AN ENGINEERING APPROACH TO PROBLEM ANALYSIS

Alan H. Morgan

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

May 1981

ABSTRACT

AN ENGINEERING APPROACH TO PROBLEM ANALYSIS

Alan H. Morgan

Since the early 1950's, engineers have relied more and more upon the computer for design and analysis. Industry's conservatism however, has dictated acceptance of formal languages barely advanced from those early days. Engineers seeking computer solutions have tended to channel their thinking into sequential processes. This is largely an effect of the language restrictions.

Here is proposed a design methodology based upon traditional language and concepts, but offering some of the advantages of more modern thinking. This computer-based system, operating through a construct similar to, but more powerful than, Dijkstra's guarded commands, is intended to provide an escape from the sequential thought process. The outcome of different logical situations may now be studied in parallel. Retention of Fortran as the host language allows this system to interface, where necessary, with existing programs.

The system was designed using the methods presented in this paper, and is itself written in Fortran. Its own structure, being the same as that of the programs it builds, is considered a proof of the workability of the overall concept. Within a problem requirement, a clear distinction

is maintained between logic and a set of actions providing the solution, as well as table components which link the two. The user provides a set of simple condition statements, a set of action statements, (both in Fortran), tabular linkage/control information, and descriptions of global data items. The system provides the means to test the logic of a proposed solution, even before all information has been assembled. It also aids in building and modifying the various components. A simple driver provides the executing power for both the system and the problem-solving procedure which it has helped to create.

By being encouraged to adopt a top-down approach to problem analysis, a user is able to produce a solution program whose structural form is directly related to that of the problem requirements. With the full capabilities of the Fortran language in no wise diminished, engineers are offered a new aid in the solution of technical problems.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

PREFACE

Practising in the field of structural engineering at the beginning of the 1960's, I experienced the lure of the computer while it was yet in its infancy. It is a marvel to me, now, that I should have taken pleasure in a machine whose most predictable feature was the regularity with which its vacuum tubes burnt out. It was in those early years of computing that we acquired our first Fortran compiler, which brought us relief from the frustrations of machine language programming.

Twenty years of R&D in the electronics industry has left that machine a mere museum piece. But what of the language? While amazing advances in hardware were being made, were the software developers really standing still? Of course not! But the engineering industry had very quickly taken advantage of the "new" computerised methods, and had soon built a considerable stock of programs. As its investment in Fortran written software increased, so did its reluctance to accept new languages whose advent might erode that investment. A fair amount of inertia then, must be overcome in order to change the programming course established within a large organisation.

Working in such an organisation, I am constantly made aware of the fact that in pursuit of computerised solutions to problems, engineers almost invariably think in terms of the sequential execution of actions, in spite of the fact that their problem requirements imply no such time-dependence. With Fortran as the only available programming tool,--no wonder!

How then could engineers profit from some of the modern software advances, within the restrictions of the Fortran language? The following text describes how that challenge has been approached in an engineering office. How successfully, remains to be seen.

My grateful thanks are owed to my supervisor, Dr. W. M. Jaworski, for his support and stimulation throughout this project. Mary Morgan and Sharron Tracy both displayed forbearance and secretarial skills which were equally remarkable. Without these three, this work could never have reached fruition.

I. INTRODUCTION

There are many instances of programming tools based upon the idea of pre-processors to a high level language such as Cobol or Fortran [1-3]. Most appear to permit of, or impose program structure by virtue of non-host-language statements, which through pre-processing are converted to high level code acceptable to a standard compiler.

Pre-processing has several shortcomings:

1. It provides essentially a programming tool; not a design one
2. It often involves the learning of new language syntax
3. Program testing can only be performed on a complete pre-processed and compiled program
4. Program code changes usually involve a pre-processor re-run, as the generated source code is likely to be meaningless to the unfamiliar user

Endeavouring to overcome some of the aforementioned inadequacies (notably 1), resulted in this design for a new system. It is formalised in the following objectives:

1. To devise an analysis/programming aid, primarily for Engineers. With this, the design of a computerised solution to a problem could be produced, bearing a close resemblance to the structure of the original problem requirements
2. To implement the proposed system in the same manner as it would handle a user's application design. Being modular in structure, each part of the growing system would be used in the building of

later ones; and the workability of problem solutions implemented in this way would be ensured

3. To demonstrate the features, and highlight weaknesses of the operational system by its application to the solution of a sizeable engineering problem

The work upon which this thesis is based was founded on concepts presented in [4], most notably that of grouping related logic tests into a single switching function, in order to provide both a better program design method, and greater clarity of program operation for the reader. An experimental system embodying these concepts has been devised and implemented in APL [5], and provided a starting point for the development of the operational system described within this text. The scheme adopted here was seen, too, as a means of providing greater software reliability. A survey of methodologies which could contribute to the prevention or early detection of software errors brought about the design of a system which united the building and testing functions of a problem solution--having already influenced its analysis. It was hoped thereby to support many, if not all of these aids to reliability improvement.

It has not been the intention here to invent another language, nor yet aggravate the burden of the programmer whose path is already strewn with miracle cures for unstructured Fortran. It is, rather, an attempt to provide the means whereby a user, by taking a structured approach to analysing his/her problem, creates a Fortran program whose strongly structured form matches that of the problem requirements.

II. BACKGROUND

Program Structure. One of the aims at the outset of this project was to provide structure to problem solutions. Why the concern for structure? Floyd [6] puts it succinctly: "The structured programming paradigm is by no means universally accepted...Yet [it] does serve to extend one's powers of design, allowing the construction of programs that are too complicated to be designed efficiently and reliably without methodological support."

The first phase of structured programming is the process of top-down design, in which the problem is decomposed into a small number of simpler sub-problems. Continuing the process further, results eventually in sub-programs which are simple enough to be coped with directly. Yet further decomposition, of course, produces a detailed algorithm.

Modules produced in this fashion can then achieve the aims of understandability, operational reliability, and simplicity of maintenance and extension, as outlined by Turner [7]; but only if the required structured coding goes hand-in-hand with structured control of those modules. A program in which control can be passed arbitrarily among modules is just as unmanageable as a single module in which GO TO statements are used indiscriminately. Turner concludes that ideally the necessary degree of control can best be implemented by building programs as pure tree structures. Such a structure would consist of a collection of modules, each having functional strength; there would be no sharing of modules between branches of the tree; and the program would maintain locality of reference. Unfortunately one is seldom able to produce the ideal; and here, as ever, compromise is in order. It becomes necessary in practical applications to allow sharing of certain types of service subprograms; but still the spirit of the pure tree structure is retained.

Module Structure. What then is to be the pattern to which modules will be built, such that the desired degree of control over them can be exercised? While not necessarily a requirement, it is obviously desirable that all should possess the same structure. Of over-riding importance here, however, lies the objective of providing a design tool rather than a programming tool. With this requirement fulfilled, the engineer who is seeking to solve a problem, can derive the algorithm representing his/her desired solution in the same terms as will be used to produce that solution. Or to put it another way: the mystical "conversion" from problem specification to someone else's incomprehensible (to the engineer) code, representing the solution algorithm, is avoided. To extend this a stage further, it is highly desirable that the same language even, be used both to define and execute the problem solution.

To talk of "engineering problems", with a sweep of the hand, is being somewhat vague. This becomes apparent as soon as one seeks common ground in problems such as arch dam analysis, hydraulic flow, prestressed concrete frame design, transmission tower design. These problems, as many engineering ones do, involve computerised modelling of real-world situations; and in all (of what are often highly complex problems) are seen groups of actions to be performed in various distinct and mutually exclusive situations. The groups or alternatives are thus "guarded" by the conditions which constitute those situations. Further, it becomes apparent that there exist many clusters of alternatives which are logically independent of each other, separated by time. For example, the sets of actions to be executed to design a bridge foundation cannot be performed until those for the superstructure are complete.

What is required then, is a single construct which can be represented in the user's preferred choice of computer language, which can incorporate a high level of logical complexity, and which retains the structure of the problem specification for the purpose of execution. The choice of the Decision Table to fill these requirements is a natural one.

Decision Tables. The existence of copious quantities of literature on the subject of use and conversion of decision tables testifies adequately to their usefulness in the field of problem specification. A plethora of information of an introductory nature [8, 9, 10, 11] precludes further description except to state that the "normal" form of condition stub, condition entry, action stub and action entry is used here.

Tausworthe [8] defines a decision table as a tabular display of the pertinent logical aspects of a programming problem, showing all relevant conditions, relationships, and actions to be taken under each set of circumstances. Among the advantages to be gained from using it as a design tool are:

1. It forces a clear problem statement and shows where information is missing
2. It completely defines, at the top hierarchic level, those decisions to be implemented
3. It permits functional definitions and descriptions that are distinct from procedural content
4. It modularises the program by forcing segmentation of the overall system into logically manageable tables
5. It is suitable for documentation and for communication of the program operation between people
6. It assists in implementing program changes, and tends to identify consequences of any one change, even in a complex program

It is only fair to point out, of course, that not quite the whole world is enthusiastic about decision logic structures. Because of the usual approach to implementing them, Anastas and Vaughan [12] tend to regard them only as "monitoring and evaluation tools", and see their use as constituting a divergence from the central development path. The conditional transfer of control associated with the typical decision logic construct (they say), does nothing more than circumvent the GO TO-less dogma. Their proposed program structure has a coded representation which is simple and at once easy to read; but they appear to miss the point that a designer need be tied to his/her source language listing, no more than to a listing of the machine code to which the program is translated. What is far more important is that the user sees the program source in the same way as it was originally presented, and that testing and debugging is carried out on that same program form. If software support can be provided such that a user can define decision tables in Fortran, and maintain that same view of his/her problem solution from conception to implementation, (worry not that it possess some other form for the purpose of (say) compilation), then the above objection to their use is satisfied.

Low [13] suggests that "...many of the difficulties inherent in...decision table programming can be avoided" by what he terms "Programming by Questionnaire." Briefly, this involves an English language questionnaire, a list of source statements, a set of decision tables and an editor program. Since Low is describing a programming tool, these four components are assumed to exist already when the programmer takes up his/her task; and therein lie several shortcomings as far as the engineer is concerned:

1. The application designer (problem solver) is not the programmer, i.e., there is a distinct separation of function betwen problem specification and computerised implementation
2. The system designer--and it is in the system that the logical complexity lies--does not have the convenience of the system to help him/her implement it. As a result of this:
   a. the plan of the system is not mapped clearly into its implementation
   b. the mapping of original problem specifications into a final application program is obscure in the extreme

Perhaps for some applications these disadvantages are of little importance, but for the engineer who is at once problem analyser and application programmer, they represent major difficulties.

Thus far, only the weaknesses of the Low system have been approached. However, with only minor modifications, the concept fulfills exactly the requirements which were sought at the outset. Firstly, the questionnaire itself is required only to provide a link between the application designer and the programmer. With those two being one and the same person, it becomes unnecessary. Secondly, the decision table format must be changed so that condition stubs and action stubs, instead of being question identifiers and source statement numbers respectively, should be represented in more understandable terms. There is no reason, for instance, why condition stubs should not appear as the Fortran logical expressions which will be used to define those conditions in the final program. Action entries must be a little different, as in general each action will be implemented by several Fortran statements, in which case visual clarity of the decision table would be lost. A simple descriptive is all that is required to define the action's function. Next, instead of being handed a library of source statements, the engineer should have only the structure of a library, so that Fortran statements can be entered and retrieved as required. Lastly, whereas Low's editor simply provided batch execution of decision tables to produce one or more programs, here is required the ability, first to build the decision tables, to modify them, to build the source library and then to execute tables to produce a program.

A notation for these components has been conceived as an Alternative Based Language (ABL), Jaworski [14], and provides a solid basis upon which to construct this system.

Software Support. "There is a software crisis and it is ubiquitous... Inordinate amounts of time and money are invested in software development, only to result in additional systems that fail to work properly." (House [15])

House continues by identifying five principal causes of poor software:

1. The magnitude of the task of developing excellent software is not fully recognised
2. The function to be performed by a given software component is not properly specified
3. The internal structure of programs is often ill-chosen
4. The major standard programming languages available are often not the best tools
5. Programs are not adequately tested

The discussion to this point is leading up to satisfying four of these five causes. By forcing modularity and top-down problem analysis, the magnitude of the task, if not appreciated from the outset, becomes clear very quickly. The second and third points are both met by the choice of decision table as the basic construct. Its use forces a close evaluation of the function which it is to perform; and while not necessarily providing the most execution-efficient solution, its use will not give rise to software errors if the more usual manual conversion of it is avoided. The question of whether or not Fortran is the best tool for problems of an engineering nature is open to debate. For reasons mentioned earlier, there is often no choice to be made in the matter. What is fairly certain, however, is that it is not the best tool for structured programming. With the use of a reasonably "intelligent" editor, structure for the language can be provided, and thus point four satisfied.

What can be done then to meet the requirements of the fifth point --that errors exist because of inadequate testing? Before being able to provide an answer, one has to examine the nature of errors which actually occur in software, and deduce therefrom the methodologies which might have detected them. Glass [16,17], Howden [18], and Myers [19] among others have conducted much research in this field. Error detection methodologies fall into two categories, static and dynamic, and some of the more important of these they identify as:

1. Static
   a. Peer Code Review--examination of requirements, design documents and program code by other programmers
   b. Subroutine Interface Analysis--automatic checking of parameter consistency (precision, number, type) between subroutines
   c. Data Flow Analysis--identification of uninitialised variables, common block omissions, erroneous value assignments
   d. Control Flow Analysis--detection of improper control transfer
   e. Statement Analysis--testing of source code statements against externally defined rules (such as programming standards)
2. Dynamic
   a. Environment Simulation--simulation of a program's operating environment
   b. Data Tracing--identification and value display of variables each time they are assigned values
   c. Test Coverage Analysis--maintenance of a count of each logic segment's execution
   d. Assertion Checking--evaluation of how a program is functioning against a predefined set of rules

e. Dumps--listing of parts of memory during program execution

f. Logic Tracing--display of logical elements' identities during execution

More detailed descriptions may be found in [16, 17, 18, 19]. The most effective single methodology has been found to be the peer code review. It would evidently be an expensive choice however, strictly on the basis of labour involved. It is also patently clear that no single tool offers the degree of error detectability required to guarantee any great level of software excellence. Myers, in particular, felt that computer assisted walk-throughs offered greater promise, and it became an aim here to try to exploit that promise in the design of this system.

Modularity and choice of the decision logic construct provide a sound basis (and indeed, a requirement) for the production of clear specification and design documentation. This, together with the aim of freeing the designer from the traditional source listing (a feature available to the tester, too) supports well the semi-automatic walk-through process. Control flow analysis is implicitly supported by virtue of the very logic structure chosen, and subroutine interface control and data flow analysis also become design objectives.

On the dynamic testing side, the environment simulator takes on less importance for programs of an engineering nature than for real-time applications in which the interaction between operating system and multiple users may have a profound effect upon their behaviour. It would be desirable though, to support it, as well as all the other dynamic methodologies listed. The constructs selected were seen as a means of offering immediate support for many of the required methodologies while providing a sound basis for later extension to support the others.

Summary. Refinement of the original objectives produced the following system design requirements:

1. Modular hierarchical structure, as close to pure tree as possible
2. Basic logical construct to be the limited entry decision table
3. User's view of problem solution to be maintained from conception to implementation
4. Powerful editing facility to structure, manipulate, and test modules
5. Testing capabilities to be based upon a variety of both static and dynamic methodologies

III. PROBLEM ANALYSIS

An engineer seeking a solution to a problem of any degree of complexity has to decide how to divide it into manageable tasks, and a hierarchy of such tasks develops: the problem is solved when tasks A and B are completed; but task A involves solution of tasks C and D etc. (Figure 1). The division into discrete tasks is usually fairly arbitrary, and most problems of appreciable size may be divided in many different ways. The emphasis at this stage is upon "discrete", as there will be no sharing of logic or executable code across these boundaries. Within each task further division will be required into sub-tasks.

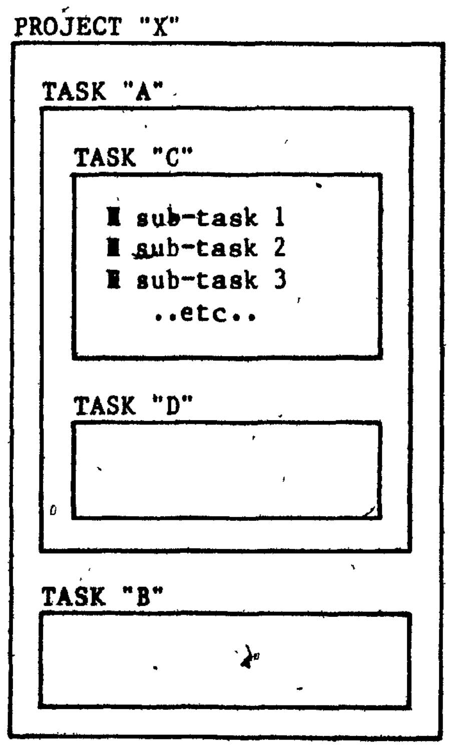


Figure 1. Division of a problem into tasks and sub-tasks

The outcome of this preliminary analysis, expressed at sub-task level, is a set of declarations of the form: "in the event that situation X arises, take the following action". One can of course, by devious means, bend the Fortran language to follow the expressed problem requirements. Most engineers however, have neither the knowledge nor the inclination to embark upon the programming exercise required. How then to express the problem requirements? Dijkstra [20] shows problem solutions expressed in terms of guarded commands, and while the principle holds good for any sized problem, certain disadvantages creep in with increasing size. Either the action statements become complex and thus specialised, this rendering sharing of code between logical situations impossible; or there must be a repetition of guards for simple actions, which leaves the reader of the program searching long lists of conditions to find the outcome of a situation.

It was mooted earlier, that often the most concise way of expressing a problem solution is through one or more decision tables; but there have always been difficulties with their implementation, a direct translation being beyond the capabilities of most computer users. Anastas and Vaughan's introduction of their "requirements-oriented program structure" [12], with a view to producing a transition machine to handle it, underlines, however, the validity of the tabular approach.

In all of these cases one is immediately aware of a clear division between statement of the logic controlling a problem solution and the executable actions of the solution itself. A complete statement of the problem to which a task applies then is the domain of the set of conditions which define the program's logic; and an instance of the evaluation of that set of conditions represents a logical "situation" (Figure 2). Since it is in terms of these situations that the engineer would like to express the actions, it falls to us to provide the means of determining the problem state in terms of defined situations, and linking to the subset of actions which that situation demands.

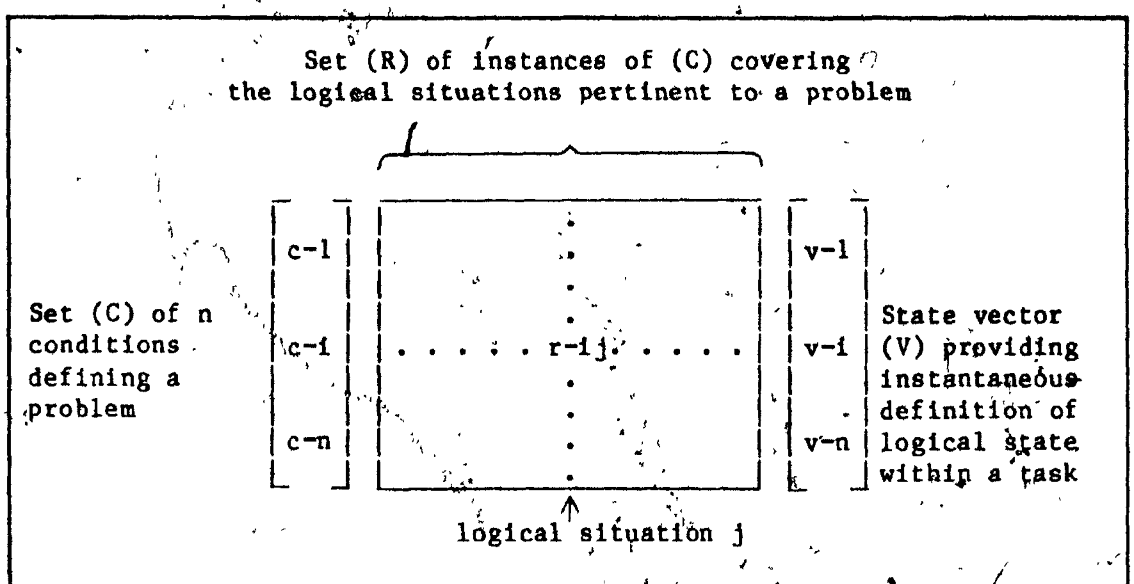


Figure 2. Expression of the logic of a problem solution in tabular form

Here then we use the conciseness of the logic table together with the requirements-oriented program structure both to express and perform problem solution.

The actual operations involved in producing a problem solution will be:

1. Name the job (or project as it is referred to here)
2. Divide it into a number of discrete named tasks
3. For each named task:
   a. Subdivide it into a number of named sub-tasks or clusters

b. For each named cluster:

   1) Analyse the conditions which have significance within this cluster and establish the logical situations which are to be considered

   2) Name the actions which are to result from each of these instances

   3) Note which cluster is to be entered next on completing the requirements of each situation (if known--it usually is)

c. Form the union of the sets of conditions (predicates) used within each cluster and assign numbers 1-n arbitrarily

d. Do the same for the named actions

e. Supply this information to the computer (appendix A). At this stage, enough information has been produced to enable the logic of the task to be tested on the computer, although there has been no attempt to produce any actual code. That can be left until after the logic has been verified.

f. When it has been established that the solution is good, the Fortran code corresponding to each of the named actions can be derived, together with details of data items which are required to be "global". These too are entered into the computer

g. Generate Fortran subroutines from the sets of actions and predicates

4. Run the complete program to obtain the problem solution

IV. DESIGN

Logical Concept. It has been stated that there is to be maintained a clear distinction between the logic of a problem solution and its functional parts, and that the decision table is to be the construct used to achieve this separation. If an engineer's view of a problem is concentrated upon sub-tasks, as now is encouraged, and his/her expression of the solution procedure is to be made in terms of our chosen construct, it follows that each sub-task should be (or at least appear to be) represented by a single decision table. In terms of the programs created to represent the actions of these sub-tasks, however, there is much to be gained from introducing a more complex building block whose lower level components are seen as, and which function as conventional decision tables.

What are these advantages? Upon examination of any conventional program of appreciable size, it is evident that there exists a large amount of duplicate source code. How many times do we read a card, increment a counter, zero a value? One of the reasons for demanding a total separation of logic and actions is to avoid this duplication of code, and that can only be achieved as long as sharing of actions is possible. Another stems from the user's desire to investigate the effects of changes of component algorithm upon the total problem solution. By managing the user's tables in the framework of a higher level tabular structure, the system will permit choice of one algorithm, out of several programmed, at run time.

From the above discussion it is seen that what is proposed is a mechanism for maintaining two distinct views of the same problem solution; one for the user and another for the system. To avoid confusion in the ensuing text, the term Cluster is applied to what the user sees as a lowest level decision table, while the term Table will be applied to the higher level construct. These then correspond respectively to the job sub-divisions of Sub-task and Task of Figure 1.

This mechanism, with its bi-directional "vision", will thus provide the user's needs for building the tables from the defined clusters; modifying them in the event of problem requirement changes, or simply for correcting errors of logic; and for the testing capabilites outlined earlier. The multi-purpose Editor will be the vehicle for all functions occurring between user and system.

The concept of clustering within a table permits a user to sub-divide the logic of a problem into small units, while still allowing the sharing of action code. He or she may concentrate upon the solution of each manageable part (Figure 3), safe in the knowledge that each is an operationally discrete unit.

The first new component required by the system is the Cluster table. This defines the relationship between logical situations,--rules defined within a task.

The suggested convenience of the Cluster, brings an awareness of a further system requirement. While the user thinks in terms of Clusters as far as building (and later, testing) of tables is concerned, at execution time correct selection of rules must be made from what the system "sees" as one table only.

Cluster 1	1	2	3
P-1	Y	N	N
P-2		N	Y
A-1	1		
A-2	3		
A-3		1	
A-4	2		
A-5		2	1
A-6		3	2
Next Step	1	1	2

Cluster 2	1	2	3	4
P-1	N	N	N	Y
P-2		Y	N	
P-3	Y	N	N	
A-1		1	1	
A-3	1			1
A-4	2	2		
A-5		3	2	
A-6			3	
Next Step	1	2	2	3

Cluster 3	1	2
P-1	N	Y
P-2	N	Y
P-4	Y	N
A-2	1	1
A-3		4
A-4	3	3
A-6	2	2
Next Step	3	0

Figure 3. The user's view of a problem task in tabular form. There are three operationally discrete units--clusters that are able to share both predicate statements (P) and action code (A)

To force the user to guarantee uniqueness of rules in the total table negates entirely the benefits obtained from clustering. So a mechanism must be provided to activate during execution, only those situations which are pertinent to a specified sub-task. In short, an execution strategy providing high level guards to selected situations. Figure 4 illustrates the system's "view" of the information which was shown from the user's standpoint in Figure 3. The set of steps (S) each corresponding to a cluster (C) forms the Strategy Table component.

At execution time, only those rules whose guards are "open" for an activated step are accessible. To achieve a solution the system will repeatedly evaluate the current process state--in terms of the declared predicates--in the state vector (Figure 2), attempt to match it to one of the accessible rules, and perform the appropriate set of actions all as per conventional decision table. An active step remains so until the guard on the currently executing rule indicates otherwise. This it does by pointing to an inactive step, whereupon the inactive becomes active and vice versa.

Table: FIG4

	1	2	3	4	5	6	7	8	9	
S-1	1			1			2			Strategy Table
S-2		1	2			2			3	
S-3					3			0		
C-1	x			x			x			Cluster Table
C-2		x	x			x			x	
C-3					x			x		
P-1	Y	N	N	N	N	N	N	Y	Y	Predicate Table
P-2			Y	N	N	N	Y	Y		
P-3		Y	N			N				
P-4					Y			N		
A-1	1		1			1				Action Table
A-2	3				1			1		
A-3		1		1				4	1	
A-4	2	2	2		3			3		
A-5			3	2		2	1			

Figure 4. The system's "view" of the problem (see Figure 3). Rules membership in clusters (C) is indicated by 'x'. Appropriate guards are shown within each step (S)

Figure 4 represents the system's view of the task solution. The user is unaware of the ordering of information within it; and indeed no ordering of rules, predicates or actions is implied by this representation. Cluster 3 rules are shown as occurring as rules 5 and 8 in the table, but they could equally have occurred as rules 1 and 2.

The Strategy Table is necessarily an integral part of the System's "view" of the total table, and in terms of operation need bear no direct relationship to the user's view through clusters; so an independent means of building and editing should be provided. However, it soon becomes apparent that logically the Strategy and Cluster Tables should be closely tied. Why then, not dispense with one of them; the Cluster Table say? The meanings of these two tables are essentially different; one representing "belonging" and the other, control. So while for the most part, these two attributes parallel each other, there are instances when for one reason or another it is desired to exclude a situation (i.e. prevent its execution), without denying its membership in a cluster. This might be desired for the purpose of investigating changes of logic upon the execution of a program, or to allow one program to provide different functions to different processes calling it. There are also occasions when a particular situation can be met in more than one sub-task, giving rise to the same required action sequence in each. In cases such as this the situation, or rule representing it, may be a member of more than one cluster, and the cluster table entry for that rule will contain more than one 'x'.

A strong case exists then for tying the construction of the Strategy Table to that of the Cluster Table, while retaining the ability to handle it independently.

What this chapter has so far introduced is an outline for a system supporting ABL. Hinterberger and Jaworski [21] developed a PASCAL system embodying the concepts, and applied it to the solution of a simple virtual storage paging algorithm. Here, by way of illustration, a problem discussed in [22] is considered. Two versions of a program to calculate square roots are investigated (Figure 5).

```
10  READ(5,11) X
11  FORMAT(F10.0)
    IF(X.GE.0.) GO TO 20
       WRITE (6,13) X
13     FORMAT(' SQRT('1PE12.4,
   1        ') UNDEFINED')
       GO TO 10
20  IF(X.GT.0) GO TO 30
       B=0.
       GO TO 50
30  B=1.0
40    A=B
      B=(X/A+A)/2.0
      IF(ABS((X/B)/B-1.0).GE.1.E-5)
   1       GO TO 40
50  WRITE(6,51) X,B
51  FORMAT(' SQRT(',1PE12.4,END
   1        ') = ', 1PE12.4)
    GO TO 10
    END
```

```
100  READ(5,110) X
110  FORMAT(F10.0)
     IF(X.LT.0.) WRITE(6,120) X
120  FORMAT('SQRT(',1PE12.4,
   1     ') UNDEFINED')
     IF(X.EQ.0.) WRITE(6,130) X,
130  FORMAT('SQRT(',1PE12.4,
   1     ') = ',1PE12.4)
     IF(X.LE.0.) GO TO 100
     B=X/2.0
200  IF(ABS(X/B-B).LT.1.E-5*B)
   1     GO TO 300
     B=(X/B+B)/2.0
     GO TO 200
300  WRITE(6,130) X,B
     GO TO 100
```

Figure 5. Two versions of a program to calculate square roots, written in Fortran (from [22])

Both of these programs can be considered together in ABL notation and, thereby, their merits studied concurrently (Figure 6).

Table SQRT

Rules:	01	02	03	04	05	06	07	08	09	10
Strategy										
1. Initialise-1	2									
2. Validate-1		2	2	3						
3. Approximate-1					3	2				
4. Initialise-2	5									
5. Validate-2		5					5	6		
6. Approximate-2									6	5
Cluster										
1. Initialise	X									
2. Validate-1		X	X	X						
3. Validate-2		X					X	X		
4. Approximate-1					X	X				
5. Approximate-2									X	X
Predicates										
1. X.LT.O.		Y	N	N			N	N		
2. X.EQ.O.							Y	N		
3. ABS((X/B)/B-1.).GE.1.E-5					Y	N				
4. ABS(X/B-B).LT.1.E-5*B									N	Y
5. X.GT.O.			N	Y						
Actions										
1. READ(5,110) X	1	2	3			2	2			2
2. WRITE(6,120) X		1								
3. B=0.			1							
4. WRITE(6,130) X,B			2			1				1
5. B=1.				1						
6. A=B				2	1					
7. B=(X/A+A)/2.0				3	2					
8. WRITE(6,130) X,X							1			
9. B=X/2.0								1		
10. B=(X/B+B)/2.0									1	

```
110 FORMAT(F10.0)
120 FORMAT('SQRT....UNDEFINED')
130 FORMAT('SQRT....IS....')
```

Figure 6. The two square root programs represented in ABL

While this provides a concrete example of ABL notation it must be remembered that it is the system's view that is shown here. The problem solver's view would have been through clusters, more like Figure 7, for example. Presented in this manner, both the logic and the action sequences involved become remarkably clear. This remains true even when applied to large complex problems. All can be broken easily into small and manageable clusters. The result of this should thus be fewer errors of logic in the problem solution, and greater ease of solution checking by other parties.

Clusters:	VALIDATE-1			VALIDATE-2		
Rules:	1	2	3	1	2	3
Predicates						
1. X.LT.0	Y	N	N	Y	N	N
2. X.EQ.0					Y	N
5. X.GT.0		N	Y			
Actions						
1. READ(5,110) X	2	3		2	2	
2. WRITE(6,120) X	1			1		
3. B=0.		1				
4. WRITE(6,130) X,B		2				
5. B=1			1			
6. A=B			2			
7. B=(X/A+A)/2.0			3			
8. WRITE(6,130) X,X					1	
9. B=X/2.						1
Next Step:	2	2	3	5	5	6

Figure 7. Two comparable clusters from the square root programs

Clearly, a problem of any magnitude will be divided into more than one task, requiring therefore more than one table to effect its solution. A means of passing control from one table to another along the lines of the pure tree structure previously mentioned must be provided. So also must exist a device for controlling data flow. A dictionary into which items are catalogued according to name, type, length, value ranges, scope, initialisation indicators and the like, can provide the degree of control required, both for data items used within a problem solution, and for control of the loading of the various tables themselves. Thus the data dictionary sits at the highest level of a user's problem solution, providing in an upward directon an interface to the Operating System, and in a downward direction management of data flow and passage of control between tables.

The entire system may be visualised as shown in Figure 8, where the arrows indicate flow of information; the solid ones during building and testing of a user's problem solution, and the dashed ones at time of execution of that solution.

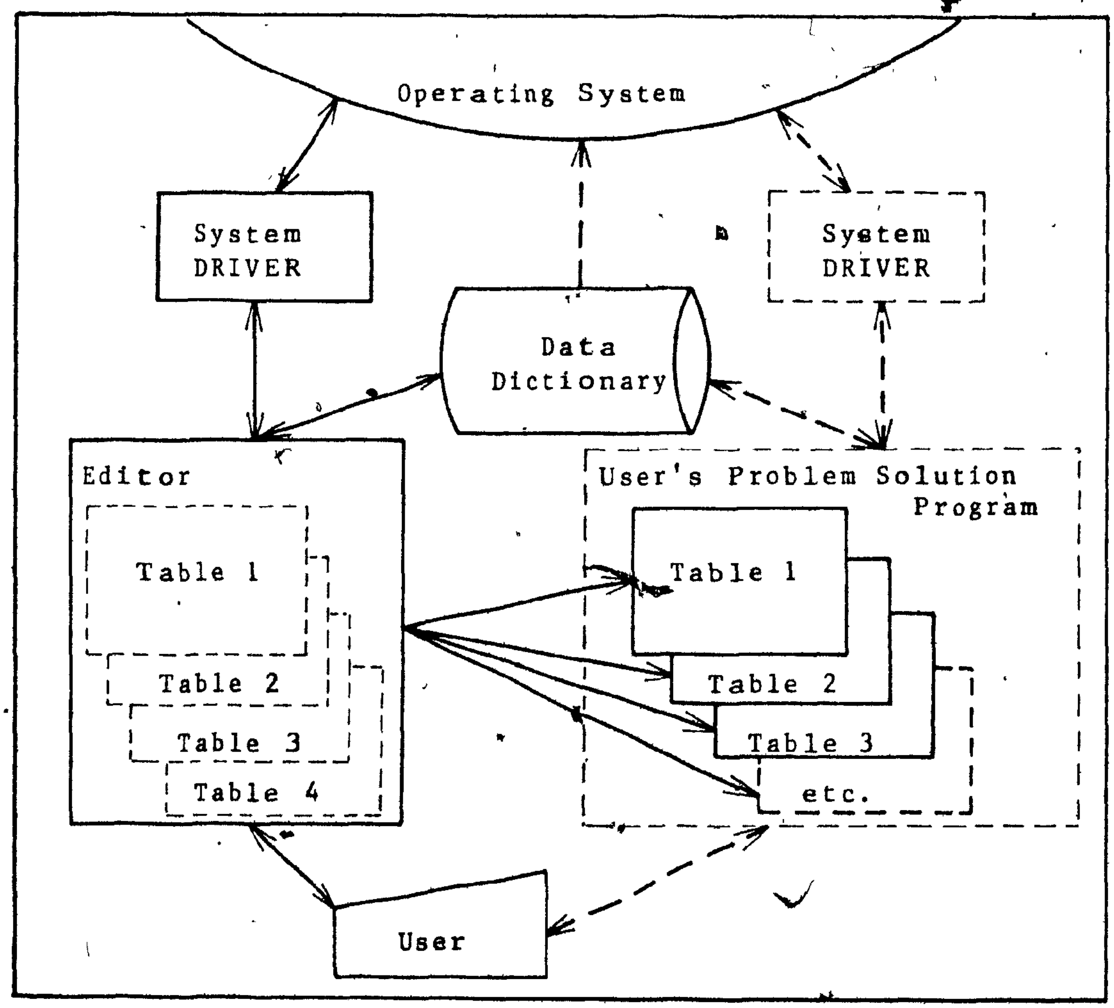


Figure 8. Conceptual view of the system. Note that the same Driver drives either the Editor or the user's problem solution program

Physical Representation. The discussion of the logical concept of the system has outlined the need for four major components:

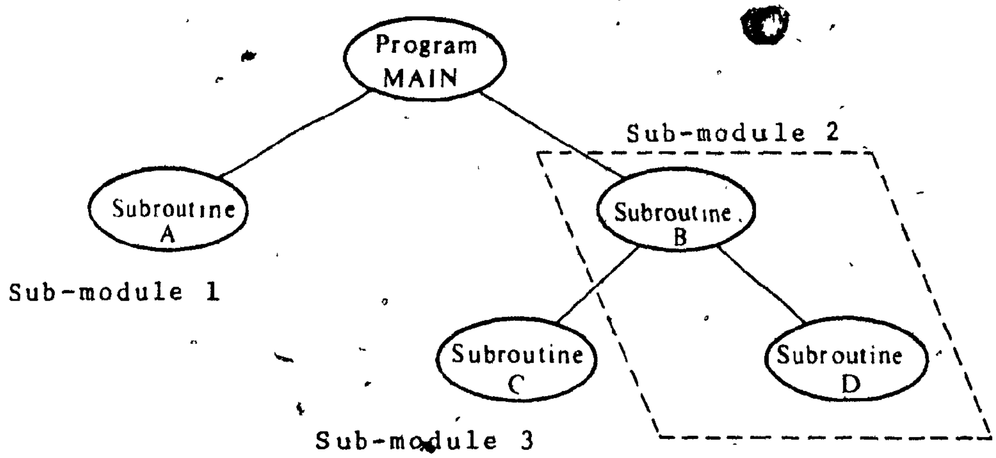
Table Constructs comprising
--decision tables
--cluster tables
--strategy tables
Table Manager/Editor for the purpose of
--building and modifying clusters
--maintaining the user's view of the tables in the form of clusters
--testing the logic of a task solution
Data Dictionary to
--store information about data items
--provide system to Operating System interface
--control data flow between tables
Executor to provide the driving force to execute tables.

How are these to be represented in physical terms?

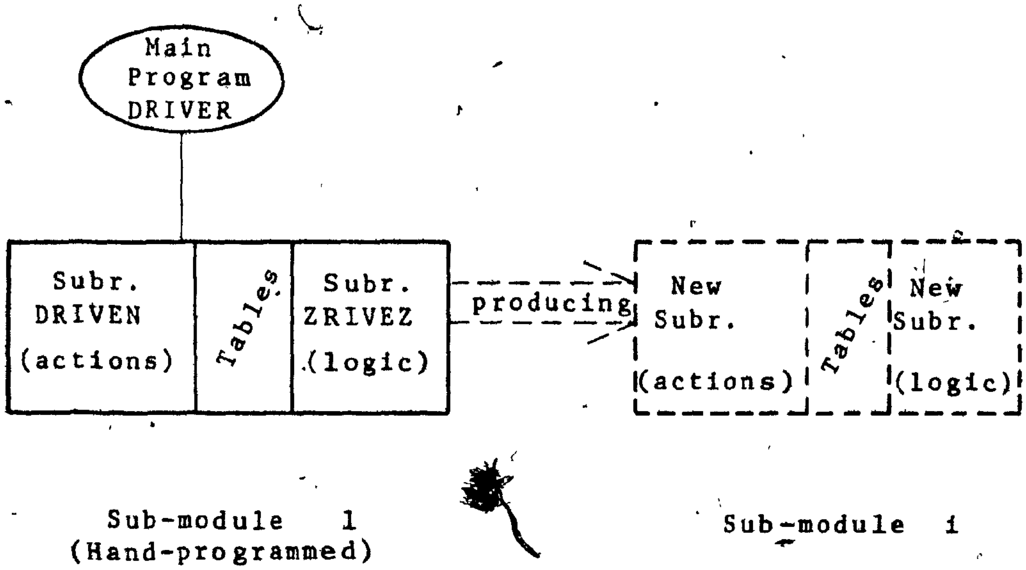
The root of the system will be a small main program whose sole task will be to "drive" an application sub-module. The first application sub-module to be written therefore, will be for the purpose of building other application sub-modules of identical structure to itself.

At this juncture an explanation of the term sub-module is perhaps required. In terms of conventional Fortran programming a sub-module is simply the executable actions and interwoven logic of a SUBROUTINE, i.e. a discrete logical unit. Figure 9a shows a typical arrangement of sub-modules, as the term might be applied to conventional Fortran programs.

Here however, because the logical and the executable parts will be separated, the sub-module will comprise the collection of logical part, executable part and that which is necessary to link the two (referred to loosely as "tables"). Figure 9b illustrates the use of the term in the proposed system, showing at the same time how the first sub-module will operate.



9a Arbitrary definition of sub-modules in a conventional Fortran program



9b Components of a "sub-module". Original hand-programmed parts of sub-module 1 to be supplemented by the generated parts of the new sub-module as development progressed

Figure 9. Illustrating the term "sub-module" as used in the text

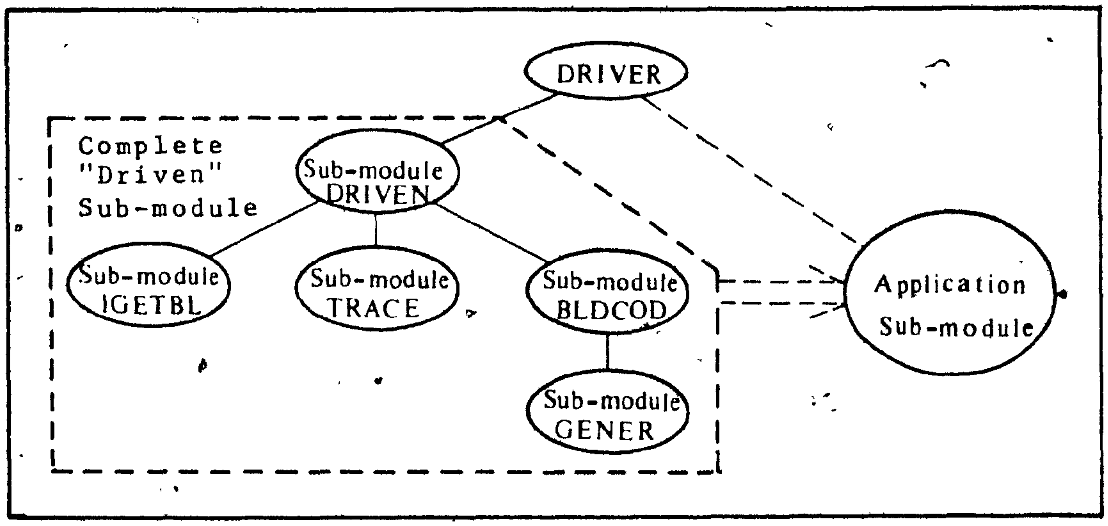


Figure 10. With the "Driven" sub-module complete a user's application sub-module(s) would be built and subsequently executed

As development continues, the "Driven" sub-module will grow from combinations of hand-written and generated parts, themselves forming sub-modules, until the full system capability is reached. At that time an engineer (or other) will be able to use the system to produce one or more sub-modules corresponding to his/her problem solution (Figure 10). Upon completion of the application sub-module, execution will be achieved by loading it with the DRIVER and omitting the system's "Driven" sub-module.

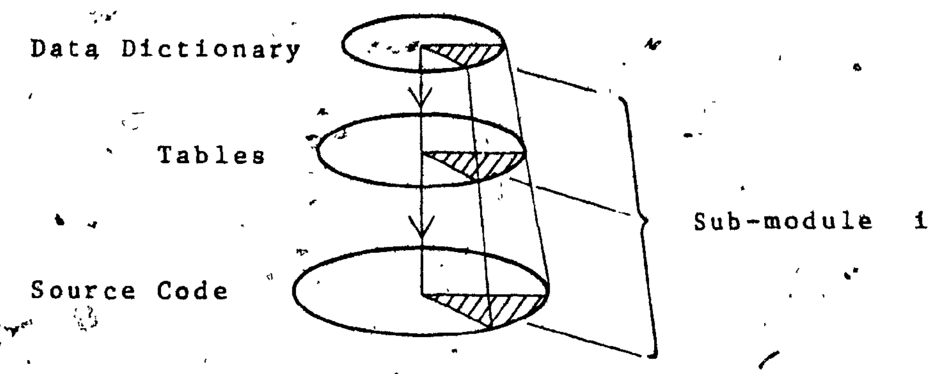
It was likely that the tables which are to provide the vital links between logic and actions would have to contain much more information (for the building of their own constituent parts) than would be required during actual execution. Since one of the more limited resources is main memory it made sense to design the tables so that only those parts required for execution will be in main memory at that time. Sub-modules will evidently differ greatly in their sizes. Thus it was decided that only a Program Descriptor table (PDT) (Figure 11) will occupy the same space and location within each of the table files, and all other parts will be located from information stored therein.

Physical Components. Stressing once again the requirement for separated logic and actions, it remains to provide the means of not only linking them during execution, but also of building and manipulating them beforehand. Figure 11a illustrates the order of control of information at overall project level, while Figure 11b shows the components of one sub-module out of the project.

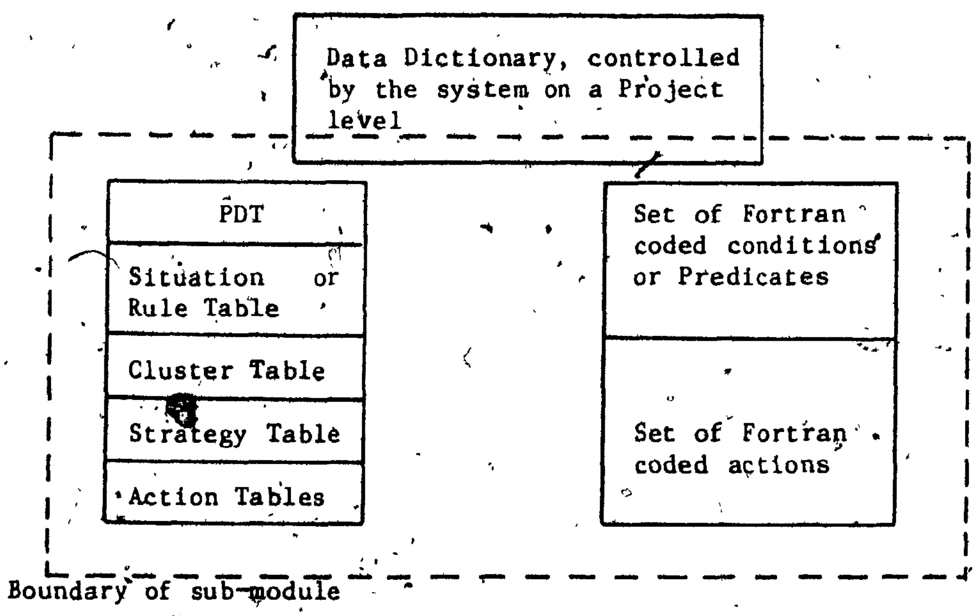
At the highest level of control comes the Data Dictionary which fills a double function: maintaining global data information, and providing links between the user's tables and the Operating System's file identification requirements.

The table component, headed by the Program Descriptor Table serves to link the data dictionary global data information to the source code file for program generation. It provides the link between predicates and actions at test or execution time. It also contains the necessary data upon which to build its own sub-components, the code component, and its own entries in the data dictionary.

The collection of information held in the four sub-components is seen as a conventional decision table; the set of logical situations being considered forming the Rule Table, while action "entries" are replaced by an Action Table.



11a Information Hierarchy within a project



11b Components of a "sub-module"

Figure 11. Subdivision of a project into sub-modules and components

To this point, the tables providing the execution link between logic and actions, as well as global data items, have been mentioned; and as a user's prime objective is the successful execution of his/her problem solution, it is perhaps logical now to pass on to the handling of the source code.

For the purposes of this System, a Simple Action is defined by any group of zero or more executable source statements having a single entry and a single exit (first and last statements respectively). The statements comprising each action are stored during sub-module construction, so that upon completion the System is able to generate an entire subroutine to represent the full action code for one table.

In the early stages of analysing a problem there is naturally a greater concern for breaking it into named actions than for the method of executing those actions. It is only as the solution logic develops that a greater refinement of actions and of their content comes about. Therefore, by storing the name of an action rather than its code, the way is open to test program logic by simulated execution as opposed to actual. Only after the logic has been proven need the source code be supplied.

When designing the tables pertaining to a problem solution it is convenient to keep track of actions by number. These serve only as labels as there is now no relationship between sequence of execution of actions and their physical location. These "labels" then are used in the building of the Action tables, wherein are defined execution sequences appropriate to the various logical situations.

The predicate source code representing the logic of a sub-module, is of similar nature to the previously described action code. It can therefore use the same file as that action code. Here again, it is not necessary for the predicate code itself to exist before simulated table execution is possible.

Execution Control. In principle, the transfer of control between tables will be simple to effect. A single action,--a mini-driver, identical in structure to the driver itself,--will make repeated calls upon the lower level sub-module until that sub-module's strategy entry indicates return of control to the higher level. This is one instance in which the sharing of the action code will be impossible. Each new table to be called will require its own mini-driver. At this point it becomes apparent that it is quite infeasible to allocate separate storage areas in memory for the tables of each sub-module to be executed. One such area will have to suffice for all. Since the system's tables are obviously not in use during execution of a lower level sub-module's, and since the storage structure of both is identical, the decision to use the same area for all was a natural one. Having made that decision, there immediately arise two important considerations:

1. What to do with the currently executing tables before overwriting them with the next set
2. How to accomplish the solution to 1 within a single action

To find the solution to these problems we must first examine the nature of the information in use during a sub-module's execution. It is for the most part static, i.e. values do not change during table execution.

Exceptions to this are the state vector, which defines the current logical situation, and the active strategy indicators. All that is necessary then, is to place these on a stack for later recovery, before generating those for the called sub-module. All the static tables can be re-read from their normal storage locations when execution of the lower level tables has terminated. The length of the state vector, however, is also variable, its current value being held in the PDT. It too must therefore be placed on the stack.

The operations required to effect the transfer and recovery of control then number eleven:

1. Stack the PDT
2. Stack the state vector
3. Identify the location of the called sub-module's tables
4. Read the new PDT
5. Update the component descriptor variables from the PDT
6. Read the called sub-module's component tables
7. Execute the mini-driver
8. Unstack the PDT
9. Update the component descriptor variables from the PDT
10. Unstack the state vector
11. Read the current sub-modules component tables

It is apparent that within these eleven operations, two only (3 and 7) are peculiar to the called sub-module. All others are manipulations of the system tables. It would be unreasonable (if not plain folly) to encumber the user with the maintenance of data structures of which he or she is unaware. The problem is handled by asking the user to make use of a single new statement to express the requirement to transfer control. The decision to implement the EXECUTE statement was not taken lightly. An alternative would have been to trap occurrences of the regular subroutine call statement and interpret them as table transfer requests. To have done so would of course, have excluded the use of normal subroutines within the aegis of the system. This would have so violated the expressed objectives as to render the whole system virtually worthless. The concept of a new operation (table execution) warranting a new statement was seen as a salve to the conscience.

Logic Testing. One of the functions to be provided by the editor would be a logic testing facility. By use of a suitable command (TRACE), the user should be able to call up any (one or more) clusters comprising the total table for display on the terminal. There cannot of course be any provision for evaluating a real state vector (which after all, is actual execution), but by examining the logical situations open, selection can be made of the rule to be "executed", whereupon the system will step through the actions appropriate to that rule. From this, the user will then be able to deduce which situation will be met next.

With a user's tabular information assembled, only two further sets of information would be required to provide the trace operation: predicate statements and action names. The former are not actually indispensable, since numerical identification of so few conditions hardly represents a major problem. However, clarity is added to the cluster display when condition statements are included. One of the design requirements was that logic testing should be possible before action code has to be made available, hence the desire for named actions. To keep the required list of names in memory would place too great a burden on the system; and besides, what more natural than to use these names as action identifiers within the source code when it becomes possible to generate the sub-module's subroutines.

<u>Data Flow.</u> With the introduction of a new source language instruction (EXECUTE) to effect control transfer, comes an awareness that the system must also provide its own means of controlling the passing of data items from one sub-module to another. The mini-drivers which will be inserted by the system upon encountering an EXECUTE statement will be, except for the name of the called sub-module, entirely independent of the user's perceived programming requirements. How then are data items in one sub-module to be made available to another in a controlled fashion?

It has always been a requirement that the system should be seen as an extension to the user's programming powers, and not a restriction. Thus it is necessary to ensure that any system implementation does not impinge upon the range of source statements normally available.

The aforementioned data dictionary was therefore introduced to organise the storage of declared global data items. It will still fall to the user to identify the items required globally, as to name, type and length; but one need not be concerned with storage location, boundary alignment, etc. This collection of data element information has to be maintained at a project level, i.e. available to each sub-module comprising the project. Since it will be the only truly global component in the user's project, one outstanding non-data-related task is assigned to it: to maintain an identification list of all tables comprising a project, and as they are created, assign to them the file identification number which will be linked to the Operating System's I/O facility at execution time.

<u>User Interface.</u> Sitting at an interactive terminal, the user knows that various building blocks are available to construct a problem solution.

Steps 1 - 4d (Chapter III) the planning/analysing steps, have now been completed. How is the system able to be manipulated to produce an actual solution? Four commands are required to be recognised by the editor for the purposes of creating or modifying a sub-module, testing a sub-module's logic, and terminating system activity. Processing options available as sub-commands to two of these will then permit management and examination of the various components of the particular sub-module. The user interface package must then be able to:

1. Create a new sub-module skeleton

2. Modify an existing sub-module
    - build/modify the cluster table
    - build/modify the rule table
    - build/modify the action table
    - build/modify the strategy table
    - perform source code oriented functions:
        enter/modify action source code
        enter/modify predicate source statements
        enter/modify local data item declarations
        generate source programs representing the sub-module
    - terminate sub-command activity

3. Test the logic of a sub-module
    - qualify one or more clusters for testing
    - display actions corresponding to qualified cluster(s)
    - display actions corresponding to a selected rule
    - terminate testing

4. Terminate system activity.

With the above command structure, our user is able to create the sub-modules required to produce the problem solution. Upon completion of a sub-module's construction the generation of the subroutines required to represent it for the purpose of execution can proceed. Compilation of the set of subroutines so generated should then produce an executable module which, when loaded with the system driver and the application's tables, provides the desired computerised solution to the specified problem.

V. IMPLEMENTATION

Environment. The proposed system would be built and would operate on an IBM 370/158, under control of the MVS Operating System.

The language of implementation would be that most familiar to engineers in the organization for which the system was to be built, and which was supported at their computer centre. Thus Fortran was to be the choice, with only a few basic utility routines, where necessary, coded in 370 Assembler.

