

AD-A132 934

INVESTIGATION OF INTRINSICALLY ERROR-FREE PROGRAMS(U)

1/1

COMPUTER SYSTEM ASSOCIATES SAN DIEGO CA

G V WINTRISS ET AL. 27 JUL 83 ARO-18465.1-EL-S

UNCLASSIFIED

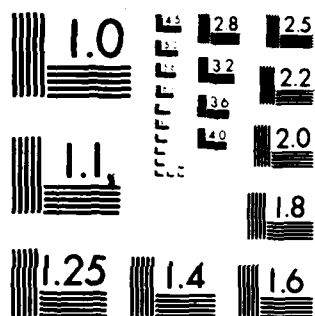
DAAG29-81-C-0021

F/G 9/2

NL

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

END  
DATE  
FILMED  
10 83  
DTIC



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS 1963-A

ARO 18465.1-EL-S  
(12)

AD-A132934



DEPARTMENT OF THE ARMY  
U. S. ARMY RESEARCH OFFICE  
P. O. BOX 12211  
RESEARCH TRIANGLE PARK NORTH CAROLINA 27709

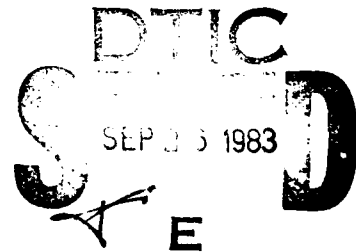
INVESTIGATION OF

# INTRINSICALLY ERROR-FREE PROGRAMS

DTIC FILE COPY

**FINAL REPORT**

Approved for Public Release,  
Distribution Unlimited.



**CSA** COMPUTER SYSTEM ASSOCIATES  
7562 Trade Street San Diego CA 92121  
Telephone 566-3911

27 JULY 1983

UNDER CONTRACT NUMBER  
DAA8-28-81-C-0021

83 09 20 023



DEPARTMENT OF THE ARMY  
U. S. ARMY RESEARCH OFFICE  
P. O. BOX 12211  
RESEARCH TRIANGLE PARK, NORTH CAROLINA 27709

INVESTIGATION OF  
**INTRINSICALLY ERROR-  
FREE PROGRAMS**

**FINAL REPORT**

Approved for Public Release,  
Distribution Unlimited.

27 JULY 1983

|               |                                     |
|---------------|-------------------------------------|
| Accession For |                                     |
| NTIS          | <input checked="" type="checkbox"/> |
| DTIC          |                                     |
| Un            |                                     |
| Just          |                                     |
| By            |                                     |
| Date          |                                     |
| Avail         |                                     |
| Dist          |                                     |
| A             |                                     |



**CSA** **COMPUTER SYSTEM ASSOCIATES**  
7562 Trade Street, San Diego, CA 92121  
Telephone 586-3911

UNDER CONTRACT NUMBER  
DAAG-28-81-C-0021

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE  
THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL  
DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO  
DESIGNATED BY OTHER DOCUMENTATION.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE                                                                                                                                                                                                                                                                                                                                                         |                       | READ INSTRUCTIONS<br>BEFORE COMPLETING FORM                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|----------------------------------------------------------------------------|
| 1. REPORT NUMBER                                                                                                                                                                                                                                                                                                                                                                  | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER                                              |
| ATD-A132                                                                                                                                                                                                                                                                                                                                                                          |                       | 934                                                                        |
| 4. TITLE (and Subtitle)<br>INVESTIGATION OF INTRINSICALLY<br>ERROR-FREE PROGRAMS                                                                                                                                                                                                                                                                                                  |                       | 5. TYPE OF REPORT & PERIOD COVERED<br>Final Report<br>20 Apr 81-19 June 83 |
|                                                                                                                                                                                                                                                                                                                                                                                   |                       | 6. PERFORMING ORG. REPORT NUMBER                                           |
| 7. AUTHOR(s)<br>G. Victor Wintriss      Nicholas Panos<br>Jeannine Wolf          Andrew Ash<br>Dr. Michael Andrews                                                                                                                                                                                                                                                                |                       | 8. CONTRACT OR GRANT NUMBER(s)<br>DAAG-29-81-C-0021                        |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Computer System Associates<br>7562 Trade Street<br>San Diego, CA 92121                                                                                                                                                                                                                                                             |                       | 10. PROGRAM ELEMENT, PROJECT, TASK<br>AREA & WORK UNIT NUMBERS             |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>U. S. Army Research Office<br>Post Office Box 12211<br>Research Triangle Park, NC 27709                                                                                                                                                                                                                                                |                       | 12. REPORT DATE<br>27 July 1983                                            |
|                                                                                                                                                                                                                                                                                                                                                                                   |                       | 13. NUMBER OF PAGES<br>45                                                  |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)                                                                                                                                                                                                                                                                                                       |                       | 15. SECURITY CLASS. (of this report)<br>Unclassified                       |
|                                                                                                                                                                                                                                                                                                                                                                                   |                       | 15a. DECLASSIFICATION/DOWNGRADING<br>SCHEDULE                              |
| 16. DISTRIBUTION STATEMENT (of this Report)<br><br>Approved for public release; distribution<br>unlimited.                                                                                                                                                                                                                                                                        |                       |                                                                            |
| 17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)                                                                                                                                                                                                                                                                                        |                       |                                                                            |
| 18. SUPPLEMENTARY NOTES<br><br>THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT<br>ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS<br>THE OFFICIAL POSITION, POLICY, OR DE-<br>CISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.                                                                                                                         |                       |                                                                            |
| 19. KEY WORDS (Continue on reverse side if necessary and identify by block number)<br>Finite state machine; automated program generation; error-free<br>software; state variables; software design; automatic program<br>synthesis                                                                                                                                                |                       |                                                                            |
| 20. ABSTRACT (Continue on reverse side if necessary and identify by block number)<br>The report analyzes the application of machine design techniques to<br>software development. The techniques of finite state machine<br>design are extended to software design. The requirements for an<br>automated programming system are developed and a prototype system<br>is described. |                       |                                                                            |

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

This Page Intentionally Blank

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## TABLE OF CONTENTS

|                                                                                                       |     |
|-------------------------------------------------------------------------------------------------------|-----|
| 1. INTRODUCTION                                                                                       | 1   |
| 1.1 Overview                                                                                          | 1   |
| 2. BACKGROUND                                                                                         | 2   |
| 2.1 Finite State Machine Design                                                                       | 2   |
| 2.2 The Finite State Machine Model                                                                    | 2   |
| 2.3 Synthesis Procedure                                                                               | 5   |
| 2.4 Application to Software Generation                                                                | 6   |
| 3. STATE DIAGRAM PROGRAM GENERATION                                                                   | 9   |
| 3.1 Program Overview                                                                                  | 9   |
| 3.2 Primary Menu                                                                                      | 9   |
| 3.3 Instruction Display                                                                               | 10  |
| 3.4 Definition of the Variables                                                                       | 10  |
| 3.5 State Diagram Creation                                                                            | 12  |
| 3.6 Subroutine Identification                                                                         | 14  |
| 3.7 Creating the Source Program                                                                       | 15  |
| BIBLIOGRAPHY                                                                                          | 17  |
| APPENDIX A. Program Implementing Mealy Machine                                                        | A-1 |
| APPENDIX B. Program Implementing Moore Table                                                          | B-1 |
| APPENDIX C. State Diagram Program Generator                                                           | C-1 |
| APPENDIX D. Skeleton Program Driver for Program Generation                                            | D-1 |
| APPENDIX E. Program Created by State Diagram Program Generator                                        | E-1 |
| APPENDIX F. List of Publications and Technical Reports and List of Participating Scientific Personnel | F-1 |

#### LIST OF FIGURES

|              |                                                                                                  |   |
|--------------|--------------------------------------------------------------------------------------------------|---|
| FIGURE 1.    | Generalized block diagram of circuit implementation of Mealy and Moore type finite state machine | 4 |
| FIGURE 2(a). | Example of a Mealy type state table                                                              | 5 |
| FIGURE 2(B). | The Moore table corresponding to the table in Figure 2(a)                                        | 5 |

A QUERY BASED AUTOMATIC PROGRAMMING SYSTEM  
BASED ON FINITE STATE MACHINE DESIGN

1. INTRODUCTION

1.1 Overview

The theoretical basis of finite state machine synthesis was developed originally by G.H. Mealy (21), building on the work of D.R. Huffman (15) and F.E. Moore (22). The objective of Mealy's research was to develop a formal method of synthesis to replace the intuitive approach commonly used. The systematic approach enables the designer to formulate an unambiguous statement of performance requirements which can then be translated into the completed design by following a sequence of quasi-algorithmic steps. The procedure imposes a useful discipline on the designer; namely, the process cannot proceed until the function of the circuit has been completely described in the form of a state transition table (hereafter simply called the state table). The conciseness of the state table yields obvious benefits in the documentation of the design. The algorithmic nature of the implementation procedure lends itself to automation on a computer (10). This paper describes an effort to extend these benefits to program design. The approach used here is to have the user submit the state table to an automated implementation procedure generator (a program generator). The result produced is a control program which directs execution to the proper subroutine for each state transition. The procedure supports modular

program development because it can be applied to the development of the subroutines as well.

## 2. BACKGROUND

### 2.1 Finite State Machine Design

The origins of finite state machine design can be traced to the efforts of practicing engineers to add rigor to the design of sequential circuits. Originally conceived as an aid to circuit design, the basic synthesizing method has been generalized and extended to apply to any finite-state machine (10, 12, 16, 20, 25, 26, 28). The method is now commonly used to analyze and/or synthesize many types of systems in communications, process control, data processing, electronics, and other applications.

The applicability of these concepts to software design has long been recognized (1, 5, 8, 17, 24). The applicability is confined to those processes which operate sequentially. In fact, many programs are not sequential, in the sense that any attempt to develop a state table yields a table which collapses to a single state. This is, of course, a truth table, and (to press the analogy further) such a program corresponds to a combinational circuit.

### 2.2 The Finite State Machine Model

Reference 11 provides a more rigorous mathematical formulation of a finite state machine. The purpose here is to provide an informal basis for the development of the application of the techniques to program generation.

The finite state machine, in its general form, can be represented as a black box with a fixed number of time-dependent

input and output variables. At time intervals signaled by a synchronizing source (called the clock), the input variables are sampled and the next appropriate response is generated at the outputs. The circuit differs from a combinational circuit in that the outputs depend on the past history of the input variables. To achieve the proper response to a sequence of input values, the machine contains memory cells, which have stored in them pertinent information about the previous sequence of input values. The status of these memory cells at any given time is called the STATE of the machine. By tagging input sequences with certain state numbers, the machine "remembers" the input sequence. To determine the correct output for the machine, it is sufficient to know the current state of the machine and the current value of the input variables. For a given state, an input to the machine generates an output and causes a transition to the next state.

