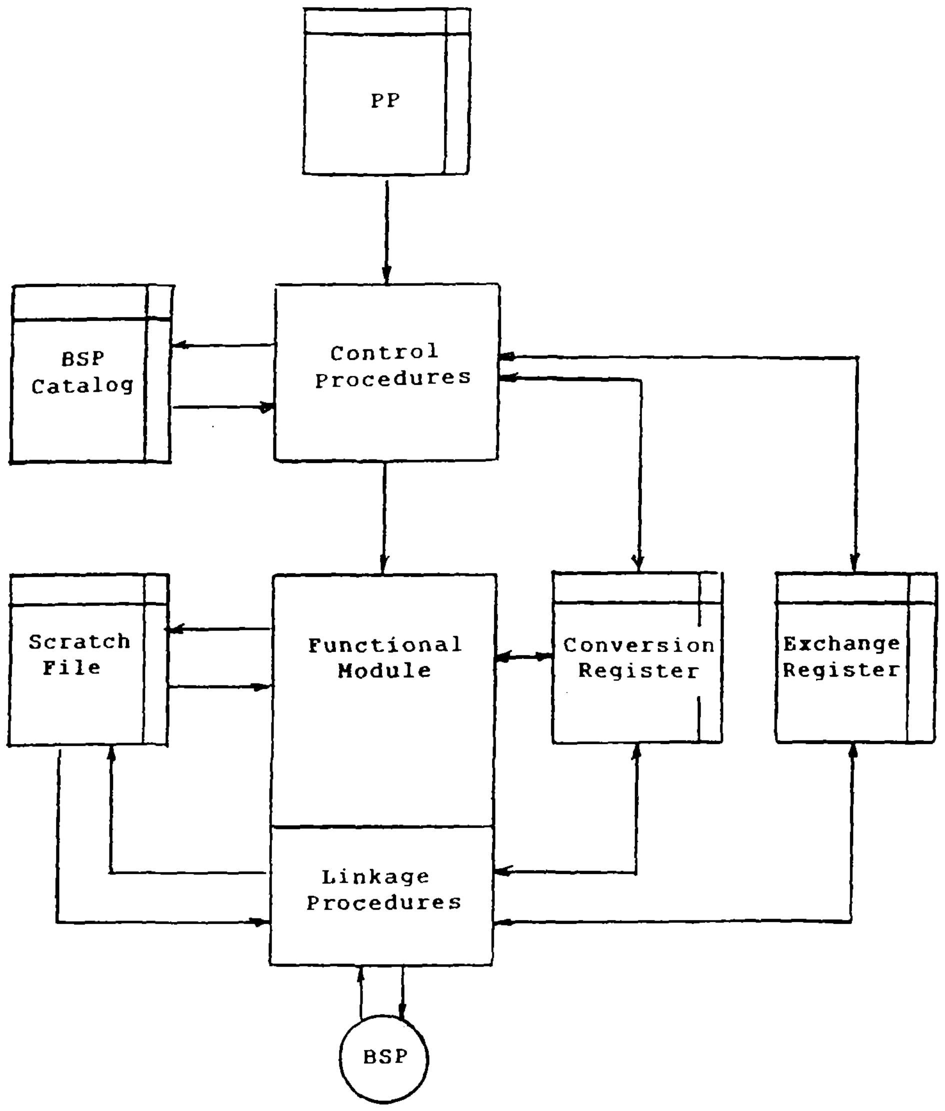

Fig. 3.1 Interaction of System Components with Functional Modules

transmission to the functional modules of information on the locations of BSP fields (containing input, control and processed data) and of the scratch file regions reserved for the storage of intermediate data.

The second is the exchange register located in the COMMON/LIST/ML (18) region and contains information on the location and status of a memory block composed of three buffers reserved for the storage of those sections of the BSP library that participate in the exchange of data. The exchange register is used by the linkage procedures (see Section 3.3).

The controlling programs define the initial values stored in the registers; the functional modules, using the linkage procedures, execute the exchange of data with the BSP library on the basis of the values stored in the registers. In the course of the exchange process the values stored in the registers are redefined in accordance with the status of the information. At the end of a data processing operation, the control of the registers is returned to the control procedures, and their parameters changed to the values corresponding to the initial condition which are then used to determine the next initial values, etc.

The elements of the conversion module are defined as follows:

MR(i),i=1,2,3	the number of the element MR in the memory block which is followed by the catalog record which contains the address of the ith BSP field (i=1,2,3 for input, control and processed data fields, respectively).
MR(3+i),i=1,2,3	the number of the current word of the ith BSP field (initially, this number is equal to the number of the first word of the ith field).
MR(7)	upper limit of the scratch file region of the module, field 0 (which is equal to the maximum allowed number of a word in the undesignated region of COMMON).
MR(8)	the number of the current word in this field (initially, this number is equal to the address of the first unused word).

After the completion of the data processing operation, the functional module assigns the value of 0 to the MR(1), MR(2) and MR(3) elements. The zero value of these register elements serves as a flag to the main program indicating that the function of this module has been performed. At the time of the end of the performed function, the main program uses only one of the remaining register elements, namely element MR(6) which contains the address of the first unused word in the 3rd BSP field, and which follows the results. This information is then used to determine the length of the output data fields as they are registered in the catalog.

The exchange register has the following structure:

```
COMMON/LIST/LC(3),LF(3),LP(3),LW(3),LT(3)
```

The value of the index corresponds to the number of the buffer, which is equal to the number of the BSP field used in the sector-wise exchange of data (i.e., that is equal to the number of the exchange channel).

The following information is contained in the ith channel of the buffers (where i=1,2,3):

LT(i)	the number of the directly accessible file that contains the ith BSP field (for the BESM-6 computer, it is the mathematical number of the installation[6]).
LF(i)	the registration number of the BSP section from that file given in the ith buffer of the memory block. If the the buffer is empty, the the value for LF(i)=-1.
LW(i)	the number of the word in the undesignated COMMON/P(i) region which indicates the location of the beginning of the ith buffer. In the case of the BESM-6 computer, the following additional information is provided:
LP(i)	the number of the 03U sector which contains the ith buffer. The LP memory block is not used in the ES computer.

With the aid of the buffers mentioned above, it is possible to effectuate the exchange of data with the use of "split channels", that is, when each channel has its own corresponding LF file number (for the BESM-6 computer it is the number of the installation) different from all of the others. When processing data from the BSP library, it is possible that two or even all three BSM fields are stored in a single file (in one installation), and consequently in the same sector. It is evident that in the exchange of data with the BSP library, the sector used is the one that is located in the operational part of the memory even if it is transferred via another exchange channel. There is a provision in the linkage procedures for an automatic transition to a more optimal operational mode, a so-called "integration of channels" mode, which comes into effect as soon as the condition for a one-to-one correspondence of the channel and file numbers (or the installation number) ceases to exist. This integrated operational mode leads to the necessity to choose the least "needed" sector in the available buffers for

[6]Translator's Note: literal translation of the original.

the subsequent restoration (to the beginning of the next cycle), and to keep track which exchange (reading or writing) mode was used in each of the buffers. At the present time, the choice criterion used (to choose the least needed sector) is based on the addressing frequency: that is, when restoration is required, the sector which was addressed the least number of times since the last exchange is omitted. The concept of "restoration" needs clarification; although it will be discussed in more detail in the description of the linkage procedures (see Section 3.3), at this point it is only necessary to note that one can extract an address from a buffer (or enter an address into a buffer) in the form of a simple variable, as well as in the form of a memory block (or more exactly, that part of the memory block that is included in the buffer). The following buffers are used in the integrated operational exchange mode:

LC(i)	- the number of addresses to the ith buffer since the last exchange performed with the BSP library.
LR(i)	- flag that determines whether a buffer participated in an entry channel exchange (channel 3); it is equal to the channel number that is used for the exchange in a sequential addressing mode until the point at which there is a channel 3 exchange, after which the last value is retained. This flag is used to determine whether the content of the ith buffer must be entered into the BSP library before its sequential restoration, or not.

The exchange register controls the operation of the functional module implicitly by using the linkage procedures, and does not expect an automatic address to be originated from the module. If the need arises to change the register values in order to alter the exchange mode, it is necessary to use the system's DISPAG procedure (see Section 3.3).

3.2. Control Procedures

The principal control procedures are: the main program (in the BESM-6 version it is the GRUKON PROGRAM), and procedures MONIT, CODE, REGIST and PROCESS. The interaction of these procedures with each other and with the functional module FM is shown in Fig. 3.2. The function of the main program consists in the setting of the global parameters of the GRUKON program; after

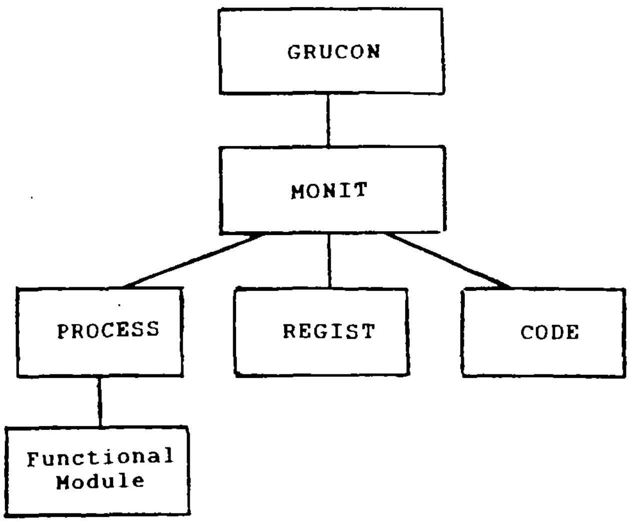

Fig. 3.2 Interactions of Control Procedures

this initialization, control is passed to the MONIT procedure. The global parameters consist of the following quantities:

- the length of the working field dedicated to a given module for the storage of intermediate data on the scratch file, and
- the unit length of the word used in the BSP library.

