

To the Graduate Council:

I am submitting herewith a thesis written by Thomas M. Tomcik entitled "Computer Aided Design of Algorithmic State Controllers for Digital Systems." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Engineering, with a major in Electrical Engineering.

Donald W. Bouldin

Donald W. Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:

Robert E. Bodenheimer

R. C. Gonzales

Accepted for the Council:

C. W. Mink

The Graduate School

4

COMPUTER AIDED DESIGN OF ALGORITHMIC STATE
CONTROLLERS FOR DIGITAL SYSTEMS

A Thesis
Presented for the
Master of Engineering
Degree
The University of Tennessee, Knoxville

Thomas M. Tomcik

June 1984

ABSTRACT

An Algorithmic State Machine (ASM) is a dedicated sequential machine which may be preferred to perform high speed data manipulation in digital systems. The design of these machines is simplified when computer aided design software is available. This thesis presents a Pascal program named ASMDESIGN which derives the logic truth table of an ASM from the user's description given in flowchart form. A typical session with this program is presented along with practical examples. The ASMDESIGN program operates on the TI-990/12 minicomputer and various software development techniques are presented. Pascal development tools, such as the configuration processor (CONFIG), are discussed using the ASMDESIGN program as an example. A detailed software presentation is included and the Texas Instruments Pascal (TIP) source code is also included for future work on this project. Certain program limitations are presented along with suggestions for future modifications and enhancements to the software.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION AND PROBLEM DESCRIPTION	1
II. ASMDESIGN USER'S GUIDE	5
Preliminary Considerations	5
Example Machine	9
Variable Table Editor	13
State Table Editor	20
Program Output	31
III. SOFTWARE DEVELOPMENT	42
Data Structure	42
Program Structure	48
Configuration Processor	60
IV. PROGRAM DESCRIPTION	71
Initialization Procedures	71
File Procedures	74
Variable Assignment Procedures	87
State Assignment Procedures	89
External Procedures	95
V. SUMMARY AND CONCLUSIONS	99
Program Limitations	99
Future Modifications	100
LIST OF REFERENCES	102
APPENDIX	104
VITA	162

LIST OF FIGURES

FIGURE	PAGE
1. ASM Generalized Block Diagram	2
2. List of Error Messages	7
3. ASMDESIGN Main Menu	8
4. ASMDESIGN State Structures	10
5. Example Machine BCDCHECK	12
6. Variable Assignment Sub-Menu	15
7. Insert Variables Prompt Screen	16
8. Variable Table for BCDCHECK	19
9. ASM Charts Sub-Menu	22
10. Screen for BCDCHECK State Zero	23
11. Screen for a Hypothetical State	24
12. Modified BCDCHECK State Zero	30
13. File Listing of ASM.BCDCHECK	32
14. File Listing of ASM.BCDALT	35
15. File Listing of ASM.B\$DCHECK	40
16. File Listing of ASM.B\$DALT	41
17. ASM Model Data Structure	43
18. Secondary Data Structure for ASMDESIGN	47
19. Pseudocode for Accepting User Input	50
20. Variable Table Manipulation Coding	50
21. State Table Manipulation Coding	52
22. ASMDESIGN Procedure List	54
23. ASMDESIGN Main Control Loop	59
24. Process Configuration Command File	64

FIGURE	PAGE
25. Configuration Processor Listing	68
26. ASM Design Help File	76

CHAPTER I

INTRODUCTION AND PROBLEM DESCRIPTION

In digital systems design, the use of a microprocessor has become popular due to its programmability and size. The major disadvantage of this approach is that the system speed is limited by the complexity of the microprocessor. When high speed data manipulation is required, a dedicated machine may be preferred to perform certain tasks. The Algorithmic State Machine or ASM [1], which is the subject of this thesis, may complement a microprocessor by performing the high speed data calculations while the microprocessor handles the slower operations.

A generalized block diagram of an ASM is shown in Figure 1. The Control Inputs may be generated by an external device or internally via the Data Manipulation Circuitry and/or microprocessor. The Control Outputs generally drive the Data Manipulation Circuitry as shown, but they may also be used to signal the microprocessor to accept processed data results. The Memory is composed of flip-flops which store the present state information of the ASM while the combinational logic contains the algorithm itself. The algorithm is usually designed in flowchart form and then it is converted into microcode and stored in a Read Only Memory (ROM).

When the number of states and Input/Output (I/O) variables become large, the conversion of the ASM flowchart into a microcode representation becomes tedious and susceptible to human errors. A computer aided design program entitled ASMDESIGN is presented in this

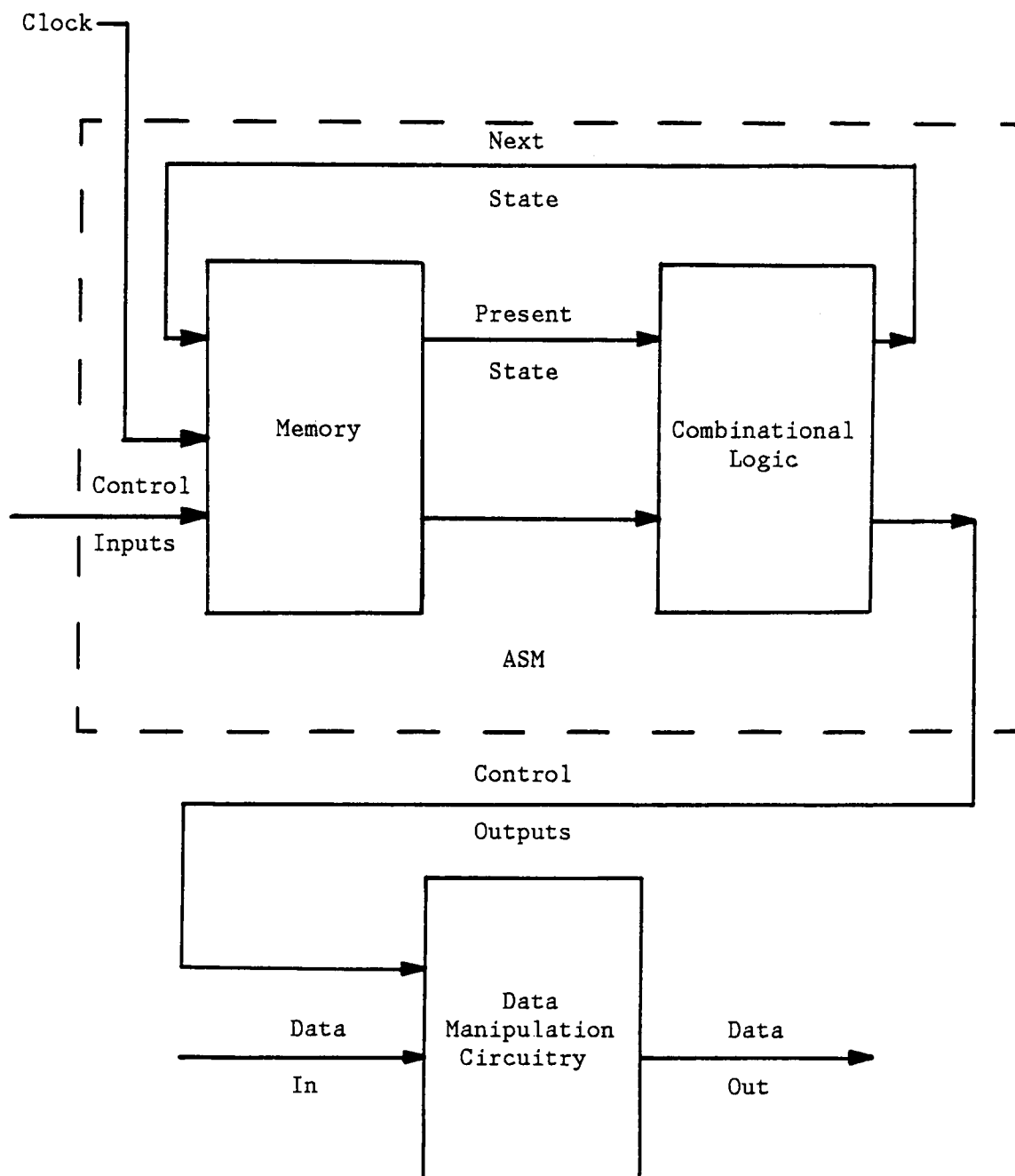


Figure 1. ASM Generalized Block Diagram.

thesis which derives the microcode in tabular form from the operator's description in flowchart form. The program operates on the Texas Instruments 990/12 computer and is written entirely in TI Pascal. The ASM modeled by the program may contain up to 32 I/O variables in 64 states.

A typical session with ASMDESIGN is discussed in Chapter II along with several preliminary considerations required to operate the computer. An example machine is presented as a vehicle upon which to demonstrate the variable and state table editing techniques available. Sample output is shown which includes a state by state description of the machine as well as its truth table output. This chapter may be considered as a general user's guide.

Software development on the TI-990/12 is presented in Chapter III. The program and data structures of ASMDESIGN are given and their versatility is discussed. Software development tools such as the Configuration Processor (CONFIG) and the Split Program utility (SPLITPGM) are also discussed in this chapter. The use of these tools helps to minimize program development time.

In Chapter IV a detailed program description of ASMDESIGN is presented. The procedures are classified into five general types. These categories are (1) initialization, (2) disk file, (3) variable table, (4) state table, and (5) external procedures. The external procedures may be sub-divided into the Video Display Terminal (VDT) procedures and the ASMDESIGN utility procedures. Since the VDT procedures are provided by Texas Instruments, and are explained in [2,3], they will not be discussed in detail.

Chapter V contains the summary and conclusions of this report. The possibility of an alternate data structure is discussed which would tend to reduce the complexity of the ASM model. The limitations of ASMDESIGN are also discussed as well as suggestions for future modifications and additions to this project. For subsequent work on this topic, a complete program listing is provided in the Appendix.

CHAPTER II

ASMDESIGN USER'S GUIDE

Preliminary Considerations

The ASMDESIGN program requires a user directory with at least 15 entries and the synonym 'ASM' must be assigned to this directory. A synonym is a shorthand representation of a quantity and may be thought of as a variable name in a programming language. The DX10 operating system maintains a table of synonyms for each user ID. For further information on the DX10 operating system refer to [4,5,6]. By convention, if the user's ID number is 'ABC.456' then the directory file name, or pathname, will be '.ABC.ABC456.ASM'. The synonym is assigned first, using the AS command, and then using the synonym 'ASM' as the directory name, the directory is created with the CFDIR command.

The initial run of ASMDESIGN constructs an index file composed of blank machine names and the current date. Up to six machines may be stored in the ASM directory and each machine is contained in two files. If the machine name is BCDCHECK, the file ASM.BCDCHECK contains a text listing of the machine and the file ASM.B\$DCHECK contains the truth table. The index file name is ASM.ASMINDEX. This name may not be used as a machine name. If the user attempts to use this name an error message will be displayed at the bottom of the screen. This file should not be deleted if there are machine files in the directory since program access to these files will be lost.

ASMDESIGN is programmed so that invalid user action will not 'crash' the system. If the user enters incorrect data either an error

message is displayed or the cursor will return to the start of the input field, or both may occur. When an error message is issued the user is prompted to enter a carriage return to continue by the message '<RETURN>' appearing in the lower right corner of the screen. A complete list of error messages is shown in Figure 2.

To run the program, enter the XPT command and answer the prompts as follows:

```
[ ]XPT
EXECUTE TI PASCAL TASK
      PROGRAM FILE: .RES RES548.ASM.PROGRAM
      TASK NAME OR ID: MAIN
      INPUT:
      OUTPUT:
      MESSAGES:
      MODE: FOREGROUND
      MEMORY: 2,3 .
```

The Main Menu is displayed as shown in Figure 3. All of the commands in ASMDESIGN are composed of one or two capitalized letters. The menu is divided into two sections. The I/O commands RD (Read from Disk), SD (Save to Disk) and DD (Delete from Disk) operate on the ASM directory, while the general commands VA (Variable Assignment), AC (ASM Charts) and MT (Machine Tables) are used to edit the machine.

When the I/O commands are chosen the contents of the index file are shown and a machine name is requested. If a machine name has been previously entered it will be displayed as a default value. The name entered is then concatenated with the value of the synonym 'ASM' to form a file name. Since the machine name will become part of the file name, several guidelines must be followed when choosing a machine name. These are:

NUMBER MESSAGE

1. ERROR: RESERVED FILE NAME !
2. ERROR: THE FIRST CHARACTER MUST BE A LETTER !
3. ERROR: ILLEGAL CHARACTER IN THE FILE NAME !
4. ERROR: VARIABLE TABLE EMPTY !
5. ERROR: MACHINE INDEX FULL !
6. ERROR: (Illegal Name) NOT FOUND !
7. ERROR: VARIABLE TABLE FULL !
8. ERROR: OPTION NOT AVAILABLE !
9. ERROR: THERE IS NO ASM IN MEMORY !
10. ERROR: THE NUMBER IS OUT OF RANGE !
11. ERROR: STATE TABLE FULL !
12. ERROR: VARIABLE NOT FOUND !
13. ERROR: STATE NOT INITIALIZED !

Figure 2. List of Error Messages.

ASM Design Program.

Main Menu.

I/O Commands:

RD <--> Read From Disk File.
SD <--> Save to Disk File.
DD <--> Delete From Disk File.

General Commands:

VA <--> Variable Assignment.
AC <--> ASM Charts.
MT <--> Machine Tables.
HF <--> Help Files.
QU <--> QUIT Program Execution.

Select an Option: VA

Figure 3. ASMDESIGN Main Menu.

1. The name length must not exceed eight characters.
2. The first character must be a letter, and the following characters may be either numbers or letters.
3. Lower case letters may be entered but they will be converted into uppercase letters.
4. The machine name must not be 'ASMINDEX'.

If these rules are not followed, an appropriate error message will be issued. Since the DD command is destructive, an extra key stroke is required. The program will ask 'Are You Sure (Y or N) ?' with a default value of N, which tends to prevent accidental erasure. These commands form a complete set of operations since the rename function may be implemented by (1) reading a machine, (2) saving it with a new name, and (3) deleting the old machine.

The general commands VA, AC and MT are the ASM editor commands and should be initially executed in this order. An example machine is presented in the next section so that these commands may be discussed in detail.

Example Machine

Algorithmic state machines are described in flowchart form and ASMDESIGN recognizes five basic state structures. These five structures are shown in Figure 4. The program incorporates the mixed-logic convention and each I/O variable may be active at either high or low levels. In a conditional branch, the active exit path is represented by a '1' and the inactive exit path is represented by a '0'.

All states in ASMDESIGN share three basic properties. They all

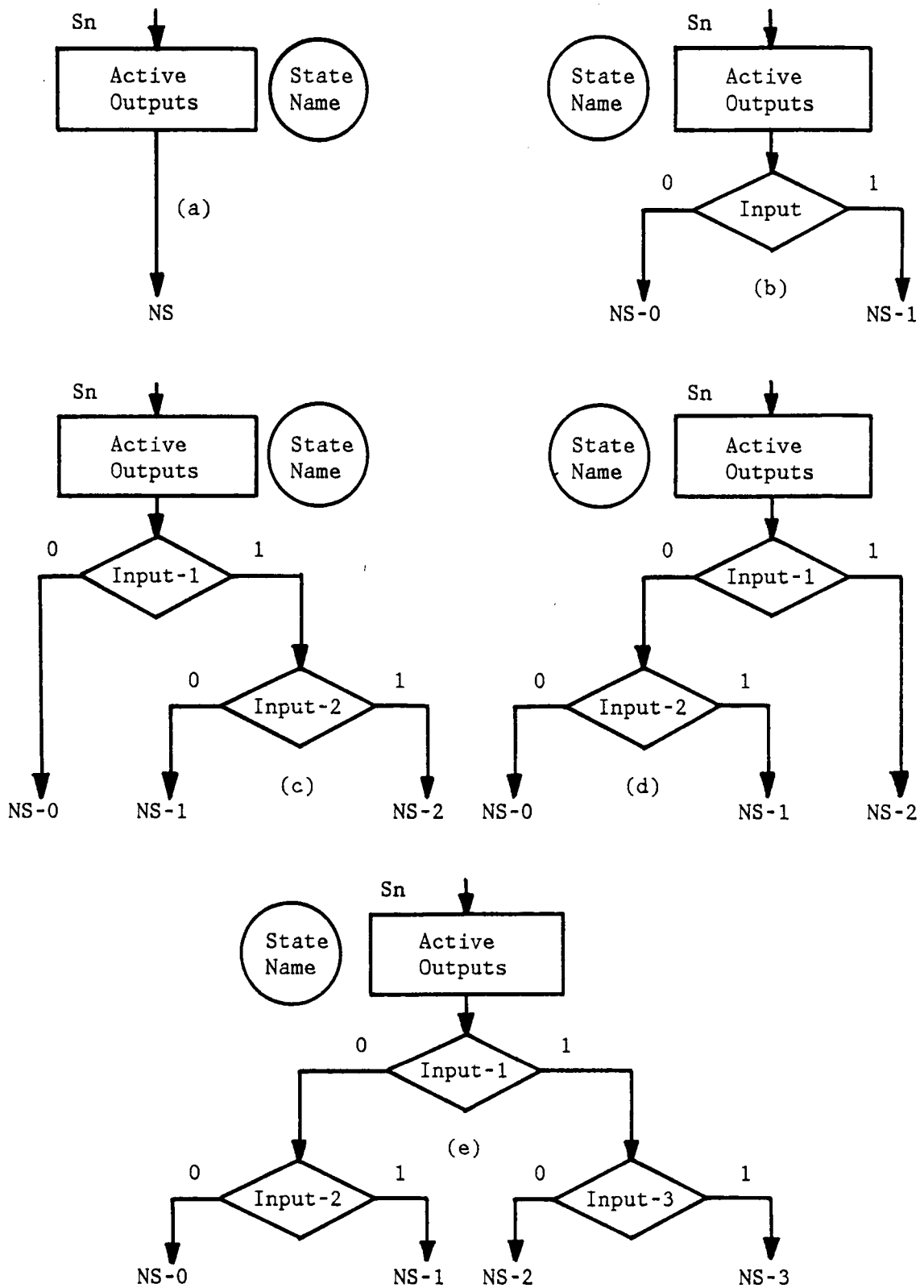


Figure 4. ASMDESIGN State Structures.

have (1) a unique state number, (2) a state name, and (3) an active output list. The state of the ASM is defined by the state of all flip-flops at a given time and the state number is the decimal equivalent of the actual binary levels. Although the state name is not necessary, it is a convenient way of describing the function of the state. The state name may be up to 20 characters in length. The active output list in the rectangular state box defines the ASM output status at a given time.

The basic difference in the five structures is the presence or absence of the diamond shaped decision box. The decision box contains a single input variable which determines the next state of the ASM. These decision boxes may be connected as shown to generate a wide variety of flowchart structures. For example, as shown in Figure 4c, if Input-1 is inactive (False) the next state is NS-0, but if Input-1 is active (True) then the next state is determined by the status of Input-2. Figure 4a, which shows the absence of a decision box, is an unconditional branch in which the next state is only a function of the present state.