Figure 1 shows the generalized block diagram of a finite state machine. The memory cells store the present state. The combinational logic computes the next state and the present outputs.

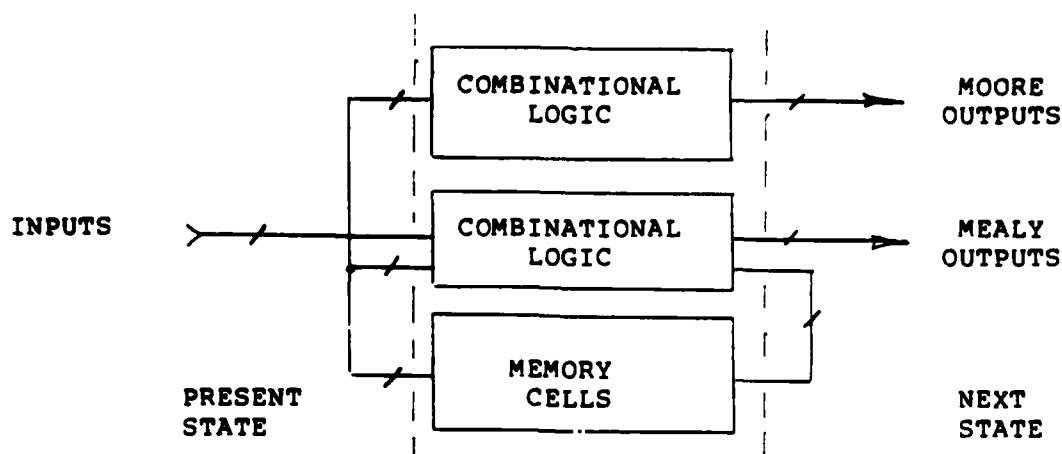


FIGURE 1. Generalized block diagram of circuit implementation of Mealy and Moore type finite state machine.

There is a slight difference in the generation of the output in the two types of machine as described by Mealy and by Moore. In the Mealy model, the current output value depends on the current state as well as the current input of the machine. In the Moore model, the current output depends only on the current state of the machine. For both, the next state transition mechanism is the same. The Mealy machine version never has more states than the corresponding Moore implementation, but this advantage comes at the expense of spurious output pulses caused by the need to have an output circuit respond to simultaneous input and state variable changes. With care, the Moore machine can be designed so that outputs are derived directly from memory cells, and thus these outputs can be made free of spurious pulses. However, the outputs in the Moore machine are delayed by one clock period, compared to the Mealy implementation of the same function.

Figure 2(a) shows a typical Mealy state table. Figure 2(b) is the corresponding Moore table for the same function (reference 14 describes how to perform Mealy/Moore table translations).

| PRES.<br>STATE | INPUT |   |   |   | INPUT  |   |   |   |
|----------------|-------|---|---|---|--------|---|---|---|
|                | 0     | 1 | 2 | 3 | 0      | 1 | 2 | 3 |
| A              | B     | C | C | A | 2      | 1 | 1 | 3 |
| B              | A     | B | B | D | 4      | 2 | 2 | 6 |
| C              | D     | D | C | C | 1      | 1 | 4 | 4 |
| D              | A     | B | C | C | 3      | 2 | 5 | 5 |
| NEXT STATE     |       |   |   |   | OUTPUT |   |   |   |

FIGURE 2(a).  
An example of a MEALY  
type state table.

| PRES.<br>STATE | INPUT |   |   |   |
|----------------|-------|---|---|---|
|                | 0     | 1 | 2 | 3 |
| 1              | 3     | 4 | 4 | 1 |
| 2              | 3     | 4 | 4 | 1 |
| 3              | 2     | 3 | 3 | 8 |
| 4              | 7     | 7 | 5 | 5 |
| 5              | 7     | 7 | 5 | 5 |
| 6              | 7     | 7 | 5 | 5 |
| 7              | 1     | 3 | 6 | 6 |
| 8              | 1     | 3 | 6 | 6 |
| NEXT STATE     |       |   |   |   |
| OUTPUT         |       |   |   |   |

FIGURE 2(B).  
The MOORE table  
corresponding to the  
table in Figure 2(a).

### 2.3 Synthesis Procedure

Once the state table has been formed, the following procedure is used to achieve a circuit realization. (This procedure implements the structure of Figure 1 in that it segregates the circuit into a sequential part, implemented by memory cells, and a combinational logic section. Thus the solution is ultimately reduces to a set of Karnaugh maps depicting the combinational circuit, which provide the next state conditions for the memory cells.)

1. Minimize the number of states, i.e., eliminate redundant states from the table.
2. Assign a unique binary code to each state. This step begins the process of converting the symbolic state table to a binary circuit implementation. This step is referred to as state assignment.
3. Refine new binary codes to redefine the state entries in the form of binary excitations applied to memory cell inputs.

4. Transfer the excitation table entries to memory cell input Karnaugh maps.

5. Derive memory cell input equations. These equations embody the circuit implementation.

The details of this synthesis procedure are described in any good textbook on logic design (9, 14). These details are not considered here because the primary goal of circuit design is to minimize circuit component count, whereas the program generator has broader objectives, namely, concise documentation, the production of unambiguous, error-free code, and the standardization of the program development phase.

#### 2.4 Application to Software Generation

In order to understand the application of the finite state machine model to program generation, we must first identify those mechanisms of the program which are analogous to the machine's synchronizing source, and the machine's inputs and outputs.

Interestingly, the synchronizing source could be implemented by either a real time clock which causes a periodic interrupt, or by a variable period default "clock" which corresponds to the execution time of the system program loop. A good example of a predetermined sampling period is a computer's real time multitasking operating system, in which a periodic interrupt (e.g., every 10 microseconds) takes the system to a task selection routine (23).

In the program generator application, the inputs are defined variously, ranging from real time binary input variables, monitored through a parallel input port, to complex system

conditions, analyzed and coded by a separate input subroutine.

Outputs can range from binary output signals appearing on parallel output ports, to system tasks implemented in subroutines or interrupt service routines.

The implications for program generation of the Mealy/Moore dichotomy center on convenience factors rather than circuit considerations. For example, the Mealy approach would seem to be appropriate when the number of states is to be minimized and/or when a system is task intensive, because in a Mealy implementation, the state table can contain fewer states than tasks. This advantage might be offset, however, by the fact that the Mealy model requires two matrices to be stored, one for the next state transitions, the other for the tasks. The Moore model has the advantage that it is easier to catalog the states and tasks, because each state is associated with a task, whereas the Mealy table associates a number of tasks with each state, corresponding to each of the input conditions.

The procedure for going from state table to implementation, outlined in section 2.3, has minimal applicability to the problem of program generation. The reason is that this procedure becomes unwieldy when applied to large state tables. Most practical software systems will contain hundreds, or even thousands, of states. For example, the sewing machine controller in reference 18 has 235 states.

The programs in appendices A and B, written in BASIC, demonstrate some of the concepts just discussed, including the difference between Mealy and Moore implementations. These

programs implement the state tables presented in Figure 2. In both programs, lines 400 through 700 form the program loop, and thus define the software "clock" period. Notice also that in the Mealy version, the determination of the next state is postponed until after the required task is executed.

### 3. STATE DIAGRAM PROGRAM GENERATOR

#### 3.1 Program Overview

The purpose of the State Diagram Program Generator is to provide a vehicle for developing computer programs which can be defined with finite state machine techniques. The program builds a table of user-designated names for program variables and their meanings. It then builds a state diagram table with cells for each state, resulting from each permutation of the variables for that state. When the state diagram has been created, the user is asked to enter the identifier of the subroutines/modules associated with each state. The user is expected to have a library of pre-coded modules and/or subroutines which perform the processing for the various states. With the user's subroutine library and the tables produced by the previous functions, the program creates a source code program which contains a state diagram driver and the user's routines. The program described in this paper and listed in Appendix C is a prototype version developed to demonstrate the feasibility of the technique. It is written in CBasic for operation on the CP/M operating system. (CBasic and CP/M are registered trademarks of Digital Research.) The only customization required for various processors is modification of the screen clear command for the user's terminal, which is specified in the data definition section of the program.

#### 3.2 Primary Menu

The program has a menu which is displayed each time the program is invoked and each time a step is completed. The menu display asks the user to select one of the program's basic

functions:

- 1) instruction display
- 2) variable (state vector) definition
- 3) state diagram specification
- 4) subroutine identification
- 5) source program creation
- 6) session termination

If the user enters an invalid response, the menu is redisplayed; otherwise program control is transferred to the specified function. When the function processors return control to the menu processor, they set status and error messages which become part of the menu display.

### 3.3 Instruction Display

When the user selects this option, a brief description of each function is displayed. The information remains on the screen until the user returns to the primary menu display by pressing the Return key.

### 3.4 Definition of the Variables

This portion of the program allows the user to describe the variables that will be used in the state table.

The user is asked if a predefined set of variables should be used. If the response is affirmative, the user is prompted for the name of the file where the set is stored. If the file is not found, an error message is displayed and the user is again asked if a predefined set of variables should be used. When a valid file name is entered, the data in the file is read into the Varlist array and the variable specification step is skipped.

If the user does not want to use a pre-defined set of variables, the program prompts the user for information about each variable. A sample dialog follows, with user entries in

boldface.

```
WHAT IS THE NAME OF VARIABLE 1?  
master switch  
WHAT DOES 0 MEAN FOR MASTER SWITCH?  
off  
WHAT DOES 1 MEAN FOR MASTER SWITCH?  
on  
WHAT IS THE NAME OF VARIABLE 2?  
....
```

The user's responses are entered into the Varlist array. Each item in the array contains the user's name for the variable, and the names assigned to the 0 and 1 states.

When the Varlist array has been filled in from user responses or from a pre-defined file, the user is asked if a display of the variables and their definitions is wanted. If the answer is yes, the permutations of all of the variables are output in the user's terminology. For example, one value of the state vector might be displayed as:

```
Vector 9      1 Master Switch = On  
              0 Safety Switch = Off  
              0 Target Status = Inactive  
              1 Damage Assessment = Neutralized
```

This display (and others) can be printed by entering the CP/M print toggle command (Control-P).

The user is then asked if the current set of variable definitions should be saved for use in another program run. If the answer is yes, the user is asked to specify a file name. If the file already exists, the user is given the choice of replacing the existing file or of specifying another file name.

In the final step of the variable definition process, the program sets the variable definition flag, sets up a status message ("Variable Definition Completed") for display on the menu, and returns control to the primary menu.