In the case of the BESM-6 computer, it is also necessary to define the BSP segment, that is, to designate and allocate the direct access files (in the BESM-6 computer the BSP segments are defined in the "job description" on the basis of the nature of the required MB, MD and ML elements). It is convenient to dedicate 2 to 3 segments in the BSP library so as to be able to manipulate data in recurrent operations using the same memory segments (see Section 2.5).

The selection of global parameters is different for the BESM-6 and ES computers. In the BESM-6 computer, all of the free memory available from the resident memory, estimated on the basis of the size of the problem and the maximum size of the functional module, is assigned to the scratch file. The standard length of a BSP sector is equal to 1024 words. Consequently, the text of the main program on the BESM-6 computer does not have any free parameters, and can therefore be recorded in a straight forward manner. On the ES computer, on the other hand, even though the larger size of the operational memory allows for an increased size of the working field of a functional module, it is not required that it be used in its entirety: only individual modules divide up the working memory available from the entire working field. In some cases an increase in the size of the field can improve the efficiency of the program, however, it is better to incorporate such improvements in the programming of the functional modules; for example by repeating calculational operations using more efficient usage of memory, or by printing out a message indicating the need to increase the size of the module working field in the scratch file. The parameter used as a standard is practically the same as that for the BESM-6 computer, and is apparently of a length which guarantees an acceptable program efficiency in the solution of typical problems.

The second global parameter, namely the length of the BSP segment, is usually taken to be equal to the track length of a disc element measured in words, and depends on the disc type being used. For 29 Mb discs it is equal to 1823. The criterion used to select these parameters, such as the number and lengths of files required for the storage of the BSP library, depends very much on the type of problem to be solved and it is difficult to give any recommendations as to their sizes. It is suggested that the users of this program define these values themselves giving due consideration to the nature of the data that are to be processed as well as to the type of processing envisaged. In the worst case, the program will issue an error message that there is not enough available memory, requiring the job to be resubmitted. If the sequence of calculational operations were properly planned, making proper use of information archiving, the computational sequence can then be picked up at the point at which it was interupted.

The listings of the main programs of the standard ES and BESM-6 versions of the GRUKON software package are given below.

For the EC computer

```
COMMON//P(20000)
COMMON/LREC/LREC
LREC = 1823
DEFINE FILE (1,500,1823,U,NREC)
CALL MONIT (20000)
STOP
END
```

For the BESM-6 computer

```
PROGRAM GRUKON
COMMON//P(16384)
COMMON/LREG/LREG
LREG = 1024
CALL MONIT
CALL EXIT
END
```

In the case of the ES computer, the working field of the module consists of 20000 words; one direct access file, which has a capacity of 500 entries, is dedicated to the BSP library; the length of an entry corresponds to the track length of a 29 Mb disc, and consists of 1823 words.

In the case of the BESM-6 computer, the length of a working field consists of 20, O3U leaves (?) and the BSP library entry is fixed at 1024 words; the CALL EXIT command implies a change from a computational to a data processing operational mode.
The MONIT procedure executes the overall data processing operation. Its sole parameter is the length of the COMMON field (i.e., the maximum value of the module working field address).

The function of the MONIT procedure consists of the following:

1. Definition of the initial status of the system's registers.
2. Definition of the initial status of the BSP catalog, and entering of the names of the data processing operations to be executed without enumerating their parameters.
3. Input of the data processing subroutine (effectuated with the help of the CODE procedure).
4. Implementation of the control commands.
5. Implementation of the allocation commands.
6. Implementation of the catalog editing commands.
7. Implementation of the data processing commands which includes:
   a) pre-designation of command parameters implied by virtue of their omission (see Section 2.3.),
   b) definition of the conversion register parameters requiring
      - the setting of the BSP fields allocation on the basis of the content of the processing commands, input data and command parameters, and
      - the allocation of a BSP field for the storage of results in accordance with the address given in the command, or if it not given, assigning a block of free space on the basis of information given in the catalog;
   c) re-definition of the exchange register parameters (with the help of the REGIST procedure);
   d) calling the functional module (with the help of the PROCES procedure);

e) entering the length of the data processing results into the BSP catalog;

f) printing of the information on the number of interruptions during the execution of the processing operation (if such did occur);

g) transfer of the buffer contents into the BSP library;

h) execution of the next data processing command.

The CODE procedure is used to transfer the data processing subroutine from punched cards in accordance with the command syntax given in Section 2.1. The following error message is printed out if the program comes across a command that it cannot understand:

```
* * *  ERROR  B  n   COMMAND PART
```

where n is one of seven parts of the command, separated by a comma.

The REGIST procedure determines the parameters of the exchange register on the basis of its preceding status. Using the free buffer LC, the correspondence between the channel number and the number of the file (number of the installation) needed at each successive processing step is provided. If, after comparing this correspondence with that stored in the LF buffer, it turns out that this file was used in a different channel, the address of the LW buffer is re-assigned. If the file was not used in the preceding step, then this buffer, which remained undetermined, is declared to be empty (i.e., LF(i)=-1) and is combined with this file (i.e., the value of LT is taken to be equal to the number of the file), and the LR and LC registers are taken to be equal to zero. As a result, the information stored in the buffers becomes available for the next processing step without any need for additional exchange or transfer operations.

The PROCES procedure establishes the correspondence between the names of the command parameters (the system's name of the functional module) and the module initiation procedure, and transfers command to that procedure.

The name of the main program is derived from the system's name by the elimination of the * symbol and the replacement of the symbols / and - by the symbol X; for instance: *R/T-S becomes

RXTXS. Before the transfer of control, the name of the module and the time when it was initiated is printed out in the form of the following message:

```
* * * MODULE ...   INITIATED AT ... SEC
```

In the event that control was not transferred, or if the module was incorrectly described (e.g., if the values of the first three elements of the conversion register were not changed to zero), the following error message is printed:

```
* * * MODULE ... IS MISSING
```

and the computation is interupted.

In the case of the BESM-6 computer, the name of the initiation procedure is derived automatically from the control parameters, and the control transfer is carried out with the help of the LOADGO procedure which loads the module initiation procedure in the form of a dynamic(?) heading. The text string that defines the name of the module initiation procedure is a parameter of the LOADGO procedure. Consequently, the names of the functional modules are not stored explicitly anywhere and the addition of new modules does not require any changes to the system.

In the case of the ES computer, the system's names are determined by the DATA operator in the PROCES procedure, and the command transfer is carried out by the conditional operator IF, under the condition that the names and parameters are equivalent to those of the system's.

The addition of a new module requires the following:

- the determination of the system's name with the help of the DATA operator, and
- the introduction of a conditional operator for addressing the module initiation procedure.

For example:

```
REAL*8 MRXTXS
DATA ARXTXS/*R/T-S
....................
IF_(M.EQ.ARXTXS)_CALL_RXTXS
```

where M is a variable which contains the names of the control parameters.

3.3. Linkage and Diagnostics Procedures

The linkage procedures are subroutines designed specifically for simplifying the access to the data in the framework of the system's rules and conventions. The utilization of these procedures by the functional modules guarantees the preservation of the integrity of the system's information content and of the information compatibility.

The linkage procedures consists of the following :

- the MFIND procedure for the search of data in the BSP library;
- the DISPAG procedure for the restructuring of the exchange register;
- procedures for the transfer of data blocks between the scratch tape and the BSP library at various levels of processing: such as procedures CANAL, MWORD, TRAC, MWTRAC, LOADR, LOADU and other procedures of the general type LOADX, where X refers to the type of processed data defined in accordance with the program's classification system (see Section 1.3).

In addition to the linkage procedures, programers of individual functional modules have at their disposal an error diagnostics system in the form of procedure ERROW, whose functions are described below.

The function of the MFIND(NAME,MAT) procedure is to organize the sequential transfer of the processed and control data. The following parameters are entered with the input:

- NAME, the name assigned to the data consisting of two alphanumeric words (one word in the case of the BESM-6);
- MAT, the material number, used also as searching criterion.

This procedure fulfills the following functions:

Mode 1 - MAT≠0. Starting with the current address of the input element MR(4), the program searches field 1 for data having the name NAME and the material number MAT if MAT > 0, or any other material number if MAT < 0. If such data exist, the address of the beginning of these data is assigned to the MR(4) element, MFIND is given the value 1, and control is returned to the calling data processing procedure. If no more data are found after the search of field 1 is completed, MFIND is given the value 0, control is returned to the originating program, and the data search is terminated.