The Driver. The ever-present heart of the System, the Driver, operates as such for "System-Driven" and "Application-Driven" alike. Its functions therefore can only comprise those common to both (Figure 12); but we see the simplicity of that skeletal procedure fade somewhat when translating it into source code. Fortran's lack of dynamic storage allocation forced the need to declare all System arrays within this module and then to make them available to subsequent procedures through argument lists in the usual way. The resulting Fortran source listing of this and other procedures is given in appendix B.

```
Driver:  Begin;
         Read System Tables;
         Prime Data Vector; /*first rule to be executed*/
         Activate first Strategy;
         Do forever;
              Call Driven;
              If Current-rule-strategy-entry is zero Then Exit;
              Else Activate next strategy;
              End;
         End;
```

Figure 12. Main Procedure DRIVER

The First Table. It was decided (perhaps unwisely, as it turned out) that the purpose of the first Driven sub-module was to be to create new sub-modules. The reason was apparent: use it then to build the rest of the System. Although the table's function was clear, the means of achieving this was clouded by the fact that it had itself to be executed in order to provide that function, (the eternal chicken and egg problem). The table components--"In-Units", as they were called--first to be built were to be the Strategy, Rule and Action Tables.

It must be remembered that at this time there was no facility for executing other than the one basic table, there was no data dictionary, and no means for testing the logic of the partially completed work. It should perhaps have come as no great surprise that a table to fulfil the declared task should have turned out to be so large. Had the full system been available to produce this table, it would undoubtedly have been divided into four or five smaller ones. Tausworthe [8] points out that tables based upon more than six predicates tend to become unwieldy; and that was certainly borne out here, where the largest of the ten clusters involves nineteen conditions. The complete set of tables that were written to provide a table creation facility, is presented in appendix B.

The Strategy table comprises two distinct parts: the numerical part containing the guards which are required for execution of the sub-module, and the titles which are used for convenience only during table building. Although, in general, the Strategy table would be sparsely populated, it was decided in view of its small size, that at least initially it would be kept as a full two dimensional array storing coded character (EBCDIC) representations of the strategy guards. In this way, a blank entry renders its corresponding logical situation inactive, while a non-blank guard activates it. Having thus disposed of the guarding function, the value of the guard may now be used to activate the next strategy. A zero entry terminates execution of a sub-module, returning control to one level higher. The titles are simply stored in EBCDIC in a two dimensional array; but as at this stage only the executable functions were being considered, they were, for the time being, ignored.

The statements of logical conditions, or predicates, are binary in nature, evaluating to True or False. The union of all conditions required to enable the individual clusters within a table, forms the set of table conditions; and each instance of its set of related values is a vector of binary values defining a logical situation or rule. The set of all such vectors relevant to a table forms a situation or rule table (Figure 2). It would have been simple to store this in an array of logical variables containing zeros and ones (False and True), but feeling that the user would be more naturally inclined to think in terms of "Yes and "No", it was decided to store EBCDIC characters again. The set of characters permitted to define a rule table is Y, N, $, * and blank, representing respectively True, False, Don't care but actually False, Don't Care but actually True, and Don't Care. This in effect represents the set of characters which has become associated with conventional decision tables [9]. In order to allow for use of the Don't Care value, it is necessary to maintain an array of binary masks corresponding to entries in the rule table. The user is unaware of this, as it is created and maintained from first to last by the system. But at the outset, of course, it was necessary to create it by hand too.

To have stored the action table in the form that was represented in Figure 4 would have been extremely wasteful of memory, since it was likely to be very large and very sparsely populated. So for each situation, there is created an ordered set of numbers referencing the appropriate actions. In this way, the array size required is kept to the number of situations times the length of the longest action list.

Writing of the Fortran subroutines to represent the first sub-module proceeded more-or-less concurrently with the construction of the table components. It was a simple matter therefore, to arrange that what appeared in the tables as action one, was actually action one to the executing subroutine. The prospective user was not going to have this insight into the arrangement of the final source subroutine, so provision had to be made to allow every action to be accessed by two different numbers: one being the user-supplied external action number, and the other being the internal system action number used for execution purposes. An Action Translation Table was introduced to permit the system to obtain its required action number from the user's. The user would then be free to supply each simple action source entry totally independently from any other. Each entry in the Action Translation Table identifies for one external action number, the location of an entry in an Internal Action Number Table. One of the items of information stored in such an entry is the sytem's internal action number.

Having decided upon the implementation methods for the three component tables it now remained to design the Program Descriptor Table (PDT) itself, before any of the other information could be stored. The file which was to hold the various components of a table would be divided internally into blocks of length 1500 bytes. A single block would then be sufficiently large to hold the entire PDT, while manageable units would be provided to store table components of any size. Information stored in the PDT is used either to describe the "physical features" of the components of the tables--for use at execution time--or to describe the contents of the table components to permit or facilitate their construction. A small utility program was written to create the "execution" parts of the PDT (Figure 13), and to write it and the three table components to disk. Compilation of the necessary action and predicate code completed this primitive sub-module, and within a very short time a table-building tool (albeit crude) was ready for use in the next step of the development.

		Location
Table Name	F	1 - 2
In-unit sizes	XF	3 - 18
In-unit presence indicators	F	19 - 20
In-unit storage block directory	XF	21 - 70
Own Fortran I/O File Number	X	71
Compound Action Identifiers	F	72 - 100
Rule Description Information	F	101 - 150
Input Formats for Rule Reading	F	151 - 253
Source Code Location	F	254 - 260
Cluster usage information	F	261 - 320
Compound Action Number lists	F	321 - 375

Figure 13. Information stored in the PDT. Items flagged with an 'X' are those required for execution of the table, while 'F' indicates those concerned with the functions it should perform

Provision had to be made at this stage for the building of the Cluster table. This is the only In-unit whose purpose is divorced from table execution, being used solely during rule and action table building and logic tracing. Its structure is almost identical to the strategy table, comprising title array and numerical array. Within the latter a non-zero value indicates membership of a specific situation in a specific cluster. Sample listings of the various In-units are included in appendix B.

Table Transfer. One operation not directly concerned with component building, but nonetheless unavoidable at a very early date, was storage block directory maintenance. The various in-units had to be correctly located in the table file, and the storage directory portion of the PDT adjusted accordingly throughout component building and modification. As there was no means of transferring control from one sub-module to another, this function had to be provided, in the early days, by a normal Fortran subroutine call. It was a simple matter to re-write this in table form; and being quite small it lent itself well to being the target for table transfer.

As discussed in Chapter IV, one new source statement ("EXECUTE sub-module-name") was to be the only deviation from the normal Fortran, providing the user with the power to transfer execution control from one table to another. It was also shown that the outcome of encountering such a control transfer request was to be the execution of eleven distinct Simple Actions. Hitherto, there had been a one for one relationship between the user's named (and numbered) actions and those actually executed by the system. Here was a requirement for a compound action; where the user sees a single action while the system "sees" eleven. Figure 14 shows how the one-for one relationship was expanded to permit a one for many relationship to exist. The user's action 6, through the Action Translation table, locates the head of a linked list of internal action numbers which, at execution time is followed to

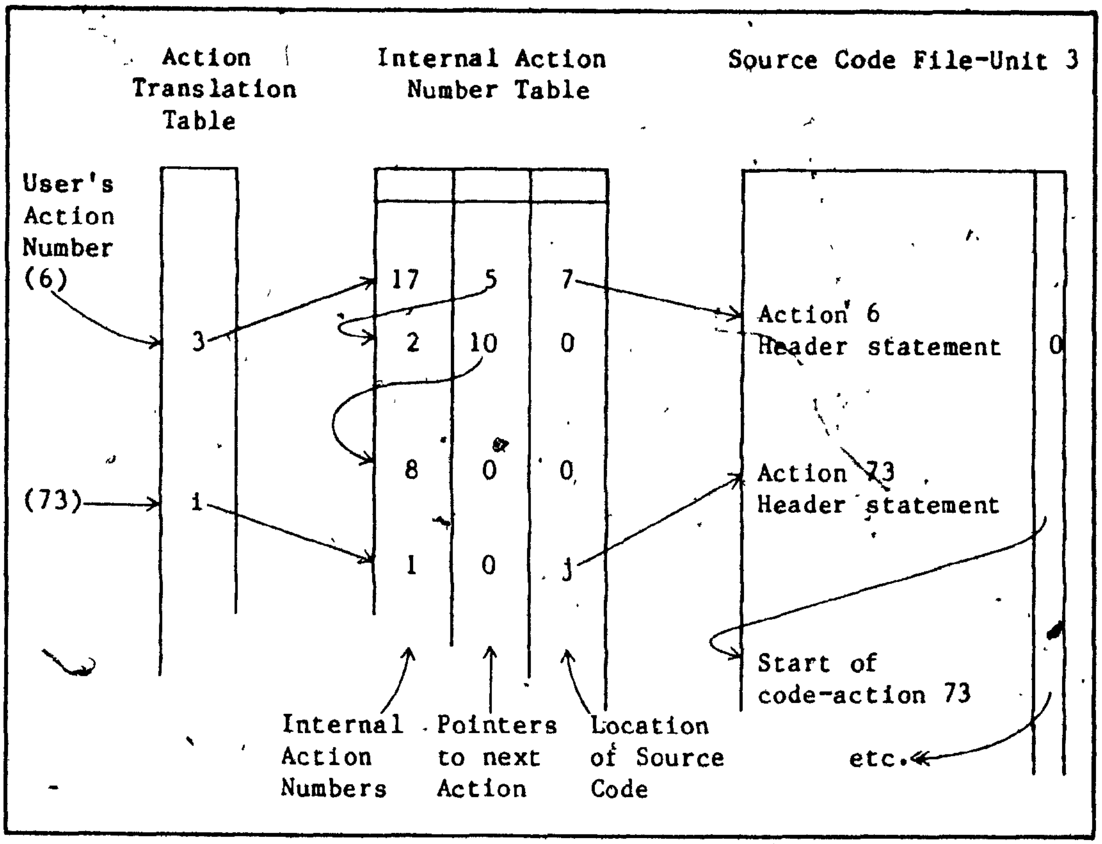


Figure 14. The relationship between user's action number, internal action numbers and source code location

termination. The Internal Action Number table is also used during the building and modification stages, to locate the action source code.

Associated with use of the EXECUTE statement is the need to maintain control of state vectors during control transfer. A Fortran sequential access I/O file is provided for the purposes of stacking and recovering status information.

<u>Action Tracing</u>. Now that table transfer was possible, there were virtually no limitations on the sub-tasks to be developed for processing by the base sub-module. The most pressing need at that time, was for a means of verifying the logic of a sub-module's tables.

All the information necessary to support this operation was now in effect, with the exception of predicate statements and action names. Both of these are held in the user's source code library. Predicate statements are located in the library, from an entry in the PDT. The action names, actually Fortran comment statements, are located from the Internal Action Number Table (Figure 14). Invoking the TRACE command, now built into the user interface, initiates what is essentially the same sequence of operations required at run time to perform the real program execution. With a selected rule identified, here directly by the user, the set of action numbers corresponding to that rule is located. But now, instead of transferring execution control to these actions, the system merely lists their external action numbers and identifying comment source statements. An example of the TRACE command in use will be found in appendix A.

The Data Dictionary. It had never been an intention to rewrite the Fortran compiler, nor to restrict the user in the range of Fortran statements available. So a means had to be found within Fortran's capabilities, of performing a task which would directly affect storage allocation within a user's program without causing him/her any inconvenience. So use of the blank common storage area was ruled out, as was any plan which was based upon a single agglomeration of different data types. Neither was it seen as desirable to make all data elements available to all sub-modules indiscriminately. This left use of labelled common storage areas as the likeliest possibility, with collections of each data type being maintained under their own separate labels. Passing of data elements through common storage areas is of course one of the most frequently occurring sources of error in Fortran programs; now that the common area is any more to blame than the bad workman's chisel. What was needed then, was a means of relieving the user from the responsibility of maintaining these storage pools.

Fortran Direct Access unit 8 was reserved for the data dictionary. It is divided into fixed length blocks of 400 bytes. The first three of these are reserved for system use, and the remainder are available as required by the user's project.

1. System area

   a. Block 3 contains: the overall project name; the number of tables comprising the project at any instant; and for each table present, the name, I/O unit number and modification indicator

   b. Block 1 is used to store data type names, their corresponding COMMON labels, and any aliases by which a type may be identified. It also contains an "in-use" block map to access the user's area of the file, and to indicate block access sequence

   c. Block 2 is divided into eight areas, each corresponding to one of the permitted data types. The system uses these areas for lists of tables requiring each type. Provision is made for linking to continuation blocks if more than twelve tables reference any single data type

2. User's area

   a. A single block for each table, referenced from the PDT, contains a list of references to data items used by its sub-module. The references are by block number and offset to the blocks described in b. below

   b. Blocks are allocated as required, to store data element information. This comprises: name, type code number, length in words, and dimension information stored in character form for array items

A sample arrangement of entries in the data dictionary is shown in Figure 15.

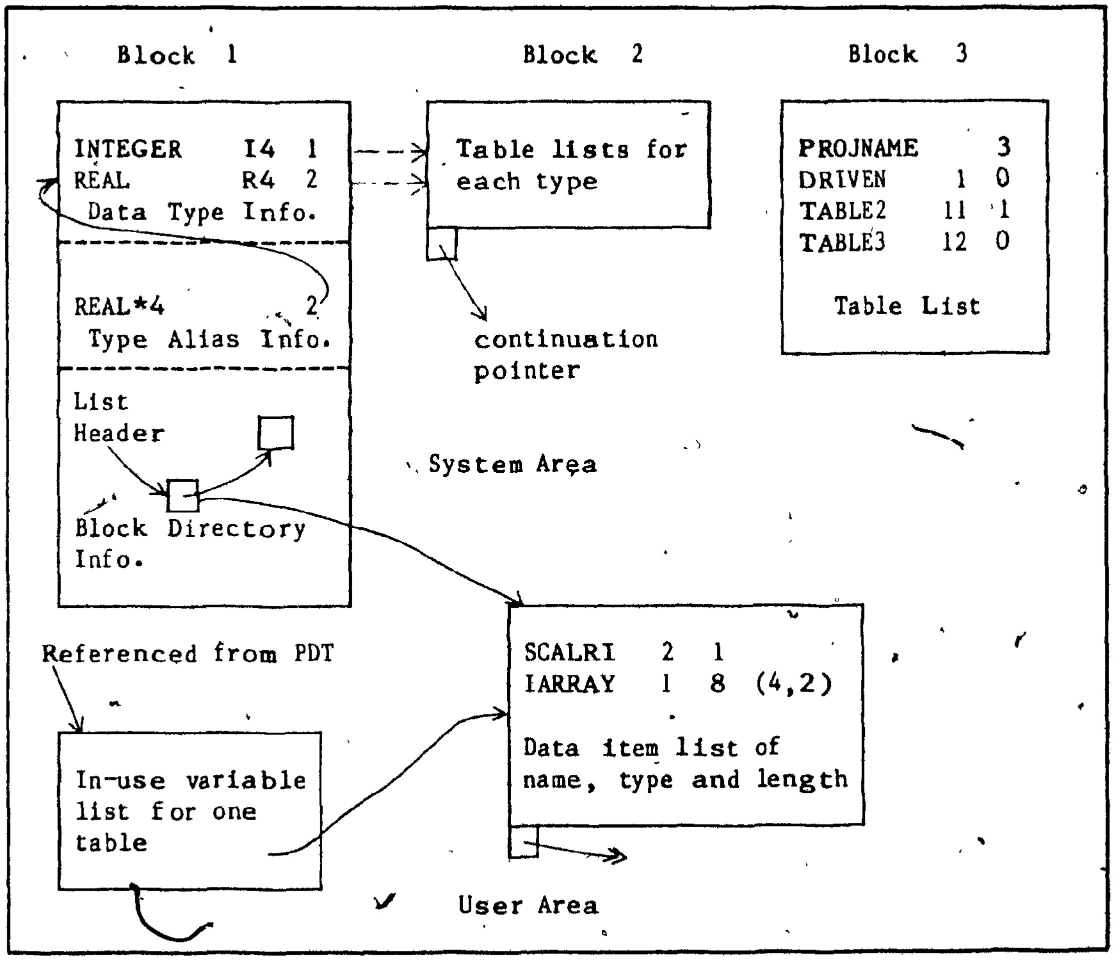


Figure 15. Arrangement of information in the Data Dictionary

Whenever the system is initiated, block 3 of the data dictionary is immediately inspected. If the first storage location is empty, a new project is assumed to be starting and its name is requested. In this instance, the name DRIVEN and Fortran unit 1 are assigned to the first sub-module. The user has no choice in this matter as it is imperative that the base sub-module be correctly linked to the driver at execution time. Subsequently occurring sub-modules are named to suit the user, and Fortran I/O unit numbers are allocated to these sequentially from 11. Associated with each table name is a modification indicator which initially off, is turned on whenever a change in the data dictionary would necessitate generation of new subroutines for that sub-module.

Apart from the aforementioned start-up procedure, the data dictionary is only accessed during global data declaration, and code generation. Information to be entered in the data dictionary is supplied in the form of normal Fortran type declaration statements, e.g.

INTEGER ARRAY1(20,10),P,Q,Z(100)

As this is the only means of entering global variables, array names must appear with their dimension details. From entries of this kind, the system identifies the type, and ensures that the current sub-module's file number appears in the table list in block 2 appropriate to that type. The table's modification indicator in block 3 is turned on. Details of each item in the statement are then extracted, checked and entered in one of the user area blocks and cross referenced through the current sub-module's own variable list block. Addition of any new data item to a type list causes the setting on of the modification indicators of all tables using that data type.

The Source Code Library. Apart from global data information, which requires special processing, all source statements or statement fragments for a sub-module are stored on Fortran Direct Access unit 3. A block size of 84 bytes provides enough space for a full card image, and also for a linking pointer outside the "card" boundaries. Action, predicate and specification code is stored wherever space is available in the file at the time of its entry. There is a logical sequencing of code within each of these three groups, provided by the pointer words.

Each simple action comprises: a header comment card, created from the name provided at action creation, and a collection of zero or more Fortran statements to perform the required action. The group forms a linked list with the header statement referenced from the Internal Action Number Table (Figure 14).

Code for local data declaration statement and predicate statement groups is stored as if each group were a simple action, except that reference to each is directly from the PDT.

Use of the compound action EXECUTE is handled somewhat differently. A single unit 3 record is allocated to the action header, which is written to that file only for the purpose of display during a TRACE operation. Although no other Fortran code is involved at this stage, the Internal Action Number Table is modified in anticipation of the eleven simple actions which will be created at code generation (Figure 14). Source code for the system generated actions is kept in a system source code library file (unit 4), having an identical structure to that of the table's. Information on how this code can be located is placed into the user's table's PDT prior to user's code input.

Code Generation. After verifying the logic of a problem solution, and having assembled a "library" of actions, and predicate and specification statements, the time has come to complete construction of the application sub-module. This is effected with a request to the system to GENERATE the appropriate Fortran subroutines. Before it does so, the contents of the PDT are checked to ensure compatibility between table components. This is necessary in view of the fact that a user has been free to assemble the components entirely independently of each other. It would obviously be wasteful to spend time on code generation if the number of predicate statements supplied, for example, differed from the number of conditions upon which the rule table was based. Any discrepancy between components gives rise to a printed warning of the potential error. With no obviously conflicting information, two subroutines are created on unit 7. The first, identified by the name of the table, represents the action code. The second, bearing a system-created name, is the state vector evaluation routine incorporating the predicate statements previously assembled.

To build the abstract machine, which is in fact the action subroutine, requires six distinct groups of source code. There are three blocks of system code, two optional blocks of user code, and one of mixed system and user. These are identified as:

1. System specification (including the SUBROUTINE) statements
2. User's global data specification statements (optional)
3. User's local data specification statements (optional)
4. System code to control action execution
5. User code for each action, encapsulated in system code
6. System code to close control loops and terminate subroutine

Items 1, 4 and 6 never change except in respect of the table name and the number of actions used. It is therefore a simple matter to copy the required code from the system's source library file (unit 4). To generate item 2, the table's global variable list block is located from the PDT and loaded from unit 8. Each possible data type is examined to determine if the current table uses it. For each one used, the global variable list is examined for entries matching in type and table number, and length information is extracted to build a type specification statement and a labelled common statement. Correct location of stored values is ensured by providing dummy items with system-generated names to match the length of the unreferenced items in the common block. User's local specification statements are copied directly from the table's source library (unit 3).

Action code, constituting the bulk of this subroutine, also represents the major part of code generation. Simple action code from the table's source library is sandwiched between system control statements providing the entry and exit points for the action "capsule", and isolating it from its neighbouring actions. Comment statements are included to allow easy understanding of the generated subroutine.

Generating code for the EXECUTE statement entails a more complex operation. When source code is first entered, details of where to find compound action code is written into the table's PDT. Using this information, together with an EXECUTE statement usage indicator, the system is able to locate in its own source library, the simple action code required to construct the compound action. The first occurrence of an EXECUTE, causes inclusion of the full set of actions described in the source code section, as the action list in the Internal Action Number table is followed. Subsequent use of the same statement requires only two new actions per table called,--one to identify the appropriate file number, and the other to provide the mini-driver itself.

Upon completion of the action subroutine, the state vector evaluation subroutine is constructed from the predicate statements input earlier.

<u>File Organisation</u>. The operations involved in achieving execution of a sub-module will be:

1. Build the tables
2. Modify the tables
3. Test the logic
4. Supply the source coding for predicates and actions

Items 2, 3 and 4 may occur in any sequence and may be repeated as often as required.

5. Generate the source deck

   (Compile the source--outside the scope of this System)

6. Execute the sub-module

The disk-resident dataset requirements for the various steps differ, and are summarised in Figure 16.

File \ Operation	1 Build Tables	2 Modify Tables	3 Test Logic	4 Add Source	5 Generate Program	6 Execute Sub-module
Load Module-DRIVER (S)	x	x	x	x	x	x
Load Module-DRIVEN (S)	x	x	x	x	x	
Tables (S)	x	x	x	x	x	
Source (S)				(x)	(x)	
Tables (A)	x	x	x	x	x	x
Source Fragments (A)			(x)	x	x	
Data Dictionary (A)				(x)	(x)	
Source Program (A)					x	
Load Module-DRIVEN (A)						x

Figure 16. Disk resident files required during execution of the various system operations. Those marked (x) are optional at that stage. Suffixes (S) and (A) indicate System and Application respectively

Each complete table and its associated source code constitutes a sub-module which "owns" at some stage or another, one of each of the files designated "Application" (Figure 16), with the exception of the Data Dictionary file which is global to all sub-modules comprising a problem solution. Load modules exist as members of a partitioned dataset [23] and all others are Fortran Direct Access data sets.

VI. OPERATION

The best way to introduce the user's side of the system, is probably by its application to a problem. Before doing so, however, a brief overview of the commands and sub-commands available might be considered appropriate. Appendix A contains a table of options open to a user, and the following text is intended to supplement the information presented there.

It is not intended here to develop a full solution to an engineering problem, but rather to show how such a solution can be approached with the aid of the system described. Let us say we wish to produce a program which can model, analyse, design, remodel etc., all, or components of a guyed microwave tower. A first level analysis of the problem may produce the result shown in Figure 17. For the purposes of this example, only the Rotational Analysis task will be developed. The level 1 tasks are thus the only ones of importance here, although some early thoughts on the breakdown into lower level tasks has been indicated. In practice, of course, greater consideration will be given to these sub-tasks, as groupings of similar types of operation might well suggest a different breakdown in the higher level tasks.

In order to be able to demonstrate a part solution,—and, too, to make it more interesting,--a number of assumptions has been necessary. The first is the existence of a data base in which is stored all information pertinent to a tower: the original numerical model, the deflections and forces after analysis, applied loads, updates to the model etc. Storage and retrieval of information to and from the data base is handled by a group of utilities which together constitute the assumed sub-module GETDAT.

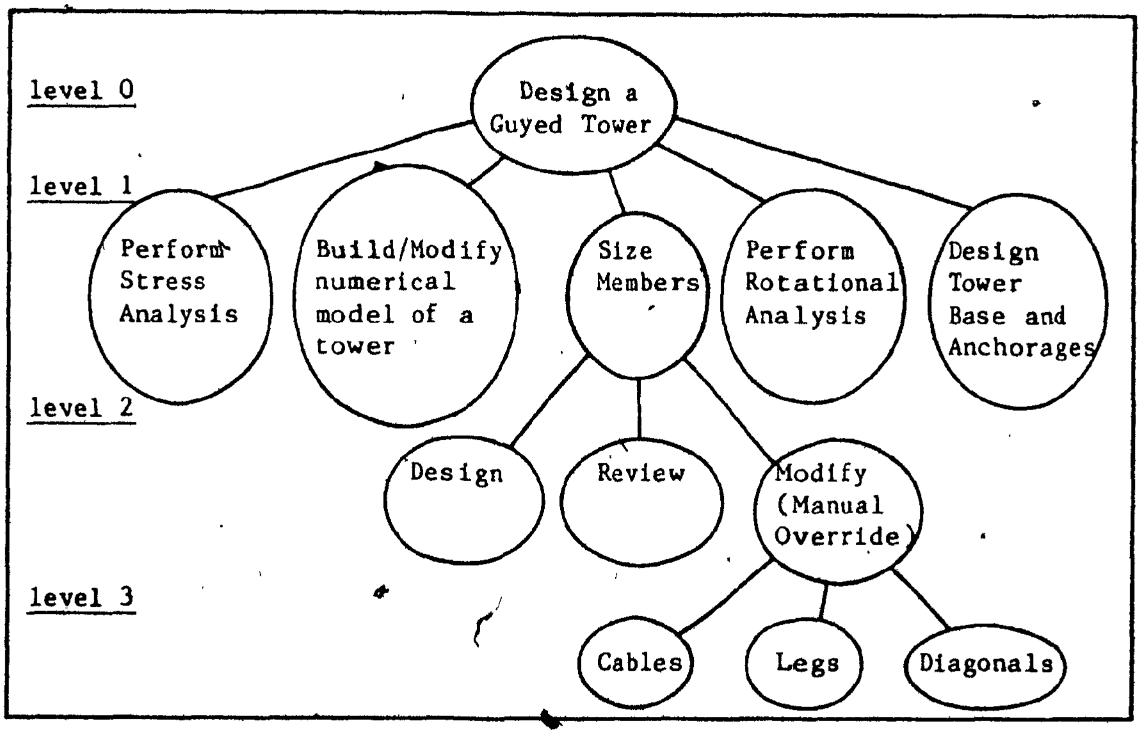


Figure 17. Proposed solution to the tower design problem (incomplete)

Further study of the rotational analysis task has now produced the result shown in Figure 18. Details of the engineering theory upon which this solution is based are not included here, but briefly, the steps involved are as follows. The initial tension in each of six supporting cables is known. From this, the horizontal component of that tension at the upper end of a cable can be determined by successive approximations. An assumed rotation of the tower causes three of the cables to stretch and the other three to relax, resulting in changes of force in the cables. The force/deflection relationship is non-linear, and here, for a pair of cables, it is solved by Newton-Raphson method. The torque required to produce the assumed rotation is then calculated from the difference between the horizontal components of the modified cable tensions, and is compared with the actual applied torque. An iterative solution is required, again using successive approximations.

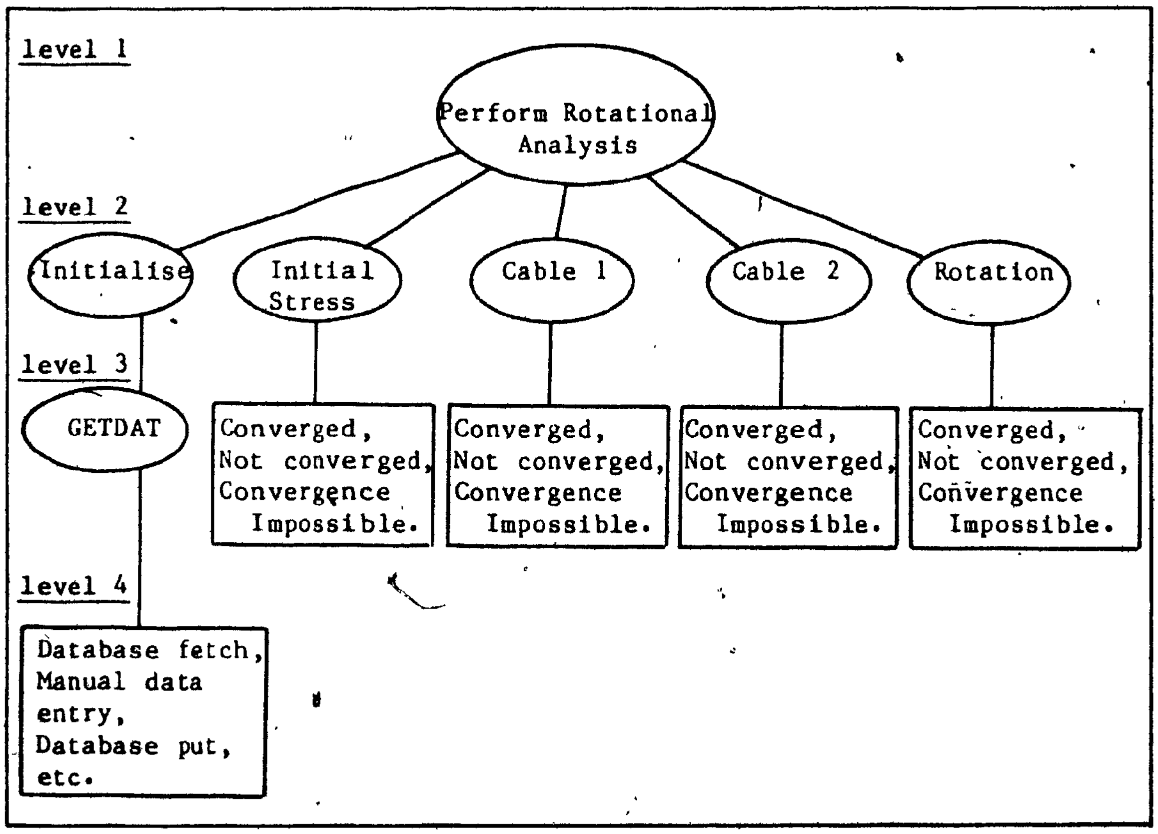


Figure 18. Subdivision of the rotational analysis task

Apart from the data base functions (to be handled by the previously introduced sub-module GETDAT), the level 2 sub-tasks (Figure 18) show a marked similarity. It is at this point that the decision is taken to group these four operations within a single sub-module (ROTATE), to take advantage of any possible code sharing.

The form of the proposed problem solution (Figures 17 and 18) now maps directly to the program structure shown in Figure 19. The sub-modules not being considered here,--STRESS, BLDMOD, SIZE and BASE,--will in reality be considerably more complex, certainly all requiring access to GETDAT. For the time being, they are simply assumed to exist.

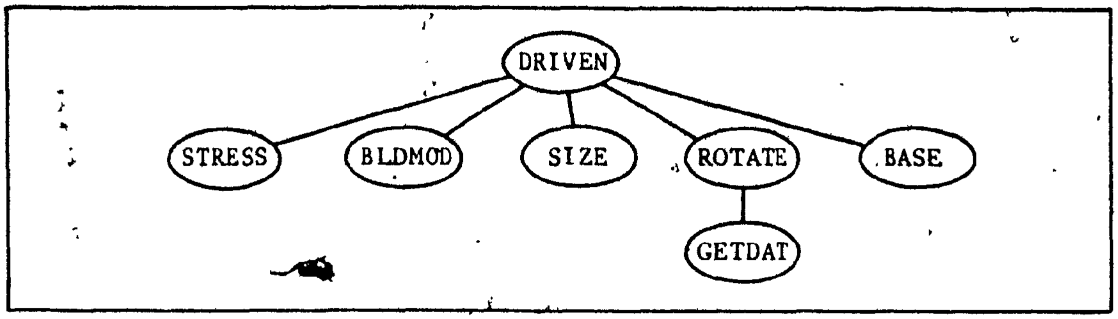


Figure 19. Conversion of proposed tasks into sub-modules. Note: again, incomplete--only the DRIVEN-ROTATE sub-modules are planned to any depth

The design of sub-module DRIVEN is quite straightforward, as it does no more than drive the appropriate level 1 sub-modules. Its simplicity suggests that a single cluster will be sufficient. For the sake of convenience, however, two are assumed; one for sub-module initialisation and the other to perform the required task. Initialisation simply comprises examining the status of information in the database, and reading the user's first command from the terminal. The contents of the one rule which comprises this cluster are immaterial as it will be executed once only upon loading the program. Its guard must be "open", and must activate the other cluster. A tabular representation of these requirements is shown in Figure 20. It will be noticed immediately, that although the cluster "CHECK COMMAND" comprises fifteen rules, guards only exist for five of them. In this way, although the sub-module will be built in its entirety, there is no intention of permitting other than these five rules to be considered for

	Cluster / Rules:	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
	Cluster																
1.	Check Command	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
2.	Initialise																x
	Strategy																
1.	Check Command	0								1	1	1				1	
2.	Initialise																1
	Predicates																
1.	Command .EQ. 'END'	Y														N	-
2.	Command .EQ. 'NEW'		Y													N	-
3.	Command .EQ. 'BUILD'			Y												N	-
4.	Command .EQ. 'STRESS'				Y	Y	Y									N	-
5.	Command .EQ. 'SIZE'							Y	Y							N	-
6.	Command .EQ. 'ROTATE'									Y	Y	Y				N	-
7.	Command .EQ. 'BASE'												Y	Y	Y	N	-
8.	D-b Descriptor 1 (Model Defined)				N		Y			N		Y	N		Y		-
9.	D-b Descriptor 2 (Loads Defined)					N	Y				N	Y		N	Y		-
10.	D-b Descriptor 3 (Stresses Computed)							N	Y								-
11.	D-b Descriptor 4 (Members Sized)																-
	Actions																
1.	Read a Command		4	3	2	2	3	2	3	2	2	3	2	2	3	2	2
2.	Get Database Descriptors			2			2		2			2			2		1
3.	Clear Database Descriptors		3														
4.	Read a project title		1														
5.	Clear Database (new project)		2														
6.	No-op	1															
7.	Execute sub-module BLDMOD			1													
8.	"No analysis - incomplete model"				1					1			1				
9.	"No analysis - no loads"					1					1			1			
10.	Execute sub-module STRESS						1										
11.	"Sizing impossible - no stresses"							1									
12.	Execute sub-module SIZE								1								
13.	Execute sub-module ROTATE											1					
14.	Execute sub-module BASE														1		
15.	"Illegal Command - re-enter"															1	
	Rules:	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16

Figure 20. Tabular representation of the problem requirements for sub-module DRIVEN

execution at this stage. With such a simple sub-module, no testing is presented here; indeed, none was carried out. Appendix A contains copies of the terminal sessions in which this sub-module was built, together with source listings of the generated Fortran subroutines.

Sub-module ROTATE is rather more interesting. Figure 21 shows it in tabular form. Although the complete sub-module is shown "assembled" (The System's view, Figure 4) for compactness, it was in fact designed and built one cluster at a time. Of course, when it came to cluster four the similarity to cluster three was obvious, and immediate use of the latter was made.

It should be realised that although Figure 21 shows the final table components (for the sake of completeness), it did not come from the pencil in quite this form. The process of building the required in-units is shown in appendix A, complete with the original errors. Following the entry of the action titles, testing with the TRACE command is demonstrated. Upon invocation of TRACE, a display of all clusters present is found useful. In testing a large sub-module, it may be necessary to examine only a small portion of the tables, just one cluster for instance. Alternatively, a comparison of two or more clusters might be more appropriate. Whatever the requirements, the clusters named or "qualified" form a single unit, and within that unit each rule is identified 1 to n as shown on the display. Here, testing started with initialisation and stepped through the clusters as it was felt that execution should proceed. It became apparent that at cluster three rule three an endless loop would be encountered. Having noted the required correction, tracing continued with cluster four. It was noticed here, that rule six could never be executed, and appropriate changes were marked. The in-units were modified in accordance with the noted errors, and Fortran source code was supplied for each of the actions. As a result of an early oversight, action 23 was not used. Local and global specification statements were provided, and a request to the system to GENERATE, produced the required Fortran subroutines. Sub-module GETDAT was produced in the manner of the above. No details are included as this comprises a single rule "dummy" sub-module, merely supplying the chosen tower data to ROTATE.

Cluster	Rules:	01	02	03	04	05	06	07	08	09	10	11	12	13	14
1.	Initialise	x													
2.	Initial H		x	x	x	x									
3.	Cable 1 Stretch				x		x	x	x	x	x				
4.	Cable 2 Relaxation				x		x		x	x	x	x			
5.	Rotation												x	x	x
Strategy															
1.	Initialise	2													
2.	Initial H		2	2	0	3									
3.	Cable 1 Stretch				0		3	3	3	0	4				
4.	Cable 2 Relaxation				0		4		4	0	5	3			
5.	Rotation												3	0	0
Predicates															
1.	Horizontal component unknown	-	Y												
2.	Computed vs Actual tension converged	-		N		Y									
3.	Iteration check 1 OK (H)	-		Y	N										
4.	Horizontal component increment unknown	-					Y								
5.	Calculated K factor acceptable	-						N	Y		Y	N			
6.	New vs Old horizontal comp. converged	-							N		Y				
7.	Iteration check 2 OK (cables)	-							Y	N					
8.	Iteration check 3 OK (rotation)	-											Y	N	
9.	Computed vs Actual torque converged	-											N		Y
	Rules:	01	02	03	04	05	06	07	08	09	10	11	12	13	14

Figure 21 (Part 1). Tabular composition of sub-module ROTATE

	Actions	01	02	03	04	05	06	07	08	09	10	11	12	13	14
1.	Get data from database	1													
2.	Zero Initial H (HZERO)	2													
3.	Zero cycle number NCYCS	3				3									
4.	Calculate HZERO		1												
5.	Calculate RZERO		2	3											
6.	Recalculate cable tension		3	4											
7.	Adjust HZERO			1											
8.	Increment NCYCS			2				2				4			
9.	"Cables too slack"				1										
10.	Initialise dL (DELL), F					1									
11.	1st Cable (CABLNO=1)					2						3	5		
12.	2nd Cable (CABLNO=2)										2				
13.	Zero cycle number CABCYC					5					3		4		
14.	Increment CABCYC								2						
15.	Zero dH (DELH)					4					5				
16.	Stress increasing (FAC=+1.)					6						2	3		
17.	Record DELH,H for cable 1, FAC=-1.										1				
18.	Initialise DELH and OLDELH						1						6		
19.	Stress-Cable EL,LBAR,H						2	3					7		
20.	Compute cable factors K,K' (KFAC,KPRIM)						3	4	3				8		
21.	Halve DELL (KFAC too big)							1				1			
22.	New dH (DELHNU) and DELH								1						
23.	not used														
24.	"Cable calcs. not converged"									1					
25.	Zero cycle number DEFCYC	4													
26.	Increment DEFCYC												1		
27.	Compute difference of cable forces DIF										4				
28.	Adjust DELL on basis of DIF												2		
29.	"Deflections not converged"													1	
30.	Compute and print rotation														1
31.	Initialise starting DELH values STRTDH					7									
	Rules:	01	02	03	04	05	06	07	08	09	10	11	12	13	14

Figure 21 (Part 2). Tabular composition of sub-module ROTATE

After compiling and linking to the system's own driver, the first run produced errors indicating that initial values for the variable STRTDH had not been provided. Action 31 was therefore added to the source code file and the ROTATE routines regenerated (appendix A). This time the test run produces correct results, and the program is complete.

VII. CONCLUSION

One measure of the success (or failure) of a project of this kind, is whether or not it meets its original objectives. Of overwhelming importance in industry, however, is how well it is received by the users for whom it was intended. This of course reflects the quality of the original specification of those objectives. After all, what is gained from the design of a new ceiling brush if the operator has to stand upside down to use it?

Here, the working system is founded firmly upon the requirements for modularity, power of logical construct, user's single viewpoint, module editor and both static and dynamic testing capabilities. This is not to say that all desirable features have already been implemented. Most have; but those which have not, by virtue of the system design, can be built into the existing structure.

What of future developments? New editor features will have to be added to the system, while existing ones require improvement. In the latter category come, for example, use of the cluster table to support the user's building of the strategy table. Also planned, but deferred on the grounds that in itself it constitutes a major design task, is the use of a single data base for the storage of all information (source code and tables) pertaining to a single project. In this way, one Fortran I/O unit will fill the requirements for both building and executing a program, and the user will be freed entirely from having to match unit numbers and table datasets at execution time.

Items in the "new features" category will certainly have precedence over the "improved performance" ones, however. The two which spring most readily to mind, and which rate evenly in importance are a LIST command, and a sub-module component editing capability. Neither of these presents serious difficulties as each is related closely to the building functions already implemented. Also required, with some degree of urgency, are the CODE sub-commands ALOC and PLOC for entering and modifying local data declaration statements. Data dictionary functions must be extended to allow for modification of the variables list entries and table references to them.

The objectives in view while designing the system's testing facility have, too, for the most part been achieved. By invoking the TRACE facility, the user is presented with a powerful tool to assist in static testing procedures. The single methodology chosen for the likelihood of its being the most effective--peer code review--is now well supported by the reviewer's ability to perform computer-assisted code walkthroughs/inspections. Subroutine interface control is exercised by the editor and thus maintains parameter consistency. Data flow between sub-modules is controlled through the data dictionary, but here further development is required to extend its capabilities. Introduction of data classification into input and output types, together with range-checking values, where appropriate, will permit checking for erroneous value assignments and uninitialised variables.

Of the dynamic testing methodologies, the test coverage analyser was used extensively throughout the early development stages, being the only means available of testing the inner workings of the modules.

Presentation of the analyser's output requires refinement, and a minor modification would permit logic tracing during execution.

Data tracing, assertion checking and dumps are not at present implemented, although the earlier-mentioned modifications to the data dictionary would support these features. Exercising control of the dynamic testing/debugging facility through another table, much in the same manner as the strategy table, will provide these services for the user as and when he/she requires, without involving further compilation of the source code and without impairing program run-time efficiency when they are no longer required.

It is not until all the above features have been developed that we can expect to elicit from the users a valid response to the question "how successful?". What is certain though, is that here is a powerful new tool to assist engineers right from awareness of a problem, through the definition of requirements, the application programming stage, to ultimate solution; the whole requiring no change of viewpoint. With the testing and debugging facilities available, this represents a strong step forward to greater software reliability.

APPENDIX A

USER COMMAND SUMMARY

Command	Sub-Commands	Function
NEW	all as for MOD	Clears the PDT and enters a new sub-module name into the list of Tables (data dictionary - block 3). Clears the source code file (unit 3). Marks all In-units as "non-existent".
MOD		Fetches an existing PDT, with a view to making changes/additions to the In-units.
	CLUS	Permits the building and (later) modification of a cluster table.
	RMTX	Sets up predicate description information which is used to define the way in which a user chooses to present rule entries. This defined, the user is able to build and (later) modify a rule table.
	AMTX	Enables the action matrix to be constructed or (later) modified.
	STRA	Allows building and modification of the strategy tables.
	CODE	Permits access to any of the source code oriented system functions.
	ACTS	Enters input mode for the entry of action headers or source statements. For each action, entries commence with the action number and header statement. If a header already exists for the specified action, it

Command	Sub-Commands		Function
MOD (cont)			
	CODE (cont)		
		ACTS (cont)	is displayed, and entry of action code can now be made. A null line terminates input. If source code already exists for a specified action, it is listed and (later) made available for changes. A second null line terminates the ACTS function and returns to CODE.
		PRED	Initiates input code to receive predicate statements. These are entered as: predicate number and Fortran logical expression (no assignment) representing the required logical condition. A null line terminates input.
		GLOB	Enters input mode and awaits entry of global specification statements. Each line entered must take the form of a Fortran type specification statement, including dimension information for array items, except that it may start in any column. A null line terminates input.
		ALOC	(later) Permits entry of local data declaration statements for the action sub-program. These take the form of ordinary Fortran specification statements. A null line terminates input.
		PLOC	(later) As for ALOC, but for the predicate sub-program.
		GENE	Causes the previously input code and code fragments to be manipulated so that appropriate Fortran subroutines can be generated. Control returns to MOD after source generation.
		END	Terminates sub-command activity, returning to command mode.
TRACE			Permits pseudo-execution of actions to proceed under the user's control, facilitating sub-module verification.

Command	Sub-Commands	Function
TRACE (cont)		
	C n1[,n2,n3...]	n1, n2 etc. are cluster numbers within the sub-module being tested. This sub-command "qualifies" the cluster(s) in the list.
	n	Causes pseudo-execution of rule n in the currently qualified cluster(s).
	A	(later) Displays the action table corresponding to the currently qualified cluster(s)
	END	Terminates TRACE activity and returns to command mode.
END		Terminates use of the system and returns the user to Operating System Command mode.

```
>EPA NEWPROJECT
 NO GENERATE DATASET ASSIGNED - DO NOT USE THE GENER SUB-COM
 CLEARING EXISTING DATA DICTIONARY - OK?  (Y OR N)
>Y
 STARTING NEW PROJECT:  ENTER NAME
>TOWER


 SUB-MODULE:   DRIVEN

                  IN-UNIT                    EXISTS
         RULE MATRIX                           NO
         ACTION MATRIX                         NO
         STRATEGY MATRIX                       NO
         CLUSTER MATRIX                        NO
         ACTION SOURCE CODE                    NO
         PREDICATE SOURCE CODE                 NO


 MOD
>CLUS

 HOW MANY CLUS ENTRIES?  (MAX. 32 ALPHANUMERICS EACH)
 ?
>2

 ENTRY?
>CHECK COMMAND
 RULES?
 ?
>15

 ENTRY?
>INITIALISE
 RULES?
 ?
>1

  CLUSTER TABLE                              (RULES  1- 16)
                                             0123456789012345678901
                                             V         V         V
  1  CHECK COMMAND                            1111111111111110
  2  INITIALISE                               0000000000000001
                                             0         1         2

 MOD
```

Terminal Session. New Project. Clusters for DRIVEN
<u>Note</u>: Lines prefixed by '>' were entered by the user

```
> RMTX

  EACH PREDICATE CODED EVALUATES TO 'Y' OR 'N'. HOWEVER, RULES
       MAY BE ENTERED USING AN EXTENDED ENTRY FORM (INTEGER).
       FOR GROUPS OF DEPENDENT CONDITIONS
  FROM HOW MANY PREDICATES ARE THE RULES DERIVED?
  ?
> 11

  INTO HOW MANY INDEPENDENT FIELDS ARE THE PREDICATES GROUPED?
  ?
> 5

  STARTING FROM PREDICATE NO.1, HOW MANY PREDICATES FORM EACH
       OF THE  5 INDEPENDENT FIELDS?
  ?
> 7 1 1 1 1

  CLUSTERS STORED AT PRESENT ARE:
   1   CHECK COMMAND
   2   INITIALISE

  CLUSTER NAME OR END
> INITIALISE

  ENTER  1 RULES.
  IAAAA
>

  CLUSTER COMPLETE

  CLUSTER NAME OR END
> CHECK COMMAND

  ENTER 15 RULES.
  IAAAA
> 7
> 6
> 5
> 4N
> 4 N
> 4YY
> 3   N
> 3   Y
> 2N
> 2 N
> 2YY
> 1N
> 1 N
> 1YY
> 0

  CLUSTER COMPLETE
```

Terminal Session. Rule Table for DRIVEN

```
 CLUSTER NAME OR END
>END

 MOD
>AMTX

 CLUSTERS STORED AT PRESENT ARE:
  1    CHECK COMMAND
  2    INITIALISE

 CLUSTER NAME OR END
>INITIALISE

 ENTER ,1 LISTS OF ACTION NUMBERS
>2 1

 CLUSTER COMPLETE

 CLUSTER NAME OR END
>CHECK COMMAND

 ENTER 15 LISTS OF ACTION NUMBERS
>6
>4 5 3 1
>7 2 1
>8 1
>9 1
>10 2 1
>11 1
>12 2 1
>8 1
>9 1
>13 2 1
>8 1
>9 1
>14 2 1
>15 1

 CLUSTER COMPLETE

 CLUSTER NAME OR END
>END

 MOD
```

Terminal Session. Action Table for DRIVEN

```
  STRAT

  HOW MANY STRA ENTRIES?  (MAX. 32 ALPHANUMERICS EACH)
  ?
> 2

  ENTRY?
> CHECK COMMAND

  ENTRY?
> INITIALISE

  STRATEGY MATRIX ENTRIES IN THE FORM:
         RULE NO. STRA NO.,CONTROL NO.     (0,0,0 ENTRY TERM
  ?
> 1 1 0
  ?
> 9 1 1
  ?
> 10 1 1
  ?
> 11 1 1
  ?
> 15 1 1
  ?
> 16 2 1
  ?
> 0 0 0

  IDENTIFY START OF EXECUTION LOCATION AS RULE NO. , STRATEGY
  ?
> 16 2


  STRATEGY TABLE                             (RULES  1- 16)
                                             01234567890123456789012
                                             V         V         V
    1  CHECK COMMAND                          0*******111***1*
    2  INITIALISE                             ***************1
                                             0         1         2

  MOD
> END

  INPUT COMPLETE

  COMMAND
> END
```

Terminal Session. Strategy Table for DRIVEN

```
-------------------------
+                       +
+  PROJECT - TOWER      +
+                       +
-------------------------


COMMAND
>MOD


SUB-MODULE:    DRIVEN

                   IN-UNIT                      EXISTS
             RULE MATRIX                          YES
             ACTION MATRIX                        YES
             STRATEGY MATRIX                      YES
             CLUSTER MATRIX                       YES
             ACTION SOURCE CODE                   NO
             PREDICATE SOURCE CODE                NO


THERE ARE   2 STRATEGY ENTRIES SPANNING  16 RULES
THERE ARE   2 CLUSTERS SPANNING  16 RULES
THERE ARE  16 RULES DERIVED FROM  11 PREDICATES
THE ACTION MATRIX COVERS  16 RULES AND  15 ACTIONS

MOD
>CODE

TYPE
>PRED

INPUT
>11 DBD(4)
>1 CMND.EQ.END
>2 CMND.EQ.NEW
>3 CMND.EQ.BUILD
>4 CMND.EQ.STRESS
>5 CMND.EQ.SIZE
>6 CMND.EQ.ROTATE
>7 CMND.EQ.BASE
>8 DBD(1)
>9 DBD(2)
>10 DBD(3)
>

MOD
>END
```

Terminal Session. Predicates for DRIVEN.

```
COMMAND
>MOD

SUB-MODULE:   DRIVEN

               IN-UNIT                     EXISTS
          RULE MATRIX                        YES
          ACTION MATRIX                      YES
          STRATEGY MATRIX                    YES
          CLUSTER MATRIX                     YES
          ACTION SOURCE CODE                 NO
          PREDICATE SOURCE CODE              YES

THERE ARE    2 STRATEGY ENTRIES SPANNING  16 RULES
THERE ARE    2 CLUSTERS SPANNING  16 RULES
THERE ARE   11 PREDICATES
THERE ARE   16 RULES DERIVED FROM  11 PREDICATES
THE ACTION MATRIX COVERS  16 RULES AND  15 ACTIONS

MOD
>CODE

TYPE
>ACTS

ACTION
>1 READ A COMMAND
>      WRITE(6,100)
> 100  FORMAT(' COMMAND')
>      READ(9,99) CMND
>  99  FORMAT(20A4)
>
END ACTION   1

ACTION
>2 GET DATABASE DESCRIPTORS DBD(1)-DBD(4)
>      CALL GETDBD(DBD)
>
END ACTION   2

ACTION
>3 ZERO DATABASE DESCRIPTORS
>
END ACTION   3

ACTION
>4 READ A PROJECT TITLE
>
END ACTION   4
```

Terminal Session. Action Code for DRIVEN

```
 ACTION
>5 CLEAR DATABASE (NEW PROJECT)
>
 END ACTION   5

 ACTION
>6 NO-OP
>
 END ACTION   6

 ACTION
>7 EXECUTE SUB-MODULE BLDMOD
>      EXECUTE BLDMOD
 END ACTION   7

 ACTION
>8 MESSAGE - 'NO ANALYSYI__IS - INCOMPLETE MODEL'
>      WRITE(6,98)
> 98   FORMAT(' NUMERICAL MODEL OF TOWER IS INCOMPLETE - REQ
>     +IS IS IMPOSSIBLE')
>
 END ACTION   8

 ACTION
>9 MESSAGE - 'NO ANALYSIS - NO LOADS'
>      WRITE(6,97)
> 97   FORMAT(' NO LOADS HAVE BEEN SPECIFIED - ANALYSIS IS I
>
 END ACTION   9

 ACTION
>10 EXECUTE SUB-MODULE STRESS
>
 END ACTION  10

 ACTION
>11 MESSAGE - 'SIZING IMPOSSIBLE - NO STRESSES'
>
 END ACTION  11

 ACTION
>12 EXECUTE SUB-MODULE SIZE
>
 END ACTION  12

 ACTION
>13 EXECUTE SUB-MODULE ROTATE
>      EXECUTE ROTATE
 END ACTION  13
```

Terminal Session. Action Code for DRIVEN (cont.)

```
 ACTION
>14 EXECUTE SUB-MODULE BASE
>
 END ACTION  14

 ACTION
>15 MESSAGE - 'ILLEGAL COMMAND'
>       WRITE(6,96) CMND
> 96    FORMAT(' ILLEGAL COMMAND ',A4)
>
 END ACTION  15

 ACTION
>

 MOD
>END

 INPUT COMPLETE

 COMMAND
>MOD


 SUB-MODULE:   DRIVEN

                IN-UNIT                   EXISTS
          RULE MATRIX                       YES
          ACTION MATRIX                     YES
          STRATEGY MATRIX                   YES
          CLUSTER MATRIX                    YES
          ACTION SOURCE CODE                YES
          PREDICATE SOURCE CODE             YES


 THERE ARE   2 STRATEGY ENTRIES SPANNING  16 RULES
 THERE ARE   2 CLUSTERS SPANNING  16 RULES
 THERE ARE  11 PREDICATES
 THERE ARE  16 RULES DERIVED FROM  11 PREDICATES
 THERE ARE  15 ACTIONS
 THE ACTION MATRIX COVERS  16 RULES AND  15 ACTIONS

 MOD
>CODE

 TYPE
>GLOB

 INPUT
>LOGICAL DBD(4)
>INTEGER CMND,TITLE(20)
>
 END GLOBAL INPUT
```

Terminal Session. Global Variables for DRIVEN

```
INPUT COMPLETE

COMMAND
>TRACE
TRACE

>C.2 1

CLUSTER:  2. INITIALISE
CLUSTER:  1. CHECK COMMAND

                                          *  ***************
                                          1     5    10   15
                                          V     V    V    V
PRED.
 1.                      CMND.EQ.END         Y$$$$$$$$$$$$$N
 2.                      CMND.EQ.NEW         $Y$$$$$$$$$$$$N
 3.                    CMND.EQ.BUILD         $$Y$$$$$$$$$$$N
 4.                   CMND.EQ.STRESS         $$$YYY$$$$$$$$N
 5.                     CMND.EQ.SIZE         $$$$$$YY$$$$$$N
 6.                   CMND.EQ.ROTATE         $$$$$$$$YYY$$$N
 7.                     CMND.EQ.BASE         $$$$$$$$$$$YYYN
 8.                           DBD(1)            N Y  N YN Y
 9.                           DBD(2)             NY   NY NY
10.                           DBD(3)               NY

                                          A     A    A    A
                                          *  ***************

TRACE
>END

COMMAND
>END
```