### 3.5 State Diagram Creation

The state diagram creation step determines if a state diagram defined in a previous session should be used, solicits state names for new diagrams, guides the user in specifying what action should be taken for each permutation of the state vector, displays the resulting state diagram, and provides an option to save the diagram for use in a subsequent program run.

The processor checks the variable definition flag to ensure that the user has already performed the variable definition step. If it is off, an error message is set up for display on the menu and control is returned to the primary menu processor.

If the variables have been defined, the user is asked if a predefined state diagram should be used. If so, the user is prompted for the name of the file where the diagram is stored. If the file is not found, an error message is displayed and the user is asked again if a predefined diagram should be used. When a valid file name is entered, the data in it is read into the Diagram array and the user is asked to specify which state is the initial program state when the target program starts executing. The state definition phase is skipped.

When a new state diagram is being built, the user is asked to enter the name of each state and to specify which state is the initial program state for the target program. The user is then asked if there are any general input conditions that apply to all states. If, for example, control always passes to the same state whenever the master switch is off, the user could specify this as a general input condition and thereby eliminate several entries

in the next state diagram definition step. If there are general input variables, the user is asked for the name of the variable, and for the name of the state processing each value of the variable. Checks are performed to validate the variable and state names and to ensure that there is no conflict with an action specified for a previously defined general input variable.

The next step asks the user a series of questions for each state. First, the user is asked if there are any general input variables for the state. If there are, the processing is similar to that for general variables applying to all states. In cases of conflict, a general variable result for all states will override a result specified for a specific state. Second, the user is asked to specify the outcome for all state vectors which have not been predetermined by the general input conditions. A sample dialog, with user responses in boldface, follows (wait and restart are names of steps).

```

YOU ARE IN THE WAIT STATE
VECTOR = 9      1 MASTER SWITCH = ON
                 0 SAFETY SWITCH = OFF
                 0 TARGET STATUS = INACTIVE
                 1 DAMAGE ASSESSMENT = NEUTRALIZED
WHAT STATE DO YOU WANT TO BE IN NEXT?
wait
YOU ARE IN THE WAIT STATE
VECTOR = 10     1 MASTER SWITCH = ON
                 0 SAFETY SWITCH = OFF
                 1 TARGET STATUS = ACTIVE
                 0 DAMAGE ASSESSMENT = NOT NEUTRALIZED
WHAT STATE DO YOU WANT TO BE IN NEXT?
restart
.....

```

For each possible value of the state vector within a state, the user must specify the next processing state. When this has been completed (or when a previously defined diagram has been read in), the user is given the option of having the state

diagram displayed. The user also has the option of saving the diagram for use in a subsequent program run.

In the final step of the diagram creation process, the program sets the diagram build flag, sets up a status message ("State Diagram Completed") for display on the menu, and returns control to the primary menu.

### 3.6 Subroutine Identification

The subroutine identification step gets the name of the user's subroutine library, scans the library to collect a list of subroutine labels, asks the user for the label of the subroutine which is to process each state, and displays the resulting subroutine state diagram.

The function first checks the variable definition and diagram build flags. If either flag has not been set, an error message is set up for display on the menu, and control is returned to the menu processor.

If the variables have been defined and the state diagram has been created, the user is asked for the name of the subroutine library to be used in building the target program. The user is expected to provide a file that includes a subroutine/module for executing each state specified in the state diagram. The program scans the library file to build a list of subroutine labels for subsequent validation routines. Each subroutine/module on the library must be preceded by a header record with the characters "REM#" in the first four positions, followed by the subroutine label.

The user is then prompted for the label of the subroutine to

be used for each state. If a label is specified which is not on the library, an error message is displayed. The program uses the labels entered to build a Jump table that corresponds to the Diagram table, substituting labels for state names.

When the subroutine version of the state diagram is completed, the user is given the option of having it displayed.

In the final step of the subroutine identification process, the program sets the subroutine identification flag, sets up a status message ("Subroutine Identification Completed") for display on the menu, and returns control to the menu processor.

### 3.7 Creating the Source Program

The source program creation module uses a special skeleton driver program to create the source program file. The skeleton driver is merged with data collected in the previous steps and with the user's subroutine library to generate the target source program.

The user is asked for a file name for the new source program. If the file already exists, the user is given the option of replacing the file or of specifying another file name.

A standard skeleton control program is used to generate code for all programs. (It is listed in Appendix D.) Its control logic examines the the jump table to determine which module should be called next, based on the current state and the current condition of the state vector. The control program for the prototype program expects subroutines to set the values in the state vector. A real-time program would, of course, examine its input lines to determine the state vector.

The file containing the skeleton driver program file is

read and each program line is copied to the target source program file until an "insert" flag is read. Whenever an insert flag is found, specific data collected in previous steps is inserted into the driver program. Insert data includes variables such as the number of states in the target program and the size of the jump table, the jump table itself, and other application-specific data.

When all of the inserts have been processed, the user's subroutine library is appended to the target program source file. A program created by the State Diagram Program Generator is listed in Appendix E.

In the final step, the program sets up a status message ("Source Program Saved on Disk...") for display on the menu, and returns control to the menu processor. The user must exit the State Diagram Program Generator to compile and execute the target program.

## BIBLIOGRAPHY

1. Barnes, B.H. and J.R. Metzner, DECISION TABLE LANGUAGES AND SYSTEMS, Academic Press, New York, 1977.
2. Beizer, Boris, THE ARCHITECTURE AND ENGINEERING OF DIGITAL COMPUTER COMPLEXES, Vol. 1, Plenum Press, New York-London, 1971.
3. Biermann, Alan W., "Approaches to Automatic Programming, "ADVANCES IN COMPUTERS, Vol. 15, pp. 1-63, Academic Press, New York, 1976.
4. Biermann, Alan W., and R. Krishnaswamy, "Constructing Programs from Example Computation, "IEEE TRANS. ON SOFTWARE ENGINEERING, Vol. SE-2, pp. 141-153, (Sept. 1976).
5. Birke, D.M., "State Transition Programming Techniques, and Their Use in Producing Teleprocessing Device Control Programs, "IEEE TRANS. ON COMMUNICATIONS, Vol. COM-20, No.3, pp.569-575, (June 1972).
6. Chow, T.S., "Testing Software Design Modelled by Finite State Machines, "IEEE TRANS. ON SOFTWARE ENGINEERING, Vol. SE-4, No.3, p.178, (May 1978).
7. Clare, C.R., DESIGNING LOGIC SYSTEMS USING STATE MACHINES, McGraw-Hill, 1973.
8. Conway, M.E., "Design of a Separate Transition-diagram Compiler, "COMMUNICATIONS OF THE ACM, Vol. 6, pp. 396-408, (July 1953).
9. Dietmeyer, Donald L., LOGIC DESIGN OF DIGITAL SYSTEMS, 2nd ed., Allyn and Bacon, Boston, 1979.
10. Dolotta, T.A., and E.J. McCluskey, "The Coding of Internal States of Sequential Circuits, "IEEE TRANS. ON ELECTRONIC COMPUTERS, Vol. EC-13, No.5, pp. 549-562, (October 1964).
11. Gill, Arthur, INTRO. TO THE THEORY OF FINITE STATE MACHINES, McGraw-Hill, New York, 1962.
12. Grasselli, A., and F. Luccio, "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks, "IEEE TRANS. ON ELECTRONIC COMPUTERS, Vol. EC-14, No. 3, pp. 330-359, (June 1965).
13. Heidorn, G.E., "Automatic Programming Through Natural Language Dialogue: A Survey, "IBM J. OF RESEARCH AND DEVELOPMENT, Vol. 20, pp. 302-313, (July 1976).
14. Hill, F.J., and G.R. Peterson, INTRO. TO SWITCHING THEORY

AND LOGIC DESIGN, 2nd Ed., Wiley, 1974.

15. Huffman, D.A., "The Synthesis of Sequential Switching Circuits," J. FRANKLIN INST., Vol. 257, No.3, pp.161-190; No.4, pp. 275-303, (March-April 1954).
16. Karp, R.M., "Some Techniques of State Assignment for Synchronous Sequential Machines," IEEE TRANS. ON ELECTRONIC COMPUTERS, Vol. EC-13, No.5, pp. 507-518, (October 1964).
17. King, P.J.H., and R.G. Johnson, Some Comments On the Use of Ambiguous Decision Tables and Their Conversion to Computer Programs, "COMMUNICATIONS OF THE ACM, Vol. 16, No.5, pp. 287-290, (May 1973).
18. Landau, Jack V., "Hardware Oriented State Description Techniques," in D.F. Stout (Ed.), MICROPROCESSOR APPLICATIONS HANDBOOK, McGraw-Hill, 1982.
19. Manna, A., and R.J. Waldinger, "Toward Automatic Program Synthesis," "COMMUNICATIONS OF THE ACM, Vol.14, pp. 151-165, (March 1971).
20. McNaughton, R., and H. Yamada, "Regular Expressions and State Graphs for Automata," IRE TRANS. ON ELECTRONIC COMPUTERS, Vol. EC-9, No. 1, pp. 39-47, (March 1960).
21. Mealy, G.H., "A Method for Synthesizing Sequential Circuits," BELL SYSTEM TECH. J., Vol. 34, No. 5, pp. 1045-1080, (September 1955).
22. Moore, E.F., "Gedanken Experiments on Sequential Machines," in C.E. Shannon and J. McCarthy (Eds.) AUTOMATA STUDIES, Princeton Univ. Press, Princeton, N.J., 1956.
23. Motorola, Inc., USERS GUIDE; EXORMACS REAL TIME MULTITASKING SYSTEM, 1981.
24. Parnas, D.L., "On the Use of Transition Diagrams in the Design of a User Interface For an Interactive Computer System," PROC. OF THE 24TH NATIONAL CONFERENCE OF THE ACM, pp.379-385, 1969.
25. Paull, M.C., and S.H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions," IRE TRANS. ON ELECTRONIC COMPUTERS, Vol. EC-8, No.3, pp.356-357, (September 1959).
26. Robin, M.O., and D. Scott, "Finite Automata and Their Decision Problems," IBM J. OF RESEARCH AND DEVELOPMENT, Vol.3, No.2, pp.114-125, (April 1959).
27. Slagle, J.R., ARTIFICIAL INTELLIGENCE: THE HEURISTIC

PROGRAMMING APPROACH, McGraw-Hill, New York, 1971.

28. Stearns, R.E., and J. Hartmanis, "On the State Assignment Problem For Sequential Machines, II, "IRE TRANS. ON ELECTRONIC COMPUTERS, Vol. EC-10, No. 4, pp.593-603, (December 1961).