Mode 2 - MAT=0. Under this condition, the search of field 1 is executed as part of the control data cycle. The location of the UD data is assigned on the basis of the element MR(5) (the address of the current word in field 2). The data whose names correspond to NAME and whose MAT numbers correspond to the identification number given in the heading of the UD data, are searched for in field 1. If the MAT number is equal to 0, the search using the MAT number is not executed. If data are found, then MR(4) is given the value of the address of the PD data block, MFIND is given the value 1, and control is returned to the originating program. After the completion of the search of field 1 for one set of UD data, the address of MR(5) is shifted by the length of the data set, and the search of field 1 starts again from the beginning. After field 1 had been searched to the end of the last of the UD data sets, MFIND is assigned the value 0, MR(1), MR(2) and MR(3) are also assigned the value 0, control is returned to the originating program, and the search is terminated.

In this manner, procedure MFIND can organize the processing of both UD and PD input data clusters using one of two schemes: "in parallel" as described under Mode 1, and "in series" as described under Mode 2.

The <u>DISPAG(NI,NO)</u> procedure makes it possible to redistribute buffers directly from a functional module. The purpose of this option is to be able to increase the transmission capacity of one of the channels by temporarily using an free channel. The parameters of this procedure are:

NI - the number of the channel from which the buffer is removed;

NO - the number of the channel to which the buffer is assigned.

The use of this procedure makes it possible to increase the number of buffers assigned to channel NO to two (provided that they were not used simultaneously before).

The redistribution of buffers is accomplished in the following manner:

- the number of the file dedicated to the NI channel is determined by the exchange register: NTI=LT(NI);
- the number of the file dedicated to the NO - NTO channel is determined by the BSP catalog with the help of the conversion register;
- if the two are equal, NTI=NTO (i.e, the channels already coincide), control is returned; otherwise:
- if the buffer sector for channel NI was used in the recording mode (LR(NI)=3), it is recorded in the BSP library;
- the NTO:LT(NI)=NO file is assigned to channel NI, and the buffer is declared to be "empty": LF(NI)=-1, LR(NI)=0, and control is returned to the originating program

After the requirement in the sector is satisfied, the value of the exchange register is redetermined; this is done with the help of the DISPARG procedure by setting the parameters NO and NI equal to the channel number from which the buffer originates: CALL DISPAG(NO,NI).

Exchange procedure. Direct access to data stored in the BSP library, consisting of input PD and UD data and the results of data processing, provides the exchange procedure with structured data blocks. In the organization of these exchange procedures, it is convenient to identify four levels of operation: from the zeroth to the third level (see Fig. 3.3). It is suggested that the programmers of functional modules use only 3rd level (for the PD data transfer) and 2nd level (for the loading of UD data and exchanging working data blocks) operations. These procedures, consisting of the LOADX family of procedures and of the TRAC and MWTRAC procedures, guarantee an effective means of data transfer and exchange, protected from programming errors with respect to the rules regarding the utilization of the BSP library. In the case of the lower level procedures MWORD and CANAL, there are no built-in controls to adhere to the rules in the utilization of the BSP library; it is therefore not recommended to call on these procedures directly from the functional modules.

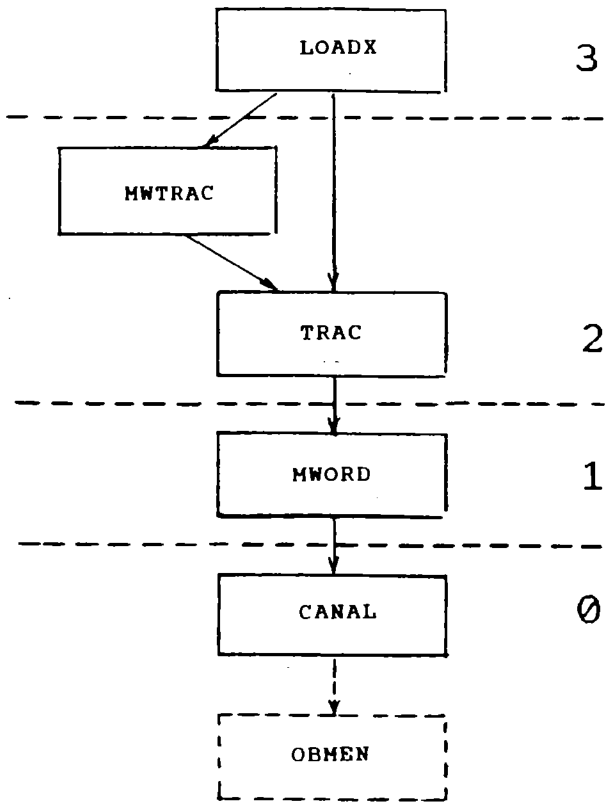

Fig. 3.3 Interactions of Exchange Procedures at Different Exchange Levels

The LOADX family of procedures are designed for the exchange of standard structured data of the type "X" (the data type is defined by the last letter of the procedure name) with the BSP library, consisting of the input and output of the data being processed. Currently, there are procedures for the processing of data of type R, S, U, F, P, A, E and M (see data classifications in Section 1.3). The parameter of this procedure is:

NC - number of the exchange channel and the direction of the exchange (NC=1 or 2 data transfer from the BSP library, NC=3 data transfer into the BSP library).

In order allocate data in the computer memory, it is necessary to define the standard COMMON region corresponding to the data type X, and the unnamed COMMON P(1) region which is the working field of the functional module. The location of data in the BSP library is determined by the value of the current conversion register address.' Loading of data in the computer memory is executed in the following manner: the simple variables and the addresses of the buffers are transferred to the COMMON region of the memory; the buffers themselves are stored one behind the other in the unnamed region of the memory starting at the first free word location. Should the size of a buffer be too large to be accommodated in the working field of a functional module (under normal circumstances singling out such a memory block requiring "individual treatment" does not present any difficulty), it is left in the BSP library; if this occurs during data input, the current location address is shifted by the length of the memory block. After the data have been loaded, the location address will correspond to the location of the memory block in the BSP library. An analogous situation may occur during data input: a data block that is formed directly in the output field of the BSP library shall be left out and the current location address shall be shifted by the length of the data block (a more detailed description of the treatment of large data blocks is given in Chapter 4 of this report).

<u>The TRAC(NC,L,A) procedure</u> serves to transfer information from the data blocks A(L) to the BSP library and vice versa. The input parameters of this procedure are:

±NC	- channel number and direction of the transfer (in the case of channel 3, the + sign is used for data entry, and the - sign is used for the reading/retrieval of data);
L	- the length of the data block;
A(L)	- the name of the data block being transferred.

The location of data in the BSP library is determined by the current location addresses, which correspond to the BSP channel number in the conversion register (element MR(NC+3)) of the COMMON/CATAL/MR(512) region of the memory. After the transfer of the information, the value of the current location address is shifted by MR(NC+3)=MR(NC+3)+L. Thus, if the value of the location address is not changed, each subsequent data block is positioned in the BSP library one after the other.

In the course of its operation, the TRAC procedure uses the MWORD procedure which is designed for the transfer of data sectors. However, its use is implemented when the first element of the shifted information is called; after that, MWORD calculates the number of sectors which were transferred into the buffer, and transfers them to the memory block A (or vice versa). After this, the call for the MWORD procedure is repeated for the transfer of the first sector that did not get into the buffer, and so on until all of the information has been transferred in the specified direction. As a result, the effectiveness of the transfer operation increases as length L increases. During the transfer procedure, a controlling operation is performed in order to ascertain that the location addresses of the shifted information corresponds to the BSP library fields which participated in the transfer. The same diagnostics procedure is used as that used in the MWORD procedure.

The MWTRAC(NCI,L,NCO) procedure serves to transfer information within memory fields allocated by the system to the functional modules. There are four such fields: input (1), control (2), output (3) fields of the BSP library, and the working field of the functional module (0) located in the operational part of the memory. Fields 1 and 2 are used only for reading, and fields 3 and 0 for reading and recording of information. This procedure has the following input parameters:

NCI	- the number of the input channel (reading channel) (NCI=0,1,2,3);
L	- the number of words in the transferred information;
NCO	- the number of the output channel (recording channel) (0,3).

The output parameter of this channel is:

MWTRAC-	the location address of the beginning of the data in the output channel.

The location of the data in the input and output fields is determined by the current location addresses stored in the conversion register (the elements /CATAL/, MR(5), MR(6) and MR(8) for fields 1, 2, 3 and 0 respectively). The location address values for the memory fields participating in the transfer are increased by L. No transfer takes place if NCI=NCO; only the current location address of the NCI=NCO field is changed.

The values of the current location addresses and the proper use of the fields are controlled by the following conditions: the size of the information cannot exceed the size of the dedicated field, and information can be recorded only in fields 0 and 3. If these conditions are not met, the program prints one or the other error message:

```
* * * INFORMATION OUTSIDE BOUNDARIES OF FIELD . . .
* * * RECORDING IN FIELD ... IS NOT ALLOWED
```

The MWORD(NC,MW) procedure makes it possible to access any word in the BSP library fields that is dedicated to the functional modules with due consideration given to their function using a sector-wise mode of transfer.

The input parameters used in this procedure are:

NC - channel number and transfer direction;

NW - location address of the word in the channel for which access is requested.

The output parameter is:

MWORD - the location address of the requested word in the unassigned COMMOM region (in the part of the region that is used by the buffers).