Terminal Session. DRIVEN Cluster Display

```
 COMMAND
>MOD

 SUB-MODULE:   DRIVEN

                IN-UNIT                  EXISTS
           RULE MATRIX                    YES
           ACTION MATRIX                  YES
           STRATEGY MATRIX                YES
           CLUSTER MATRIX                 YES
           ACTION SOURCE CODE             YES
           PREDICATE SOURCE CODE          YES


 THERE ARE   2 STRATEGY ENTRIES SPANNING  16 RULES
 THERE ARE   2 CLUSTERS SPANNING  16 RULES
 THERE ARE  11 PREDICATES
 THERE ARE  16 RULES DERIVED FROM  11 PREDICATES
 THERE ARE  15 ACTIONS
 THE ACTION MATRIX COVERS  16 RULES AND  15 ACTIONS

 MOD
>CODE

 TYPE
>GENER
 ***WARNING*** THERE ARE NO LOCAL VARIABLE DECLARATIONS FOR

 WARNING MESSAGES ISSUED - HIT C/R TO GENERATE, OR TYPE '99'
>

 GENERATION COMPLETE

 AT EXECUTION TIME, TABLES FOR THE FOLLOWING SUB-MODULES MUS
 WITH THE FORTRAN UNIT NUMBER SHOWN

          TABLE         UNIT

         DRIVEN            1
         STRESS           11
         SIZE             12
         BASE             13
         GETDAT           14
         BLDMOD           15
         ROTATE           16

 MOD
>END

 INPUT COMPLETE

 COMMAND
>END
```

Terminal Session. Code Generation of DRIVEN

```
>EPA
NO GENERATE DATASET ASSIGNED - DO NOT USE THE GENER SUB-COM

                    ---------------------------
                    +                         +
                    +  PROJECT - TOWER        +
                    +                         +
                    ---------------------------

 COMMAND
>NEW
 NEW SUB-MODULE NAME?
>ROTATE

 MOD
>CLUS

 HOW MANY CLUS ENTRIES?  (MAX. 32 ALPHANUMERICS EACH)
 ?
>5

 ENTRY?
>INITIALISE
 RULES?
 ?
>1

 ENTRY?
>INITIAL H
 RULES?
 ?
>4

 ENTRY?
>CABLE 1 STRETCH
 RULES?
 ?
>5

 ENTRY?
>CABLE 2 RELAXATION
 RULES?
 ?
>1

 ENTRY?
```

Terminal Session. New Sub-module ROTATE. Clusters

```
>ROTATION
 RULES?
 ?
>3

 CLUSTER TABLE                                      (RULES   1- 14)
                                                    01234567890123456789012
                                                    V         V         V
  1  INITIALISE                                      10000000000000
  2  INITIAL H                                       01111000000000
  3  CABLE 1 STRETCH                                 00000111110000
  4  CABLE 2 RELAXATION                              00000000001000
  5  ROTATION                                        00000000000111
                                                    0         1         2

 MOD
>STRAT

 HOW MANY STRA ENTRIES?  (MAX. 32 ALPHANUMERICS EACH)
 ?
>5

 ENTRY?
>INITIALISE

 ENTRY?
>INITIAL H

 ENTRY?
>CABLE 1 STRETCH

 ENTRY?
>CABLE 2 RELAXATION

 ENTRY?
>ROTATION

 STRATEGY MATRIX ENTRIES IN THE FORM:
          RULE NO. STRA NO.,CONTROL NO.     (0,0,0 ENTRY TERM
 ?
>14 5 0
 ?
>13 5 0
 ?
>12 5 3
 ?
>11 4 3
 ?
```

Terminal Session. Clusters/Strategy for ROTATE

```
> 10 4 5
 ?
> 10 3 4
 ?
> 9 4 0
 ?
> 9 3 0
 ?
> 8 4 4
 ?
> 8 3 3
 ?
> 7 3 3
 ?
> 6 4 4
 ?
> 6 3 3
 ?
> 5 2 3
 ?
> 4 2 0
 ?
> 3 2 2
 ?
> 2 2 2
 ?
> 1 1 2
 ?
> 0 0 0

 IDENTIFY START OF EXECUTION LOCATION AS RULE NO. , STRATEGY
 ?
> 1 1
 STRATEGY TABLE                          (RULES  1- 14)
                                         01234567890123456789012
                                         V         V         V
  1  INITIALISE                           2*************
  2  INITIAL H                            *2203*********
  3  CABLE 1 STRETCH                      *****33304****
  4  CABLE 2 RELAXATION                   *****4*4053***
  5  ROTATION                             ***********300
                                         0         1         2

 MOD
> END

 INPUT COMPLETE

 COMMAND
> END
```

Terminal Session. Strategy for ROTATE

```
------------------------
+                      +
+  PROJECT - TOWER     +
+                      +
------------------------
```

```
COMMAND
>MOD


SUB-MODULE:   ROTATE

            IN-UNIT                     EXISTS
       RULE MATRIX                        NO
       ACTION MATRIX                      NO
       STRATEGY MATRIX                    YES
       CLUSTER MATRIX                     YES
       ACTION SOURCE CODE                 NO
       PREDICATE SOURCE CODE              NO


THERE ARE   5 STRATEGY ENTRIES SPANNING  14 RULES
THERE ARE   5 CLUSTERS SPANNING  14 RULES

MOD
>RMTX.

EACH PREDICATE CODED EVALUATES TO 'Y' OR 'N'.  HOWEVER, RULI
     FOR GROUPS OF DEPENDENT CONDITIONS
FROM HOW MANY PREDICATES ARE THE RULES DERIVED?
?
>9

INTO HOW MANY INDEPENDENT FIELDS ARE THE PREDICATES GROUPED'
?
>9
CLUSTER NAME OR END
>INITIALISE

ENTER  1 RULES.
AAAAAAAAA
>

CLUSTER COMPLETE
```

Terminal Session. Rules for ROTATE

```
  CLUSTER NAME OR END
> INITIAL H

  ENTER  4 RULES.
  AAAAAAAAA
> Y
>  NY
>   N
>  Y

  CLUSTER COMPLETE

  CLUSTER NAME OR END
> CABLE 1 STRETCH

  ENTER  5 RULES.
  AAAAAAAAA
>    Y
>      N
>        NY
>         N
>        Y

  CLUSTER COMPLETE

  CLUSTER NAME OR END
> CABLE 2 RELAXATION

  ENTER  1 RULES.
  AAAAAAAAA
>      N

  CLUSTER COMPLETE

  CLUSTER NAME OR END
> ROTATION

  ENTER  3 RULES.
  AAAAAAAAA
>         YN
>         N
>          Y

  CLUSTER COMPLETE

  CLUSTER NAME OR END
> END
```

Terminal Session. Rules for ROTATE

```
> MOD

  SUB-MODULE:   ROTATE

                IN-UNIT                       EXISTS
          RULE MATRIX                          YES
          ACTION MATRIX                        NO
          STRATEGY MATRIX                      YES
          CLUSTER MATRIX                       YES
          ACTION SOURCE CODE                   NO
          PREDICATE SOURCE CODE                NO

  THERE ARE   5 STRATEGY ENTRIES SPANNING   14 RULES
  THERE ARE   5 CLUSTERS SPANNING   14 RULES
  THERE ARE  14 RULES DERIVED FROM    9 PREDICATES

  MOD
> AMTX

  CLUSTERS STORED AT PRESENT ARE:
   1    INITIALISE
   2    INITIAL H
   3    CABLE 1 STRETCH
   4    CABLE 2 RELAXATION
   5    ROTATION

  CLUSTER NAME OR END
> INITIALISE

  ENTER  1 LISTS OF ACTION NUMBERS
> 1 2 3 25

  CLUSTER COMPLETE

  CLUSTER NAME OR END
> INITIAL H

  ENTER  4 LISTS OF ACTION NUMBERS
> 4 5 6
> 7 8 5 6
> 9
> 10 11 3 15 13 16

  CLUSTER COMPLETE
```

† Terminal Session. Action Table for ROTATE

```
 CLUSTER NAME OR  END
> CABLE 1 STRETCH

 ENTER  6 LISTS OF ACTION NUMBERS
> 9
> 18  19 20
> 21  8
> 22  14 20
> 24
> 17  12 13  27 15

 CLUSTER COMPLETE

 CLUSTER NAME OR  END
> CABLE 2 RELAXATION

 ENTER  1 LISTS OF ACTION NUMBERS
> 21  16 11  8

 CLUSTER COMPLETE

 CLUSTER NAME OR  END
> ROTATION

 ENTER  3 LISTS OF ACTION NUMBERS
> 26  28 16  13 11  18 19  20
> 29
> 30

 CLUSTER COMPLETE

 CLUSTER NAME OR  END
> END

 MOD
> END

 INPUT COMPLETE

 COMMAND
```

Terminal Session. Action Table for ROTATE.

```
  MOD
> CODE

  TYPE
> PRED

  INPUT
> 9  ABS(DIF-F).LE..01
> 1  HZERO.EQ.0
> 2  ABS(T-TI).LE..01
> 3  NCYCS.LE.10
> 4  DELH.EQ.0
> 5  KFAC.LT..9
> 6  ABS(DELHNU-OLDELH).LE..005
> 7  CABCYC.LE.10
> 8  DEFCYC.LE.10
>

  MOD
> END

  COMMAND
> MOD

  SUB-MODULE:   ROTATE

                   IN-UNIT                 EXISTS
           RULE MATRIX                      YES
           ACTION MATRIX                    YES
           STRATEGY MATRIX                  YES
           CLUSTER MATRIX                   YES
           ACTION SOURCE CODE               NO
           PREDICATE SOURCE CODE            YES

  THERE ARE   5 STRATEGY ENTRIES SPANNING  14 RULES
  THERE ARE   5 CLUSTERS SPANNING  14 RULES
  THERE ARE   9 PREDICATES
  THERE ARE  14 RULES DERIVED FROM   9 PREDICATES
  THE ACTION MATRIX COVERS  14 RULES AND  30 ACTIONS

  MOD
> CODE

  TYPE
> ACTS

  ACTION
> 1 GET DATA FROM DATABASE
>
  END ACTION   1
```

Terminal Session. Predicates/Action Headers for ROTATE

```
 ACTION
> 22 NEW DH  (DELHNU)  AND DELH
>
 END ACTION  22

 ACTION
> 24 "CABLE CALCS. NOT CONVERGED"
>
 END ACTION  24

 ACTION
> 25 ZERO CYCLE NUMBER DEFCYC
>
 END ACTION  25

 ACTION
> 26 INCREMENT DEFCYC
>
 END ACTION  26

 ACTION
> 27 COMPUTE DIFFERENCE OF CABLE FORCES   DIF
>
 END ACTION  27

 ACTION
> 28 ADJUST DELL ON BASIS OF DIF
>
 END ACTION  28

 ACTION
> 29 "DEFLECTIOND_S NOT CONVERGED"
>
 END ACTION  29

 ACTION
> 30 COMPUTE AND PRINT ROTATION
>
 END ACTION  30

 ACTION
>

 MOD
>END

 INPUT COMPLETE

 COMMAND
>END
```

Terminal Session. Action Headers (end) for ROTA

```
>EPA

 NO GENERATE DATASET ASSIGNED - DO NOT USE THE GENER SUB-COM

                        ------------------------
                        +                      +
                        +  PROJECT - TOWER     +
                        +                      +
                        ------------------------


 COMMAND
>TRACE

 TRACE
>C 1 2 3 4 5


 CLUSTER:  1. INITIALISE
 CLUSTER:  2. INITIAL H
 CLUSTER:  3. CABLE 1 STRETCH
 CLUSTER:  4. CABLE 2 RELAXATION
 CLUSTER:  5. ROTATION
                                  *  ****  ******  ******  ***
                                  1     5     10     15     20
                                  V     V      V      V      V
 PRED.
  1.                HZERO.EQ.O.      Y
  2.        ABS(T-TI).LE..01          N Y
  3.                NCYCS.LE.10       YN   N       N
  4.                DELH.EQ.0               Y       Y
  5.                KFAC.LT..9               N          N
  6. ABS(DELHNU-OLDELH).LE..005               N Y    N Y
  7.            CABCYC.LE.10                  YN     YN
  8.            DFFCYC.LE.10                               YN
  9.        ABS(DIF-F).LE..01                              N Y

                                  A     A      A      A      A
                                  *  ****  ******  ******  ***
```

Terminal Session. Testing ROTATE

```
 TRACE
>1


  1. EXECUTE      GETDAT
  2.  ZERO INITIAL H (HZERO)
  3.  ZERO CYCLE NUMBER NCYCS
 26.  ZERO CYCLE NUMBER DEFCYC
 TRACE
>C 2


 CLUSTER:  2. INITIAL H
                                                                 ****
                                                                 1
                                                                 V
 PRED.
  1.                                              HZERO.EQ.0     Y
  2.                                        ABS(T-TI).LE..01      N Y
  3.                                              NCYCS.LE.10     YN

                                                                 A
                                                                 ****
 TRACE
>1


  4.  CALCULATE HZERO
  5.  CALCULATE RZERO
  6.  RECALCULATE CABLE TENSION
 TRACE
>2


  7.  ADJUST HZERO
  8.  INCREMENT NCYCS
  5.  CALCULATE RZERO
  6.  RECALCULATE CABLE TENSION
 TRACE
```

Terminal Session. Testing ROTATE

```
> 4

10.  INITIALISE DELL AND F
11.  1ST CABLE  (CABLNO=1)
 3.  ZERO CYCLE NUMBER NCYCS
15.  ZERO DELH
13.  ZERO CYCLE NUMBER CABCYC
16.  STRESS INCREASING  (FAC=+1.)
TRACE
> C 3


CLUSTER:  3. CABLE 1 STRETCH
                                                            ******
                                                            1   5
                                                            V   V
PRED.
 3.                                      NCYCS.LE.10        N
 4.                                      DELH.EQ.0           Y
 5.                                      KFAC.LT..9           N
 6.                    ABS(DELHNU-OLDELH).LE..005              N Y
 7.                                  CABCYC.LE.10              YN

                                                            A   A
                                                            ******

TRACE
> 2


18.  INITIALISE DELH AND OLDELH
19.  STRESS CABLE     EL,LBAR,H
20.  COMPUTE CABLE FACTORS KFAC,KPRIM
TRACE
> 3


21.  HALVE DELL  (KFAC TOO BIG)
 8.  INCREMENT NCYCS
TRACE
> 4


22.  NEW DH  (DELHNU) AND DELH
14.  INCREMENT CABCYC
20.  COMPUTE CABLE FACTORS KFAC,KPRIM
TRACE
```

CABLE FACTORS NOT RECALCULATED — STUCK IN A LOOP.

19. Stress cable
20. Compute cable factors

Terminal Session. Testing ROTATE

```
> 6

 17.  RECORD DELH,H FOR CABLE 1,  FAC=-1.
 12.  2ND CABLE  (CABLNO=2)
 13.  ZERO CYCLE NUMBER CABCYC
 27.  COMPUTE DIFFERENCE OF CABLE FORCES   DIF
 15.  ZERO DELH
 TRACE
>C 4


 CLUSTER:  4. CABLE 2 RELAXATION
                                                                ******
                                                                1   5
                                                                V   V

 PRED.
  3.                                          NCYCS.LE.10   N
  4.                                          DELH.EQ.0      Y
  5.                                          KFAC.LT..9      Y YN
  6.                             ABS(DELHNU-OLDELH).LE..005    N Y
  7.                                      CABCYC.LE.10         YN

                                                                A   A
                                                                ******

 TRACE
> 2


 18.  INITIALISE DELH AND OLDELH
 19.  STRESS CABLE     EL,LBAR,H
 20.  COMPUTE CABLE FACTORS KFAC,KPRIM
 TRACE
> 3


 22.  NEW DH  (DELHNU)  AND DELH
 14.  INCREMENT CABCYC
 20.  COMPUTE CABLE FACTORS KFAC,KPRIM
 TRACE
```

RULE 6 CANNOT BE REACHED

Fix thus:

Terminal Session. Testing ROTATE

```
> 5

 17.  RECORD DELH,H FOR CABLE 1,  FAC=-1.
 12.  2ND CABLE  (CABNO=2)
 13.  ZERO CYCLE NUMBER CABCYC
 27.  COMPUTE DIFFERENCE OF CABLE FORCES   DIF
 15.  ZERO DELH
 TRACE
> C 5


 CLUSTER:  5.  ROTATION
                                                      ***
                                                      1
                                                      V
 PRED.
  8.                                 DEFCYC.LE.10     YN
  9.                              ABS(DIF-F).LE..01   N Y


                                                      A
                                                      ***
 TRACE
> 3


 30.  COMPUTE AND PRINT ROTATION
 TRACE
> END


 COMMAND
> END
```

Terminal Session. Testing ROTATE

```
SUB-MODULE:   ROTATE

              IN-UNIT                    EXISTS
         RULE MATRIX                       YES
         ACTION MATRIX                     YES
         STRATEGY MATRIX                   YES
         CLUSTER MATRIX                    YES
         ACTION SOURCE CODE                YES
         PREDICATE SOURCE CODE             YES

 THERE ARE    5 STRATEGY ENTRIES SPANNING  14 RULES
 THERE ARE    5 CLUSTERS SPANNING  14 RULES
 THERE ARE    9 PREDICATES
 THERE ARE   14 RULES DERIVED FROM    9 PREDICATES
 THERE ARE   30 ACTIONS
 THE ACTION MATRIX COVERS  14 RULES AND   30 ACTIONS

 MOD
>CODE

 TYPE
>ACTS

 ACTION
>1
 C ACTION   1 GET DATA FROM DATABASE

 INPUT
>      EXECUTE GETDAT
 END ACTION    1

 ACTION
>2
 C ACTION   2 ZERO INITIAL H (HZERO)

 INPUT
>      HZERO=0
>
 END ACTION    2

 ACTION
>3
 C ACTION   3 ZERO CYCLE NUMBER NCYCS

 INPUT
>      NCYCS=0
>
 END ACTION    3
```

Terminal Session. Addition of Action Code for ROTATE

```
 ------------------------------
 +                            +
 +   PROJECT - TOWER          +
 +                            +
 ------------------------------

COMMAND
> MOD

SUB-MODULE:   ROTATE

              IN-UNIT                     EXISTS
         RULE MATRIX                       YES
         ACTION MATRIX                     YES
         STRATEGY MATRIX                   YES
         CLUSTER MATRIX                    YES
         ACTION SOURCE CODE                YES
         PREDICATE SOURCE CODE             YES

THERE ARE   5 STRATEGY ENTRIES SPANNING  14 RULES
THERE ARE   5 CLUSTERS SPANNING  14 RULES
THERE ARE   9 PREDICATES
THERE ARE  14 RULES DERIVED FROM   9 PREDICATES
THERE ARE  30 ACTIONS
THE ACTION MATRIX COVERS  14 RULES AND  30 ACTIONS

MOD
> CODE

TYPE
> ACTS

ACTION
> 31 INITIALISE STARTING DELH VALUES STRTDH
>       STRTDH(1(_)=1.
>       STRTDH(2)=1.
>
END ACTION  31

ACTION
>

MOD
```

Terminal Session. Addition of Action 31

```
> MOD

 SUB-MODULE:   ROTATE

               IN-UNIT                       EXISTS
          RULE MATRIX                          YES
          ACTION MATRIX                        YES
          STRATEGY MATRIX                      YES
          CLUSTER MATRIX                       YES
          ACTION SOURCE CODE                   YES
          PREDICATE SOURCE CODE                YES

 THERE ARE    5 STRATEGY ENTRIES SPANNING  14 RULES
 THERE ARE    5 CLUSTERS SPANNING  14 RULES
 THERE ARE    9 PREDICATES
 THERE ARE   14 RULES DERIVED FROM    9 PREDICATES
 THERE ARE   31 ACTIONS
 THE ACTION MATRIX COVERS  14 RULES AND  31 ACTIONS

 MOD
>CODE

 TYPE
>GENER
 ***WARNING*** THERE ARE NO LOCAL VARIABLE DECLARATIONS FOR

 WARNING MESSAGES ISSUED - HIT C/R TO GENERATE, OR TYPE '99'
>

 GENERATION COMPLETE

 AT EXECUTION TIME, TABLES FOR THE FOLLOWING SUB-MODULES MUS
 WITH THE FORTRAN UNIT NUMBER SHOWN

          TABLE          UNIT

         DRIVEN            1    MOD
         STRESS           11
         SIZE             12
         BASE             13
         GETDAT           14
         BLDMOD           15
         ROTATE           16

 TABLES MARKED "MOD" REQUIRE SOURCE (RE-)GENERATION TO MAINT

 MOD
>END
```

Terminal Session. Code Generation for ROTATE

```
> ROTATE

  SOUTH INDIAN LAKE  -  405 FT.  MICROWAVE TOWER

  CABLE DATA

      WEIGHT/FOOT            0.002 KIPS
      VERTICAL LENGTH        385.0 FEET
      PLAN LENGTH            300.0 FEET
      A X E                 10970. KIPS
      INITIAL TENSION        15.25 KIPS

  TOWER DATA

      A X E FOR 1 LEG     205000. KIPS
      CABLE LEVER ARM        9.44 FEET

  APPLIED TORQUE RESOLVED AT OUTRIGGER 13.90 KIP-FT

  OUTPUT

      INITIAL HORIZONTAL COMPONENT OF CABLE TENSION  9.139 KI
      ROTATION AT CABLE LEVEL  0.235 DEGREES
      LINEAR MOVEMENT OF CABLE SUPPORT  0.039 FEET
      FINAL CABLE FORCES    9.423 AND   8.855 KIPS

  COMMAND
> END
```

Terminal Session. Output from Sub-module ROTATE.

```
RELEASE 2.0                DRIVEN                DATE = 80141           16/28/46
          SUBROUTINE DRIVEN(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
         + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,
         + MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
    C SYSTEM SPECIFICATIONS ***************************************************
          INTEGER TABSIZ,CURULE,ACTIVE,BLANK/' '/
          INTEGER PDTABL(TABSIZ),STGY(MAXRLS,MAXSTR),ACTNOS(MAXACT,MAXRLS),
         + ATRANS(MACTS),AINTNO(MACTIG,2)
          LOGICAL*1 TABL(8),RULES(MAXPRD,MAXRLS),MASKS(MAXPRD,MAXRLS),
         + VECTOR(MAXPRD),INIT/.TRUE./,TRUE/.TRUE./
          LOGICAL MATCH
    C APPLICATION SPECIFICATIONS **********************************************
    C --- GLOBAL DECLARATIONS ---
          LOGICAL    DBD
          INTEGER    CMND,TITLE
          COMMON/I4/ CMND,TITLE(20)
          COMMON/L4/ DBD(4)
    C --- LOCAL DECLARATIONS ----
    C END OF SPECIFICATIONS ***************************************************
          IF(.NOT.INIT)CALL $DRIVS  (VECTOR,NRPRDS)
          INIT=.FALSE.
          CURULE=0
          DO 99999 I99999=1,NRULES
            IF(STGY(I99999,ACTIVE).EQ.BLANK) GO TO 99999
            IF(.NOT.MATCH(RULES(1,I99999),MASKS(1,I99999),VECTOR,NRPRDS))
         +        GO TO 99999
            CURULE=I99999
            IF(STGY(CURULE,ACTIVE).EQ.0) INIT=.TRUE.
            DO 99000 I99000=1,MAXACT
              NEXTAC=ACTNOS(I99000,CURULE)
              IF(NEXTAC.EQ.0) RETURN
              NXTACT=ATRANS(NEXTAC)
              DO 99998 I99998=1,MAXACT
                IF(NXTACT.EQ.0) GO TO 99000
                NACSHN=AINTNO(NXTACT,1)
```

Project TOWER. Sub-module DRIVEN. Subroutine DRIVEN

```
      GO TO
     +(99997,99996,99995,99994,99993,99992,99991,99990,99989,99988
     +,99987,99986,99985,99984,99983,99982,99981,99980,99979,99978
     +,99977,99976,99975,99974,99973,99972
     +),NACSHN
C ACTION ENTRIES START HERE ****************************************************
C ACTION   1 READ A COMMAND
C
99997 CONTINUE
      WRITE(6,100)
 100  FORMAT(' COMMAND')
      READ(9,99) CMND
 99   FORMAT(20A4)
      GO TO 99998
C ACTION   2 GET DATABASE DESCRIPTORS DBD(1)-DBD(4)
C
99996 CONTINUE
      CALL GETDBD(DBD)
      GO TO 99998
C ACTION   3 ZERO DATABASE DESCRIPTORS
C
99995 CONTINUE
      GO TO 99998
C ACTION   4 READ A PROJECT TITLE
C
99994 CONTINUE
      GO TO 99998
C ACTION   5 CLEAR DATABASE (NEW PROJECT)
C
99993 CONTINUE
      GO TO 99998
C ACTION   6 NO-OP
C
99992 CONTINUE
```

Project TOWER. Sub-module DRIVEN. Subroutine DRIVEN

```
RELEASE 2.0            DRIVEN            DATE = 80141          16/28/46
         GO TO 99998
C ACTION    7EXECUTE      BLDMOD
C
99991 CONTINUE
         GO TO 99998
C EXECUTE ACTION 1      STACK CURRENT PDT
C
99990 CONTINUE
         WRITE(30) PDTABL
         GO TO 99998
C EXECUTE ACTION 2    -    STACK STATUS VARIABLES
C
99989 CONTINUE
         WRITE(30) (VECTOR(I),I=1,NRPRDS),ACTIVE,CURULE,NXTROW
         GO TO 99998
C EXECUTE ACTION 3    -  IDENTIFY TABLE TO BE CALLED    (NTF=    ).
C
99988 CONTINUE
         NTF=15
         GO TO 99998
C EXECUTE ACTION 4    -    READ NEW PDT
C
99987 CONTINUE
         READ(NTF'1) PDTABL
         GO TO 99998
C EXECUTE ACTIONS 5 & 9    -    UPDATE COUNTER VARIABLES
C
99986 CONTINUE
         NSTRTS=PDTABL(3)
         NRPRDS=PDTABL(5)
         NACTS=PDTABL(6)
         NSTROW=PDTABL(7)
         NSTCOL=PDTABL(8)
         NSTRLS=PDTABL(9)
```

Project TOWER. Sub-module DRIVEN. Subroutine DRIVEN

```
      NRULES=PDTABL(11)
      NARLS=PDTABL(12)
      MACACT=PDTABL(14)
      NACTG=PDTABL(18)
      NTF=PDTABL(71)
      GO TO 99998
C EXECUTE ACTIONS 6 & 11   -   FETCH NEW EXECUTION TABLES.
C
99985 CONTINUE
      II=PDTABL(21)-PDTABL(21)/256*256-29
      READ(NTF'II) ((RULES(I,J),I=1,NRPRDS),J=1,NRULES),
     +             ((MASKS(I,J),I=1,NRPRDS),J=1,NRULES)
      II=PDTABL(22)-PDTABL(22)/256*256-29
      READ(NTF'II) ((ACTNOS(I,J),I=1,MACACT),J=1,NARLS)
      II=PDTABL(23)-PDTABL(23)/256*256-29
      READ(NTF'II) ((STGY(I,J),I=1,NSTRLS),J=1,NSTRTS)
      II=PDTABL(25)-PDTABL(25)/256*256-29
      READ(NTF'II) (ATRANS(I),I=1,NACTS),((AINTNO(I,J),I=1,NACTG),
     +  J=1,2)
      GO TO 99998
C EXECUTE ACTION 7   -   MINI-DRIVER FOR CALLED TABLE
C
99984 CONTINUE
      ACTIVE=NSTROW
      DO 99971 I99971=1,NRPRDS
99971  VECTOR(I99971)=.NOT.(RULES(I99971,NSTCOL).AND.TRUE)
      N99970=1
      DO  99970 I99970=1,N99970
       N99970=N99970+1
       CALL BLDMOD    (VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
     + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,
     + ATRANS,MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
       IF(CURULE.EQ.0) GO TO    99969
       NXTROW=STGY(CURULE,ACTIVE)
```

```
      IF(NXTROW.EQ.0) GO TO   99969
99970 ACTIVE=NXTROW
99969 CONTINUE
      GO TO 99998
C EXECUTE ACTION 8    -   UNSTACK PDT
C
99983 CONTINUE
      BACKSPACE 30
      BACKSPACE 30
      READ(30) PDTABL
      GO TO 99998
C EXECUTE ACTION 10    -   UNSTACK STATUS VARIABLES
C
99982 CONTINUE
      READ(30) (VECTOR(I),I=1,NKPRDS),ACTIVE,CURULE,NXTROW
      BACKSPACE 30
      BACKSPACE 30
      GO TO 99998
C ACTION   8 MESSAGE - "NO ANALYSIS - INCOMPLETE MODEL"
C
99981 CONTINUE
      WRITE(6,98)
 98   FORMAT(' NUMERICAL MODEL OF TOWER IS INCOMPLETE - REQUESTED ANALYS
     +IS IS IMPOSSIBLE')
      GO TO 99998
C ACTION   9 MESSAGE - "NO ANALYSIS - NO LOADS"
C
99980 CONTINUE
      WRITE(6,97)
 97   FORMAT(' NO LOADS HAVE BEEN SPECIFIED - ANALYSIS IS IMPOSSIBLE')
      GO TO 99998
C ACTION  10 EXECUTE SUB-MODULE STRESS
C
99979 CONTINUE
```

```
RELEASE 2.0                DRIVEN              DATE = 80141          16/28/46
      GO TO 99998
C ACTION  11 MESSAGE - "SIZING IMPOSSIBLE - NO STRESSES"
C
99978 CONTINUE
      GO TO 99998
C ACTION  12 EXECUTE SUB-MODULE SIZE
C
99977 CONTINUE
      GO TO 99998
C ACTION  13EXECUTE     ROTATE
C
99976 CONTINUE
      GO TO 99998
C EXECUTE ACTION 3   -   IDENTIFY TABLE TO BE CALLED   (NTF=   )
C
99975 CONTINUE
      NTF=16
      GO TO 99998
C EXECUTE ACTION 7   -   MINI-DRIVER FOR CALLED TABLE
C
99974 CONTINUE
      ACTIVE=NSTROW
      DO 99968 I99968=1,NRPRDS
99968  VECTOR(I99968)=.NOT.(RULES(I99968,NSTCOL).AND.TRUE)
      N99967=1
      DO  99967 I99967=1,N99967
       N99967=N99967+1
       CALL ROTATE    (VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
     + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,
     + ATRANS,MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
       IF(CURULE.EQ.0) GO TO   99966
       NXTROW=STGY(CURULE,ACTIVE)
       IF(NXTROW.EQ.0) GO TO   99966
99967  ACTIVE=NXTROW
```

```
RELEASE 2.0                DRIVEN                DATE = 80141           16/28/46

  99966 CONTINUE
        GO TO 99998
  C ACTION  14 EXECUTE SUB-MODULE BASE
  C
  99973 CONTINUE
        GO TO 99998
  C ACTION  15 MESSAGE - "ILLEGAL COMMAND"
  C
  99972 CONTINUE
        WRITE(6,96) CMND
     96 FORMAT(' ILLEGAL COMMAND ',A4)
        GO TO 99998
  C END OF ACTIONS ****************************************************************
  99998 NXTACT=AINTNO(NXTACT,2)
  99000 CONTINUE
        RETURN
  99999 CONTINUE
        WRITE(6,90001) PDTABL(1),PDTABL(2),ACTIVE,(VECTOR(I),I=1,NRPRDS)
  90001 FORMAT(' VECTOR NOT MATCHED IN ',A4,A2,' STRATEGY ',I2/' VECTOR WA
       +S ',50L1)
        RETURN
        END
```

Project TOWER. Sub-module DRIVEN. Subroutine DRIVEN

```
RELEASE 2.0              $DRIV$              DATE = 80141           16/28/46

      SUBROUTINE    $DRIV$  (V,NRPRDS)
C SYSTEM SPECIFICATIONS ***********************************************
      LOGICAL*1 V(NRPRDS)
C APPLICATION SPECIFICATIONS *******************************************
C --- GLOBAL DECLARATIONS ---
      LOGICAL     DBD
      INTEGER     CMND,TITLE
      COMMON/I4/ CMND,TITLE(20)
      COMMON/L4/ DBD(4)
C --- LOCAL DECLARATIONS ----
      INTEGER END,NEW,BUILD,STRESS,SIZE,ROTATE,BASE
      DATA END,NEW,BUILD,STRESS,SIZE,ROTATE,BASE/'END','NEW','BUIL',
     +   'STRE','SIZE','ROTA','BASE'/
      V( 1)= CMND.EQ.END
      V( 2)= CMND.EQ.NEW
      V( 3)= CMND.EQ.BUILD
      V( 4)= CMND.EQ.STRESS
      V( 5)= CMND.EQ.SIZE
      V( 6)= CMND.EQ.ROTATE
      V( 7)= CMND.EQ.BASE
      V( 8)= DBD(1)
      V( 9)= DBD(2)
      V(10)= DBD(3)
      V(11)= DBD(4)
      RETURN
      END
```

```
:ELEASE 2.0                ROTATE              DATE = 80147          12/17/16

      SUBROUTINE ROTATE(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
     + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,
     + MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
C SYSTEM SPECIFICATIONS ************************************************
      INTEGER TABSIZ,CURULE,ACTIVE,BLANK/' '/
      INTEGER PDTABL(TABSIZ),STGY(MAXRLS,MAXSTR),ACTNOS(MAXACT,MAXRLS),
     + ATRANS(MACTS),AINTNO(MACTIG,2)
      LOGICAL*1 TABL(8),RULES(MAXPRD,MAXRLS),MASKS(MAXPRD,MAXRLS),
     + VECTOR(MAXPRD),INIT/.TRUE./,TRUE/.TRUE./
      LOGICAL MATCH
C APPLICATION SPECIFICATIONS *******************************************
C --- GLOBAL DECLARATIONS ---
      REAL        DELHNU,A,DIF,F
      REAL        TOR,OLDELH,HZERO,LZERO,HT,TI,Q,DELH,AEC,AEL,T,KFAC
      INTEGER     ZI411,TITLE,NCYCS,CABCYC,DEFCYC
      COMMON/I4/ ZI411,TITLE(20),NCYCS,CABCYC,DEFCYC
      COMMON/R4/ TOR,OLDELH,HZERO,LZERO,HT,TI,Q,DELH,AEC,AEL,T,KFAC
      COMMON/R4/ DELHNU,A,DIF,F
C --- LOCAL DECLARATIONS ----
      INTEGER CABLNO
      REAL STRTDH(2),LBAR,KPRIM,NEWH
       ASHIN(X)=ALOG(X+SQRT(X*X+1.))
      PI=4.*ATAN(1.)
C END OF SPECIFICATIONS ************************************************
      IF(.NOT.INIT)CALL $ROTA$  (VECTOR,NRPRDS)
      INIT=.FALSE.
      CURULE=0
      DO 99999 I99999=1,NRULES
        IF(STGY(I99999,ACTIVE).EQ.BLANK) GO TO 99999
        IF(.NOT.MATCH(RULES(1,I99999),MASKS(1,I99999),VECTOR,NRPRDS))
     +        GO TO 99999
        CURULE=I99999
        IF(STGY(CURULE,ACTIVE).EQ.0) INIT=.TRUE.
        DO 99000 I99000=1,MAXACT
```

Project TOWER. Sub-module ROTATE. Subroutine ROTATE

```
      NEXTAC=ACTNOS(I99000,CURULE)
      IF(NEXTAC.EQ.0) RETURN
      NXTACT=ATRANS(NEXTAC)
      DO 99998 I99998=1,MAXACT
        IF(NXTACT.EQ.0) GO TO 99000
        NACSHN=AINTNO(NXTACT,1)
        GO TO
     +(99997,99996,99995,99994,99993,99992,99991,99990,99989,99988
     +,99987,99986,99985,99984,99983,99982,99981,99980,99979,99978
     +,99977,99976,99975,99974,99973,99972,99971,99970,99969,99968
     +,99967,99966,99965,99964,99963,99962,99961,99960,99959
     +),NACSHN
C ACTION ENTRIES START HERE ********************************************
C ACTION   1EXECUTE     GETDAT
C
99997 CONTINUE
      GO TO 99998
C EXECUTE ACTION 1    STACK CURRENT PDT
C
99968 CONTINUE
      WRITE(30) PDTABL
      GO TO 99998
C EXECUTE ACTION 2   -  STACK STATUS VARIABLES
C
99967 CONTINUE
      WRITE(30) (VECTOR(I),I=1,NRPROS),ACTIVE,CURULE,NXTROW
      GO TO 99998
C EXECUTE ACTION 3   -   IDENTIFY TABLE TO BE CALLED   (NTF=  )
C
99966 CONTINUE
      NTF=14
      GO TO 99998
C EXECUTE ACTION 4   -   READ NEW PDT
C
```

```
99965 CONTINUE
      READ(NTF'1) PDTABL
      GO TO 99998
C EXECUTE ACTIONS 5 & 9   -   UPDATE COUNTER VARIABLES
C
99964 CONTINUE
      NSTRTS=PDTABL(3)
      NRPRDS=PDTABL(5)
      NACTS=PDTABL(6)
      NSTROW=PDTABL(7)
      NSTCOL=PDTABL(8)
      NSTRLS=PDTABL(9)
      NRULES=PDTABL(11)
      NARLS=PDTABL(12)
      MACACT=PDTABL(14)
      NACTG=PDTABL(18)
      NTF=PDTABL(71)
      GO TO 99998
C EXECUTE ACTIONS 6 & 11   -   FETCH NEW EXECUTION TABLES
C
99963 CONTINUE
      II=PDTABL(21)-PDTABL(21)/256*256-29
      READ(NTF'II) ((RULES(I,J),I=1,NRPRDS),J=1,NRULES),
     +             ((MASKS(I,J),I=1,NRPRDS),J=1,NRULES)
      II=PDTABL(22)-PDTABL(22)/256*256-29
      READ(NTF'II) ((ACTNOS(I,J),I=1,MALACT),J=1,NARLS)
      II=PDTABL(23)-PDTABL(23)/256*256-29
      READ(NTF'II) ((STGY(I,J),I=1,NSTRLS),J=1,NSTRTS)
      II=PDTABL(25)-PDTABL(25)/256*256-29
      READ(NTF'II) (ATRANS(I),I=1,NACTS),((AINTNO(I,J),I=1,NACTG),
     +  J=1,2)
      GO TO 99998
C EXECUTE ACTION 7   -   MINI-DRIVER FOR CALLED TABLE
```

```
ELEASE 2.0 ,              ROTATE              DATE = 80147           12/17/16

 99962 CONTINUE
       ACTIVE=NSTROW
       DO 99958 I99958=1,NRPRDS
 99958  VECTOR(I99958)=.NOT.(RULES(I99958,NSTCOL).AND.TRUE)
       N99957=1
       DO  99957 I99957=1,N99957
        N99957=N99957+1
        CALL GETDAT    (VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
      + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,
      + ATRANS,MACTS,AININO,MACTIG,MAXSTR,ACTIVE)
        IF(CURULE.EQ.0) GO TO   99956
        NXTROW=STGY(CURULE,ACTIVE)
        IF(NXTROW.EQ.0) GO TO   99956
 99957  ACTIVE=NXTROW
 99956 CONTINUE
       GO TO 99998
 C EXECUTE ACTION 8   -   UNSTACK PDT
 C
 99961 CONTINUE
       BACKSPACE 30
       BACKSPACE 30
       READ(30) PDTABL
       GO TO 99998
 C EXECUTE ACTION 10   -   UNSTACK STATUS VARIABLES
 C
 99960 CONTINUE
       READ(30) (VECTOR(I),I=1,NRPRDS),ACTIVE,CURULE,NXTROW
       BACKSPACE 30
       BACKSPACE 30
       GO TO 99998
 C ACTION   2 ZERO INITIAL H (HZERO)
 C
 99996 CONTINUE
       HZERO=0
```

Project TOWER. Sub-module ROTATE. Subroutine ROTATE

ELEASE 2.0 ROTATE . DATE = 80147 12/17/16

```
      GO TO 99998
C ACTION    3 ZERO CYCLE NUMBER NCYCS
C
99995 CONTINUE
      NCYCS=0
      GO TO 99998
C ACTION    4 CALCULATE-HZERO
C
99994 CONTINUE
      HZERO=LZERO/SQRT(HT*HT+LZERO*LZERO)*TI
      GO TO 99998
C ACTION    5 CALCULATE RZERO
C
99993 CONTINUE
      RZERO=Q*LZERO/(2.*HZERO)
      Z=Q*HT/(2.*HZERO*SINH(RZERO))
      A1=ASHIN(Z)-RZERO
      GO TO 99998
C ACTION    6 RECALCULATE CABLE TENSION
C
99992 CONTINUE
      T=HZERO*COSH(Q/HZERO*LZERO+A1)
      GO TO 99998
C ACTION    7 ADJUST HZERO
C
99991 CONTINUE
      HZERO=TI/T*HZERO
      GO TO 99998
C ACTION    8 INCREMENT NCYCS
C
99990 CONTINUE
      NCYCS=NCYCS+1
      GO TO 99998
C ACTION    9 "CABLES TOO SLACK"
```

Project TOWER. Sub-module ROTATE. Subroutine ROTATE

```
ELEASE 2.0                 ROTATE                 DATE = 80147              12/17/16

C
99989 CONTINUE
      WRITE(6,99)
 99   FORMAT(' CABLES ARE TOO SLACK - CANNOT CONVERGE ON HZERO')
      GO TO 99998
C ACTION  10 INITIALISE DELL AND F
C
99988 CONTINUE
      DELL=.1
      F=TOR/(3.*A)
      WRITE(6,98) TITLE,Q,HT,LZERO,AEC,TI,AEL,A,TOR,HZERO
 98   FORMAT(/1X,20A4//' CABLE DATA'//5X,'WEIGHT/FOOT',T25,F6.3,' KIPS'/
     +5X,'VERTICAL LENGTH',T25,F6.1,' FEET'/5X,'PLAN LENGTH',T25,F6.1,
     +' FEET'/5X,'A X E',T25,F6.0,' KIPS'/5X,'INITIAL TENSION',T25,F6.2,
     +' KIPS'//' TOWER DATA'//5X,'A X E FOR 1 LEG',T24,F7.0,' KIPS'/
     +5X,'CABLE LEVER ARM',T25,F6.2,' FEET'//' APPLIED TORQUE RESOLVED '
     +,'AT OUTRIGGER',F6.2,' KIP-FT'//' OUTPUT'//5X,'INITIAL ',
     +'HORIZONTAL COMPONENT OF CABLE TENSION ',F6.3,' KIPS')
      GO TO 99998
C ACTION  11 1ST CABLE  (CABLNO=1)
C
99987 CONTINUE
      CABLNO=1
      GO TO 99998
C ACTION  12 2ND CABLE  (CABLNO=2)
C
99986 CONTINUE
      CABLNO=2
      GO TO 99998
C ACTION  13 ZERO CYCLE NUMBER CABCYC
C
99985 CONTINUE
      CABCYC=0
      GO TO 99998
```

Project TOWER. Sub-module ROTATE. Subroutine ROTATE

```
LEASE 2.0              ROTATE              DATE = 80147          12/17/16

C ACTION  14 INCREMENT CABCYC
C
99984 CONTINUE
      CABCYC=CABCYC+1
      GO TO 99998
C ACTION  15 ZERO DELH
C
99983 CONTINUE
      DELH=0.
      GO TO 99998
C ACTION  16 STRESS INCREASING  (FAC=+1.)
C
99982 CONTINUE
      FAC=1.
      GO TO 99998
C ACTION  17 RECORD DELH,H FOR CABLE 1,  FAC=-1.
C
99981 CONTINUE
      FAC=-1.
      STRTDH(1)=DELH*2./F
      H1=HZERO+DELHNU
      GO TO 99998
C ACTION  18 INITIALISE DELH AND OLDELH
C
99980 CONTINUE
      DELHNU=0.
      DELH=F/2.*STRTDH(CABLNO)
      OLDELH=DELH
      GO TO 99998
C ACTION  19 STRESS CABLE      EL,LBAR,H
C
99979 CONTINUE
      EL=LZERO+FAC*DELL
      LBAR=SQRT(HT*HT+EL*EL)
```

Project TOWER. Sub-module ROTATE. Subroutine ROTATE.

```
      H=HZERO+FAC*DELH
      GO TO 99998
C ACTION  20 COMPUTE CABLE FACTORS KFAC,KPRIM
C
99978 CONTINUE
      ELSQ=EL*EL
      R=G*EL/(2.*H)
      RI=(R+RZERO)/2.
      RISQ=RI*RI
      S=LBAR+ELSQ*R*R/(6.*LBAR)
      AK=(1.+.4*RISQ)*DELL*FAC
      SCUB=S**3
      BK=SCUB*(1.-RISQ/6.*(4.-ELSQ/(S*S)))*DELH*FAC
      CK=ELSQ*AEC*(1.+RZERO*RZERO/6.)*(1.+RISQ/10.)
      DK=HT**3*(1.-RISQ/3.)*DELH*FAC*2.
      EK=CK*AEL/AEC
      KFAC=6./(RZERO*RZERO*EL)*(AK-BK/CK-DK/EK)
      AKPRIM=(-.4*RI*R*DELL/H)*FAC
      BKPRIM=FAC*(SCUB*DELH*(R*RI/(6.*H)*(4.-ELSQ/(S*S))+
     +   (R*RI*ELSQ)**2/(9.*LBAR*SCUB*H))+((1.-RISQ/6.*
     +   (4.-ELSQ/(S*S)))*(SCUB-(R*EL*S)**2/(LBAR*H)*DELH)))
      CKPRIM=(-ELSQ*AEC*R*RI*(1.+RZERO*RZERO/6.)/(10.*H))
      EKPRIM=CKPRIM*AEL/AEC
      DKPRIM=HT**3*(1.-RISQ/3.+DELH*R*RI/(3.*H))*FAC*2.
      KPRIM=6./(RZERO*RZERO*EL)*(AKPRIM-(CK*BKPRIM-BK*CKPRIM)
     +   /CK**2-(EK*DKPRIM-DK*CKPRIM)/EK**2)
      GO TO 99998
C ACTION  21 HALVE DELL   (KFAC TOO BIG)
C
99977 CONTINUE
      DELL=DELL/2.
      GO TO 99998
C ACTION  22 NEW DH  (DELHNU)  AND DELH
C
```

```
:LEASE 2.0            ROTATE              DATE = 80147          12/17/16

99976 CONTINUE
      NEWH=HZERO/SQRT(1.-KFAC)
      DELHNU=(NEWH-HZERO)*FAC
      FPRIMH=HZERO*KPRIM/(2.*(1.-KFAC)**1.5)*FAC-1.
      FH=NEWH-H
      H=H-FH/FPRIMH
      DELH=FAC*(H-HZERO)
      OLDELH=DELH
      GO TO 99998
C ACTION  24 "CABLE CALCS. NOT CONVERGED"
C
99975 CONTINUE
      WRITE(6,97) CABLNO
 97   FORMAT(' CABLE ',I2,' NOT CONVERGED')
      GO TO 99998
C ACTION  25 ZERO CYCLE NUMBER DEFCYC
C
99974 CONTINUE
      DEFCYC=0
      GO TO 99998
C ACTION  26 INCREMENT DEFCYC
C
99973 CONTINUE
      DEFCYC=DEFCYC+1
      GO TO 99998
C ACTION  27 COMPUTE DIFFERENCE OF CABLE FORCES   DIF
C
99972 CONTINUE
      H2=HZERO-DELHNU
      STRTDH(2)=DELH*2./F
      DIF=(H1-H2)*COS(PI/6.)
      GO TO 99998
C ACTION  28 ADJUST DELL ON BASIS OF DIF
C
```

```
99971 CONTINUE
      DELL=DELL-(DIF-F)/DIF*DELL
      GO TO 99998
C ACTION  29 "DEFLECTIONS NOT CONVERGED"
C
99970 CONTINUE
      WRITE(6,96)
  96  FORMAT(' DEFLECTION NOT CONVERGED')
      GO TO 99998
C ACTION  30 COMPUTE AND PRINT ROTATION
C
99969 CONTINUE
      ROT=DELL/A*180./PI
      WRITE(6,95) ROT,DELL,H1,H2
 95   FORMAT(5X,'ROTATION AT CABLE LEVEL ',F6.3,' DEGREES'/
     +5X,'LINEAR MOVEMENT OF CABLE SUPPORT ',F6.3,' FEET'/5X,
     +'FINAL CABLE FORCES ',F7.3,' AND ',F7.3,' KIPS'//)
      GO TO 99998
C ACTION  31 INITIALISE STARTING DELH VALUES STRTDH
C
99959 CONTINUE
      STRTDH(1)=1.
      STRTDH(2)=1.
      GO TO 99998
C END OF ACTIONS ************************************************************
99998 NXTACT=AINTNO(NXTACT,2)
99000 CONTINUE
      RETURN
99999 CONTINUE
      WRITE(6,90001) PDTABL(1),PDTABL(2),ACTIVE,(VECTOR(I),I=1,NRRDS)
90001 FORMAT(' VECTOR NOT MATCHED IN ',A4,A2,' STRATEGY ',I2/' VECTOR WA
     +S ',50L1)
      RETURN
      END
```

Project TOWER. Sub-module ROTATE. Subroutine ROTATE

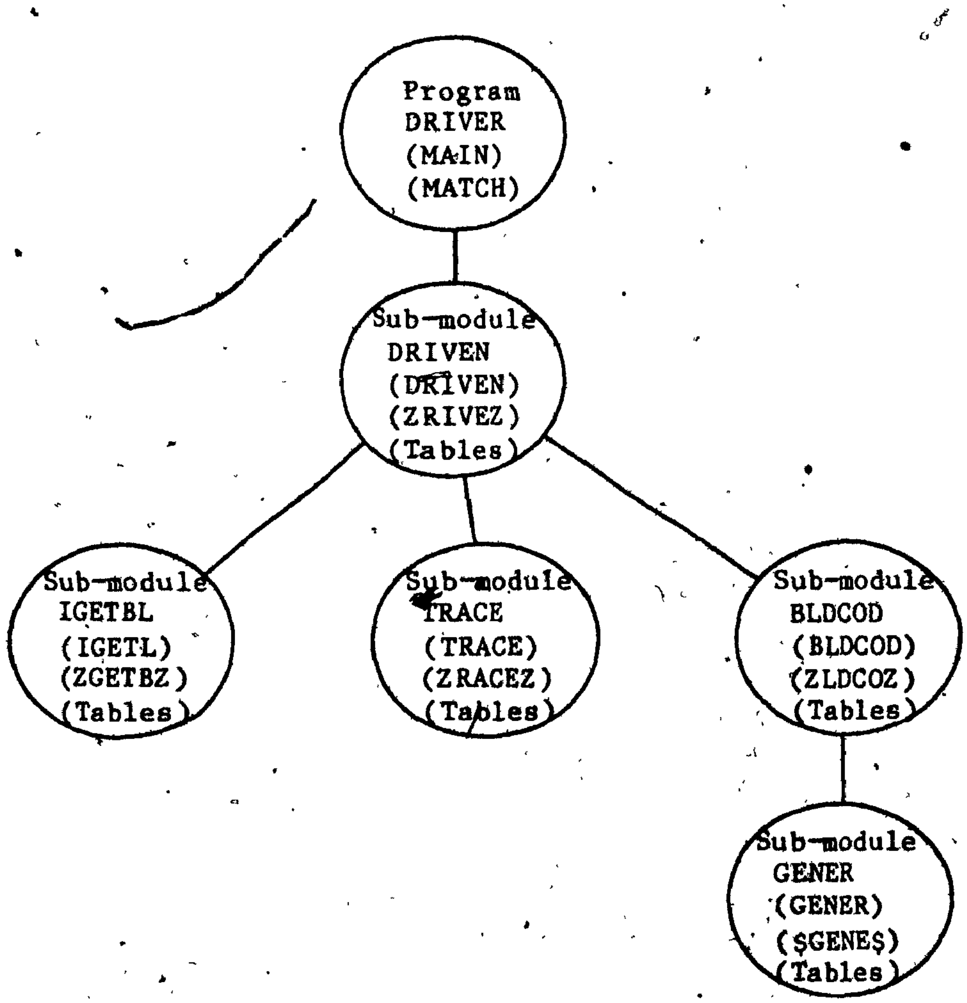
```
:LEASE 2.0              $ROTAS              DATE = 80147          12/17/16

      SUBROUTINE    $ROTAS  (V,NRPRDS)
C SYSTEM SPECIFICATIONS ***************************************************
      LOGICAL*1 V(NRPRDS)
C APPLICATION SPECIFICATIONS **********************************************
C --- GLOBAL DECLARATIONS ---
      REAL         DELHNU,A,DIF,F
      REAL         TOR,OLDELH,HZERO,LZERO,HT,TI,G,DELH,AEC,AEL,T,KFAC
      INTEGER      ZI411,TITLE,NCYCS,CABCYC,DEFCYC
      COMMON/I4/ ZI411,TITLE(20),NCYCS,CABCYC,DEFCYC
      COMMON/R4/ TOR,OLDELH,HZERO,LZERO,HT,TI,G,DELH,AEC,AEL,T,KFAC
      COMMON/R4/ DELHNU,A,DIF,F
C --- LOCAL DECLARATIONS ----
      V( 1)= HZERO.EQ.0
      V( 2)= ABS(T-TI).LE..01
      V( 3)= NCYCS.LE.10
      V( 4)= DELH.EQ.0
      V( 5)= KFAC.LT..9
      V( 6)= ABS(DELHNU-OLDELH).LE..005
      V( 7)= CABCYC.LE.10
      V( 8)= DEFCYC.LE.10
      V( 9)= ABS(DIF-F).LE..01
      RETURN
      END
```

Project TOWER. Sub-module ROTATE. Subroutine $ROTAS

APPENDIX B

SOURCE LISTING OF SYSTEM PROGRAMS

Fortran compiler listings of the main program and the various sub-modules' subroutines follow in the sequence implied by their execution hierarchy.