## Appendix A

```

1  REM APPENDIX A
5  REM PROGRAM IMPLEMENTING MEALY MACHINE
10 DIM NXT%(4,3):REM MATRIX WHICH STORES STATE TRANSITION TABLE
20 DIM TASK%(4,3):REM MATRIX WHICH STORES TASKS <OUTPUTS>
25 REM READ IN STATE TABLE
30 FOR S=1 TO 4:FOR I=0 TO 3
40 READ X%:NXT%(S,I)=X%:NEXT I :NEXT S
45 REM SET UP TASK ARRAY
50 FOR S=1 TO 4:FOR I=0 TO 3
60 READ X%:TASK%(S,I)=X%:NEXT I:NEXT S
90 REM NEXT STATE MATRIX
100 DATA 2,3,3,1
110 DATA 1,2,2,4
120 DATA 4,4,3,3
130 DATA 1,2,3,3
190 REM TASK MATRIX
200 DATA 2,1,1,3
210 DATA 4,2,2,6
220 DATA 1,1,4,4
230 DATA 3,2,5,5
300 PRES%=1:REM START IN STATE 1
350 PRINT"STATE"PRES%;
400 GOSUB 9000:REM GO TO INPUT SUBROUTINE
500 TSK%=TASK%(PRES%,INP%)
600 ON TSK% GOSUB 1000,2000,3000,4000,5000,6000
700 PRES%=NXT%(PRES%,INP%):PRINT" STATE"PRES%;:GOTO 400
999 REM DUMMY TASK SUBROUTINES
1000 PRINT"TSK 1";:RETURN
2000 PRINT"TSK 2";:RETURN
3000 PRINT"TSK 3";:RETURN
4000 PRINT"TSK 4";:RETURN
5000 PRINT"TSK 5";:RETURN
6000 PRINT"TSK 6";:RETURN
9000 READ INP%:PRINT"INPUT"INP%:IF INP%=1 THEN PRINT"DONE":END
9003 GET PS:IF PS="" THEN 9003:REM HIT ANY KEY TO PROCEED TO NEXT TASK
9005 RETURN
9009 REM INPUT SEQUENCE TO VISIT ALL NEXT STATE ENTRIES ON STATE TABLE
9010 DATA 3,0,1,2,0,1,2,3,1,1,3,0,2,0,2,0,3,-1

```

# APPENDIX B

```

1  REM APPENDIX B
4  REM PROGRAM IMPLEMENTING MOORE TABLE
5  DIM NXT$(8,7):REM MATRIX WHICH STORES STATE TRANSITION TABLE
10 FOR S=1 TO 8:FOR I=0 TO 3:REM READ IN STATE TABLE
20 READ X$:NXT$(S,I)=X$
30 NEXT I:NEXT S
35 REM NEXT STATE MATRIX
40 DATA 3,4,4,1
50 DATA 3,4,4,1
60 DATA 2,3,3,8
70 DATA 7,7,5,5
80 DATA 7,7,5,5
90 DATA 7,7,5,5
92 DATA 1,3,6,6
94 DATA 1,3,6,6
100 PRES%=1:REM START IN STATE 1
200 GOTO 600
400 GOSUB 9000:REM GO TO INPUT SUBROUTINE
500 PRES%=NXT$(PRES%,INP%):REM GET NEXT STATE
600 ON PRES% GOSUB 1000,2000,3000,4000,5000,6000,7000,8000
700 GOTO 400
999 REM DUMMY TASK SUBROUTINES
1000 PRINT"STATE 1 TSK 3":RETURN
2000 PRINT"STATE 2 TSK 4":RETURN
3000 PRINT"STATE 3 TSK 2":RETURN
4000 PRINT"STATE 4 TSK 1":RETURN
5000 PRINT"STATE 5 TSK 4":RETURN
6000 PRINT"STATE 6 TSK 5":RETURN
7000 PRINT"STATE 7 TSK 1":RETURN
8000 PRINT"STATE 8 TSK 6":RETURN
8999 REM THE INPUT SUBROUTINE
9000 READ INP$:PRINT" INPUT"INP$:IF INP%=1 THEN PRINT"DONE":END
9003 GET PS:IF PS="" THEN 9003:REM HIT ANY KEY TO PROCEED TO NEXT STATE
9005 RETURN
9009 REM INPUT SEQUENCE TO VISIT ALL NEXT STATE ENTRIES ON STATE TABLE
9010 DATA 3,0,1,2,0,3,1,2,2,3,0,0,2,1,3,1,2,0,1,3,1,3,2
9020 DATA 2,1,1,3,3,3,0,1,0,0,3,0,1,0,1,0,1,3,1,1,0,2,-1

```

# APPENDIX C

```

REM*****
REM* PROGRAM:   STATE DIAGRAM PROGRAM GENERATOR      *
REM* AUTHOR:    JEANNINE WOLF, COMPUTER SYSTEM ASSOCIATES  *
REM* CONTRACT:  ARMY RESEARCH ORGANIZATION DAAG-29-81-C-0021 *
REM*****

```

100

```

REM*****
REM*           DATA DEFINITION AND INITIALIZATION      *
REM*****
ON%=-1
OFF%=0
PROG.RUN%=ON%
MAX.VARS%=8
MAX.STATES%=10
MAX.SUB.LABELS%=500
SUB.HEADS="REM#"
STATUS.MSGS=""
SPACES=" "
NULLS=""
JUMP.TBL.SIZE%=2^MAX.VARS%+1
DEFINE.VARS%=OFF%
BUILD.STATE%=OFF%
IDENTIFY.SUBS%=OFF%
SCREEN.CLEAR=CHRS(30)+CHRS(26)
HYPHENS$="-----"
DIM VARLISTS(MAX.VARS%,3)
DIM DIAGRAMS(MAX.STATES%,JUMP.TBL.SIZE%)
DIM JUMPS(MAX.STATES%,JUMP.TBL.SIZE%)
DIM SUBIDS(MAX.SUB.LABELS%)

```

2000

```
REM*****
REM*                               MAINLINE PROCESSOR                               *
REM*****
IF PROG.RUN%=OFF% THEN GOTO 99000
PRINT SCREEN.CLEAR$
PRINT
PRINT
PRINT
PRINT
PRINT TAB(25);"DISPLAY INSTRUCTIONS"
PRINT
PRINT TAB(25);"DEFINE VARIABLES"
PRINT
PRINT TAB(25);"BUILD STATE DIAGRAM"
PRINT
PRINT TAB(25);"IDENTIFY SUBROUTINES"
PRINT
PRINT TAB(25);"CREATE SOURCE PROGRAM"
PRINT
PRINT TAB(25);"EXIT SESSION"
PRINT
PRINT STATUS.MSG$
STATUS.MSG$=""
PRINT
PRINT
PRINT "SELECT OPERATION (ENTER 2 OR MORE CHARACTERS)"
PRINT
INPUT FUNCTIONS
IF LEFT$(FUNCTION$,2)="DE" THEN GOSUB 20000
IF LEFT$(FUNCTION$,2)="BU" THEN GOSUB 30000
IF LEFT$(FUNCTION$,2)="ID" THEN GOSUB 40000
IF LEFT$(FUNCTION$,2)="CR" THEN GOSUB 50000
IF LEFT$(FUNCTION$,2)="DI" THEN GOSUB 90000
IF LEFT$(FUNCTION$,2)="EX" THEN PROG.RUN%=OFF%
GOTO 2000                                \RESUME DISPLAY UNLESS END
```

```

20000  REM*****
      REM*          20000 - DEFINE VARIABLES          *
      REM*****
      REM VROW%=VARIABLE LIST ROW INDEX
      REM VCOL%=VARIABLE LIST COLUMN INDEX
      REM NCOL%=NAME COLUMN FOR VARIABLE LIST
      NCOL%=1
      NR.VARS%=0
      PRINT SCREEN.CLEAR$
      PRINT
      PRINT
      PRINT TAB(25);"DEFINE VARIABLES"
      PRINT

21000  REM*****SEE IF EXISTING FILE SHOULD BE USED*****
      PRINT "DO YOU WANT TO USE A PRE-DEFINED SET OF VARIABLES? (Y/N)"
      INPUT TEMPS
      IF TEMPS="N" THEN GOTO 22000
      PRINT "ENTER THE FILE NAME FOR THE VARIABLE SET"
      INPUT VAR.FILE.NAMES
      IF SIZE(VAR.FILE.NAMES)=0 THEN \
        PRINT "***NO FILE FOR ";VAR.FILE.NAMES :\
        GOTO 21000
      OPEN VAR.FILE.NAMES AS 1
      FOR VROW%=1 TO MAX.VARS%
        FOR VCOL%=1 TO 3
          READ #1; VARLIST$(VROW%,VCOL%)
          IF END #1 THEN 21900
        NEXT VCOL%
      NEXT VROW%
      NR.VARS%=MAX.VARS%
21900  IF NR.VARS%=0 THEN NR.VARS%=VROW%-1
      CLOSE 1
      GOTO 25000

22000  REM*****SOLICIT VARIABLE NAMES AND MEANINGS*****
      PRINT
      PRINT "YOU WILL BE ASKED THREE QUESTIONS ABOUT EACH STATE VARIABLE."
      PRINT "PRESS RETURN TO END THE DIALOG"
      PRINT
      FOR VROW%=1 TO MAX.VARS%
        PRINT "WHAT IS THE NAME OF VARIABLE ";VROW%
        INPUT LINE VAR.NAMES
        IF VAR.NAMES=NULL$ THEN \
          NR.VARS%=VROW%-1 :\
          GOTO 25000
        VARLIST$(VROW%,NCOL%)=VAR.NAMES
        PRINT "WHAT DOES 0 MEAN FOR "; VARLIST$(VROW%,NCOL%)
        INPUT VARLIST$(VROW%,2)
        PRINT "WHAT DOES 1 MEAN FOR "; VARLIST$(VROW%,NCOL%)
        INPUT VARLIST$(VROW%,3)
      NEXT VROW%