The MWORD procedure performs the following functions on the basis of the values stored in the exchange register:

1) using a given location address MW and the global parameter LREC, the procedure calculates the number of the requested sector NF of the MT=LT(NC) file.

2) using the memory block LF, the procedure checks whether a requested information sector is located in one of the buffers.

Assuming that a requested sector is found in the NB buffer,

LT(NB=NT and LF(NB)=NF

In this case, the address counter (that counts the number of times that this sector has been addressed) is increased by one, that is LC(NB)=LC(NB)+1; if the NB buffer did not take part in the 3rd (recording) channel transfer, the following channel number is registered:

```
IF_(LR(NB).NE.3)_LR(NB)=NC
```

and control is returned to the originating program. Otherwise:

3) the procedure determines whether there is an "empty" buffer; if there is a buffer NB, such that

LT(NB)=NT and LF(NB)=-1

the required sector NF is read into the buffer, it is assumed that LF(NB)=NF and LR(NB)=NC, and the address counter for all buffers that are dedicated to that same file are assumed to be zero; furthermore,

LC(IB)=0, if LT(IB)=NT for IB=1,2,3

the location address of the word in the working field of the MWORD procedure is calculated and the control is returned to the originating program.

4) If all of the buffers are occupied, there is one among them that is dedicated to the NT file, which has been addressed a minimum number of times, thus:

LC(NB)=minLC(IB) and LC(IB)=NT for IB=1,2,3.

If it was implemented in channel 3 (i.e., LR(NB)=3), then it is registered in the BSP library under the number LF(NB); the required sector NF is then read in, it is assumed that LF(NB)=NF, and the address counter is set to zero:

LC(IB)=0 if LT(IB)=NT for IB=1,2,3

the location address of the word in the working field MWORD is calculated, and the control is returned to the originating program.

During the transfer procedure, a controlling operation is performed by the MWORD procedure to ascertain that the MW location address corresponds to the BSP library field which participates in the transfer. If it turns out that the requested word lies outside of the memory boundary, the following message is printed out:

```
THE OUTPUT LIES OUTSIDE OF FIELD n
```

where n is the number of the field in which this event has been registered.

In this manner, the MWTRAC procedure regulates the exchange of data in accordance with the system's rules and conventions, minimizing at the same time whenever possible the number of times the memory is addressed.

The CALL(NC) procedure controls the read-write operations of BSP sectors at the zero level of exchange. BSP sectors are used as units of exchange. The input parameter for this procedure is:

±NC - the exchange channel number with a (+ or -) sign specifying the direction of the transferred information. For the input direction (indicated by the + sign) the following convention is observed: read (output) only for the 1st and 2nd channels, and write (input) for the 3rd channel (where the number of the BSP field that is taking part in the transfer corresponds to the channel number). In order to read data from the output field of the BSP library, the 3rd channel can also be used to transfer data in the opposite direction (using the - sign). The length of the sector (data transfer unit) is a global parameter used throughout the program; the location of a sector in the operational part of the memory and in the other parts of the system is determined by the LW(NC) and LF(NC) buffer elements of the exchange register (in the case of the BESM computer, the sector number O3U-LP(NC) is used as well).

In the ES computer, the reading and writing of BSP sectors is executed with the use of the READ and WRITE commands. On the BESM-6 computer, the data transfer functions are executed by means of the "self-coded" procedure OBMEN [15], by transposing the O3U sector into an MB channel or an ML zone and vice versa, avoiding the use of the system's buffers.

The diagnostics procedure ERROR(NAME,N,I,R) is designed to issue messages to identify the reasons for typical errors. There are two kinds of errors: incidental and fatal. In the case of incidental errors, the procedure prints out the reason for the interruption of the program and then proceeds with the computation. If the number of messages exceeds 5, their output is stopped, and instead, the information on all of the interruptions during any given processing step is printed out at the end of that step. In the case of a fatal error, only one error message is issued, and the computation is stopped.
The parameters of this procedure are:

NAME - the name of the procedure where the error occurred (an alphanumeric constant consisting of 6 bytes).

±N - the number of the error, with an indication as to the type of error (incidental error by a + sign, fatal error by a - sign);

I - integer identifying a given error parameter (see examples below);

R - a number giving the value of an error parameter (see examples below).

Error Messages

```
1. * * *  THE OUTPUT LIES OUTSIDE OF FIELD I
2. * * *  WRITING IN FIELD I IS FORBIDDEN
3. * * *  ITERATION WAS INTERUPTED AT STEP I AT WHICH POINT
          THE ACHIEVED ACCURACY WAS R
4. * * *  ERROR IN THE CONTROL PARAMETERS
5. * * *  CROSS-SECTION OF TYPE I IS NEGATIVE AT POINT R
6. * * *  MODULE IS MISSING
```

This list can be extended to include additional types of errors by adding two statements to the procedure:

```
IF_(NI.EQ7)   PRINT 7
7 FORMAT (1X,/_***_/,/ message text)
```

(the I and R parameters are assigned upon inspection).

4. FUNCTIONAL MODULES

The GRUKON software package is made up of a number of programs designed to accept standardized input; this implies the existence of fixed data formats and well-defined rules in the construction of functional modules which guarantee the reliability of the system. On one hand, such rules would seem to curtail the programmers freedom, and increase the complexity of the programming; on the other hand they permit the use of a number of operations inherent in the module's logic and in the technological process under consideration, thereby reducing the need for time consuming creative effort.

This chapter describes the system "as seen by an insider", such as it would be seen by the programmer of a functional module. It describes the information structure as it is used by the functional module, the programming methods and procedures used in its development, and the typical construction of the principal procedures of functional modules. In conclusion, the methods and procedures used in the development of the functional modules are illustrated in examples using simplified data formats and data processing algorithms.

4.1 The Information Environment of the Module

The module's resources. Three BSP library fields are dedicated to a functional module[7]: an input field (field 1) used for the storage of input PD data, a control data field (field 2) used for the storage of UD data, and an output data field (field 3), used for the storage of data processing results in standard PD format. A BSP library field consists of a region in a memory segment defined by the location address of the first word and the number of words in the set. In each segment, the numbering of words starts with a zero address; the transfer of information proceeds through the buffers in fixed length sectors.

In addition to these BSP fields, a module can use a working field located in the operational part of the memory. The allocation of such a field is executed by the statements

```
COMMON P(1)
DIMENSION M(1)
EQUIVALENCE_(P(1),M(1))
```

The location address of data in a working field is defined by the number of the memory element P(1) or M(1), which marks the location of integer or numerical data respectively. The working field of a module is used to store arrays or sub-arrays of input data, for the storage of auxiliary blocks of data consisting of intermediary calculational results, and for the storage of output data arrays (or sub-arrays) prior to their transfer to the BSP library. Parts of field 3 which do not contain any useful data can also be used as working fields.

Initial information that defines the BSP fields at the outset of a run is stored in the linkage register. This register is accessed through the region COMMON/CATAL/MR(512) (see Section 3.1). The BSP catalog, together with the complete field allocation information pertinent to temporary modules, is also located in that region.

---

[7]This chapter describes data processing modules only. As input-output modules, conversion modules and data reduction modules are modifications of data processing modules, they do not need to be described separately.

The order in which data are called from the BSP fields is determined by the processing rules for PD and UD data clusters: field 1 is foreseen for the sequential buildup of UD data called from field 2, and the PD data which are to be processed are loaded into the operational part of memory. The input PD data can be processed one at a time, in the given order, or in bunches in the order specified by the UD data. The series of data processing operations is terminated after the last UD data set has been processed. Depending on the type of data processing operation, the number of output data will be equal to either the product of the number of input PD parameters and the number of UD data, or to the number of UD data. It is suggested to use procedure MFIND to organize the sequential input of data clusters (see Section 3.3); its use should guarantee the correspondence of the input data with the type of processing; at times however, this is not enough and it is necessary to check the correspondence of the input PD parameters and the UD data explicitely (for instance, to insure the correspondence of the energy intervals specified in the PD and UD data). If the PD parameters do not agree with the processing conditions, they are ignored.

End of module utilization. At the end of the utilization of a module, the values of the current location addresses of fields 1, 2 and 3, MR(4), MR(5) and MR(6) must be reset to point to the first word which follows the last word of the corresponding BSP field, and their values must be set to zero. The control of the program is returned with the use of the RETURN command.

4.2. Organization of Data Transfer

Exchange of data with the BSP library. The exchange of data between the operational memory and the BSP library is performed with the use of the system's exchange procedures (see Section 3.3). The standard quantity parameters in the operational memory are assigned to standard COMMON regions. In the case of PD data, the names as well as the structures of these regions are fixed (see Appendix A); in the case of UD data on the other hand, because of the uniqueness of each processing operation, the structure of these regions must be worked out independently. The names assigned to these regions are constructed by using the name of the main module, with a symbol B added as a prefix: for instance, for the module RXTXS, the name of the COMMON region would be COMMON/BRXTXS.

The order in which the variables and the buffer addresses are stored in the standard COMMON region is the same as that in which they are presented in the standard parameter set (including the block of SP heading data).

When data are loaded, only the simple variables and the buffer addresses are loaded into the COMMON region, the actual buffers are stored in the module's working field or left in the BSP library. When loading data with the LOADX procedure, the last data block, consisting of the first data table, is left in the BSP library because its size is, as a rule, too large to fit into the operational memory. The address of this data block, allocated at the end of the LOADX procedure, actually corresponds to the location of the data block in the BSP library. In such a case, it is the responsibility of the programmer to organize the transfer of data block sections in such a way so as to be most efficient in the data processing operation.