Printout of each sub-module's tables follows its source listing.

```
RELEASE 2.0                MAIN                DATE = 80124          14/25/50
      INTEGER PDTABL(375),STGY(120,15),ACTNOS(20,120),ACTIVE,TABSIZ,
     +  CURULE,ATRANS(130),AINTNO(151,2)
      LOGICAL*1 TABL(8),RULES(50,120),MASKS(50,120),VECTOR(50),TRUE
      EQUIVALENCE (PDTABL(19),TABL(1))
      DATA TRUE/.TRUE./
C FILE SPECIFICATIONS FOR USER'S EXECUTABLE SUB-MODULE TABLES
      DEFINE FILE 1(20,1500,L,II),12(20,1500,L,III2),11(20,1500,L,III1)
      DEFINE FILE 13(20,1500,L,III3),14(20,1500,L,III4)
C MAXRLS IS BREADTH OF ALL TABLES  -  STGY,ACTNOS,RULES
      MAXRLS=120
C MAXPRD IS THE LENGTH OF RULES, MASKS AND VECTOR (IE. MAX NO OF PREDS)
      MAXPRD=50
C TABSIZ IS THE SIZE OF THE PROGRAM DESCRIPTOR TABLE (PDTABL)
      TABSIZ=375
C MAXACT IS MAX POSSIBLE NO OF ACTIONS PER RULE (IE. LENGTH OF ACTNOS)
      MAXACT=20
C MACTIG IS MAX NO OF ACTIONS INPUT AND GENERATED
      MACTS=130
      MAXGEN=20
      MACTIG=MACTS+MAXGEN+1
C MAXSTR IS MAXIMUM NO OF STRATEGY ENTRIES
      MAXSTR=15
      CALL ERRSET(215,256,5,1)
      READ(1'1) PDTABL
      NSTROW=PDTABL(7)
      NSTCOL=PDTABL(8)
      NRULES=PDTABL(11)
      NRPROS=PDTABL(5)
      MACACT=PDTABL(14)
      NACTS=PDTABL(6)
      NARLS=PDTABL(12)
      NSTRTS=PDTABL(3)
      NSTRLS=PDTABL(9)
      NACTG=PDTABL(18)
```

Program DRIVER

```
RELEASE 2.0                MAIN                   DATE = 80124          14/25/50

C READ IN REQUIRED PARTS OF SUB-MODULE DRIVEN'S TABLES
      II=PDTABL(21)-(PDTABL(21)/256*256)-29
      READ(1'II) ((RULES(I,J),I=1,NRPRDS),J=1,NRULES)
     +,((MASKS(I,J),I=1,NRPRDS),J=1,NRULES)
      II=PDTABL(22)-(PDTABL(22)/256*256)-29
      READ(1'II) ((ACTNOS(I,J),I=1,MACACT),J=1,NARLS)
      II=PDTABL(23)-(PDTABL(23)/256*256)-29
      READ(1'II) ((STGY(I,J),I=1,NSTRLS),J=1,NSTRTS)
      II=PDTABL(25)-(PDTABL(25)/256*256)-29
      READ(1'II) (ATRANS(I),I=1,NACTS),((AINTNO(I,J),I=1,NACTG),J=1,2)
C PRIME THE STATUS VECTOR FOR THE FIRST RULE TO BE EXECUTED
      ACTIVE=NSTROW
      DO 10 I=1,NRPRDS
C NOTE...THE SYMBOLS USED IN THE RULE MATRIX REQUIRE NEGATION TO PRODUCE
C                 THEIR INTENDED VALUES
 10   VECTOR(I)=.NOT.(RULES(I,NSTCOL).AND.TRUE)
C
C DO FOREVER  -  "DRIVE" THE USER'S BASE SUB-MODULE - DRIVEN
C
      NEVER=1
      DO 9000 NOW=1,NEVER
        NEVER=NEVER+1
        CALL DRIVEN(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES
     +    ,MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,
     +     MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
        IF(CURULE.EQ.0) GO TO 20
        NXTROW=STGY(CURULE,ACTIVE)
        IF(NXTROW.NE.0) GO TO 9000
 20     CONTINUE
      STOP
 9000 ACTIVE=NXTROW
      STOP
      END
```

Program DRIVER

```
RELEASE 2.0              MATCH              DATE = 80124          14/25/50

      LOGICAL FUNCTION MATCH(R,M,V,NPREDS)                              00105000
      LOGICAL*1 R(NPREDS),M(NPREDS),V(NPREDS),TRUE/.TRUE./,X,Y          00105100
C COMPARES THE STATUS VECTOR WITH A RULE VECTOR & RETURNS .TRUE. IF     00105200
C     THEY MATCH, . ELSE .FALSE.                                        00105210
      MATCH=.TRUE.                                                      00105300
      DO 10 I=1,NPREDS                                                  00105400
C X IS A STATUS VECTOR BYTE VALUE                                       00105450
      X=V(I).AND.M(I)                                                   00105500
C Y IS THE CORRESPONDING RULE BYTE VALUE AFTER ELIMINATING THE         00105510
C     UNWANTED PART OF THE RULE SYMBOL, NEGATING (BECAUSE THE           00105520
C     SYMBOLS CHOSEN FOR THE TABLE GIVE THE OPPOSITE RESULT TO          00105530
C     WHAT IS REQUIRED), AND MASKING.                                   00105540
      Y=(.NOT.(R(I).AND.TRUE)).AND.M(I)                                 00105600
      MATCH=((X.AND.Y).OR.(.NOT.X.AND..NOT.Y)).AND.MATCH                00105700
      IF(.NOT.MATCH) RETURN                                             00105800
   10 CONTINUE                                                          00105900
      RETURN                                                            00106000
      END                                                               00106100
```

Program DRIVER. Function MATCH

```
RELEASE 2.0              DRIVEN              DATE = 80124          14/25/50

      SUBROUTINE DRIVEN(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
     + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,
     + MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
      INTEGER TABSIZ,CURULE
      INTEGER PDTABL(TABSIZ),STGY(MAXRLS,MAXSTR),ACTNOS(MAXACT,MAXRLS),
     +  ATRANS(MACTS),AINTNO(MACTIG,2),BLANK,ACTIVE
      LOGICAL*1 TABL(8),RULES(MAXPRD,MAXRLS),VECTOR(MAXPRD),
     +   MASKS(MAXPRD,MAXRLS)
      LOGICAL*1 INIT/.TRUE./,TRUE/.TRUE./
      LOGICAL MATCH
      DATA BLANK/' '/
C APPLICATION SPECIFICATIONS*******************************************
C---GLOBAL VARIABLES--
      INTEGER APDTAB,CMND,INUNO,AMAXST,NENTS,LINUM,RULNO,NASTRL,INUNIT
      INTEGER NFEOF,NFLDS,NARPRD,CLUSTN,CLUSLN,NACRLS,ICTAB,FLDCTR
      INTEGER INTRUL,ALFRUL,LGTH,IFREST,AACTNO,ACASE,AATRNS,AAINTN
      INTEGER ACTIT,NACASE,LIMIT,NCLUSR,NCL,MORE,KKCOND,JPRPTR,KONTIN
      INTEGER ZI411,NAPRDS,NAACTS,NAACTG,ZI412,NF3REC,ZI413,IBLK1,IBLK3
      INTEGER NAIACS,ZI414,NARULE,NAARL,ZI415
      LOGICAL*1 CLINBL,PLINBL,INUNID,PDT,NEWOP,TSTBIT,ZL111,ARULES
      LOGICAL*1 AMASKS,LOGNO,LOGYES,LOGDOL,LOGBLK,LOGV,LOGSTR,LOGA
      LOGICAL*1 ZL112,ATAB,ZL113
      COMMON/I4/APDTAB(375),CMND,INUNO,AMAXST,NENTS,LINUM,RULNO,NASTRL
      COMMON/I4/INUNIT,NFEOF,NFLDS,NARPRD,CLUSTN,CLUSLN,NACRLS,ICTAB
      COMMON/I4/FLDCTR,INTRUL(50),ALFRUL(50),LGTH,IFREST,AACTNO(20,120)
      COMMON/I4/ACASE(120,20),AATRNS(120),AAINTN(150,3),ACTIT(8,20)
      COMMON/I4/NACASE,LIMIT,NCLUSR,NCL,MORE,KKCOND,JPRPTR,KONTIN
      COMMON/I4/ZI411(6),NAPRDS,NAACTS,NAACTG,ZI412(3),NF3REG,ZI413(5)
      COMMON/I4/IBLK1(100),IBLK3(100),NAIACS,ZI414(3),NARULE,NAARL
      COMMON/I4/ZI415(4)
      COMMON/L1/CLINBL,PLINBL,INUNID,PDT,NEWOP,TSTBIT,ZL111(2)
      COMMON/L1/ARULES(50,100),AMASKS(50,100),LOGNO,LOGYES,LOGDOL,LOGBLK
      COMMON/L1/LOGV,LOGSTR,LOGA,ZL112(4),ATAB(8),ZL113
C--LOCAL VARIABLES---
```

Sub-module DRIVEN

```
      INTEGER BYTPOS,FMTSYM(6),FMAT(103),NAME(7),JFLD(50),AMAXPR,AMAXAC
      INTEGER ASTGY(120,20),ATABSZ,ASTIT(8,20),STNO,YES,NO,CLBASE
      INTEGER EXISTS(8),CARD(20),BASNAM(2),FTUNIT
      LOGICAL*1 ATABL(8),AVECTR(50),AINIT/.TRUE./,LOGRUL(50)
      REAL*8 ISYM,INTSYM
      EQUIVALENCE(INTSYM,LOGNO),(APDTAB(151),FMAT(1)),(APDTAB(101),
     +   JFLD(1)),(APDTAB(1),PGMNAM)
      DATA ISYM/'NYS V*A'/,BASNAM/'DRIV','EN'/
      DATA FMTSYM/'A1,','I1,','(','T1,','80A1',')'/
      EQUIVALENCE (APDTAB(19),ATABL(1))
      DATA NBLNK/' '/,NEW,MMOD/'NEW','MOD'/,YES,NO/'YES','NO'/
      REAL*8 TABNAM(4)/'    RULE','  ACTION','STRATEGY',' CLUSTER'/
      REAL*8 PGMNAM
      DEFINE FILE 2(20,1500,L,II2)
      DEFINE FILE 3(420,84,L,II3)
      DEFINE FILE 8(35,400,L,II8)
      DEFINE FILE 4(100,84,L,II4)
      INTSYM=ISYM
      AMAXRL=120
      AMAXPR=50
      AMAXSY=20
      AMAXAC=20
      NF3REC=420
      IF(AINIT) PDT=.FALSE.
      IF(AINIT) NF=2
      AINIT=.FALSE.
C ***********************************************************************
      IF(.NOT.INIT) CALL ZRIVEZ(VECTOR,NRPRDS)
      INIT=.FALSE.
      CURULE=0
      DO 99999 I99999=1,NRULES
      IF(STGY(I99999,ACTIVE).EQ.BLANK) GO TO 99999
      IF(.NOT.MATCH(RULES(1,I99999),MASKS(1,I99999),VECTOR,NRPRDS))
     +     GO TO 99999
```

Sub-module DRIVEN

```
      CURULE=I99999
      IF(STGY(CURULE,ACTIVE).EQ.0) INIT=.TRUE.
      DO 99000 I99000=1,MAXACT
      NEXTAC=ACTNOS(I99000,CURULE)
      IF(NEXTAC.EQ.0) RETURN
      NXTACT=ATRANS(NEXTAC)
      DO 99998 I99998=1,MAXACT
      IF(NXTACT.EQ.0) GO TO 99000
      NACSHN=AINTNQ(NXTACT,1)
      GO TO (99997,99996,99995,99994,99993,99992,99991,99990,99969,99988
     +,99987,99986,99985,99984,99983,99982,99981,99980,99979,99978
     +,99977,99976,99975,99974,99973,99972,99971,99970,99969,99968
     +,99967,99966,99965,99964,99963,99962,99961,99960,99959,99958
     +,99957,99956,99955,99954,99953,99952,99951,99950,99949,99948
     +,99947,99946,99945,99944,99943,99942,99941,99940,99939,99938
     +,99937,99936,99935,99934,99933,99932,99931,99930,99929,99928
     +,99927,99926,99925,99924,99923,99922,99921,99920,99919,99918
     +,99917,99916,99915,99914,99913,99912,99911,99910,99909,99908
     +,99907,99906,99905,99904,99903,99902,99901,99900,99899,99898
     +,99897,99896,99895,99894,99893,99892,99891,99890,99889,99888
     +,99887,99886,99885,99884,99883,99882,99881,99880,99879,99878
     +,99877,99876,99875,99874,99873,99872),NACSHN
C ACTION ENTRIES START HERE
C ACTION 1
C
99997 CONTINUE
      WRITE(6,999)
  999 FORMAT(/' COMMAND')
      READ(9,998) CMND
  998 FORMAT(20A4)
      GO TO 99998
C ACTION 2
C
99996 CONTINUE
```

Sub-module DRIVEN

```
RELEASE 2.0                DRIVEN               DATE = 80124          14/25/50
      WRITE(6,997) CMND
 997  FORMAT(/' COMMAND ',A4,' NOT RECOGNISED')
      GO TO 99998
C ACTION 3 INITIALISE NEW PDT
C
99995 CONTINUE
      DO 3 I=1,375
 3      APDTAB(I)=0
      APDTAB(258)=-1
      APDTAB(1)=BASNAM(1)
      APDTAB(2)=BASNAM(2)
      APDTAB(71)=FTUNIT
      GO TO 99998
C ACTION 4    READ NAME OF IN-UNIT
C
99994 CONTINUE
      WRITE(6,995)
 995  FORMAT(/' MOD')
      READ(9,998) INUNIT
      GO TO 99998
C ACTION 5    WRITE PDT TO DISK
C
99993 CONTINUE
      WRITE(NF'1) APDTAB
      GO TO 99998
C ACTION 6    UPDATE IN-UNIT VARIABLES IN PDT
C
99992 CONTINUE
      APDTAB(3)=NASTRT
      APDTAB(4)=NACASE
      APDTAB(5)=NARPRD
      APDTAB(6)=NAACTS
      APDTAB(7)=NASTRD
      APDTAB(8)=NASTCO
```

Sub-module DRIVEN

```
      APDTAB(9)=NASTRL
      APDTAB(10)=NACRLS
      APDTAB(11)=NARULE
      APDTAB(12)=NAARL
      APDTAB(13)=NAIACS
      APDTAB(14)=MAACTA
      APDTAB(15)=NAPRDS
      APDTAB(16)=NAACTA
      APDTAB(17)=NFLDS
      APDTAB(18)=NAACTG
      DO 5 I=1,8
 5    ATABL(I)=ATAB(I)
      GO TO 99998
C ACTION 7
C
99991 CONTINUE
      INUNIT=BLANK
      GO TO 99998
C ACTION 8
C
99990 CONTINUE
      CMND=BLANK
      GO TO 99998
C ACTION 9
C
99989 CONTINUE
      WRITE(6,994) INUNIT
 994  FORMAT(/'  END ',A4)
      GO TO 99998
C ACTION 10   NO-OP
C
99988 CONTINUE
      GO TO 99998
C ACTION 11
```

Sub-module DRIVEN

```
C
99987 CONTINUE
      WRITE(6,993) INUNIT
 993  FORMAT(/' UNIT ',A4,' EXISTS')
      GO TO 99998
C ACTION 12
C
99986 CONTINUE
      WRITE(6,992) INUNIT
 992  FORMAT(/' UNIT ',A4,' NOT RECOGNISED')
      GO TO 99998
C ACTION 13   ESTABLISH NUMBER OF PREDICATE GROUPS
C
99985 CONTINUE
      NFLDS=NENTS
      GO TO 99998
C GENERATED ACTION - STACK THE PDT
C
99984 CONTINUE
      WRITE(30) PDTABL
      GO TO 99998
C ACTION 15   BLANK THE NEW PORTION OF THE STRATEGY TABLE
C
99983 CONTINUE
      J=NASTRL+1
      K=RULNO
      DO 10 L=J,K
      DO 10 I=1,AMAXST
  10  ASTGY(L,I)=NBLNK
      GO TO 99998
C ACTION 16   MARK CURRENT LINE "NOT BLANK"
C
99982 CONTINUE
      CLINBL=.FALSE.
```

Sub-module DRIVEN

```
      GO TO 99998
C ACTION 17   ESTABLISH START OF STRATEGY TABLE
C
99981 CONTINUE
      WRITE(6,980)
  980 FORMAT(/' IDENTIFY START OF EXECUTION LOCATION AS RULE NO. , STRAT
     +EGY NO.')
      READ(9,*) NASTCO,NASTRO
      GO TO 99998
C ACTIONS 18 - 24 SET IDENTIFIER FOR VARIOUS IN-UNITS...(THIS 1 UNKNOWN)
C
99980 CONTINUE
      INUNO=0
      GO TO 99998
C ACTION 19    ... RULES...
C
99979 CONTINUE
      INUNO=1
      CLBASE=260
      GO TO 99998
C ACTION 20    ... ACTIONS...
C
99978 CONTINUE
      CLBASE=290
      INUNO=2
      GO TO 99998
C ACTION 21    ... STRATEGIES...
C
99977 CONTINUE
      INUNO=3
      GO TO 99998
C ACTION 22    ... CLUSTERS...
C
99976 CONTINUE
```

Sub-module DRIVEN

```
          INUNO=4
          GO TO 99998
C ACTION 23     ... CODE...
C
99975 CONTINUE
          INUNO=5
          GO TO 99998
C ACTION 24     ... CODE
C
99974 CONTINUE
          INUNO=6
          GO TO 99998
C GENERATED ACTION   - GET COUNTER VARIABLES FROM PDT
C
99973 CONTINUE
          NSTROW=PDTABL(7)
          NSTCOL=PDTABL(8)
          NRULES=PDTABL(11)
          NRPRDS=PDTABL(5)
          MACACT=PDTABL(14)
          NACTS=PDTABL(6)
          NARLS=PDTABL(12)
          NSTRTS=PDTABL(3)
          NSTRLS=PDTABL(9)
          NACTG=PDTABL(18)
          NTF=PDTABL(71)
          GO TO 99998
C GENERATED ACTION   - STACK STATUS VARIABLES
C
99972 CONTINUE
          WRITE(30) (VECTOR(I),I=1,NRPRDS),ACTIVE,CURULE,NXTROW
          GO TO 99998
C ACTION 27
C
```

Sub-module DRIVEN

```
RELEASE 2.0                DRIVEN              DATE = 80124          14/25/50

  99971 CONTINUE
        WRITE(6,982) INUNIT
   982  FORMAT(/' HOW MANY ',A4,' ENTRIES?  (MAX. 32 ALPHANUMERICS EACH)')
        GO TO 99998
 C ACTION 28
 C
  99970 CONTINUE
        READ(9,*) NENTS
        GO TO 99998
 C ACTION 29
 C
  99969 CONTINUE
        WRITE(6,988)
   988  FORMAT(/' **** ERROR **** POSITIVE VALUES ONLY - RE-ENTER')
        GO TO 99998
 C ACTION 30   ESTABLISH NUMBER OF STRATEGIES
 C
  99968 CONTINUE
        NASTRT=NENTS
        GO TO 99998
 C ACTION 31
 C
  99967 CONTINUE
        LINUM=0
        GO TO 99998
 C ACTION 32
 C
  99966 CONTINUE
        WRITE(6,987)
   987  FORMAT(/' ENTRY?')
        GO TO 99998
 C ACTION 33   READ A STRATEGY TITLE
 C
  99965 CONTINUE
```

Sub-module DRIVEN

```
      READ(9,998) (ASTIT(I,LINUM),I=1,8)
      GO TO 99998
C ACTION 34
C
99964 CONTINUE
      LINUM=LINUM+1
      GO TO 99998
C ACTION 35
C
99963 CONTINUE
      WRITE(6,986) TABNAM(INUNO),INUNIT
 986  FORMAT(/1X,A8,' MATRIX ENTRIES IN THE FORM:'/10X,'RULE NO. ',A4,
     +' NO.,CONTROL NO.   (0,0,0 ENTRY TERMINATES INPUT)')
      GO TO 99998
C ACTION 36   REGISTER SIZE OF STRATEGY TABLE (NO. OF RULES COVERED)
C
99962 CONTINUE
      NASTRL=RULNO
      GO TO 99998
C ACTION 37   FILL IN STRATEGY ENTRIES
C
99961 CONTINUE
      ASTGY(RULNO,STNO)=NEXTST
      GO TO 99998
C ACTION 38   POINT TO BEGINNING OF DISK STORAGE FOR IN-UNIT,
C
99960 CONTINUE
      CLINBL=.TRUE.
      I12=IFREST
      ISTART=I12
      GO TO 99998
C ACTION 39
C
99959 CONTINUE
```

Sub-module DRIVEN

```
RELEASE 2.0              DRIVEN              DATE = 80124          14/25/50

      NENTS=0
      GO TO 99998
C ACTION 40   MARK IN-UNIT AS "EXISTING"
C
99958 CONTINUE
      ATAB(INUND)=.TRUE.
      GO TO 99998
C GENERATED ACTION  - SET FILE NUMBER FOR TRANSFER TO NEW SUB-MODULE
C
99957 CONTINUE
      NTF=11
      GO TO 99998
C ACTION 42   READ IN PDT
C
99956 CONTINUE
      READ(NF'1) APDTAB
      PDT=.TRUE.
      GO TO 99998
C ACTION 43   PRINT STRATEGY TABLE
C
99955 CONTINUE
      J=1
      K=60
      L=(NASTRL-1)/60+1
      DO 25 I=1,L
        IF(I.EQ.L) K=NASTRL
        WRITE(6,979) TABNAM(INUND),J,K,((M,M=1,9),N=1,6)
        DO 20 N=1,NASTRT
 20       WRITE(6,985) N,(ASTIT(M,N),M=1,8),(ASTGY(M,N),M=J,K)
        WRITE(6,978) (M,M=1,6)
        J=K+1
 25     K=J+59
 979  FORMAT(//1X,A8,' TABLE',22X,'(RULES',I3,'-',I3,')'/
     +37X,'0',6(9I1,'0')/28X,7(9X,'V'))
```

Sub-module DRIVEN

```
RELEASE 2.0                DRIVEN                DATE = 80124           14/25/50
  985  FORMAT(1X,I2,2X,8A4,1X,60I1)
  978  FORMAT(37X,'0',6I10)
       GO TO 99998
C ACTION 44    READ STRATEGY ENTRY
C
99954 CONTINUE
       READ(9,*) RULNO,STNO,NEXTST
       GO TO 99998
C GENERATED ACTION  - READ PDT BEFORE/AFTER CONTROL TRANSFER
C
99953 CONTINUE
       READ(NTF'1) PDTABL
       GO TO 99998
C ACTION 46
C
99952 CONTINUE
       WRITE(6,984)
  984  FORMAT(/' COMMAND OR UNIT NOT IMPLEMENTED YET')
       GO TO 99998
C ACTION 47    FETCH IN-UNIT VARIABLES FROM PDT
C
99951 CONTINUE
       NASTRT=APDTAB(3)
       NACASE=APDTAB(4)
       NARPRD=APDTAB(5)
       NAACTS=APDTAB(6)
       NASTRU=APDTAB(7)
       NASTCO=APDTAB(8)
       NASTRL=APDTAB(9)
       NACRLS=APDTAB(10)
       NARULE=APDTAB(11)
       NAARL=APDTAB(12)
       NAIACS=APDTAB(13)
       MAACTA=APDTAB(14)
```

Sub-module DRIVEN

```
      NAPRDS=APDTAB(15)
      NAACTA=APDTAB(16)
      NFLDS =APDTAB(17)
      NAACTG=APDTAB(18)
      DO 15 I=1,8
 15   ATAB(I)=ATABL(I)
      GO TO 99998
C ACTION 48
C
99950 CONTINUE
      NEWOP=.FALSE.
      GO TO 99998
C ACTION 49
C
99949 CONTINUE
      NEWOP=.TRUE.
      GO TO 99998
C GENERATED ACTION  - READ IN-UNITS BEFORE/AFTER CONTROL TRANSFER
C
99948 CONTINUE
      II=PDTABL(21)-(PDTABL(21)/256*256)-29
      READ(NTF'II) ((RULES(I,J),I=1,NRPRDS),J=1,NRULES),
     +    ((MASKS(I,J),I=1,NRPRDS),J=1,NRULES)
      II=PDTABL(22)-PDTABL(22)/256*256-29
      READ(NTF'II) ((ACTNOS(I,J),I=1,MACACT),J=1,NARLS)
      II=PDTABL(23)-PDTABL(23)/256*256-29
      READ(NTF'II) ((STGY(I,J),I=1,NSTRLS),J=1,NSTRTS)
      II=PDTABL(25)-PDTABL(25)/256*256-29
      READ(NTF'II) (ATRANS(I),I=1,NACTS),((AINTNO(I,J),I=1,NACTG),
     +    J=1,2)
      GO TO 99998
C ACTION 51   SET SIZE OF CLUSTER TABLE (NO. OF CLUSTERS)
C
99947 CONTINUE
```

Sub-module DRIVEN

```
RELEASE 2.0              DRIVEN            DATE = 80124          14/25/50
        NACASE=NENTS
        GO TO 99998
C ACTION 52
C
99946 CONTINUE.
      CMND=MMOD
      GO TO 99998
C ACTION 53
C
99945 CONTINUE
      CMND=NEW
      GO TO 99998
C ACTION 54   PRINT IN-UNIT SUMMARY
C
99944 CONTINUE
      WRITE(6,977) APDTAB(1),APDTAB(2)
      DO 30 I=1,8
        EXISTS(I)=NO
 30     IF(ATABL(I)) EXISTS(1)=YES
      WRITE(6,976) (EXISTS(I),I=1,6)
      IF(ATABL(3)) WRITE(6,975) NASTRT,NASTRL
      IF(ATABL(4)) WRITE(6,974) NACASE,NACKLS
      IF(ATABL(6)) WRITE(6,973) NAPRDS
      IF(ATABL(1)) WRITE(6,972) NARULE,NARPRD
      IF(ATABL(5)) WRITE(6,971) NAACTS
      IF(ATABL(2)) WRITE(6,970) NAARL,NAACTA
 977  FORMAT(//' SUB-MODULE:   ',A4,A2)
 976  FORMAT(/15X,'IN-UNIT',15X,'EXISTS'/9X,'RULE MATRIX',T40,A3/
     +9X,'ACTION MATRIX',T40,A3/9X,'STRATEGY MATRIX',T40,A3/
     +9X,'CLUSTER MATRIX',T40,A3/9X,'ACTION SOURCE CODE',T40,A3/
     +9X,'PREDICATE SOURCE CODE',T40,A3//)
 975  FORMAT(' THERE ARE ',I3,' STRATEGY ENTRIES SPANNING ',I3,' RULES')
 974  FORMAT(' THERE ARE ',I3,' CLUSTERS SPANNING ',I3,' RULES')
 973  FORMAT(' THERE ARE ',I3,' PREDICATES')
```

Sub-module DRIVEN

```
  972  FORMAT(' THERE ARE ',I3,' RULES DERIVED FROM ',I3,' PREDICATES')
  971  FORMAT(' THERE ARE ',I3,' ACTIONS')
  970  FORMAT(' THE ACTION MATRIX COVERS ',I3,' RULES AND ',I3,' ACTIONS'
      +)
       GO TO 99998
 C ACTION 55   CALCULATE SIZE OF STRATEGY IN-UNIT IN BYTES
 C
 99943 CONTINUE
       LGTH=NASTRT*(8+NASTRL)*4
       GO TO 99998
 C ACTION 56   - GENERATED ACTION - MINI-DRIVER FOR IGETBL
 C
 99942 CONTINUE
       ACTIVE=NSTROW
       DO 99002 I99002=1,NRPRDS
 99002   VECTOR(I99002)=.NOT.(RULES(I99002,NSTCOL).AND.TRUE)
       N99003=1
       DO 99003 I99003=1,N99003
         N99003=N99003+1
         CALL IGETBL(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
      +       MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,
      +       ATRANS,MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
         IF(CURULE.EQ.0) GO TO 99004
         NXTROW=STGY(CURULE,ACTIVE)
         IF(NXTROW.EQ.0) GO TO 99004
 99003   ACTIVE=NXTROW
 99004 CONTINUE
       GO TO 99998
 C ACTION 57   STORE STRATEGY IN-UNIT
 C
 99941 CONTINUE
       WRITE(NF'II2) ((ASTGY(I,J),I=1,NASTRL),J=1,NASTRT),
      +              ((ASTIT(I,J),I=1,8),J=1,NASTRT)
       GO TO 99998
```

Sub-module DRIVEN

```
RELEASE 2.0              DRIVEN              DATE = 80124          14/[illegible]5/50
      C ACTION 58   STORE RULE IN-UNIT
      C
      99940 CONTINUE
            WRITE(NF'II2) ((ARULES(I,J),I=1,NARPRD),J=1,NARULE),
           +              ((AMASKS(I,J),I=1,NARPRD),J=1,NARULE)
            GO TO 99998
      C ACTION 59   STORE ACTION IN-UNIT
      C
      99939 CONTINUE
            WRITE(NF'II2) ((AACTNO(I,J),I=1,MAACTA),J=1,NAARL)
            GO TO 99998
      C ACTION 60   STORE CLUSTER IN-UNIT
      C
      99938 CONTINUE
            WRITE(NF'II2) ((ACASE(I,J),I=1,NACRLS),J=1,NACASE),
           +              ((ACTIT(I,J),I=1,8),J=1,NACASE)
            GO TO 99998
      C ACTION 61
      C
      99937 CONTINUE
            WRITE(6,969) NFEOF,INUNIT
        969 FORMAT(/' INSUFFICIENT SPACE AVAILABLE ON D.A. UNIT ',I2,
           +       ' TO STORE ',A4/)
            GO TO 99998
      C ACTION 62
      C
      99936 CONTINUE
            NFEOF=0
            GO TO 99998
      C ACTION 63   ZERO EXTRA SPACE IN CLUSTER TABLE
      C
      99935 CONTINUE
            NCOUNT=0
            NCLRLS=0
```

Sub-module DRIVEN

```
      DO 80 I=1,NACRLS
      IF(ACASE(I,CLUSLN).EQ.0) GO TO 80
      NCLRLS=NCLRLS+1
      IF(I.GT.NCOUNT) NCOUNT=I
 80   CONTINUE
      GO TO 99998
C ACTION 64   PRINT CLUSTER TABLE
C
99934 CONTINUE
      J=1
      K=60
      L=(NACRLS-1)/60+1
      DO 40 I=1,L
        IF(I.EQ.L) K=NACRLS
        WRITE(6,979) TABNAM(INUND),J,K,((M,M=1,9),N=1,6)
        DO 35 N=1,NACASE
 35       WRITE(6,985) N,(ACTIT(M,N),M=1,7),NBLNK,(ACASE(M,N),M=J,K)
        WRITE(6,978) (M,M=1,6)
      J=K+1
 40   K=J+59
      GO TO 99998
C ACTION 65   CALCULATE LENGTH OF CLUSTER IN-UNIT IN BYTES
C
99933 CONTINUE
      LGTH=NACASE*(8+NACRLS)*4
      GO TO 99998
C ACTION 66   READ A CLUSTER ENTRY
C
99932 CONTINUE
      READ(9,998) (ACTIT(I,LINUM),I=1,7)
      ACTIT(8,LINUM)=0
      WRITE(6,967)
 967  FORMAT(' RULES?')
      GO TO 99998
```

Sub-module DRIVEN

```
RELEASE 2.0                DRIVEN                DATE = 80124        14/25/50

C ACTION 67
C
99931 CONTINUE
      READ(9,*) NCLUSR
      GO TO 99998
C ACTION 68   FILL IN BLOCK USAGE DIRECTORY OF PDT FOR CURRENT IN-UNIT
C
99930 CONTINUE
      J=NACRLS+1
      NACRLS=NACRLS+NCLUSR
      DO 50 I=J,NACRLS
      DO 45 K=1,NACASE
 45      ACASE(I,K)=0
 50    ACASE(I,LINUM)=1
      GO TO 99998
C ACTION 69
C
99929 CONTINUE
      IEND=II2-1
      DO 55 I=ISTART,IEND
 55    APDTAB(I+29)=1
      APDTAB(20+INUNO)=(II2-ISTART)*256+ISTART+29
      GO TO 99998
C ACTION 70
C
99928 CONTINUE
      WRITE(6,959)
 959  FORMAT(/' EACH PREDICATE CODED EVALUATES TO ''Y'' OR ''N''.  HOWEV
     +ER, RULES MAY BE ENTERED USING AN EXTENDED ENTRY FORM (INTEGER)'/
     +5X,' FOR GROUPS OF DEPENDENT CONDITIONS'/' FROM HOW MANY PREDICATE
     +S ARE THE RULES DERIVED?')
      GO TO 99998
C ACTION 71   SET NUMBER OF PREDICATES
C
```

Sub-module DRIVEN

```
99927 CONTINUE
      NARPRD=NENTS
      WRITE(6,966)
  966 FORMAT(/' INTO HOW MANY INDEPENDENT FIELDS ARE THE PREDICATES GROU
     +PED?')
      GO TO 99998
C ACTION 72   MARK ALL PREDICATES AS "INDEPENDENT"
C
99926 CONTINUE
      DO 60 I=1,NENTS
   60 JFLD(I)=1
      GO TO 99998
C ACTION 73   ESTABLISH PREDICATE "GROUPS"
C
99925 CONTINUE
      WRITE(6,965) NENTS
  965 FORMAT(/' STARTING FROM PREDICATE NO.1,  HOW MANY PREDICATES FORM
     +EACH OF THE ',I2,' INDEPENDENT FIELDS?')
      READ(9,*) (JFLD(I),I=1,NENTS)
      GO TO 99998
C ACTION 74   SET UP FORMAT LINE TO READ RULES
C
99924 CONTINUE
      FMAT(1)=FMTSYM(3)
      K=1
      DO 70 I=1,2
      DO 65 J=1,NENTS
      M=1
      IF(JFLD(J).GT.1) M=2
   65 FMAT(K+J)=FMTSYM(M)
      K=NENTS+K+1
   70 FMAT(K)=FMTSYM(4)
      FMAT(K+1)=FMTSYM(5)
      FMAT(K+2)=FMTSYM(6)
```

Sub-module DRIVEN

```
      GO TO 99998
C ACTION 75
C
99923 CONTINUE
      WRITE(6,964) TABNAM(INUNO)
 964  FORMAT(/' CLUSTER TABLE MUST EXIST BEFORE ',A8,' ENTRIES CAN BE MA
     +DE')
      GO TO 99998
C ACTION 76   SET COUNT LIMIT TO NUMBER OF CLUSTERS
C
99922 CONTINUE
      NENTS=NACASE
      GO TO 99998
C ACTION 77   PRINT CLUSTER TITLES
C
99921 CONTINUE
      WRITE(6,963) (J,(ACTIT(I,J),I=1,7),J=1,NACASE)
 963  FORMAT(/' CLUSTERS STORED AT PRESENT ARE:'/(1X,I2,3X,7A4))
      GO TO 99998
C ACTION 78   CHECK SPECIFIED NAME AGAINST CLUSTER TABLE
C
99920 CONTINUE
      CLUSLN=LINUM
      DO 73 J=1,7
 73   IF(ACTIT(J,LINUM).NE.NAME(J)) CLUSLN=0
      LINUM=LINUM-CLUSLN
      GO TO 99998
C ACTION 79
C
99919 CONTINUE
      WRITE(6,961)
 961  FORMAT(/' CLUSTER COMPLETE')
      APDTAB(CLBASE+CLUSLN)=NCLKLS
      GO TO 99998
```

Sub-module DRIVEN

```
RELEASE 2.0                DRIVEN              DATE = 80124            14/25/50
      C ACTION 80
      C
      99918 CONTINUE
            IF(NCOUNT.GT.NARULE) NARULE=NCOUNT
            K=NFLDS+1
            WRITE(6,960) NCLRLS,(FMAT(1),I=2,K)
        960 FORMAT(/' ENTER ',I2,' RULES.'/1X,80A1)
            GO TO 99998
      C ACTION 81
      C
      99917 CONTINUE
            FLDCTR=0
            GO TO 99998
      C ACTION 82
      C
      99916 CONTINUE
            FLDCTR=FLDCTR+1
            GO TO 99998
      C ACTION 83   READ A RULE
      C
      99915 CONTINUE
            READ(9,FMAT) (LOGRUL(I),I=1,NFLDS),(INTRUL(I),I=1,NFLDS)
           +,(ALFRUL(I),I=1,NFLDS)
            BYTPOS=0
            GO TO 99998
      C ACTION 84   CLEAR RULE MEMBERSHIP (TEMPORARILY) TO AVOID REPEAT
      C
      99914 CONTINUE
            ACASE(LINUM,CLUSLN)=0
            GO TO 99998
      C ACTIONS 85 - 90  MAKE RULE/MASK ENTRIES FROM INPUT LINE
      C
      99913 CONTINUE
            J=JFLD(FLDCTR)
```

Sub-module DRIVEN

```
      DO 85 I=1,J
 85   AMASKS(BYTPOS+I,LINUM)=.FALSE.
      GO TO 99998
C ACTION 86
C
99912 CONTINUE
      BYTPOS=BYTPOS+JFLD(FLDCTR)
      GO TO 99998
C ACTION 87
C
99911 CONTINUE
      J=JFLD(FLDCTR)
      DO 90 I=1,J
 90   AMASKS(BYTPOS+I,LINUM)=.TRUE.
      GO TO 99998
C ACTION 88
C
99910 CONTINUE
      J=JFLD(FLDCTR)
      DO 93 I=1,J
 93   ARULES(BYTPOS+I,LINUM)=LOGRUL(FLDCTR)
      GO TO 99998
C ACTION 89
C
99909 CONTINUE
      J=JFLD(FLDCTR)
      DO 95 I=1,J
 95   ARULES(BYTPOS+I,LINUM)=LOGNO
      GO TO 99998
C ACTION 90
C
99908 CONTINUE
      J=JFLD(FLDCTR)
      DO 100 I=1,J
```

Sub-module DRIVEN

```
  100  ARULES(BYTPOS+I,LINUM)=LOGDOL
       ARULES(BYTPOS+J+1-INTRUL(FLDCTR),LINUM)=LOGYES
       GO TO 99998
C ACTION 91   READ IN CLUSTER IN-UNIT
C
99907 CONTINUE
       II2=APDTAB(24)-(APDTAB(24)/256*256)-29
       READ(2'II2) ((ACASE(I,J),I=1,NACRLS),J=1,NACASE),
     +             ((ACTIT(I,J),I=1,6),J=1,NACASE)
       GO TO 99998
C ACTION 92   FETCH RULE MEMBERSHIP FLAG
C
99906 CONTINUE
       ICTAB=ACASE(LINUM,CLUSLN)
       GO TO 99998
C ACTION 93
C
99905 CONTINUE
       CLUSTN=BLANK
       GO TO 99998
C ACTION 94
C
99904 CONTINUE
       WRITE(6,958)
  958  FORMAT(/' CLUSTER NOT FOUND ')
       GO TO 99998
C ACTION 95   CLEAR AN INPUT RULE FIELD (MAY BE ILLEGAL OR DEPENDANT)
C
99903 CONTINUE
       LOGRUL(FLDCTR)=LOGBLK
       INTRUL(FLDCTR)=0
       ALFRUL(FLDCTR)=BLANK
       GO TO 99998
C ACTION 96   CALCULATE SIZE OF RULE IN-UNIT IN BYTES
```

Sub-module DRIVEN

```
RELEASE 2.0                DRIVEN              DATE = 80124           14/25/50
C
99902 CONTINUE
      LGTH=NARULE*NARPRD*2
      GO TO 99998
C ACTION 97
C
99901 CONTINUE
      WRITE(6,956) TABNAM(INUNO)
 956  FORMAT(/A8,'S MAY ONLY BE SUPPLIED FOR EXISTING EMPTY CLUSTERS')
      GO TO 99998
C GENERATED ACTION   - UNSTACK THE PDT
C
99900 CONTINUE
      BACKSPACE 30
      BACKSPACE 30
      READ(30) PDTABL
      GO TO 99998
C ACTION 99   FIND NUMBER OF RULES IN THIS CLUSTER
C
99899 CONTINUE
      FLDCTR=APDTAB(CLBASE+LINUM+CLUSLN)
      NCLRLS=FLDCTR
      GO TO 99998
C ACTION 100  ZERO CLUSTER ENTRIES UP TO START OF NEW MEMBERSHIP
C
99898 CONTINUE
      DO 105 I=1,NACRLS
 105  ACASE(I,LINUM)=0
      GO TO 99998
C ACTION 101  SET COUNT LIMIT = NO.OF RULES IN CLUSTER TABLE
C
99897 CONTINUE
      LINUM=NACRLS
      GO TO 99998
```

Sub-module DRIVEN

```
RELEASE 2.0                DRIVEN                DATE = 80124             14725/50

      C ACTION 102  READ IN RULE IN-UNIT                                       00082900
      C                                                                        00083000
      99896 CONTINUE                                                           00083100
            II2=APDTAB(21)-(APDTAB(21)/256*256) - 29                           00083200
            READ(2'II2) ((ARULES(I,J),I=1,NARPRD),J=1,NARULE),                 00083300
           +            ((AMASKS(I,J),I=1,NARPRD),J=1,NARULE)                  00083400
            GO TO 99998                                                        00083500
      C ACTION 103                                                             00083600
      C                                                                        00083700
      99895 CONTINUE                                                           00083800
            CLUSLN=0                                                           00083900
            GO TO 99998                                                        00084000
      C ACTION 104  CALCULATE SIZE OF ACTION TABLE IN BYTES                    00084100
      C                                                                        00084200
      99894 CONTINUE                                                           00084300
            LGTH=MAACTA*NAARL*4                                                00084400
            GO TO 99998                                                        00084500
      C ACTION 105                                                             00084600
      C                                                                        00084700
      99893 CONTINUE                                                           00084800
            IF(NCOUNT.GT.NAARL) NAARL=NCOUNT                                   00084900
            WRITE(6,955) NCLRLS                                                00085000
        955 FORMAT(/' ENTER ',I2,' LISTS OF ACTION NUMBERS')                   00085100
            GO TO 99998                                                        00085400
      C ACTION 106  FILL IN THE ACTION TABLE                                   00085500
      C                                                                        00085600
      99892 CONTINUE                                                           00085700
            NCOL=1                                                             00085800
            READ(9,998) CARD                                                   00085810
            DO 107 N=1,40                                                      00085820
              CALL STRING(CARD,NCOL,NUM,MORE,LAST,4)                           00085830
              CALL CONVRT(NUM,AACTNO(N,LINUM),IOK)                             00085840
              IF(MORE.EQ.1) GO TO 108                                          00085850
        107   CONTINUE                                                         00085860
```

Sub-module DRIVEN

```
RELEASE 2.0                DRIVEN               DATE = 80124          14/25/50
  108  AACTNO(N+1,LINUM)=0
       IF(N.GT.MAACTA) MAACTA=N
       DO 110 I=1,N
  110  IF(AACTNO(I,LINUM).GT.NAACTA) NAACTA=AACTNO(I,LINUM)
       GO TO 99998
C ACTION 107  READ IN THE ACTION IN-UNIT
C
99891 CONTINUE
      II2=APDTAB(22)-(APDTAB(22)/256*256)-29
      READ(2'II2) ((AACTNO(I,J),I=1,MAACTA),J=1,NAARL)
      GO TO 99998
C GENERATED ACTION  - UNSTACK THE STATUS VARIABLES
C
99890 CONTINUE
      READ(30) (VECTOR(I),I=1,NRPRDS),ACTIVE,CURULE,NXTROW
      BACKSPACE 30
      BACKSPACE 30
      GO TO 99998
C ACTION 108
C
99889 CONTINUE
      WRITE(6,954)
  954 FORMAT(/' RULE MATRIX AND ACTION CODE (HEADINGS AT LEAST) MUST EXI
     +ST BEFORE TRACE IS POSSIBLE')
      GO TO 99998
C ACTION 109  READ IN ACTION TRANSLATION TABLES
C
99888 CONTINUE
      II2=APDTAB(25)-(APDTAB(25)/256*256)-29
      READ(2'II2)(AATRNS(I),I=1,NAACTS),((AAINTN(I,J),I=1,NAACTG),J=1,3)
      GO TO 99998
C GENERATED ACTION  - SET FILE NUMBER FOR CONTROL TRANSFER (TRACE)
C
99887 CONTINUE
```

```
      NTF=12
      GO TO 99998
C ACTION 110  - GENERATED ACTION - MINI-DRIVER FOR TRACE
C
99886 CONTINUE
      ACTIVE=NSTROW
      DO 99005 I99005=1,NRPRDS
99005   VECTOR(I99005)=.NOT.(RULES(I99005,NSTCOL).AND.TRUE)
      N99006=1
      DO 99006 I99006=1,N99006
        N99006=N99006+1
        CALL TRACE(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,MACACT,
     +    CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,MACTS,
     +    AINTNO,MACTIG,MAXSTR,ACTIVE)
        IF(CURULE.EQ.0) GO TO 99007
        NXTROW=STGY(CURULE,ACTIVE)
        IF(NXTROW.EQ.0) GO TO 99007
99006   ACTIVE=NXTROW
99007 CONTINUE
      GO TO 99998
C ACTION 111
C
99885 CONTINUE
      CLINBL=ATABL(1).AND.ATABL(5)
      GO TO 99998
C ACTION 14   SEE IF THIS IS A NEW PROJECT
C
99884 CONTINUE
      READ(8'3) IBLK3
      CMND=IBLK3(1)
      GO TO 99998
C ACTION 25   SET UNIT NUMBER = 1 FOR SUB-MODULE DRIVEN
C
99883 CONTINUE
```

Sub-module DRIVEN

```
RELEASE 2.0              DRIVEN              DATE = 80124          14/25/50

      WRITE(6,953)
  953 FORMAT(' STARTING NEW PROJECT:  ENTER NAME')
      READ(9,998) IBLK3(1),IBLK3(2)
      CMND=IBLK3(1)
      IBLK3(3)=0
      IBLK3(4)=1
      IB3PTR=5
      FTUNIT=1
      GO TO 99998
C ACTION 26   ENTER SUB-MODULE NAME & UNIT NO. INTO DDICT.
C
99882 CONTINUE
      IBLK3(IB3PTR)=BASNAM(1)
      IBLK3(IB3PTR+1)=BASNAM(2)
      IBLK3(IB3PTR+2)=FTUNIT
      IBLK3(IB3PTR+3)=0
      GO TO 99998
C ACTION 41   WRITE DATA DICTIONARY BLOCK 3 TO DISK
C
99881 CONTINUE
      WRITE(8'3) IBLK3
      GO TO 99998
C ACTION 45   WRITE PROJECT TITLE BLOCK
C
99880 CONTINUE
      WRITE(6,952) IBLK3(1),IBLK3(2)
  952 FORMAT(//24X,24('-')/24X,'+',22X,'+'/24X,'+  PROJECT - ',
     + 2A4,'  +'/24X,'+',22X,'+'/24X,24('-')//)
      GO TO 99998
C ACTION 50   READ NEW SUB-MODULE NAME FOR ENTRY IN DATA DICT.
C
99879 CONTINUE
      WRITE(6,951)
  951 FORMAT(' NEW SUB-MODULE NAME?')
```

Sub-module DRIVEN

```
      READ(9,998) BASNAM
      IBLK3(4)=IBLK3(4)+1
      IB3PTR=IBLK3(4)
      FTUNIT=9+IB3PTR
      IB3PTR=IB3PTR*4+1
      GO TO 99998
C ACTION 112  CALCULATE LENGTH OF ACTION TRANSLATE TABLES IN BYTES
C
99878 CONTINUE
      LGTH=(NAACTS+3*NAACTG)*4
      GO TO 99998
C ACTION 113  WRITE ACTION TRANSLATE TABLES TO DISK
C
99877 CONTINUE
      WRITE(NF'II2) (AATRNS(I),I=1,NAACTS),((AAINTN(I,J),I=1,NAACTG),
     +    J=1,3)
      GO TO 99998
C GENERATED ACTION  - SET FILE NUMBER FOR TRANSFER (BLDCOD)
C
99876 CONTINUE
      NTF=13
      GO TO 99998
C GENERATED ACTION  - MINI DRIVER FOR BLDCOD
C
99875 CONTINUE
      ACTIVE=NSTROW
      DO 99008 I99008=1,NRPRDS
99008   VECTOR(I99008)=.NOT.(RULES(I99008,NSTCOL).AND.TRUE)
      N99009=1
      DO 99009 I99009=1,N99009
        N99009=N99009+1
        CALL BLDCOD(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,MACACT,
     +    CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,MACTS,
     +    AINTNO,MACTIG,MAXSTK,ACTIVE)
```

```
RELEASE 2.0              DRIVEN              DATE = 80124            14/25/50
              IF(CURULE.EQ.0) GO TO 99010
              NXTROW=STGY(CURULE,ACTIVE)
              IF(NXTROW.EQ.0) GO TO 99010
  99009       ACTIVE=NXTROW
  99010 CONTINUE
        GO TO 99998
C ACTION 114  PUT COMPOUND ACTION NUMBERS INTO PDT FOR BLDCOD
C
 99874 CONTINUE
       K=PDTABL(72)
       APDTAB(72)=K
       IF(K.LE.0) GO TO 125
       J=72
       DO 120 I=1,K
         DO 115 L=1,5
  115      APDTAB(J+L)=PDTABL(J+L)
         APDTAB(J+6)=0
         L=APDTAB(J+4)-1
         M=APDTAB(J+3)
         DO 117 N=1,M
  117      APDTAB(L+N)=PDTABL(L+N)
  120    J=J+6
  125  CONTINUE
       GO TO 99998
C ACTION 115  INITIALISE INTERNAL ACTION NUMBER TABLE
C
 99873 CONTINUE
       AAINTN(1,1)=0
       AAINTN(1,2)=0
       AAINTN(1,3)=0
       NAACTG=1
       GO TO 99998
C ACTION 116  IDENTIFY REQUIRED CLUSTER
C
```

Sub-module DRIVEN

RELEASE 2.0 DRIVEN DATE = 80124 14/25/50

```
99872 CONTINUE
      WRITE(6,962)
 962  FORMAT(/' CLUSTER NAME OR END')
      READ(9,998) NAME
      CLUSTN=NAME(1)
      CLUSLN=0
      GO TO 99998
C END OF ACTIONS
99998 NXTACT=AINTND(NXTACT,2)
99000 CONTINUE
      RETURN
99999 CONTINUE
      WRITE(6,90001) (VECTOR(I),I=1,NRPRDS)
90001 FORMAT(' VECTOR NOT MATCHED.  VECTOR IS ',50L1)
      RETURN
      END
```

Sub-module DRIVEN

```
RELEASE 2.0          ZRIVEZ                DATE = 80124           14/25/50
      SUBROUTINE ZRIVEZ(V,NPREDS)                                         00103600
      LOGICAL*1 V(NPREDS)                                                 00103700
C*****APPLICATION SPECIFICATIONS******************************************00103800
C---GLOBAL VARIABLES--                                                    00103900
      INTEGER APDTAB,CMND,INUNO,AMAXST,NENTS,LINUM,RULNO,NASTRL,INUNIT    00104000
      INTEGER NFEOF,NFLDS,NARPRD,CLUSTN,CLUSLN,NACRLS,ICTAB,FLDCTR        00104010
      INTEGER INTRUL,ALFRUL,LGTH,IFREST,AACTNO,ACASE,AATRNS,AAINTN        00104020
      INTEGER ACTIT,NACASE,LIMIT,NCLUSR,NCL,MORE,KRCOND,JPRPTR,KONTIN     00104030
      INTEGER ZI411,NAPRDS,NAACTS,NAACTG,ZI412,NF3REC,ZI413,IBLK1,IBLK3   00104040
      INTEGER NAIACS,ZI414,NARULE,NAARL,ZI415                             00104050
      LOGICAL*1 CLINBL,PLINBL,INUNID,PDT,NEWOP,TSTBIT,ZL111,ARULES        00104060
      LOGICAL*1 AMASKS,LOGNO,LOGYES,LOGDOL,LOGBLK,LOGV,LOGSTR,LOGA        00104070
      LOGICAL*1 ZL112,ATAB,ZL113                                          00104080
      COMMON/I4/APDTAB(375),CMND,INUNO,AMAXST,NENTS,LINUM,RULNO,NASTRL    00104090
      COMMON/I4/INUNIT,NFEOF,NFLDS,NARPRD,CLUSTN,CLUSLN,NACRLS,ICTAB      00104100
      COMMON/I4/FLDCTR,INTRUL(50),ALFRUL(50),LGTH,IFREST,AACTNO(20,120)   00104110
      COMMON/I4/ACASE(120,20),AATRNS(120),AAINTN(150,3),ACTIT(8,20)       00104120
      COMMON/I4/NACASE,LIMIT,NCLUSR,NCL,MORE,KRCOND,JPRPTR,KONTIN         00104130
      COMMON/I4/ZI411(6),NAPRDS,NAACTS,NAACTG,ZI412(3),NF3REC,ZI413(5)    00104140
      COMMON/I4/IBLK1(100),IBLK3(100),NAIACS,ZI414(3),NARULE,NAARL        00104150
      COMMON/I4/ZI415(4)                                                  00104160
      COMMON/L1/CLINBL,PLINBL,INUNID,PDT,NEWOP,TSTBIT,ZL111(2)            00104170
      COMMON/L1/ARULES(50,100),AMASKS(50,100),LOGNO,LOGYES,LOGDOL,LOGBLK00104180
      COMMON/L1/LOGV,LOGSTR,LOGA,ZL112(4),ATAB(6),ZL113                   00104190
C--LOCAL VARIABLES---                                                     00105800
      LOGICAL*1 ATABL(8)                                                  00105900
      EQUIVALENCE (APDTAB(19),ATABL(1))                                   00106000
      INTEGER END,BLANK,NEW,MOD,TRACE,STRAT,CASE,PREDIC,ACSHN,RLMTX       00106100
     +,ACTMTX,YES,NO,STAR,DOLLR                                           00106200
      DATA END,BLANK,NEW,MOD,TRACE,STRAT,CASE,PREDIC,ACSHN,RLMTX,ACTMTX   00106300
     +/'END',' ','NEW','MOD','TRAC','STRA','CLUS','CODE','   #','RMTX',   00106400
     + 'AMTX'/,YES,NO,STAR,DOLLR/'Y','N','*','$'/                         00106500
C*************************************************************************00106600
      V(1)=CMND.EQ.END                                                    00106700
```