In order to demonstrate the ASMDESIGN program, an example machine named BCDCHECK is presented in Figure 5. This two input/six output machine checks the validity of Binary Coded Decimal (BCD) numbers entered with the Most Significant Bit (MSB) first. When the START input becomes active (Low), the BCD number will be entered in the X-IN input on the following clock pulse. The external device or microprocessor monitors the BUSY output to determine when the machine is ready to accept data. Only when the BUSY output is inactive (High), the external

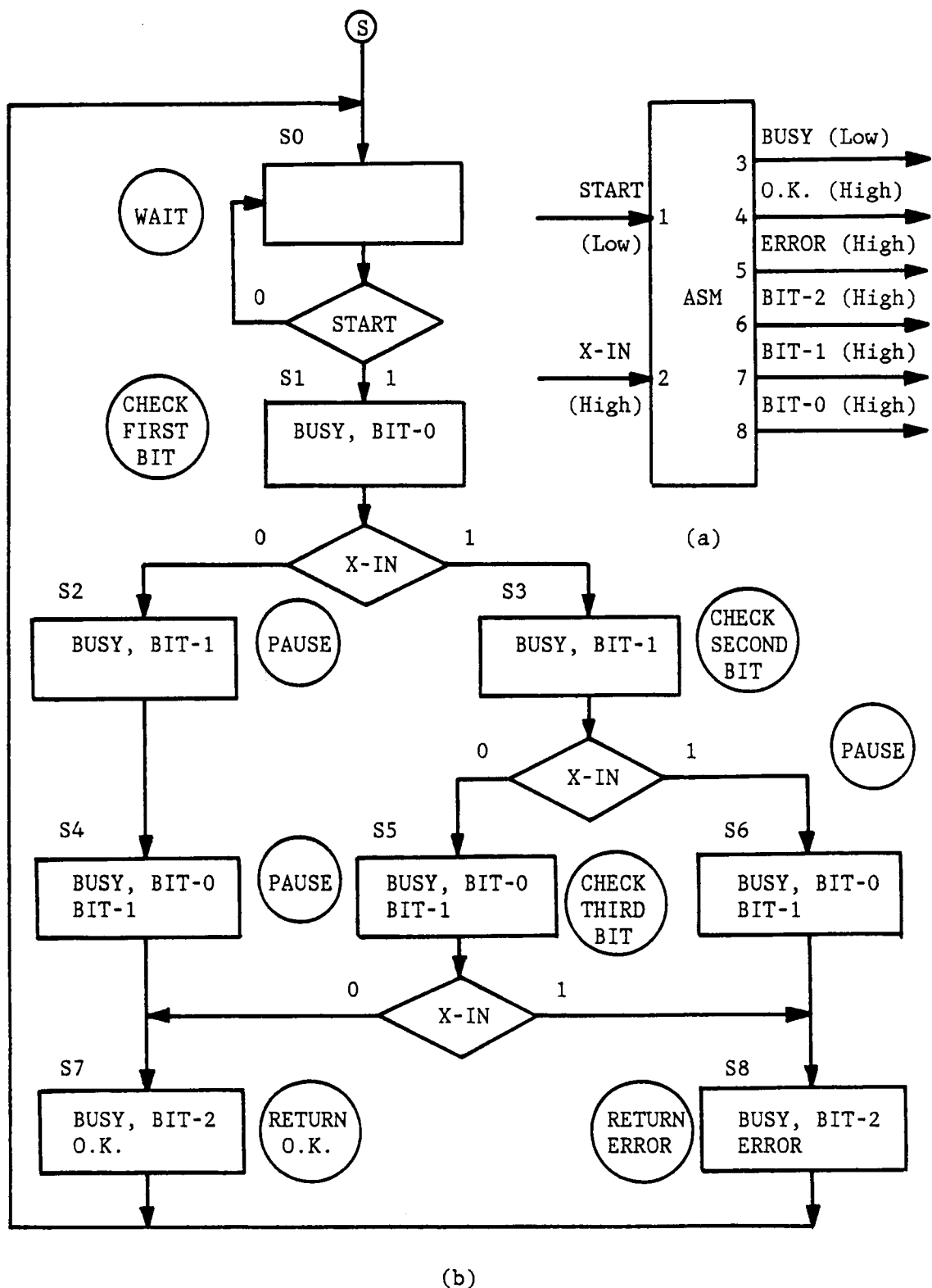


Figure 5. Example Machine BCDCHECK. (a) Block Diagram, (b) Flowchart.

device activates the START input and begins transmission. This half handshake scheme is sufficient since the result will appear on the O.K. or ERROR output in exactly four clock cycles.

The machine also acts as a counter with the outputs BIT-2, BIT-1 and BIT-0 representing the number of bits processed. In state five, for example, the value of the BIT-2, BIT-1, BIT-0 combination is 011 which represents three bits processed. The ASM contains nine states which requires four state variables and four flip-flops to uniquely describe the machine.

The following section describes the variable editing capabilities of ASMDESIGN using the BCDCHECK machine as an example. Once the variable table is constructed, the state table may be entered to completely define the machine.

Variable Table Editor

In the machine modeled by the ASMDESIGN program, there may be up to 32 I/O variables and each variable has four properties associated with it. The first property is a variable or I/O number which is assigned by the program. The numbers are assigned in consecutive order beginning with variable number one and ending with variable number 32. The program equates the user defined variable name to its number. The second property, the variable name, may contain up to eight characters. These characters may be uppercase letters, lowercase letters, numbers or keyboard symbols arranged in any order. When a variable name is requested by ASMDESIGN it must be entered exactly as it was defined. This does not present a problem since a list of valid variable names is

displayed whenever a name is asked for. The third property is the active level of the variable. The correct response to this property is either HIGH or LOW entered in capitalized letters. The fourth property is the variable use or I/O status. The use of the variable may be INPUT or OUTPUT in capitalized letters.

Entering the VA command from the Main Menu causes ASMDESIGN to display the screen shown in Figure 6. The RM (Return to Main Menu) command returns control back to the Main Menu which is the only place in ASMDESIGN in which the QU (Quit) command may be executed. This sub-menu may be considered as a lower node on a tree, and the RM command permits the user to return up the tree to the root node.

Initially, the IV (Insert Variables), VT (Variable Table) and MV (Modify Variables) commands should be executed in this order. If the VT or MV command is executed with an empty variable table or the IV command is requested with 32 variables assigned, an error message is displayed and the user must enter a new command.

Entering the IV command displays the screen shown in Figure 7. The 'Insert Variables' title appears at the top of the display to remind the user that he or she is in the IV mode. This is a useful feature since the 'Modify Variables' prompts are the same but the MV mode of operation is slightly different. These differences will be discussed in a subsequent paragraph.

Following the title line the four variable properties are displayed. The variable number cannot be accessed in this mode since ASMDESIGN automatically assigns these numbers, and the cursor is positioned at the first character of the 'Variable Name' field.

ASM Design Program.

Variable Assignment.

IV <--> Insert Variables.
VT <--> Variable Table.
MV <--> Modify Variables.
HF <--> Help Files.
RM <--> Return to Main Menu.

Select an Option: IV

Figure 6. Variable Assignment Sub-Menu.

ASM Design Program.

Insert Variables.

Variable Number: 1

Variable Name:

Active Level (HIGH or LOW) : HIGH

Variable Use (INPUT or OUTPUT) : OUTPUT

Another Variable (Y or N) : Y

Figure 7. Insert Variables Prompt Screen.

Although all keyboard characters including spaces are allowed in a variable name a blank field is not acceptable. If the user enters a carriage return or an erase field key, no error message is issued but the cursor will not advance. In the BCDCHECK example, the variable START may be entered at this time. The cursor will move to the 'Active Level' field.

The correct set of responses for the active level property is given in the prompt, and a default value of HIGH is displayed. This value is chosen since a large majority of active high variables are usually present in sequential machine design. In the ASMDESIGN program, whenever a limited set of responses is allowed, the correct alternatives are given. Since the active level of START is low, the correct value may be entered and the cursor will advance to the next field.

The choice of the default value of OUTPUT in the 'Variable Use' field is made for the same reasons as described before and INPUT may be entered in place of the default value. If the user wishes to leave the Insert Variables mode, a response of N to the 'Another Variable' prompt will branch ASMDESIGN back to the Variable Assignment menu. The value of Y remains the default until the variable table is full.

The other seven variables in the BCDCHECK example may be entered in the same manner as previously described. Although not necessary, it is convenient to keep the input and output variables together. The X-IN variable should be entered next.

The user may review the variable table at any time in ASMDESIGN by executing the VT command from the Variable Assignment menu. For the

BCDCHECK example a complete variable table is shown in Figure 8. The variable list is displayed showing the present value of the four variable properties as well as the total number of inputs and outputs in the list. The maximum number of variables that appear on one screen is 20. If the number of variables in the list is greater than 20, the bottom table line is not drawn and the prompt 'Press Return to Continue' appears. When the carriage return is entered, the table entries scroll up by 10 lines. The variable table has been completely displayed when the bottom line is drawn. The VT command is useful when modifying the variable list.

If the user detects an error while reviewing the variable table, it may be corrected by noting the variable number and executing the MV command from the Variable Assignment menu. The prompts are identical to those shown in Figure 7, but the title is now 'Modify Variables'. The only default values given in this mode are the 'Variable Number' and 'Another Variable' values. The variable number is one and the cursor is positioned on this default.

The user may now enter the number of the variable to be edited. In the BCDCHECK example, if the user incorrectly entered BUSY, then its number (3) will be entered. The present values of its properties appear as defaults and the user may make the correction by advancing the cursor to the error.

In the MV mode, a blank field is acceptable and it represents a variable deletion. By erasing the variable name, it is not included in the ASM and its variable number is left undefined. If at a later date

Variable Number	Variable Name	Active Level	Input/Output
1	START	LOW	INPUT
2	X-IN	HIGH	INPUT
3	BUSY	LOW	OUTPUT
4	O.K.	HIGH	OUTPUT
5	ERROR	HIGH	OUTPUT
6	BIT-2	HIGH	OUTPUT
7	BIT-1	HIGH	OUTPUT
8	BIT-0	HIGH	OUTPUT

Total Inputs: 2 Total Outputs: 6

<RETURN>

Figure 8. Variable Table for BCDCHECK.

a variable is to be added using the IV command, the lowest undefined number will be used first to help fill in the table.

Another useful feature of the MV command is the automatic incrementing of the variable number default value. The initial value is one and if the 'Another Variable' answer is Y, the variable number increments to two. The user advances through every property of every variable by continuously entering carriage returns. When the variable table has been completely scanned, the 'Another Variable' default value changes to N and the program branches to the Variable Assignment menu.

When the variable table is completed or the user must log off the computer, he or she may return to the Main Menu and save the variable list by executing the SD command. Any part of an incomplete machine may be saved for future additions or modifications. In the next section, the states of BCDCHECK are entered using the AC command from the Main Menu.

State Table Editor

The ASMDESIGN program may contain up to 64 states in a single machine. The BCDCHECK example has nine states and each state may be described by one of the state structures listed in Figure 4, page 10. The only restriction imposed by ASMDESIGN is that a variable table must be present to execute any of the state editing commands. If one is not present, a variable table may be generated either by the methods described in the previous section or by executing the RD command to read in an existing machine.

Entering the AC command from the Main Menu causes ASMDESIGN to

display the sub-menu shown in Figure 9. There are only two editing commands in this mode and they are AS (Assign States) and MS (Modify States). An error message will be issued if one of the following conditions is true:

1. The variable list is empty.
2. The state table is full and the AS command is entered.
3. The state table is empty and the MS command is entered.

The ASM editor is designed to give the user a graphic representation of each state. The limited graphics capability of the Texas Instruments Model 911 VDT is sufficient for representing diamond and rectangular shapes as well as lines and arrows. Prompts are displayed at the bottom of the screen while the cursor is positioned at key points in the state entry process, giving the user a feeling of sketching a state on paper. All prompts enclosed in ellipses indicate that the user response is entered away from the prompt line. Since the creation of states in the ASMDSIGN program is a dynamic process, only completed screens are shown in the next two figures. Figure 10 shows a complete screen for the BCDCHECK state zero assignment and Figure 11 shows a screen for a complex hypothetical state. The reader may wish to refer to these figures as the following dissertation continues.

By executing the AS command from the ASM Charts sub-menu the example machine BCDCHECK may be entered. The screen clears and the prompt 'Total Number of States:' is displayed. The correct range of values is between two and 64. If an incorrect value is entered an error message will be given. For this example, the correct value of nine is entered.

ASM Design Program.

ASM Charts.

AS <--> Assign States.

MS <--> Modify States.

HF <--> Help Files.

RM <--> Return to Main Menu.

Select an Option: AS

Figure 9. ASM Charts Sub-Menu.

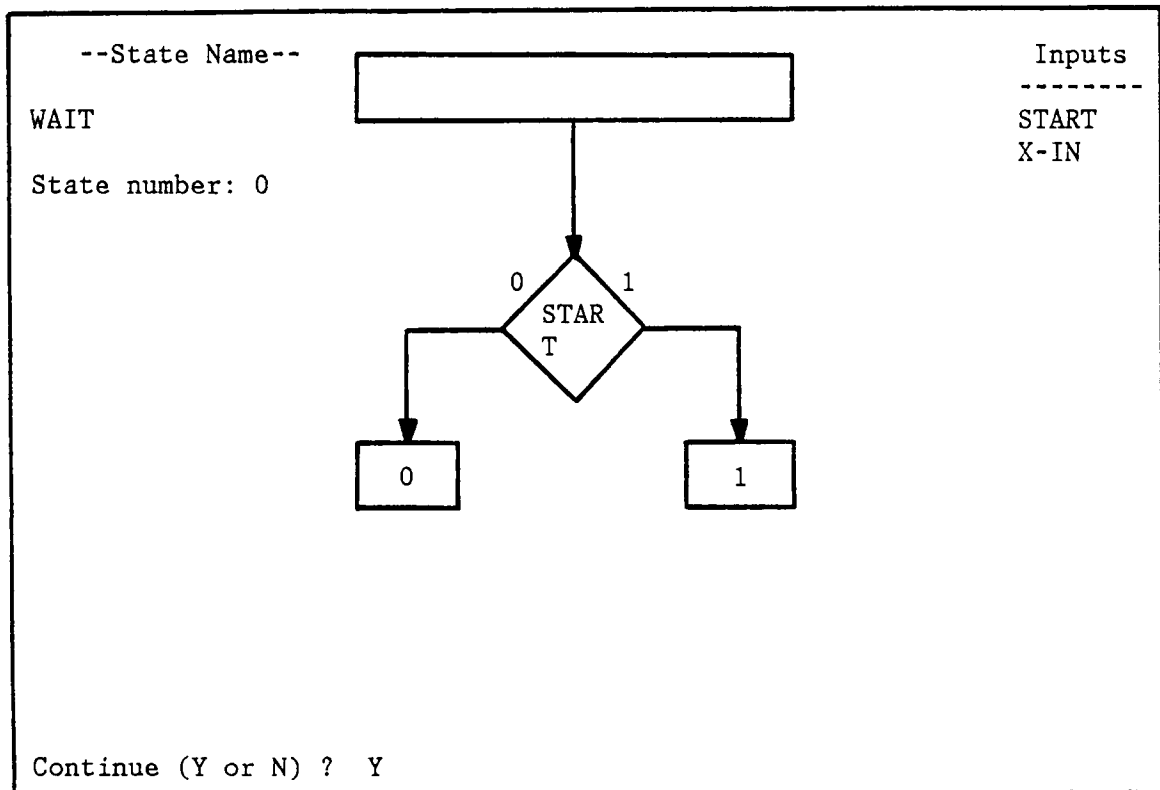


Figure 10. Screen for BCDCHECK State Zero.

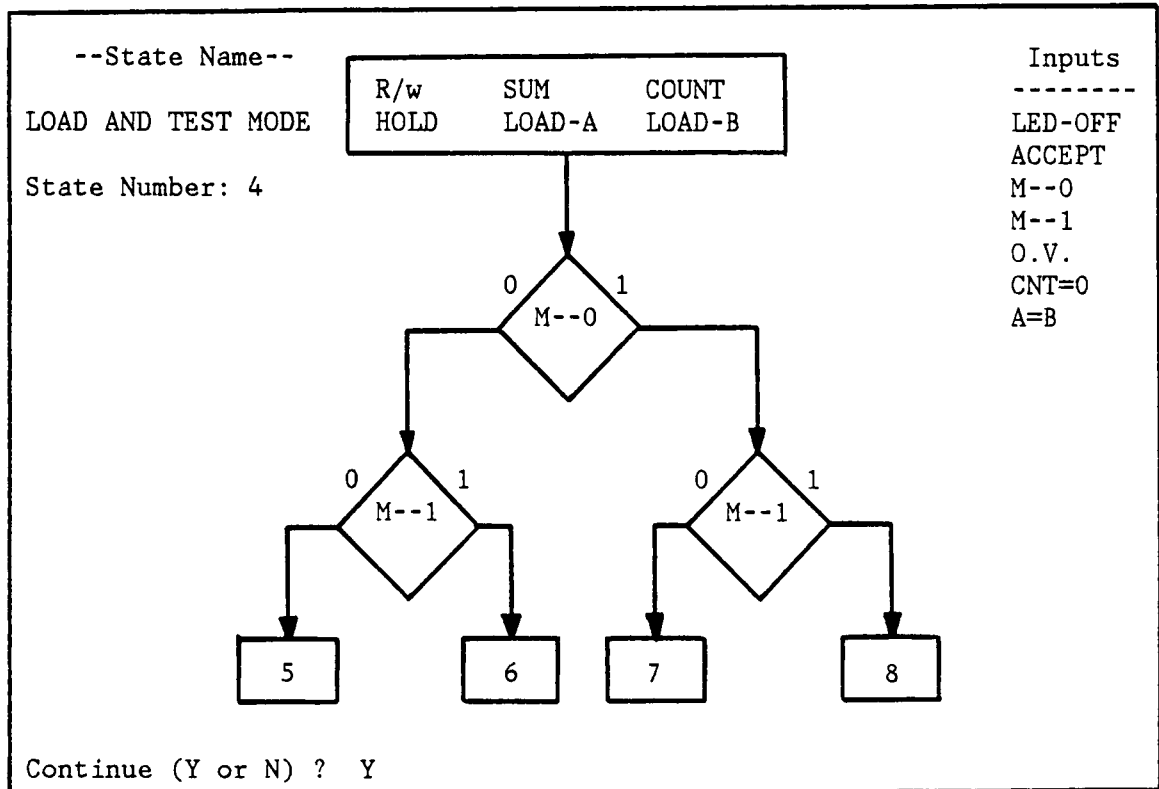


Figure 11. Screen for a Hypothetical State.

The screen clears and a state number of zero appears in the top left corner of the display. The state number is automatically incremented in the same manner as in the variable table editor. In this case the numbers will range from state zero through eight, inclusive. The prompt '...Enter State Name...' is displayed and the user types in 'WAIT' in the top left corner of the screen. The state name is not necessary, as previously stated, but the author recommends its use.

After the state name is entered, the prompt '...Enter Active Variables...' appears on the prompt line. The top of the state box is drawn and the cursor is positioned inside it to begin the output variable list. On the right side of the screen a list of output variable names which may be valid inside the state box is displayed. If the number of valid names is greater than 20 a second column will be displayed to the right of the first one. Up to three names may be entered per line. After the first variable name is entered the cursor advances to the right until the line is filled. If a fourth variable is to be entered, the state box is lengthened and the cursor is positioned below the first variable name. If the user attempts to enter the same variable twice it will not be accepted; the field is erased and the user must try again. Entering a blank field (carriage return) signals ASMDESIGN that all output variables have been entered. The bottom of the state box is drawn along with an arrow pointing down. In the BCDCHECK example, there are no active variables in state zero so a carriage return is all that is necessary.

If there are input variables present in the variable table, the next prompt that is displayed is 'Is There a Decision Block (Y or N) ?'

with a default value of N. It is assumed that no branches will occur until the user requests one. If the answer to this prompt is N a small state box is drawn below the arrow and the cursor is positioned inside it. The prompt '...Enter Next State...' appears and the user enters the next state number.

In the BCDCHECK example there is a decision box and the user should answer Y. A decision block is drawn and the output list on the right side of the screen is replaced by a similar input variable list. The prompt '...Enter Input Variable...' is displayed on the prompt line and the cursor is positioned inside the decision box. All decision type primitives in ASMDESIGN are drawn with the '0' (inactive) exit path on the left and the '1' (active) exit path on the right. Because of the small size of the decision block, the variable name is entered on two lines of four characters each. If the variable is four characters or less it may be typed in directly with two carriage returns. The variable START must be entered by typing STAR <carriage return> T <carriage return>.

After the input variable is entered the prompt '...Enter State or Decision (S or D)...' is displayed. The cursor is positioned at the inactive exit path of the decision block and when a response is entered, it moves to the active exit path for a response. If a response is D a decision block is drawn with its exit paths pointing to small state boxes and another input variable is requested as previously mentioned. If the response is S a small state box is drawn and the prompt '...Enter Next State...' is displayed. In the BCDCHECK example both exit paths

point to states (S response) and the next state numbers are entered as states zero and one from left to right.