The loading of all logical control data also rests with the programmer. It is suggested that the exchange procedures TRAC (for simple variables) and MWTRAC (for the exchange of data arrays and address allocation) be used for such operations.

In the data processing operation, the initial table of output data is built up in sections. As they are generated, these sections are stored in specially reserved space in field 3 of the BSP library. After all of the sections of the table have been entered and its size has been determined, the data table heading, the other defining parameters, and the output data array are formulated. They are loaded by means of the LOADX procedure which stores all of the data blocks, except the initial data table which is already located in the BSP library (instead of storing that table, a gap is introduced and the register element MR(6) is increased by an increment equal to its length). After the set of input PD and UD data has been processed and the results stored, the current location addresses of the BSP library fields must be reset to point to the first word which follows the stored data, and the address of the working field must be set to equal to the value it had at its initial entry point. The program can then proceed with the processing of the next data section.

Moving data within the module. As a rule, a data processing module is made up of a few procedures. For instance, it is convenient to separate linkage functions from calculational functions; the former are usually the same in all modules and are constructed using the same logic, and the latter are specific for each module and are often modified. The degree to which a module can be structured depends on the complexity of the processing, that is, on the extent to which general procedures and utilities can be used in its execution, as well as on other factors. Aiming at a standardized procedure design, the following method is proposed to be adopted for the transfer of information from one procedure to another. All input data arrays are stored in the working field of a functional module one behind the other starting at the location of the first free word (the address of its location is stored in the MR(8) register element). These blocks are followed by space allocated for the output data arrays. At the same time, the value of the MR(8) register element is increased by the length of these data blocks in accordance with its function. Parameters consisting of simple variables and addresses of data blocks are combined in the COMMON region by the calling procedure, and its name is modified by prefixing the letter B to the name of the procedure. In this manner, the basic information is transmitted through the COMMON region whose structure is determined by the requirements of the calling procedure. The names of the procedure parameters are suggested to be used primarily as mnemonics, in order to make it easier to remember their function. It is therefore desirable to choose a parameter name which characterizes the principal function of the computation.

It is possible that additional memory blocks may be needed by the procedure for the execution of the computation. In such a case, before storing the various memory blocks in the working field one behind the other, it is necessary to record the content of the MR(8) register element in the beginning of the procedure. Should it be necessary to change to a lower level procedure, a COMMON region is created by the calling procedure, its input and output memory blocks are assigned in the working field, and the MR(8) register element is increased by the length of the memory blocks, etc... (see above). When returning control to the higher level procedure, it is mandatory to reinstate the initial entry point value of the MR(8) register element. In the process of allocating the memory blocks in the working field, it is necessary to control the extent to which it is filled: that is, the condition MR(8)≤MR(7) must be satisfied. If this condition is not satisfied, it is necessary to change over to a more economical utilization of the working field required in the computation (for instance, by decreasing the size of the data arrays, or by making provisions for space in field 3 of the BSP library, etc...). Alternatively, if such measures are not taken, the program can be designed to issue an error message (using the ERROR procedure) and interrupt the computation process.

4.3. Methods Using Addresses

Arrays of numeric variables and integer data.
With the scheme describe above, it is possible to allocate the working field of a module in a dynamic manner by reducing all limitations imposed on the length of a memory block to one common parameter, namely to the length of the working field. The price one pays for this is in the loss of clarity in working with memory blocks, and it becomes necessary to deal with addresses. In order to compensate for this shortcoming it is suggested to use the names of corresponding memory blocks in the naming of addresses by adding the prefix letter M to their names. For instance, the address of the memory block A would be MA; respectively, reference to the element A(I) would have to be replaced by P(MA+I-1). The procedure is the same in the case of integer arrays, except that instead of array P(1) one must refer to the array M(I) with which it was joined.

Operations with Double Precision Arrays.
The use of double precision causes certain difficulties. At this stage of the program development, the following method is used to overcome these difficulties. After creating a double precision array it is entered in the working field using the statements:

```
                    COMMON P(1)
                    DOUBLE PRECISION_DP(1)
                    EQUIVALENCE_(P(1),DP(1)
```

In order to allocate the double precision array in the working field, the MR(8) address is changed so as to account for the double length of the word, namely: MD=((MR(8)-1)/2)+1. The element D(I) of the double precision array is handled in the usual manner, namely DP(MD+I-1).

The address of the first free word in the working field following the double precision array of length LD is determined by using equation MR(8)=2*(MD+LD)-1.

4.4. The Functions of Main Procedures

In each functional module there is a procedure that is responsible for the interaction of the module with the system, and whose functions are determined to a large extent by these interactions rather than by the more specific execution of data processing operations. For instance, each functional module must retrieve all of the initial data and control parameters from the BSP library, and transfer the resulting data to the designated part of the linkage register, regardless of its assigned data processing objective and independent of the structure of the data. It is therefore possible to develop for this part of the module one programming scheme, common to a group of modules designed to perform similar functional operations, and to devote the rest of the programming to the technological objective of the module. It is therefore natural to locate the functions of the module devoted to its interaction with the system in the main procedure of the module.

At the present time, two separate schemes are used in the programming (of the main functional procedures) of data processing modules. The first scheme assumes that the input data are independent of each other and can therefore be processed independently. In this case, the required functions must be executed in the following order.

1. Record the address of the beginning of the working field, element MR(8).

2. Call mode 2 of the procedure function MFIND. If its value is equal to zero, then execute the RETURN command which stops the data processing operation. Otherwise:

3. Call procedure LOADX for the first channel and load the PD input data parameters into the operational part of the memory (in this operation, the initial data table is left in the BSP library).

4. Using the exchange procedures, enter the UD data in the second channel: groups of simple variables located in the COMMON region together with the control data, are read in from the BSP library using the TRAC procedure. The blocks of parameters are stored in the working field and are simultaneously assigned addresses using the procedure function MWTRAC.

5. Perform a check to ascertain whether the PD data parameters satisfy the conditions given by the UD data. If the input data do not satisfy these conditions, change the value of the MR(8) element to its original value and return to point 2. Otherwise:

6. Proceed to formulate the output format: determine the values of the simple variables on the basis of the input PD and UD parameters and store the data arrays in the working field (when possible together with the input data arrays).

7. Define the COMMON region for the procedure designed to execute the algorithmic part of the data processing operation: determine the values of the simple variables, the addresses of the input data arrays and allocate space on the scratch file for the storage of output data arrays.

8. Determine the value of the MR(8) element corresponding to the beginning of the scratch file region not occupied by data.

9. Record the address of the channel for field 3 data (value of the MR(6) element).

10. Determine the current addresses in the first and third channels corresponding to the beginning of the input and output data tables (MR(4) and MR(6) registers).

11. Execute the data processing operation by reading from field 1 one set of data at a time required for the execution of the computational step and store the resulting data in the space allocated in field 3 as they are generated.

12. If necessary, complete the determination of the output parameter formats.

13. Determine the value of the current address in channel 3 (MR(6) register) of the location corresponding to the beginning of the data and complete the writing of the output format parameters in the third BSP field using the LOADX procedure.

14. Confirm the agreement of the value of the current channel 1 address with the end of the data field (the MR(4) value

must point to the word which follows the last word of the input PD data).

15. Restore the initial value of the MR(8) register, and return to point 2.

The second scheme is implemented in those cases when there is a requirement for a simultaneous processing of more than one input PD data. This scheme is implemented by the execution of the following steps:

1. Record the scratch file channel address MR(8).

2. Determine the upper boundary of the BSP field which holds the control data (field 2).

3. Determine whether all of the UD data have been read in by comparing the current field 2 address with the upper boundary. If all data have been read in, set MR(1)=MR(2)=MR93)=0, and execute the RETURN command. Otherwise:

4. Search field 1 using the mode 1 procedure function MFIND and record the addresses of the input PD data which must be processed; retrieve the parameters needed for the formulation of the output format parameters.

5. Formulate the output format parameters.

6. Record the addresses of the location corresponding to the beginning PD data field in field 3 (MR(6)) and determine the current address MR(6) of the beginning of the initial output data table.

7. Read data from field 1 in sets as required for their processing, execute the processing operation, and store the results in field 3 as they are generated.

8. If necessary, complete the determination of the output parameter formats (e.g., the length of the main data table).

9. Determine the current field 3 address (MR(6)) corresponding to the beginning of the output PD data array; complete the formulation of the headings and parameter designations of the main data table with the help of a dedicated procedure.

10. Determine the current address of the end of the input data field in field 1 (MR(4)).

11. Restore the initial value of the MR(8) register and return to point 3.

4.5. Convertibility of the Programming Language

The major part of the GRUKON program is written in a hybrid version of the FORTRAN languages existing on the BESM-6 and ES computers. As a rule, it is possible to achieve complete parity in the coding of functions for these types of computers. Certain differences may come up in file handling procedures or in the use of system names for control data (such as in the conversion of UD data). The following convention has been adopted: that part of the coding which is machine dependent is written in two different versions and is written within parentheses:

```
C(ES
    coding for the ES computer
C)ES

C(BESM
    coding for the BESM-6 computer
C)MESM
```

This gives the possibility to run the program on either one of the two machines using normal means available to programmers.