Sub-module DRIVEN

```
RELEASE 2.0            ZRIVEZ            DATE = 80124           14/25/50
      V(2)=CMND.EQ.BLANK
      V(3)=CMND.EQ.NEW
      V(4)=CMND.EQ.MOD
      V(5)=CMND.EQ.TRACE
      V(6)=INUNIT.EQ.BLANK
      V(7)=INUNIT.EQ.STRAT
      V(8)=INUNIT.EQ.CASE
      V(9)=INUNIT.EQ.PREDIC
      V(10)=INUNIT.EQ.ACSHN
      V(11)=INUNIT.EQ.RLMTX
      V(12)=INUNIT.EQ.ACTMTX
      V(13)=INUNIT.EQ.END
      V(14)=INUNO.NE.0
      V(15)=ATABL(INUNO)
      V(16)=NFLDS.EQ.0
      V(17)=NARPRD.EQ.0
      V(18)=NARPRD.EQ.NENTS
      V(19)=.NOT.ATABL(4)
      V(20)=CLUSTN.EQ.BLANK
      V(21)=CLUSTN.EQ.END
      V(22)=CLUSLN.EQ.0
      V(23)=LINUM.EQ.0
      V(24)=LINUM.GT.NACRLS
      V(25)=ICTAB.EQ.0
      V(26)=FLDCTR.EQ.0
      V(27)=FLDCTR.GT.NFLDS
      V(28)=APDTAB(100+FLDCTR).EQ.1
      V(29)=ALFRUL(FLDCTR).EQ.BLANK
      V(30)=ALFRUL(FLDCTR).EQ.YES.OR.ALFRUL(FLDCTR).EQ.NO
      V(31)=ALFRUL(FLDCTR).EQ.STAR.OR.ALFRUL(FLDCTR).EQ.DOLLR
      V(32)=INTRUL(FLDCTR).EQ.0
      V(33)=INTRUL(FLDCTR).GT.9.OR.INTRUL(FLDCTR).LT.0
      V(34)=NENTS.GT.LINUM
      V(35)=CLINBL
```

Sub-module DRIVEN

```
RELEASE 2.0              ZRIVEZ              DATE = 80124          14/25/50

      V(36)=NFEOF.EQ.0                                                   00110200
      V(37)=NENTS.LT.1                                                   00110300
      V(38)=NEWOP                                                        00110400
      V(39)=RULNO.EQ.0                                                   00110500
      V(40)=RULNO.LT.0                                                   00110600
      V(41)=RULNO.GT.NASTRL                                              00110700
      RETURN                                                             00110800
      END                                                                00110900
```

Sub-module DRIVEN

```
CLUSTER:  START                                      RULES:  000000
                                                             123456

 1. CMND.EQ.END                                              $Y$$$N
 2. CMND.EQ.BLANK                                            Y$$$$N
 3. CMND.EQ.NEW                                              $$Y$$N
 4. CMND.EQ.MOD                                              $$$Y$N
 5. CMND.EQ.TRACE                                            $$$$YN

                                                             ------
STRATEGY

 1. START                                                    106251
 2. MODIFY PROGRAM - CASE AND ST
 5. TABLE TRACING
 6. NEW PROGRAM - CASE AND STGY

                                                             ------

CLUSTER:  START                                      RULES:  000000
                                                             123456

  1 READ A COMMAND                                           A    A
  2 ILLEGAL COMMAND                                            D
  3 INITIALISE NEW PDT                                         E
  5 WRITE PDT TO DISK                                          FD
  7 CLEAR IN-UNIT NAME                                            B
  8 CLEAR COMMAND                                             A
 10 NO-OP                                                      GC
 18 IN-UNIT NO. = 0                                            B
 26 ENTER SUB-MOD. NAME IN D.D.                                C
 41 D.D. BLK 3 TO DISK                                          AA
 42 READ IN PDT                                                HHH
 47 FETCH PDT VARIABLES                                        A
 50 READ NEW SUB-MOD NAME                                       E
 54 PRINT TABLE SUMMARY                                          C
111 STATUS RULES.AND.ACTIONS

                                                             ------
STRATEGY

 1. START                                                    106251
 2. MODIFY PROGRAM - CASE AND ST
 5. TABLE TRACING
 6. NEW PROGRAM - CASE AND STGY

                                                             ------
```

Sub-module DRIVEN. Tables.

```
CLUSTER:  MODIFY PROGRAM - CASE & STGY                 RULES:  00000000011111111112
                                                               12345678901234567890

  6. INUNIT.EQ.BLANK                                           $$$$YN$$$$$$$$$$$$$$
  7. INUNIT.EQ.STRAT                                           $$$$$NY$$$$$$$YY$$$$
  8. INUNIT.EQ.CLUS                                            $$$$$N$Y$$$$Y$$$Y$$$
  9. INUNIT.EQ.PRED                                            $$$$$N$$Y$$$$Y$$$$Y$
 10. INUNIT.EQ.ACSHN                                           $$$$$N$$$Y$$$$$$$$$Y
 11. INUNIT.EQ.RULMTX                                          $$Y$$N$$$$Y$$$$$$Y$$
 12. INUNIT.EQ.ACTMTX                                          Y$$Y$N$$$$$Y$$$$$$$$
 13. INUNIT.EQ.END                                             $Y$$$N$$$$$$$$$$$$$$
 14. INUNO.NE.0                                                Y YY  NNNNNNYYYYYYYY
 15. ATABL(INUNO)                                              N YY        YYYNNNNN

STRATEGY                                                       --------------------

  1. START
  2. MODIFY PROGRAM - CASE AND ST                              71482222222222266666
  4. MODIFY PROGRAM - RULE MATRIX
  6. NEW PROGRAM - CASE AND STGY
  7. NEW UNIT - ACTION MATRIX
  8. MODIFY PROGRAM - ACTION MATR
                                                               --------------------
```

Sub-module DRIVEN. Tables.

```
CLUSTER:  MODIFY PROGRAM - CASE & STGY         RULES:  00000000011111111112
                                                       12345678901234567890

  4 NAME OF IN-UNIT?                                       A
  5 WRITE PDT TO DISK                                                   K
  6 UPDATE PDT VARIABLES                                                C
  7 CLEAR IN-UNIT NAME                                      B      CLC
  8 CLEAR COMMAND                                       B
  9 "END IN-UNIT"                                       A
 12 "ILLEGAL IN-UNIT"                                       A
 16 NON-BLANK CURRENT LINE                                 B CBBHCB
 18 IN-UNIT NO. = 0                                        CC      D D
 19 IN-UNIT NO. = 1                                              A
 20 IN-UNIT NO. = 2                                               A
 21 IN-UNIT NO. = 3                                          A
 22 IN-UNIT NO. = 4                                           A
 23 IN-UNIT NO. = 5                                            AA   E
 31 ZERO LINE NO.                                      C DD           CCC
 38 LOCATE IN-UNIT BLOCK                                            G
 39 ZERO NO. OF ENTRIES                                B EE           HBB
 40 IN-UNIT "EXISTS"                                         B
 46 "NOT IMPLEMENTED"                                              A A
 47 FETCH PDT VARIABLES                                            B B
 49 NEWOP = .T.                                                       D
 53 COMMAND = "NEW"                                    A              AAAAA
 56 EXECUTE IGETBL                                                  F
 69 MARK BLOCKS USED                                                I
 77 PRINT CLUS TITLES                                  E  G       C
 91 READ CLUS FROM DISK                                  HB
 93 BLANK OUT CLUSTER NAME                             D CC             D
 97 "SOME ENTRIES EXISTS"                                AA
 98 EXECUTE BLOCOD                                                  B
102 READ RULES FROM DISK                                 F
103 ZERO CLUSTER LINE NO.                                         B
107 READ AMTX FROM DISK                                   F
109 READ ATRANS FROM DISK                                           A
112 ACTION TRANS TABLE SIZE                                         D
113 WRITE ATRANS TO DISK                                            H

                                                       --------------------
STRATEGY

  1. START
  2. MODIFY PROGRAM - CASE AND ST                      71482222222222266666
  4. MODIFY PROGRAM - RULE MATRIX
  6. NEW PROGRAM - CASE AND STGY
  7. NEW UNIT - ACTION MATRIX
  8. MODIFY PROGRAM - ACTION MATR
```


Sub-module DRIVEN. Tables.

```
CLUSTER:  NEW UNIT - RULE MATRIX

16. NFLDS.EQ.0
17. NARPRD.EQ.0
18. NARPRD.EQ.NENTS
19. .NOT.ATABL(4)
20. CLUSTN.EQ.BLANK
21. CLUSTN.EQ.END
22. CLUSLN.EQ.0
23. LINUM.EQ.0
24. LINUM.GT.NACRLS
25. ICTAB.EQ.0
26. FLDCTR.EQ.0
27. FLDCTR.GT.NFLDS
28. APDTAB(100+FLDCTR).EQ.1
29. ALFRUL(FLDCTR).EQ.BLANK
30. ALFRUL(FLDCTR).EQ.YES.OR.ALFRUL(FLDCTR).EQ.NO
31. ALFRUL(FLDCTR).EQ.STAR.OR.ALFRUL(FLDCTR).EQ.DOLLR
32. INTRUL(FLDCTR).EQ.0
33. INTRUL(FLDCTR).GT.9.OR.INTRUL(FLDCTR).LT.0
34. NENTS.GT.LINUM
```

```
RULES:   0000000001111111111222
         1234567890123456789012

16.      YYYNNNNNNNNNNNNNNNNNNN
17.      YNNNNNNNNNNNNNNNNNNNNN
18.       NY
19.         YNNNNNNNNNNNNNNNNNNN
20.          YSNNNNNNNNNNNNNNNN
21.          SYNNNNNNNNNNNNNNNN
22.            YNNNNNNNNNNNNNNY
23.             NYYNNNNNNNNNNN
24.             YNNNNNNNNNNNNN
25.                YNNNNNNNNNN
26.              YN YNNNNNNNNN
27.                  YNNNNNNNN
28.                   YYYYNNNN
29.                   YSSNYN
30.                   SYSNSN
31.                   SSYNSN
32.                        YN
33.                       NNNY
34.            Y              N
         ----------------------

STRATEGY

 3. NEW UNIT - RULF MATRIX         3336363333333333333333
 6. NEW PROGRAM - CASE AND STGY
                                   ----------------------
```

Sub-module DRIVEN: Tables.

```
CLUSTER:  NEW UNIT - RULE MATRIX                     RULES:  0000000001111111111222
                                                             1234567890123456789012
  5 WRITE PDT TO DISK                                         EE  H
  6 UPDATE PDT VARIABLES                                      DD  G
  7 CLEAR IN-UNIT NAME                                          B F
 13 NO. OF PRED. GRPS                                         AA
 18 IN-UNIT NO. = 0                                             C I
 28 READ NENTS                                               B
 31 ZERO LINE NO.                                                C  C             C
 34 INCREMENT LINUM                                                A AEA C
 38 LOCATE IN-UNIT BLOCK                                          C
 40 IN-UNIT "EXISTS"                                                 FB
 47 FETCH PDT VARIABLES                                         D
 56 EXECUTE IGETBL                                                B
 58 WRITE RULES TO DISK                                           D
 63 EXPAND CLUS TABLE                                                B
 69 MARK BLOCKS USED                                              E
 70 "HOW MANY PREDS?"                                        A
 71 "HOW MANY GROUPS?"                                       C
 72 ALL PREDS INDEPENDENT                                      B
 73 READ SIZES OF GROUPS                                      B
 74 BUILD RULE FORMAT                                         CC
 75 "CLUS MUST EXIST"                                           A
 76 LIMIT = NO. CLUS.                                         FF D
 77 PRINT CLUS TITLES                                         II
 78 CHECK CLUS NAME                                                B
 79 "CLUSTER COMPLETE"                                              A C
 80 "ENTER RULES"                                                    C
 81 ZERO FIELD COUNTER                                        HH     DA  B
 82 INCREMENT FIELD COUNTER                                             B DDD EDD
 83 READ A RULE/ZERO CHAR. COUNTER                                      A
 84 CLEAR CLUSTER ENTRY(TEMP.)                                           A
 85 SET MASK BIT "OFF"                                                   A  A A
 86 ADJUST CHARACTER COUNTER                                              CCC DCC
 87 SET MASK BITS "ON"                                                     A   AA
 88 MAKE RULE ENTRY FROM LINE                                             BHH C
 89 SET RULE BITS = "NO"                                                       H
 90 SET RULE BITS = $ AND 1'Y'                                                  H
 91 READ CLUS FROM DISK                                         A
 92 FETCH RULE MEM'SHP INDICATOR                                    E B D
 93 BLANK OUT CLUSTER NAME                                    GG
 94 "CLUSTER NOT FOUND"                                                          A
 95 CLEAR INPUT RULE FIELD                                                 AH   A
 96 RULE TABLE SIZE                                               A
 99 NO. OF RULES IN THIS CLUSTER                                   CD
101 COUNT LIMIT = NO. CLUS RULES                                      D
116 IDENTIFY REQUIRED CLUSTER                                   B  B             B
```

Sub-module DRIVEN. Tables.

```
CLUSTER:  MODIFY PROGRAM - RULE MATRIX                    RULES:   0000
                                                                   1234

 26. FLDCTR.EQ.0                                                    YN
 34. NENTS.GT.LINUM                                                NY N
 37. NENTS.LT.1                                                    YNNN

STRATEGY                                                           ----

  3. NEW UNIT - RULE MATRIX
  4. MODIFY PROGRAM - RULE MATRIX                                  4443
  7. NEW UNIT - ACTION MATRIX
  8. MODIFY PROGRAM - ACTION MATR                                  8887

                                                                   ----


CLUSTER:  MODIFY PROGRAM - RULE MATRIX                    RULES:   0000
                                                                   1234

 31 ZERO LINE NO.                                                     C
 34 INCREMENT LINUM                                                AAH
 53 COMMAND = "NEW"                                                   A
 76 LIMIT = NO. CLUS.                                              H
 99 NO. OF RULES IN THIS CLUSTER                                   CHC
100 ZERO CLUS. ENTRIES FOR EXISTING RULFS                            A
116 IDENTIFY REQUIRED CLUSTER                                         H

                                                                   ----
STRATEGY

  3. NEW UNIT - RULE MATRIX
  4. MODIFY PROGRAM - RULE MATRIX                                  4443
  7. NEW UNIT - ACTION MATRIX
  8. MODIFY PROGRAM - ACTION MATR                                  8887

                                                                   ----
```

Sub-module DRIVEN. Tables.

```
CLUSTER: TABLE TRACING                                RULES:  00
                                                              12
35. CLINBL                                                    NY
                                                              --
STRATEGY

 1. START                                                     11
 5. TABLE TRACING                                             --


CLUSTER:  TABLE TRACING                               RULES:  00
                                                              12
  8 CLEAR COMMAND                                             HH
 42 READ IN PDT                                                A
 47 FETCH PDT VARIABLES                                        B
 91 READ CLUS FROM DISK                                        D
102 READ RULES FROM DISK                                       F
107 READ AMTX FROM DISK                                        C
108 "TRACE INFO DOESN'T EXIST"                                A
109 READ ATRANS FROM DISK                                      E
110 EXECUTE TRACE                                              G
                                                              --
STRATEGY

 1. START                                                     11
 5. TABLE TRACING                                             --
```

Sub-module DRIVEN. Tables.

```
CLUSTER:  NEW PROGRAM - CASE AND STGY     RULES:  0000000001111111111222222222223
                                                  123456789012345678901234567890

 6. INUNIT.EQ.BLANK                               $$$$$$$$$YN$$$$$$$$$$$$$$$$$$$
 7. INUNIT.EQ.STRAT                               $Y$$$$$$Y$$NY$$YYYY$YYYY$$$$$$
 8. INUNIT.EQ.CLUS                                $$$Y$$$$Y$N$Y$$$$$Y$$$$$Y$YY$$
 9. INUNIT.EQ.PRED                                $$$$$$$$$$N$$$$$$$$$$$$$$$$$YY
10. INUNIT.EQ.ACSHN                               $$$$$$$$$$N$$$$$$$$$$$$$$$$$$$
11. INUNIT.EQ.RULMTX                              $$Y$$$Y$$$N$$Y$$$$$$$$$$Y$$$$$
12. INUNIT.EQ.ACTMTX                              $$$$YY$$$$N$$$$$$$$$$$$$$Y$$$$
13. INUNIT.EQ.END                                 Y$$$$$$$$$N$$$$$$$$$$$$$$$$$$$
14. INUNO.NE.0                                     YYYYNYYY  NNNYYYYYYYYYYYYYYY
15. ATABL(INUNO)                                   YYYY N       NNNNNNNNNNNNNNYN
34. NENTS.GT.LINUM                                                YN        YN
35. CLINBL                                               YY     NNNNNNNNYYYYNN
36. NFEOF.EQ.0                                           YY             NNNN
37. NENTS.LT.1                                                  YYNNYNNN    NN
38. NEWOP                                                       YNNN YYY
39. RULNO.EQ.0                                                       NNY
40. RULNO.LT.0                                                       NNS
41. RULNO.GT.NASTRL                                                  NYN

                                                  ------------------------------
STRATEGY

 1. START
 2. MODIFY PROGRAM - CASE AND ST
 3. NEW UNIT - RULE MATRIX
 6. NEW PROGRAM - CASE AND STGY                   122227366666666666666666666622
 7. NEW UNIT - ACTION MATRIX

                                                  ------------------------------
CLUSTER:  NEW PROGRAM - CASE AND STGY     RULES:  0000000001111111111222222222223
                                                  123456789012345678901234567890

 4 NAME OF IN-UNIT?                                        A
 5 WRITE PDT TO DISK                                    EE                    KL
 6 UPDATE PDT VARIABLES                                 AA                    CD
 7 CLEAR IN-UNIT NAME                                   DD  B           CCCC  LM
 8 CLEAR COMMAND                                  B
 9 "END IN-UNIT"                                  A
10 NO-OP                                                A
11 "IN-UNIT EXISTS"                                BBBB
12 "ILLEGAL IN-UNIT"                                        A
15 EXPAND STRAT TABLE                                               B
16 NON-BLANK CURRENT LINE                              E   B BBB   B AA
17 "START OF STRATS?"                                                A
18 IN-UNIT NO. = 0                                     FF                 DDDD
19 IN-UNIT NO. = 1                                           A
20 IN-UNIT NO. = 2                                     A
21 IN-UNIT NO. = 3                                           A
22 IN-UNIT NO. = 4                                            A
```

Sub-module DRIVEN. Tables.

```
 23 IN-UNIT NO. = 5
 27 "HOW MANY ENTRIES?"
 28 READ NENTS
 29 "ENTRY ERROR"
 30 NO. OF STRATS.
 31 ZERO LINE NO.
 32 "ENTRY?"
 33 READ STRAT TITLE
 34 INCREMENT LINUM
 35 "STRAT/CLUS ENTRIES"
 36 STRAT. TABLE SIZE
 37 MAKE STRATEGY ENTRY
 38 LOCATE IN-UNIT BLOCK
 39 ZERO NO. OF ENTRIES
 40 IN-UNIT "EXISTS"
 43 PRINT STRAT TABLE
 44 READ STRAT ENTRY
 48 NEWOP = .F.
 49 NEWOP = .T.
 51 CLUS. TABLE SIZE
 52 COMMAND = "MOD"
 55 STRAT IN-UNIT LENGTH
 56 EXECUTE IGETBL
 57 WRITE STRAT TO DISK
 60 WRITE CLUS TO DISK
 61 "OUT OF D.A. SPACE"
 62 ZERO EOF FLAG
 64 PRINT CLUS TABLE
 65 CLUS IN-UNIT LENGTH
 66 READ CLUS ENTRY
 67 READ NCLUSR
 68 RESERVE NCLUSR RULES
 69 MARK BLOCKS USED
 77 PRINT CLUS TITLES
 93 BLANK OUT CLUSTER NAME
 98 EXECUTE BLDCOD
109 READ ATRANS FROM DISK
112 ACTION TRANS TABLE SIZE
113 WRITE ATRANS TO DISK
114 COMPOUND ACT NOS INTO PDT
115 INITIALISE INTERNAL ACT. TABLE

STRATEGY

 1. START
 2. MODIFY PROGRAM - CASE AND ST
 3. NEW UNIT - RULE MATRIX
 6. NEW PROGRAM - CASE AND STGY
 7. NEW UNIT - ACTION MATRIX
```

```
                            FF
              A   A
              BB  B
               A
               DC
            DD
                A         A
                C
                B         B
                 A
                    C
                    BD
                     E       DGH
            EE
     C      FF     E
                     B
                 D  CE
              C   C
     B      CC   C
                  D
   AAAA
                     C
                     D      CFG
       B
        B
                        AAAA
                        BBBB
                           A
                           B
                          C
                          D
                          E
        CC                    IJ
     F
     D
                             BC
                            A
                             DE
                             HI
                             A
                             B
------------------------------

1222736666666666666666666622
```

Sub-module DRIVEN. Tables.

```
CLUSTER:  NEW UNIT - ACTION MATRIX                  RULES:  0000000001
                                                            1234567890

19. .NOT.ATABL(4)                                           YNNNNNNNNN
20. CLUSTN.EQ.BLANK                                          YSNNNNNNN
21. CLUSTN.EQ.END                                            $YNNNNNNN
22. CLUSLN.EQ.0                                                YYNNNNN
23. LINUM.EQ.0                                                    YY
24. LINUM.GT.NACRLS                                              Y
25. ICTAB.EQ.0                                                      YN
26. FLDCTR.EQ.0                                                   YN
34. NENTS.GT.LINUM                                             YN

                                                            ----------
STRATEGY

 6. NEW PROGRAM - CASE AND STGY
 7. NEW UNIT - ACTION MATRIX                                6767777777

                                                            ----------
  5 WRITE PDT TO DISK                                         H
  6 UPDATE PDT VARIABLES                                      G
  7 CLEAR IN-UNIT NAME                                      B F
 18 IN-UNIT NO. = 0                                         C I
 31 ZERO LINE NO.                                            C  CC
 34 INCREMENT LINUM                                            A  AEAB
 38 LOCATE IN-UNIT BLOCK                                      C
 40 IN-UNIT "EXISTS"                                              FB
 47 FETCH PDT VARIABLES                                     D
 56 EXECUTE IGETBL                                            B
 59 WRITE AMTX TO DISK                                        D
 63 EXPAND CLUS TABLE                                             B
 69 MARK BLOCKS USED                                          E
 75 "CLUS MUST EXIST"                                       A
 76 LIMIT = NO. CLUS.                                        D
 78 CHECK CLUS NAME                                            B
 79 "CLUSTER COMPLETE"                                           A C
 81 ZERO FIELD COUNTER                                            DA
 91 READ CLUS FROM DISK                                      A
 92 FETCH RULE MEM'SHR INDICATOR                                  E HC
 94 "CLUSTER NOT FOUND"                                         A
 99 NO. OF RULES IN THIS CLUSTER                               C D
101 COUNT LIMIT = NO. CLUS RULES                                   D
104 ACTION TABLE SIZE                                         A
105 "LIST OF ACTION-NOS?"                                         C
106 PUT LIST IN ACTION TABLE                                         A
116 IDENTIFY REQUIRED CLUSTER                                B  BB
```

Sub-module DRIVEN. Tables.

```
CLUSTER:  MODIFY PROGRAM - ACTION MATX                    RULES:  0000
                                                                  1234
26. FLDCTR.EQ.0                                                    YN
34. NENTS.GT.LINUM                                                NY N
37. NENTS.LT.1                                                    YNNN

STRATEGY                                                          ----

 3. NEW UNIT - RULE MATRIX
 4. MODIFY PROGRAM - RULE MATRIX                                  4443
 7. NEW UNIT - ACTION MATRIX
 8. MODIFY PROGRAM - ACTION MATR                                  8887

                                                                  ----
CLUSTER:  MODIFY PROGRAM - ACTION MATX                    RULES:  0000
                                                                  1234
 31 ZERO LINE NO.                                                    C
 34 INCREMENT LINUM                                               AAB
 53 COMMAND = "NEW"                                                  A
 76 LIMIT = NO. CLUS.                                             B
 99 NO. OF RULES IN THIS CLUSTER                                  CBC
100 ZERO CLUS. ENTRIES FOR EXISTING RULES                           A
116 IDENTIFY REQUIRED CLUSTER                                        B

STRATEGY                                                          ----

 3. NEW UNIT - RULE MATRIX
 4. MODIFY PROGRAM - RULE MATRIX                                  4443
 7. NEW UNIT - ACTION MATRIX
 8. MODIFY PROGRAM - ACTION MATR                                  8887

                                                                  ----
```

Sub-module DRIVEN. Tables.

CLUSTER: INITIALISE	RULES: 0 1	0 2	0 3
1. CMND.EQ.END	Y	$	N
2. CMND.EQ.BLANK	Y	Y	N
3. CMND.EQ.NEW	Y	$	N
4. CMND.EQ.MOD	Y	$	N
5. CMND.EQ.TRACE	Y	$	N
STRATEGY	-	-	-
1. START			
6. NEW PROGRAM - CASE AND STGY			
9. INITIALISE	9	6	1
	-	-	-

CLUSTER: INITIALISE	RULES: 0 1	0 2	0 3
3 INITIALISE NEW PDT		E	
5 WRITE PDT TO DISK		G	
7 CLEAR IN-UNIT NAME		H	
8 CLEAR COMMAND			B
14 EXAMINE D.D. - NEW PROJECT?	A		
18 IN-UNIT NO. = 0		I	
25 SET UNIT NO = 1 FOR DRIVEN		A	
26 ENTER SUB-MOD. NAME IN D.D.		B	
41 D.D. BLK 3 TO DISK		D	
45 PRINT PROJECT TITLE BLOCK			A
47 FETCH PDT VARIABLES		J	
53 COMMAND = "NEW"		C	
54 PRINT TABLE SUMMARY		F	
STRATEGY	-	-	-
1. START			
6. NEW PROGRAM - CASE AND STGY			
9. INITIALISE	9	6	1
	-	-	-

Sub-module DRIVEN. Tables.

```
RELEASE 2.0              IGETBL              DATE = 80124          14/25/50

      SUBROUTINE IGETBL(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,   00106200
     + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,    00106300
     + MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)                                00106400
      INTEGER TABSIZ,CURULE,ACTIVE                                       00106500
      INTEGER PDTABL(TABSIZ),STGY(MAXRLS,MAXSTR),ACTNOS(MAXACT,MAXRLS)  00106600
      INTEGER BLANK,ATRANS(MACTS),AINTNO(MACTIG,2)                       00106700
      LOGICAL*1 TABL(8),RULES(MAXPRD,MAXRLS),MASKS(MAXPRD,MAXRLS)       00106800
      LOGICAL*1 VECTOR(MAXPRD)                                           00106900
      LOGICAL*1 INIT/.TRUE./,TRUE/.TRUE./                                00107000
      LOGICAL MATCH                                                      00107100
      DATA BLANK/' '/                                                    00107300
C APPLICATION SPECIFICATIONS********************************************00107400
C---GLOBAL VARIABLES--                                                   00107450
      INTEGER APDTAB,ZI411,INUNO,ZI412,NFEUF,ZI413,LGTH,IFREST,ZI414    00107455
      INTEGER LSTHDR,IBLPTR,IFRCNT,LIMPTR,NBLKRQ,NBLKOC,ZI415           00107460
      COMMON/I4/APDTAB(375),ZI411,INUNO,ZI412(6),NFEOF,ZI413(107),LGTH  00107465
      COMMON/I4/IFREST,ZI414(5538),LSTHDR,IBLPTR,IFRCNT,LIMPTR,NBLKRQ   00107470
      COMMON/I4/NBLKOC,ZI415(212)                                        00107475
C**********************************************************************00108400
      IF(.NOT.INIT) CALL ZGETBZ(VECTOR,NRPRDS)                           00108500
      INIT=.FALSE.                                                       00108600
      CURULE=0                                                           00108700
      DO 99999 I99999=1,NRULES                                           00108800
      IF(STGY(I99999,ACTIVE).EQ.BLANK) GO TO 99999                       00108900
      IF(.NOT.MATCH(RULES(1,I99999),MASKS(1,I99999),VECTOR,NRPRDS))     00109000
     +     GO TO 99999                                                   00109100
      CURULE=I99999                                                      00109200
      IF(STGY(CURULE,ACTIVE).EQ.0) INIT=.TRUE.                           00109300
      DO 99000 I99000=1,MAXACT                                           00109900
      NEXTAC=ACTNOS(I99000,CURULE)                                       00110000
      IF(NEXTAC.EQ.0) RETURN                                             00110100
      NXTACT=ATRANS(NEXTAC)                                              00110200
      DO 99998 I99998=1,MAXACT                                           00110300
      IF(NXTACT.EQ.0) GO TO 99000                                        00110400
```

Sub-module IGETBL

```
      NACSHN=AINTNO(NXTACT,1)
      GO TO (99997,99996,99996,99996,99996,99996,99991,99990,99989,99988
     +,99987,99986,99985,99984,99983,99982,99981,99980),NACSHN
C ACTION ENTRIES START HERE
C ACTION   1 INITIALISE VARIABLES
C
99997 CONTINUE
      NBLKRQ=(LGTH+1499)/1500
      LSTHDR=INUNO+20
      NBLKOC=APDTAB(LSTHDR)/256
      LISTRT=APDTAB(LSTHDR)-NBLKOC*256
      IBLPTR=31
      LIMPTR=71-NBLKRQ
      GO TO 99998
C ACTION 2-6 UNUSED......COMBINED IN ACTION 1
C
99996 CONTINUE
      GO TO 99998
C ACTION   7 INITIALISE COUNTER
C
99991 CONTINUE
      IFRCNT=1
      GO TO 99998
C ACTION   8 INCREMENT BLOCK POINTER
C
99990 CONTINUE
      IBLPTR=IBLPTR+IFRCNT
      GO TO 99998
C ACTION   9 MESSAGE - 'OUT OF SPACE'
C
99989 CONTINUE
 999  FORMAT(/' *** NO DIRECT ACCESS SPACE AVAILABLE.  TABLE NOT SAVED')
      WRITE(6,999)
      GO TO 99998
```

Sub-module IGETBL

```
RELEASE 2.0              IGETBL              DATE = 80124          14/25/50

C ACTION  10 NO-OP
C
99988 CONTINUE
      GO TO 99998
C ACTION  11 INCREMENT COUNTER
C
99987 CONTINUE
      IFRCNT=IFRCNT+1
      GO TO 99998
C ACTION  12 SET START-OF-FREE-SPACE TO BLOCK POINTER
C
99986 CONTINUE
      IFREST=IBLPTR
      GO TO 99998
C ACTION  13 SET START-OF-FREE-RECORD NUMBER FOR RETURN
C
99985 CONTINUE
      IFREST=IFREST-29
      GO TO 99998
C ACTION  14 SET OUT-OF-SPACE INDICATOR
C
99984 CONTINUE
      NFEOF=2
      GO TO 99998
C ACTION  15 SET START-OF-FREE-RECORD-NUMBER TO EXISTING FOR RETURN
C
99983 CONTINUE
      IFREST=LISTRT-29
      GO TO 99998
C ACTION  16 SET BLOCK "OCCUPIED"
C
99982 CONTINUE
      APDTAB(IBLPTR+IFRCNT-1)=1
      GO TO 99998
```

Sub-module IGETBL

```
C ACTION  17 SET BLOCK "VACANT"
C
99981 CONTINUE
      APDTAB(IBLPTR+IFRCNT-1)=0
C ACTION 18 SET LIST HEADER "VACANT"
      GO TO 99998
C
99980 CONTINUE
      APDTAB(LSTHDR)=0
C ACTION  19 SET LIST HEADER BACK TO ORIGINAL
C
99979 CONTINUE
      APDTAB(LSTHDR)=NBLKOC*256+LISTRT
      GO TO 99998
C END OF ACTIONS
99998 NXTACT=AINTND(NXTACT,2)
99000 CONTINUE
      RETURN
99999 CONTINUE
      WRITE(6,90001) (VECTOR(I),I=1,NRPRDS)
90001 FORMAT(/' VECTOR NOT MATCHED: ',50L1)
      RETURN
      END
```

Sub-module IGETBL

```
RELEASE 2.0              ZGETBZ              DATE = 80124          14/25/50

      SUBROUTINE ZGETBZ(V,NPREDS)
      LOGICAL*1 V(NPREDS)
C*****APPLICATION SPECIFICATIONS******************************************
C---GLOBAL VARIABLES--
      INTEGER APDTAB,ZI411,INUNO,ZI412,NFEOF,ZI413,LGTH,IFREST,ZI414
      INTEGER LSTHDR,IBLPTR,IFRCNT,LIMPTR,NBLKRQ,NBLKOC,ZI415
      COMMON/I4/APDTAB(375),ZI411,INUNO,ZI412(6),NFEOF,ZI413(107),LGTH
      COMMON/I4/IFREST,ZI414(5538),LSTHDR,IBLPTR,IFRCNT,LIMPTR,NBLKRQ
      COMMON/I4/NBLKOC,ZI415(212)
C************************************************************************
      V(1)=APDTAB(LSTHDR).EQ.0
      V(2)=APDTAB(IBLPTR+IFRCNT-1).EQ.0
      V(3)=IBLPTR.LE.LIMPTR
      V(4)=IFRCNT.LE.NBLKRQ
      V(5)=IFRCNT.EQ.1
      V(6)=NBLKRQ.EQ.NBLKOC
      V(7)=NBLKRQ.GT.NBLKOC
      V(8)=IFRCNT.LE.NBLKOC
      V(9)=NBLKOC.EQ.0
      RETURN
      END
```

Sub-module IGETBL

Sub-module IGETBL. Tables.


```
CLUSTER:  FIND FREE SPACE                          RULES:  0000000001111111
                                                           1234567890123456
 1. APDTAB(LSTHDR).EQ.0                                    YYYYYNNNNYYYNNNY
 2. APDTAB(IBLPTR+IFRCNT-1).EQ.0                           N YYY          Y
 3. IBLPTR.LE.LIMPTR                                       YNYYY    NNN   Y
 4. IFRCNT.LE.NBLKRQ                                         YN           Y
 5. IFRCNT.EQ.1                                              NNY YNNYNNYNNY
 6. NBLKRQ.EQ.NBLKOC                                            YSSS   NNNY
 7. NBLKRQ.GT.NBLKOC                                            SYYY   NNNS
 8. IFRCNT.LE.NBLKOC                                              YN YN YNY
 9. NBLKOC.EQ.0                                              Y        NNN  Y
                                                           ----------------

STRATEGY                                                   1010101111110111
                                                                          1
 1. FIND FREE SPACE                                        ----------------
 2. START

CLUSTER:  FIND FREE SPACE                          RULES:  0000000001111111
                                                           1234567890123456
 1 INITIALISE VARIABLES                                                   A
 7 INITIALISE COUNTER                                      B      B      BB
 8 INCREMENT BLOCK POINTER                                 A
 9 "OUT OF SPACE ON D.A. FILE"                              A      A
10 NO-OP                                                    C B B    A
11 INCREMENT COUNTER                                         A B AA CA AA
12 START-OF-FREE-SPACE = BLOCK POINTER                          A
13 SET START-OF-FREE-RECORD NUMBER                             A
14 SET OUT-OF-SPACE INDICATOR                              B       B
15 START-OF-FREE-RECORD NUMBER = EXISTING                      A
16 BLOCK = "OCCUPIED"                                              DB
17 BLOCK = "VACANT"                                            BB     BB
18 LIST HEADER = "VACANT"                                        A     A
19 LIST HEADER REVERTS TO ORIGINAL                                 E
                                                           ----------------
STRATEGY
                                                           1010101111110111
 1. FIND FREE SPACE                                                       1
 2. START                                                  ----------------
```

```
RELEASE 2.0              TRACE              DATE = 80124          14/25/50

      SUBROUTINE TRACE(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,      00121700
     + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,      00121800
     + MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)                                 00121900
      INTEGER TABSIZ,CURULE,ACTIVE                                        00122000
      INTEGER PDTABL(TABSIZ),STGY(MAXRLS,MAXSTR),ACTNOS(MAXACT,MAXRLS)    00122100
      INTEGER BLANK,ATRANS(MACTS),AINTNO(MACTIG,2)                        00122200
      LOGICAL*1 TABL(8),RULES(MAXPRD,MAXRLS),MASKS(MAXPRD,MAXRLS)         00122300
      LOGICAL*1 VECTOR(MAXPRD)                                            00122400
      LOGICAL*1 INIT/.TRUE./,TRUE/.TRUE./                                 00122500
      LOGICAL MATCH                                                       00122600
      DATA BLANK/' '/                                                     00122800
C APPLICATION SPECIFICATIONS*********************************************00122900
C---GLOBAL VARIABLES--                                                   00122902
      INTEGER APDTAB,CMND,ZI411,LINUM,RULNO,ZI412,NARPRD,ZI413,NACRLS     00122904
      INTEGER ZI414,AACTNO,ACASE,AATRNS,AAINTN,ACTIT,NACASE,LIMIT,NCLUSR00122906
      INTEGER NCL,MORE,KRCOND,JPRPTR,KONTIN,ZI415                         00122908
      LOGICAL*1 ZL111,TSTBIT,ZL112,ARULES,AMASKS,ZL113,LOGBLK,LOGV        00122910
      LOGICAL*1 LOGSTR,LOGA,ZL114                                         00122912
      COMMON/I4/APDTAB(375),CMND,ZI411(3),LINUM,RULNO,ZI412(4),NARPRD     00122914
      COMMON/I4/ZI413(2),NACRLS,ZI414(104),AACTNO(20,120),ACASE(120,20)   00122916
      COMMON/I4/AATRNS(120),AAINTN(150,3),ACTIT(8,20),NACASE,LIMIT        00122918
      COMMON/I4/NCLUSR,NCL,MORE,KRCOND,JPRPTR,KONTIN,ZI415(218)           00122920
      COMMON/L1/ZL111(5),TSTBIT,ZL112(2),ARULES(50,100),AMASKS(50,100)    00122922
      COMMON/L1/ZL113(3),LOGBLK,LOGV,LOGSTR,LOGA,ZL114(12)                00122924
C---LOCAL VARIABLES---                                                   00122926
      INTEGER CARD(20),TITLIN(15),                                        00123000
     +NBLNK/' '/,CEE/'C'/,STR(2),FMT1(52),FMT2(128),FMT3(52),             00123100
     +PRDFMT(128),T66FMT,T11FMT,I3FMT,X2FMT,A1FMT,ENDFMT,TFMT,X1FMT,      00123200
     +FIVA4,X2COM1,NCLR(20),LIST(120),X4FMT                               00123300
      LOGICAL*1 RULIN(64)                                                 00123500
      DATA T66FMT,T11FMT,I3FMT,X2FMT,A1FMT,X2COM1,X1FMT,FIVA4,ENDFMT      00124600
     +/'(T67','(T7','I3,','2X,','A1,','2X,1','1X,','5A4,','T1)'/          00124700
     +,X4FMT/'4X,'/                                                       00124800
      DATA PRDFMT/'(1X,','I2,','1H.,'/                                    00124900
```

Sub-module TRACE

```
RELEASE 2.0              TRACE             DATE = 80124          14/25/50

         DATA FMT1/' ',',I3,',',1X,'/                                             00125000
         DATA FMT2/' ',',2X,'/                                                    00125100
         DATA FMT3/' ',',2X,',',A1,',',3X,'/                                      00125200
C************************************************************************00125300
         IF(.NOT.INIT) CALL ZRACEZ(VECTOR,NRPRDS)                                 00125400
         INIT=.FALSE.                                                             00125500
         CURULE=0                                                                 00125600
         DO 99999 I99999=1,NRULES                                                 00125700
         IF(STGY(I99999,ACTIVE).EQ.BLANK) GO TO 99999                             00125800
         IF(.NOT.MATCH(RULES(1,I99999),MASKS(1,I99999),VECTOR,NRPRDS))            00125900
     +      GO TO 99999                                                           00126000
         CURULE=I99999                                                            00126100
         IF(STGY(CURULE,ACTIVE).EQ.0) INIT=.TRUE.                                 00126200
         DO 99000 I99000=1,MAXACT                                                 00126800
         NEXTAC=ACTNOS(I99000,CURULE)                                             00126900
         IF(NEXTAC.EQ.0) RETURN                                                   00127000
         NXTACT=ATRANS(NEXTAC)                                                    00127100
         DO 99998 I99998=1,MAXACT                                                 00127200
         IF(NXTACT.EQ.0) GO TO 99000                                              00127300
         NACSHN=AINTNO(NXTACT,1)                                                  00127400
         GO TO (99997,99996,99995,99994,99993,99992,99991,99990,99989,9998800127800
     +,99987,99986,99985,99984,99983,99982,99981,99980,99979,99978               00127900
     +,99977,99976,99975,99974,99973,99972,99971,99970,99969,99968               00128000
     +,99967,99966,99965,99964,99963,99962,99961,99960,99959,99958               00128100
     +,99957,99956,99955),NACSHN                                                  00128200
C ACTION ENTRIES START HERE                                                       00128300
C ACTION    1 BLANK OUT COMMAND                                                   00128400
C                                                                                 00128500
 99997 CONTINUE                                                                   00128600
         CMND=NBLNK                                                               00128700
         GO TO 99998                                                              00128800
C ACTION    2 ZERO CLUSTER NUMBER                                                 00128900
C                                                                                 00129000
 99996 CONTINUE                                                                   00129100
```

Sub-module TRACE

```
RELEASE 2.0                TRACE              DATE = 80124          14/25/50
      NCL=0
      GO TO 99998
C ACTION    3 READ AN INSTRUCTION
C
99995 CONTINUE
      WRITE(6,85)
 85   FORMAT(' TRACE')
      READ(9,99) CARD
      NCOL=1
      CALL STRING(CARD,NCOL,STR,MORE,LAST,8)
      CMND=STR(1)
 99   FORMAT(20A4)
      WRITE(6,84)
 84   FORMAT(/)
      GO TO 99998
C ACTION    4 FILL TITLE LINE WITH BLANKS
C
99994 CONTINUE
      DO 5 I=1,15
 5    TITLIN(I)=NBLNK
      GO TO 99998
C ACTION    5 ZERO RULE NUMBER
C
99993 CONTINUE
      RULNO=0
      GO TO 99998
C ACTION    6 ZERO LINE NUMBER
C
99992 CONTINUE
      LINUM=0
      GO TO 99998
C ACTION    7 ERROR MESSAGE - NO CLUSTER NUMBER
C
99991 CONTINUE.
```

Sub-module TRACE

```
       WRITE(6,98)
 98    FORMAT(/' ERROR - NO CLUSTERS QUALIFIED.  ENTER CLUSTER NUMBER')
       GO TO 99998
C ACTION   8 NO-OP
C
99990 CONTINUE
       GO TO 99998
C ACTION   9 READ CLUSTER NUMBER
C
99989 CONTINUE
       READ(9,*) NCL
       GO TO 99998
C ACTION  10 SET COMMAND= 'C' .(TO QUALIFY CLUSTER)
C
99988 CONTINUE
       CMND=CEE
       GO TO 99998
C ACTION  11 FIND WHICH CLUSTER
C
99987 CONTINUE
       CALL STRING(CARD,NCOL,STR,MORE,LAST,8)
       KEYNO=STR(1)
       GO TO 99998
C ACTION  12 CONVERT DISPLAY CODED NUMBER TO VALUE
C
99986 CONTINUE
       CALL CONVRT(KEYNO,NCL,KO)
       GO TO 99998
C ACTION  13 PROMPT
C
99985 CONTINUE
       WRITE(6,97)
 97    FORMAT(/' ENTER CLUSTER NUMBER')
       GO TO 99998
```

Sub-module TRACE

```
C ACTION  14 CLUSTER/RULE NUMBER ERROR MESSAGE
C
99984 CONTINUE
      WRITE(6,96) NCL
 96   FORMAT(/' NUMBER ',I5,' OUT OF RANGE.  RE-ENTER')
      GO TO 99998
C ACTION  15 SET POINTER TO START OF PREDICATES
C
99983 CONTINUE
      JPRPTR=APDTAB(254)
      GO TO 99998
C ACTION  16 ZERO CLUSTER RULE COUNTER
C
99982 CONTINUE
      NCLUSR=0
      GO TO 99998
C ACTION  17 INCREMENT RULE NUMBER
C
99981 CONTINUE
      RULNO=RULNO+1
      GO TO 99998
C ACTION  18 FIND STATUS OF CURRENT RULE FOR THIS CLUSTER
C
99980 CONTINUE
      KRCOND=ACASE(RULNO,NCL)
      GO TO 99998
C ACTION  19 SET LOOP LIMIT TO TOTAL NUMBER OF RULES
C
99979 CONTINUE
      LIMIT=NACRLS
      GO TO 99998
C ACTION  20 RECORD ACTIVE RULE NUMBER
C
99978 CONTINUE
```

```
      NCLUSR=NCLUSR+1
      LIST(NCLUSR)=RULNO
      GO TO 99998
C ACTION  21 SET LOOP LIMIT TO NUMBER OF ACTIVE RULES
C
99977 CONTINUE
      LIMIT=NCLUSR
      GO TO 99998
C ACTION  22 GET PREDICATE, POINT TO NEXT, RIGHT CORRECT.
C
99976 CONTINUE
      READ(3'JPRPTR,95) KONTIN,TITLIN,NXTPRD
 95   FORMAT(5X,A1,6X,15A4,8X,I4)
      JPRPTR=NXTPRD
      IF(TITLIN(15).NE.NBLNK) GO TO 20
      DO 15 I=1,14
        IF(TITLIN(15-I).EQ.NBLNK) GO TO 15
        J=15-I
        DO 10 K=1,J
          TITLIN(16-K)=TITLIN(J+1-K)
          TITLIN(J+1-K)=NBLNK
 10       CONTINUE
        GO TO 20
 15     CONTINUE
 20   CONTINUE
      GO TO 99998
C ACTION  23 INCREMENT LINE NUMBER COUNTER
C
99975 CONTINUE
      LINUM=LINUM+1
      GO TO 99998
C ACTION  24 SEE IF RULE ENTRY FOR THIS PREDICATE IS SIGNIFICANT
C
99974 CONTINUE
```

Sub-module TRACE

```
RELEASE 2.0              TRACE              DATE = 80124          14/25/50
      TSTBIT=AMASKS(LINUM,LIST(RULNO))
      GO TO 99998
C ACTION  25 SET KEYNO=CMND IN CASE IT'S A RULE NUMBER
C
99973 CONTINUE
      KEYNO=CMND
      GO TO 99998
C ACTION  26 MOVE RULE ENTRY INTO PRINT LINE
C
99972 CONTINUE
      RULIN(RULNO)=ARULES(LINUM,LIST(RULNO))
      GO TO 99998
C ACTION  27 PRINT PREDICATE CONTINUATION
C
99971 CONTINUE
      WRITE(6,94) TITLIN
 94   FORMAT(6X,15A4)
      GO TO 99998
C ACTION  28 PRINT PREDICATE AND-RULE ENTRIES
C
99970 CONTINUE
      WRITE(6,PRDFMT) LINUM,TITLIN,(RULIN(I),I=1,NCLUSR)
      GO TO 99998
C ACTION  29 GET CURRENT EXTERNAL ACTION NUMBER
C
99969 CONTINUE
      LINUM=AACTNO(RULNO,LIST(NCL))
      GO TO 99998
C ACTION  30 PRINT ACTION HEADER
C
99968 CONTINUE
      IFLPTR=AAINTN(AATRNS(LINUM),3)
      NFS=IFLPTR/10000
      NREC=IFLPTR-NFS*10000
```

Sub-module TRACE

```
RELEASE 2.0              TRACE              DATE = 80124          14/25/50

      READ(NFS'NREC,92) TITLIN
      WRITE(6,91) LINUM,TITLIN
 92   FORMAT(12X,15A4)
 91   FORMAT(1X,I2,'.',1X,15A4)
      GO TO 99998
C ACTION  31 CLEAR LINE FOR RULE ENTRIES
C
99967 CONTINUE
      DO 25 I=1,64
 25     RULIN(I)=LOGBLK
      GO TO 99998
C ACTION  32 PRINT CLUSTER BOTTOM LINE
C
99966 CONTINUE
      WRITE(6,84)
      J=ISC+4
      IF(IFC.GE.J) WRITE(6,FMT3) LOGA,(LOGA,I=J,IFC,5)
      IF(IFC.LT.J) WRITE(6,FMT3) LOGA
      WRITE(6,FMT2) (LOGSTR,I=ISC,IFC)
      GO TO 99998
C ACTION  33 UNUSED
C
99965 CONTINUE
      GO TO 99998
C ACTION  34 SET UP AND PRINT CLUSTER HEADINGS
C
99964 CONTINUE
      IF1=4
      IF2=3
      IF3=5
      NUMCOL=-2
      ISTRT=IFIN+1
      ISC=1
      IFC=0
```

Sub-module TRACE

```
RELEASE 2.0              TRACE              DATE = 80124          14/25/50

          FMT1(1)=TFMT
          FMT2(1)=TFMT
          FMT3(1)=TFMT
          DO 35 I=ISTRT,NCLS
             IF(NUMCOL+NCLR(I).GT.LL-2) GO TO 40
             NUMCOL=NUMCOL+NCLR(I)+2
             IFIN=I
             L=IFC+1
             IFC=IFC+NCLR(I)
             DO 30 J=L,IFC
                FMT2(IF2)=A1FMT
                IF2=IF2+1
                IF(J/5*5.NE.J) GO TO 30
                   FMT1(IF1+1)=X2FMT
                   FMT1(IF1)=I3FMT
                   IF1=IF1+2
                FMT3(IF3)=A1FMT
                FMT3(IF3+1)=X4FMT
                IF3=IF3+2
   30           CONTINUE
          FMT3(IF3)=X2FMT
          IF3=IF3+1
          FMT3(IF3)=ENDFMT
             FMT2(IF2)=X2FMT
             IF2=IF2+1
             FMT2(IF2)=ENDFMT
             FMT1(IF1)=X2FMT
             IF1=IF1+1
             FMT1(IF1)=ENDFMT
   35        CONTINUE
   40     WRITE(6,FMT2) (LOGSTR,I=ISC,IFC)
          IF(IFC-ISC.GE.4) GO TO 37
             WRITE(6,FMT1) ISC
             WRITE(6,FMT3) LOGV
```

Sub-module TRACE

```
RELEASE 2.0              TRACE              DATE = 80124          14/25/50
            GO TO 42
 37         J=ISC+4
            WRITE(6,FMT1) ISC,(I,I=J,IFC,5)
            WRITE(6,FMT3) LOGV,(LOGV,I=J,IFC,5)
 42      WRITE(6,89)
 89      FORMAT(' PRED.')
         KONTIN=0
         DO 45 I=6,127
 45         PRDFMT(I)=FMT2(I-4)
         GO TO 99998
C ACTION  35 PRINT RULE LINE
C
99963 CONTINUE
         WRITE(6,PRDFMT) LINUM,(RULIN(I),I=1,NCLUSR)
         GO TO 99998
C ACTION  36 INCREMENT NO OF CLUSTERS NCLS
C
99962 CONTINUE
         NCLS=NCLS+1
         GO TO 99998
C ACTION  37 ZERO NO OF RULES FOR CURRENT CLUSTER  NCLR(NCLS)
C
99961 CONTINUE
         NCLR(NCLS)=0
         GO TO 99998
C ACTION  38 ZERO NO OF CLUSTERS NCLS
C
99960 CONTINUE
         NCLS=0
         GO TO 99998
C ACTION  39 INCREMENT NO OF RULES IN CURRENT CLUSTER
C
99959 CONTINUE
         NCLR(NCLS)=NCLR(NCLS)+1
```

Sub-module TRACE

```
      GO TO 99998
C ACTION  40 SELECT PIECES OF FORMAT FOR LONG PRINT LINE
C
99958 CONTINUE
      TFMT=T66FMT
      LL=64
      PRDFMT(4)=X2COM1
      PRDFMT(5)=FIVA4
      GO TO 99998
C ACTION  41 SELECT PIECES OF FORMAT FOR SHORT PRINT LINE
C
99957 CONTINUE
      TFMT=T11FMT
      LL=121
      PRDFMT(4)=X1FMT
      PRDFMT(5)=X1FMT
      GO TO 99998
C ACTION  42 ZERO ENDING CLUSTER NO  IFIN
C
99956 CONTINUE
      IFIN=0
      GO TO 99998
C ACTION  43 PRINT CLUSTER TITLES
C
99955 CONTINUE
      WRITE(6,83) NCL,(ACTIT(K,NCL),K=1,7)
 83   FORMAT(' CLUSTER:',I3,'. ',7A4)
      GO TO 99998
C END OF ACTIONS
99998 NXTACT=AINTNO(NXTACT,2)
99000 CONTINUE
      RETURN
99999 CONTINUE
      WRITE(6,90001) (VECTOR(I),I=1,NRPRDS)
```

Sub-module TRACE

```
RELEASE 2.0                TRACE               DATE = 80124          14/25/50

 90001 FORMAT(/' VECTOR NOT MATCHED: ',50L1)                           00159800
       RETURN                                                          00159900
       END                                                             00160000
```

Sub-module TRACE

RELEASE 2.0 ZRACEZ DATE = 80124 14/25/50

```
      SUBROUTINE ZRACEZ(V,NPREDS)
      LOGICAL*1 V(NPREDS)
C APPLICATION SPECIFICATIONS********************************************
C---GLOBAL VARIABLES--
      INTEGER APDTAB,CMND,ZI411,LINUM,RULNO,ZI412,NARPRD,ZI413,NACRLS
      INTEGER ZI414,AACTNO,ACASE,AATRNS,AAINTN,ACTIT,NACASE,LIMIT,NCLUSR
      INTEGER NCL,MORE,KRCOND,JPRPTR,KONTIN,ZI415
      LOGICAL*1 ZL111,TSTBIT,ZL112,ARULES,AMASKS,ZL113,LOGBLK,LOGV
      LOGICAL*1 LOGSTR,LOGA,ZL114
      COMMON/I4/APDTAB(375),CMND,ZI411(3),LINUM,RULNO,ZI412(4),NARPRD
      COMMON/I4/ZI413(2),NACRLS,ZI414(104),AACTNO(20,120),ACASE(120,20)
      COMMON/I4/AATRNS(120),AAINTN(150,3),ACTIT(8,20),NACASE,LIMIT
      COMMON/I4/NCLUSR,NCL,MORE,KRCOND,JPRPTR,KONTIN,ZI415(218)
      COMMON/L1/ZL111(5),TSTBIT,ZL112(2),ARULES(50,100),AMASKS(50,100)
      COMMON/L1/ZL113(3),LOGBLK,LOGV,LOGSTR,LOGA,ZL114(12)
C---LOCAL VARIABLES---
      INTEGER CEE/'C'/,END/'END'/,IPLUS/'+'/
C**********************************************************************
      V(1)= CMND.EQ.CEE
      V(2)= CMND.EQ.END
      V(3)= NCL.LE.0
      V(4)= NCL.GT.NACASE
      V(5)= MORE.EQ.0
      V(6)= RULNO.EQ.0
      V(7)= RULNO.LE.LIMIT
      V(8)= KRCOND.EQ.1
      V(9)= JPRPTR.EQ.0
      V(10)=LINUM.EQ.0
      V(11)=TSTBIT
      V(12)=KONTIN.EQ.IPLUS
      V(13)=LINUM.LT.NARPRD
      V(14)=NCL.GT.NCLUSR
      RETURN
      END
```

Sub-module TRACE

```
CLUSTER:  START                                          RULES:  0
                                                                 1
  1. CMND.EQ.CEE                                                 Y
  2. CMND.EQ.END                                                 Y
  3. NCL.LE.0                                                    Y
  4. NCL.GT.NACASE                                               Y
  5. MORE.EQ.0                                                   Y
  6. RULNO.EQ.0                                                  Y
  7. RULNO.LE.LIMIT                                              Y
  8. KRCOND.EQ.1                                                 Y
  9. JPRPTR.EQ.0                                                 Y
 10. LINUM.EQ.0                                                  Y
 11. TSTBIT                                                      Y
 12. KONTIN.EQ.IPLUS                                             Y
 13. LINUM.LT.NARPRD                                             Y
 14. NCL.GT.NCLUSR                                               Y

STRATEGY                                                         -

 1. START                                                        2
 2. SWITCH CLUSTER                                               -


CLUSTER:  START                                          RULES:  0
                                                                 1
  1 BLANK COMMAND                                                A
  2 ZERO CLUSTER NUMBER                                          B
  3 READ AN INSTRUCTION                                          C
  4 BLANK TITLE LINE                                             D
  5 ZERO RULE NUMBER                                             E
  6 ZERO LINE NUMBER                                             F
 15 POINT TO START OF PREDICATES                                 G

STRATEGY                                                         -

 1. START                                                        2
 2. SWITCH CLUSTER                                               -
```

Sub-module TRACE. Tables.

```
CLUSTER:  SWITCH CLUSTER                                        RULES:  0000000
                                                                        1234567
 1. CMND.EQ.CEE                                                         $NYYYYY
 2. CMND.EQ.END                                                         YN$$$$$
 3. NCL.LE.0                                                              YY NN
 4. NCL.GT.NACASE                                                           YNN
 5. MORE.EQ.0                                                             Y N
 9. JPRPTR.EQ.0                                                              NY

STRATEGY                                                                -------

 2. SWITCH CLUSTER                                                      0222233
 3. PRINT LINE?
                                                                        -------
CLUSTER:  SWITCH CLUSTER                                        RULES:  0000000
                                                                        1234567
  7 "NO CLUSTERS QUALIFIED"                                              A
  8 NO-OP                                                               A
  9 READ CLUSTER NUMBER                                                  B BB
 10 SET COMMAND = 'C'                                                    C
 11 EXTRACT CLUSTER NUMBER FROM CARD                                      A
 12 CONVERT EBCDIC TO INTEGER                                             B
 13 "ENTER CLUSTER NUMBER"                                                 A
 14 "NUMBER OUT OF RANGE"                                                   A
 16 ZERO RULE COUNTER                                                     CC
 17 INCREMENT RULE NUMBER                                                    AA
 18 STATUS OF CURRENT RULE                                                   BB
 19 SET LIMIT = TOTAL NO. OF RULES                                           CC
 36 INCREMENT CLUSTER COUNTER                                                EE
 37 ZERO NO. RULES FOR CURRENT CLUSTER                                       FF
 38 ZERO CLUSTER COUNTER                                                  DD
 40 SELECT LONG LINE FORMAT                                                  D
 41 SELECT SHORT LINE FORMAT                                                  D
 42 ZERO ENDING CLUSTER NO.                                               EE
 43 PRINT CLUSTER TITLES                                                     GG
                                                                        -------
STRATEGY

 2. SWITCH CLUSTER                                                      0222233
 3. PRINT LINE?
                                                                        -------
```

Sub-module TRACE. Tables.

```
CLUSTER:  PRINT LINE?                                        RULES:  00000000
                                                                     12345678
  5. MORE.EQ.0                                                         NNNNNY
  7. RULNO.LE.LIMIT                                                  YYNNYNYN
  8. KRCOND.EQ.1                                                     NY
  9. JPRPTR.EQ.0                                                       YN
 10. LINUM.EQ.0                                                      YYYYNNN
 11. TSTBIT                                                              N Y

STRATEGY                                                             --------

  2. SWITCH CLUSTER
  3. PRINT LINE?                                                     33333452
  4. NOPRINT
  5. YESPRINT
                                                                     --------
CLUSTER:  PRINT LINE?                                        RULES:  00000000
                                                                     12345678
   5 ZERO RULE NUMBER                                                  DD  AC
   8 NO-OP                                                                A
  11 EXTRACT CLUSTER NUMBER FROM CARD                                       A
  12 CONVERT EBCDIC TO INTEGER                                              B
  17 INCREMENT RULE NUMBER                                           AHEFA B
  18 STATUS OF CURRENT RULE                                          BC
  20 RECORD ACTIVE RULE NUMBER                                        A
  21 SET LIMIT = NO. OF ACTIVE RULES                                   CC
  22 GET PREDICATE, POINT TO NEXT                                       E
  23 INCREMENT LINE NUMBER                                             FG
  24 FETCH MASK BIT FOR CURRENT PRED/RULE                              GHB D
  31 CLEAR RULE ENTRY LINE                                             AA  C
  34 PRINT CLUSTER HEADINGS                                            BB
  39 INCREMENT CURRENT CLUS. RULE CTR.

STRATEGY                                                             --------

  2. SWITCH CLUSTER
  3. PRINT LINE?                                                     33333452
  4. NOPRINT
  5. YESPRINT
                                                                     --------
```


Sub-module TRACE. Tables.

```
CLUSTER:  NOPRINT                                        RULES:   00000
                                                                  12345
 6. RULNO.EQ.0                                                    N NYY
 9. JPRPTR.EQ.0                                                   Y NNN
12. KONTIN.EQ.IPLUS                                               N NNY
13. LINUM.LT.NARPRD                                               YN

STRATEGY                                                          -----

 3. PRINT LINE?
 4. NOPRINT                                                       36434
 6. TRACE
                                                                  -----


CLUSTER:  NOPRINT                                        RULES:   00000
                                                                  12345
  3 READ AN INSTRUCTION                                            B
  5 ZERO RULE NUMBER                                              BFB
  6 ZERO LINE NUMBER                                               E
 12 CONVERT EBCDIC TO INTEGER                                      D
 22 GET PREDICATE, POINT TO NEXT                                    A A
 23 INCREMENT LINE NUMBER                                         A  A
 25 SET KEYNO = CMND (RULE NO.?)                                   C
 32 PRINT CLUSTER BOTTOM LINE                                      A
                                                                  -----
STRATEGY

 3. PRINT LINE?
 4. NOPRINT                                                       36434
 6. TRACE
                                                                  -----
```

Sub-module TRACE. Tables.