After a state is completely defined the next prompt is 'Continue (Y or N) ?'. The default value is Y until the state table is full, which is similar to the variable table editor's IV mode. If the response to this prompt is Y the screen is erased and state number one is entered in the same manner. If the answer is N then program control is returned to the ASM Charts sub-menu.

If the user incorrectly enters the ASM it may be edited with the MS command. When this command is executed the screen clears and the prompt 'Total Number of States:' is displayed with a default value. The value of the default is the number entered in the AS mode. The user may modify states by entering the default, insert states by increasing the default number or delete states by decreasing this value. These three types of editing techniques will be presented in this order.

In the BCDCHECK example, if state zero is to be modified the default value of nine is entered. The next prompt that is displayed is 'Enter State Number (0 to 8):' with state zero as its default. The user may edit any state in any order but if an invalid state number is entered, an error message will be displayed. After the state number is entered, the state is modified exactly as it was defined with the AS command. The major difference in the MS mode is that all previous responses will appear as default values. The user may accept correct data by entering carriage returns to advance the cursor to the defect. If active output variables are present in a state box, they also will be displayed and the cursor will be positioned over the first variable

name. To delete active outputs the variable name may be erased by filling its field with spaces or entering the erase field key. To add new variables in the state box the user advances the cursor to the first empty field and enters the new variable name. Entering a blank field after the default values have been edited completes the list in the state box.

The default value in the 'Continue (Y or N) ?' prompt operates in the same manner as in the AS mode. When the state with the highest number has been edited the default value changes to N. The 'Enter State Number (0 to 8):' default value increments by one every time through the editor cycle and the user may step through every property of every state in the same manner as with the MV command in the variable table editor.

If the user wishes to insert a state the process requires two steps. First, the state is inserted by increasing the 'Total Number of States:' value. The state number of the unassigned state is the old maximum state number plus one. The new state number appears in the prompt if one state is added. By increasing the number of states in the BCDCHECK example to 10, the prompt 'Enter State Number (0 to 9):' indicates that the new state number is nine. If this state is requested in the MS mode ASMDESIGN operates as if it were in the AS mode.

The second step involved in state insertion is to connect the existing states to the new state. This may be accomplished by the methods previously explained. It is the user's responsibility to make these connections as required, since the machines modeled by ASMDESIGN may contain terminal states.

The BCDCHECK example may be modified by specifying that the

first bit of the BCD number is waiting on the X-IN input in the same clock cycle that the START input becomes active. To change the machine structure requires that state one must be eliminated and the output variable BIT-0 should be moved to the state zero output list.

The delete state mode of the ASM editor is implemented in the opposite manner from the insert state procedure. It is also a twofold process. First, all references to the state to be deleted (state one) made by the other states in the ASM should be eliminated. The only reference to this state is made by state zero. Also the geometry of the decision structure must be altered to incorporate two input tests. By executing the MS command and entering the default number of states, state zero is changed to the specifications shown in Figure 12.

The second step in the deletion process is to execute the MS command and specify a total of eight states. The prompt 'Enter State to be Deleted (0 to 8):' is displayed at the bottom of the screen. The correct response to this prompt is one and when the value is entered, control returns to the ASM Charts sub-menu indicating that state one has been deleted.

When a state is deleted the remaining states are renumbered to assure that all state numbers are assigned. In the BCDCHECK example, state two is reassigned state number one and all states with numbers greater than the deleted state are renumbered accordingly. The presence of a state name helps the user to identify the original states.

If the user does not eliminate all references to the state being deleted, ASMDESIGN informs the user of these unresolved references and requests that new state numbers be assigned before the old numbers are

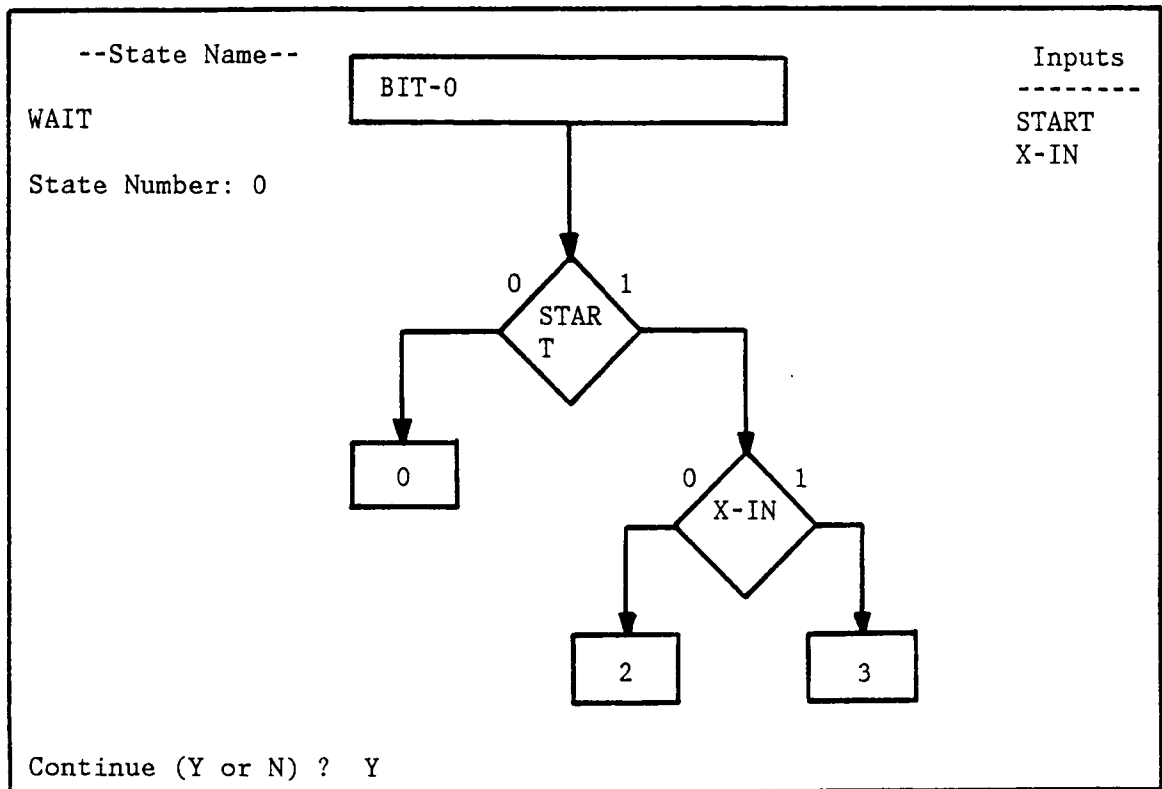


Figure 12. Modified BCDCHECK State Zero.

resequenced. If the geometry of the remaining states does not change, the first step in this process may be eliminated making the deletion process easier to implement.

This altered machine may be saved with the SD command from the Main Menu with the name BCDALT. In the next section, the program output for these two machines will be compared.

Program Output

The files generated by ASMDESIGN are text files which may be printed using the PF (Print File) command or displayed on the VDT using the SF (Show File) command from the DX10 operating system. These files must not be altered with the text editor. The program reads these files when the user enters the RD command from the Main Menu and if they are altered, the results may be unpredictable. There are two different file types generated by ASMDESIGN. The first type is created when the user saves an ASM with the SD command and the second type is generated with the MT command.

In the previous examples, two machines were saved in the ASM directory. Figure 13 shows the contents of the file ASM.BCDCHECK and Figure 14 lists the contents of ASM.BCDALT. These files begin with several global parameters of the machine such as the name, the number of states and the current date. The next section of the file contains the variable table which is similar to the VDT output generated by the VT command from the Variable Assignment menu.

The third part of these files contains a state by state description of the machine. Each state is enclosed in asterisks to help

Machine Name: BCDCHECK
 Number of States: 9
 Number of Inputs: 2
 Number of Outputs: 6

*****Variable Listing*****

Variable Number	Variable Name	Active Level	Input/Output
01	START	LOW	INPUT
02	X-IN	HIGH	INPUT
03	BUSY	LOW	OUTPUT
04	O.K.	HIGH	OUTPUT
05	ERROR	HIGH	OUTPUT
06	BIT-2	HIGH	OUTPUT
07	BIT-1	HIGH	OUTPUT
08	BIT-0	HIGH	OUTPUT

*****State Listing*****

State Number: 0
 State Name: WAIT
 Number of Outputs: 0
 Output Listing:
 Decision Type: SINGLE

START Next State:
 0 0
 1 1

State Number: 1
 State Name: CHECK FIRST BIT
 Number of Outputs: 2
 Output Listing: 3 8
 Decision Type: SINGLE

X-IN Next State:
 0 2
 1 3

State Number: 2
 State Name: PAUSE
 Number of Outputs: 2

Figure 13. File Listing of ASM.BCDCHECK.

Output Listing: 3 7
Decision Type: NONE

Next State: 4

State Number: 3
State Name: CHECK SECOND BIT
Number of Outputs: 2
Output Listing: 3 7
Decision Type: SINGLE

X-IN Next State:
0 5
1 6

State Number: 4
State Name: PAUSE
Number of Outputs: 3
Output Listing: 3 7 8
Decision Type: NONE

Next State: 7

State Number: 5
State Name: CHECK THIRD BIT
Number of Outputs: 3
Output Listing: 3 7 8
Decision Type: SINGLE

X-IN Next State:
0 7
1 8

State Number: 6
State Name: PAUSE
Number of Outputs: 3
Output Listing: 3 7 8
Decision Type: NONE

Next State: 8

Figure 13 (continued)

```

*****
State Number:      7
State Name:        RETURN O.K.
Number of Outputs:  3
Output Listing:    3  4  6
Decision Type:     NONE

Next State:        0
*****

*****
State Number:      8
State Name:        RETURN ERROR
Number of Outputs:  3
Output Listing:    3  5  6
Decision Type:     NONE

Next State:        0
*****

```

Figure 13 (continued)

Machine Name: BCDALT
 Number of States: 8
 Number of Inputs: 2
 Number of Outputs: 6

*****Variable Listing*****

Variable Number	Variable Name	Active Level	Input/Output
01	START	LOW	INPUT
02	X-IN	HIGH	INPUT
03	BUSY	LOW	OUTPUT
04	O.K.	HIGH	OUTPUT
05	ERROR	HIGH	OUTPUT
06	BIT-2	HIGH	OUTPUT
07	BIT-1	HIGH	OUTPUT
08	BIT-0	HIGH	OUTPUT

*****State Listing*****

State Number: 0
 State Name: WAIT
 Number of Outputs: 1
 Output Listing: 8
 Decision Type: DUAL SD

START	X-IN	Next State:
0	-	0
1	0	1
1	1	2

State Number: 1
 State Name: PAUSE
 Number of Outputs: 2
 Output Listing: 3 7
 Decision Type: NONE

Next State: 3

State Number: 2
 State Name: CHECK SECOND BIT
 Number of Outputs: 2
 Output Listing: 3 7

Figure 14. File Listing of ASM.BCDALT.

Decision Type: SINGLE

X-IN Next State:

0	4
1	5

State Number: 3
State Name: PAUSE
Number of Outputs: 3
Output Listing: 3 7 8
Decision Type: NONE

Next State: 6

State Number: 4
State Name: CHECK THIRD BIT
Number of Outputs: 3
Output Listing: 3 7 8
Decision Type: SINGLE

X-IN Next State:

0	6
1	7

State Number: 5
State Name: PAUSE
Number of Outputs: 3
Output Listing: 3 7 8
Decision Type: NONE

Next State: 7

State Number: 6
State Name: RETURN O.K.
Number of Outputs: 3
Output Listing: 3 4 6
Decision Type: NONE

Next State: 0

Figure 14 (continued)

```

*****
State Number:      7
State Name:        RETURN ERROR
Number of Outputs:  3
Output Listing:    3  5  6
Decision Type:     NONE

Next State:        0
*****

```

Figure 14 (continued)

the user in the analysis of the machine. The state name and number are given as well as the number of active outputs in the state box. The output listing is composed of a set of variable numbers which correspond to the I/O numbers assigned by ASMDESIGN and these numbers may be cross-referenced with the variable listing. The decision type corresponds to one of the five state structures shown in Figure 4, page 10. The value of the decision type is equated to the decision structure as follows:

1. 'NONE ' represents Figure 4a;
2. 'SINGLE ' represents Figure 4b;
3. 'DUAL SD' represents Figure 4c;
4. 'DUAL DS' represents Figure 4d;
5. 'TRIPLE ' represents Figure 4e.

The BCDALT example contains the modified state zero which is of type three.

Finally, each state contains a simple truth table giving the next state information. The input variable name(s) are given and the value of '0' corresponds to the inactive exit path, a '1' represents the active exit path and a '-' corresponds to a don't care situation. The notation is similar to the ASM editor interpretation.

The correct procedure for generating a truth table is as follows. The ASM must first be read into memory using the RD command from the Main Menu. If the user has been editing a machine it should be saved so that the two files will be compatible. In both cases a file I/O command must be executed from the Main Menu. Once the ASM is in memory, the MT command may be executed to generate the truth table.

The machine table generated using the BCDCHECK example is shown in Figure 15 and the table for the BCDALT machine is shown in Figure 16. The new file name is generated by replacing the second character of the machine name with a dollar sign. The ones and zeros entered in the table represent the actual variable levels with a '1' representing HIGH and a '0' corresponding to LOW levels. The truth table is transparent to the variable levels assigned in the variable table editor section of ASMDESIGN.

All I/O variables are referenced by their numbers and the state variables are labeled from S5 to S0 with S0 representing the least significant bit of the state number. Blank characters in the present and next state variables indicate that these variables are not necessary. For example, Figure 15 shows that four state variables are required to define BCDCHECK and three are needed to define the BCDALT machine.

The ASMDESIGN program explained in this chapter is easy to use once the operator enters the example machines. The error message capacity of the program helps the user to quickly and efficiently correct problems that are encountered. The following chapter describes the program and data structure of ASMDESIGN as well as some of the software tools available on the TI-990/12 computer.

Machine Name: BDCHECK
Number of States: 9

State Name	Inputs/PS		NS/Outputs	
	00	SSSSSS	SSSSSS	000000
	12	543210	543210	345678
WAIT	0-	0000	0001	100000
	1-	0000	0000	100000
CHECK FIRST BIT	-0	0001	0010	000001
	-1	0001	0011	000001
PAUSE	--	0010	0100	000010
CHECK SECOND BIT	-0	0011	0101	000010
	-1	0011	0110	000010
PAUSE	--	0100	0111	000011
CHECK THIRD BIT	-0	0101	0111	000011
	-1	0101	1000	000011
PAUSE	--	0110	1000	000011
RETURN O.K.	--	0111	0000	010100
RETURN ERROR	--	1000	0000	001100
Active Level	LH	HHHH	HHHH	LHHHHH

Figure 15. File Listing of ASM.B\$DCHECK.

Machine Name: BCDALT
Number of States: 8

	Inputs/PS		NS/Outputs	
	+-----+		+-----+	
	00 SSSSSS SSSSSS 000000			
State Name	12 543210 543210 345678			
	+-----+		+-----+	
WAIT	00	000	001	100001
	01	000	010	100001
	1-	000	000	100001
	+-----+		+-----+	
PAUSE	--	001	011	000010
	+-----+		+-----+	
CHECK SECOND BIT	-0	010	100	000010
	-1	010	101	000010
	+-----+		+-----+	
PAUSE	--	011	110	000011
	+-----+		+-----+	
CHECK THIRD BIT	-0	100	110	000011
	-1	100	111	000011
	+-----+		+-----+	
PAUSE	--	101	111	000011
	+-----+		+-----+	
RETURN O.K.	--	110	000	010100
	+-----+		+-----+	
RETURN ERROR	--	111	000	001100
	+-----+		+-----+	
	+-----+		+-----+	
Active Level	LH	HHH	HHH	LHHHHH
	+-----+		+-----+	

Figure 16. File Listing of ASM.B\$DALT.

CHAPTER III

SOFTWARE DEVELOPMENT

The ASMDESIGN program contains a model of a sequential machine and the application data structure is complex. This is the primary reason for choosing Pascal as the programming language. The availability of simple data types and the opportunity to define more complex structures make Pascal a good choice for this application. The promotion of structured programming in Pascal helps to facilitate the design of large programs since individual functions may be developed independently of each other. Also, the Pascal development tools available on the TI-990/12 computer save time in the design of large programs. These three aspects of software development will be considered in this chapter.

Data Structure

For clarity, the program data structure may be segregated into two parts. The first part contains the ASM model and the second part, miscellaneous structures which are important to the general operation of ASMDESIGN. A complete listing of the program is contained in the Appendix and it shows the integration of these two parts.

The structure of the ASM model is given in Figure 17. The four constants MAXNUMST, MAXNUMVAR, MAXLENVAR and MAXLENST are referred to throughout the program and they may be adjusted to control the machine size. This is a convenient development technique since the machine size depends upon the total ammount of memory available. The DX10

```

const MAXNUMST = 63; (* Maximum State Number. *)
      MAXNUMVAR = 32; (* Maximum Number of Variables. *)
      MAXLENVAR = 8;  (* Maximum Length of Variable Name. *)
      MAXLENST  = 20; (* Maximum Length of State Name. *)

type VARNUM    = 1..MAXNUMVAR;
   VARNUM0     = 0..MAXNUMVAR;
   VARNAME     = packed array [ 1..MAXLENVAR ] of char;
   VARLEV      = (HIGH,LOW);
   VARTYPE     = (INPV,OUTV);
   STATENUM    = 0..MAXNUMST;
   LONGWORD    = packed array [ VARNUM ] of 0..1;
   STATENAME    = packed array [ 1..MAXLENST ] of char;
   DECTYPE     = (NON,STASTA,STADDEC,DECSTA,DECDEC);
   SPTR        = @STATE;
   DPTR        = @DECISION;

   ONEVAR      = record
       VNAME    : VARNAME;
       VACTIVE  : VARLEV;
       VUSE     : VARTYPE
   end;

   ALLVAR      = record
       VNUMIN,VNUMOUT : VARNUM0;
       ALL           : array [ VARNUM ] of ONEVAR
   end;

   DECISION    = record
       INPVAR      : VARNUM0;
       case BRANCH : DECTYPE of
           NON      : (NEXT0 : SPTR);
           STASTA   : (NEXT1,NEXT2 : SPTR);
           STADDEC  : (NEXT3 : SPTR; NEXT4 : DPTR);
           DECSTA   : (NEXT5 : DPTR; NEXT6 : SPTR);
           DECDEC   : (NEXT7,NEXT8 : DPTR)
       end;

   STATE       = record
       NUMBER      : STATENUM;
       NAME        : STATENAME;
       VARLIST     : LONGWORD;
       DECS        : DECISION
   end;

var ASMVAR      : ALLVAR;
   NOVARS       : boolean;
   MXSTNUM      : STATENUM;
   STATESET     : array [ STATENUM ] of SPTR;

```

Figure 17. ASM Model Data Structure.

operating system allots 64K bytes to each operator and this memory must include (1) the program image, (2) stack space for static variables, (3) heap space for dynamic variables and (4) an extra 192 bytes of overhead. The data structure in Figure 17 may be divided again with the first part representing the variable table and the second part defining the state table. These divisions are presented in the following discussion.

The variable number types VARNUM and VARNUM0 are similar with the former representing the variable number and the latter defining the number of variables with a common property. The variable name type VARNAME is a string variable and its length is adjusted by the constant MAXLENVAR. The two enumeration types VARLEV and VARTYPE represent the active level and the I/O status of a single variable. These five simple types are sufficient to represent the four properties of each variable as well as the statistics of the complete variable list.

Three of the four variable properties are contained in the record ONEVAR and these are the variable name, the active level and the I/O status. The fourth property, the variable number, is contained in the record ALLVAR. This record includes the complete variable table in the array ALL with the index of this array representing the variable number. This record also contains the global parameters VNUMIN and VNUMOUT so that ASMDESIGN may have fast access to these important properties.