4.6. Data Processing Example

Let us see how the rules and conventions described above apply to a typical GRUKON data processing problem: namely, the calculation of the detailed cross-section dependence from resolved resonance parameters. In order to be able to solve this problem in the framework of this report, the problem has to be simplified. The nature of this simplification can be visualized by comparing the hypothetical data structure *R* and *S* with the actual data given in Appendix 1. The data processing algorithm is simplified to a greater extent. Nevertheless, it is possible to maintain the basic features of the functional module.

Description of the Problem.
The standard format *R* designed for the storage of resolved resonance parameters has been fixed within the system:

NL	- number of resonances
NR	- number of types of reactions
LR(NR)	- list of reaction types

EL	- lower boundary of the energy interval within which the cross-section is calculated
EH	- upper boundary of that interval
C	- coefficient
E_l,G_l(NR)	- table of resonance parameters: resonance energies E_l for l=1,...NL and resonance widths Γ_{lr}; $\Gamma_{l1}=\Gamma_{ln}$; r=1,...NR.

The data are assigned to the following standard common region:

COMMON/RRR/MRRR,NL,NR,MLR,EL,EH,MELGL

The requirement of this problem is to formulate the functional module which would calculate the cross-section for given energies using the calculational model described by the following equation:

$$\sigma_r(E) = \frac{C}{E}\sum_{l=1}^{NL}\frac{\Gamma_{ln}\Gamma_{lr}}{\Gamma_l^{I}}\frac{1}{1+(E-E_{lr})^2}$$

$$EL \le E \le EH$$

$$\Gamma_l = \sum_{r=1}^{NR}\Gamma_{lr}$$

The cross-section data parameters are not provided by the system.

Formulation of the data structures.

On the basis of the requirements of the problem, it is necessary to develop two standard structures: one for the cross-sections and the other for the processing control parameters. The first set of data fall into the PD data category and requires additional LOADX type procedures. This set of data is given the system name *S*. The data set structure is defined in accordance with the data representation rules using the established standard data formats:

NC	- number of cross-section types,
LC(NS)	- list of cross-section types,
E	- energy value,
S(NS)	- set of cross-section values,

which will be assigned to a standard memory region

COMMON/SSS/MSSS,NS,MLS,E,MS

(where MSSS is the memory block address of the display title).

The conversion of the data given in the *P* format to the *S* format is done by supplying only one parameter, namely the value of the energy E. Having done this conversion, the program then assigns the system's name *R/E-S which also serves as the name of the corresponding UD data. The name assigned to the main procedure of the module is RXEXS, and the name and structure of the COMMON region for the UD data is: COMMON/BRXEXS/MRXEXS,E.

Program Execution.

Two types of procedures can be identified in a functional module:

- the main procedure, responsible for the preprocessing of the input data, and the formulation of the output data representation (SUBROUTINE RXEXS); and

-the computational procedures composed of algorithms designed to calculate cross-sections from resonance parameters (SUBROUTINE SIGMA(E)).

In accordance with the established rules, a COMMON region is established to facilitate the exchange of data

```
            COMMON/BSIGMA/NL,NG,MLR,C,MELGL,MS
```

(the nomenclature used corresponds to the description of the data given above)

In the given example, there is a need to write a procedure for the exchange of data in the standard *S* format:

```
            SUBROUTINE LOADS(NC)
```

(where NC is the number of the exchange channel).
The coding of the procedures is given in Appendix 2.

Inclusion of the Module in the Program Package.

The module *R/E-S can be included in the BESM-6 version of the GRUKON program by adding it to the existing modules stored in the subroutine library of the program. To include it in the ES computer version, it is necessary to edit the PROCES procedure by adding the following statements:

```
              REAL*8   ARXEXS
              DATA ARXEXS/ '*R/ E-S'/
              . . . . . . . . . . . . .
              IF_(M.EQ.ARXEXS)_CALL_RXEXS
```

REFERENCES

1. NIKOLAEV, M.N., Nuclear Data for the Calculation of Fast Reactors, Problems in Atomic Science and Technology. Ser. Nuclear Constants 8(1) (1972) 3 (in Russian).

2. ABAGYAN, L.P., NIKOLAEV, M.N., SINITSA, V.V., Program "MUF" for the Multi-Level Calculation of Cross-Sections from Resonance Parameters. Moscow, Atomizdat, Nuclear Physics Research in the USSR, Computer Program Descriptions Vol. 15 (1973) 38 (in Russian).

3. ABAGYAN, L.P., NIKOLAEV, M.N., Computer Program for the Calculation of Cross-Sections in the Unresolved Resonance Region. Moscow, Atomizdat, Nuclear Physics Research in the USSR, Computer Program Descriptions Vol.15 (1973) 32 (in Russian).

4. SINITSA, V.V., Computer Program for Averaging Cross-Sections. Moscow, Atomizdat, Nuclear Physics Research In the USSR, Computer Program Descriptions Vol. 15 (1973) 40 (in Russian).

5. NIKOLAEV, M.N., BAZAZYANTS, N.O., Computer Program "UMBLOK" for the Calculation of the Scattering Cross-Section Angular Distribution Parameters and of Transmission Functions. Moscow, Atomizdat, Nuclear Physics Research in the USSR, Computer Program Descriptions Vol. 14 (1973) 43 (in Russian).

6. BAZAZYANTS, N.O., STAROSTENKA, M.B., Computer Program "MANNERS" for the Calculation of Multi-Group Matrices of Elastic and Inelastic Neutron Scattering Cross-Section Angular Distribution Parameters with Self-Shielding. Moscow, Atomizdat, Nuclear Physics Research in the USSR, Computer Program Descriptions Vol. 15 (1973) 45 (in Russian).

7. SINITSA, V.V., et al., GRUKON, A Library of computer Programs for the Calculation of Multi-Group Data. Moscow, Atomizdat, Nuclear Physics Research in the USSR Vol. 27 (1979) (in Russian).

8. SINITSA, V.V., The SOKRATOR System: Subsystem GRUKON. General Outline of the Programming System. Moscow, TsNIIatominform, Military Technology and Economics. General Technology Ser. Number 12 (1978) (in Russian).

9. SINITSA, V.V., The GRUKON Computer Code Package. Part 1. The Data Processing Program. Obninsk, Preprint FEI-1188 (1981) (in Russian).

10. SINITSA, V.V., The GRUKON Computer Code Package. Part 2. The Control Data. Obninsk, Preprint FEI-1189 (1981) (in Russian).

11. SINITSA, V.V., The GRUKON Computer Code Package. Part 3. Modification of the Input Command Language and Extension of the Possibilities to Process Tabulated Cross-Sections. Obninsk, Preprint FEI-1332 (1982)(in Russian).

12. SINITSA, V.V., The GRUKON Computer Code Package. Part 4. Calculation of the Resonance Self-Shielding Cross-Sections. Obninsk, Preprint FEI-1429 (1982)(in Russian).

13. SINITSA, V.V., RINEISKIJ, A.A., BULEEVA, N.N., The GRUKON Computer Code Package. Part 5. The Calculation of the Subgroup Parameters of the Resonance Structure of the Cross-Section. Modification of the Program to Run on the ES Computer. Obninsk, Preprint FEI-1666 (1985) (in Russian).

14. YEVSTIFEEV, V.V., RINEISKIJ, A.A., The GRUKON Computer Code Package. Utilization of Computer Graphics. Obninsk, Preprint FEI-1747 (1985) (in Russian).

15. ZININ, A.I., SUSLOV, I.R., Exchange Programs and Unformatted Data Input. Obninsk, Preprint FEI-1082 1980 (in Russian).

APPENDIX 1. Structure of Processed Data.

R Parameter Set : Resonance Parameters.

NFORM	Formula type
	NFORM1 = Single level Breit-Wigner formula
	NFORM2 = Multi-level Breit Wigner formula
	NFORM4 = Adler-Adler formula
	NFORM5 = Reich-Moore formula for non-fissile nuclei

NU	Number of levels
NL	Number of resonances
NR	Number of reaction types
LR(NR)	List of reaction types
LC(NU)	Array of orbital momentum values
LF(NF)	Data determining the means of including the contribution of far-lying resonances (NF=NR if NFORM=4, otherwise NF=NU)
LP(NR)	Number of resonance parameters for each reaction
LNU(NL)	Array of level values
EL,EH	boundaries of energy region (eV)
EPS	Data accuracy (in relative units)
A	Nuclear mass (normalized to C12)
B	Nuclear concentration (in relative units)
SI	spin of target nucleus
R	Channel width (FM)
QR(NR)	Array of reaction energy values
SJ(NU)	Array of total momenta
RC(NU)	Array of effective scattering radii (FM)
RF(NRF)	Array of background parameters (NRF is determined from the values of LF)
TAB(EL,GL(IP),IP=1,NRP),IL=1,NL)	Tables of resolved resonance parameters: EL=resonance energy, GL=resonance width vector or Adler-Adler parameters (where NRP is the vector length determined from the LP array)

U Parameter Set : Average Resonance Parameters.