```

```

NR.VARS%=MAX.VARS%

25000 REM*****PRINT THE MEANING OF EACH VECTOR*****
PRINT "DO YOU WANT TO PRINT THE VARIABLES AND DEFINITIONS? (Y/N)"
INPUT TEMPS
IF TEMPS="N" THEN GOTO 27000
FOR VAL.LOOP%=1 TO 2^NR.VARS%
  VECTOR%=VAL.LOOP%-1
  MASK%=(2^NR.VARS%)/2 REM INITIALIZE VECTOR BIT MASK
  PRINT
  PRINT "VECTOR";VECTOR%;
  FOR VROW%=1 TO NR.VARS%
    RESULT%=VECTOR% AND MASK%
    IF RESULT% = 0 THEN \
      VCOL%=2 \
    ELSE \
      RESULT%=1 :\
      VCOL%=3
    MASK%=MASK%/2 REM RESET MASK FOR NEXT BIT POS
    PRINT TAB(15);RESULT%;VARLISTS(VROW%,NCOL%);"="; \
      VARLIST$(VROW%,VCOL%)
  NEXT VROW%
  PRINT
NEXT VAL.LOOP%
PRINT
INPUT "PRESS RETURN TO CONTINUE";LINE TEMPS

27000 REM*****OPTION TO SAVE VARIABLES ON DISK*****
PRINT "DO YOU WANT TO SAVE THIS SET OF VARIABLES? (Y/N)"
INPUT TEMPS
IF TEMPS="N" THEN GOTO 29000
27100 PRINT "ENTER THE FILE NAME FOR THIS VARIABLE SET"
INPUT VAR.FILE.NAMES
IF SIZE(VAR.FILE.NAMES)=0 THEN GOTO 27500
PRINT "DO YOU WANT TO REPLACE THE CURRENT ";VAR.FILE.NAMES;"? (Y/N)"
INPUT TEMPS
IF TEMPS="N" THEN GOTO 27100
27500 CREATE VAR.FILE.NAMES AS 1
FOR VROW%=1 TO NR.VARS%
  FOR VCOL%=1 TO 3
    PRINT #1; VARLISTS(VROW%,VCOL%)
  NEXT VCOL%
NEXT VROW%
CLOSE 1

29000 REM*****WRAP-UP PROCESSING*****
IF NR.VARS% > 0 THEN DEFINE.VARS%=ON%
STATUS.MSCS = "****VARIABLE DEFINITION COMPLETED"
29999 RETURN

```

```

30000 REM*****
REM*          30000 - BUILD STATE DIAGRAM          *
REM*****
REM DROW%=STATE DIAGRAM ROW INDEX
REM DCOL%=STATE DIAGRAM COLUMN INDEX
REM VROW%=VARIABLE LIST ROW INDEX
REM VCOL%=VARIABLE LIST COLUMN INDEX
REM NCOL%=NAME COLUMN FOR BOTH STATE DIAGRAM AND VARIABLE LIST
NCOL%=1
NR.STATES%=0
IF DEFINE.VARS%=OFF% THEN GOTO 39050

31000 REM*****SEE IF EXISTING FILE SHOULD BE USED *****
PRINT "DO YOU WANT TO USE A PRE-DEFINED STATE DIAGRAM? (Y/N)"
INPUT TEMPS
IF TEMPS="N" THEN GOTO 33000
PRINT "ENTER THE FILE NAME FOR THIS STATE DIAGRAM"
INPUT DIAGRAM.FILE.NAMES
IF SIZE(DIAGRAM.FILE.NAMES)=0 THEN \
    PRINT "****NO FILE FOR ";DIAGRAM.FILE.NAMES :\
    GOTO 31000
OPEN DIAGRAM.FILE.NAMES AS 1
FOR DROW%=1 TO MAX.STATES%
    FOR DCOL%=1 TO 2^NR.VARS%+1
        READ #1; DIAGRAMS(DROW%,DCOL%)
        IF END #1 THEN 31100
    NEXT DCOL%
NEXT DROW%
NR.STATES%=MAX.STATES%
31100 IF NR.STATES%=0 THEN NR.STATES%=DROW%-1
CLOSE 1
31200 PRINT "WHICH STATE IS THE INITIAL PROGRAM STATE?"
INPUT INITIAL.PROG.STATES
MATCH%=OFF%                                REM VALIDATE STATE NAME
FOR DROW%=1 TO NR.STATES%
    IF DIAGRAMS(DROW%,NCOL%)=INITIAL.PROG.STATES THEN \
        MATCH%=DROW%
NEXT DROW%
IF MATCH%=OFF% THEN \
    PRINT "****";INITIAL.PROG.STATES;" IS NOT A VALID STATE" :\
    GOTO 31200
GOTO 35000

33000 REM ****SOLICIT STATE NAMES AND GENERAL VARIABLES*****
PRINT SCREEN.CLEAR$
PRINT
PRINT
PRINT
PRINT TAB(25);"BUILD STATE DIAGRAM"
PRINT
PRINT "YOU WILL BE ASKED FOR THE NAME OF EACH STATE."
PRINT "PRESS RETURN TO END THE DIALOG"

```

```

PRINT
NCOL%=1
FOR DROW%=1 TO MAX.STATES%
  PRINT "WHAT IS THE NAME OF STATE "; DROW%
  INPUT LINE STATE.NAMES
  IF STATE.NAMES=NULLS THEN \
    NR.STATES%=DROW%-1 :\
    GOTO 33100 \
  ELSE \
    DIAGRAMS(DROW%,NCOL%)=STATE.NAMES
NEXT DROW%
NR.STATES%=MAX.STATES%
33100 PRINT "WHICH STATE IS THE INITIAL PROGRAM STATE?"
INPUT INITIAL.PROG.STATES
MATCH%=OFF% REM VALIDATE STATE NAME
FOR DROW%=1 TO NR.STATES%
  IF DIAGRAMS(DROW%,NCOL%)=INITIAL.PROG.STATES THEN \
    MATCH%=DROW%
NEXT DROW%
IF MATCH%=OFF% THEN \
  PRINT "****";INITIAL.PROG.STATES;" IS NOT A VALID STATE" :\
  GOTO 33100
33200 PRINT "ARE THERE ANY GENERAL INPUT CONDITIONS THAT APPLY TO ";
PRINT "ALL STATES? (Y/N)"
INPUT TEMPS
IF TEMPS="N" THEN GOTO 34000
33300 PRINT "WHICH VARIABLE?"
INPUT LINE TEMPS
IF TEMPS=NULLS THEN GOTO 34000
MATCH%=OFF% REM CHECK FOR VALID VARIABLE
FOR VROW%=1 TO NR.VARS%
  IF VARLISTS(VROW%,NCOL%)=TEMPS THEN MATCH%=VROW%
NEXT VROW%
IF MATCH%=OFF% THEN \
  PRINT "****";TEMPS;" IS NOT A VALID VARIABLE" : \
  GOTO 33300
VROW%=MATCH%
FOR VCOL%=2 TO 3
33400 PRINT "WHAT STATE DO YOU WANT TO BE IN WHEN ";
PRINT VARLISTS(VROW%,NCOL%);"=";VARLISTS(VROW%,VCOL%);"?"
PRINT "PRESS RETURN IF STATE WILL VARY"
INPUT LINE TEMPS
IF TEMPS=NULLS THEN GOTO 33500
MATCH%=OFF% REM VALIDATE STATE NAME
FOR DROW%=1 TO NR.STATES%
  IF DIAGRAMS(DROW%,NCOL%)=TEMPS THEN \
    MATCH%=DROW%
NEXT DROW%
IF MATCH%=OFF% THEN \
  PRINT "****";TEMPS;" IS NOT A VALID STATE" :\
  GOTO 33400
MASK%=(2^(NR.VARS%+1-VROW%))/2
FOR VAL.LOOP%=1 TO 2^NR.VARS%
  MATCH%=OFF%
  VECTOR%=VAL.LOOP%-1

```

```

RESULT%=VECTOR% AND MASK%
IF (RESULT%=0 AND VCOL%=2) OR (RESULT%<>0 AND VCOL%=3) THEN \
  DCOL%=VECTOR%+2 : \
  MATCH%=ON%
FOR DROW%=1 TO NR.STATES%
  IF (MATCH%=ON%) AND (DIAGRAMS(DROW%,DCOL%)<>NULLS) AND \
    (DIAGRAMS(DROW%,DCOL%)<>TEMPS) THEN \
    PRINT "****CONFLICTING GENERAL VARIABLE ALREADY "; : \
    PRINT "DEFINED FOR "; : \
    PRINT DIAGRAMS(DROW%,NCOL%);" VECTOR";VECTOR%
  IF (MATCH%=ON%) AND (DIAGRAMS(DROW%,DCOL%)=NULLS) THEN \
    DIAGRAMS(DROW%,DCOL%)=TEMPS
  NEXT DROW%
NEXT VAL.LOOP%
33500 NEXT VCOL%
PRINT "MORE GENERAL INPUT VARIABLES? (Y/N)"
INPUT TEMPS
IF TEMPS="Y" THEN GOTO 33300

34000 REM*****GET NEXT STATE FOR EACH VECTOR*****
PRINT SCREEN.CLEAR$
PRINT "THE NEXT SERIES OF QUESTIONS WILL BE REPEATED FOR EACH STATE."
FOR DROW%=1 TO NR.STATES%
  DCOL%=2
  PRINT
  PRINT "-----START STATE DEFINITION-----"
  PRINT
  PRINT "YOU ARE IN THE ";DIAGRAMS(DROW%,NCOL%);" STATE"
  PRINT "ARE THERE ANY GENERAL INPUT VARIABLES FOR THIS STATE? (Y/N)"
  INPUT TEMPS
  IF TEMPS="N" THEN GOTO 34300
  PRINT "WHICH VARIABLE?"
  INPUT LINE TEMPS
  IF TEMPS=NULLS THEN GOTO 34300
  MATCH%=OFF%
  REM CHECK FOR VALID VARIABLE
  FOR VROW%=1 TO NR.VARS%
    IF VARLIST$(VROW%,NCOL%)=TEMPS THEN MATCH%=VROW%
  NEXT VROW%
  IF MATCH%=OFF% THEN \
    PRINT "****";TEMPS;" IS NOT A VALID VARIABLE" : \
    GOTO 34100
  VROW%=MATCH%
  FOR VCOL%=2 TO 3
    PRINT "WHAT STATE DO YOU WANT TO BE IN WHEN ";
    PRINT VARLIST$(VROW%,NCOL%);"=";VARLIST$(VROW%,VCOL%);"?"
    PRINT "PRESS RETURN IF STATE WILL VARY"
    INPUT LINE TEMPS
    IF TEMPS=NULLS THEN GOTO 34250
    MATCH%=OFF%
    REM CHECK FOR VALID STATE NAME
    FOR DROW1%=1 TO NR.STATES%
      IF DIAGRAMS(DROW1%,NCOL%)=TEMPS THEN \
        MATCH%=ON%
    NEXT DROW1%
    IF MATCH%=OFF% THEN \
      PRINT "****";TEMPS;" IS NOT A VALID STATE, "; : \