CLUSTER: YESPRINT

	RULES: 1	2	3	4	5	6	7
	0	0	0	0	0	0	0
6. RULNO.EQ.0	N	N	N	Y	Y	N	
7. RULNO.LE.LIMIT	Y	N	N			N	
9. JPRPTR.EQ.0			N			Y	
12. KONTIN.EQ.IPLUS		Y	N	Y	N	N	
13. LINUM.LT.NARPRD							N
STRATEGY							
3. PRINT LINE?							
5. YESPRINT	5	5	5	5	3	5	6
6. TRACE							

CLUSTER: YESPRINT

	RULES: 1	2	3	4	5	6	7
	0	0	0	0	0	0	0
3 READ AN INSTRUCTION							A
5 ZERO RULE NUMBER			C			B	
6 ZERO LINE NUMBER							D
12 CONVERT EBCDIC TO INTEGER							C
17 INCREMENT RULE NUMBER	B				B		
22 GET PREDICATE, POINT TO NEXT		B	B	B			
23 INCREMENT LINE NUMBER					A		
24 FETCH MASK BIT FOR CURRENT PRED/RULE					C		
25 SET KEYNO = CMND (RULE NO.?)							B
26 RULE ENTRY INTO PRINT LINE	A						
27 PRINT PRED. CONTINUATION		A		A			
28 PRINT PRED & RULE ENTRIES			A				
35 PRINT RULE LINE						A	
STRATEGY							
3. PRINT LINE?							
5. YESPRINT	5	5	5	5	3	5	6
6. TRACE							

Sub-module TRACE. Tables.

```
CLUSTER:  TRACE                                  RULES:  0000000
                                                         1234567

  1. CMND.EQ.CEE                                         Y$
  2. CMND.EQ.END                                         $Y
  3. NCL.LE.0                                              Y NNN
  6. RULNO.EQ.0                                              YNN
 10. LINUM.EQ.0                                               YN
 14. NCL.GT.NCLUSR                                         YN

STRATEGY                                                 -------

  2. SWITCH CLUSTER
  6. TRACE                                               2066666

                                                         -------

CLUSTER:  TRACE                                  RULES:  0000000
                                                         1234567

  2 ZERO CLUSTER NUMBER                                  A
  3 READ AN INSTRUCTION                                       A
  4 BLANK TITLE LINE                                     B
  5 ZERO RULE NUMBER                                     C    D
  6 ZERO LINE NUMBER                                     D    E
  8 NO-OP                                                 A
  9 READ CLUSTER NUMBER                                    BB
 12 CONVERT EBCDIC TO INTEGER                                 C
 14 "NUMBER OUT OF RANGE"                                  AA
 15 POINT TO START OF PREDICATES                         E
 17 INCREMENT RULE NUMBER                                    A B
 25 SET KEYNO = CMND (RULE NO.?)                              B
 29 CURRENT EXTERN. ACT. NO.                                 B C
 30 PRINT ACTION HEADER                                        A

STRATEGY                                                 -------

  2. SWITCH CLUSTER
  6. TRACE                                               2066666

                                                         -------
```

Sub-module TRACE. Tables.

```
RELEASE 2.0              BLDCOD              DATE = 80124         14/25/50

      SUBROUTINE BLDCOD(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
     + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,
     + MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
      INTEGER TABSIZ,CURULE
      INTEGER PDTABL(TABSIZ),STGY(MAXRLS,MAXSTR),ACTNOS(MAXACT,MAXRLS),
     +  ATRANS(MACTS),AINTNO(MACTIG,2),BLANK,ACTIVE
      LOGICAL*1 TABL(8),RULES(MAXPRD,MAXRLS),VECTOR(MAXPRD),
     +   MASKS(MAXPRD,MAXRLS)
      LOGICAL*1 INIT/.TRUE./,TRUE/.TRUE./
      LOGICAL MATCH
      DATA BLANK/' '/
C APPLICATION SPECIFICATIONS*******************************************
C---GLOBAL VARIABLES
      INTEGER APDTAB,ZI411,LINUM,ZI412,INUNIT,ZI413,AATRNS,AAINTN,ZI414
      INTEGER LIMIT,ZI415,MORE,ZI416,NAPRDS,NAACTS,NAACTG,NPR,NCOL
      INTEGER NOCODE,NF3REC,IVAR,TYPID,TYPNO,IACTN,JVAR,IBLK1,IBLK3
      INTEGER NAIACS,ZI417
      LOGICAL*1 ZL111,CMPND,INITDD,STVARL,LOGBLK,ZL112,ATAB,ZL113
      COMMON/I4/APDTAB(375),ZI411(4),LINUM,ZI412(2),INUNIT,ZI413(4910)
      COMMON/I4/AATRNS(120),AAINTN(150,3),ZI414(161),LIMIT,ZI415(2)
      COMMON/I4/MORE,ZI416(9),NAPRDS,NAACTS,NAACTG,NPR,NCOL,NOCODE
      COMMON/I4/NF3REC,IVAR,TYPID,TYPNO,IACTN,JVAR,IBLK1(100),IBLK3(100)
      COMMON/I4/NAIACS,ZI417(9)
      COMMON/L1/ZL111(10011),LOGBLK,ZL112(4),CMPND,INITDD,STVARL,ATAB(8)
      COMMON/L1/ZL113
C---LOCAL VARIABLES---
      INTEGER LINE(21),CARD(20),CHARST(4),GLOB,CACT(2),CNPR,UNPAK
      INTEGER VARNAM(2),OLDFRE,FREEAD,CODE,ACTS,PRED,CNUMS(15)
      LOGICAL*1 LLINE(84),LCARD(80),LBLNK
      DATA GLOB,CODE,ACTS,PRED/'GLOB','CODE','ACTS','PRED'/,NBLNK/' '/
      DATA CACT/'C AC','TION'/,CNPR/' NPR'/
      EQUIVALENCE (LINE(1),LLINE(1)),(CARD(1),LCARD(1)),(NBLNK,LBLNK)
C *********************************************************************
      IF(.NOT.INIT) CALL ZLDCOZ(VECTOR,NRPRDS)
```

Sub-module BLDCOD

```
      INIT=.FALSE.
      CURULE=0
      DO 99999 I99999=1,NRULES
      IF(STGY(I99999,ACTIVE).EQ.BLANK) GO TO 99999
      IF(.NOT.MATCH(RULES(1,I99999),MASKS(1,I99999),VECTOR,NRPRDS))
     +     GO TO 99999
      CURULE=I99999
      IF(STGY(CURULE,ACTIVE).EQ.0) INIT=.TRUE.
      DO 99000 I99000=1,MAXACT
      NEXTAC=ACTNOS(I99000,CURULE)
      IF(NEXTAC.EQ.0) RETURN
      NXTACT=ATRANS(NEXTAC)
      DO 99998 I99998=1,MAXACT
      IF(NXTACT.EQ.0) GO TO 99000
      NACSHN=AINTNO(NXTACT,1)
      GO TO (99997,99996,99995,99994,99993,99992,99991,99990,99989,99988
     +,99987,99986,99985,99984,99983,99982,99981,99980,99979,99978
     +,99977,99976,99975,99974,99973,99972,99971,99970,99969,99968
     +,99967,99966,99965,99964,99963,99962,99961,99960,99959,99958
     +,99957,99956,99955,99954,99953,99952,99951,99950,99949,99948
     +,99947,99946,99945,99944,99943,99942,99941,99940,99939,99938
     +,99937,99936,99935,99934,99933,99932,99931,99930,99929,99928
     +,99927,99926,99925,99924,99923,99922,99921,99920,99919,99918
     +,99917,99916,99915,99914,99913,99912,99911),NACSHN
C ACTION ENTRIES START HERE
C ACTION  1 PROMPT FOR CODE TYPE
C
99997 CONTINUE
      WRITE(6,99)
 99   FORMAT(/' TYPE')
      GO TO 99998
C ACTION  2 READ TYPE IDENTIFIER
C
99996 CONTINUE
```

Sub-module BLDCOD

```
      READ(9,98) INUNIT
 98   FORMAT(A4)
      GO TO 99998
C ACTION  3 NO-OP
C
99995 CONTINUE
      GO TO 99998
C ACTION  4 READ DATA DICT TYPE LIST FROM UNIT 8
C
99994 CONTINUE
      READ(8'1) IBLK1
      INITDD=IBLK1(1).EQ.NBLNK
      GO TO 99998
C ACTION  5 INITIALISE TYPE LIST FOR UNIT 8
C
99993 CONTINUE
      CALL DDINIT(IBLK1)
      GO TO 99998
C ACTION  6 INITIALISE SOURCE CODE FILE 3
C
99992 CONTINUE
      NF3R=NF3REC-1
      DO 10 I=1,NF3R
        J=I+1
 10     WRITE(3'I,82) J
 82     FORMAT(80X,I4)
      J=0
      WRITE(3'NF3REC,82) J
      APDTAB(258)=1
      GO TO 99998
C ACTION  7 PROMPT FOR INPUT
C
99991 CONTINUE
      WRITE(6,97)
```

Sub-module BLDCOD

```
 97   FORMAT(/' INPUT')
      GO TO 99998
C ACTION  8 READ A CARD
C
99990 CONTINUE
      READ(9,96) CARD
 96   FORMAT(20A4)
      GO TO 99998
C ACTION  9 ISOLATE A CHAR STRING
C
99989 CONTINUE
      LAST=0
      CALL STRING(CARD,NCOL,CHARST,MORE,LAST,12)
      GO TO 99998
C ACTION 10 CONVERT DISPLAY CODED CHARS TO VALUE
C
99988 CONTINUE
      CALL CONVRT(CHARST,NPR,IOK)
      GO TO 99998
C ACTION 11 SUB-COMMAND PROMPT
C
99987 CONTINUE
      WRITE(6,95)
 95   FORMAT(/' ACTION')
      GO TO 99998
C ACTION 12 ZERO COLUMN NO.
C
99986 CONTINUE
      NCOL=0
      GO TO 99998
C ACTION 13 ZERO LINE NUMBER
C
99985 CONTINUE
      LINUM=0
```

Sub-module BLDCOD

```
RELEASE 2.0              BLDCOD              DATE = 80124           14/25/50
        GO TO 99998
C ACTION 14 ESTABLISH START OF TABLE'S GLOBAL VARIABLE LIST ON UNIT 8
C
99984 CONTINUE
      LISTRT=APDTAB(255)
      GO TO 99998
C ACTION 15 PREDICATE OUT OF RANGE MSGE
C
99983 CONTINUE
      WRITE(6,94) NPR
 94   FORMAT(' ILLEGAL PREDICATE NUMBER ',I10,'.  RETYPE LINE')
      GO TO 99998
C ACTION 16 RESERVE NRECS NEW RECORDS ON UNIT 3
C
99982 CONTINUE
      LFREE=APDTAB(258)
      NPTR=LFREE
      DO 15 I=1,NRECS
        NOCODE=NPTR
        IF(NOCODE.EQ.0) GO TO 20
 15     READ(3'NOCODE,82) NPTR
      APDTAB(258)=NPTR
      GO TO 25
 20   WRITE(6,93)
 93   FORMAT(' OUT OF SPACE ON SOURCE FILE - UNIT 3')
      NPR=0
      MORE=1
 25   CONTINUE
      GO TO 99998
C ACTION 17 UPDATE POINTER TO HEAD OF PRED. LIST
C
99981 CONTINUE
      APDTAB(254)=LFREE
      GO TO 99998
```

Sub-module BLDCOD

```
C ACTION 18 UPDATE NO. OF PREDS.
C
99980 CONTINUE
      NAPRDS=NPR
      GO TO 99998
C ACTION 19 TRACE TO END OF PRED LIST  (NUMREC RECORDS)
C
99979 CONTINUE
      NPTR=APDTAB(254)
      DO 30 I=1,NUMREC
        IPTR=NPTR
        READ(3'IPTR,92) LINE
 30     NPTR=LINE(21)
 92   FORMAT(20A4,I4)
      GO TO 99998
C ACTION 20 SET CURRENT LINE POINTER TO START OF NEW GROUP OF PREDS
C
99978 CONTINUE
      LINE(21)=LFREE
      GO TO 99998
C ACTION 21 FILL LINE FROM CARD
C
99977 CONTINUE
      JCOL=58
      LLINE(13)=LBLNK
      IF(NCOL.GT.14) JCOL=72-NCOL
      KDIF=NCOL-1
      DO 35 I=1,JCOL
 35     LLINE(13+I)=LCARD(I+KDIF)
      JCOL=JCOL+1
      DO 40 I=JCOL,80
 40     LLINE(I)=LBLNK
      GO TO 99998
C ACTION 22 WRITE LINE TO UNIT 3
```

Sub-module BLDCOD

```
RELEASE 2.0              BLDCOD              DATE = 80124         14/25/50
C
99976 CONTINUE
      READ(3'IPTR,82) NPTR
      WRITE(3'IPTR,92) LINE
      IPTR=NPTR
      GO TO 99998
C ACTION 23 BLANK OUT INUNIT
C
99975 CONTINUE
      INUNIT=NBLNK
      GO TO 99998
C ACTION 24 MARK THIS LINE AS END OF GROUP
C
99974 CONTINUE
      LINE(21)=0
      GO TO 99998
C ACTION 25 SET NO. OF UNIT-3 RECORDS TO BE FOLLOWED (NUMREC=NPR)
C
99973 CONTINUE
      NUMREC=NPR
      GO TO 99998
C ACTION 26 ZERO NEW BLOCK OF AATRNS
C
99972 CONTINUE
      JSTRT=NAACTS+1
      DO 45 I=JSTRT,NPR
 45     AATRNS(I)=0
      GO TO 99998
C ACTION 27 FIND NEXT FREE RECORD IN AAINTN
C
99971 CONTINUE
      FREEAD=AAINTN(1,1)
      IF(FREEAD.NE.0) GO TO 50
      FREEAD=NAACTG+1
```

Sub-module BLDCOD

```
RELEASE 2.0                BLDCOD              DATE = 8012?          14/25/60
      AAINTN(FREEAD,1)=0
 50   AAINTN(1,1)=AAINTN(FREEAD,1)
      GO TO 99998
C ACTION 28 MAKE INTERNAL ACTION ENTRIES
C
99970 CONTINUE
      AAINTN(FREEAD,1)=NAIACS
      AATRNS(NPR)=FREEAD
      LINE(21)=0
      AAINTN(FREEAD,3)=30000+IPTR
      AAINTN(FREEAD,2)=0
      GO TO 99998
C ACTION 29 RECORD PRESENCE OF ACTION CODE
C
99969 CONTINUE
      ATAB(5)=.TRUE.
      NAACTS=NPR
      GO TO 99998
C ACTION 30 INCREMENT TOTAL NO. OF ACTIONS (INTERNAL)
C
99968 CONTINUE
      NAACTG=NAACTG+1
      GO TO 99998
C ACTION 31 SET CARD POINTER TO COLUMN 7
C
99967 CONTINUE
      NCOL=7
      GO TO 99998
C ACTION 32 CHECK WORD AGAINST COMPOUND ACTION NAMES
C
99966 CONTINUE
      N=APDTAB(72)
      MCMPAD=73
      CMPND=.FALSE.
```

Sub-module BLDCOD

```
      IF(N.EQ.0) GO TO 56
      DO 55 I=1,N
        IF(APDTAB(MCMPAD).NE.CHARST(1)) GO TO 55
        IF(APDTAB(MCMPAD+1).NE.CHARST(2)) GO TO 55
        CMPND=.TRUE.
        GO TO 56
 55     MCMPAD=MCMPAD+5
 56   CONTINUE
      GO TO 99998
C ACTION 33 LIST ACTION HEADER
C
99965 CONTINUE
      READ(3'IPTR,92) LINE
      NOCODE=LINE(21)
      FREEAD=AATRNS(NPR)
      WRITE(6,88) (LINE(I),I=1,18)
 88   FORMAT(1X,20A4)
      GO TO 99998
C ACTION 34 INCREMENT ACTION LINE NUMBER
C
99964 CONTINUE
      LINUM=LINUM+1
      GO TO 99998
C ACTION 35 SET GLOBAL TYPE NOT IDENTIFIED
C
99963 CONTINUE
      TYPID=0
      GO TO 99998
C ACTION 36 DECLARE END OF GLOBAL VARIABLE INPUT
C
99962 CONTINUE
      WRITE(6,87)
 87   FORMAT(' END GLOBAL INPUT')
      GO TO 99998
```

Sub-module BLDCOD

```
RELEASE 2.0              BLDCOD              DATE = 80124          14/25/50

C ACTION 37 FORCE SWITCH TO NEXT INPUT CARD
C
99961 CONTINUE
      INUNIT=GLOB
      GO TO 99998
C ACTION 38 MATCH DATA TYPE IN DATA DICT. LIST & REFERENCE THIS TABLE
C
99960 CONTINUE
      DO 65 I=1,12
        K=(I-1)*5
        DO 60 J=1,3
          IF(IBLK1(K+J).NE.CHARST(J)) GO TO 65
 60       CONTINUE
        TYPNO=IBLK1(K+5)
        CALL ADDTAB(IBLK3,TYPNO,APDTAB(71),IBLK1)
        GO TO 70
 65     CONTINUE
 70   CONTINUE
      GO TO 99998
C ACTION 39 ZERO GLOBAL VARIABLE TYPE NUMBER
C
99959 CONTINUE
      TYPNO=0
      GO TO 99998
C ACTION 40 INDICATE TYPE SEARCH HAS BEEN MADE
C
99958 CONTINUE
      TYPID=1
      GO TO 99998
C ACTION 41 "TYPE NOT RECOGNISED"
C
99957 CONTINUE
      WRITE(6,86) (CHARST(I),I=1,3)
 86   FORMAT(' TYPE ',3A4,' NOT RECOGNISED. - CARD SKIPPED')
```

Sub-module BLDCOD

```
      GO TO 99998
C ACTION 42 MOVE 'C ACTION NPR' INTO LINE
C
99956 CONTINUE
      LINE(1)=CACT(1)
      LINE(2)=CACT(2)
      LINE(3)=UNPAK(NPR)
      GO TO 99998
C ACTION 43 SPECIFY "VARIABLE LIST NOT YET STARTED"
C
99955 CONTINUE
      STVARL=.FALSE.
      MODFY=0
      GO TO 99998
C ACTION 44 PROCESS A VARIABLE THROUGH THE DATA DICTIONARY
C
99954 CONTINUE
      STVARL=.TRUE.
      CALL ANALYS(CARD,NCOL,VARNAM,CHARST,LGTH,MORE)
      CALL ADDVAR(VARNAM,CHARST,IBLK1,TYPNO,LGTH,ISEQ,IOFFST,IBLK3,
     +   MODFY)
      CALL AD2LST(LISTRT,ISEQ,IOFFST,IBLK3,IBLK1,APDTAB(71))
      APDTAB(255)=LISTRT
      GO TO 99998
C ACTION 45 MOVE COMPOUND STATEMENT INTO LINE
C
99953 CONTINUE
      DO 75 I=1,3
 75     LINE(3+I)=CHARST(I)
      CALL STRING(CARD,NCOL,CHARST,MORE,LAST,12)
      DO 80 I=1,3
        LINE(6+I)=CHARST(I)
        K=(I-1)*4+9
        DO 80 J=1,4
```

Sub-module BLDCOD

Sub-module BLDCOD


```
 80        LINE(J+K)=NBLNK
      LINE(21)=0
      GO TO 99998
C ACTION 46 SET LIMIT = NO. OF ACTIONS IN THIS COMPOUND
C
99952 CONTINUE
      LIMIT=APDTAB(MCMPAD+2)
      IACTN=1
      GO TO 99998
C ACTION 47 UNUSED
C
99951 CONTINUE
      GO TO 99998
C ACTION 48 REMEMBER CURRENT POSITION IN AAINTN
C
99950 CONTINUE
      OLDFRE=FREEAD
      GO TO 99998
C ACTION 49 MAKE INTERNAL ACTION ENTRIES FOR COMPOUND
C
99949 CONTINUE
      AAINTN(OLDFRE,2)=FREEAD
      AAINTN(FREEAD,2)=0
      AAINTN(FREEAD,1)=CNUMS(IACTN)
      AAINTN(FREEAD,3)=0
      GO TO 99998
C ACTION 50 SET POINTER TO FIRST COMPOUND ACTION
C
99948 CONTINUE
      IPTR=APDTAB(MCMPAD+3)
      GO TO 99998
C ACTION 51 "END OF ACTION"
C
99947 CONTINUE
```

```
      WRITE(6,85) NPR
 85   FORMAT(' END ACTION ',I3)
      GO TO 99998
C ACTION 52 MARK COMPOUND ACTION "USED"
C
99946 CONTINUE
      APDTAB(MCMPAD+4)=NPR
      GO TO 99998
C ACTION 53 INCREMENT ACTION COUNTER AND POINTER
C
99945 CONTINUE
      IACTN=IACTN+1
      IPTR=IPTR+1
      GO TO 99998
C ACTION 54 INITIALISE NCOL
C
99944 CONTINUE
      NCOL=1
      GO TO 99998
C ACTION 55 MODIFY ACTIONS
C
99943 CONTINUE
      WRITE(6,84)
 84   FORMAT(' ''MODIFY'' NOT IMPLEMENTED YET')
      GO TO 99998
C ACTION 56 NEXT COMMAND INTO INUNIT
C
99942 CONTINUE
      INUNIT=CHARSI(1)
      GO TO 99998
C ACTION 57 SET VALUE FOR REQUD NO. OF UNIT-3 RECORDS
C
99941 CONTINUE
      NRECS=NPR-NAPRDS
```

Sub-module BLDCOD

```
      GO TO 99998
C ACTION 58 REQUEST 1 UNIT-3 RECORD
C
99940 CONTINUE
      NRECS=1
      GO TO 99998
C ACTION 59 MOVE ENTIRE CARD INTO LINE
C
99939 CONTINUE
      DO 83 I=1,20
 83     LINE(I)=CARD(I)
      GO TO 99998
C ACTION 60 SET NO. OF UNIT-3 RECORDS TO BE FOLLOWED (NUMREC=NAPRDS)
C
99938 CONTINUE
      NUMREC=NAPRDS
      GO TO 99998
C ACTION 61 FETCH UNIT-3 FREE RECORD POINTER
C
99937 CONTINUE
      IVAR=APDTAB(258)
      GO TO 99998
C ACTION 62 FETCH PREDICATE CODE,HEADER
C
99936 CONTINUE
      IVAR=APDTAB(254)
      GO TO 99998
C ACTION 63 FETCH INTERNAL ACTION ADDR. FOR ACTION NPR
C
99935 CONTINUE
      IVAR=AATRNS(NPR)
      GO TO 99998
C ACTION 64 FETCH INTERNAL ACTION NO FOR COMPOUND ACTION
C
```

```
99934 CONTINUE
      IVAR=APDTAB(IPTR)
      GO TO 99998
C ACTION 65 FETCH "IN-USE" INDICATOR FOR COMPOUND ACTION
C
99933 CONTINUE
      JVAR=APDTAB(MCMPAD+4)
      GO TO 99998
C ACTION 66 MODIFY DD TABLE LIST TO SHOW CHANGES
C
99932 CONTINUE
      CALL MODTAB(IBLK1,IBLK3,TYPNO,MODFY)
      GO TO 99998
C ACTION 67 MARK ACTION CODE EXISTS
C
99931 CONTINUE
      ATAB(5)=.TRUE.
      GO TO 99998
C ACTION 68 MARK PREDICATE CODE EXISTS
C
99930 CONTINUE
      ATAB(6)=.TRUE.
      GO TO 99998
C ACTION 69 NOTE POSITION OF LAST CHARACTER
C
99929 CONTINUE
      TYPNO=LAST
      GO TO 99998
C ACTION 70 UPDATE RECORD POINTER TO NEXT FREE
C
99928 CONTINUE
      IPTR=LFREE
      GO TO 99998
C ACTION 71 POINT TO START OF ACTION CODE FOR THIS ACTION
```

Sub-module BLDCOD

```
C
99927 CONTINUE
      IPTR=AAINTN(AATRNS(NPR),3)
      N=IPTR/10000
      IPTR=IPTR-N*10000
      GO TO 99998
C ACTION 72 UPDATE RECORD POINTER
C
99926 CONTINUE
      IPTR=NOCODE
      GO TO 99998
C ACTION 73 UPDATE POINTER IN EXISTING COMPOUND LIST
C
99925 CONTINUE
      IACNO=AAINTN(IACNO,2)
      GO TO 99998
C ACTION 74 POINT TO START OF EXISTING COMPOUND LIST
C
99924 CONTINUE
      IACNO=AATRNS(JVAR)
      GO TO 99998
C ACTION 75 MODIFY ACTION ENTRIES FOR COMPOUND ACTION ALREADY USED
C
99923 CONTINUE
      AAINTN(FREEAD,1)=AAINTN(IACNO,1)
      GO TO 99998
C ACTION 76 INCREMENT NO OF INTERNAL ACTIONS IN USE
C
99922 CONTINUE
      NAIACS=NAIACS+1
      GO TO 99998
C ACTION 77 EXECUTE ACTION 1
C
99921 CONTINUE
```

Sub-module BLDCOD

```
RELEASE 2.0              BLDCOD              DATE = 80124 .        14/25/50
      WRITE(30) PDTABL
      GO TO 99998
C EXECUTE ACTION 2
C
99920 CONTINUE
      WRITE(30) (VECTOR(I),I=1,NRPRDS),ACTIVE,CURULE,NXTROW
      GO TO 99998
C EXECUTE ACTION 3
C
99919 CONTINUE
      NTF=14
      GO TO 99998
C EXECUTE ACTION 4
C
99918 CONTINUE
      READ(NTF'1) PDTABL
      GO TO 99998
C EXECUTE ACTION 5
C
99917 CONTINUE
      NSTRTS=PDTABL(3)
      NRPRDS=PDTABL(5)
      NACTS=PDTABL(6)
      NSTROW=PDTABL(7)
      NSTCOL=PDTABL(8)
      NSTRLS=PDTABL(9)
      NRULES=PDTABL(11)
      NARLS=PDTABL(12)
      MACACT=PDTABL(14)
      NACTG=PDTABL(18)
      NTF=PDTABL(71)
      GO TO 99998
C EXECUTE ACTION 6
C
```

Sub-module BLDCOD

```
RELEASE 2.0              BLDCOD              DATE = 80124              14/25/50

99916 CONTINUE
      II=PDTABL(21)-(PDTABL(21)/256*256)-29
      READ(NTF'II) ((RULES(I,J),I=1,NRPRDS),J=1,NRULES),
     +        ((MASKS(I,J),I=1,NRPRDS),J=1,NRULES)
      II=PDTABL(22)-PDTABL(22)/256*256-29
      READ(NTF'II) ((ACTNOS(I,J),I=1,MACACT),J=1,NARLS)
      II=PDTABL(23)-PDTABL(23)/256*256-29
      READ(NTF'II) ((STGY(I,J),I=1,NSTRLS),J=1,NSTRTS)
      II=PDTABL(25)-PDTABL(25)/256*256-29
      READ(NTF'II) (ATRANS(I),I=1,NACTS),((AINTNO(I,J),I=1,NACTG),
     +     J=1,2)
      GO TO 99998
C EXECUTE ACTION 7
C
99915 CONTINUE
      ACTIVE=NSTROW
      DO 99011 I99011=1,NRPRDS
99011   VECTOR(I99011)=.NOT.(RULES(I99011,NSTCOL).AND.TRUE)
      N99012=1
      DO 99012 I99012=1,N99012
        N99012=N99012+1
        CALL GENER(VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,MACACT,
     +     CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,MACTS,
     +     AINTNO,MACTIG,MAXSTR,ACTIVE)
        IF(CURULE.EQ.0) GO TO 99013
        NXTROW=STGY(CURULE,ACTIVE)
        IF(NXTROW.EQ.0) GO TO 99013
99012   ACTIVE=NXTROW
99013 CONTINUE
      GO TO 99998
C EXECUTE ACTION 8
C
99914 CONTINUE
      BACKSPACE 30
```

Sub-module BLDCOD

```
RELEASE 2.0            BLDCOD            DATE = 80124          14/25/50
      BACKSPACE 30
      READ(30) PDTABL
      GO TO 99998
C EXECUTE ACTION 9
C
99913 CONTINUE
      READ(30) (VECTOR(I),I=1,NRPRDS),ACTIVE,CURULE,NXTROW
      BACKSPACE 30
      BACKSPACE 30
      GO TO 99998
C ACTION 78 TALLY COMP ACT NOS IN CNUMS
C
99912 CONTINUE
      CNUMS(IACTN)=NAIACS
      GO TO 99998
C ACTION 79 REPEATED COMP ACT NOS IN CNUMS
C
99911 CONTINUE
      CNUMS(9)=CNUMS(5)
      CNUMS(11)=CNUMS(6)
      GO TO 99998
C END OF ACTIONS
99998 NXTACT=AINTNO(NXTACT,2)
99000 CONTINUE
      RETURN
99999 CONTINUE
      WRITE(6,90001) (VECTOR(I),I=1,NRPRDS)
90001 FORMAT(' VECTOR NOT MATCHED. VECTOR IS ',50L1)
      RETURN
      END
```

Sub-module BLDCOD

```
      SUBROUTINE ZLDCOZ(V,NRPRDS)
      LOGICAL*1 V(NRPRDS)
C*****APPLICATION SPECIFICATIONS****************************************
C---GLOBAL VARIABLES
      INTEGER APDTAB,ZI411,LINUM,ZI412,INUNIT,ZI413,AATRNS,AAINTN,ZI414
      INTEGER LIMIT,ZI415,MORE,ZI416,NAPRDS,NAACTS,NAACTG,NPR,NCOL
      INTEGER NOCODE,NF3REC,IVAR,TYPID,TYPNO,IACTN,JVAR,IBLK1,IBLK3
      INTEGER NAIACS,ZI417
      LOGICAL*1 ZL111,CMPND,INITDD,STVARL,LOGBLK,ZL112,ATAB,ZL113
      COMMON/I4/APDTAB(375),ZI411(4),LINUM,ZI412(2),INUNIT,ZI413(4910)
      COMMON/I4/AATRNS(120),AAINTN(150,3),ZI414(161),LIMIT,ZI415(2)
      COMMON/I4/MORE,ZI416(9),NAPRDS,NAACTS,NAACTG,NPR,NCOL,NOCODE
      COMMON/I4/NF3REC,IVAR,TYPID,TYPNO,IACTN,JVAR,IBLK1(100),IBLK3(100)
      COMMON/I4/NAIACS,ZI417(9)
      COMMON/L1/ZL111(10011),LOGBLK,ZL112(4),CMPND,INITDD,STVARL,ATAB(8)
      COMMON/L1/ZL113
C---LOCAL VARIABLES---
      INTEGER GLOB,CODE,ACTS,PRED,GEN
      DATA GLOB,CODE,ACTS,PRED,GEN/'GLOB','CODE','ACTS','PRED','GENE'/
C**********************************************************************
      V(1)=INUNIT.EQ.CODE
      V(2)=INUNIT.EQ.GLOB
      V(3)=INUNIT.EQ.ACTS
      V(4)=INUNIT.EQ.PRED
      V(5)=IVAR.LT.0
      V(6)=IVAR.GT.0
      V(7)=NPR.GT.NAACTS
      V(8)=NPR.GT.NAPRDS
      V(9)=NPR.EQ.0
      V(10)=NPR.GT.0
      V(11)=MORE.EQ.1
      V(12)=NCOL.EQ.0
      V(13)=CMPND
      V(14)=INITDD
```

Sub-module BLDCOB

```
      V(15)=NOCODE.EQ.0
      V(16)=LINUM.EQ.0
      V(17)=TYPID.EQ.0
      V(18)=TYPNO.GT.0
      V(19)=STVARL
      V(20)=IACTN.GT.LIMIT
      V(21)=JVAR.EQ.0
      V(22)=INUNIT.EQ.GEN
      RETURN
      END
```

Sub-module BLDCOD

```
CLUSTER:  START                                   RULES:  00
                                                          12

 1. INUNIT.EQ.CODE                                        YY
 2. INUNIT.EQ.GLOB                                        $$
 3. INUNIT.EQ.ACTS                                        $$
 4. INUNIT.EQ.PRED                                        $$
 5. IVAR.LT.0                                             $
 6. IVAR.GT.0                                             Y
 7. NPR.GT.NAACTS                                         Y
 8. NPR.GT.NAPRDS                                         Y
 9. NPR.EQ.0                                              $
10. NPR.GT.0                                              Y
11. MORE.EQ.1                                             Y
12. NCOL.EQ.0                                             Y
13. CMPND                                                 Y
14. INITDD                                                Y
15. NOCODE.EQ.0                                           Y
16. LINUM.EQ.0                                            Y
17. TYPID.EQ.0                                            Y
18. TYPND.GT.0                                            Y
19. STVARL                                                Y
20. IACTN.GT.LIMIT                                        Y
21. JVAR.EQ.0                                             Y
22. INUNIT.EQ[illegible]N                                 Y

STRATEGY                                                  --

 1. START
 2. IDENTIFY CODE                                         22

CLUSTER:  START                                   RULES:  00
                                                          12

 1 PROMPT FOR CODE TYPE                                   AA
 2 READ TYPE ID                                           BB
 4 READ D.D. TYPE LIST                                    C
61 SOURCE FREE RECORD POINTER                             DC

STRATEGY                                                  --

 1. START
 2. IDENTIFY CODE                                         22

                                                          --
```

Sub-module BLDCOD. Tables.

```
CLUSTER:  IDENTIFY CODE                              RULES:  00000000
                                                             12345678

  1. INUNIT.EQ.CODE                                           NSSSYS
  2. INUNIT.EQ.GLOB                                           NSSYSY
  3. INUNIT.EQ.ACTS                                           NSYSSS
  4. INUNIT.EQ.PRED                                           NYSSSS
  5. IVAR.LT.0                                               Y
  6. IVAR.GT.0                                               $
 14. INITDD                                                      N Y
 22. INUNIT.EQ.GEN                                            N     Y
                                                             --------
STRATEGY

  1. START
  2. IDENTIFY CODE                                           20345152
  3. PREDICATE CODE
  4. ACTION HEADER
  5. GLOBAL VARIABLES
                                                             --------

CLUSTER:  IDENTIFY CODE                              RULES:  00000000
                                                             12345678

  3 NO-OP                                                     A   A
  5 INITIALISE D.D. TYPE LIST                                       A
  6 INITIALISE SOURCE CODE FILE                              A
  7 PROMPT FOR INPUT                                           A A B
  8 READ A CARD                                                BC
  9 ISOLATE A CHAR. STRING                                     DD
 10 CONVERT EBCDIC TO INTEGER                                  EE
 11 "ACTION"                                                    B
 12 ZERO COLUMN NO.                                             I
 13 ZERO LINE NO.                                               F
 14 POINT TO TABLE'S GLOBAL VAR. LIST                            B C
 23 BLANK INUNIT                                                       B
 54 INITIALISE NCOL                                          A CA
 61 SOURCE FREE RECORD POINTER                               B
 62 PREDICATE CODE HEADER                                      F
 63 ADDR. INTERNAL ACTION FOR NPR                               G
 67 MARK ACTION CODE "EXISTS"                                   H
 68 NOTE POSITION OF LAST CHAR.                                G
 77 TALLY COMP. ACT. NOS IN CNUMS                                  A
                                                             --------
```

Sub-module BLDCOD. Tables.

```
CLUSTER:  PREDICATE CODE                                     RULES:  00000000
                                                                     12345678

  5. IVAR.LT.0                                                         YN$ - $
  6. IVAR.GT.0                                                         $NY   Y
  8. NPR.GT.NAPRDS                                                     YYY   N
  9. NPR.EQ.0                                                        NY$$$YN$
 10. NPR.GT.0                                                        N$YYY$NY
 11. MORE.EQ.1                                                       NN    YY

                                                                     --------
STRATEGY

  2. IDENTIFY CODE
  3. PREDICATE CODE                                                  33333223

                                                                     --------
CLUSTER:  PREDICATE CODE                                     RULES:  00000000
                                                                     12345678

  8 READ A CARD                                                      BBJJM  F
  9 ISOLATE A CHAR. STRING                                           CCKKN  G
 10 CONVERT EBCDIC TO INTEGER                                        DDLLO  H
 15 "ILLEGAL PRED. NO."                                              AA
 16 RESERVE NRECS SRCE. FILE RECS.                                    HBB
 17 UPDATE PRED. LIST PTR. TO HEAD                                    CC
 18 UPDATE NO. OF PREDS.                                              EEG
 19 TRACE TO END OF PRED LIST                                           D   B
 20 POINT TO NEW GROUP OF PREDS                                         E
 21 FILL LINE FROM CARD                                               FFH   C
 22 WRITE LINE TO SOURCE FILE                                         GGF   D
 23 BLANK INUNIT                                                          AA
 24 MARK LINE AS END OF GROUP                                         III
 25 SET NO OF PRED RECS TO FOLLOW                                           A
 54 INITIALISE NCOL                                                   HHL   E
 57 SET NO. SRCE. RECORDS REQUIRED                                    AAA
 60 NO. SRCE RECS. TO FOLLOW                                            C
 62 PREDICATE CODE HEADER                                             MM
 72 UPDATE COMPOUND LIST POINTER                                      DDJ

                                                                     --------
STRATEGY

  2. IDENTIFY CODE
  3. PREDICATE CODE                                                  33333223
                                                                     --------
```

Sub-module BLDCOD. Tables.

```
CLUSTER:  ACTION HEADER                              RULES:  00000000
                                                             12345678

  5. IVAR.LT.0                                                  N$$$
  6. IVAR.GT.0                                                  NYYY
  7. NPR.GT.NAACTS                                           YNNNNNNN
  9. NPR.EQ.0                                                $$$$$$$Y
 10. NPR.GT.0                                                YYYYYYY$
 12. NCOL.EQ.0                                                NNYYYY
 13. CMPND                                                    YN
 15. NOCODE.EQ.0                                                  YN
 16. LINUM.EQ.0                                                  YNN

:STRATEGY                                                    --------

  2. IDENTIFY CODE
  4. ACTION HEADER                                           46744482
  6. COMPOUND ACTION
  7. SIMPLE ACTION
  8. LIST/MOD ACTIONS

                                                             --------
  7 PROMPT FOR INPUT                                             D
  8 READ A CARD                                              N  L A
  9 ISOLATE A CHAR. STRING                                   E  H C
 12 ZERO COLUMN NO.                                            B
 13 ZERO LINE NO.                                                 E
 16 RESERVE NRECS SRCE. FILE RECS.                           F  C
 21 FILL LINE FROM CARD                                      J  I
 23 BLANK INUNIT                                                    A
 26 ZERO NEW BLOCK OF ATRANS                                  A
 27 NEXT FREE REC IN AAINTN                                  B  A
 28 MAKE INTERNAL ACTION ENTRIES                             I  F
 29 ACTION CODE "PRESENT"                                    L
 30 INCRMNT. TOTAL NO. INTERNAL ACTS.                        M  K
 31 POINT TO COLUMN 7                                        O  M B
 32 CHECK COMPOUND ACT. NAME LIST                            G  O D
 33 LIST ACTION HEADER                                            B
 34 INCRMNT. ACTION LINE NO                                   AA  C
 42 COMMENT HEADER INTO LINE                                 K  J
 54 INITIALISE NCOL                                          D  G
 58 REQUEST 1 SRCE. RECORD                                   C  B
 61 SOURCE FREE RECORD POINTER                                      B
 70 POINT TO START OF ACTION CODE                            G  D
 71 UPDATE RECORD POINTER                                        A
 72 UPDATE COMPOUND LIST POINTER                                   A
 76 EXECUTE GENER                                            H  E
```

Sub-module BLDCOD. Tables.

```
CLUSTER:  GLOBAL VARIABLES                          RULES:  0000000
                                                            1234567

  1. INUNIT.EQ.CODE                                         $
  2. INUNIT.EQ.GLOB                                         Y
  3. INUNIT.EQ.ACTS                                         $
  4. INUNIT.EQ.PRED                                         $
 11. MORE.EQ.1                                               YNNNNY
 12. NCOL.EQ.0                                                NNNNN
 17. TYPID.EQ.0                                              YYNNNN
 18. TYPNO.GT.0                                                NYYY
 19. STVARL                                                     NY

STRATEGY                                                    -------

  2. IDENTIFY CODE
  5. GLOBAL VARIABLES                                       5255555
                                                            -------


CLUSTER:  GLOBAL VARIABLES                          RULES:  0000000
                                                            1234567

  1 PROMPT FOR CODE TYPE                                     B
  2 READ TYPE ID                                             C
  8 READ A CARD                                             C
  9 ISOLATE A CHAR. STRING                                  E
 23 BLANK INUNIT                                            B
 35 GLOBAL TYPE NOT IDENTIFIED                              A
 36 "END GLOBAL INPUT"                                       A
 37 SWITCH TO NEXT INPUT CARD                                  B  A
 38 MATCH TYPE IN D.D. LIST                                   A
 39 ZERO TYPE NO.                                           F
 40 SHOW TYPE "SEARCHED"                                      B
 41 "TYPE NOT RECOGNISED"                                      A
 43 VARIABLE LIST "NOT STARTED"                             G
 44 PROCESS VBLE THROUGH D.D.                                   AA
 54 INITIALISE NCOL                                         D
 66 MODIFY D.D. BLOCK TO SHOW CHANGES                             B
                                                            -------
```

Sub-module BLDCOD.   Tables.

```
CLUSTER:  COMPOUND ACTION                                  RULES:  0000000
                                                                   1234567
  5. IVAR.LT.0                                                       N YSN
  6. IVAR.GT.0                                                       N $YN
 12. NCOL.EQ.0                                                     N
 20. IACTN.GT.LIMIT                                                 Y
 21. JVAR.EQ.0                                                       YY

STRATEGY                                                           -------

  2. IDENTIFY CODE
  6. COMPOUND ACTION                                               6266666

                                                                   -------


CLUSTER:  COMPOUND ACTION                                  RULES:  0000000
                                                                   1234567
 12 ZERO COLUMN NO.                                                D
 22 WRITE LINE TO SOURCE FILE                                      B
 27 NEXT FREE REC IN AAINTN                                          BDDBB
 30 INCRMNT. TOTAL NO. INTERNAL ACTS.                                DFFDD
 45 COMPOUND STMT INTO LINE                                        A
 46 LIMIT = NO. ACTS. IN THIS COMPOUND                             E
 48 REMEMBER POSITION IN AAINTN                                       ACCAA
 49 INTERNAL ACTION ENTRIES FOR COMPOUND                             CEECC
 50 POINT TO FIRST COMPOUND  ACTION                                C
 51 "END OF ACTION"                                                 A
 52 MARK COMPOUND ACTIONS "USED"                                   J
 53 INCRMNT ACTION COUNTER & POINTER                                 EGGEE
 64 INTERNAL ACT. NO FOR COMPOUND                                  I FHHFF
 65 "IN-USE" INDICATOR FOR COMPOUND                                F
 73 POINT TO START OF COMPOUND LIST                                H   IHH
 74 MODIFY ACTION ENTRIES FOR USED COMPOUND                        G
 75 INCRMNT. NO INTERNAL ACTIONS USED.                                  GG
 76 EXECUTE GENER                                                     AA
 78 REPEATED COMP. ACT. NOS IN CNUMS                                  BB

                                                                   -------
```

Sub-module BLDCOD. Tables.

```
CLUSTER:  SIMPLE ACTION                                     RULES:  000
                                                                    123
12. NCOL.EQ.0                                                       YNN
18. TYPNO.GT.0                                                       NY
                                                                    ---
STRATEGY

 2. IDENTIFY CODE
 7. SIMPLE ACTION                                                   727
                                                                    ---


CLUSTER:  SIMPLE ACTION                                     RULES:  000
                                                                    123
  8 READ A CARD                                                       H
  9 ISOLATE A CHAR. STRING                                          B J
 16 RESERVE NRECS SRCE. FILE RECS.                                    B
 20 POINT TO NEW GROUP OF PREDS                                       C
 22 WRITE LINE TO SOURCE FILE                                        BD
 24 MARK LINE AS END OF GROUP                                         G
 51 "END OF ACTION"                                                  A
 54 INITIALISE NCOL                                                 A I
 58 REQUEST 1 SRCE. RECORD                                            A
 59 MOVE ENTIRE CARD INTO LINE                                        F
 61 SOURCE FREE RECORD POINTER                                       C
 69 POINT TO NEXT FREE RECORD                                       C K
 70 POINT TO START OF ACTION CODE                                     E
                                                                    ---
STRATEGY

 2. IDENTIFY CODE
 7. SIMPLE ACTION                                                   727
                                                                    ---
```

Sub-module BLDCOD. Tables.

```
CLUSTER:  LIST/MOD ACTIONS                                RULES:  00
                                                                  12
15. NOCODE.EQ.0                                                   NY

STRATEGY                                                          --

 2. IDENTIFY CODE
 8. LIST/MOD ACTIONS                                              82

                                                                  --


CLUSTER:  LIST/MOD ACTIONS                                RULES:  00
                                                                  12
  1 PROMPT FOR CODE TYPE                                           C
  8 READ A CARD                                                    D
  9 ISOLATE A CHAR. STRING                                         F
 13 ZERO LINE NO.                                                  B
 33 LIST ACTION HEADER                                            A
 34 INCRMNT. ACTION LINE NO                                       B
 54 INITIALISE NCUL                                                E
 55 "MODIFY NOT IMPLEMENTED"                                       A
 56 INUNIT = NEXT COMMAND                                          G
 72 UPDATE COMPOUND LIST POINTER                                  C

STRATEGY                                                          --

 2. IDENTIFY CODE
 8. LIST/MOD ACTIONS                                              82

                                                                  --
```

Sub-module BLDCOD. Tables.