NFORM=N1,N2,N3,N4	Flags with following meanings: N1 = flag number which determines the nature of the calculated parameters (1 - cross-section parameters, 2 - transmission function) N2 = inclusion of resonance width fluctuations (1 - fluctuation included, 2 - not included) N3 = inclusion of fluctuations of distance between levels (1- included, 2- not included) N4 = model used for the resonance cross-section calculations (1 - single level Breit-Wigner, 2 - multi-level Breit-Wigner, 7 - equal equidistant isolated resonances model, 8 - equal equidistant isolated resonances model corrected for interference between levels)
NR	Number of reactions
NE	Number of energy points in the MERDG table
INT	Number of the energy interpolation rule
LC	Value of the orbital momentum
LR(NR)	List of reaction types
LO(NR+1)	List of statistical distribution laws (0 - does not fluctuate, 1 - Porter-Thomas law, LO is the number of degrees of freedom, 2 - Wigner law)

EL,EH	Lower and upper energy boundaries
EPS	data accuracy
A	Atomic weight of the target nucleus (normalized to C12)
B	Nuclear concentrations in the medium (in relative units)
SI	Spin of target nucleus
SJ	Value of total momentum
RC	Channel radius (in FM units)
PD(NP)	Array of statistical distribution parameters (NP is determined from the LO array)

TAB(E,R,D,(GR(IR),IR=1,NR),(E=1,NI) Table of the energy dependence of unresolved resonance parameters; the following parameters are given as a function of energy E: R - effective scattering radius, D = average distance between levels, GR - resonance width vector

***S* Parameter set : Cross-Section Tables**

NEP	Number of energy points
NS	Number of reaction types
NT	Number of different temperatures
NINT	Number of the energy interpolation scheme
LS(NS)	List of the cross-section types
EL,EH	Lower and upper energy region boundaries
EPS	Data accuracy (relative units)
A	Nuclear mass (normalized to C12)
B(NS)	Reaction energy (eV)
T(NT)	Temperature array

TAB(E,((S(IT,IS), IT=1,NT),IS=1,NS),(E=1,NEP)
Table of cross-sections as a function of energy E

***A* Parameter Set : Angular Distributions of Secondary Neutrons**

LS	Reaction type

LTT	Type of data presentations
	LTT=0 isotropic scattering
	LTT=1 Legendre polynomial coefficients
	LTT=2 tabular presentation

LCT	Frame of reference
	LCT=1 laboratory
	LCT=2 center of mass

LRFLAG	Flag identifying the particle x in the (n,nx) reaction
NK	Number of elements in the transfer matrix
NINT	Number of the energy interpolation scheme
NE	Number of energy points at which data are given
N(NE)	Expansion series, or the number of cosines given at each energy point
EL,EH	Lower and upper energy boundaries
EPS	Accuracy (relative units)
AWR	Nuclear to neutron mass ratio
O	Reaction energy yield
V(NK)	Transfer matrix elements
AD	Array of angular data, format depends on the LTT value: LTT=0 the array is not used LTT=1 ((E,A(L),L=1,N(J)),J=1,NE)) LTT=2 ((E,MU,P(MU))(I),I=1,N(J)),J=1,NE) E=energy, A(L)=expansion coefficients, MU=cosine of scattering angle, P(MU)=probability function

E Parameter Set : Energy Distribution of Secondary Neutrons

LS	Reaction type
LF	Type of presentation
	LF=1 tabular
	LF=5 general evaporation spectrum
	LF=7 simple fission spectrum
	LF=9 evaporation spectrum
	LF=11 Watt spectrum with energy dependent parameters
NP	Number of energy points for which the "distribution fraction" is given
NE	Number of energy points for which data are given
NX	Number of points available for additional tabular dependence (NT is not equal to 0 for LF=5 and LF=11)
INT	Energy interpolation scheme
INX	Interpolation scheme used for the additional tabular dependence
NF(NE)	Number of points in each table for each energy (used only with LF=1)
EL,EH	Energy interval
EPS	Accuracy (relative units)
AWR	Nuclear to neutron mass ratio
U	Minimum energy loss in scattering event
D(NP)	Distribution fraction data, D=(E,P(E))
ED	Scattering data array:
	LF=1 ((E,(E′,P(E′))(I),I=1,NF(J)),J=1,NE)
	LF=5 (E,T(E))(NE),(X,G(X))(NX)
	LF=7 (E,T(E))(NE)
	LF=9 (E,T(E))(NE)
	LF=11 (E,A(E))(NE),(X,B(X))(NX)
	E=energy, E′=energy after scattering, P(E′)=probability function

F Parameter Set : Cross-Section Attributes

NFUN	Type of attribute (1 - cross-section moment, 2 - transmission function
NG	Number of groups or energy points
NINT	Energy interpolation scheme
NT	Number of individual temperatures
NS	Number of cross-sections
NP	Number of real parameter attributes
NL,NH	Boundaries indicating a change of an attribute parameter
LS(NS)	List of cross-sections
EL,EH	Boundaries of energy region (eV)
EPS	Data accuracy (relative units)
A	Nuclear mass (normalizes to C12)
T(JT)	Temperature array
P(NP)	Array of real parameter values
TAB(RNG,ELG,EHG,SWG,((((((F(IN,IS,IT,IP),IN=NL,NH),IS=1,NS),IT=1,NT),IP=1,NT),IP=1,NP),IG=1,NG)	Table of attributes: RNG=FLOAT(number of group), ELG and EHG are the group boundaries, SWG is the weight function integral, F(NN,NS,NT,NP) are the values of the cross-section attributes(NN=NH-NL+1)

***P* Parameter Set : Subgroup Parameters**

NG	Number of groups
NP	Number of subgroups
NS	Number of cross-sections
NT	Number of individual temperatures
LS(NS)	List of cross-section types
EL,EH	Boundaries of energy region (eV)
EPS	MAX(EPSN) achieved approximation accuracy (relative units)
FL,FH	Boundaries of the region where the real parameter of the attribute changes (T.N.:the "region" refers to the region for which the subgroup approximation is being calculated)
A	Atomic weight (normalizes to C12)
T(NT)	Temperature values
TAB(RNG,ELG,EHG,SWG,(EPSN,(AN(INT,((SN(IT,IS,IN),IT=1,NT),IS=1,NS),IN=1,NN=1,NM),IG=1,NG):	Subgroup table: RNG=FLOAT(GROUPS); ELG,EHG=group boundaries; SWG=weight function integral; EPSN=accuracy of the approximation of the attribute in the indicated region where there is a parameter change for the NN number of subgroups; AN,SN(NT,NS)=fractions and subgroup cross-sections; two format modifications can be implemented: Mode 1 - NP>0: only the maximum number of subgroups are listed (NN=NP,NM=1), Mode 2 - NP<0: only those subgroups for which NN=1,2,,,NM (where NM=IABS(NP) are listed.

***D* Parameter Set : Neutron Scattering Probability**

NW	Flag determining which spectrum is going to be used in the data averaging operation
LS	Reaction Type
NG	Number of groups
NL	Number of Legendre coefficients to be used
NB	Number of weight functions
MPC(NPC)	Array of the NT,NGS,IGLO parameters, where NT=the point number, NGS=number of groups into which scattering occurs, IGLO=lowest number of the NGS groups
EL,EH	Energy interval
EPS	Accuracy
AWR	Nuclear to neutron mass ratio
T	Number of energy points in the tabulated spectrum
EG(NG(1)	Array of group boundaries
SPECT (E,S)(IFIX(T))	- Tabulated spectrum
TAB(E,(((P(IB,IL,IG),IB=1,NB),IL=1,NL),IG=1,NGS))	- Tabulated scattering probabilities as a function of energy

***M* Parameter Set : Neutron Scattering Matrix**

NW	Flag determining which spectrum is going to be used in the data averaging operation
LS	Reaction type
NG	Number of groups
NL	Number of Legendre expansion elements