```

```

        PRINT "RE-ENTER THE STATE NAME" :\
        GOTO 34210
    MASK%=(2^(NR.VARS%+1-VROW%))/2
    FOR VAL.LOOP%=1 TO 2^NR.VARS%
        MATCH%=OFF%
        VECTOR%=VAL.LOOP%-1
        RESULT%=VECTOR% AND MASK%
        IF (RESULT%=0 AND VCOL%=2) OR (RESULT%<>0 AND VCOL%=3) \
            THEN \
                MATCH%=ON% :\
                DCOL1%=VECTOR%+2
        IF (MATCH%=ON%) AND (DIAGRAMS(DROW%,DCOL1%)=NULLS) THEN \
            DIAGRAMS(DROW%,DCOL1%)=TEMPS
    NEXT VAL.LOOP%
34250 NEXT VCOL%
    PRINT "MORE GENERAL VARIABLES FOR THIS STATE? (Y/N)"
    INPUT TEMPS
    IF TEMPS="Y" THEN GOTO 34100

34300 FOR VAL.LOOP%=1 TO 2^NR.VARS%
    IF DIAGRAMS(DROW%,DCOL%)<>NULLS THEN GOTO 34500
    VECTOR%=VAL.LOOP%-1
    MASK%=(2^NR.VARS%)/2      REM INITIALIZE VECTOR BIT MASK
                                REM E.G. 100,010,001

    PRINT
    PRINT "YOU ARE IN THE ";DIAGRAMS(DROW%,NCOL%);" STATE, ";
    PRINT "VECTOR =";VECTOR%
    FOR VROW%=1 TO NR.VARS%
        RESULT%=VECTOR% AND MASK%
        IF RESULT% = 0 THEN \
            VCOL%=2 \
        ELSE \
            VCOL%=3
        MASK%=MASK%/2      REM RESET MASK FOR NEXT BIT POS
        PRINT TAB(8);VARLISTS(VROW%,NCOL%);"=";
        PRINT VARLISTS(VROW%,VCOL%)
    NEXT VROW%
    PRINT "WHAT STATE DO YOU WANT TO BE IN NEXT?"
34400 INPUT STATE.NAMES
    MATCH%=OFF%      REM CHECK FOR VALID STATE NAME
    FOR DROW1%=1 TO NR.STATES%
        IF DIAGRAMS(DROW1%,NCOL%)=STATE.NAMES THEN \
            MATCH%=ON%
    NEXT DROW1%
    IF MATCH%=OFF% THEN \
        PRINT "****";STATE.NAMES;" IS NOT A STATE, "; :\
        PRINT "RE-ENTER THE STATE NAME" :\
        GOTO 34400
    DIAGRAMS(DROW%,DCOL%)=STATE.NAMES
34500 DCOL%=DCOL% + 1
    NEXT VAL.LOOP%
NEXT DROW%

```

```

35000 REM*****PRINT THE STATE DIAGRAM*****
PRINT "DO YOU WANT TO DISPLAY THE STATE DIAGRAM? (Y/N)"
INPUT TEMPS
IF TEMPS="N" THEN GOTO 37000
IF NR.STATES% > 7 THEN \
    PRINT.LIMIT% = 7 \
ELSE \
    PRINT.LIMIT% = NR.STATES%
PRINT SCREEN.CLEAR$
REM PRINT DIAGRAM FOR STATES 1-7
PRINT
PRINT TAB(1);LEFT$(HYPHEN$S,6);"  ";
FOR DROW%=1 TO PRINT.LIMIT%
    PRINT LEFT$(HYPHEN$S,10);
NEXT DROW%
TPOS%=(PRINT.LIMIT% * 10) / 2
PRINT TAB(1);"INPUT";TAB(TPOS%);"N E X T   S T A T E"
TPOS%=10
PRINT "VECTOR";
FOR DROW%=1 TO PRINT.LIMIT%
    PRINT TAB(TPOS%);LEFT$((DIAGRAMS(DROW%,1)),8);
    TPOS%=TPOS% + 10
NEXT DROW%
PRINT TAB(1);LEFT$(HYPHEN$S,6);"  ";
FOR DROW%=1 TO PRINT.LIMIT%
    PRINT LEFT$(HYPHEN$S,10);
NEXT DROW%
FOR DCOL%=2 TO 2^NR.VARS% + 1
    VECTOR%=DCOL% - 2
    PRINT TAB(1);VECTOR%;
    TPOS%=10
    FOR DROW%=1 TO PRINT.LIMIT%
        PRINT TAB(TPOS%);LEFT$((DIAGRAMS(DROW%,DCOL%)),8);
        TPOS%=TPOS% + 10
    NEXT DROW%
NEXT DCOL%
PRINT
PRINT
PRINT
REM PRINT DIAGRAM FOR STATES 8-MAX
IF NR.STATES% < 8 THEN GOTO 35900
PRINT.LIMIT%=NR.STATES%
PRINT TAB(1);LEFT$(HYPHEN$S,6);"  ";
FOR DROW%=8 TO PRINT.LIMIT%
    PRINT LEFT$(HYPHEN$S,10);
NEXT DROW%
TPOS%=((PRINT.LIMIT% - 8) * 10) / 2
PRINT TAB(1);"INPUT";TAB(TPOS%);"N E X T   S T A T E"
TPOS%=10
PRINT "VECTOR";
FOR DROW%=8 TO PRINT.LIMIT%
    PRINT TAB(TPOS%);LEFT$((DIAGRAMS(DROW%,1)),8);
    TPOS%=TPOS% + 10
NEXT DROW%
PRINT TAB(1);LEFT$(HYPHEN$S,6);"  ";

```

```

FOR DROW%=8 TO PRINT.LIMIT%
  PRINT LEFT$(HYPHENSS,10);
NEXT DROW%
FOR DCOL%=2 TO 2^NR.VARS% + 1
  VECTOR%=DCOL% - 2
  PRINT TAB(1);VECTOR%;
  TPOS%=10
  FOR DROW%=8 TO PRINT.LIMIT%
    PRINT TAB(TPOS%);LEFT$((DIAGRAMS(DROW%,DCOL%)),8);
    TPOS%=TPOS% + 10
  NEXT DROW%
NEXT DCOL%
PRINT
PRINT
35900 INPUT "PRESS RETURN TO CONTINUE";LINE TEMPS

37000 REM*****OPTION TO SAVE DIAGRAM ON DISK*****
PRINT "DO YOU WANT TO SAVE THIS STATE DIAGRAM? (Y/N)"
INPUT TEMPS
IF TEMPS="N" THEN GOTO 39000
37100 PRINT "ENTER THE FILE NAME FOR THIS STATE DIAGRAM"
INPUT DIAGRAM.FILE.NAMES
IF SIZE(DIAGRAM.FILE.NAMES)=0 THEN GOTO 37500
PRINT "DO YOU WANT TO REPLACE THE CURRENT ";DIAGRAM.FILE.NAMES;
PRINT " FILE? (Y/N)"
INPUT TEMPS
IF TEMPS="N" THEN GOTO 37100
37500 CREATE DIAGRAM.FILE.NAMES AS 1
FOR DROW%=1 TO NR.STATES%
  FOR DCOL%=1 TO 2^NR.VARS%+1
    PRINT #1; DIAGRAMS(DROW%,DCOL%)
  NEXT DCOL%
NEXT DROW%
CLOSE 1

39000 REM*****WRAP-UP AND ERROR PROCESSING*****
STATUS.MSGS="***STATE DIAGRAM COMPLETED"
BUILD.STATE%=ON%
GOTO 39999
39050 STATUS.MSGS="***YOU MUST DEFINE THE VARIABLES FIRST"
39999 RETURN

```

```

40000 REM*****
REM*          40000 - IDENTIFY SUBROUTINES
REM*****
IF DEFINE.VARS% = OFF% THEN GOTO 49000
IF BUILD.STATE% = OFF% THEN GOTO 49000
PRINT SCREEN.CLEAR$
PRINT
PRINT
PRINT TAB(25); "IDENTIFY SUBROUTINES"
PRINT
PRINT "WHAT IS THE FILE NAME OF THE SUBROUTINE LIBRARY?"
40100 INPUT SUBROUTINE.LIB.NAMES
IF SIZE(SUBROUTINE.LIB.NAMES) = 0 THEN \
    PRINT "****NO FILE FOR "; SUBROUTINE.LIB.NAMES : \
    PRINT "RE-ENTER THE NAME OF THE SUBROUTINE LIBRARY" : \
    GOTO 40100

40200 REM*****BUILD LIST OF LIBRARY SUBROUTINE LABELS*****
OPEN SUBROUTINE.LIB.NAMES AS 1
SROW% = 1
40300 READ #1; LINE SUB.LINES
IF END #1 THEN 40900
IF LEFT$(SUB.LINES, 4) = SUB.HEAD$ THEN \
    TEMPS = MIDS(SUB.LINES, 5, 31) : \
    TEMP% = LEN(TEMPS) - 1 : \
    SUBIDS(SROW%) = LEFT$(TEMPS, TEMP%) : \
    SROW% = SROW% + 1
IF SROW% = MAX.SUB.LABELS% THEN \
    GOTO 49100 REM ABORT PROCESS
REM CONTINUE READING UNTIL END OF FILE
GOTO 40300
40900 NR.SUB.LABELS% = SROW%
CLOSE 1

41000 REM*****CORRELATE STATES WITH SUBROUTINES*****
PRINT
PRINT "ENTER THE STATEMENT NUMBER TO BE CALLED FOR EACH OF THE ";
PRINT "FOLLOWING STATES"
FOR IS1% = 1 TO NR.STATES%
    PRINT
    ISSTATES% = DIAGRAMS(IS1%, 1)
41200 PRINT ISSTATES%; " STATE SUBROUTINE"
    INPUT ISLABELS
    MATCH% = OFF% REM VALIDATE LABEL
    FOR SROW% = 1 TO NR.SUB.LABELS%
        IF SUBIDS(SROW%) = ISLABELS THEN \
            MATCH% = SROW%
    NEXT SROW%
    IF MATCH% = OFF% THEN \
        PRINT "****"; ISLABELS; " NOT ON "; SUBROUTINE.LIB.NAMES; : \
        PRINT " LIBRARY" : \
        GOTO 41200