The variable table data structure is designed so that a single global variable, ASMVAR, contains all of the variable information for a specific machine. The boolean variable NOVARS provides the program with fast access to the presence or absence of an ASM in memory since a

machine must have a variable list regardless of the number of states it possesses.

The data structure of the state table contains the types STATENUM and STATENAME which are similar to the VARNUM and VARNAME types. The length of the state name is adjusted by the value of the constant MAXLENST. The type LONGWORD consists of a string of zeros and ones and the position of each bit corresponds to one of the I/O variable numbers. A variable of this type may be thought of as a set of flag bits and the bits that are set indicate that the corresponding variable is included in an active variable list. The pointers SPTR and DPTR point to state and decision locations respectively.

Each state in the model is represented by the record type STATE. This record contains (1) a state number, (2) a state name, (3) an active variable list, and (4) a decision record. The DECISION type is a variant record which may take the form of one of the structures shown in Figure 4, page 10. The type of branching desired is defined by the variable BRANCH which is a DECTYPE variable. A variable of this enumeration type may have one of five values and each value determines the combination of STATE and DECISION pointers needed. For example, if BRANCH is assigned the value STADEC, a decision structure is constructed with the inactive (zero) path pointing to a state and the active (one) exit path pointing to another decision. Each decision also contains the variable number INPVAR which is the input variable to be tested.

The global variable MXSTNUM contains the maximum state number of a machine. The value of this variable is initialized to zero and ASMDESIGN checks this variable to determine if there are any states in

the current machine. The STATESET array contains the state locations in memory and this gives the program fast access to the states. The index of the array may be considered as an internal state number while the actual state number is contained in the STATE record. This independence allows the program to sequentially manipulate the ASM while allowing the machine to remain as it was defined by the user.

The secondary data structure is shown in Figure 18 and it contains the general data types of the program. The three types SCISTRING, BYTE, and BTYPE are used by the external VDT procedures provided by Texas Instruments and must be included. The SCISTRING type is a string with the character element zero indicating the length. For example, if the variable X is an SCISTRING with a length of 10, then chr(10) is assigned to X[0]. The BYTE type contains a number sign in its definition which denotes that the following integer number is represented as a hexadecimal value. The BTYPE array contains a supervisor call (SVC) block in which the VDT I/O procedures may communicate with the operating system. For an extensive discussion of the SVC block data structure, refer to [6].

The four string types FILENAME, STRING2, STRING4 and STRING8 are simple strings and the STRING2 type is used in accepting user commands. The enumeration type CMD is composed of all of the two character commands displayed on the menus. The CSET type represents a set of valid CMD responses that the user may enter.

The variables BLOCK and ENDMES are required by the external VDT procedures and are explained in [3]. The text file MACHINX is the ASM index file and the file name PREFIX contains the long pathname of a disk

```

const MAXNUMACH = 6; (* Maximum Number of Machines on Disk. *)
      MAXLENFIL = 24; (* Maximum Length of File Name. *)
      BLANK     = '      '; (* Eight Spaces. *)

type SCISTRING = packed array [ 0..MAXLENFIL ] of char;
      FILENAME = packed array [ 1..MAXLENFIL ] of char;
      STRING2  = packed array [ 1..2 ] of char;
      STRING4  = packed array [ 1..4 ] of char;
      STRING8  = packed array [ 1..8 ] of char;
      BTYPE    = array [ 1..16 ] of integer;
      BYTE     = 0..#FF;
      CMD      = (SD,DD,RD,VA,AC,MT,HF,QU,NU,IV,MV,VT,AS,MS,RM);
      CSET     = set of CMD;

var  BLOCK      : BTYPE;
      ENDMES    : char;
      MACHINX   : text;
      PREFIX    : FILENAME;
      CHOICES   : CSET;
      CHOSEN    : CMD;
      DONE      : boolean;
      LEVTABLE  : array [ VARLEV ] of STRING8;
      TYPETABLE : array [ VARTYPE ] of STRING8;
      MACHNAME  : STRING8;

```

Figure 18. Secondary Data Structure for ASMDESIGN.

file without a machine name. The first 15 characters of this variable are assigned the value of the synonym 'ASM' as discussed in Chapter II; character 16 is a period and characters 17 through 24 are assigned spaces. The value of the variable MACHNAME is concatenated with the value of PREFIX to generate a complete file name for disk I/O.

The variables CHOICES, CHOSEN and DONE are used in the main control loop and will be discussed in the following section. Since enumeration types may not be entered directly from the terminal, the lookup tables LEVTABLE and TYPETABLE are implemented to decode user responses. A string is entered by the user and compared with the strings in the table. When a match is found the enumeration variable is assigned the value of the array index.

The next section of this chapter describes various Pascal program structures implemented in the ASMDESIGN program. Methods for manipulation of the data structure presented in this section are also discussed. Finally, the program structure of ASMDESIGN is presented on a procedural level and a procedure tree is derived which is based upon the nesting levels in the program.

Program Structure

The difference between interactive computer programs and batch programs is that the former tends to be event-driven. The program waits for the user to enter a command, performs the function requested and then waits for a new command. For this reason, the ergonomics of the man-machine interface should be considered and several guidelines must be followed in the design of interactive programs. The guidelines

chosen in the design of ASMDESIGN are:

1. The interactive flow maintains a steady rate and the dialogue is biased toward the user's language.
2. Visual feedback is provided to give the user confidence that a function has been properly executed.
3. The error handling functions are user friendly and the program must help the user to learn its operation.
4. Memorization is minimized by displaying correct response options to limited response entries.
5. The display is consistently structured to help minimize the user's learning time.

These design goals optimize user efficiency as opposed to program efficiency and the size of the program becomes an important factor. However if these rules can be followed, the design time for an ASM may be minimized.

The man--machine dialogue is implemented by the pseudocode structure shown in Figure 19. This loop is used when menus are displayed as well as when the user input requires some form of error checking. By consistently using this structure the five guidelines above are met.

The variable table data structure is accessed by decomposing the variable ASMVAR in the manner shown in Figure 20. The presence of a variable in the table is determined by comparing its name to the constant BLANK since all valid variables in the table must have a name. Variables are deleted from the table by erasing the name and new variables are added by looking for table entries with names equal to

```

repeat
  Display User Options;
  Wait to Accept User Response;
  if (User Response is Invalid) then Display Error Message
  else case -- Branch to the appropriate procedure or
            perform assignments as necessary;
until Done -- Correct User Response.

```

Figure 19. Pseudocode for Accepting User Input.

```

for I:=1 to MAXNUMVAR do
  with ASMVAR, ALL[I] do
    if VNAME <> BLANK then
      begin
        The Variable Name is VNAME;
        The Variable Number is I;
        The Active Level is VACTIVE;
        The I/O Status is VUSE;
        The Total Number of Inputs is VNUMIN;
        The Total Number of Outputs is VNUMOUT;
      end.
    end.
  end.
end.

```

Figure 20. Variable Table Manipulation Coding.

BLANK. The variable table is full when the sum of VNUMIN and VNUMOUT is equal to the value of the constant MAXNUMVAR. If a specific variable is to be analyzed, the 'for' statement may be eliminated and the 'with' statement is modified to accomodate the specific variable.

The state table is manipulated as shown in Figure 21 and the next state pointers refer to the structures in Figure 4, page 10. A complete breakdown is shown and the complexity is due to the variant part of the DECISION record. Although ASMDESIGN limits the length of the decision list to a depth of two, the data structure is capable of implementing longer lists. The value of the BRANCH variable is checked to determine if there is a decision variable in the state but the value of the INPVAR variable may also be checked. If the branch is unconditional, the value of the INPVAR variable is assigned zero which is an invalid variable number. The 'case' statement is used when addressing a state since it parallels the variant part of the DECISION record. When a state is to be deleted the order in which the DECISION and STATE records are disposed of is important since pointers may be lost in the process. The correct procedure is to dispose of the last DECISION records in the list before the state is returned to the heap, thereby conserving memory space.

Texas Instruments Pascal (TIP) has provisions for the development of library modules. These modules may contain procedures or functions and they are accessed by the main program with the EXTERNAL statement. The structure of a library module is the same as a TIP program without a program body. Since by definition, a Pascal program must contain a program body, the compiler must be directed not to

```

for I:=1 to MAXNUMST do
if (STATASET[I] <> nil) then
with STATASET[I]@, DECS@, ASMVAR do
begin
  The State Name is NAME;
  The State Number is NUMBER;
  for J:=1 to MAXNUMVAR do if (VARLIST[J] = 1) then ALL[J] is an Active
  Output Variable;
  if (BRANCH <> NON) then the First Decision Variable is ALL[INPVAR];
  case BRANCH of
    NON      : NEXT0@ is NS in Figure 4a;
    STASTA   : NEXT1@ is NS-0 in Figure 4b;
               NEXT2@ is NS-1 in Figure 4b;
    STADEC   : NEXT3@ is NS-0 in Figure 4c;
               NEXT4@.NEXT1@ is NS-1 in Figure 4c;
               NEXT4@.NEXT2@ is NS-2 in Figure 4c;
               ALL[NEXT4@.INPVAR] is Input-2 in Figure 4c;
    DECSTA   : NEXT5@.NEXT1@ is NS-0 in Figure 4d;
               NEXT5@.NEXT2@ is NS-1 in Figure 4d;
               NEXT6@ is NS-2 in Figure 4d;
               ALL[NEXT5@.INPVAR] is Input-2 in Figure 4d;
    DECDEC   : NEXT7@.NEXT1@ is NS-0 in Figure 4e;
               NEXT7@.NEXT2@ is NS-1 in Figure 4e;
               NEXT8@.NEXT1@ is NS-2 in Figure 4e;
               NEXT8@.NEXT2@ is NS-3 in Figure 4e;
               ALL[NEXT7@.INPVAR] is Input-2 in Figure 4e;
               ALL[NEXT8@.INPVAR] is Input-3 in Figure 4e;
  end case
end with.

```

Figure 21. State Table Manipulation Coding.

generate object code for it while generating code for the procedures and functions. This is accomplished by constructing a dummy program composed of the begin--end pair with the single compiler option (*\$NO OBJECT *) included. The NO OBJECT option is required since, without it, the presence of the 'begin' and 'end' keywords would produce an object module. If the object module were created, it would be necessary to delete it in order to properly link the main program with the library procedures.

Once a TIP program is compiled it consists of an object code file. If it contains external procedure declarations, the library object code must be concatenated with the main object code before the program image is built. This is accomplished with the link editor by using an INCLUDE command in the link control file. The link control file for ASMDESIGN is presented in the following section.

When a TIP program has been properly linked it may be executed and tested for run time or logical errors in the various algorithms. The advantage of using library modules in program development is that the complete program does not always have to be recompiled if errors are detected. If there is a fault in the main program it may be corrected, recompiled and relinked to the old object code for the library procedures. This saves development time since the compilation operation is slow and the time required depends upon the size and complexity of the source code.

A listing of the procedures used in the ASMDESIGN system of programs is shown in Figure 22. The complete program is composed of (1) the main program with 23 procedures, (2) the dummy program ASM1 with

ASMDESIGN PROCEDURES

1.0.0	ASMDESIGN	Main Program.
1.1.0	INITPTR	Initializes STATE and DECISION pointers.
1.2.0	INITIALIZE	Initializes program variables and index file.
1.3.0	HELP	Displays help messages.
1.4.0	FILEIO	Manages the ASM directory file.
1.4.1	SHOWINDEX	Displays the contents of the index file.
1.4.2	UPDATEIX	Updates the contents of the index file.
1.4.3	GETFNAME	Accepts and checks syntax of machine names.
1.4.4	SAVEASM	Writes ASM machine files.
1.4.5	READASM	Reads ASM machine files.
1.4.6	DELASM	Deletes ASM machine files.
1.5.0	VARASIGN	Manages the variable table editor.
1.5.1	VARTABLE	Displays the variable table on the VDT.
1.5.2	MODVARS	Inserts, modifies or deletes variables.
1.6.0	ASMCHART	Manages the state table editor.
1.6.1	DRAWVARS	Displays the I/O variable names.
1.6.2	GETSTR	Draws decisions and accepts input variables.
1.6.3	GETSPTR	Accepts next state numbers.
1.6.4	INVARS	Constructs decision structures.
1.6.5	OTVARS	Accepts active output variable names.
1.6.6	DELSTATE	Deletes states from the ASM.
1.6.7	MODSTATES	Inserts or modifies states in the state table.
1.7.0	ASMTABLE	Writes the machine truth table on disk.
1.7.1	BINARY	Converts decimal integers into binary strings.

EXTERNAL PROCEDURES

2.0.0	ASM1	Utility procedures.
2.1.0	WAIT	Displays <RETURN> and waits for a carriage return.
2.2.0	IDISPLAY	Displays integers on the VDT.
2.3.0	IACCEPT	Accepts integers from the VDT.
2.4.0	MENU	Displays program menus.
2.5.0	ERRORMSG	Displays error messages.
2.6.0	GETCMD	Accepts two character commands.
2.7.0	GRAPHICS	Draws states, decisions, lines and arrows.
3.0.0	VDTP	VDT Procedures.
3.1.0	ACCEPT	Accepts strings from the VDT.
3.2.0	CLEARSCREEN	Clears the VDT screen.
3.3.0	DISPLAY	Displays strings on the VDT.
3.4.0	INITSCREEN	Opens the VDT for I/O.
3.5.0	SCROLL	Scrolls the VDT screen.
3.6.0	DELFILE	Deletes files from the ASM directory.
-----	SET\$ACNM	Equates text file types to a pathname.
-----	FIND\$SYN	Returns the value of a specified synonym.
-----	SVC\$	Supervisor call procedure.
-----	STATIONID	Returns the VDT station number.
-----	OVL\$	Overlay Manager.

Figure 22. ASMDESIGN Procedure List.

seven external procedures, (3) the dummy program VDTP with six external procedures and (4) the five systems procedures written in assembly language. The procedures for each program are ordered in the standard Pascal method with the nesting levels illustrated as indentations. For example, the procedure VARASIGN is contained in the program ASMDESIGN and the procedures VARTABLE and MODVARS are contained in the procedure VARASIGN. The procedures are numbered for clarity using the format 'X.Y.Z', where X is the program number, Y is the procedure number inside the program and Z is the procedure number inside procedure Y. Each program and procedure also has a brief description of its operation.

The dummy program VDTP contains the external VDT procedures provided by Texas Instruments as well as two special purpose procedures required by ASMDESIGN. The TIP procedures include ACCEPT, CLEARSCREEN, DISPLAY and INITSCREEN and they are discussed in [3]. These procedures are written for the TI-990/10 computer and they are modified for the TI-990/12 requirements. The procedure DELFILE deletes files from the ASM directory and is not a VDT procedure. This procedure is included here so that it may have access to the SVC block data structure contained in this program. The SCROLL procedure scrolls the VDT screen up by reading and writing directly to the screen buffer in the 911 VDT.

The external procedures contained in the program ASM1 may be considered as utility procedures for ASMDESIGN. These procedures are used in the main program at all nesting levels. The WAIT procedure is required to hold various screens and it provides a necessary function for interactive programs. The two procedures IDISPLAY and IACCEPT represent the integer counterparts to the DISPLAY and ACCEPT procedures

contained in the VDTP program. The range of integer numbers allowed in these procedures is zero to 99 which is sufficient for this application. The MENU procedure displays the menus as well as the text screens for the variable table editor. The ERRORMSG procedure displays error messages and the GETCMD procedure accepts user commands, given a valid set of responses. The GRAPHICS procedure displays the special graphics primitives which include decisions, states, lines and arrows for the state table editor.

The ASMDESIGN main program contains the state and variable table editors as well as the disk I/O functions. The INITPTR procedure creates the STATE and DECISION records for the model. If the pointer type to be initialized is other than 'nil', it is not initialized. The INITIALIZE procedure operates in two modes. The first time that it is called, the lookup tables are defined and the index file is maintained. On subsequent calls to this procedure machines are cleared from memory and the variable table is erased. The HELP procedure reads a text file and displays it on the VDT. The contents of this file is discussed in Chapter IV of this report and may be printed by executing the PF command from the DX10 operating system. The file name to read or print is '.RES.RES548.ASM.HELP\$'.

The FILEIO procedure contains six procedures which implement the three file operations in ASMDESIGN. The procedure READASM is accessed by the RD command, SAVEASM by the SD command and DELASM by the DD command from the Main Menu. Procedure SHOWINDEX displays the index file contents on the VDT and procedure UPDATEIX manages the index file contents. The GETFNAME procedure accepts a machine name and creates a

file name using the global variable PREFIX. This procedure also checks the machine name for syntax errors.

Procedure VARASIGN contains the two procedures necessary to implement the variable table editor. The Variable Assignment menu is displayed and the user selects one of the options presented. If the VT command is chosen, procedure VARTABLE is called and the variable table is displayed on the screen. If the IV or MV command is chosen, the procedure MODVARS is called and the value of the command determines its mode of operation. The structure of VARASIGN is similar to the main program structure.

The procedure ASMCHART contains seven procedures which implement the state table editor. The structure of this procedure is the same as in the VARASIGN case. If the user chooses the AS or MS command, the MODSTATES procedure is invoked and the mode of operation is determined by the value of the command requested. The procedure DRAWVARS displays input or output type variable names on the right side of the VDT screen. The inclusion of this feature helps to minimize memorization since the valid alternatives are shown. The GETSTR and GETSPTR procedures accept input variable names and next state numbers respectively. The INVARS procedure is used to create or edit the decision structure of a state and the OTVARS procedure constructs the active variable list inside the state box. The DELSTATE procedure deletes states from the machine. It scans the state table to look for any unresolved references to the state being deleted. If any unresolved references are found, the user must make the corrections before the deletion process is complete.

The ASMTABLE procedure constructs the truth table from the

machine in memory and it contains one procedure. The BINARY procedure converts a state number into a binary string segment. Each table line is constructed and written into the machine table file in an iterative manner. The ASMTABLE procedure may be considered as the final result of ASMDESIGN.

The main control loop of ASMDESIGN is shown in Figure 23. This flowchart shows the repeat--case structure required by an event-driven program. Three variables are used in the main program and they are CHOICES, CHOSEN and DONE. The set CHOICES represents the valid user responses at this menu level and the variable CHOSEN is a member of this set. The boolean variable DONE is initialized to false and it becomes true when the user enters the 'QU' command. The FILEIO procedure may be considered as an extension of the 'case' statement and the procedures VARASIGN and ASMCHART have identical structures to the main control loop. The stepwise refinement process is evident from this flowchart and the hierarchy of the procedures may be represented as a tree. In Figure 22 the nesting levels of the procedures are represented as indentations and the tree structure is visualized with the root node (ASMDESIGN) having seven offspring. Conceptualizing a program in this manner is important since it indicates the dependency of the procedure bodies on the data descriptions. This will become more apparent in the final section of this chapter when the operation of the configuration processor is developed.