NB	Number of weight functions
MPC(NPC)	Array of NG,NGS,IGLO parameters, where NG=group number, NGS=number of groups into which scattering occurs, IGLO=lowest number of the NGS groups
EL,EH	Energy interval
EPS	Accuracy
AWR	Nuclear to neutron mass ratio
T	Temperature
EG(NG+1)	Array of group boundaries: TAB(R(NB*2-1), ((((P(IB,JB,IL,IG),IB=1,NB),JB=1,NB), IL=1,NL), IG=1,NGS)(NFO) - Tabulated integrals for the weight function R and elements of the scattering probability matrix for scattering from that group.

```
C
      SUBROUTINE RXEXS
C*****************************************************************
C     MODULE *R/E-S CALCULATES THE CROSS-SECTION AT A GIVEN
C     ENERGY POINT FROM GIVEN RESONANCE PARAMETERS
C*****************************************************************
C *** MODULE REGISTERS
      COMMON/CATAL/MR(512)
C *** WORKING FIELD OF THE MODULE
      COMMON P(1)
      DIMENSION M(1)
      EQUIVALENCE (P(1),M(1))
C *** INPUT DATA *R*
      COMMON/RRR/MRRR,NL,NR,MLR,EL,EH,C,MELGL
C *** CONTROL DATA
      COMMON/BRXEXS/MRXEXS,EC
C *** OUTPUT DATA *S*
      COMMON/SSS/MSSS,NS,MLS,E,MS
C *** ADDITIONAL DATA
      COMMON/BSIGMA/NLB,NRB,CB,MELGLB,MSB
C *** SYSTEM NAMES
      DATA MRM/''*R*''/,MSM/''*S*''/
C *** ADDRESS OF THE BEGINNING OF THE WORKING FIELD
      MR8=MR(8)
   10 MR(8)=MR8
C *** EXECUTE DATA SEARCH
      IF (MFIND(MRM.0),EQ,0) RETURN
C *** DATA ARE RETRIEVED
C *** LOAD INPUT DATA
      CALL LOADR(1)
C *** LOAD CONTROL DATA
      MRXEXS=MWTRAC(2,6,0)
      CALL TRAC(2,1,EC)
C *** CHECK CORRESPONDENCE OF *R* DATA PARAMETERS
      IF(EC,LT,EL,OR,EC,GT,EH) GO TO 10
C *** FORMULATE OUTPUT DATA ARRAY
C *** ARRAY OF INTEGERS
      NS=NR
      MLS=MLR
C *** ARRAY OF NUMERIC VARIABLES
      E=EC
C *** DATA HEADING
      MSSS=MRRR
      M(MSSS)=MSM
      M(MSSS+4)=1+NS
      M(MSSS+5)=1+NS
C *** FORMULATE /BSIGMA/ ARRAY
      NLB=NL
      NRB=NR
      CB=C
C *** ALLOCATE WORKING FIELD MEMORY TO DATA BLOCKS
      MELGLB=MR(8)
      MSB=MELGLB+NR+1
      MR(8)=MSB+NS
C *** CHECK IF THERE IS ENOUGH ROOM IN THE WORKING FIELD FOR THE
C     DATA BLOCKS
      IF(MB(8),GT,MB(7)) CALL ERROR('RXEXS',-1,0,0,)
C *** DETERMINE THE CURRENT ADDRESSES OF THE 1ST AND 3RD FIELDS
C     OF THE BSP LIBRARY ASSIGNED TO THE BEGINNING OF THE ELGL
```

```
C       AND S BLOCKS
        MR(4)=MELGL
        MR6=MR(6)
        MR(6)=MR(6)+8+NS
C ***  EXECUTE THE CALCULATION
        CALL SIGMA(EC)
C ***  DETERMINE THE CURRENT ADDRESS OF THE BEGINNING OF THE DATA
C       ARRAY IN THE S FIELD
        MR(6)=MR6
C ***  RECORD THE RESULTS
        CALL LOADS(3)
C ***  PROCEED WITH THE SEARCH FOR THE NEXT DATA
        GO TO 10
        END
C
        SUBROUTINE SIGMA(E)
C
        *****************************************************
C       CALCULATION OF THE CROSS-SECTION AT ENERGY E
C       *****************************************************
        COMMON P(1)
        COMMON/BSIGMA/NL,NR,C,MELGL,MS
C ***  PRE-CALCULATE THE COEFFICIENTS
        CE=C/E
C ***  ASSIGN ZEROES TO THE CROSS-SECTION BLOCK
        DO 10 IR=1,NR
   10   P(MS*IR-1)=0
C ***  SUM UP ALL LEVELS
        DO 100 IL=1,NL
C ***  RETRIEVE THE IL LEVEL PARAMETERS FROM THE BSP LIBRARY
        CALL TRAC(1,NR+1,P(MELGL)
C ***  LEVEL ENERGY
        EL=P(MELGL)
C ***  NEUTRON WIDTH
        GN=P(MELGL+1)
C ***  CALCULATE TOTAL WIDTH
        GT=0
        DO 20 IR=1,NR
   20   GT=GT+P(MELGL+IR)
C ***  CALCULATE THE COMMON COEFFICIENT
        X=2,*(E-EL)/GT
        SE=CE*GN/GT**2/(1,*X**2)
C ***  CALCULATE THE REACTION CROSS-SECTION
        DO 30  IR=1,NR
   30   P(MS+IR-1)=P(MS+IR-1)+SE*P(MELGL+IR)
  100   CONTINUE
C ***  ENTER THE CROSS-SECTION ARRAY INTO THE BSP LIBRARY
        CALL TRAC(3,NR,P(MS))
        RETURN
        END
C
C
        SUBROUTINE LOADR(NC)
C       ***********************************************************
C       PROCEDURE TO EXCHANGE *S* DATA WITH THE BSP LIBRARY
C       PARAMETER SET *S* CROSS-SECTION VECTOR AT POINT
C       NS - NUMBER OF REACTION TYPES
C       LS(NS) - LIST OF REACTION TYPES
C       E - ENERGY VALUE
C       S(NS) - ENERGY VALUES ARRAY
C       ***********************************************************
```

```
      COMMON P(1)
      DIMENSION M(1)
      EQUIVALENCE (P(1),M(1))
      COMMON/SSS/MSSS,NS,MLS,E,MS
      IF(NC,EQ,3) GO TO 10
C *** READ IN DATA USING CHANNEL NC
      MSSS=MWTRAC(NC,6,0)
      CALL TRAC(NS,1,NS)
      MLS=MWTRAC(NC,NS,0)
      CALL TRAC(NC,1,E)
C *** TRANSMIT CROSS-SECTION TABLE
      MS=MWTRAC(NC,NS,NC)
      RETURN
C *** RECORD DATA
   10 CALL TRAC(NC,6,M(MSSS))
      CALL TRAC(NC,1,NS)
      CALL TRAC(NC,NS,M(MLS))
      CALL TRAC(NC,1,E)
C *** TRANSMIT CROSS-SECTION TABLE
      MS=MWTRAC(NC,NS,NC)
      RETURN
      END
C
C
      SUBROUTINE LOADR(NC)
C     ***********************************************************
C     PROCEDURE TO EXCHANGE *R* DATA WITH THE BSP LIBRARY
C     PARAMETER SET *R* - RESONANCE PARAMETERS
C     NL - NUMBER OF RESONANCES
C     NR - NUMBER OF REACTION TYPES
C     LR(NR) - LIST OF REACTION TYPES
C     EL - LOWER ENERGY INTERVAL BOUNDARY
C     EH - UPPER ENERGY INTERVAL BOUNDARY
C     C - CONSTANT
C     E,GL(NR) - TABLE OF RESONANCE ENERGIES AND WIDTHS
C     ***********************************************************
      COMMON P(1)
      DIMENSION M(1)
      EQUIVALENCE (P(1),M(1))
      COMMON/RRR/MRRR.NL,NR,MLR,EL,EH,C,MELGL
      IF (NC,EQ,3) GO TO 10
C *** READ IN DATA FROM THE BSP LIBRARY
      MRRR=MWTRAC(NC,6,0)
      CALL TRAC(NC,2,NL)
      MLR=MWTRAC(NC,NR,0)
      CALL TRAC(NC,3,EL)
C *** TRANSMIT TABLE *R*
      MELGL=MWTRAC(NC,NL*(NR+1),NC)
      RETURN
C *** RECORD DATA IN THE BSP LIBRARY
   10 CALL TRAC((NC,6,M(MRRR))
      CALL TRAC(NC,2,NL)
      CALL TRAC(NC,NR,M(MLR))
      CALL TRAC(NC,3,EL)
C *** TRANSMIT TABLE. *R*
      MELGL=MWTRAC(NC,NL*(NR+1),NC)
      RETURN
      END
```

EXAMPLE OF A PROBLEM SOLUTION AND OUTPUT OF RESULTS

```
*****************************************************************
*                   DATA PROCESSING PROGRAM                    *
*****************************************************************
*                                                               *
***************          ENTER CONTROL DATA     *****************
*                       (DATA CLUSTER *2/E-5)                  *
*                          ,65,1,*R/E-S                         *
*                                                               *
***************          ENTER INPUT DATA *R*   *****************
*                          ,65,10,*R*                           *
*                                                               *
***************          PRINT BSP CATALOG     ******************
*                            1,,,                               *
*                                                               *
***************            PRINT RESULTS       ******************
*                             20,66                             *
*                                                               *
***************          COMPLETE PROCESSING    *****************
*                            ,,,END                             *
*****************************************************************

  MODULE *INPUT WAS CALLED AT 0.0 SEC

   *R/E-S; E=1.E3
   *R/E-S; E=5.E3
   *R/E-S; E=10,E3

  MODULE *INPUT WAS CALLED AT 0.123 SEC

    *R*; NL=2, NR=2,LR=2,102
    EL=1.E3, EH=10.E3
    C=2.61E6
    ER=1.E3,GN=90., GG=10
    ER=1.E4,GN=90., GG=10

  MODULE *R/E-S WAS CALLED AT 0.253 SEC

*****************************************************************
*     N     *    DATA NAME     *       DATA ADDRESS (K=1611)     *
*****************************************************************
*     1     *    *R/E-S        *   1    0K+   0     0K+    21   *
*    10     *    *R*           *   1    0K+  21     0K+    19   *
*    20     *    *S*           *   1    0K+  40     0K+    36   *
*****************************************************************

  MODULE *OUTPU WAS CALLED AT 0.357 SEC

           *S*               0           0
                  2          2         102
        1.00000+3   2.11416+3   2.34907+2

           *S*               0           0
                  2          2         102
        5.00000+3   1.08333-1   1.20370-2

           *S*               0           0
                  2          2         102
        1.00000+4   2.11416+2   2.34907+1
```