Sub-module GENER

```
RELEASE 2.0             GENER                DATE = 80124          13/19/59
      SUBROUTINE GENER (VECTOR,RULES,MASKS,ACTNOS,STGY,NRPRDS,NRULES,
     + MACACT,CURULE,PDTABL,TABL,TABSIZ,MAXRLS,MAXPRD,MAXACT,ATRANS,
     + MACTS,AINTNO,MACTIG,MAXSTR,ACTIVE)
C SYSTEM SPECIFICATIONS **********************************************
      INTEGER TABSIZ,CURULE,ACTIVE,BLANK/' '/
      INTEGER PDTABL(TABSIZ),STGY(MAXRLS,MAXSTR),ACTNOS(MAXACT,MAXRLS),
     + ATRANS(MACTS),AININO(MACTIG,2)
      LOGICAL*1 TABL(8),RULES(MAXPRD,MAXRLS),MASKS(MAXPRD,MAXRLS),
     + VECTOR(MAXPRD),INIT/.TRUE./,TRUE/.TRUE./
      LOGICAL MATCH
C APPLICATION SPECIFICATIONS *****************************************
C --- GLOBAL DECLARATIONS ---
      LOGICAL*1  ZL111,LOGDOL,LOGBLK,ZL112,ATAB,STBLK,LATAB
      INTEGER    IACTNO,CMPFLG
      INTEGER    CACTNO,CPTR,GLOBVL,LACT,LPRDL,LNARPR,LNASTR,LNAARL
      INTEGER    NAIACS,EXFLAG,COUNTR,BLKNO,NARULE,NAARL,IPTR,ACTNUM
      INTEGER    AATRNS,AAINTN,ZI415,NAPRDS,NAACTS,ZI416,IBLK1,ZI417
      INTEGER    APDTAB,ZI411,LINUM,ZI412,NASTRL,ZI413,NARPRD,ZI414
      COMMON/I4/ APDTAB(375),ZI411(4),LINUM,ZI412,NASTRL,ZI413(3),NARPRD
      COMMON/I4/ ZI414(4907),AATRNS(120),AAINTN(150,3),ZI415(178),NAPRDS
      COMMON/I4/ NAACTS,ZI416(10),IBLK1(100),ZI417(100),NAIACS,EXFLAG
      COMMON/I4/ COUNTR,BLKNO,NARULE,NAARL,IPTR,ACTNUM,CACTNO,CPTR
      COMMON/I4/ GLOBVL,LACT,LPRDL,LNARPR,LNASTR,LNAARL,IACTNO,CMPFLG
      COMMON/L1/ ZL111(10010),LOGDOL,LOGBLK,ZL112(7),ATAB(8),STBLK
      COMMON/L1/ LATAB(8)
C --- LOCAL DECLARATIONS ----
      INTEGER BLCODP(10),BSEQNO,TN(2),IB(2),UNPAK,LINE4(20)
      INTEGER BASE,EXTN4(2),TFN,DRVALS(3),PRDNUM,VCODE(2)
      INTEGER NBLNK/' '/,MFLG(2)/' ','MOD'/
      LOGICAL*1 LTN(8),LB(8),LABEL1(4),LVC(8)
      INTEGER*2 LINE(40),ICOM9,IBRK9,IPLUS,IBLNK,LABEL2(2),EXTNAM(4)
      EQUIVALENCE (TN(1),LTN(1)),(IB(1),LB(1)),(LABEL,LABEL1(1),
     +     LABEL2(1)),(LINE(1),LINE4(1)),(EXTNAM(1),EXTN4(1)),
     +     (VCODE(1),LVC(1))
```

```
RELEASE 2.0                GENER                    DATE = 80124           13/19/59
        DATA ICOM9/',9'/,IBRK9/'(9'/,IPLUS/' +'/,IBLNK/' '/,VCODE/'  V(',
     +        ' )='/
C END OF SPECIFICATIONS ***************************************************
        IF(.NOT.INIT)CALL $GENES  (VECTOR,NRPRDS)
        INIT=.FALSE.
        CURULE=0
        DO 99999 I99999=1,NRULES
          IF(SIGY(I99999,ACTIVE).EQ.BLANK) GO TO 99999
          IF(.NOT.MATCH(RULES(1,I99999),MASKS(1,I99999),VECTOR,NRPRDS))
     +          GO TO 99999
          CURULE=I99999
          IF(SIGY(CURULE,ACTIVE).EQ.0) INIT=.TRUE.
          DO 99000 I99000=1,MAXACT
            NEXTAC=ACTNOS(I99000,CURULE)
            IF(NEXTAC.EQ.0) RETURN
            NXTACT=ATRANS(NEXTAC)
            DO 99998 I99998=1,MAXACT
              IF(NXTACT.EQ.0) GO TO 99000
              NACSHN=AINTNO(NXTACT,1)
              GO TO
     +(99997,99996,99995,99994,99993,99992,99991,99990,99989,99988
     +,99987,99986,99985,99984,99983,99982,99981,99980,99979,99978
     +,99977,99976,99975,99974,99973,99972,99971,99970,99969,99968
     +,99967,99966,99965,99964,99963,99962,99961,99960,99959,99958
     +,99957,99956,99955,99954,99953,99952,99951,99950,99949,99948
     +,99947,99946,99945,99944,99943,99942,99941,99940,99939,99938
     +,99937,99936,99935,99934,99933,99932,99931,99930,99929
     +),NACSHN
C ACTION ENTRIES START HERE ***********************************************
C ACTION 1 INITIALISE LOCAL VARIABLES
C
99997 CONTINUE
        BSEQNO=0
        DO 197 I=1,8
```

Sub-module GENER

```
  197    LATAB(I)=ATAB(I)
         GLOBVL=APDTAB(255)
         LACTL=APDTAB(260)
         LNARPR=NARPRD
         LNASTR=NASTRL
         LNAARL=NAARL
         CMPFLG=-IABS(APDTAB(77))
         GO TO 99998
C ACTION    2 WARNING MESSAGE
C
99996 CONTINUE
         WRITE(6,999)
  999    FORMAT(' ***WARNING*** THERE IS NO RULE TABLE - EXECUTION IS IMPOS
        +SIBLE')
         LATAB(1)=.TRUE.
         GO TO 99998
C ACTION    3 WARNING MESSAGE
C
99995 CONTINUE
         WRITE(6,998)
  998    FORMAT(' ***WARNING*** THERE IS NO ACTION TABLE - EXECUTION IS IMP
        +OSSIBLE')
         LATAB(2)=.TRUE.
         GO TO 99998
C ACTION    4 WARNING MESSAGE
C
99994 CONTINUE
         WRITE(6,997)
  997    FORMAT(' ***WARNING*** THERE IS NO STRATEGY TABLE - EXECUTION IS I
        +MPOSSIBLE')
         LATAB(3)=.TRUE.
         GO TO 99998
C ACTION    5 ERROR MESSAGE
C
```

Sub-module GENER

```
RELEASE 2.0              GENER              DATE = 80124          13/19/59

 99993 CONTINUE
       WRITE(6,996)
   996 FORMAT(' ****ERROR**** THERE IS NO ACTION CODE')
       LATAB(5)=.TRUE.
       GO TO 99998
C ACTION   6 ERROR MESSAGE
C
 99992 CONTINUE
       WRITE(6,995)
   995 FORMAT(' ****ERROR**** THERE IS NO PREDICATE CODE')
       LATAB(6)=.TRUE.
       GO TO 99998
C ACTION   7 WARNING MESSAGE
C
 99991 CONTINUE
       WRITE(6,994)
   994 FORMAT(' ***WARNING*** NUMBER OF PREDICATE SOURCE STATEMENTS DOES
      +NOT MATCH DECLARED NUMBER OF PREDICATES')
       LNARPR=NAPRDS
       GO TO 99998
C ACTION   8 WARNING MESSAGE
C
 99990 CONTINUE
       WRITE(6,993)
   993 FORMAT(' ***WARNING*** NUMBER OF RULES DOES NOT MATCH NUMBER OF SE
      +TS OF GUARDS')
       LNASTR=NARULE
       GO TO 99998
C ACTION   9 WARNING MESSAGE
C
 99989 CONTINUE
       WRITE(6,992)
   992 FORMAT(' ***WARNING*** NUMBER OF RULES DOES NOT MATCH NUMBER OF SE
      +TS OF ACTIONS')
```

Sub-module GENER

```
      LNAARL=NARULE
      GO TO 99998
C ACTION  10 WARNING MESSAGE
C
99988 CONTINUE
      WRITE(6,991)
  991 FORMAT(' ***WARNING*** THERE ARE NO GLOBAL VARIABLE DECLARATIONS F
     +OR THIS SUB-MODULE')
      GLOBVL=1
      GO TO 99998
C ACTION  11 WARNING MESSAGE
C
99987 CONTINUE
      WRITE(6,990)
  990 FORMAT(' ***WARNING*** THERE ARE NO LOCAL VARIABLE DECLARATIONS FO
     +R THIS SUB-MODULE')
      LACTL=1
      GO TO 99998
C ACTION  12 WARNING MESSAGE
C
99986 CONTINUE
      WRITE(6,989)
  989 FORMAT(' ***WARNING*** THERE ARE NO LOCAL VARIABLE DECLARATIONS FO
     +R THE PREDICATE SUBROUTINE')
      LPRDL=1
      GO TO 99998
C ACTION  13 CLEAR EXECUTE FLAG
C
99985 CONTINUE
      EXFLAG=0
      GO TO 99998
C ACTION  14 SET WARNING FLAG
C
99984 CONTINUE
```

Sub-module GENER

```
RELEASE 2.0               GENER                DATE = 80124          13/19/59
      EXFLAG=MAX0(1,EXFLAG)
      GO TO 99998
C ACTION  15 SET ERROR FLAG
C
99983 CONTINUE
      EXFLAG=MAX0(2,EXFLAG)
      GO TO 99998
C ACTION  16 'CONTINUE?' MESSAGE
C
99982 CONTINUE
      WRITE(6,988)
  988 FORMAT(/' WARNING MESSAGES ISSUED - HIT C/R TO GENERATE, OR TYPE '
     +'99'' TO END')
      READ(9,987) EXFLAG
  987 FORMAT(I4)
      GO TO 99998
C ACTION  17 NO-OP
C
99981 CONTINUE
      GO TO 99998
C ACTION  18 NO-OP
C
99980 CONTINUE
      GO TO 99998
C ACTION  19 'GENERATION SUPPRESSED' MESSAGE
C
99979 CONTINUE
      WRITE(6,986)
  986 FORMAT(' SOURCE GENERATION SUPPRESSED')
      GO TO 99998
C ACTION  20 SET START OF BLOCK INDICATOR .TRUE.
C
99978 CONTINUE
      STBLK=.TRUE.
```

Sub-module GENER

```
      GO TO 99998
C ACTION  21 SET START OF BLOCK INDICATOR .FALSE.
C
99977 CONTINUE
      STBLK=.FALSE.
      GO TO 99998
C ACTION  22 ZERO COUNTER
C
99976 CONTINUE
      COUNTR=0
      GO TO 99998
C ACTION  23 ZERO BLOCK NUMBER
C
99975 CONTINUE
      BLKNO=0
      GO TO 99998
C ACTION  24 SET IPTR TO START OF CURRENT BLK OF CONTROL CODE
C
99974 CONTINUE
      IPTR=BLCODP(BSEQNO)
      GO TO 99998
C ACTION  25 GET LIST OF CONTROL BLK-POINTERS FROM DISK
C
99973 CONTINUE
      READ(4'1,985) BLCODP
  985 FORMAT(30X,10I5)
      GO TO 99998
C ACTION  26 POINT TO SYSTEMS SOURCE FILE (NTF=4)
C
99972 CONTINUE
      NTF=4
      GO TO 99998
C ACTION  27 READ CARD INTO BLOCK - UPDATE IPTR
C
```

Sub-module GENER

```
RELEASE 2.0              GENER                DATE = 80124        13/19/59
 99971 CONTINUE
       J=(COUNTR-1)*20+1
       K=J+19
       READ(NTF'IPTR,984) (IBLK1(I),I=J,K),IPTR
  984  FORMAT(20A4,I4)
       GO TO 99998
C ACTION  28 INCREMENT COUNTER
C
99970 CONTINUE
       COUNTR=COUNTR+1
       GO TO 99998
C ACTION  29 FETCH TABLE NAME & INSERT IN SUBROUTINE STATEMENT
C
99969 CONTINUE
       DO 691 I=1,2
         TN(I)=APDTAB(I)
  691    IB(I)=IBLK1(4+I)
       DO 692 I=1,6
  692    LB(I+1)=LIN(I)
       IBLK1(5)=IB(1)
       IBLK1(6)=IB(2)
       GO TO 99998
C ACTION  30 INCREMENT BLOCK NUMBER
C
99968 CONTINUE
       BLKNO=BLKNO+1
       GO TO 99998
C ACTION  31 WRITE BLOCK TO OUTPUT FILE
C
99967 CONTINUE
       J=COUNTR*20
       WRITE(7,983) (IBLK1(I),I=1,J)
  983  FORMAT(20A4)
       GO TO 99998
```

Sub-module GENER

```
RELEASE 2.0              GENER              DATE = 80124           13/19/59
  C ACTION  32 SET IPTR TO START OF LOCAL ACTION CODE LIST N FILE 3
  C
  99966 CONTINUE
        IPTR=APDTAB(259)
        GO TO 99998
  C ACTION  33 POINT TO USER'S SOURCE FILE UNIT 3
  C
  99965 CONTINUE
        NTF=3
        GO TO 99998
  C ACTION  35 INCREMENT BLSQNO' ( CODE BLOCK SEQUENCE NUMBER )
  C
  99964 CONTINUE
        BSEQNO=BSEQNO+1
        GO TO 99998
  C ACTION  36 OUTPUT 'C LOCAL VARIABLES....'
  C
  99963 CONTINUE
        WRITE(7,982)
   982  FORMAT('C --- LOCAL DECLARATIONS ----')
        GO TO 99998
  C ACTION  37 INSERT PREDICATE SUBROUTINE NAME INTO BLOCK
  C
  99962 CONTINUE
        LABEL=NBLNK
        DO 611 I=1,4
          LABEL1(1)=LB(6-I)
   611    IF(LABEL.EQ.NBLNK) LB(6-I)=LOGDOL
        LB(1)=LOGDOL
        LB(6)=LOGDOL
        LB(7)=LOGBLK
        LB(8)=LOGBLK
        IBLK1(27)=IB(1)
        IBLK1(28)=IB(2)
```

```
      TN(1)=IB(1)
      TN(2)=IB(2)
      GO TO 99998
C ACTION  38 OUTPUT COMPUTED GOTO NUMBERS
C
99961 CONTINUE
      L=10
      N=(NAIACS+9)/10
      NR=NAIACS-(N-1)*10
      LINE(1)=IBLNK
      LINE(2)=IBLNK
      LINE(3)=IPLUS
      LINE(4)=IBRK9
      MM=9998
      DO 603 I=1,N
        NUM=5
        IF(I.EQ.N) L=NR
        DO 601 J=1,L
          MM=MM-1
          LABEL=UNPAK(MM)
          LINE(NUM)=LABEL2(1)
          LINE(NUM+1)=LABEL2(2)
          LINE(NUM+2)=ICOM9
601       NUM=NUM+3
        NUM=NUM-1
        DO 602 J=NUM,40
602       LINE(J)=IBLNK
        WRITE(7,980) LINE
603     LINE(4)=ICOM9
980   FORMAT(40A2)
      WRITE(7,981)
981,  FORMAT(5X,'+),NACSMN'/'C ACTION ENTRIES START HERE ',44('*'))
      AAINTN(1,3)=90000+MM-1
      GO TO 99998
```

Sub-module GENER

```
RELEASE 2.0                GENER              DATE = 80124           13/19/59
  C ACTION  39 USE IPTR TO IDENTIFY ACTION, SIMPLE VS COMPOUND
  C
  99960 CONTINUE
        IPTR=AAININ(IACTNO,2)
        GO TO 99998
  C ACTION  40 OUTPUT ACTION HEADER LABEL STATEMENT
  C
  99959 CONTINUE
        LABNO=99998-AAINTN(IACTNO,1)
        WRITE(7,979) LABNO
   979  FORMAT('C'/I5,1X,'CONTINUE')
        GO TO 99998
  C ACTION  41 SET IACTNO = CURRENT INTERNAL ACTION NUMBER
  C
  99958 CONTINUE
        IACTNO=AATRNS(ACTNUM)
        GO TO 99998
  C ACTION  42 READ A LINE OF CODE FROM UNIT 3
  C
  99957 CONTINUE
        READ(NTF'IPTR,984) LINE4,IPTR
        GO TO 99998
  C ACTION  43 SET IPTR = UNIT 3 LOCATION OF SOURCE CODE FOR THIS ACTION
  C
  99956 CONTINUE
        IPTR=AAINTN(IACTNO,3)
        NN=IPTR/10000
        IPTR=IPTR-NN*10000
        GO TO 99998
  C ACTION  44 OUTPUT LINE OF CODE
  C
  99955 CONTINUE
        WRITE(7,984) LINE4
        GO TO 99998
```

Sub-module GENER

```
RELEASE 2.0              GENER                DATE = 80124         13/19/59
 C ACTION  45 OUTPUT ACTION TRAILER STATEMENT
 C
 99954 CONTINUE
       WRITE(7,978)
  978  FORMAT(6X,'GO TO 99998')
       GO TO 99998
 C ACTION  46 ZERO COMPOUND ACTION COUNTER
 C
 99953 CONTINUE
       CACTNO=0
       GO TO 99998
 C ACTION  47 INCREMENT COMPOUND ACTION COUNTER
 C
 99952 CONTINUE
       CACTNO=CACTNO+1
       GO TO 99998
 C ACTION  48 REMEMBER NAME OF TABLE BEING CALLED
 C
 99951 CONTINUE
       EXTN4(1)=LINE4(7)
       EXTN4(2)=LINE4(8)
       GO TO 99998
 C ACTION  49 MARK COMPOUND ACTION AS 'USED'
 C
 99950 CONTINUE
       CMPFLG=IABS(CMPFLG)
       GO TO 99998
 C ACTION  50 UPDATE INTERNAL ACTION NUMBER IACTNO TO NEXT ACTION IN LIST
 C
 99949 CONTINUE
       IACTNO=AAINTN(IACTNO,2)
       GO TO 99998
 C ACTION  51 SET CPTR TO HEAD OF SOURCE CODE FOR THIS ACTION
 C
```

Sub-module GENER

```
RELEASE 2.0              GENER              DATE = 80124         13/19/59
  99948 CONTINUE
        CPTR=IABS(APDTAB(BASE+CACTNO))
        NNF=CPTR/10000
        CPTR=CPTR-NNF*10000
        GO TO 99998
C ACTION  52 SET LOCATION OF START OF COMPOUND ACTION LIST
C
  99947 CONTINUE
        BASE=APDTAB(76)-1
        GO TO 99998
C ACTION  53 UPDATE IPTR FROM CPTR
C
  99946 CONTINUE
        IPTR=CPTR
        GO TU 99998
C ACTION  54 ZERO LINUM
C
  99945 CONTINUE
        LINUM=0
        GO TO 99998
C ACTION  55 INCREMENT LINUM
C
  99944 CONTINUE
        LINUM=LINUM+1
        GO TO 99998
C ACTION  56 LOCATE FILE NUMBER FROM TABLE NAME (TFN)
C
  99943 CONTINUE
        READ(8'3) IBLK1
        TFN=99999
        N=IBLK1(4)*4
        DO 422 I=4,N,4
          DU 421 J=1,2
            IF(IBLK1(I+J).NE.EXTN4(J)) GO TO 422
```

Sub-module GENER

```
 421       CONTINUE
         TFN=IBLK1(I+3)
         GO TO 423
 422     CONTINUE
 423   CONTINUE
       GO TO 99998
C ACTION  57 OUTPUT COMPOUND ACTION 'NTF=TFN' STATEMENT
C
99942 CONTINUE
       WRITE(7,977) TFN
 977   FORMAT(6X,'NTF=',I2)
       GO TO 99998
C ACTION  58 ZERO ACTION  NUMBER ACTNUM
C
99941 CONTINUE
       ACTNUM=0
       GO TO 99998
C ACTION  59 INCREMENT ACTNUM
C
99940 CONTINUE
       ACTNUM=ACTNUM+1
       GO TO 99998
C ACTION  60 INSERT LABELS, TABLE NAME INTO MINI-DRIVER & WRITE 1ST PART
C
99939 CONTINUE
       MM=AAINTN(1,3)
       DO 381 I=1,3
 381     DRVALS(I)=MM+1-I
       AAINTN(1,3)=MM-3
       WRITE(7,976) (DRVALS(1),I=1,5),(DRVALS(2),I=1,6)
 976   FORMAT(6X,'ACTIVE=NSTROW'/6X,'DO ',I5,' I',I5,'=1,NRPRDS'/I5,2X,
     +'VECTOR(I',I5,')=.NOT.(RULES(I',I5,',NSTCOL).AND.TRUE)'/6X,'N',I5,
     +'=1'/6X,'DO  ',I5,' I',I5,'=1,N',I5/7X,'N',I5,'=N',I5,'+1')
       IBLK1(4)=EXTN4(1)
```

```
      IBLK1(5)=EXTN4(2)
      IBLK1(69)=UNPAK(DRVALS(3)-90000)
      IBLK1(109)=IBLK1(69)
      LABEL=UNPAK(DRVALS(2)-90000)
      IADD=120
      DO 383 J=1,2
        IB(1)=IBLK1(IADD+1)
        IB(2)=IBLK1(IADD+2)
        DO 382 I=1,4
382       LB(I+1)=LABEL1(I)
        IBLK1(IADD+1)=IB(1)
        IBLK1(IADD+2)=IB(2)
        IADD=IADD+20
383     LABEL=IBLK1(69)
      GO TO 99998
C ACTION  61 INSERT PREDICATE SUBROUTINE NAME INTO BLOCK
C
99938 CONTINUE
      IBLK1(6)=TN(1)
      IBLK1(7)=TN(2)
      GO TO 99998
C ACTION  62 SET IPTR TO START OF PREDICATE LOCAL VARIABLE
C
99937 CONTINUE
      IPTR=APDTAB(260)
      GO TO 99998
C ACTION  63 CALL GLOBAL DATA GENERATION UTILITY
C
99936 CONTINUE
      CALL DDSGEN(APDTAB(1),APDTAB(255),APDTAB(256),APDTAB(257),
     +        APDTAB(258))
      GO TO 99998
C ACTION  64 SET IPTR TO HEAD OF GLOBAL STATEMENT CODE ON UNIT 3
C
```

```
RELEASE 2.0                GENER                DATE = 80124           13/19/59
99935 CONTINUE
      IPTR=APDTAB(256)
      GO TO 99998
C ACTION  65 OUTPUT 'C GLOBAL VARIABLES....'
C
99934 CONTINUE
      WRITE(7,975)
 975  FORMAT('C --- GLOBAL DECLARATIONS ---')
      GO TO 99998
C ACTION  66 SET IPTR TO START OF PREDICATE SOURCE CODE ON UNIT 3
C
99933 CONTINUE
      IPTR=APDTAB(254)
      GO TO 99998
C ACTION  67 INSERT 'V(..)=' INTO BLOCK
C
99932 CONTINUE
      DO 315 I=1,COUNTR
        J=(I-1)*20
        IF(IBLK1(J+2).NE.NBLNK) GO TO 315
        PRDNUM=PRDNUM+1
        LABEL=UNPAK(PRDNUM)
        LVC(5)=LABEL1(3)
        LVC(6)=LABEL1(4)
        IBLK1(J+2)=VCODE(1)
        IBLK1(J+3)=VCODE(2)
 315    CONTINUE
      GO TO 99998
C ACTION  68 OUTPUT 'RETURN' & 'END' STATEMENTS
C
99931 CONTINUE
      WRITE(7,974)
 974  FORMAT(6X,'RETURN'/6X,'END')
      GO TO 99998
```

Sub-module GENER

```
RELEASE 2.0                 GENER                DATE = 80124          13/19/59
      C ACTION  69 LIST FILE USAGE TABLE WITH MODIFICATION FLAGS
      C
      99930 CONTINUE
            EXFLAG=0
            WRITE(6,973)
        973 FORMAT(/' GENERATION COMPLETE'//' AT EXECUTION TIME, TABLES FOR TH
           +E FOLLOWING SUB-MODULES MUST BE ASSOCIATED'/' WITH THE FORTRAN UNI
           +T NUMBER SHOWN'//9X,'TABLE',8X,'UNIT'/)
            READ(8'3) IBLK1
            DO 295 I=7,99,4
              IF(IBLK1(I).EQ.0) GO TO 299
              IF(IBLK1(I).NE.APDTAB(71)) GO TO 291
              IBLK1(I+1)=0
        291   IPTR=IBLK1(I+1)
              J=I-2
              EXFLAG=EXFLAG+IPTR
        295   WRITE(6,972) (IBLK1(L),L=J,I),MFLG(IPTR+1)
        972   FORMAT(8X,2A4,5X,I5,3X,A4)
        299 IF(EXFLAG.GT.0) WRITE(6,971)
        971 FORMAT(/' TABLES MARKED "MOD" REQUIRE SOURCE GENERATION TO MAINTAI
           +N COMPATIBILITY WITH OTHER SUB-MODULES'/)
            GO TO 99998
      C ACTION 70 ZERO PREDICATE COUNTER.
      C
      99929 CONTINUE
            PRDNUM=0
            GO TO 99998
      C END OF ACTIONS **********************************************************
      99998 NXTACT=AININU(NXTACT,2)
      99000 CONTINUE
            RETURN
      99999 CONTINUE
            WRITE(6,90001) PDTABL(1),PDTABL(2),ACTIVE,(VECTOR(I),I=1,NRPRDS)
      90001 FORMAT(' VECTOR NOT MATCHED IN ',A4,A2,' STRATEGY ',I2/' VECTOR WA
```

```
RELEASE 2.0                    GENER              DATE = 80124         13/19/59
      +S ',50L1)
       RETURN
       END
```

Sub-module GENER

```
RELEASE 2.0            $GENES            DATE = 80124         13/19/59
          SUBROUTINE    $GENES  (V,NRPRDS)
      C SYSTEM SPECIFICATIONS ********************************************
          LOGICAL*1 V(NRPRDS)
      C APPLICATION SPECIFICATIONS ***************************************
      C --- GLOBAL DECLARATIONS ---
          LOGICAL*1  ZL111,LOGDOL,LOGBLK,ZL112,ATAB,STBLK,LATAB
          INTEGER    IACINU,CMPFLG
          INTEGER    CACTNO,CPTR,GLOBVL,LACT,LPRDL,LNARPR,LNASTR,LNAARL
          INTEGER    NAIACS,EXFLAG,COUNTR,BLKNO,NARULE,NAARL,IPTR,ACTNUM
          INTEGER    AATRNS,AAINTN,ZI415,NAPRDS,NAACTS,ZI416,IBLK1,ZI417
          INTEGER    APDTAB,ZI411,LINUM,ZI412,NASTRL,ZI413,NARPRD,ZI414
          COMMON/I4/ APDTAB(375),ZI411(4),LINUM,ZI412,NASTRL,ZI413(3),NARPRD
          COMMON/I4/ ZI414(4907),AATRNS(120),AAINTN(150,3),ZI415(174),NAPRDS
          COMMON/I4/ NAACTS,ZI416(10),IBLK1(100),ZI417(100),NAIACS,EXFLAG
          COMMON/I4/ COUNTR,BLKNO,NARULE,NAARL,IPTR,ACTNUM,CACTNO,CPTR
          COMMON/I4/ GLOBVL,LACT,LPRDL,LNARPR,LNASTR,LNAARL,IACTNO,CMPFLG
          COMMON/L1/ ZL111(10010),LOGDOL,LOGBLK,ZL112(7),ATAB(8),STBLK
          COMMON/L1/ LATAB(8)
      C --- LOCAL DECLARATIONS ----
          V( 1)= .NOT.ATAB(1)
          V( 2)= .NOT.ATAB(2)
          V( 3)= .NOT.ATAB(3)
          V( 4)= .NOT.ATAB(5)
          V( 5)= .NOT.ATAB(6)
          V( 6)= NARPRD.NE.NAPRDS
          V( 7)= NASTRL.NE.NARULE
          V( 8)= NAARL.NE.NARULE
          V( 9)= APDTAB(255).EQ.0
          V(10)= APDTAB(259).EQ.0
          V(11)= APDTAB(260).EQ.0
          V(12)= EXFLAG.EQ.0
          V(13)= EXFLAG.EQ.1
          V(14)= STBLK
          V(15)= BLKNO.EQ.0
```

Sub-module GENER

```
      V(16)= IPTR.EQ.0
      V(17)= COUNTR.LT.10
      V(18)= COUNTR.EQ.0
      V(19)= ACTNUM.GT.NAACTS
      V(20)= CACTNO.GT.APDTAB(75)
      V(21)= APDTAB(77).LT.0
      V(22)= LINUM.EQ.0
      V(23)= CPTR.EQ.0
      V(24)= CACTNO.EQ.3
      V(25)= CACTNO.EQ.7
      V(26)= IACTNO.EQ.0
      RETURN
      END
```

Sub-module GENER

```
CLUSTER:  CHECK TABLE                          RULES:  000000000111111
                                                       123456789012345
 1. .NOT.ATAB(1)                                       Y          NNNN
 2. .NOT.ATAB(2)                                        Y         NNNN
 3. .NOT.ATAB(3)                                         Y        NNNN
 4. .NOT.ATAB(5)                                          Y       NNNN
 5. .NOT.ATAB(6)                                           Y      NNNN
 6. NARPRD.NE.NAPRDS                                        Y     NNNN
 7. NASTRL.NE.NARULE                                         Y    NNNN
 8. NAARL.NE.NARULE                                           Y   NNNN
 9. APDTAB(255).EQ.0                                           Y  NNNN
10. APDTAB(259).EQ.0                                            Y NNNN
11. APDTAB(260).EQ.0                                             YNNNN
12. EXFLAG.EQ.0                                                   YY N
13. EXFLAG.EQ.1                                                   NYYN
                                                       ---------------
STRATEGY
 1. CHECK TABLE                                        111111111112110
 2. CONTROL CODE-1

CLUSTER:  CHECK TABLE                          RULES:  ---------------
                                                       000000000111111
                                                       123456789012345
 1 INITIALISE LOCAL VARIABLES                                      A
 2 WARNING MESSAGE                                     A
 3 WARNING MESSAGE                                      A
 4 WARNING MESSAGE                                       A
 5 ERROR MESSAGE                                          A
 6 ERROR MESSAGE                                           A
 7 WARNING MESSAGE                                          A
 8 WARNING MESSAGE                                           A
 9 WARNING MESSAGE                                            A
10 WARNING MESSAGE                                             A
11 WARNING MESSAGE                                              A
12 WARNING MESSAGE                                               A
13 CLEAR EXECUTE FLAG                                              B
14 SET WARNING FLAG                                    BBB  BBBBBB
15 SET ERROR FLAG                                         BB
16 'CONTINUE?' MESSAGE                                              A
19 'GENERATION SUPPRESSED' MESSAGE                                   A
20 SET START OF BLOCK INDICATOR .TRUE.                            A
25 GET LIST OF CONTROL BLOCK POINTERS FROM DISK                   B
26 POINT TO SYSTEMS SOURCE FILE (NTF=4)                           C
```

Sub-module GENER. Tables.

```
CLUSTER:  CONTROL CODE-1                                    RULES:  0000000
                                                                    1234567
 14. STBLK                                                          YNNNNNN
 15. BLKNO.EQ.0                                                      YYNN
 16. IPTR.EQ.0                                                      N Y YY
 17. COUNTR.LT.10                                                   YNYNYY
 18. COUNTR.EQ.0                                                      N NY
                                                                    -------
STRATEGY
  2. CONTROL CODE-1                                                 2223233
  3. GLOBAL CODE-A                                                   3  344
  4. LOCAL SPECS-A                                                   4  455
  5. CONTROL CODE-2                                                 55  566
  6. ACTIONS
  9. CONTROL CODE-3                                                 99  9EE
  A. GLOBAL CODE-P                                                   A  ABB
  B. LOCAL SPECS-P                                                   B  BCC
  C. PREDICATE CODE                                                  C
  E. CONTROL CODE-4                                                 EE  EAA
                                                                    -------

CLUSTER:  CONTROL CODE-1                                    RULES:  0000000
                                                                    1234567
 20 SET START OF BLOCK INDICATOR .TRUE.                                  DA
 21 SET START OF BLOCK INDICATOR .FALSE.                            B
 22 ZERO COUNTER                                                    C DDCC
 23 ZERO BLOCK NUMBER                                               D
 24 SET IPTR TO START OF CURRENT BLK OF CONTROL CODE                E
 26 POINT TO SYSTEMS SOURCE FILE (NTF=4)                            F
 27 READ CARD INTO BLOCK - UPDATE IPTR                               B
 28 INCREMENT COUNTER                                                A
 29 FETCH TABLE NAME & INSERT IN SUBROUTINE STATEMENT                 AA
 30 INCREMENT BLOCK NUMBER                                            CCBB
 31 WRITE BLOCK TO OUTPUT FILE                                        BBAA
 35 INCREMENT BLSQNO ( CODE BLOCK SEQUENCE NUMBER )                  A
                                                                    -------
```

Sub-module GENER. Tables.

```
CLUSTER:  GLOBAL CODE-A                                   RULES:  0000000
                                                                  1234567

 14. STBLK                                                        NNNNNNY
 15. BLKNO.EQ.0                                                    NN YY
 16. IPTR.EQ.0                                                    N YY Y
 17. COUNTR.LT.10                                                 YNYYNY
 18. COUNTR.EQ.0                                                    NY N

STRATEGY                                                          -------

  2. CONTROL CODE-1                                               2233
  3. GLOBAL CODE-A                                                3344333
  4. LOCAL SPECS-A                                                445544
  5. CONTROL CODE-2                                               5566
  6. ACTIONS
  9. CONTROL CODE-3                                               99EE99
  A. GLOBAL CODE-P                                                AABBAA
  B. LOCAL SPECS-P                                                BBCCBB
  C. PREDICATE CODE                                               C
  E. CONTROL CODE-4                                               EEAA

                                                                  -------

CLUSTER:  GLOBAL CODE-A                                   RULES:  0000000
                                                                  1234567

 20 SET START OF BLOCK INDICATOR .TRUE.                            DA
 21 SET START OF BLOCK INDICATOR .FALSE.                                B
 22 ZERO COUNTER                                                   CC CCD
 23 ZERO BLOCK NUMBER                                                   E
 27 READ CARD INTO BLOCK - UPDATE IPTR                            B
 28 INCREMENT COUNTER                                             A
 30 INCREMENT BLOCK NUMBER                                         BB BB
 31 WRITE BLOCK TO OUTPUT FILE                                     AA AA
 33 POINT TO USER'S SOURCE FILE UNIT 3                                  F
 35 INCREMENT BLSQNO ( CODE BLOCK SEQUENCE NUMBER )                     C
 63 CALL GLOBAL DATA GENERATION UTILITY                                 A
 64 SET IPTR TO HEAD OF GLOBAL STATEMENT CODE ON UNIT 3                 G
 65 OUTPUT 'C GLOBAL VARIABLES....'                                     H

                                                                  -------
```

Sub-module GENER. Tables.

```
CLUSTER:  LOCAL SPECS-A                                  RULES:  0000000
                                                                 1234567

 14. STBLK                                                       NNNNYNN
 15. BLKNO.EQ.0                                                   NN  YY
 16. IPTR.EQ.0                                                   N YY  Y
 17. COUNTR.LT.10                                                YNYY NY
 18. COUNTR.EQ.0                                                   NY  N

STRATEGY                                                         -------

  2. CONTROL CODE-1                                              2233
  3. GLOBAL CODE-A                                               3344 33
  4. LOCAL SPECS-A                                               4455444
  5. CONTROL CODE-2                                              5566
  6. ACTIONS
  9. CONTROL CODE-3                                              99EE 99
  A. GLOBAL CODE-P                                               AABB AA
  B. LOCAL SPECS-P                                               BBCC BB
  C. PREDICATE CODE                                              C
  E. CONTROL CODE-4                                              EEAA

                                                                 -------


CLUSTER:  LOCAL SPECS-A                                  RULES:  0000000
                                                                 1234567

 20 SET START OF BLOCK INDICATOR .TRUE.                            DA
 21 SET START OF BLOCK INDICATOR .FALSE.                             B
 22 ZERO COUNTER                                                 CC CCC
 23 ZERO BLOCK NUMBER                                                D
 27 READ CARD INTO BLOCK - UPDATE IPTR                           B
 28 INCREMENT COUNTER                                            A
 30 INCREMENT BLOCK NUMBER                                        BB  BB
 31 WRITE BLOCK TO OUTPUT FILE                                    AA  AA
 32 SET IPTR TO START OF LOCAL ACTION CODE LIST N FILE 3               A
 33 POINT TO USER'S SOURCE FILE UNIT 3                                 E
 35 INCREMENT BLSUNO ( CODE BLOCK SEQUENCE NUMBER )                    F
 36 OUTPUT 'C LOCAL VARIABLES....'                                     G

                                                                 -------
```

Sub-module GENER. Tables.

```
CLUSTER:  CONTROL CODE-2                                      RULES:  0000000
                                                                      1234567

 20 SET START OF BLOCK INDICATOR .TRUE.                                  DA
 21 SET START OF BLOCK INDICATOR .FALSE.                              B
 22 ZERO COUNTER                                                      C CC DD
 23 ZERO BLOCK NUMBER                                                 D
 24 SET IPTR TO START OF CURRENT BLK OF CONTROL CODE                  E
 26 POINT TO SYSTEMS SOURCE FILE (NTF=4)                              F
 27 READ CARD INTO BLOCK - UPDATE IPTR                                 B
 28 INCREMENT COUNTER                                                  A
 30 INCREMENT BLOCK NUMBER                                              BB CC
 31 WRITE BLOCK TO OUTPUT FILE                                          AA BB
 35 INCREMENT BLSQNO ( CODE BLOCK SEQUENCE NUMBER )                   A
 37 INSERT PREDICATE SUBROUTINE NAME INTO BLOCK                             AA
                                                                      -------

CLUSTER:  CONTROL CODE-2                                      RULES:  0000000
                                                                      1234567
14. STBLK                                                             YNNNNNN
15. BLKNO.EQ.0                                                          NN YY
16. IPTR.EQ.0                                                          N YY Y
17. COUNTR.LT.10                                                       YNYYNY
18. COUNTR.EQ.0                                                           NY N
                                                                      -------
STRATEGY

 2. CONTROL CODE-1                                                    22233
 3. GLOBAL CODE-A                                                      3344
 4. LOCAL SPECS-A                                                      4455
 5. CONTROL CODE-2                                                    5556655
 6. ACTIONS
 9. CONTROL CODE-3                                                    999EE
 A. GLOBAL CODE-P                                                      AABB
 B. LOCAL SPECS-P                                                      BBCC
 C. PREDICATE CODE                                                     C
 E. CONTROL CODE-4                                                    EEEAA
                                                                      -------
```

Sub-module GENER. Tables.

```
CLUSTER:  ACTIONS                                          RULES:  00000
                                                                   12345

 14. STBLK                                                         YNNNN
 16. IPTR.EQ.0                                                      YN Y
 19. ACTNUM.GT.NAACTS                                               NNYN
 26. IACTNO.EQ.0                                                    N  Y

STRATEGY                                                           -----

  6. ACTIONS                                                       67896
  7. SIMPLE ACTION
  8. COMPOUND ACTION
  9. CONTROL CODE-3
                                                                   -----
CLUSTER:  ACTIONS                                          RULES:  00000
                                                                   12345

 20 SET START OF BLOCK INDICATOR .TRUE.                              GA
 21 SET START OF BLOCK INDICATOR .FALSE.                           B
 33 POINT TO USER'S SOURCE FILE UNIT 3                              AA
 35 INCREMENT BLSQNO ( CODE BLOCK SEQUENCE NUMBER )                C
 38 OUTPUT COMPUTED GOTO NUMBERS                                   A
 39 USE IPTR TO IDENTIFY ACTION, SIMPLE VS COMPOUND                G   C
 40 OUTPUT ACTION HEADER LABEL STATEMENT                            FF
 41 SET IACTNO = CURRENT INTERNAL ACTION NUMBER                    FBB B
 42 READ A LINE OF CODE FROM UNIT 3                                 DD
 43 SET IPTR = UNIT 3 LOCATION OF SOURCE CODE FOR THIS A            CC
 44 OUTPUT LINE OF CODE                                             EE
 58 ZERO ACTION  NUMBER ACTNUM                                     D
 59 INCREMENT ACTNUM                                               E   A

STRATEGY                                                           -----

  6. ACTIONS                                                       67896
  7. SIMPLE ACTION
  8. COMPOUND ACTION
  9. CONTROL CODE-3
                                                                   -----
```

Sub-module GENER. Tables.

```
CLUSTER:  SIMPLE ACTION                                   RULES:  00
                                                                  12
16. IPTR.EQ.0                                                     YN
                                                                  --
STRATEGY

 6. ACTIONS
 7. SIMPLE ACTION                                                 67
                                                                  --


CLUSTER:  SIMPLE ACTION                                   RULES:  00
                                                                  12
 39 USE IPTR TO IDENTIFY ACTION, SIMPLE VS COMPOUND               D
 41 SET IACTNO = CURRENT INTERNAL ACTION NUMBER                   C
 42 READ A LINE OF CODE FROM UNIT 3                                A
 44 OUTPUT LINE OF CODE                                            B
 45 OUTPUT ACTION TRAILER STATEMENT                               A
 59 INCREMENT ACTNUM                                              B
                                                                  --
STRATEGY

 6. ACTIONS
 7. SIMPLE ACTION                                                 67
                                                                  --
```

Sub-module GENER. Tables.

```
CLUSTER:  COMPOUND ACTION                               RULES:  000000000111111
                                                                123456789012345

14.  STBLK                                                      YNNNNNNNNNNNNNN
16.  IPTR.EQ.0                                                    NY   N YY
17.  COUNTR.LT.10                                                        YNYY
18.  COUNTR.EQ.0                                                    /     NY
20.  CACTNO.GT.APDTAB(75)                                        NNNNNNNNNYNNNN
21.  APDTAB(77).LT.0                                             YYY Y      NNNN
22.  LINUM.EQ.0                                                  YNN NNNNN  YN Y
23.  CPTR.EQ.0                                                   NNNYNNNNN
24.  CACTNO.EQ.3                                                  NN YNNNN  NNYN
25.  CACTNO.EQ.7                                                  NN NYYYY  NNNY

STRATEGY                                                        ---------------

 6.  ACTIONS
 8.  COMPOUND ACTION                                            888888888868888

                                                                ---------------
```

Sub-module GENER. Tables.

CLUSTER: COMPOUND ACTION

		RULES: 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
21	SET START OF BLOCK INDICATOR .FALSE.	C														
22	ZERO COUNTER		F						D	D			C		E	F
23	ZERO BLOCK NUMBER		G										D		F	G
26	POINT TO SYSTEMS SOURCE FILE (NTF=4)	G														
27	READ CARD INTO BLOCK - UPDATE IPTR							B								
28	INCREMENT COUNTER							A								
30	INCREMENT BLOCK NUMBER								C	C						
31	WRITE BLOCK TO OUTPUT FILE								B	B						
33	POINT TO USER'S SOURCE FILE UNIT 3											E				
39	USE IPTR TO IDENTIFY ACTION, SIMPLE VS COMPOUND											D				
40	OUTPUT ACTION HEADER LABEL STATEMENT		D												D	D
41	SET IACTNO = CURRENT INTERNAL ACTION NUMBER											C				
42	READ A LINE OF CODE FROM UNIT 3		B	A											B	B
44	OUTPUT LINE OF CODE		C	B											C	C
45	OUTPUT ACTION TRAILER STATEMENT	E			A		C				B				I	
46	ZERO COMPOUND ACTION COUNTER	A														
47	INCREMENT COMPOUND ACTION COUNTER	B			C	A	D				C			B	J	
48	REMEMBER NAME OF TABLE BEING CALLED	D														
49	MARK COMPOUND ACTION AS 'USED'											A				
50	UPDATE INTERNAL ACTION NUMBER IACTNO TO NEXT ACTION	F			D	C	E				D			D	K	
51	SET CPTR TO HEAD OF SOURCE CODE FOR THIS ACTION	I			E	B	F				E			C	L	
52	SET LOCATION OF START OF COMPOUND ACTION LIST	H														
53	UPDATE IPTR FROM CPTR		A										A	A	A	A
54	ZERO LINUM	J			B		B				A			E	H	
55	INCREMENT LINUM		E					C					B			E
56	LOCATE FILE NUMBER FROM TABLE NAME (TFN)	K														
57	OUTPUT COMPOUND ACTION 'NTF=TFN' STATEMENT						A								G	
59	INCREMENT ACTNUM											B				
60	INSERT LABELS, TABLE NAME INTO MINI-DRIVER & WRITE 1								A	A						

STRATEGY

6. ACTIONS
8. COMPOUND ACTION

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		8	8	8	8	8	8	8	8	8	8	8	8	8	8	8

Sub-module GENER. Tables.

```
CLUSTER:  CONTROL CODE-3                                        RULES:  0000000
                                                                        1234567

 14. STBLK                                                              YNNNNNN
 15. BLKNO.EQ.0                                                           NN YY
 16. IPTR.EQ.0                                                           N YY Y
 17. COUNTR.LT.10                                                        YNYYNY
 18. COUNTR.EQ.0                                                           NY N

STRATEGY                                                                -------

  2. CONTROL CODE-1                                                     22233
  3. GLOBAL CODE-A                                                       334433
  4. LOCAL SPECS-A                                                       445544
  5. CONTROL CODE-2                                                     55566
  6. ACTIONS
  9. CONTROL CODE-3                                                     999EE99
  A. GLOBAL CODE-P                                                       AABBAA
  B. LOCAL SPECS-P                                                       BBCCBB
  C. PREDICATE CODE                                                      C
  E. CONTROL CODE-4                                                     EEEAA

                                                                        -------


CLUSTER:  CONTROL CODE-3                                        RULES:  0000000
                                                                        1234567

 20 SET START OF BLOCK INDICATOR .TRUE.                                     DA
 21 SET START OF BLOCK INDICATOR .FALSE.                                B
 22 ZERO COUNTER                                                        C CC CC
 23 ZERO BLOCK NUMBER                                                   D
 24 SET IPTR TO START OF CURRENT BLK OF CONTROL CODE                    E
 26 POINT TO SYSTEMS SOURCE FILE (NTF=4)                                F
 27 READ CARD INTO BLOCK - UPDATE IPTR                                   B
 28 INCREMENT COUNTER                                                    A
 30 INCREMENT BLOCK NUMBER                                                BB BB
 31 WRITE BLOCK TO OUTPUT FILE                                            AA AA
 35 INCREMENT BLSQNO ( CODE BLOCK SEQUENCE NUMBER )                     A

                                                                        -------
```

Sub-module GENER. Tables.

```
CLUSTER:  GLOBAL CODE-P                                   RULES:  0000000
                                                                  1234567

 14. STBLK                                                        NNNNNNY
 15. BLKNO.EQ.0                                                    NN YY
 16. IPTR.EQ.0                                                    N YY Y
 17. COUNTR.LT.10                                                 YNYYNY
 18. COUNTR.EQ.0                                                    NY N

STRATEGY                                                          -------

  2. CONTROL CODE-1                                               2233
  3. GLOBAL CODE-A                                                334433
  4. LOCAL SPECS-A                                                445544
  5. CONTROL CODE-2                                               5566
  6. ACTIONS
  9. CONTROL CODE-3                                               99EE99
  A. GLOBAL CODE-P                                                AABBAAA
  B. LOCAL SPECS-P                                                BBCCBB
  C. PREDICATE CODE                                               C
  E. CONTROL CODE-4                                               EEAA

                                                                  -------


CLUSTER:  GLOBAL CODE-P                                   RULES:  0000000
                                                                  1234567

 20 SET START OF BLOCK INDICATOR .TRUE.                             DA
 21 SET START OF BLOCK INDICATOR .FALSE.                                  A
 22 ZERO COUNTER                                                   CC CCC
 23 ZERO BLOCK NUMBER                                                    D
 27 READ CARD INTO BLOCK - UPDATE IPTR                            B
 28 INCREMENT COUNTER                                             A   A
 30 INCREMENT BLOCK NUMBER                                         BB BB
 31 WRITE BLOCK TO OUTPUT FILE                                     AA AA
 33 POINT TO USER'S SOURCE FILE UNIT 3                                   E
 35 INCREMENT BLSQNO ( CODE BLOCK SEQUENCE NUMBER )                      B
 64 SET IPTR TO HEAD OF GLOBAL STATEMENT CODE ON UNIT 3                  F
 65 OUTPUT 'C GLOBAL VARIABLES....'                                      G

                                                                  -------
```

Sub-module GENER. Tables.

```
CLUSTER:  LOCAL SPECS-P                                  RULES:  0000000
                                                                 1234567

 14. STBLK                                                       NNNNNNY
 15. BLKNO.EQ.0                                                   NN YY
 16. IPTR.EQ.0                                                   N YY Y
 17. COUNTR.LT.10                                                YNYYNY
 18. COUNTR.EQ.0                                                   NY N
                                                                 -------
STRATEGY

  2. CONTROL CODE-1                                              2233
  3. GLOBAL CODE-A                                               334433
  4. LOCAL SPECS-A                                               445544
  5. CONTROL CODE-2                                              5566
  6. ACTIONS
  9. CONTROL CODE-3                                              99EE99
  A. GLOBAL CODE-P                                               AABBAA
  B. LOCAL SPECS-P                                               BBCCBBB
  C. PREDICATE CODE                                              C
  E. CONTROL CODE-4                                              EEAA
                                                                 -------

CLUSTER:  LOCAL SPECS-P                                  RULES:  0000000
                                                                 1234567

 20 SET START OF BLOCK INDICATOR .TRUE.                            DA
 21 SET START OF BLOCK INDICATOR .FALSE.                               B
 22 ZERO COUNTER                                                  CC CCC
 23 ZERO BLOCK NUMBER                                                  D
 27 READ CARD INTO BLOCK - UPDATE IPTR                           B
 28 INCREMENT COUNTER                                            A
 30 INCREMENT BLOCK NUMBER                                        BB BB
 31 WRITE BLOCK TO OUTPUT FILE                                    AA AA
 33 POINT TO USER'S SOURCE FILE UNIT 3                                 E
 35 INCREMENT BLSQNO ( CODE BLOCK SEQUENCE NUMBER )                    F
 36 OUTPUT 'C LOCAL VARIABLES....'                                     G
 62 SET IPTR TO START OF PREDICATE LOCAL VARIABLE                      A
                                                                 -------
```

Sub-module GENER. Tables.

```
CLUSTER:  PREDICATE CODE                                  RULES:  00000
   4                                                              12345

 14. STBLK                                                        NYNNN
 16. IPTR.EQ.0                                                    N  YY
 17. COUNTR.LT.10                                                 Y N
 18. COUNTR.EQ.0                                                     NY
                                                                  -----
STRATEGY

  2. CONTROL CODE-1                                               2
  3. GLOBAL CODE-A                                                3
  4. LOCAL SPECS-A                                                4
  5. CONTROL CODE-2                                               5
  9. CONTROL CODE-3                                               9
  A. GLOBAL CODE-P                                                A
  B. LOCAL SPECS-P                                                B
  C. PREDICATE CODE                                               CCCDD
  D. LIST FILE INFO
  E. CONTROL CODE-4                                               E
                                                                  -----


CLUSTER:  PREDICATE CODE                                  RULES:  00000
                                                                  12345

 20 SET START OF BLOCK INDICATOR .TRUE.                              EB
 21 SET START OF BLOCK INDICATOR .FALSE.                          B
 22 ZERO COUNTER                                                  CDDA
 23 ZERO BLOCK NUMBER                                             D
 27 READ CARD INTO BLOCK - UPDATE IPTR                            B
 28 INCREMENT COUNTER                                             A
 30 INCREMENT BLOCK NUMBER                                          CC
 31 WRITE BLOCK TO OUTPUT FILE                                      BB
 33 POINT TO USER'S SOURCE FILE UNIT 3                            E
 35 INCREMENT BLSQNO ( CODE BLOCK SEQUENCE NUMBER )               F
 66 SET IPTR TO START OF PREDICATE SOURCE CODE ON UNIT 3          A
 67 INSERT 'V(..)=' INTO BLOCK                                      AA
 68 OUTPUT 'RETURN' & 'END' STATEMENTS                               FC
 70 ZERO PREDICATE COUNTER                                        G
                                                                  -----
```

Sub-module GENER. Tables.

Sub-module GENER. Tables.

CLUSTER: LIST FILE INFO	RULES:	01
14. STBLK		Y
		-
STRATEGY		
D. LIST FILE INFO		0
		-

CLUSTER: LIST FILE INFO	RULES:	01
69 LIST FILE USAGE TABLE WITH MODIFICATION FLAGS		A
		-
STRATEGY		
D. LIST FILE INFO		0
		-

252

```
CLUSTER:  CONTROL CODE-4                                  RULES:  0000000
                                                                  1234567

14. STBLK                                                         YNNNN
15. BLKNO.EQ.0                                                     NN YY
16. IPTR.EQ.0                                                     N YY Y
17. COUNTR.LT.10                                                  YNYYNY
18. COUNTR.EQ.0                                                     NY N

STRATEGY                                                          -------

 2. CONTROL CODE-1                                                22233
 3. GLOBAL CODE-A                                                  3344
 4. LOCAL SPECS-A                                                  4455
 5. CONTROL CODE-2                                                55566
 6. ACTIONS
 9. CONTROL CODE-3                                                999EE
 A. GLOBAL CODE-P                                                  AABB
 B. LOCAL SPECS-P                                                  BBCC
 C. PREDICATE CODE                                                 C
 E. CONTROL CODE-4                                                EEAAAEA
                                                                  -------


CLUSTER:  CONTROL CODE-4                                  RULES:  0000000
                                                                  1234567

20 SET START OF BLOCK INDICATOR .TRUE.                              DA E
21 SET START OF BLOCK INDICATOR .FALSE.                           B
22 ZERO COUNTER                                                   C CC DD
23 ZERO BLOCK NUMBER                                              D
24 SET IPTR TO START OF CURRENT BLK OF CONTROL CODE                E
26 POINT TO SYSTEMS SOURCE FILE (NTF=4)                           F
27 READ CARD INTO BLOCK - UPDATE IPTR                              B
28 INCREMENT COUNTER                                               A
30 INCREMENT BLOCK NUMBER                                           BB CC
31 WRITE BLOCK TO OUTPUT FILE                                       AA BB
35 INCREMENT BLSUNO ( CODE BLOCK SEQUENCE NUMBER )                A
61 INSERT PREDICATE SUBROUTINE NAME INTO BLOCK                          AA
                                                                  -------
```

Sub-module GENER. Tables.

APPENDIX C

UTILITY PROGRAMS

Certain functions have been filled by means of utility procedures. These are accessed by conventional Fortran subroutine CALLS. They are written in either Fortran or 370 Assembler and their source listings follow. They fall into three categories.

1. General purpose utilities

STRING	CONVRT	UNPAK
DNULL		

2. Data Dictionary manipulation

DDINIT	AD2LST	ADDVAR
GETBL3	MODTAB	ANALYS
PUSH	GET	FLUSH
NONBLK	POP	ADDTAB

3. Source code generation

DDSGEN	DEGLOB	SORT
BLDVAR	ADDTYP	ADDCOM
PRTCOM	PRTTYP	MOVCH
ADDIGS		

STRING - 370 Assembler - Used widely throughout system

Isolates and returns a string of characters delimited by commas or blanks, from an 80 byte card image. The position of the start of the next character string (if any) is also determined. This is used throughout the system for handling user entries.

CONVRT - 370 Assembler - Used widely throughout system

Translates a string of display-coded (EBCDIC) digits into its binary integer value.

UNPAK - Fortran - Used widely throughout system

Translates a binary integer number into a string of display-coded (EBCDIC) digits.

DNULL - 370 Assembler - Used widely throughout system

Zeros an array.

DDINIT - Fortran - Called from BLDCOD

Initialises the data dictionary (for a new project) by clearing the type table lists (block 2) and setting up the list of permitted data types (block 1).

AD2LST - Fortran - Called from BLDCOD

Adds a reference to a global data item to a table's variable list.

ADDVAR - Fortran - Called from BLDCOD

Searches the global variables list for existence of a data item; adding it to the list if it does not.

GETBL3 - Fortran - Called from ADDVAR

Fetches a block containing the variables list into memory from the data dictionary file (unit 8).

MODTAB - Fortran - Called from BLDCOD

If changes have been made to the variables list, all tables referencing elements of the type changed, are marked for code re-generation.

ANALYS - Fortran - Called from BLDCOD

A simple table-driven parser which analyses a string of characters on a user's global specification statement, according to the following rules.

```
1   #      ::=   name   diminfo

2   name   ::=   ident
```

```
3   diminfo  ::=   (  dims
7            ::=
10           ::=   ,
5   dims     ::=   num  rest
6   rest     ::=   ,  dims
4            ::=   )  end
8   end      ::=   ,
9            ::=
```

Values for element name, length in words, and dimension information (EBCDIC) are returned.

PUSH - Fortran - Called from ANALYS
POP - Fortran - Called from ANALYS

Stack maintenance utilities.

GET - 370 Assembler - Called from ANALYS

Extracts and identifies tokens which are returned to ANALYS

FLUSH - Fortran - Called from ANALYS.

Flushes the stack when an illegal character sequence is detected, and prints the unidentifiable string.

NONBLK - Fortran - Called from ANALYS and FLUSH

Locates the start of the next non-blank sequence of characters on the input card.

ADDTAB - Fortran - Called from BLDCOD

Enters the table's file number in the table list for the current data type.

DDSGEN - Fortran - Called from GENER

Generates Fortran source specification statements from global data elements in the data dictionary.

DEGLOB - Fortran - Called from DDSGEN

Clears previously generated global specification statements from the source code file, before re-generation proceeds.