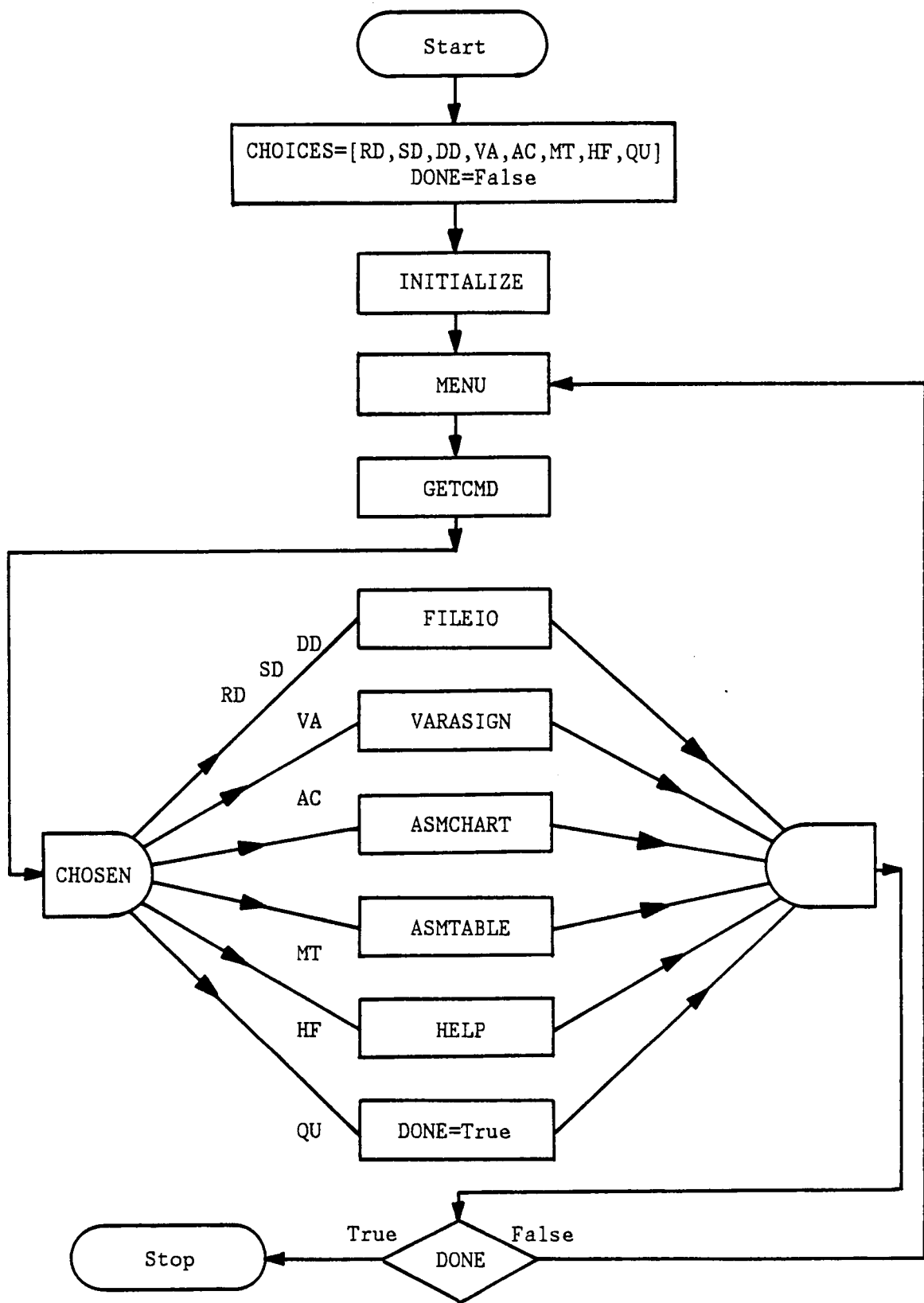


Figure 23. ASMDESIGN Main Control Loop.

Configuration Processor

In the preceeding section the use of library modules was presented to expedite program development time. This method saves time since the modules may be compiled separately. The disadvantage of this approach is that the procedures and functions in the library modules must exist in the same environment in which they will be executed. In a structured programming language such as TIP, separate compilation is more difficult due to the scope rules of Pascal and the capability of passing parameters by value or by reference. For example, if the procedure MODVARS is to be separately compiled, the declaration part of VARASIGN must be included as well as the declaration section of ASMDESIGN. Also, to generate a single object module for MODVARS, the program bodies of VARASIGN and ASMDESIGN must be replaced by dummy program segments with the NO OBJECT option specified to the compiler. Although this process could be accomplished manually with the text editor, it becomes tedious and prone to errors.

The configuration processor (CONFIG) supports the separate compilation of programs and procedures by performing the following functions:

1. It manages a library of source modules and combines declaration and body segments as required for separate compilation.
2. It builds a process configuration that contains the procedure tree as well as the locations of the source and object modules.
3. It prepares source programs for separate compilation by

using the process configuration along with user commands.

4. It maintains a library of object modules and joins them to be submitted to the link editor.

This TIP software is powerful but complex and a general description is given in the following paragraphs. For a more extensive description of this software, refer to [2,7].

The development of ASMDESIGN began in the traditional Pascal manner and when it became necessary to implement CONFIG, the source code had to be split into separate files. To prepare source code for CONFIG, certain formatting rules must be followed. The first rule is that every procedure must be forward declared so that each calling sequence is correctly defined. The configuration processor tends to alphabetize the procedures and the forward declaration allows the procedures to occur in any order. The second rule is that the reserved words 'begin' and 'end' in the compound statement of the program or its procedures start in the first column of the source files. All statements between the begin--end pair must be indented so that CONFIG can locate these modules. The third rule is that the compiler option NULLBODY should not be specified in any of the source modules. The fourth rule for formatting is that the first character position should not contain an asterisk unless it is inside a comment and this position should never contain a minus sign unless the second character position also has one.

The configuration processor recognizes a documentation section for each module where comments may be inserted. This section preceeds the 'const', 'type' or 'var' keywords. Comments in this section must begin in character position one and end in character position

72. The configuration processor may be instructed to print these sections without the program code.

The CONFIG utility requires four library synonyms and these should be assigned to directory pathnames. The directory represented by the synonym MASTER is intended for completed source modules and the synonym OBJLIB corresponds to the object modules for members in the MASTER library. The synonym LIBRARY represents the location of source modules under development and its corresponding object library is represented by the synonym ALTOBJ. Although LIBRARY is the only synonym required by CONFIG, it is advisable to assign the four synonyms before executing the program.

The Split Program utility (SPLITPGM) breaks the program into separate source modules composed of a single program, procedure or function. The user must insert markers in the source code preceeding the program or procedure name. The marker consists of the Pascal comment '(*&NAME *)' and the individual modules are written into the directory associated with the synonym LIBRARY. The member names in this directory are composed of the first six characters of NAME and they must be unique. The value of NAME should be the corresponding program or procedure name since CONFIG checks this condition by default. The SPLITPGM utility also writes an input command file for CONFIG which contains the commands required to build the process configuration corresponding to the original source program structure.

The SPLITPGM utility is executed as a Pascal task by entering the XPT command and answering the prompts as follows:

```

[]XPT
EXECUTE TI PASCAL TASK
    PROGRAM FILE: .TIP.PROGRAM
    TASK NAME OR ID: SPLITPGM
    INPUT: LIBRARY.PROG
    OUTPUT: LIBRARY.OUTPUT
    MESSAGES: ME
    MODE(F,B,D): F (Foreground)
    MEMORY: .

```

The response to the INPUT prompt is the file name of the source code to be split and the file specified by the OUTPUT response contains the process configuration commands.

The contents of the file LIBRARY.OUTPUT is shown in Figure 24. This file shows the process configuration commands for ASMDESIGN in its development phase. The operation of the configuration processor is controlled by the commands shown with each command beginning with an asterisk. The set of commands for CONFIG represents a very high level language and the command set is large. Many of the commands include a '<location>' which is defined as <DIR,MEM>, where DIR is a library synonym and MEM is a file in the library. The default library synonyms are LIBRARY and ALTOBJ and a location is required for all non-default specifications. The SPLITPGM utility generates a command file with default locations and the file in Figure 24 has been edited. The strategy in the development of ASMDESIGN is to put the procedures in the MASTER and OBJLIB libraries when they are completed.

The commands that define a process configuration are (1) *BUILD PROCESS, (2) *ADD, (3) *USE OBJECT, and (4) *CAT PROCESS. The BUILD PROCESS command initializes a configuration process as the current configuration process. It may contain a location parameter and if it is omitted, one must be present in the CAT PROCESS command. The location

```

*BUILD PROCESS
*ADD ASMDES <MASTER,ASMDES>
*ADD ASMDES:INITPT
*ADD ASMDES:INITIA <MASTER,INITIA>
*ADD ASMDES:HELP
*ADD ASMDES:FILEIO <MASTER,FILEIO>
*ADD FILEIO:SHOWIN <MASTER,SHOWIN>
*ADD FILEIO:UPDATE <MASTER,UPDATE>
*ADD FILEIO:GETFNA <MASTER,GETFNA>
*ADD FILEIO:SAVEAS
*ADD FILEIO:READAS
*ADD FILEIO:DELASM <MASTER,DELASM>
*ADD ASMDES:VARASI <MASTER,VARASI>
*ADD VARASI:VARTAB <MASTER,VARTAB>
*ADD VARASI:MODVAR <MASTER,MODVAR>
*ADD ASMDES:ASMCHA
*ADD ASMCHA:DRAWVA
*ADD ASMCHA:GETSTR
*ADD ASMCHA:GETSTR
*ADD ASMCHA:GETSPT
*ADD ASMCHA:INVAR
*ADD ASMCHA:OTVAR
*ADD ASMCHA:DELSTA
*ADD ASMCHA:MODSTA
*ADD ASMDES:ASMTAB
*ADD ASMTAB: BINARY
*USE OBJECT ASMDES <OBJLIB,ASMDES>
*USE OBJECT INITIA <OBJLIB,INITIA>
*USE OBJECT FILEIO <OBJLIB,FILEIO>
*USE OBJECT SHOWIN <OBJLIB,SHOWIN>
*USE OBJECT UPDATE <OBJLIB,UPDATE>
*USE OBJECT GETFNA <OBJLIB,GETFNA>
*USE OBJECT DELASM <OBJLIB,ELASM>
*USE OBJECT VARASI <OBJLIB,VARASI>
*USE OBJECT VARTAB <OBJLIB,VARTAB>
*USE OBJECT MODVAR <OBJLIB,MODVAR>
*CAT PROCESS <LIBRARY,PROCES>

```

Figure 24. Process Configuration Command File.

represents the file name of a data base that contains the procedure tree as well as the source and object file locations. The CAT PROCESS command creates this data base and it must be the last command in the file.

The ADD command is used to add nodes to the procedure tree and may have one of two forms. The first ADD command specifies the root node for a process configuration and it represents the main program. A location is optional and when it is not used, the node name is used as a member of the default library. The second form of the ADD command specifies the name and location of one or more nodes as sons of a specified node. For example, the command '*ADD ASMDES:INITPT' defines the node INITPT as a son of the root node ASMDES. The absence of the location parameter tells CONFIG that the procedure INITPTR is catalogued under the file name LIBRARY.INITPT. The USE OBJECT is not necessary to define a process configuration but it is included to instruct CONFIG that a non default object location is specified.

The configuration processor operates in either interactive or batch mode. To create the process configuration corresponding to the command file in Figure 24, the command XCONFIG is entered and the prompts are answered as follows:

```
[ ]XCONFIG
EXECUTE CONFIGURATION PROCESSOR<VESION:X.X.X YYDDD>
  COMMANDS:  LIBRARY.OUTPUT
  CRT FILE:  DUMY
  LISTING:   LIBRARY.LISTING
  MESSAGES:  ME
  MODE:      BATCH
  SOURCE:
  OBJECT:
  MEMORY:    4,8 .
```

The file LIBRARY.PROCES is created which contains the process data base and the file LIBRARY.LISTING contains a listing of the commands executed. Once the process file has been created, it does not have to be changed unless the process configuration is altered.

Once the data base for CONFIG is generated, separate program blocks may be compiled by executing the configuration processor in the interactive mode. The program generates a temporary source file to be submitted to the compiler and it also generates a deferred command file which is submitted back to CONFIG after compilation. For every node on the procedure tree the configuration processor maintains a set of flags. The flags are numbered zero to eight and the user may also define specialized flags. The flags relevant to the following discussion are the declaration (zero), body (one) and collect (seven) flags.

To execute the configuration processor interactively, the user enters the XCONFIGI command and the prompts are answered as follows:

```
[ ]XCONFIGI
EXECUTE CONFIG PROCESSOR INTERACTIVELY<VERSION:X.X.X YYDDD>
      LISTING:  LIBRARY.LISTING
      SOURCE:
      OBJECT:  ALTOBJ.FILES
      MEMORY:  4,8 .
```

The default source file is the temporary file .COMPFI_{nn} where _{nn} is the user's station number and the default object file is .OBJECT_{nn}. The file .COMPFI_{nn} contains the source code to be submitted to the compiler and the file ALTOBJ.FILES contains the compiled object code.

The first command entered is '*USE PROCESS <LIBRARY,PROCES>' which specifies that the current process configuration is for ASMDESIGN. The COMPILE command instructs CONFIG which modules are to be included in

.COMPFI_{nn}. If '*COMPILE ALL' is entered the complete program is included. This command sets the declaration and body flags for every node. By entering '*COMPILE DELASM' the declaration flags for the nodes ASMDES and FILEIO are set and the body flag is set for the DELASM node. The COMPILE command may also take the form '*COMPILE FILEIO ALL'. In this case, the procedure FILEIO and all of its sons are included in the compiler file. The final command entered is '*COLLECT ALL' which sets the collect flag for every node. The result of this command is that all procedure and program object modules are concatenated during the deferred run of CONFIG. When all commands have been specified they may be processed by pressing the <ENTER> key. This causes CONFIG to write the file .COMPFI_{nn}, the deferred command file .CPTEMP_{nn} and the listing file LIBRARY.LISTING. An example of the listing file is shown in Figure 25 which shows the process and flag listings.

After compilation, the file ALTOBJ.FILES contains the object code from the compiler and CONFIG is run in the batch mode. The input command file is .CPTEMP_{nn} and the object file is ALTOBJ.FILES. If the commands shown in Figure 25 are entered, the object file will contain the modules for FILEIO and all of its sons. The configuration processor splits ALTOBJ.FILES and stores the individual modules in the object libraries and then rewrites this file by including the object modules with the COLLECT flag set. After the deferred run of CONFIG is completed, the file ALTOBJ.FILES may be submitted to the link editor.

Overlay structures are supported by 'TI Pascal. This technique allows specific procedures to share the same memory block during program execution. The procedure OVLY\$ is called before the overlaid procedure

DXPSCLP 1.7.1 81.254 TI990 CONFIGURATION PROCESSOR 02/04/84 12:05:29
 *USE PROCESS <LIBRARY,PROCES>
 *COMPILE FILEIO ALL
 *COLLECT FILEIO ALL

PROCESS NAME	SOURCE LOCATION	OBJECT LOCATION	FLAGS SET
ASMDDESIG	<MASTER ,ASMDDES >	<OBJLIB ,ASMDDES >	0
ASMCHART	<LIBRARY ,ASMCHA >		
DELSTATE	<LIBRARY ,DELSTA >		
DRAWVARS	<LIBRARY ,DRAWVA >		
GETSPTR	<LIBRARY ,GETSPT >		
GETSTR	<LIBRARY ,GETSTR >		
INVARS	<LIBRARY ,INVARS >		
MODVARS	<LIBRARY ,MODVAR >		
OTVARS	<LIBRARY ,OTVARS >		
ASMTABLE	<LIBRARY ,ASMTAB >		
BINARY	<LIBRARY ,BINARY >		
FILEIO	<MASTER ,FILEIO >	<OBJLIB ,FILEIO >	0 1 7
DELASM	<MASTER ,DELASM >	<OBJLIB ,DELASM >	0 1 7
GETFNAME	<MASTER ,GETFNA >	<OBJLIB ,GETFNA >	0 1 7
READASM	<LIBRARY ,READAS >		0 1 7
SAVEASM	<LIBRARY ,SAVEAS >		0 1 7
SHOWINDE	<MASTER ,SHOWIN >	<OBJLIB ,SHOWIN >	0 1 7
UPDATEIX	<MASTER ,UPDATE >	<OBJLIB ,UPDATE >	0 1 7
HELP	<LIBRARY ,HELP >		
INITIALI	<MASTER ,INITIA >	<OBJLIB ,INITIA >	
INITPTR	<LIBRARY ,INITPT >		
VARASIGN	<MASTER ,VARASI >	<OBJLIB ,VARASI >	
MODVARS	<MASTER ,MODVAR >	<OBJLIB ,MODVAR >	
VARTABLE	<MASTER ,VARTAB >	<OBJLIB ,VARTAB >	

INPUT = ST05
 CRTFIL = ST05
 OUTPUT = .RES.RES548.CONFIG.SRC.LISTING
 COMPFIL = .COMPFIO5
 CPTEMP = .CPTEMPO5
 OBJECT = .RES.RES548.CONFIG.OBJ.FILES

MASTER = .RES.RES548.MASTER
 LIBRARY = .RES.RES548.CONFIG.SRC
 OBJLIB = .RES.RES548.OBJLIB
 ALTOBJ = .RES.RES548.CONFIG.OBJ

Figure 25. Configuration Processor Listing.

is called and a single parameter selects which module is to be loaded. The link control file handles the overlay mechanism and is discussed in [1,10]. The control file for ASMDESIGN is composed of the following statements:

```
NOSYMT;
FORMAT IMAGE,REPLACE;
TASK MAIN;
INCLUDE (MAIN);
INCLUDE OBJJLIB.DISPLAY; ( VDT Procedures. )
INCLUDE OBJJLIB.ASM1;    ( Utility procedures. )
INCLUDE ALTOBJ.ASM;      ( ASMDESIGN Procedures. )
PHASE 1,OVERLAY1;
INCLUDE ALTOBJ.FILES;    ( FILEIO Procedures. )
PHASE 1,OVERLAY2;
INCLUDE ALTOBJ.VARS;     ( VARASIGN Procedures. )
PHASE 1,OVERLAY3;
INCLUDE ALTOBJ.CHART;    ( ASMCHART Procedures. )
END .
```

The object code for each overlay module is constructed with the configuration processor and the external procedures are included in the main program section. If the link editor executes without errors or warnings, the program runs with the XPT command as explained in Chapter II.

When selecting modules for recompilation, several guidelines must be followed. When the declaration part of the program is changed the entire program must be recompiled. Also, when the declaration part of a procedure is changed, the module that contains the changes is recompiled along with the descendants of its node. If a statement within a module is changed, the module is recompiled. If these rules are not followed, unpredictable results may occur.

The general aspects of CONFIG have been developed in this chapter and its operation may seem involved. However, once a TIP

program has been set up for the configuration processor the power of the software becomes apparent. Since individual procedures may be recompiled, program development time is shortened. This freedom lets the software engineer design at the most primitive levels of the program as effectively as at the most general levels.

CHAPTER IV

PROGRAM DESCRIPTION

In Chapter III, the ASMDESIGN system of programs was presented at the procedure level and various program structures were described. The procedures may be grouped into five general categories which correspond to their functions in the program. These categories are (1) initialization, (2) disk file, (3) variable assignment, (4) state assignment, and (5) external procedures. This chapter presents a detailed description of the procedures in ASMDESIGN. The reader may wish to refer to the Appendix where a complete program listing is given.

Initialization Procedures

There are two initialization procedures in ASMDESIGN. They are INITPTR and INITIALIZE. Both are sons of the root program node and may be called by any non-external procedure. INITPTR creates state and/or decision records while INITIALIZE sets up the initial conditions for the ASM model.

The call to INITPTR has the form 'INITPTR(S,D,DEC,NUM)'. The variable parameters S and D are state and decision pointers respectively, DEC is a boolean expression and NUM is a state number. If DEC is true, the decision pointer D is initialized and the state pointer S is not affected. The variable BRANCH is assigned the value STASTA and the next state pointers NEXT1 and NEXT2 are initialized to 'nil'. This type of initialization is required when there is more than one input

variable in a state. The calling procedure connects this decision pointer to an existing state and it assigns the input variable number.

If the value of DEC is false, a state and decision are created only if the value of S is 'nil'. This extra check on S prevents the loss of partially completed states since the user may reference a next state number that does not currently exist in the model. For example, if the user enters state zero in the AS mode and one of the next states is 'nil', a new state is created and given the user's number. If the user references this state number again, it is not recreated since its state pointer is no longer 'nil'.

The state number is assigned the value of NUM, the active variable list is zeroed and the state name is assigned spaces. The decision pointer D is connected to the state S@ and it is initialized to an incomplete mode. The variable BRANCH is assigned the value NON and NEXT0 is assigned 'nil'. These two conditions are checked in the state assignment mode to determine if a state is partially initialized as previously explained. When the ASM editor senses this condition the user is allowed to complete the assignment process.

The call to INITIALIZE has the form 'INITIALIZE(FIRSTTIME)'. The boolean variable FIRSTTIME determines the type of initialization requested. The variable table ASMVAR is zeroed by assigning the constant BLANK to the variable names and the boolean variable NOVARS is assigned the value of FIRSTTIME. The global variable MXSTNUM is assigned zero which indicates the absence of states.