```

```

    FOR IS2%=1 TO NR.STATES%
      FOR IS3%=1 TO 2^NR.VARS%+1
        IF DIAGRAMS(IS2%,IS3%)<>ISSTATES THEN GOTO 41500
        LET JUMPS(IS2%,IS3%)=ISLABELS
41500      NEXT IS3%
      NEXT IS2%
    NEXT IS1%

42000  REM*****PRINT SUBROUTINE DIAGRAM*****
    PRINT "DO YOU WANT TO PRINT THE SUBROUTINE STATE DIAGRAM? (Y/N)"
    INPUT TEMPS
    IF TEMPS="N" THEN GOTO 45000
    IF NR.STATES% > 7 THEN \
      PRINT.LIMIT%=7 \
    ELSE \
      PRINT.LIMIT%=NR.STATES%
    PRINT SCRAEN.CLEAR$
    REM PRINT DIAGRAM FOR STATES 1-7
    PRINT
    PRINT TAB(1);LEFTS(HYPHENSS,6);" ";
    FOR JROW%=1 TO PRINT.LIMIT%
      PRINT LEFTS(HYPHENSS,10);
    NEXT JROW%
    TPOS%=(PRINT.LIMIT% * 10) / 2
    PRINT TAB(1);" ";TAB(TPOS%);"N E X T   S T A T E"
    TPOS%=10
    PRINT "INPUT";
    FOR JROW%=1 TO PRINT.LIMIT%
      PRINT TAB(TPOS%);LEFTS((DIAGRAMS(JROW%,1)),8);
      TPOS%=TPOS% + 10
    NEXT JROW%
    TPOS%=10
    PRINT TAB(1);"VECTOR";
    FOR JROW%=1 TO PRINT.LIMIT%
      PRINT TAB(TPOS%);LEFTS((JUMPS(JROW%,1)),8);
      TPOS%=TPOS% + 10
    NEXT JROW%
    PRINT TAB(1);LEFTS(HYPHENSS,6);" ";
    FOR JROW%=1 TO PRINT.LIMIT%
      PRINT LEFTS(HYPHENSS,10);
    NEXT JROW%
    FOR JCOL%=2 TO 2^NR.VARS% + 1
      VECTOR%=JCOL% - 2
      PRINT TAB(1);VECTOR%;
      TPOS%=10
      FOR JROW%=1 TO PRINT.LIMIT%
        PRINT TAB(TPOS%);LEFTS((JUMPS(JROW%,JCOL%)),8);
        TPOS%=TPOS% + 10
      NEXT JROW%
    NEXT JCOL%
    PRINT
    PRINT
    PRINT
    REM PRINT DIAGRAM FOR STATES 8-MAX
    IF NR.STATES% < 8 THEN GOTO 42900

```

```

PRINT.LIMIT%=NR.STATES%
PRINT TAB(1);LEFT$(HYPHENSS,6);" ";
FOR JROW%=8 TO PRINT.LIMIT%
  PRINT LEFT$(HYPHENSS,10);
NEXT JROW%
TPOS%=((PRINT.LIMIT% - 8) * 10) / 2
PRINT TAB(1);" ";TAB(TPOS%);"N E X T   S T A T E"
TPOS%=10
PRINT "INPUT";
FOR JROW%=8 TO PRINT.LIMIT%
  PRINT TAB(TPOS%);LEFT$((DIAGRAMS(JROW%,1)),8);
  TPOS%=TPOS% + 10
NEXT JROW%
TPOS%=10
PRINT TAB(1);"VECTOR";
FOR JROW%=8 TO PRINT.LIMIT%
  PRINT TAB(TPOS%);LEFT$((JUMPS(JROW%,1)),8);
  TPOS%=TPOS% + 10
NEXT JROW%
PRINT TAB(1);LEFT$(HYPHENSS,6);" ";
FOR JROW%=8 TO PRINT.LIMIT%
  PRINT LEFT$(HYPHENSS,10);
NEXT JROW%
FOR JCOL%=2 TO 2^NR.VARS% + 1
  VECTOR%=JCOL% - 2
  PRINT TAB(1);VECTOR%;
  TPOS%=10
  FOR JROW%=8 TO PRINT.LIMIT%
    PRINT TAB(TPOS%);LEFT$((JUMPS(JROW%,JCOL%)),8);
    TPOS%=TPOS% + 10
  NEXT JROW%
NEXT JCOL%
PRINT
PRINT
42900 INPUT "PRESS RETURN TO CONTINUE";LINE TEMPS
45000 REM*****WRAP-UP AND ERROR PROCESSING*****
IDENTIFY.SUBS%=ON%
STATUS.MSGS="***SUBROUTINE IDENTIFICATION COMPLETED"
GOTO 49999
49000 STATUS.MSGS= \
  "****YOU MUST DEFINE THE VARIABLES AND BUILD THE DIAGRAM FIRST"
GOTO 49999
49100 STATUS.MSGS= \
  "****TOO MANY LABELS ON SOURCE LIBRARY; CANNOT CONTINUE PROCESSING"
49999 RETURN

```

```

50000  REM*****
      REM*          50000 - CREATE SOURCE PROGRAM          *
      REM*****

      IF DEFINE.VARS%=OFF% THEN GOTO 59100
      IF BUILD.STATE%=OFF% THEN GOTO 59100
      IF IDENTIFY.SUBS%=OFF% THEN GOTO 59100

50100  PRINT "WHAT IS THE NAME OF THE NEW SOURCE PROGRAM?"
      INPUT SOURCE.PROG.NAMES
      IF SIZE(SOURCE.PROG.NAMES)=0 THEN GOTO 51000
      PRINT "DO YOU WANT TO REPLACE THE CURRENT ";SOURCE.PROG.NAMES;"? (Y/N)"
      INPUT TEMPS
      IF TEMPS="Y" THEN GOTO 51000
      GOTO 50100

51000  REM GET SKELETON PROGRAM DRIVER FILE AND CREATE SOURCE PROGRAM FILE
      IF SIZE("SDPGMAIN.LIB")=0 THEN GOTO 59200
      PRINT "PROCESSING...PLEASE DON'T INTERRUPT"
      OPEN "SDPGMAIN.LIB" AS 1
      CREATE SOURCE.PROG.NAMES AS 2

52000  REM COPY DRIVER TO SOURCE FILE UNTIL INSERT FLAG FOUND IN DRIVER
      READ #1; LINE SG.LINES
      IF END #1 THEN 53000
      IF LEFT$(SG.LINES,6) = "INSERT" THEN \
          SG.INSERT =VAL(MIDS(SG.LINES,7,1)) :\
          ON SG.INSERT GOSUB 54000, 55000, 56000, 57000 :\
          GOTO 52000 \
      ELSE \
          PRINT USING "&"; #2; SG.LINES :\
          GOTO 52000

53000  REM DRIVER ROUTINE COMPLETE; CONCATENATE IT WITH SUBROUTINES
      OPEN SUBROUTINE.LIB.NAMES AS 3
53100  READ #3; LINE SG.LINES
      IF END #3 THEN 53500
      REM INSERT LOGIC TO SCREEN FOR SPECIFIED SUBROUTINES
      PRINT USING "&"; #2; SG.LINES
      GOTO 53100

53500  REM SOURCE PROGRAM GENERATION COMPLETE; CLOSE FILES AND TELL USER
      PRINT USING "&"; #2; "END"
      CLOSE 1
      CLOSE 2
      CLOSE 3
      STATUS.MSGS="***SOURCE PROGRAM SAVED ON DISK AS " + SOURCE.PROG.NAMES
      GOTO 59999

54000  REM INSERT1 SUBROUTINE: PROGRAM HEADER DATA
      PRINT USING "&"; #2; "REM      *      "+SOURCE.PROG.NAMES
      RETURN

```

```

55000  REM INSERT 2 SUBROUTINE:  VARIABLE INITIALIZATION
      READ #1; LINE SG.LINES                      REM CURRENT.STATE% =
      FOR SG.ROW%=1 TO NR.STATES%
          IF DIAGRAMS(SG.ROW%,1)=INITIAL.PROG.STATES THEN \
              SG.LINES=SG.LINES+JUMPS(SG.ROW%,1) :\
              GOTO 55100
      NEXT SG.ROW%
55100  PRINT USING "&"; #2; SG.LINES
      READ #1; LINE SG.LINES                      REM NR.STATES% =
      SG.LINES = SG.LINES + STRS(NR.STATES%)
      PRINT USING "&"; #2; SG.LINES
      READ #1; LINE SG.LINES                      REM JUMP.TABLE.SIZE% =
      SG.LINES = SG.LINES + STRS(2^NR.VARS%+1)
      PRINT USING "&"; #2; SG.LINES
      RETURN

56000  REM INSERT3 SUBROUTINE: JUMP TABLE INSERTION
      PRINT USING "&"; #2; "REM STATE DIAGRAM TABLE"
      REM INSERT REST OF DIAGRAM HERE

      PRINT USING "&"; #2; "REM JUMP TABLE"
      REM INSERT JUMP TABLE PRINTOUT HERE

      REM INSERT DATA STATEMENTS WITH JUMP TABLE VALUES
      PRINT USING "&"; #2; "REM JUMP TABLE VALUES"
      FOR SG.ROW%=1 TO NR.STATES%
          SG.LINES="DATA "
          FOR SG.COL%=1 TO 2^NR.VARS%+1
              SG.LINES=SG.LINES+JUMPS(SG.ROW%,SG.COL%)+", "
              IF LEN(SG.LINES)>75 THEN \
                  SG.LINES=LEFTS(SG.LINES,(LEN(SG.LINES)-1)) :\
                  PRINT USING "&"; #2; SG.LINES : \
                  SG.LINES="DATA "
          NEXT SG.COL%
          SG.LINES=LEFTS(SG.LINES,(LEN(SG.LINES)-1)) REM STRIP LAST COMMA
          PRINT USING "&"; #2; SG.LINES
      NEXT SG.ROW%
      RETURN

57000  REM INSERT4 SUBROUTINE: GENERATE GOSUB DESTINATIONS
      FOR SG.ROW%=1 TO NR.STATES%
          SG.SUBS=JUMPS(SG.ROW%,1)
          SG.LINES="          IF CURRENT.STATE%="+SG.SUBS+ \
              " THEN GOSUB "+SG.SUBS
          PRINT USING "&"; #2; SG.LINES
      NEXT SG.ROW%
      RETURN

59000  REM *****ERROR PROCESSING ROUTINES*****
59100  STATUS.MSGS="***YOU MUST COMPLETE THE FIRST 3 STEPS FIRST"
      GOTO 59999
59200  STATUS.MSGS="***FILE SDPGMAIN.LIB IS MISSING; CANNOT CREATE PROGRAM"
      GOTO 59999
59999  RETURN