SORT - Fortran - Called from DDSGEN

Bubble sort utility.

BLDVAR - Fortran - Called from DDSGEN

Builds an identifier (for either a type or a Common statement) from character strings (s,t) and numerics (n,m) if non-zero. The resulting identifier appears as st[n][(m)] where items enclosed in square brackets are optional.

ADDTYP - Fortran - Called from DDSGEN

Adds an identifier to an existing type card, creating a new card if the current one overflows.

ADDCOM - Fortran - Called from DDSGEN

As for ADDTYP, but to a Common card.

PRTCOM - Fortran - Called from DDSGEN and ADDCOM

Writes a COMMON statement in the source code file (unit 3), chaining it to any previous ones.

PRTTYP - Fortran - Called from DDSGEN and ADDTYP

As above for a type specification statement.

MOVCH - Fortran - Called from BLDVAR

Moves non-blank characters into the identifier being built.

ADDIGS - Fortran - Called from BLDVAR

Converts a binary integer into display-coded (EBCDIC) characters and moves them into the identifier being built.

```
STMT   SOURCE STATEMENT                                        ASM 0201 13.58 05/01/80

  UTILS   CSECT
* SUBROUTINE TO ISOLATE A CHARACTER STRING & LOCATE START OF NEXT
* INVOKED BY: CALL STRING(CARD,COLUMN,CHARST,MORE,LAST,NCHARS)
* COLUMN IS START LOCATION OF SEARCH,  MODIFIED AFTER EACH FIND.
* CHARST IS NCHARS BYTES TO RECEIVE THE STRING.
* LAST IS THE COLUMN NUMBER OF THE LAST NON-BLANK CHARACTER LOCATED
* MORE RETURNS 0 IF THERE IS MORE ON THE CARD.    ELSE, 1.
          ENTRY  STRING
          DC     CL7'STRING'
          DC     X'7'
  STRING  STM    14,12,12(13)
          BALR   11,0
          USING  *,11
          LM     3,7,4(1)          ADDR COLUMN,CHARST,MORE,LAST,NCHARS
          L      14,0(7)            NO OF CHARS
          MVI    0(4),X'40'        BLANK OUT FIRST CHAR.
          BCTR   14,0
          BCTR   14,0
          EX     14,BLSTR           BLANK OUT REST OF CHARST
          L      7,0(1)            ADDR CARD
          LR     10,7
          LR     12,3
          L      3,0(3)            COLUMN NUMBER TO START
          BCT    3,N1              REDUCE COLUMN NUMBER BY ONE
  N1      AR     7,3               START LOCATION ON CARD
          LA     9,72
          SR     9,3               NUMBER OF CHARACTERS TO BE CHECKED
          EX     9,SCHBLK          SCAN FOR PRECEDING BLANKS
          BC     7,NOTBLK
          LA     8,1               CARD IS BLANK
          ST     8,0(5)            SET NO-MORE-CHARS FLAG
          BC     15,ENDUP
  NOTBLK  LR     7,1
          LR     3,7
          SR     3,10
          LA     9,0(14,7)            SET LIMIT ON NUMBER OF NON-BLANK CHARS
```

Utility subroutine STRING

```
STMT   SOURCE STATEMENT                                      ASM 0201 13.58 05/01/80

 37             LA     8,1
 38 N3          CLI    0(7),X'6B'
 39             BC     8,ENDWRD
 40             CLI    0(7),X'40'
 41             BC     8,ENDWRD
 42             MVC    0(1,4),0(7)    MOVE CHARACTER INTO STRING
 43             LA     4,1(4)          NEXT CHARACTER IN STRING
 44             LA     3,1(3)
 45             BXLE   7,8,N3
 46 * NOT YET FOUND END OF STRING - SPACE OVER NON-BLANK CHARS.
 47             LR     9,10
 48             LA     9,71(9)
 49 N4          CLI    0(7),X'6B'      LOOP TO END OF STRING
 50             BC     8,ENDWRD
 51             CLI    0(7),X'40'
 52             BC     8,ENDWRD
 53             LA     3,1(3)          KEEP TRACK OF POSITION
 54             BXLE   7,8,N4
 55 * NON-BLANK CHARS TO END OF CARD
 56             LA     8,1             SET NO-MORE-CHARS FLAG
 57             ST     8,0(5)
 58             BC     15,ENDUP
 59 ENDWRD      ST     3,0(6)          STORE POSITION OF LAST NON-BLANK
 60             LA     9,71
 61             SR     9,3             NUMBER OF CHARS TO BE CHECKED
 62             EX     9,SCHBLK        LOOK FOR NEXT NON-BLANK
 63             BC     7,N5
 64             LA     8,1             REST OF CARD IS BLANK
 65             ST     8,0(5)
 66             BC     15,ENDUP
 67 N5          SR     1,10
 68             LA     1,1(1)
 69             ST     1,0(12)
 70             SR     1,1
 71             ST     1,0(5)
 72 ENDUP       LM     14,12,12(13)
```

Utility subroutine STRING

```
STMT    SOURCE STATEMENT                                          ASM 0201 13.58 05/01/80

  73               MVI    12(13),X'FF'
  74               BALR   15,14
  75               DS     F
  76 BLSTR         MVC    1(0,4),0(4)
  77 SCHBLK        TRT    0(0,7),TAB1
  78 TAB1          DC     74X'0',7X'1',9X'0',8X'1',10X'0',4X'1'
  79               DC     10X'0',6X'1',X'0',9X'1',7X'0',9X'1',8X'0'
  80               DC     8X'1',23X'0',9X'1',7X'0',9X'1',8X'0',8X'1'
  81               DC     6X'0',10X'1'
  82               END
```

Utility subroutine STRING

```
STMT   SOURCE STATEMENT                                              ASM 0201 13.59 05/01/80

   1 UTILS    CSECT
   2 * SUBROUTINE TO CONVERT A DISPLAY CODED STRING OF DIGITS TO ITS INTEGER
   3 *        VALUE
   4 * INVOKED BY:  CALL CONVRT(DISPL,NUMBR,VALID)
   5 * DISPL - 4 BYTE WORD CONTAINING EBCDIC CHARS
   6 * NUMBR - 4 BYTE WORD TO RECEIVE NUMERIC VALUE
   7 * VALID - VALIDITY INDICATOR.     0=OK    1=NOT
   8          ENTRY CONVRT
   9          DC    CL7'CONVRT'
  10          DC    X'7'
  11 CONVRT   STM   14,12,12(13)
  12          BALR  11,0
  13          USING *,11
  14          LM    8,10,0(1)
  15          LA    7,3(8)
  16          LA    6,1
  17          SR    3,3
  18          SR    4,4
  19          SR    5,5
  20          ST    5,0(10)          SET VALID FLAG
  21          ST    5,0(9)           ZERO OUT NUMBER
  22          TRT   0(1,8),TABLE
  23          BC    2,INVAL
  24 NEXT     PACK  LAST(1),0(1,8) CONVERT CHAR: (1) TO P.D.
  25          CVB   3,CHAR                         (2) TO BINARY
  26          AR    5,3              ADD DIGIT INTO TOTAL
  27          BXH   8,6,ENDNUM
  28          TRT   0(1,8),TABLE     TST NEXT CHARACTER
  29          BC    2,ENDNUM
  30          M     4,=F'10'
  31          BC    15,NEXT
  32 INVAL    ST    6,0(10)          SET INVALID FLAG
  33          SR    5,5
  34          ST    5,0(9)           RETURN ZERO VALUE
  35          BC    15,ENDIT
  36 ENDNUM   CLI   0(8),X'40'       MAKE SURE NEXT CHARACTER IS BLANK ...
```

Utility subroutine CONVRT

```
STMT   SOURCE STATEMENT                                    ASM 0201 13.59 05/01/80

 37            BC      8,STFLG            .....IT WAS .....
 38            CLI     0(8),X'00'      ... OR NULL ...
 39            BC      7,INVAL            .....IT WASN'T
 40 STFLG      ST      5,0(9)          RETURN VALUE
 41 ENDIT      LM      14,12,12(13)
 42            MVI     12(13),X'FF'
 43            BALR     15,14
 44            DS      D
 45 CHAR       DC      X'00000000000000'
 46 LAST       DC      X'00'
 47 TABLE      DC      240X'1',10X'0',6X'1'
 48            END
 49                    =F'10'
```

Utility subroutine CONVRT

```
RELEASE 2.0                  UNPAK                DATE = 80122          14/00/23

          INTEGER FUNCTION UNPAK(N)
C TRANSLATES A BINARY INTEGER VALUE (N) INTO A STRING OF EBCDIC CHARS.
C       MAXIMUM LENGTH IS FOUR DECIMAL DIGITS.
          DIMENSION NDIGS(3)
          LOGICAL*1 LDIGS(12),LWORD(4)
          EQUIVALENCE (NDIGS(1),LDIGS(1)),(NWORD,LWORD(1))
          DATA NDIGS/'0123','4567','89'/,NBLNK/' '/
          IF(N.GE.0.AND.N.LE.9999) GO TO 10
          UNPAK=NBLNK
          WRITE(6,99) N
   99     FORMAT(' N = ',I10,' OUT OF RANGE')
          RETURN
   10     NWORD=NBLNK
          IF(N.GT.0) GO TO 20
          LWORD(4)=LDIGS(1)
          GO TO 40
   20     K=N
          DO 30 I=1,4
            IF(K.EQ.0) GO TO 40
            J=K-K/10*10
            LWORD(5-I)=LDIGS(J+1)
   30       K=K/10
   40     UNPAK=NWORD
          RETURN
          END
```

Utility subroutine UNPAK

```
RELEASE 2.0                DDINIT            DATE = 80122           14/00/23

      SUBROUTINE DDINIT(IBLK1)
C INITIALISES THE DATA DICTIONARY.
      INTEGER IBLK1(100),TYPLST(65)
C LIST OF PERMITTED DATA TYPES
      DATA TYPLST/'INTE','GER',' ','I4',1,'REAL',' ',' ','R4',2,
     +'LOGI','CAL',' ','L4',3,'INTE','GER*','2','I2',4,
     +'LOGI','CAL*','1','L1',5,'REAL','*8',' ','R8',6,
     +'COMP','LEX',' ','C8',7,'COMP','LEX*','16','C6',8,
     +'LOGI','CAL*','4',' ',3,'INTE','GER*','4',' ',1,
     +'REAL','*4',' ',' ',2,'COMP','LEX*','8',' ',7,
     +4*' ',6/
C CLEAR THE TYPE TABLE LISTS
      CALL DNULL(IBLK1,50,1)
      WRITE(8'2) IBLK1
C SET UP THE LIST OF TYPES
      DO 5 I=1,65
 5      IBLK1(I)=TYPLST(I)
      DO 10 I=66,100
 10     IBLK1(I)=0
      IBLK1(67)=999
      IBLK1(68)=999
      WRITE(8'1) IBLK1
      RETURN
      END
```

Utility subroutine DDINIT

```
RELEASE 2.0                 AD2LST                DATE = 80122             14/00/23

      SUBROUTINE AD2LST(LISTRT,ISEQ,IOFFST,IBLK4,IBLK1,TFNO)
C ADDS A REFERENCE TO A DATA ITEM TO THIS TABLE'S VARIABLE LIST (IF ANY)
      INTEGER IBLK4(100),IBLK1(100),TFNO
C DOES THIS TABLE ALREADY HAVE A VARIABLE LIST?....
      IF(LISTRT.GT.0) GO TO 10
C     ...NO... READ THE TABLE LIST
       READ(8'3) IBLK4
       NTABS=IBLK4(4)
       IB4POS=NTABS*4+3
C MARK THE TABLE AS "DATA DICTIONARY ALTERED"
      DO 3 I=7,IB4POS,4
        IF(IBLK4(I).NE.TFNO) GO TO 3
        IBLK4(I+1)=1
        GO TO 4
3       CONTINUE
       WRITE(8'3) IBLK4
C CLEAR A BLOCK TO START THE LIST
4      CALL DNULL(IBLK4,50,1)
C FIND SOMEWHERE TO STORE IT
       DO 5 I=69,100
         IF(IBLK1(I).NE.0) GO TO 5
           IBLK1(I)=999
           WRITE(8'1) IBLK1
           LISTRT=I-65
           GO TO 20
5        CONTINUE
       WRITE(6,99)
99    FORMAT(' NO SPACE FOR TABLE ',A4,A2,' VARIABLE LIST ON UNIT 8')
      RETURN
C   ...YES... READ THE BLOCK CONTAINING IT
10    READ(8'LISTRT) IBLK4
C STORE THE REFERENCE IN THE VARIABLE LIST
20    DO 30 I=1,100
        IF(IBLK4(I).NE.0) GO TO 30
```

Utility subroutine AD2LST

```
RELEASE 2.0                AD2LST              DATE = 80122           14/00/23

            IBLK4(I)=ISEQ*1000+IOFFST
            WRITE(8'LISTRT) IBLK4
            RETURN
   30     CONTINUE
          WRITE(6,98)
   98     FORMAT(' NO SPACE IN VARIABLE LIST FOR ITEM')
          RETURN
          END
```

Utility subroutine AD2LST.

```
RELEASE 2.0                ADDVAR                 DATE = 80122          14/00/23

         SUBROUTINE ADDVAR(VARNAM,CHARST,IBLK1,TYPNO,LGTH,ISEQ,IOFFST,IBLK3
       +,MODFY)
C ADDS A DATA ITEM (NAME,LENGTH,DIMENSION INFO) TO THE GLOBAL VARIABLES
C        LIST
         INTEGER VARNAM(2),CHARST(4),IBLK1(100),IBLK3(100),TYPNO
         ISEQ=0
         I3P=66
C SEARCH EACH BLOCK CONTAINING THE VARIABLES LIST
         DO 100 I=1,35
C GET THE NEXT BLOCK IN SEQUENCE
           CALL GETBL3(IBLK1,I3P,NEWBL,IBLK3)
           ISEQ=ISEQ+1
C LOOK FOR EXISTENCE OF THIS VARIABLE NAME
           DO 15 J=1,89,8
             IOFFST=J
             IF(IBLK3(J).NE.0) GO TO 10
C WE'RE AT THE LIST'S END, SO ADD THE ITEM AND MARK THIS TABLE "ALTERED"
               IBLK3(J)=VARNAM(1)
               IBLK3(J+1)=VARNAM(2)
               IBLK3(J+2)=TYPNO
               IBLK3(J+3)=LGTH
               DO 5 K=4,7
 5               IBLK3(J+K)=CHARST(K-3)
               WRITE(8'I3P-65) IBLK3
               IF(NEWBL.GT.0) WRITE(8'1) IBLK1
               MODFY=1
               RETURN
CHECK AGAINST EXISTING VARIABLE LIST
 10          IF(IBLK3(J).NE.VARNAM(1)) GO TO 15
             IF(IBLK3(J+1).NE.VARNAM(2)) GO TO 15
C               VARIABLE ALREADY DEFINED
             RETURN
 15          CONTINUE
C VARIABLE NOT FOUND IN THIS BLOCK
```

Utility subroutine ADDVAR

```
RELEASE 2.0                ADDVAR                DATE = 80122          14/00/23

  100     CONTINUE
          WRITE(6,99)
   99     FORMAT(' LOOPING IN BLOCK-3 LIST IN ADDVAR')
          RETURN
          END
```

Utility subroutine ADDVAR.

```
RELEASE 2.0              GETBL3              DATE = 80122          14/00/23
        SUBROUTINE GETBL3(IBLK1,I3P,NEWBL,IBLK3)
C BRINGS INTO MEMORY A BLOCK CONTAINING ALL/PART OF THE VARIABLES
C       LIST (BLOCK TYPE 3)
        INTEGER IBLK1(100),IBLK3(100)
        NEWBL=0
C IS THER ONE TO GET, OR DO WE USE A NEW BLOCK?...
        IF(IBLK1(I3P).LE.0) GO TO 50
         I3P=IBLK1(I3P)
         JBL=I3P-65
         READ(8'JBL) IBLK3
         RETURN
C ...NEW BLOCK 3 REQUIRED
 50     DO 5 I=69,100
          IF(IBLK1(I).NE.0) GO TO 5
            NEWBL=I
            IBLK1(NEWBL)=-1
            IBLK1(I3P)=NEWBL
            I3P=NEWBL
            CALL DNULL(IBLK3,50,1)
            RETURN
 5        CONTINUE
        WRITE(6,99)
 99     FORMAT(' NO BLOCK-3 SPACE AVAILABLE IN DATA DICTIONARY')
        RETURN
        END
```

Utility subroutine GETBL3

RELEASE 2.0 FLUSH DATE = 80122 14/00/23

```
      SUBROUTINE FLUSH(LCHARS,LCARD,NCOL,STACK,MSTK,NCHARS,
C FLUSHES THE STACK AND PRINTS THE STRING EXTRACTED TO DATE.
     +    MORE,/NN/,NBLNK,NCOM,/LCHAR/,LSTRNG)
      INTEGER STACK(MSTK)
      LOGICAL*1 LCHARS(16),LCARD(80),LCHAR,LSTRNG(12)
C FIRST EMPTY STRING ...
      NCHAR=NCHAR+1
      LCHARS(NCHAR)=LSTRNG(1)
      DO 5 I=2,12
        LCHAR=LSTRNG(I)
        IF(NN.EQ.NBLNK) GO TO 10
         NCHAR=NCHAR+1
  5      LCHARS(NCHAR)=LCHAR
C ... AND LOCATE THE NEXT BLANK OR COMMA ...
 10   J=NCOL
      DO 15 I=J,72
        LCHAR=LCARD(NCOL)
        IF(NN.EQ.NBLNK.OR.NN.EQ.NCOM) GO TO 20
         NCHAR=NCHAR+1
         LCHARS(NCHAR)=LCHAR
 15      NCOL=NCOL+1
      MORE=1
      GO TO 80
C ... NOW FIND THE START OF THE NEXT STRING
 20   CALL NONBLK(LCARD,NCOL,LCHAR,NN,NBLNK,NCOM,MORE)
 80   WRITE(6,99) LCHARS
 99   FORMAT(' INVALID CHARACTER STRING ',16A1,' IGNORED')
      RETURN
      END
```

Utility subroutine FLUSH

```
RELEASE 2.0                 NONBLK              DATE = 80122           14/00/23

      SUBROUTINE NONBLK(LCARD,NCOL,/LCHAR/,/NN/,NBLNK,NCOM,MORE)        00009800
C LOCATES THE NEXT NON-BLANK/NON-COMMA ON THE CARD.                     00009900
C     MORE=1 IF REST OF CARD IS EMPTY.                                  00010000
      LOGICAL*1 LCARD(80),LCHAR                                         00010100
      J=NCOL                                                            00010200
      DO 50 I=J,72                                                      00010300
        LCHAR=LCARD(I)                                                  00010400
        IF(NN.NE.NBLNK.AND.NN.NE.NCOM) RETURN                           00010500
   50   NCOL=NCOL+1                                                     00010600
      MORE=1                                                            00010700
      RETURN                                                            00010800
      END                                                               00010900
```

Utility subroutine NONBLK

```
RELEASE 2.0                 ADDTAB              DATE = 80122          14/00/23
      SUBROUTINE ADDTAB(IBLK2,TYPNO,TFNO,IBLK1)
C SEARCHES FOR THE CURRENT TABLE'S NUMBER IN THE TABLE LIST OF THE
C      CURRENT DATA TYPE, ADDING IT TO THE LIST IF IT'S NOT THERE
      INTEGER IBLK1(100),IBLK2(100),TYPNO,TFNO
      J=(TYPNO-1)*12+1
      K=J+11
      ICONT=2
C ONCE FOR EACH POSSIBLE TYPE-2 BLOCK
      DO 20 I=1,10
        READ(8'ICONT) IBLK2
        DO 10 L=J,K
          IF(IBLK2(L).EQ.TFNO) RETURN
          IF(IBLK2(L).NE.0) GO TO 10
           IBLK2(L)=TFNO
           WRITE(8'ICONT) IBLK2
           RETURN
 10       CONTINUE
C THIS TYPE'S TABLE LIST IS FULL, SEE IF THERE'S ANOTHER.....
        ICONT=IBLK2(100)
        IF(ICONT.GT.0) GO TO 20
C   .... NO, THERE ISN'T.  MUST CREATE ONE.
         DO 15 L=69,100
           IF(IBLK1(L).NE.0) GO TO 15
            IBLK1(L)=999
            IBLK2(100)=L-65
            WRITE(8'ICONT) IBLK2
            CALL DNULL(IBLK2,50,1)
            ICONT=L-65
            WRITE(8'ICONT) IBLK2
            WRITE(8'1) IBLK1
            GO TO 20
 15        CONTINUE
         WRITE(6,99)
 99      FORMAT(' NO BLOCK-2 SPACE AVAILABLE IN DATA DICTIONARY')
```

Utility subroutine ADDTAB

```
RELEASE 2.0            ADDTAB            DATE = 80122          14/00/23
          RETURN
C  ...  YES THERE IS.
 20       CONTINUE
        RETURN
        END
```

Utility subroutine ADDTAB

```
RELEASE 2.0              DDSGEN              DATE = 80122          14/00/23

      SUBROUTINE DDSGEN(NTFIL,LSTRT,STGLOB,ENGLOB,FREE3)                00163200
C GENERATES FORTRAN SPECIFICATION STATEMENTS (TYPE AND COMMON) FROM      00163210
C   INFORMATION IN THE DATA DICTIONARY                                  00163220
      DIMENSION IB1(100),IB2(100),IB3(100),IB4(100)                     00163300
      INTEGER VBLSTR(4),TYPCRD(21),COMCRD(21),ZED/'Z'/,NBLNK/' '/       00163400
      INTEGER STGLOB,ENGLOB,FREE3                                       00163500
      IF(STGLOB.GT.0) CALL DEGLOB(STGLOB,ENGLOB,FREE3,COMCRD)           00163600
      READ(8'1) IB1                                                     00163700
      READ(8'2) IB2                                                     00163800
      II2=2                                                             00163900
      II4=LSTRT                                                         00164000
      READ(8'LSTRT) IB4                                                 00164100
      NONZRO=0                                                          00164200
      DO 5 I=1,100                                                      00164300
        IF(IB4(I).EQ.0) GO TO 6                                         00164400
    5   NONZRO=NONZRO+1                                                 00164500
    6 CALL SORT(IB4,NONZRO)                                             00164600
      WRITE(8'LSTRT) IB4                                                00164700
C      FOR EACH TYPE                                                    00164750
      DO 100 I1=1,8                                                     00164800
        IB4PTR=1                                                        00164900
        IB1PTR=(I1-1)*5+1                                               00165000
        J1YPTR=0                                                        00165100
        JCOPTR=0                                                        00165200
        IB2PTR=(I1-1)*12+1                                              00165300
        IB2LIM=IB2PTR+11                                                00165400
C         FOR EACH POSSIBLE TYPE-2 BLOCK (MAX.5 I.E. 60 TABLES)         00165450
        DO 80 I2=1,5                                                    00165500
          IF(II2.NE.2) READ(8'2) IB2                                    00165600
C           FOR EACH POSSIBLE TABLE NUMBER IN LIST                      00165650
          DO 70 I3=IB2PTR,IB2LIM                                        00165700
            IF(IB2(I3).EQ.0) GO TO 100                                  00165800
            IF(IB2(I3).NE.NTFIL) GO TO 70                               00165900
              NDUMMY=11                                                 00166000
```

Utility subroutine DDSGEN

```
      NTYP=I1
      LSTPTR=IB1(66)
      LGTHSM=0
C      FOR EACH POSSIBLE BLOCK COMPRISING THE VARIABLE LIST
      DO 60 I4=1,34
        II3=LSTPTR-65
        LSTPTR=IB1(LSTPTR)
        READ(8'II3) IB3
C        FOR EACH VARIABLE NAME
        DO 50 I5=1,89,8
          IF(IB4PTR.GT.NONZRO) GO TO 90
          IF(IB3(I5+2).NE.NTYP) GO TO 50
          IB3PTR=(I4*1000+I5)
          DO 40 I6=IB4PTR,NONZRO
            IF(IB4(I6).LT.IB3PTR) GO TO 40
              IF(IB4(I6).EQ.IB3PTR) GO TO 20
C               VARIABLE NOT GLOBAL FOR THIS TABLE, ADD
C                   LENGTH TO DUMMY VARIABLE
                LGTHSM=LGTHSM+IB3(I5+3)
                GO TO 50
   20         IF(LGTHSM.EQ.0) GO TO 30
C               BUILD DUMMY VAR. BEFORE PROCEEDING WITH REAL...
                CALL BLDVAR(ZED,IB1(IB1PTR+3),NDUMMY,0,
     +                 VBLSTR,NCHARS)
C               ... AND JOIN IT TO BOTH TYPE & COMMON STMTS.
                CALL ADDTYP(VBLSTR,NCHARS,IB1(IB1PTR),JTYPTR,
     +                 TYPCRD,STGLOB,ENGLOB,FREE3)
                CALL BLDVAR(VBLSTR,NBLNK,0,LGTHSM,VBLSTR,NCHARS)
                CALL ADDCOM(VBLSTR,NCHARS,IB1(IB1PTR+3),JCUPTR,
     +                 COMCRD,STGLOB,ENGLOB,FREE3)
                LGTHSM=0
                NDUMMY=NDUMMY+1
   30           CONTINUE
C               NOW PUT TOGETHER THE REAL VAR. NAME, AND
```

Utility subroutine DDSGEN

```
RELEASE 2.0              DDSGEN              DATE = 80122          14/00/23
C                  JOIN IT TO THE TYPE STMT.                               00168775
              CALL BLDVAR(IB3(I5),NBLNK,0,0,VBLSTR,NCHARS)                 00168800
              CALL ADDTYP(VBLSTR,NCHARS,IB1(IB1PTR),JTYPTR,                00168900
     +             TYPCRD,STGLOB,ENGLOB,FREE3)                             00169000
C              APPEND DIMENSION INFO. IF REQU'D. AND JOIN TO              00169050
C                  THE COMMON STMT.                                        00169075
              IF(IB3(I5+4).NE.NBLNK) CALL BLDVAR(VBLSTR,IB3(I5+400169100
     +             ),0,0,VBLSTR,NCHARS)                                    00169200
              CALL ADDCOM(VBLSTR,NCHARS,IB1(IB1PTR+3),JCOPTR,              00169300
     +             COMCRD,STGLOB,ENGLOB,FREE3)                             00169400
              IB4PTR=I6+1                                                  00169500
              GO TO 50                                                     00169600
  40          CONTINUE                                                     00169700
            GO TO 90                                                       00169800
  50        CONTINUE                                                       00169900
          IF(LSTPTR.LT.0) GO TO 90                                         00170000
  60      CONTINUE                                                         00170100
        WRITE(6,999)                                                       00170200
 999    FORMAT(' TOO MANY VARIABLE LIST BLOCKS')                           00170300
        STOP                                                               00170400
  70  CONTINUE                                                             00170500
     IF(IB2(100).EQ.0) GO TO 90                                            00170600
     I12=IB2(100)                                                          00170700
  80 CONTINUE                                                              00170800
C    WRITE THE LAST TYPE AND COMMON CARDS INTO THE SOURCE FILE (3)00170850
  90 CALL PRTCOM(COMCRD,STGLOB,ENGLOB,FREE3)                               00170900
     CALL PRTTYP(TYPCRD,STGLOB,ENGLOB,FREE3)                               00171000
 100 CONTINUE                                                              00171100
     RETURN                                                                00171200
     END                                                                   00171300
```

Utility subroutine DDSGEN

```
RELEASE 2.0              DEGLOB              DATE = 80122          14/00/23

      SUBROUTINE DEGLOB(STGLOB,ENGLOB,FREE3,CARD)                        00171400
C RETURNS ANY EXISTING GLOBAL SPEC. STMTS. TO FREE SPACE LIST BEFORE      00171450
C        REGENERATION                                                    00171475
      INTEGER STGLOB,ENGLOB,FREE3,CARD(21),NBLNK/' '/                    00171500
      DO 10 I=1,100                                                      00171600
        NXTFRE=STGLOB                                                    00171700
        READ(3'NXTFRE,999) CARD                                          00171800
        STGLOB=CARD(21)                                                  00171900
        CARD(21)=FREE3                                                   00172000
        WRITE(3'NXTFRE,999) CARD                                         00172100
        FREE3=NXTFRE                                                     00172200
        IF(STGLOB.EQ.0) GO TO 20                                         00172300
 10     CONTINUE                                                         00172400
      STOP 10                                                            00172500
 20   ENGLOB=0                                                           00172600
      RETURN                                                             00172700
 999  FORMAT(20A4,I4)                                                    00172800
      END                                                                00172900
```

Utility subroutine DEGLOB

```
RELEASE 2.0             SORT                 DATE = 80122        14/00/23

      SUBROUTINE SORT(IARAY,LGTH)                                    00178600
C BUBBLE SORT                                                        00178650
      DIMENSION IARAY(LGTH)                                          00178700
      LIM=LGTH                                                       00178800
      LIM1=LIM-1                                                     00178900
      DO 20 I=1,LIM1                                                 00179000
        K=I+1                                                        00179100
        DO 20 J=K,LIM                                                00179200
          IF(IARAY(I).LE.IARAY(J)) GO TO 20                          00179300
            ITMP=IARAY(I)                                            00179400
            IARAY(I)=IARAY(J)                                        00179500
            IARAY(J)=ITMP                                            00179600
   20       CONTINUE                                                 00179700
      RETURN                                                         00179800
      END                                                            00179900
```

Utility subroutine. SORT

```
RELEASE 2.0              BLDVAR               DATE = 80122          14/00/23

      SUBROUTINE BLDVAR(S1,S2,N1,N2,VAR,NCHAR)                           00173000
C BUILDS A VARIABLE NAME OUT OF STRINGS S1 S2 AND NUMERICS N1 N2.        00173050
C    THE RESULTING STRING IS S1S2N1(N2), WHERE N1 AND (N2) ARE           00173055
C    ONLY INCLUDED IF NON-ZERO.                                          00173060
      LOGICAL*1 S1(1),S2(1),VAR(16),LBLNK,LINTVA(4),LDIG(12)             00173100
      EQUIVALENCE (NBLNK,LBLNK),(INTVAR,LINTVA),(LDIG,IDIG)              00173200
      DIMENSION IDIG(3)                                                  00173300
      DATA NBLNK/' '/,IDIG/'0123','4567','89()'/                         00173400
      NCHAR=0                                                            00173500
      INTVAR=NBLNK                                                       00173600
      CALL MOVCH(S1,VAR,LINTVA,INTVAR,NBLNK,NCHAR)                       00173700
      CALL MOVCH(S2,VAR,LINTVA,INTVAR,NBLNK,NCHAR)                       00173800
      IF(N1.GT.0) CALL ADDIGS(N1,VAR,LDIG,NCHAR)                         00173900
      VAR(NCHAR+1)=LBLNK                                                 00174000
      IF(N2.LE.1) RETURN                                                 00174100
      NCHAR=NCHAR+1                                                      00174200
      VAR(NCHAR)=LDIG(11)                                                00174300
      CALL ADDIGS(N2,VAR,LDIG,NCHAR)                                     00174400
      NCHAR=NCHAR+1                                                      00174500
      VAR(NCHAR)=LDIG(12)                                                00174600
      IF(NCHAR.GE.16) RETURN                                             00174700
      J=NCHAR+1                                                          00174800
      DO 5 I=J,16                                                        00174900
    5   VAR(I)=LBLNK                                                     00175000
      RETURN                                                             00175100
      END                                                                00175200
```

Utility subroutine BLDVAR

```
RELEASE 2.0              MODTAB              DATE = 80122          14/00/23

      SUBROUTINE MODTAB(IBLK2,IBLK3,TYPNO,MODFY)
C MARK AS "CHANGED", ALL TABLES REFERENCING DATA ITEMS
C     OF TYPE TYPNO
      INTEGER IBLK2(100),IBLK3(100),TYPNO
      IF(MODFY.EQ.0) RETURN
      READ(8'3) IBLK3
      ICONT=2
C DO ONCE FOR EACH POSSIBLE TYPE-2 BLOCK (LISTS OF TABLE NUMBERS)
      DO 20 I=1,10
        IF(ICONT.EQ.0) GO TO 30
        READ(8'ICONT) IBLK2
        J=(TYPNO-1)*12+1
        K=J+11
C DO ONCE FOR EACH POSSIBLE TABLE USING THIS TYPE
        DO 10 L=J,K
          IF(IBLK2(L).EQ.0) GO TO 35
C SET THE MOD FLAG AFTER LOCATING THE APPROPRIATE TABLE
          DO 5 M=7,99,4
            IF(IBLK3(M).EQ.0) GO TO 10
            IF(IBLK3(M).NE.IBLK2(L)) GO TO 5
            IBLK3(M+1)=1
 5          CONTINUE
 10       CONTINUE
 20     ICONT=IBLK2(100)
      WRITE(6,99)
 99   FORMAT(' LOOPING IN MODTAB ON TYPE-2 BLOCKS')
C RECOVER THE TYPE-LIST BLOCK FOR THE RETURN
 30   READ(8'1) IBLK2
      RETURN
 35   IF(MODFY.EQ.1) WRITE(8'3) IBLK3
      MODFY=0
      READ(8'1) IBLK2
      RETURN
      END
```

Utility subroutine MODTAB

```
RELEASE 2.0              ANALYS              DATE = 80122          14/00/23

      SUBROUTINE ANALYS(LCARD,NCOL,VARNAM,LCHARS,LGTH,MORE)
C A TABLE-DRIVEN PARSER.    OUT OF A USER'S GLOBAL SPECIFICATION
C         STATEMENT, INFORMATION IS EXTRACTED AS TO NAME, LENGTH,
C         AND DIMENSIONS, FOR EACH DATA ELEMENT MENTIONED.
      INTEGER VARNAM(2),STRING(3),STACK(10),TARGET
      INTEGER RULNO,TOKNID
      INTEGER TABLE(7,12),RULTAB(3,6)
      LOGICAL*1 LCHARS(16),LSTRNG(12),LCHAR,LCARD(80)
      EQUIVALENCE (LCHAR,NN),(STRING(1),LSTRNG(1))
      DATA NBLNK/' '/,NCOM/','/
      DATA TABLE/
     +    -1,     0,     0,     0,     0,     0,     0,
     +     0,    -3,     0,     0,     0,     0,     0,
     +     0,     0,    -2,     0,     0,     0,     0,
     +     0,     0,     0,    -2,     0,     0,     0,
     +     0,     0,     0,     0,    -2,     0,     0,
     +     0,     0,     0,     0,     0,    -4,     0,
     +     0,     0,     0,    -2,     6,     0,     0,
     +     0,     5,     0,     0,     0,     0,     0,
     +     0,     0,     3,     0,    -4,    -4,     0,
     +     2,     0,     0,     0,     0,     0,     0,
     +     0,     0,     0,     0,    -4,    -4,     0,
     +     1,     0,     0,     0,     0,     0,     0/
      DATA RULTAB/
     +  2,  9, 10,     1,  1,  0,     2,  8,  3,     2,  6,  4,
     +  2,  7,  2,     2,  8,  5 /
      DATA STACK / 0,3,4,5,6,7,8,9,10,0/
      NN=NBLNK
      DO 3 I=1,16
    3    LCHARS(I)=LCHAR
      TARGET=12
      MSTK=10
      CALL PUSH(TARGET,STACK,MSTK)
C GET FIRST TOKEN
```

Utility subroutine ANALYS

```
RELEASE 2.0              ANALYS              DATE = 80122          14/00/23
      CALL GET(TOKNID,STRING,LCARD,NCOL)
      DO 50 I=1,100
        IF(STACK(1).NE.0) GO TO 5
C STACK IS EMPTY SO END OF ITEM.  FIND NEXT NON-BLANK CHAR. & QUIT
        CALL NONBLK(LCARD,NCOL,LCHAR,NN,NBLNK,NCOM,MORE)
        RETURN
 5      RULNO=TABLE(TOKNID,STACK(STACK(1)))
        IF(RULNO.NE.0) GO TO 10
C ILLEGAL TOKEN
         CALL FLUSH(LCHARS,LCARD,NCOL,STACK,MSTK,NCHARS,MORE,NN,
     *              NBLNK,NCOM,LCHAR,LSTRNG)
         RETURN
 10     IRN=-RULNO
        GO TO (20,30,25,35),IRN
         CALL POP(STACK,MSTK)
         J=RULTAB(1,RULNO)
         IF(J.EQ.0) GO TO 50
          DO 15 K=1,J
 15         CALL PUSH(RULTAB(K+1,RULNO),STACK,MSTK)
          GO TO 50
 20     VARNAM(1)=STRING(1)
        VARNAM(2)=STRING(2)
        LGTH=1
        NCHAR=0
        CALL GET(TOKNID,STRING,LCARD,NCOL)
        GO TO 35
 25      CALL CONVRT(STRING,NUM,IOK)
         LGTH=LGTH*NUM
 30     DO 33 J=1,12
          LCHAR=LSTRNG(J)
          IF(NN.EQ.NBLNK) GO TO 34
 33         LCHARS(NCHAR+J)=LCHAR
 34     NCHAR=NCHAR+J-1
        CALL GET(TOKNID,STRING,LCARD,NCOL)
```

Utility subroutine ANALYS

```
RELEASE 2.0            ANALYS            DATE = 80122          14/00/23

   35      CALL POP(STACK,MSTK)                                   00006900
   50      CONTINUE                                               00007000
          RETURN                                                  00007100
          END                                                     00007200
```

Utility subroutine ANALYS

```
RELEASE 2.0              PUSH                DATE = 80122          14/00/23

      SUBROUTINE PUSH(N,S,MS)
C STACK MANAGEMENT UTILITY
      INTEGER S(MS)
      IF(S(2).NE.0) GO TO 5
        WRITE(6,99)
 99     FORMAT(' STACK FULL')
        RETURN
 5    IP=S(2)
      S(2)=S(IP)
      S(IP)=N
      S(1)=IP
      RETURN
      END
```

Utility subroutine PUSH.

```
RELEASE 2.0              POP                    DATE = 80122         14/00/23

      SUBROUTINE POP(S,MS)
C STACK MANAGEMENT UTILITY
      INTEGER S(MS)
      IF(S(1).NE.0) GO TO 5
        WRITE(6,99)
 99     FORMAT(' STACK EMPTY')
 5    S(S(1))=S(2)
      S(2)=S(1)
      S(1)=S(1)-1
      IF(S(1).LT.3) S(1)=0
      RETURN
      END
```

Utility subroutine POP

```
STMT   SOURCE STATEMENT                                              ASM 0201 13.58 05/01/80

   1 TRANS   CSECT
   2 *.SUBROUTINE TO EXTRACT AND IDENTIFY STRINGS OF CHARACTERS FOR ANALYS
   3 * INVOKED BY CALL GET(IDNO,STRING,CARD,NCOL)
   4 *     IDNO - INTEGER WORD RCVING STRING CODE NUMBER FOR THE PARSE TABLE
   5 *     STRING - 3 WORDS TO RECEIVE THE CHARACTERS EXTRACTED
   6 *     CARD - 80 BYTE CARD IMAGE FROM WHICH STRINGS ARE DRAWN
   7 *     NCOL - POINTS TO START OF STRING ON CARD -- UPDATED ON FINISHING
   8 *
   9         ENTRY  GET
  10         DC     CL7'GET'
  11         DC     X'07'
  12 GET     STM    14,12,12(13)
  13         BALR   11,0
  14         USING  *,11
  15         LM     4,7,0(1)          ADDR OF IDNO,STRING,CARD,NCOL
  16         MVI    0(5),X'40'        BLANK FIRST CHARACTER
  17         MVC    1(11,5),0(5)      ... AND REST OF STRING
  18         SR     2,2
  19         LR     9,7               REMEMBER WHERE'S NCOL
  20         L      7,0(7)            VALUE OF NCOL
  21         BCTR   7,0
  22         AR     6,7               START ADDRESS OF INPUT STRING
  23         TRT    0(1,6),TABLE1-64     LOOK AT FIRST CHARACTER
  24         ST     2,0(4)            SET IDNO FOR RETURN
  25         LA     8,2
  26         CR     2,8               EXAMINE IDNO
  27         LA     1,1(1)            INCR CHAR ADDR IN CASE IT WAS NON-ALPHNUM
  28         BH     FINISH            CHARACTER WAS NON-ALPHANUMERIC
  29         BL     ALPHA             ITS AN IDENTIFIER
  30         LA     8,TABLEN-64       THIS ONE MUST BE A NUMERIC
  31         B      SECOND
  32 ALPHA   LA     8,TABLEA-64
  33 SECOND  TRT    1(11,6),0(8)      CHECK REST OF CHARACTERS FOR VALIDITY
  34 FINISH  SR     1,6               NUMBER OF CHARS TO BE STORED
  35         BCTR   1,0               DECR. CHAR COUNT FOR MOVE
  36         EX     1,MOVSTR          MOVE CHARS INTO STRING
```

Utility subroutine GET

```
STMT   SOURCE STATEMENT                                         ASM 0201 13.58 05/01/80

  37          LA     1,2(1)
  38          AR     7,1              ADJUST NCOL ...
  39          ST     7,0(9)           ...  AND STORE IT
  40          LM     14,12,12(13)
  41          BALR   15,14
  42          DS     F
  43 MOVSTR   MVC    0(0,5),0(6)      MOVE INSTR FOR CHAR STRING
  44 TABLE1   DC     X'06'            SPACE     *
  45          DC     12X'07',X'03'    LEFT BRKT *
  46          DC     15X'07',X'04'    RGHT BRKT *
  47          DC     13X'07',X'05'    COMMA     *
  48          DC     85X'07'          RUBBISH   *
  49          DC     9X'01'           A-I       * FIRST CHARACTER
  50          DC     7X'07'           RUBBISH   * TRANSLATE TABLE
  51          DC     9X'01'           J-R       *
  52          DC     8X'07'           RUBBISH   *
  53          DC     8X'01'           S-Z       *
  54          DC     6X'07'           RUBBISH   *
  55          DC     10X'02'          0-9       *
  56 TABLEN   DC     176X'01'                   =
  57          DC     10X'00'                    =NUM. VALID. TABLE
  58 TABLEA   DC     129X'01'                   ?
  59          DC     9X'00'                     ?
  60          DC     7X'01'                     ?
  61          DC     9X'00'                     ? ALPHANUMERIC VALIDITY
  62          DC     8X'01'                     ?       TABLE
  63          DC     8X'00'                     ?
  64          DC     6X'01'                     ?
  65          DC     10X'00'                    ?
  66          END
```

Utility subroutine GET

```
RELEASE 2.0              ADDTYP              DATE = 80122          14/00/23

          SUBROUTINE ADDTYP(VAR;NCHAR,TYPNAM,JPTR,CARD,SCEHDR,SCETLR,FREE3) 00182500
C ADDS THE DATA ELEMENT SUPPLIED (VAR) TO THE END OF THE EXISTING          00182550
C    TYPE SPECIFICATION CARD,ESTARTING A NEW ONE IF THIS IS TOO LONG       00182575
      LOGICAL*1 VAR(NCHAR),CARD(84),TYPNAM(12),LSYM(4)                     00182600
      INTEGER SCEHDR,SCETLR,FREE3,ISYM/',','/                              00182700
      EQUIVALENCE (ISYM,LSYM)                                              00182800
      IF(JPTR+NCHAR+1.LT.73) GO TO 5                                       00182900
        CALL PRTTYP(CARD,SCEHDR,SCETLR,FREE3)                              00183000
        JPTR=0                                                             00183100
 5    IF(JPTR.GT.0) GO TO 25                                               00183200
        DO 15 I=1,84                                                       00183300
 15       CARD(I)=LSYM(2)                                                  00183400
        DO 20 I=7,16                                                       00183500
 20       CARD(I)=TYPNAM(I-6)                                              00183600
        JPTR=16                                                            00183700
 25   JPTR=JPTR+1                                                          00183800
      IF(JPTR.GT.17) CARD(JPTR)=LSYM(1)                                    00183900
      DO 30 I=1,NCHAR                                                      00184000
        JPTR=JPTR+1                                                        00184100
 30     CARD(JPTR)=VAR(I)                                                  00184200
      RETURN                                                               00184500
      END                                                                  00184600
```

Utility subroutine ADDTYP

```
RELEASE 2.0              ADDCOM              DATE = 80122          14/00/23

      SUBROUTINE ADDCOM(VAR,NCHAR,BLKNAM,JPTR,CARD,SCEHDR,SCETLR,FREE3) 00180000
C ADDS THE DATA ELEMENT SUPPLIED (VAR) TO THE END OF THE EXISTING        00180050
C   COMMON STATEMENT, CREATING A NEW CARD IF THE CURRENT ONE OVERFLOWS  00180075
      LOGICAL*1 VAR(NCHAR),BLKNAM(2),CARD(84),LSYM(4),COMTIT(8)         00180100
      INTEGER SCEHDR,SCETLR,ISYM/',/'/,ICOMT(2)/'COMM','ON/'/,FREE3     00180200
      EQUIVALENCE (LSYM,ISYM),(COMTIT,ICOMT)                            00180300
      IF(JPTR+NCHAR+1.LT.73) GO TO 5                                    00180400
        CALL PRTCOM(CARD,SCEHDR,SCETLR,FREE3)                           00180500
        JPTR=0                                                          00180600
5     IF(JPTR.GT.0) GO TO 25                                            00180700
        DO 15 I=1,84                                                    00180800
15        CARD(I)=LSYM(3)                                               00180900
        DO 20 I=7,13                                                    00181000
20        CARD(I)=COMTIT(I-6)                                           00181100
        CARD(14)=BLKNAM(1)                                              00181200
        CARD(15)=BLKNAM(2)                                              00181300
        CARD(16)=LSYM(2)                                                00181400
        JPTR=16                                                         00181500
25    JPTR=JPTR+1                                                       00181600
      IF(JPTR.GT.17) CARD(JPTR)=LSYM(1)                                 00181700
      DO 30 I=1,NCHAR                                                   00181800
        JPTR=JPTR+1                                                     00181900
30      CARD(JPTR)=VAR(I)                                               00182000
      RETURN                                                            00182300
      END                                                               00182400
```

Utility subroutine ADDCOM

```
RELEASE 2.0              PRTCOM              DATE = 80122          14/00λ23

      SUBROUTINE PRTCOM(CARD,STGLOB,ENGLOB,FREE3)                     00184700
C WRITES A COMMON STATEMENT TO UNIT 3                                  00184750
      INTEGER CARD(21),STGLOB,ENGLOB,FREE3                            00184800
      IF(FREE3.GT.0) GO TO 5                                          00185100
        WRITE(6,999)                                                  00185200
  999 FORMAT(' NO SPACE AVAILABLE ON UNIT 3. NOTHING WRITTEN')        00185300
        RETURN                                                        00185400
  5   READ(3'FREE3,997) NEXT                                          00185500
  997 FORMAT(80X,I4)                                                  00185600
  998 FORMAT(20A4,I4)                                                 00185700
      CARD(21)=0                                                      00185800
      WRITE(3'FREE3,998) CARD                                         00186100
      IF(ENGLOB.GT.0) GO TO 15                                        00186200
        STGLOB=FREE3                                                  00186300
        GO TO 20                                                      00186400
  15  READ(3'ENGLOB,998) CARD                                         00186500
      CARD(21)=FREE3                                                  00186600
      WRITE(3'ENGLOB,998) CARD                                        00186800
  20  ENGLOB=FREE3                                                    00186900
      FREE3=NEXT                                                      00187000
      RETURN                                                          00187300
      END                                                             00187400
```

Utility subroutine PRTCOM

```
RELEASE 2.0              PRTTYP              DATE = 80122          14/00/23

       SUBROUTINE PRTTYP(CARD,STGLOB,ENGLOB,FREE3)                  00187500
C WRITES A TYPE SPECIFICATION STATEMENT TO UNIT 3                    00187550
       INTEGER CARD(21),STGLOB,ENGLOB,FREE3                         00187600
       IF(FREE3.GT.0) GO TO 5                                       00187800
         WRITE(6,999)                                               00187900
  999  FORMAT(' NO SPACE AVAILABLE ON UNIT 3.  NOTHING WRITTEN')    00188000
         RETURN                                                     00188100
 5     READ(3'FREE3,997) NEXT                                       00188200
 997   FORMAT(80X,I4)                                               00188300
 998   FORMAT(20A4,I4)                                              00188400
       CARD(21)=STGLOB                                              00188500
       IF(STGLOB.EQ.0) ENGLOB=FREE3                                 00188700
       STGLOB=FREE3                                                 00188800
       WRITE(3'FREE3,998) CARD                                      00188900
       FREE3=NEXT                                                   00189000
       RETURN                                                       00189200
       END                                                          00189600
```

Utility subroutine PRTTYP

```
RELEASE 2.0              MOVCH                DATE = 80122          14/00/23

          SUBROUTINE MOVCH(SS,VAR,LINTVA,INTVAR,NBLNK,NCHAR)                 00177100
    C MOVES NON-BLANK CHARS FROM SS INTO VAR, JOINING THEM TO ANY EXISTING  00177150
          LOGICAL*1 SS(1),VAR(16),LINTVA(4)                                  00177200
          INTEGER INTVAR(1)                                                  00177300
          DO 5 I=1,100                                                       00177400
            LINTVA(1)=SS(I)                                                  00177500
            IF(INTVAR(1).EQ.NBLNK) GO TO 10                                  00177600
            NCHAR=NCHAR+1                                                    00177700
            VAR(NCHAR)=SS(I)                                                 00177800
      5     CONTINUE                                                         00177900
          STOP 9000                                                          00178000
     10   VAR(NCHAR+1)=LINTVA(2)                                             00178100
          RETURN                                                             00178400
          END                                                                00178500
```

Utility subroutine MOVCH

```
RELEASE 2.0              ADDIGS              DATE = 80122              14/00/23

          SUBROUTINE ADDIGS(NN,VAR,LDIG,NCHAR)                           00175300
C CONVERTS INTEGER NN TO EBCDIC IN VAR                                   00175350
          LOGICAL*1 VAR(16),LDIG(12)                                     00175400
          N=1                                                            00175500
          DO 10 I=1,5                                                    00175600
            N=10*N                                                       00175700
            IF(NN.LT.N) GO TO 15                                         00175800
   10       CONTINUE                                                     00175900
          WRITE(6,99) NN                                                 00176000
   99     FORMAT(' MAX. 5 DIGITS IN ADDIGS.  NN=',I10)                   00176100
          STOP                                                           00176200
   15     N=NN                                                           00176300
          DO 20 J=1,I                                                    00176400
            M=N/10**(I-J)                                                00176500
            N=N-M*10**(I-J)                                              00176600
            NCHAR=NCHAR+1                                                00176700
   20       VAR(NCHAR)=LDIG(M+1)                                         00176800
          RETURN                                                         00176900
          END                                                            00177000
```

Utility subroutine ADDIGS

LIST OF REFERENCES

1. Higgins, D.S., "A Structured FORTRAN Translator," ACM SIGPLAN Notices, February 1975.

2. Polster, F.J., "Using a Preprocessor to Implement a Data Manipulation Language for a Minicomputer Data Base System," ACM SIGMINI Newsletter, August 1978.

3. Malkosh, M., "Internal Procedure Parameters in Structured Fortran Precompilers," ACM SIGPLAN Notices, September 1977.

4. Fancott, T., and Jaworski, W.M., "Primitive Logic Constructs Considered Harmful in Structured Programs," Conference Proceedings, Canadian Computer Conférence, Montreal 1976.

5. Belkin, G. and Jaworski, W.M., "Towards Logic and Performance Analysis of Unwritten Programs," Conference Proceedings, Canadian Computer Conference, Montreal 1976.

6. Floyd, W.F., "The Paradigms of Programming," Comm. ACM 22, August 1979.

7. Turner, J., "The Structure of Modular Programs," Comm. ACM 23, May, 1980.

8. Tausworthe, R.C., Standardized Development of Computer Software, Prentice-Hall, 1977.

9. Pollack, S.L., Decision Tables-Theory and Practice, John Wiley and Sons, Inc., 1971.

10. Davis, G.B., Computer Data Processing, McGraw-Hill, 1973.

11. Silver, G.A., and Silver, J.B., Introduction to Systems Analysis, Prentice-Hall 1976.

12. Anastas, M.S., and Vaughan, R.F., "Direct Architectural Implementation of a Requirements-Oriented Computing Structure," ACM SIGMICRO Newsletter, December 1979.

13. Low, D.W., "Programming by Questionnaire: An Effective Way to Use Decision Tables," Comm. ACM 16, May 1973.

14. Jaworski, W.M., "Software Development Using ABL," Unpublished Report, Concordia University, Montreal.

15. House, R., "Comments on Program Specification and Testing," Comm. ACM 23, June 1980.

16. Glass, R.L., "A Benefit Analysis of Some Software Reliability Methodologies," ACM SIGSOFT Newsletter, April 1980.

17. Glass, R.L., "Real-Time: The 'Lost World' of Software Debugging and Testing," Comm ACM 23, May 1980.

18. Howden, W.E., "Applicability of Software Validation Techniques to Scientific Programs," ACM TOPLAS 2, July 1980.

19. Myers, G.J., "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections," Comm. ACM 21, September 1978.

20. Dijkstra, E.W., A Discipline of Programming, Prentice-Hall, 1976.

21. Hinterberger, H., and Jaworski, W.M., "Controlled Program Design by Use of the Programming Concept ABL," awaiting publication in Angewandte Informatik.

22. Kernighan, B.W., and Plauger, P.J., The Elements of Programming Style, McGraw-Hill, 1978.

23. IBM Systems Reference Library, IBM OS Linkage Editor and Loader, GC28-6538.

GLOSSARY

CLUSTER, a statement of membership of one or more rules within a sub-module. Used for the construction of that sub-module. The information set defining all of a sub-module's clusters constitutes the Cluster IN-UNIT.

COMPOUND ACTION, a logically related set of SIMPLE ACTIONS, seen by the user as one such.

DRIVEN, name given to the sub-module which is directly operated upon by the DRIVER.

DRIVER, name given to the module which provides the driving power, both to the system's and the application's DRIVEN sub-modules.

EBCDIC, extended binary coded decimal interchange code: an 8-bit alphanumeric code.

IN-UNIT, component of the tabular part of a sub-module. There are four in-units,--Rule, Strategy, Cluster, Action--which are built from a user's input.

MINI-DRIVER, a small version of the DRIVER, which is inserted in the code of a sub-module to provide the driving power for a lower level sub-module.

PDT, the Program Descriptor Table. Held in block one of a sub-module's table file. It contains information to control building of in-units, and to locate them for subsequent sub-module execution.

SIMPLE ACTION, a set of zero or more executable Fortran statements with a single entry and a single exit (first and last statements). It represents the expression of a user's single logical action.

STRATEGY, the IN-UNIT in which the user overrides the natural execution sequence of a sub-module.

SUB-MODULE, an abstract machine, built from Fortran statements expressing the logic and the executable actions, using the IN-UNITS of the tabular information as its program.