If the parameter FIRSTTIME is true, the system variables are initialized and the index file is maintained. The lookup tables

LEVTABLE and TYPETABLE are created and the variable MACHNAME is assigned spaces. This global variable is assigned in the GETFNAME procedure which is discussed in the following section. The VDT is opened for I/O by the call to INITSCREEN. This TI procedure has been modified for the TI-990/12 and will be discussed in the final section of this chapter.

Disk I/O is initialized by constructing the long pathname VAL and using its value to open the index file. The variable PREFIX is constructed by the call to FIND\$SYN. The synonym 'ASM' is assigned to the SCISTRING variable SYN and after the call to FIND\$SYN, the SCISTRING variable VAL contains the ASM directory location. The value of PREFIX is derived from VAL while VAL is modified to represent the index file by concatenating the string '.ASMINDEX' to it. Procedure SET\$ACNM equates a long pathname or access name to a Pascal text file. It must be called before opening a file for disk I/O. The file MACHINX represents the index file and since it is a global file, SET\$ACNM does not have to be called again for index file operations.

Two TIP routines IOTERM and STATUS allow the user to perform error checking without terminating the TIP task. The procedure IOTERM has three arguments. The first parameter is the file in question and the remaining two boolean arguments represent the old and new checking modes for the file. The second argument must be a variable and it is assigned by IOTERM. If the third parameter is true, automatic error checking is invoked which is the default condition for all files. By specifying this value as false, the designer may perform error checking in the software by monitoring the value of the function STATUS. The

index file is opened for reading by calling the 'reset' procedure and if the value of STATUS is zero an index file exists in the ASM directory.

If the value of STATUS is other than zero, the 'reset' procedure produces an error and an index file is created with blank machine names. The file is composed of long pathnames and the current date. The first record contains the index file name and date and the remaining records contain the value of PREFIX and the date. The global constant MAXNUMACH determines the number of records in the file.

If the value of FIRSTTIME is false, the states in memory are returned to the heap which conserves memory. The order in which the decision pointers are disposed of is important and this is accomplished by clearing the deepest decisions first. Once the decisions are disposed of the state may be cleared and assigned the value 'nil'.

File Procedures

The disk file procedures are contained in the outer procedures HELP, FILEIO and ASMTABLE. The HELP procedure reads a text file and displays it on the screen. The procedure FILEIO is called whenever the user enters the RD, SD or DD commands from the Main Menu. It is composed of a single 'case' statement and six procedures. The ASMTABLE procedure is called with the MT command and it contains one procedure. These three procedures and their descendants will be discussed in this section.

The HELP procedure reads a text file which is located in the file '.RES.RES548.ASM.HELP\$'. This file contains information on the operation of the ASMDESIGN program and it is constructed with the DX10

text editor. The file is accessed in the same manner as the index file, using the SET\$ACNM procedure. The length of the help file is arbitrary and the HELP procedure reads the file until an end of file (EOF) terminator is detected. Each page of the help file is displayed and when the VDT screen is filled, the WAIT procedure is called. This allows the user to examine the contents on a page by page basis. The pages are advanced by entering carriage returns until the complete file is displayed.

The contents of the help file may be printed on the line printer using the PF command from the DX10 operating system. The file printed is single spaced and the file displayed on the VDT is double spaced. The contents of the help file is given in Figure 26 where the VDT version is shown. Each two character command has a brief description of its operation and the maximum machine size is also presented. The flexibility of this help mechanism permits updating the help file without altering the source code in ASMDESIGN.

The procedures called by FILEIO are SAVEASM, READASM and DELASM. These procedures call SHOWINDEX, UPDATEIX and GETFNAME. The FILEIO procedure contains the FORWARD declarations for the six inner procedures as required by CONFIG. The external procedure DELFILE is also declared and is discussed in the final section of this chapter. The FILEIO procedure consists of a single 'case' statement and it may be considered as an extension to the 'case' structure in the main program loop. Although the procedures in FILEIO could be directly placed within ASMDESIGN, they are contained within FILEIO for separation purposes.

The call to SHOWINDEX has the form 'SHOWINDEX(EMPTY,CH)'. The

ASM Design Help File.

ASMDSIGN is a program that models a sequential machine. The user enters the machine in flowchart form and the program generates the truth table. The command list is shown below:

SD : Saves the current machine on a Disk file.

RD : Reads a machine from a Disk file.

DD : Deletes a machine from the ASM Directory.

VA : Displays the Variable Assignment menu.

AC : Displays the ASM Charts menu.

MT : Calculates the Machine truth Table.

HF : Displays this file on the VDT.

QU : QUits program execution.

<RETURN>

Figure 26. ASM Design Help File.

IV : Inserts Variables.
MV : Modifies Variables.
VT : Displays the Variable Table.
RM : Returns control to the Main Menu.
AS : Assigns States.
MS : Modifies States.

The maximum machine size is limited by the following parameters:

Number of States : 64
Number of Variables : 32
Variable Name Length : 8
State Name Length : 20
Machine Name Length : 8

<RETURN>

Figure 26 (continued)

boolean variable EMPTY is assigned true if the file does not contain any machines and the CMD type variable CH is used for displaying one of three titles. The value of CH is transferred from the user input entered at the Main Menu level. This extra parameter adds flexibility to the SHOWINDEX procedure since the calling procedure may be determined by the value of CH. The constants HEADER1 and HEADER2 are text strings with blank entries. These entries are filled in from the data in the index file. HEADER1 contains space for the index file access name and its creation date while HEADER2 contains space for machine names and their creation dates.

The 'case' statement displays a title line which is determined by the value of CH. This title information shows the user which command was requested from the Main Menu. This feature promotes the minimization of memorization goal of ASMDESIGN and it also informs the user that the correct command was properly executed. The variable EMPTY is assigned true and it is changed to false if a machine name is detected in the index file.

The variables FNAME and CDATE are read from the file and if FNAME does not equal PREFIX a valid machine name has been read. The constant HEADER2 is displayed and filled in with the (1) machine number, (2) machine name, and (3) date of creation. The machines are numbered to provide the user with an indication of the remaining space in the directory. The eight character machine name is assigned to the variable SNAME and is displayed along with the update date CDATE. The screen line variable LIN is incremented and the variable EMPTY is assigned

false. Once the index file is scanned the value of EMPTY is checked and if it is true, the message 'Index File Empty.' is displayed.

The call to UPDATEIX has the form 'UPDATEIX(F,DELETE,FULL)'. The file name F is to be updated in the index file and the boolean parameter DELETE determines the operation to be performed. The boolean variable FULL is returned by UPDATEIX and it may be considered as an error flag.

Since Pascal files may not be open for simultaneous read/write operations, the index file is updated in three steps. The first step reads the index file into the arrays NAMES and DATES. These arrays are updated by searching for a match between the value of F and the value of NAMES[I]. If a match is found the variable DELETE determines the type of modification necessary. If DELETE is true the machine name is erased otherwise the date is changed to the current date. If one of these conditions occur the variable FOUND is assigned true indicating a successful modification of the index file.

After the index file has been read into the arrays the value of FOUND is tested. If its value is false an attempt is made to enter the new machine name into the index file. This is accomplished by scanning the array NAMES while searching for an entry equal to PREFIX. If an entry is found the members of NAMES and DATES are replaced by F and TODAY, respectively. The variable FOUND is assigned true, indicating that a successful operation has been performed.

If the value of FOUND is true after these operations, a new index file is written which is composed of the modified arrays NAMES and DATES. A value of false for FOUND indicates that the index file is full

and the attempts have failed. An error message is displayed and the variable FULL is assigned true.

The boolean variable FULL serves a dual purpose for the calling procedures. If the calling procedure is DELASM, a true value for FULL indicates that the file name F is not present in the index file. If the calling procedure is SAVEASM and FULL is true, there is no room in the index file to save the current machine. This logic works since the READASM procedure performs its own error checking on the index file and it does not call UPDATEIX.

The call to GETFNAME has the form 'GETFNAME(F)', where the variable F is a file name constructed from the user entered machine name. This procedure also performs error checking on the entered machine name. It is composed of a 'repeat' loop and control is returned to the calling procedure when a valid file name has been constructed. The prompt 'Enter Machine Name:' is displayed with the default value contained in the global variable MACHNAME. The user's machine name is stored in the STRING8 type variable SNAME.

The processing of SNAME begins by converting lowercase letters into their uppercase equivalents. The boolean variable GOOD is then assigned the value of 'SNAME <> 'ASMINDEX''. If this expression is false, the error message 'ERROR: RESERVED FILENAME !' is displayed, processing is terminated and the user must enter a new machine name. By displaying the default machine inside the 'repeat' statement, invalid user input is erased. The processing of SNAME continues by assigning the value of 'SNAME[1] in LET' to GOOD. If this expression is false,

the error message 'ERROR: THE FIRST CHARACTER MUST BE A LETTER !' is displayed and the user must enter a new machine name.

The second through eighth characters are scanned using the boolean variables GOOD and FOUND. The variable FOUND becomes true when a space character is detected in the string. If the character is not a space, the variable GOOD is assigned the value of 'SNAME[I] in NUM+LET' since numbers and letters are valid in a file name. If a space has been detected (FOUND = true), the remaining characters must also be spaces since isolated spaces are not allowed in a file name. Once the string has been processed, the value of GOOD is tested to determine the validity of the machine name. If GOOD is false, the error message 'ERROR: ILLEGAL CHARACTER IN THE FILE NAME !' is displayed and the user must reenter a new machine name.

If the value of GOOD is true, the 'repeat' structure terminates and the file name F is constructed from PREFIX and SNAME. Although the dollar sign is a valid character in a TIP file name, it is excluded to avoid writing over the truth table files. It is also noted that this procedure was constructed using boolean algebra and the sequence in which the various tests are made is important to the correct operation of GETFNAME.

The SAVEASM procedure transfers the ASM in memory into a human readable text file. The converse procedure READASM reads a disk file constructed by SAVEASM and constructs the machine model. The similarity of these two procedures becomes evident by examining the source code. Since both procedures operate on global variables, their parameter lists are empty.

Procedure SAVEASM contains 18 string constants which are written into the file. The four constants ZER, ONE, DC and BL5 are used to represent the decision logic and the constants S1 through S14 contain text headings. The procedure begins with the calls to SHOWINDEX, GETFNAME and UPDATEIX. The value of the boolean variable IER is tested to determine if there is space available in the 'ASM' directory. If IER is false the file is opened by (1) constructing the string MNAME, (2) equating MNAME to the text file MACHFIL with SET\$ACNM, and (3) opening the file with a call to 'rewrite'.

The first section of the machine file contains the global attributes of the machine. These parameters are the machine name, the number of states, the number of input variables and the number of output variables. The global variable MACHNAME is constructed from the file name FNAME and the number of states is equal to the maximum state number plus one (MXSTNUM + 1). The number of input and output variables is contained in the variable ASMVAR.

The second section of the file saves the variable listing and it contains the four properties associated with each I/O variable. The format of the variable listing is identical to the variable listing displayed on the VDT. It is constructed by assigning the data to the string LINE and writing it into the file. All variables with names not equal to BLANK are entered into the table since, by convention, a valid variable must have a name.

The third section of the machine file contains a listing of the states currently assigned in the machine. The states are separated by a string of asterisks for clarity. The array STATESSET is scanned using a

'for' loop and when a valid state is found it is written into the file. The common state properties are written first, such as the state number and state name. The number of active output variables is determined by counting the number of ones in the VARLIST variable. The position of each entry in VARLIST corresponds to a variable number and if it is active, the variable number is listed in the file.

The next state information is written using a 'case' structure since the DECISION structure is a variant record. The type of decision structure is written first and the next state data is then saved in the form of a simple truth table. The zeros and ones in the table represent inactive and active variable values and a hyphen represents a 'don't care' entry. There may be up to three input variables and they are stored in the strings NAM0, NAM1 and NAM2. If there are three decisions involved in a state, there may be either two or three input variables involved in the structure. This is determined by checking the values of NAM2 and NAM3 for equality. If they are not equal, three variables must be entered in the truth table.

The READASM procedure reads files created by SAVEASM and constructs the machine model in memory. The procedure begins by calling SHOWINDEX to display the index file. If the value of IER is true, the index file is empty and there are no files available for reading. This is not considered an error since the user cannot view the index file without requesting a file operation. If there are machines in the directory, GETFNAME is called and the index file contents are scanned to verify that the machine requested is present in the directory. If a match is found (IER = true), the file is opened for reading in the same

manner as in the SAVEASM procedure. If a match is not found an error message is displayed and the user must reenter a machine name. The error message contains the invalid file name.

The machine is read into memory in the same sequence in which it was saved. The call to INITIALIZE clears existing states from memory, equates NOVARS to false and erases the variable names contained in ASMVAR. The string variable DUMY20 and the character CH are used to skip over the text strings advancing the text pointer to relevant data. The variable list ASMVAR is read and the lookup tables LEVTABLE and TYPETABLE are used to assign enumeration values to VACTIVE and VUSE.

The states are read using a 'case' structure similar to the structure used in SAVEASM. INITPTR is called to create new STATE and DECISION records. It is noted that INITPTR does not create new states unless the pointer value is 'nil'. The members of the array STATESSET are assigned 'nil' in the call to INITIALIZE. This insures that all of the states in the file are assigned and that duplicate states are not created. To assign the next state pointers, the decision structures are read using a 'for' loop and a 'case' structure.

The procedure DELASM deletes files created by SAVEASM. Standard Pascal does not support the deletion of disk files. The external procedure DELFILE accomplishes this task by calling the TIP procedure SVC\$. The DELFILE procedure is presented in the final section of this chapter. DELASM begins by displaying the index file and if it is not empty, a file name is requested by GETFNAME. The index file is searched for the file name returned by GETFNAME, and if it is not found, an error

message is issued. This process is identical to the READASM screening process explained earlier.

If the file name is valid, the user is given a chance to abort the file deletion. The prompt 'Are You Sure (Y or N) ?' is displayed and the user must change the default value of N to Y in order to continue. The extra key stroke tends to reduce accidental erasure of machine files. If the answer to this prompt is Y, the string variable PATHNAME is constructed and it is passed to DELFILE for disk deletion. The machine file name is then deleted from the index file by calling UPDATEIX and the default machine name MACHNAME is erased, completing the deletion process.

The ASMTABLE procedure constructs the truth table for the machines. It contains the BINARY procedure which converts state numbers into binary strings. The call to BINARY has the form 'BINARY(LIN,P,NUM)'. The variable LIN is a 48 character table line, P is the starting position in the string and NUM is the state number to be converted. The integer variable NBIT represents the number of bits required to uniquely define each state. It is generated and used in ASMTABLE but it is also used by BINARY for loop control.

Procedure ASMTABLE begins by calling FILEIO to read a machine into memory. This insures that the truth table file and the machine listing file contain the same data. The SCISTRING variable MNAME is constructed from the machine name by replacing the second character with a dollar sign. The Pascal file to be written is the text file TABLE. The variables INS and OUTS are sets of input and output variable numbers which are constructed from the global variable ASMVAR. The maximum

number of lines that a table entry may have is four (BRANCH = DECDEC) and the array ENTRY contains these strings. The variable N is used to determine the number of lines in ENTRY to be written into the file. The variable CP is defined as the center point of the table which separates the input and output entries. The variable LINE is composed of hyphens and plus symbols and is used to separate the states.

For each state in the ASM, there may be up to four lines of data. Each line contains four sections. These are the (1) present state number, (2) next state number, (3) input listing, and (4) output listing. The present state number is common to the four lines of ENTRY and since conditional outputs are not allowed in ASMDESIGN, the output listing is also common. These two quantities are calculated for the first line of ENTRY and the other three lines are assigned its value before the decision structure is analyzed.

The decision structure is determined by the value of BRANCH. The next state numbers and input listings are constructed within a 'case' statement. The next state numbers are assigned with the BINARY procedure and the value of N is assigned in the 'case' statement. The input listing is assigned 'don't care' symbols and the zeros and ones are entered by writing over the 'don't care' entries. Once the array ENTRY is constructed, it is written into the file with a 'for' statement. After the states have been written, the active level information is determined from the variable ASMVAR. By convention, the active level of each state variable is high.

Variable Assignment Procedures

The variable assignment procedures are contained in the procedure VARASIGN and they are VARTABLE and MODVARS. Procedure VARASIGN is called from the main program with the VA command and its structure is similar to the main program. The variable CHOICES contains the legal commands in this mode and the boolean variable DONE returns control to the main program. The MENU procedure is called to display the Variable Assignment menu and GETCMD is called to accept user commands. Error checking is performed at this point to determine if the user--requested command is valid. Variables may not be inserted if the variable table is full and they may not be modified if the table is empty. The variable table will not be displayed if it is empty. If any of these conditions are true an error message is issued and the user must request a new command. The variable DONE is assigned true when the user enters the RM command.

The VARTABLE procedure displays the contents of ASMVAR on the VDT. The procedure begins by clearing the screen and displaying the constant HEADER on line one. The GRAPHICS procedure is called to display four vertical lines which segregate the entries in HEADER. For each variable in ASMVAR with VNAME not equal to BLANK, the four variable properties are displayed and the current line variable LIN is incremented. Since the maximum number of variables (32) is greater than the number of lines on the VDT (24), the value of LIN is checked to determine the current screen position. The procedure displays the first 20 variables and if there are more variables in the list, the prompt 'Press Return to Continue:' is displayed and procedure WAIT is

called. After the user enters a carriage return, the entries are scrolled up by 10 lines and the next set of variables is displayed. This process continues until the complete variable table has been displayed.

Once ASMVAR has been scanned, the bottom line of the table is drawn indicating the end of the variable list. The number of input and output variables in the listing is also displayed on line 22 of the VDT. The WAIT procedure is called to hold the screen until a carriage return is entered. The table structure displayed is identical to the structure written by SAVEASM.

The call to MODVARS has the form 'MODVARS(CH)', where CH is a CMD type variable which specifies either the insert or modify variables mode. The procedure begins by calling MENU which displays the screen shown in Figure 7, page 6. If the value of CH is IV (Insert Variables), the counter variable CNTR is incremented until an unused variable position is found in ASMVAR. If the value of CH is MV (Modify Variables), the variable number CNTR is displayed as a default value and the operator may accept this value or enter a different number.

Once the present variable number is entered, the properties of the variable are displayed as default values and the user may accept or modify these values. The variable is deleted from the table by erasing its name. The variable use information and active level are entered as string variables and the enumeration values are determined with the lookup tables LEVTABLE and TYPETABLE. If invalid responses are given for these values, the cursor will not advance to the next field. This

is accomplished by accepting user responses within a 'repeat' statement and checking the value of the boolean variable DONE.

The procedure continues until the variable CON is 'N '. The procedure calculates the value of VNUMIN and VNUMOUT by counting the number of INPV and OUTV variables in ASMVAR. The boolean variable NOVARS is assigned true if the sum of VNUMIN plus VNUMOUT is zero.

State Assignment Procedures

The seven state assignment procedures are contained in the procedure ASMCHART. The structure of this procedure is the same as the structure of VARASIGN. The procedures contained in ASMCHART are DRAWVARS, GETSTR, GETSPTR, INVARS, OTVARS, DELSTATE and MODSTATES. The ASMCHART procedure is called from the main program with the AC command and control is returned to the main program with the RM command. The variable CHOICES contains the valid set of commands for this mode and the variable CHOSEN contains the user's response. The MENU procedure is called to display the ASM Charts menu and GETCMD is called to accept user commands. Error checking is performed to determine if the user--requested command is valid. States may not be assigned or modified if the variable list is empty and states may not be modified if the state list is empty. An error message is displayed if any one of these conditions is true. The user must enter a new command.

The call to DRAWVARS has the form 'DRAWVARS(CH)', where CH may be either INPV or OUTV. This procedure displays the input or output variable names on the right side of the VDT. The list heading 'INPUT' or 'OUTPUT' is displayed to inform the user which variable type is

displayed. Since there are more variable entries than screen lines, two columns of variable names may be displayed. This situation is similar to the VARTABLE discussion. Once the variable list is displayed, the remaining table entries are erased preventing incorrect variable names in the list.