```

```

90000 REM*****
REM*          90000 - DISPLAY MASTER MENU INSTRUCTIONS          *
REM*****
PRINT
PRINT "DEFINE VARIABLES"
PRINT "-----"
PRINT TAB(8);"THIS FUNCTION WILL PROMPT YOU FOR THE NAME OF EACH ";
PRINT "STATE VARIABLE."
PRINT TAB(8);"YOU WILL ALSO BE ASKED TO SPECIFY THE MEANING OF ";
PRINT "EACH VALUE THE"
PRINT TAB(8);"VARIABLE CAN TAKE (0 AND 1).  EXAMPLE:  MASTER ";
PRINT "SWITCH, 0=OFF, 1=ON."
PRINT "BUILD STATE DIAGRAM"
PRINT "-----"
PRINT TAB(8);"THIS FUNCTION WILL PROMPT YOU FOR THE NAME OF EACH ";
PRINT "STATE IN YOUR"
PRINT TAB(8);"PROGRAM.  FOR EACH STATE YOU DEFINE, YOU WILL BE ";
PRINT "ASKED TO SPECIFY"
PRINT TAB(8); "WHAT ACTION IS TO BE TAKEN (I.E., THE NEXT STATE) ";
PRINT "FOR EACH COMBINATION"
PRINT TAB(8); "OF THE STATE VARIABLES"
PRINT "IDENTIFY SUBROUTINES"
PRINT "-----"
PRINT TAB(8);"THIS FUNCTION WILL PROMPT YOU FOR THE IDENTIFIER OF ";
PRINT "THE SUBROUTINE"
PRINT TAB(8);"ASSOCIATED WITH EACH STATE.  THESE ARE SUBROUTINES ";
PRINT "WHICH HAVE ALREADY"
PRINT TAB(8);"BEEN CODED AND PLACED ON A SUBROUTINE LIBRARY."
PRINT "CREATE SOURCE PROGRAM"
PRINT "-----"
PRINT TAB(8);"THIS FUNCTION USES YOUR SUBROUTINE LIBRARY AND THE ";
PRINT "TABLES PRODUCED BY"
PRINT TAB(8);"THE PREVIOUS FUNCTIONS TO GENERATE SOURCE CODE FOR ";
PRINT "YOUR PROGRAM."
PRINT
PRINT
INPUT "PRESS RETURN TO CONTINUE";LINE TEMP1$
RETURN

```

```

99000 REM*****
REM*          PROGRAM SHUTDOWN          *
REM*****
STOP

```

END

# APPENDIX D

## SKELETON PROGRAM DRIVER USED TO GENERATE PROGRAMS

```

REM *****
INSERT1 (USER PROGRAM NAME)
REM *****

REM      NEEDED FOR SIMULATION, REMOVE WHEN VECTOR INPUTS CAN BE READ
STATE.VECTOR%=0
INPUT "PLEASE HIT RETURN TO START PROGRAM SIMULATION"; LINE TEMPS
RANDOMIZE

REM      THESE VARIABLES ARE SET TO ACTUAL VALUES DURING PROGRAM GENERATION
INSERT2 (INITIAL VARIABLE VALUES)
CURRENT.STATE%=
NR.STATES%=
JUMP.TABLE.SIZE%=

REM      DATA FOR THE JUMP TABLE IS CREATED DURING SOURCE PROGRAM GENERATION
REM      AND THEN READ INTO THE TABLE AT THE START OF THE PROGRAM RUN
DIM      JUMP.TABLE%(NR.STATES%,JUMP.TABLE.SIZE%)
FOR PDX1%=1 TO NR.STATES%
    FOR PDX2%=1 TO JUMP.TABLE.SIZE%
        READ JUMP.TABLE%(PDX1%,PDX2%)
    NEXT PDX2%
NEXT PDX1%
INSERT3 (JUMP TABLE DATA)

REM      MAIN PROGRAM LOOP
REM      CHECK STATE VECTOR VALUE RANGE; STOP PROGRAM IF INVALID
100      IF STATE.VECTOR% < 0 OR STATE.VECTOR% > JUMP.TABLE.SIZE% - 2 THEN \
        PRINT "STATE VECTOR OUT OF RANGE" :\
        PRINT TAB(5);"VALUE: ";STATE.VECTOR%;" SET BY: ";CURRENT.STATE% :\
        GOTO 199 :\
        REM STOP PROGRAM RUN

REM      SEARCH JUMP TABLE TO DETERMINE NEXT STATE, BASED ON CURRENT STATE
REM      AND STATE VECTOR
FOR PDX1%=1 TO NR.STATES%
    IF JUMP.TABLE%(PDX1%,1) = CURRENT.STATE% THEN \
        CURRENT.STATE% = JUMP.TABLE%(PDX1%,STATE.VECTOR%+2) :\
        PDX1%=NR.STATES%
NEXT PDX1%

REM      CALL NEXT STATE
INSERT4 (SUBROUTINE CALLS FOR EACH STATE)

REM      ANALYZE STATE VECTOR (SIMULATED BY SUBROUTINES WHICH SET X%)
STATE.VECTOR%=X%

REM      RESUME MAIN PROGRAM LOOP
GOTO 100

```

REM  
REM  
199      END OF PROGRAM (WILL NEVER STOP UNLESS SUBROUTINE EXECUTES STOP  
         OR INVALID STATE VECTOR IS DETECTED)  
         STOP

# APPENDIX E

## PROGRAM CREATED BY STATE DIAGRAM PROGRAM GENERATOR

```

REM *****
REM *   EXAMPLE.BAS
REM *****

REM   NEEDED FOR SIMULATION, REMOVE WHEN VECTOR INPUTS CAN BE READ
STATE VECTOR%=0
INPUT "PLEASE HIT RETURN TO START PROGRAM SIMULATION"; LINE TEMPS
RANDOMIZE

REM   THESE VARIABLES ARE SET TO ACTUAL VALUES DURING PROGRAM GENERATION
CURRENT.STATE%=1000
NR.STATES%=4
JUMP.TABLE.SIZE%=65

REM   DATA FOR THE JUMP TABLE IS CREATED DURING SOURCE PROGRAM GENERATION
REM   AND THEN READ INTO THE TABLE AT THE START OF THE PROGRAM RUN
DIM JUMP.TABLE%(NR.STATES%,JUMP.TABLE.SIZE%)
FOR PDX1%=1 TO NR.STATES%
  FOR PDX2%=1 TO JUMP.TABLE.SIZE%
    READ JUMP.TABLE%(PDX1%,PDX2%)
  NEXT PDX2%
NEXT PDX1%

REM STATE DIAGRAM TABLE
REM JUMP TABLE
REM JUMP TABLE VALUES
DATA 1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
DATA 1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
DATA 1000,1000,1000,1000,1000,1000,2000,1000,2000,1000,2000,1000,2000,1000,2000
DATA 1000,2000,1000,2000,1000,2000,1000,2000,1000,2000,1000,2000,1000,2000,1000
DATA 2000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
DATA 2000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
DATA 1000,1000,1000,1000,1000,1000,2000,1000,2000,1000,3000,1000,3000,1000,3000
DATA 1000,2000,1000,2000,1000,3000,1000,3000,1000,2000,1000,2000,1000,2000,1000
DATA 3000,1000,3000,1000,2000,1000,2000,1000,3000,1000,3000,1000,3000,1000,1000
DATA 3000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
DATA 1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
DATA 1000,1000,1000,1000,1000,1000,2000,1000,2000,1000,3000,1000,4000,1000,4000
DATA 1000,2000,1000,2000,1000,3000,1000,4000,1000,2000,1000,2000,1000,2000,1000
DATA 3000,1000,4000,1000,2000,1000,2000,1000,3000,1000,3000,1000,4000,1000,1000
DATA 4000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
DATA 1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
DATA 1000,1000,1000,1000,1000,1000,2000,1000,2000,1000,2000,1000,2000,1000,2000
DATA 1000,2000,1000,2000,1000,3000,1000,4000,1000,2000,1000,2000,1000,2000,1000
DATA 2000,1000,2000,1000,2000,1000,2000,1000,2000,1000,2000,1000,2000,1000,1000

```

```

REM      MAIN PROGRAM LOOP
REM      CHECK STATE VECTOR VALUE RANGE; STOP PROGRAM IF INVALID
100      IF STATE.VECTOR% < 0 OR STATE.VECTOR% > JUMP.TABLE.SIZE% - 2 THEN \
          PRINT "STATE VECTOR OUT OF RANGE" :\
          PRINT TAB(5);"VALUE: ";STATE.VECTOR%;" SET BY: ";CURRENT.STATE% :\
          GOTO 199 :\
          REM STOP PROGRAM RUN

REM      SEARCH JUMP TABLE TO DETERMINE NEXT STATE, BASED ON CURRENT STATE
REM      AND STATE VECTOR
      FOR PDX1%=1 TO NR.STATES%
          IF JUMP.TABLE%(PDX1%,1) = CURRENT.STATE% THEN \
              CURRENT.STATE% = JUMP.TABLE%(PDX1%,STATE.VECTOR%+2) :\
              PDX1%=NR.STATES%
      NEXT PDX1%

REM      CALL NEXT STATE
      IF CURRENT.STATE%=1000 THEN GOSUB 1000
      IF CURRENT.STATE%=2000 THEN GOSUB 2000
      IF CURRENT.STATE%=3000 THEN GOSUB 3000
      IF CURRENT.STATE%=4000 THEN GOSUB 4000

REM      ANALYZE STATE VECTOR (SIMULATED BY SUBROUTINES WHICH SET X%)
      STATE.VECTOR%=X%

REM      RESUME MAIN PROGRAM LOOP
      GOTO 100

REM      END OF PROGRAM (WILL NEVER STOP UNLESS SUBROUTINE EXECUTES STOP
REM      OR INVALID STATE VECTOR IS DETECTED)
199      STOP
REM#1000
1000     REM*****
      REM*                TEST SUBROUTINE 10000
      REM*****
      X%=INT%(RND*100)
      IF X% < 0 OR X% > 63 THEN GOTO 1000
      PRINT "SUBROUTINE 1000, VECTOR =" ;X%
      RETURN

REM#2000
2000     REM*****
      REM*                TEST SUBROUTINE 20000
      REM*****
      X%=INT%(RND*100)
      IF X% < 0 OR X% > 63 THEN GOTO 2000
      PRINT "SUBROUTINE 2000, VECTOR =" ;X%
      RETURN

REM#3000
3000     REM*****
      REM*                TEST SUBROUTINE 3000
      REM*****
      X%=INT%(RND*100)
      IF X% < 0 OR X% > 63 THEN GOTO 3000

```

```
PRINT "SUBROUTINE 3000, VECTOR =" ; X%  
STOP  
RETURN
```

```
REM#4000  
4000
```

```
REM*****  
REM*          TEST SUBROUTINE 40000  
REM*****  
X%=INT%(RND*100)  
IF X% < 0 OR X% > 63 THEN GOTO 4000  
PRINT "SUBROUTINE 4000, VECTOR =" ; X%  
RETURN
```

```
END
```

## APPENDIX F

### LIST OF PUBLICATIONS AND TECHNICAL REPORTS

"Research Directions in Multi-Micros"; Wrightsville Beach, North Carolina, May 1981

"Software Methodology for Microprocessors"; IECON Proceedings, Palo Alto, California, October 1982; IEEE Southcon, Atlanta, Georgia, January 1983

### LIST OF SCIENTIFIC PERSONNEL

G. Victor Wintriss  
Jeannine Wolf  
Dr. Michael Andrews  
Nicolas Panos  
Andrew Ash

ND

ATE

LMED

-83

TIC