The call to GETSTR has the form 'GETSTR(L,C,INP,CH)'. This procedure assigns an input variable number to INP and draws decision boxes at the screen coordinates L and C. The variable CH is passed to the GRAPHICS procedure since different decision primitives are available. If the value of INP is not zero, a default variable name is displayed in the decision box.

The procedure begins with a call to GRAPHICS to draw the requested decision box. Procedure DRAWVARS is called to display the list of input variables available. The STRING8 type variable DEF represents the default variable name and the two STRING4 variables are constructed from DEF. These two variables are displayed and the user may accept them or enter an alternate name. The variable NAMEV is constructed from HALF1 and HALF2 and the variable list is searched for a correct match. The variable must be present in the list, it must not equal BLANK and it must be an input variable. If these conditions are not met, an error message is displayed and the name must be reentered.

The call to GETSPTR has the form 'GETSPTR(L,C,NS)'. This procedure returns the state pointer NS which corresponds to the user's state number. If the value of NS is not 'nil', a default state number is displayed. The procedure begins with a call to GRAPHICS and a small state box is drawn at the screen coordinates L and C. The prompt

'...Enter Next State...' is displayed and the default state number is positioned inside the graphic primitive. If the state number returned is greater than the maximum state number in the present machine, an error message is issued. When a valid state number is accepted, INITPTR is called to initialize the pointer in the array STATESSET and the pointer NS is assigned this value.

The INVARS procedure constructs decision structures from the user responses. The call to INVARS has the form 'INVARS(L,C,DEC)', where DEC is the decision variable to be assigned. The procedure begins by assigning the STRING2 variables ZERO and ONE. These are the default values for the prompt and they depend on the value of BRANCH in the DEC record. The GETSTR procedure is called to accept the first decision box and the prompt is displayed. The default value of ZERO is displayed below the zero branch of the decision structure and the operator may change the value or enter the default. The same technique is used to enter the ONE variable below the one branch of the decision box.

If a decision is requested in the variable ZERO or ONE and the screen line is greater than eight, the next decision structure will not fit on the screen without wraparound. The screen is scrolled up L minus nine lines and DRAWVARS is called to reposition the variable list. If a decision structure is modified to a simpler configuration, DECISION records must be disposed of to conserve heap space. This is accomplished by comparing the present value of BRANCH with the requested values contained in ZERO and ONE. For example, if the new value of BRANCH is STASTA (ZERO = ONE = 'S ') and the old value is DECSTA, the decision pointer NEXT5 must be disposed of before BRANCH is equated to

STASTA. Once BRANCH is assigned STASTA, the new state pointers NEXT1 and NEXT2 may be assigned with calls to GETSPTR. Each of the four combinations of ZERO and ONE must be checked in this manner to insure that memory is not lost and the lists are correctly implemented.

The call to OTVARS has the form 'OTVARS(L,C,LW)', where the LONGWORD variable LW is assigned. This procedure constructs the present state box from the user's response. The procedure begins with calls to DRAWVARS and GRAPHICS which list the output variable names and draws the top of the state box respectively. The prompt '...Enter Active Variables...' is displayed and the variable names are entered in three stages.

The first stage checks the values contained in LW for default variable names. If there are default variables specified in LW they are displayed, the screen coordinates L and C are advanced and the variable NVAR is incremented. The array VSET is constructed with the index representing the order of the variable listing and the array value representing the variable number.

The second stage allows the user to modify the default values displayed in the first stage. The screen coordinates L and C are reset to the starting position. This operation is controlled with a 'for' loop counting from one to NVAR. The STRING8 type variable NAMEV is read and it is checked for three different possibilities. A variable is deleted from the list by erasing its name. If this condition occurs, the variable LW is modified using VSET to delete the variable. If the default value is entered, the array LW does not change and the next field is tested. If the default name is replaced by a valid variable

name, two entries of LW must be modified. If none of these conditions occur the cursor does not advance and the user must enter another variable name.

In the third stage of variable entry, the user enters new variable names until a blank field is encountered. The variable NAMEV must (1) be present in the variable table, (2) be an output variable, (3) not be equal to BLANK, and (4) not be a duplicate name. Once a blank field is encountered, the bottom of the state box is drawn and the screen coordinates L and C are advanced to point to the starting position for INVARS.

The call to DELSTATE has the form 'DELSTATE(N)', where N is the number of states to be deleted. This variable is used for loop control and the set BAD represents the set of deleted state numbers. The procedure begins by requesting the state number to be deleted. An error message is issued if the state number given is larger than the maximum state number or it is contained in BAD.

The deletion process begins by checking every state in the machine for references to the bad state number. This is accomplished by generating the boolean array STAT which contains 13 possibilities. An array is used so that it may be scanned with a 'for' loop. If one of the members of STAT is true, the program halts and the user must change the next state number. The state name and number are displayed, the input variable name and number are displayed and a next state number is requested. The decision structure may not be changed with this process.

A valid state number is entered with a call to GETSPTR and the correct pointer is connected using the value of K and the temporary

state pointer TEMP. Once the states have been deleted from the machine they are disposed of and the state numbers are resequenced, eliminating unused state numbers. The variable MXSTNUM is decremented which completes the deletion process.

The call to MODSTATES has the form 'MODSTATES(CH)', where CH may be either AS (Assign States) or MS (Modify States). This procedure has a similar structure to the MODVARS procedure discussed in the previous section of this chapter. The procedure begins by prompting the operator to enter the total number of states with the default value given. If an invalid number is entered an error message is displayed.

If the procedure is called to assign states, the state number PRES is advanced until either an unassigned state is found or a partially completed state is found. If an incomplete state does not exist, an error message is issued and control is returned to the ASMCHART procedure. If the procedure is called to modify states, a state number is requested and error checking is performed on the entered state number. If an error occurs the operator must enter a new state number.

Once a state number has been determined it is initialized by calling INITPTR, the state name is entered and the OTVARS procedure is called to construct VARLIST. The prompt 'Is There a Decision Block (Y or N) ?' is displayed and the variable ANS is read. If ANS is 'Y ', the INVARS procedure is called, otherwise the variable BRANCH is assigned NON and the GETSPTR procedure is called. After a state has been completed, the variable PRES is incremented and the user is asked to

continue. The default value for this prompt is Y until the variable PRES is larger than the maximum state number.

External Procedures

The external procedures for ASMDESIGN are contained in the two dummy programs ASM1 and VDTP. The ASM1 program contains utility procedures specific to ASMDESIGN and the VDTP program, provided by Texas Instruments, contains procedures which perform VDT I/O. Some of these procedures have been modified by the author to accommodate ASMDESIGN and two (SCROLL and DELFILE) have been added to take advantage of the data structure it contains. The ASM1 program will be discussed in detail and the VDTP program will be discussed in general.

The ASM1 program contains seven procedures. These are WAIT, IDISPLAY, IACCEPT, MENU, ERRORMSG, GETCMD and GRAPHICS. The WAIT procedure contains a DISPLAY and an ACCEPT statement and is used to suspend program operation until the user enters a carriage return. This simple operation is useful for interactive programs.

The IDISPLAY and IACCEPT procedures display and accept integer values from the VDT. The range of integers for these procedures is zero through 99. IDISPLAY converts the integer into the STRING2 variable SNUM using the 'div' and 'mod' operators. IACCEPT converts the STRING2 variable USER into the integer NUM using the boolean variables DIG1 and DIG2 to validate correct character input.

The MENU procedure displays the five menus required by ASMDESIGN and the variable CH selects the menu. The procedure is composed of a call to CLEARSCREEN, a title line and a 'case' statement.

The ERRORMSG procedure displays the 13 error messages and its call has the form 'ERRORMSG(BLOCK,ER,F,PAUSE)'. The BLOCK array must be passed to all external procedures which use the VDT procedures. The variable ER is the error number corresponding to the numbers shown in Figure 2, page 7. The FILENAME type variable F is used for error message number six. The boolean variable PAUSE determines if a call to WAIT is required.

The GETCMD procedure accepts user commands from the various menus. Its call has the form 'GETCMD(BLOCK,OKSET,CH)'. The parameter OKSET contains the set of valid commands and the variable CH is returned to the calling program or procedure. The procedure begins by constructing the lookup table CSET and the variable CH is assigned NU. The value of NU is used to display the Main Menu and since it is never in OKSET, it is assigned to CH to prevent interference from the calling procedure. The prompt 'Select an Option:' is displayed and if the user enters an invalid response an error message is issued and the user must reenter a command.

The GRAPHICS procedure displays the graphic primitives required by the ASMDESIGN program. The call to this procedure has the form 'GRAPHICS(BLOCK,CH,L,C)', where the character CH determines one of the eight primitives to be drawn at screen coordinates L and C. The 'V' and 'H' values construct vertical and horizontal lines for VARTABLE; the 'T', 'S' and 'B' values represent the top, side and bottom of the large state box; the 'X' character constructs the small state box; the 'D' and 'E' values draw decision boxes. The decision boxes are constructed in the two dimensional array DSTR. Since some characters may not be

entered in the text editor, the graphics characters are calculated by subtracting 40 from the ASCII code for the characters entered.

The procedures ACCEPT, CLEARSCREEN, INITSCREEN and DISPLAY are provided by Texas Instruments. The INITSCREEN procedure opens the VDT for I/O and it must be called before the remaining three are called. This procedure has been modified to assign a LUNO to the user's VDT. The ACCEPT procedure has been modified allowing user input to be entered in the high intensity mode. For further information on these procedures, refer to [3].

The SCROLL procedure scrolls the VDT screen by reading and writing lines directly to the screen buffer of the 911 VDT. The call to SCROLL has the form 'SCROLL(BLOCK,TOP,BOTTOM,NLINES)', where TOP and BOTTOM are the top and bottom lines of the portion of the screen to be scrolled and NLINES represents the number of lines to scroll. Access to the DX10 operating system is gained with the call to SVC\$ using the I/O call block data structure contained in the VDTP program. For an extensive discussion of this data structure, refer to[5,6].

The DELFILE procedure deletes files from the ASM directory. It is included in the VDTP program for access to the I/O call block. the call to this procedure has the form 'DELFILE(PATH)' where PATH is a SCISTRING in which the pathname is contained. The deletion is accomplished with a call to SVC\$.

This chapter has presented a detailed description of the procedures contained in ASMDESIGN. It is intended to be read concurrently with the Pascal program contained in the Appendix and it provides a word description of the code for future development of the

ASMDESIGN program. The final chapter of this thesis contains a summary of the project with a discussion of the program limitations and suggestions for future modifications of the program.

CHAPTER V

SUMMARY AND CONCLUSIONS

The ASMDESIGN program presented in this thesis is complex due to the design goals outlined in Chapter III. Complexity is also introduced into the software due to the data structure. When this project began, little was known about the internal structures of the TI-990/12 computer and knowledge was gained in the development of the program. Limitations were imposed on the ASM model during program development, since the author soon realized that all of the desired features could not be implemented in a single program. This chapter contains a discussion of the limitations of ASMDESIGN as well as suggestions for future work in this field.

Program Limitations

The machine modeled by ASMDESIGN may contain up to 32 I/O variables in 64 states. These limitations are not critical but the limited complexity of the decision structures presents an important problem. The program does not recognize decision structures in which four or more input variables are to be tested in a single state. The data structure can accommodate larger linked lists of DECISION records but the variant nature of the DECISION record makes the implementation of the lists cumbersome. This becomes apparent when the complexity of the SAVEASM, READASM and ASMTABLE procedures is realized. A DECISION record must be designed that lends itself well to 'for' loops with the absence of 'case' statements.

Another limitation to the model is that it does not support conditional output variables. The original plan for the program was to have such outputs by including a LONGWORD type variable in the DECISION record. The VARTYPE type would then be composed of three members with the third representing a conditional output. This limitation was imposed when the size of the program became an important factor. A large ammount of effort was made to reduce the amount of code and as a result, the dual mode versions of the MODVARS and MODSTATES procedures were developed.

Future Modifications

The state table data structure must be modified so that the state manipulation code may be minimized without loosing the error checking capabilities of the program. This could be accomplished by eliminating the variant part of the DECISION record and replacing it with arrays of state and decision pointers. The problem arises from the fact that decisions may point to states or decisions and compile time errors result from assigning decision pointers to state pointer type variables.

The ASMDESIGN program may be divided into a family of programs which communicate through the ASM directory data base. This may be accomplished by sharing the FILEIO procedures with smaller programs for (1) variable assignment, (2) state assignment, and (3) machine table functions. The link editor on the TI-990/12 is capable of sharing common procedures with different programs or tasks. Once the program is

split into smaller programs, other programs may be written to perform a variety of tasks.

A program may be written to read a machine table and optimize the logic it contains. A 'Quine-McClusky' program may be developed which minimizes the truth table logic and derives minimal logic equations for an FPLA implementation. A program may also be written to read the table and use the PROM programmer utility available on the TI-990 computer.

When this project was conceived it was also desired to generate timing charts on the line printer. Although idealized charts are relatively simple to generate, an extensive timing analysis program could be developed with component dependent access times included in the analysis. A program like this could calculate maximum clock rates for various component combinations.

LIST OF REFERENCES

LIST OF REFERENCES

1. Peatman, John B. Digital Hardware Design. New York: McGraw-Hill, 1980.
2. Texas Instruments. Model 990 Computer DX10 Pascal Programmer's Guide. Part Number: 2270528-9701, n.p.: January 1984.
3. Texas Instruments. Model 990 Computer TI Pascal Reference Manual. Part Number: 2270519-9701, n.p.: January 1984.
4. Texas Instruments. Model 990 Computer DX10 Operating System Concepts and Facilities. Volume I. Part Number: 946250-9701, n.p.: September 1983.
5. Texas Instruments. Model 990 Computer DX10 Operating System Operations Guide. Volume II. Part Number: 946250-9702, n.p.: September 1983.
6. Texas Instruments. Model 990 Computer DX10 Operating System Application Programming Guide. Volume III. Part Number: 946250-9703, n.p.: September 1983.
7. Texas Instruments. Model 990 Computer TI Pascal Configuration Processor Tutorial. Part Number: 2250098-9701, n.p.: January 1984.
8. Foley, James D., and Andries Van Dam. Fundamentals of Interactive Computer Graphics. Reading, Mass.: Addison-Wesley, 1982.
9. Kohavi, Zvi. Switching and Finite Automata Theory. 2nd ed. New York: McGraw-Hill, 1978.
10. Texas Instruments. Model 990 Computer Link Editor Reference Manual. Part Number: 949617-9701, n.p.: December 1979.
11. Zaks, Rodney. Introduction to Pascal (Including UCSD Pascal). 2nd ed. Berkeley: Sybex Inc., 1981.
12. Welsh, Jim, and John Elder. Introduction to Pascal. 2nd ed. Englewood Cliffs, N.J.: Prentice-Hall International, 1982.
13. Jensen, Randall W. "Structured Programming." Computer, March 1981, pp 31--48.

APPENDIX

```

program ASMDESIGN;
(*****
* TITLE      : ASMDESIGN          BY : Thomas Tomcik          1.0.0 *
* LANGUAGE   : TI PASCAL                      *
* COMPUTER   : TI 990/12 with DX10 3.6.0      *
* ABSTRACT   : This program models a sequential machine for the design *
*              of digital circuits. The user enters the machine by *
*              specifying the variables used and then the states are *
*              entered in flowchart form. The program then derives the *
*              logic tables according to the user responses.          *
* FILES      : ASMINDEX -- Index of the machine stored on disk.      *
*              MACHFIL  -- ASM machine file.                          *
*              --PROCEDURES CALLED--                                *
* INTERNAL   : INITIALIZE, FILEIO, VARASIGN, ASMCHART, ASMTABLE, HELP. *
* EXTERNAL   : MENU, GETCMD, CLEARSCREEN, OVLY$.                      *
*****)

const MAXNUMST = 63; (* Maximum State Number. *)
      MAXNUMVAR = 32; (* Maximum Number of Variables. *)
      MAXNUMACH = 6;  (* Maximum Number of Machines on Disk. *)
      MAXLENST  = 20; (* Maximum Length of State Name. *)
      MAXLENVAR = 8;  (* Maximum Length of Variable Name. *)
      MAXLENFIL = 24; (* Maximum Length of File Name. *)
      BLANK     = '      '; (* Eight Spaces. *)

type  VARNUM    = 1..MAXNUMVAR;
      VARNUM0   = 0..MAXNUMVAR;
      BYTE      = 0..#FF;
      VARNAME   = packed array[ 1..MAXLENVAR ] of char;
      SCISTRING = packed array[ 0..MAXLENFIL ] of char;
      FILENAME  = packed array[ 1..MAXLENFIL ] of char;
      STRING2   = packed array[ 1..2 ] of char;
      STRING4   = packed array[ 1..4 ] of char;
      STRING8   = packed array[ 1..8 ] of char;
      BTYPE     = array[ 1..16 ] of integer;
      VARLEV    = (HIGH,LOW);
      VARTYPE   = (INPV,OUTV);
      CMD       = (SD,DD,RD,VA,AC,MT,HF,QU,NU,IV,MV,VT,AS,MS,RM);
      CSET      = set of CMD;

      ONEVAR    = record
                      VNAME   : VARNAME;
                      VACTIVE : VARLEV;
                      VUSE    : VARTYPE
                    end;

      ALLVAR    = record
                      VNUMIN  : VARNUM0;
                      VNUMOUT : VARNUM0;
                      ALL     : array[ VARNUM ] of ONEVAR
                    end;

```

```

STATENUM = 0..MAXNUMST;
LONGWORD = packed array[ VARNUM ] of 0..1;
STATENAME = packed array[ 1..MAXLENST ] of char;
DECTYPE = (NON,STASTA,STADDEC,DECSTA,DECDEC);
SPTR = @STATE;
DPTR = @DECISION;

DECISION = record
    INPVAR : VARNUM0;
    case BRANCH : DECTYPE of
        NON : (NEXT0 : SPTR);
        STASTA : (NEXT1,NEXT2 : SPTR);
        STADDEC : (NEXT3 : SPTR; NEXT4 : DPTR);
        DECSTA : (NEXT5 : DPTR; NEXT6 : SPTR);
        DECDEC : (NEXT7,NEXT8 : DPTR)
    end;

STATE = record
    NUMBER : STATENUM;
    NAME : STATENAME;
    VARLIST : LONGWORD;
    DECS : DECISION
end;

var BLOCK : BTYPE;
ENDMES : char;
MACHINX : text;
PREFIX : FILENAME;
ASMVAR : ALLVAR;
CHOICES : CSET;
CHOSEN : CMD;
DONE,NOVARS : boolean;
LEVTABLE : array[ VARLEV ] of STRING8;
TYPETABLE : array[ VARTYPE ] of STRING8;
MACHNAME : STRING8;
MXSTNUM : STATENUM;
STATASET : array[ STATENUM ] of SPTR;

procedure CLEARSCREEN(var BLOCK:BTYPE); EXTERNAL;
procedure SET$ACNM(var F:text; var ACNM:SCISTRING); EXTERNAL;
procedure DISPLAY(var BLOCK:BTYPE; LINE,COLUMN:integer;
    BUFFER:packed array[ 1..? ] of char; LENGTH:integer); EXTERNAL;
procedure ACCEPT(var BLOCK:BTYPE; LINE,COLUMN:integer;
    var BUFFER:packed array[ 1..? ] of char;
    LENGTH:integer; var TC:char); EXTERNAL;
procedure SCROLL(var BLOCK:BTYPE; TOP,BOTTOM,NLINES:BYTE); EXTERNAL;
procedure OVLY$(LOAD:integer); EXTERNAL;
procedure WAIT(var BLOCK:BTYPE); EXTERNAL;
procedure IDISPLAY(var BLOCK:BTYPE; LIN,COL,NUM:integer); EXTERNAL;
procedure IACCEPT(var BLOCK:BTYPE; LIN,COL:integer;
    var NUM:integer); EXTERNAL;

```

```

procedure MENU(var BLOCK:BTYP;  MODE:CMD);  EXTERNAL;
procedure ERRORMSG(var BLOCK:BTYP;  ER:integer;
    F:FILENAME;  PAUSE:boolean);  EXTERNAL;
procedure GETCMD(var BLOCK:BTYP;  OKSET:CSET;  var CH:CMD);  EXTERNAL;
procedure GRAPHICS(var BLOCK:BTYP;  CH:char;  L,C:integer);  EXTERNAL;

procedure INITIALIZE(FIRSTTIME:boolean);  FORWARD;
procedure INITPTR(var S:SPTR;  var D:DPTR;  DEC:boolean;
    NUM:STATENUM);  FORWARD;
procedure HELP();  FORWARD;
procedure FILEIO(CH:CMD);  FORWARD;
procedure VARASIGN();  FORWARD;
procedure ASMCHART();  FORWARD;
procedure ASMTABLE();  FORWARD;

begin (* MAIN *)
    OVLY$(0);  INITIALIZE(true);
    DONE:=false;  CHOICES:=[SD,DD,RD,VA,AC,MT,QU,HF];
    repeat
        MENU(BLOCK,NU);  GETCMD(BLOCK,CHOICES,CHOSEN);
        case CHOSEN of
            SD,DD,RD :  begin
                            OVLY$(1);  FILEIO(CHOSEN)
                        end;
            VA      :  begin
                            OVLY$(2);  VARASIGN
                        end;
            AC      :  begin
                            OVLY$(3);  ASMCHART
                        end;
            MT      :  ASMTABLE;
            HF      :  HELP;
            QU      :  DONE:=true
        end (* case *)
    until DONE
end.  (* MAIN *)

```

```

procedure INITPTR(* var S:SPTR; var D:DPTR; DEC:boolean;
                  NUM:STATENUM *);
(*****
* PROCEDURE INITPTR;                                     1.1.0 *
* PURPOSE   : This procedure initializes decision and/or state *
*             pointers.                                     *
* INPUTS    : DEC -- If true, only a decision pointer is initialized. *
*             NUM -- Default state number if DEC is false.         *
* OUTPUTS   : S -- Initialized state pointer.                   *
*             D -- Initialized decision pointer.                 *
*             --PROCEDURES CALLED--                             *
* INTERNAL  : NONE.                                             *
* EXTERNAL  : NONE.                                             *
*****)

var I : integer;

begin (* INITPTR *)
  if DEC then
    begin
      new(D);
      with D@ do
        begin
          INPVAR:=0; BRANCH:=STASTA; NEXT1:=nil; NEXT2:=nil
        end
      end
    end
  else if STATESET[NUM] = nil then
    begin
      new(S); new(D); S@.DECS:=D;
      with S@,D@ do
        begin
          for I:=1 to MAXLENST do NAME[I]:=' ';
          NUMBER:=NUM; INPVAR:=0; BRANCH:=NON; NEXT0:=nil;
          for I:=1 to MAXNUMVAR do VARLIST[I]:=0
        end
      end
    end
  end; (* INITPTR *)

```

```

procedure INITIALIZE;(* FIRSTTIME:boolean *);
(*****
* PROCEDURE INITIALIZE;                                1.2.0 *
* PURPOSE   : This procedure initializes the system variables and the *
*             index file.  If no index file is found in the ASM *
*             directory, one is created with empty members.      *
* INPUT      : FIRSTTIME -- Boolean variable which determines the type *
*             of initialization requested.                         *
* OUTPUT     : NONE.                                             *
*             --PROCEDURES CALLED--                               *
* INTERNAL   : NONE.                                             *
* EXTERNAL   : INITSCREEN, FIND$SYN, SET$ACNM.                   *
*****)

var  OLD      : boolean;
     TODAY    : STRING8;
     SYN,VAL   : SCISTRING;
     I        : integer;
     ASM      : packed array[ 1..9 ] of char;

procedure INITSCREEN(var BLOCK:BTYP);  EXTERNAL;
procedure FIND$SYN(var SYNONYM,VALUE:SCISTRING);  EXTERNAL;

begin (* INITIALIZE *)
  NOVARS:=FIRSTTIME;  MXSTNUM:=0;  ASM:='.ASMINDEX';
  for I:=1 to MAXNUMVAR do with ASMVAR,ALL[I] do
  begin
    VNUMIN:=0;  VNUMOUT:=0;  VNAME:=BLANK;  VACTIVE:=HIGH;  VUSE:=OUTV
  end;
  if FIRSTTIME then
  begin
    LEVTABLE[HIGH]:= 'HIGH      ';  LEVTABLE[LOW]:= 'LOW      ';
    TYPETABLE[INPV]:= 'INPUT    ';  TYPETABLE[OUTV]:= 'OUTPUT   ';
    MACHNAME:=BLANK;  INITSCREEN(BLOCK);
    SYN[0]:=chr(3);  VAL[0]:=chr(MAXLENFIL);
    for I:=1 to 3 do SYN[I]:=ASM[I+1];  FIND$SYN(SYN,VAL);
    for I:=1 to MAXLENFIL do if I < 16 then PREFIX[I]:=VAL[I]
    else begin
      PREFIX[I]:= ' ';  VAL[I]:=ASM[I-15]
    end;
    VAL[0]:=chr(MAXLENFIL);  PREFIX[16]:= '.';
    SET$ACNM(MACHINX,VAL);  IOTERM(MACHINX,OLD,false);  reset(MACHINX);
    if STATUS(MACHINX) <> 0 then
    begin
      rewrite(MACHINX);  DATE(TODAY);
      for I:=1 to MAXLENFIL do write(MACHINX,VAL[I]);
      writeln(MACHINX,TODAY);
      for I:=1 to MAXNUMMACH do writeln(MACHINX,PREFIX,TODAY)
    end;
    close(MACHINX)
  end else

```

```

for I:=0 to MAXNUMST do if STATESET[I] <> nil then
with STATESET[I]@ do
begin
  case DECS@.BRANCH of
    NON,STASTA : ;
    STADEC      : dispose(DECS@.NEXT4);
    DECSTA      : dispose(DECS@.NEXT5);
    DECDEC      : begin
                        dispose(DECS@.NEXT7); dispose(DECS@.NEXT8)
                      end
  end; (* case *)
  dispose(DECS); dispose(STATESET[I])
end;
for I:=0 to MAXNUMST do STATESET[I]:=nil
end; (* INITIALIZE *)

```

```

procedure HELP;
(*****
* PROCEDURE HELP;                                     1.3.0 *
* PURPOSE : This procedure displays the help file on the screen. *
*           The file name is '.RES.RES548.ASM.HELP$'.           *
*           --PROCEDURES CALLED--                               *
* INTERNAL : NONE.                                             *
* EXTERNAL : CLEARSCREEN, SET$ACNM, DISPLAY, WAIT.           *
*****)

var  HF      : text;
     LIN     : integer;
     LOC     : FILENAME;
     HNAME   : SCISTRING;
     LINE    : packed array[ 1..72 ] of char;

begin (* HELP *)
  LOC:='.RES.RES548.ASM.HELP$  ';
  CLEARSCREEN(BLOCK); HNAME[0]:=chr(MAXLENFIL);
  for LIN:=1 to MAXLENFIL do HNAME[LIN]:=LOC[LIN];
  LIN:=1; SET$ACNM(HF,HNAME); reset(HF);
  while not eof(HF) do
    begin
      readln(HF,LINE); DISPLAY(BLOCK,LIN,1,LINE,72);
      if LIN = 23 then
        begin
          WAIT(BLOCK); CLEARSCREEN(BLOCK); LIN:=1
        end else LIN:=LIN+2
      end;
      close(HF); if LIN <> 1 then WAIT(BLOCK)
    end; (* HELP *)

```

```

procedure FILEIO(* CH:CMD *);
(*****
* PROCEDURE FILEIO;                                     1.4.0 *
* PURPOSE   : This procedure contains three file operations used in *
*             the ASMDESIGN program. They are 1): read from disk, 2): *
*             save to disk and 3): delete from disk.                *
* INPUT      : CH -- Command type variable which selects the operation.*
* OUTPUT     : Machine file corresponding to the name 'MACHNAME'.    *
*             --PROCEDURES CALLED--                                *
* INTERNAL   : SAVEASM, READASM, DELASM, ERRORMSG.                  *
* EXTERNAL   : NONE.                                                *
*****)

procedure DELFILE(PATH:packed array[ 0..? ] of char); EXTERNAL;
procedure SHOWINDEX(var EMPTY:boolean; CH:CMD); FORWARD;
procedure UPDATEIX(F:FILENAME; DELETE:boolean;
    var FULL:boolean); FORWARD;
procedure GETFNAME(var F:FILENAME); FORWARD;
procedure SAVEASM(); FORWARD;
procedure READASM(); FORWARD;
procedure DELASM(); FORWARD;

begin (* FILEIO *)
    case CH of
        SD : if NOVARS then ERRORMSG(BLOCK,9,PREFIX,true) else SAVEASM;
        RD : READASM;
        DD : DELASM
    end (* case *)
end; (* FILEIO *)

```

```

procedure SHOWINDEX(* var EMPTY:boolean; CH:CMD *);
(*****
* PROCEDURE SHOWINDEX;                                     1.4.1 *
* PURPOSE   : This procedure displays the contents of the index file. *
*           : Three different titles are displayed depending upon *
*           : the calling procedure.                               *
* INPUT     : CH -- Determines the type of operation to be performed. *
* OUTPUT    : EMPTY -- True if the machine index is empty.         *
*           : --PROCEDURES CALLED--                                *
* INTERNAL  : NONE.                                               *
* EXTERNAL  : CLEARSCREEN, DISPLAY, IDISPLAY.                     *
*****)

const HEADER1 =
    'Index Filename:                                     Date Created: ';
    HEADER2 = '):   Filename:                           Date: ';

var  FNAME      : FILENAME;
     CDATE,SNAME : STRING8;
     LIN,I       : BYTE;
     ST          : STRING2;

begin (* SHOWINDEX *)
    EMPTY:=true; LIN:=5; CLEARSCREEN(BLOCK);
    ST[1]:=chr(15); reset(MACHINX);
    case CH of
        RD : DISPLAY(BLOCK,1,30,'Read From Disk File.',20);
        SD : DISPLAY(BLOCK,1,31,'Save to Disk File.',18);
        DD : DISPLAY(BLOCK,1,29,'Delete From Disk File.',22)
    end; (* case *)
    DISPLAY(BLOCK,3,9,HEADER1,80); readln(MACHINX,FNAME,CDATE);
    DISPLAY(BLOCK,3,25,FNAME,MAXLENFIL); DISPLAY(BLOCK,3,68,CDATE,8);
    for I:=1 to 80 do DISPLAY(BLOCK,4,I,ST,1);
    while not eof(MACHINX) do
        begin
            readln(MACHINX,FNAME,CDATE);
            if FNAME <> PREFIX then
                begin
                    for I:=1 to 8 do SNAME[I]:=FNAME[I+16];
                    DISPLAY(BLOCK,LIN,20,HEADER2,40); IDISPLAY(BLOCK,LIN,17,LIN-4);
                    DISPLAY(BLOCK,LIN,35,SNAME,8); DISPLAY(BLOCK,LIN,52,CDATE,8);
                    LIN:=LIN+1; EMPTY:=false
                end
            end;
        close(MACHINX);
        if EMPTY then DISPLAY(BLOCK,6,31,'Index File Empty.',18)
    end; (* SHOWINDEX *)

```

```

procedure UPDATEIX(* F:FILENAME; DELETE:boolean; var FULL:boolean *);
(*****
* PROCEDURE UPDATEIX;                                1.4.2 *
* PURPOSE   : This procedure maintains the index file.      *
* INPUTS    : F -- Filename to be inserted or deleted.     *
*            : DELETE -- True if filename F is to be deleted.*
* OUTPUT    : FULL -- True if the machine index is full.    *
*            : --PROCEDURES CALLED--                        *
* INTERNAL  : NONE.                                         *
* EXTERNAL  : ERRORMSG.                                    *
*****)

var  NAMES : array[ 0..MAXNUMACH ] of FILENAME;
     DATES : array[ 0..MAXNUMACH ] of STRING8;
     FOUND : boolean;
     TODAY : STRING8;
     I,J   : BYTE;

begin (* UPDATEIX *)
  FULL:=false; FOUND:=false; DATE(TODAY); reset(MACHINX);
  for I:=0 to MAXNUMACH do
    begin
      readln(MACHINX,NAMES[I],DATES[I]);
      if NAMES[I] = F then
        begin
          if DELETE then for J:=1 to 8 do NAMES[I,J+16]:=' '
          else DATES[I]:=TODAY; FOUND:=true
        end
      end;
    for I:=1 to MAXNUMACH DO do if not FOUND and (NAMES[I] = PREFIX) then
      begin
        NAMES[I]:=F; DATES[I]:=TODAY; FOUND:=true
      end;
    if FOUND then
      begin
        rewrite(MACHINX);
        for I:=0 to MAXNUMACH do writeln(MACHINX,NAMES[I],DATES[I])
      end
    else begin
      ERRORMSG(BLOCK,5,PREFIX,FALSE); FULL:=true
    end;
    close(MACHINX)
  end; (* UPDATEIX *)

```

```

procedure GETFNAME(* var F:FILENAME *);
(*****
* PROCEDURE GETFNAME;                                     1.4.3 *
* PURPOSE : This procedure returns a filename entered by the user. *
*           The procedure also checks the filename for validation. *
* INPUT    : User machine name.                                *
* OUTPUT   : F -- Long pathname for disk I/O.                 *
*           --PROCEDURES CALLED--                             *
* INTERNAL : NONE.                                           *
* EXTERNAL : DISPLAY, ACCEPT, ERRORMSG.                      *
*****)

var  SNAME      : STRING8;
     FOUND,GOOD : boolean;
     NUM,LET    : set of char;
     I          : integer;

begin (* GETFNAME *)
  NUM:=['0'..'9']; LET:=['A'..'Z'];
  DISPLAY(BLOCK,MAXNUMACH+6,1,'Enter Machine Name:',19);
  repeat
    DISPLAY(BLOCK,MAXNUMACH+6,21,MACHNAME,8);
    ACCEPT(BLOCK,MAXNUMACH+6,21,SNAME,8,ENDMES); FOUND:=false;
    for I:=1 to 8 do if SNAME[I] in ['a'..'z'] then
      SNAME[I]:=chr(ord(SNAME[I])-32);
    GOOD:=SNAME <> 'ASMINDEX';
    if not GOOD then ERRORMSG(BLOCK,1,PREFIX,false)
  else begin
    GOOD:=SNAME[1] in LET;
    if not GOOD then ERRORMSG(BLOCK,2,PREFIX,false)
  else begin
    for I:=2 to 8 do
      begin
        if not FOUND then FOUND:=SNAME[I] = ' ';
        if GOOD and not FOUND then GOOD:=SNAME[I] in NUM+LET;
        if GOOD and FOUND then GOOD:=SNAME[I] = ' ';
      end;
    if not GOOD then ERRORMSG(BLOCK,3,PREFIX,false)
  end
  end
  until GOOD;
  F:=PREFIX; for I:=1 to 8 do F[I+16]:=SNAME[I]
end; (* GETFNAME *)

```

```

procedure SAVEASM;
(*****
* PROCEDURE SAVEASM;                                1.4.4 *
* PURPOSE   : This procedure saves the machine on disk *
*           : in tabular form.                        *
*           : --PROCEDURES CALLED--                  *
* INTERNAL  : SHOWINDEX, UPDATEIX, GETFNAME.          *
* EXTERNAL  : ERRORMSG, WAIT, SET$ACNM.              *
*****)

const ZER=' 0      '; ONE=' 1      '; DC=' -      '; BLS='      ';
      S1 = 'ASMDESIGN Sequential Machine Summary';
      S2 = 'Machine Name:      ';
      S3 = 'Number of States:  ';
      S4 = 'Number of Inputs:  ';
      S5 = 'Number of Outputs: ';
      S6 = '*****Variable Listing*****';
      S7 =
'Variable Number  Variable Name  Active Level  Input/Output';
      S8 = '*****State Listing*****';
      S9 = 'State Number:      ';
      S10= 'State Name:        ';
      S11= 'Output Listing:      ';
      S12= 'Next State:        ';
      S13= 'Decision Type:      ';
      S14= '*****';

var  FNAME      :  FILENAME;
     MACHFIL    :  text;
     TODAY     :  STRING8;
     LINE      :  packed array[ 1..62 ] of char;
     NAM0,NAM1,NAM2 :  VARNAME;
     MNAME     :  SCISTRING;
     IER       :  boolean;
     N,I,J     :  integer;

begin (* SAVEASM *)
  SHOWINDEX(IER,SD);  GETFNAME(FNAME);  UPDATEIX(FNAME,false,IER);
  if IER then WAIT(BLOCK) else
  begin
    MNAME[0]:=chr(MAXLENFIL);
    for I:=1 to MAXLENFIL do MNAME[I]:=FNAME[I];
    for I:=1 to 8 do MACHNAME[I]:=FNAME[I+16];
    SET$ACNM(MACHFIL,MNAME);  rewrite(MACHFIL);
    DATE(TODAY);  writeln(MACHFIL,S1,BLANK,TODAY);  writeln(MACHFIL);
    writeln(MACHFIL,S2,MACHNAME);  writeln(MACHFIL,S3,MXSTNUM+1:1);
    with ASMVAR do
    begin
      writeln(MACHFIL,S4,VNUMIN:1);  writeln(MACHFIL,S5,VNUMOUT:1);
      writeln(MACHFIL);  writeln(MACHFIL,S6);  writeln(MACHFIL);
      writeln(MACHFIL,S7);  for I:=1 to 62 do LINE[I]:=' ';
    end
  end

```

```

for I:=1 to MAXNUMVAR do with ALL[I] do
if VNAME <> BLANK then
begin
  LINE[8]:=chr(I div 10 + ord('0'));
  LINE[9]:=chr(I mod 10 + ord('0'));
  for J:=1 to 8 do
  begin
    LINE[J+20]:=VNAME[J]; LINE[J+39]:=LEVTABLE[VACTIVE,J];
    LINE[J+53]:=TYPETABLE[VUSE,J]
  end;
  writeln(MACHFIL,LINE)
end;
if MXSTNUM <> 0 then
begin
  writeln(MACHFIL); writeln(MACHFIL,S8);
  for I:=0 to MXSTNUM do if STATESSET[I] <> nil then
  with STATESSET[I]@,DECS@ do
  if (BRANCH <> NON) or (NEXT0 <> nil) then
  begin
    writeln(MACHFIL); writeln(MACHFIL,S14);
    writeln(MACHFIL,S9,NUMBER:3);
    writeln(MACHFIL,S10,NAME); N:=0;
    for J:=1 to MAXNUMVAR do if VARLIST[J] = 1 then N:=N+1;
    writeln(MACHFIL,S5,N:3); write(MACHFIL,S11);
    for J:=1 to MAXNUMVAR do if VARLIST[J] = 1 then
    write(MACHFIL,J:3);
    writeln(MACHFIL);
    case BRANCH of
      NON      : writeln(MACHFIL,S13,'NONE      ');
      STASTA   : writeln(MACHFIL,S13,'SINGLE     ');
      STADDEC  : writeln(MACHFIL,S13,'DUAL SD');
      DECSTA   : writeln(MACHFIL,S13,'DUAL DS');
      DECDEC   : writeln(MACHFIL,S13,'TRIPLE   ');
    end; (* case *)
    writeln(MACHFIL);
    if BRANCH <> NON then NAMO:=ALL[INPVAR].VNAME;
    case BRANCH of
      NON      : writeln(MACHFIL,S12,NEXT0@.NUMBER:3);
      STASTA   : begin
                    writeln(MACHFIL,NAMO,' ',S12);
                    writeln(MACHFIL,ZER,BL5,NEXT1@.NUMBER:1);
                    writeln(MACHFIL,ONE,BL5,NEXT2@.NUMBER:1)
                  end;
      STADDEC  : begin
                    NAM1:=ALL[NEXT4@.INPVAR].VNAME;
                    writeln(MACHFIL,NAMO,' ',NAM1,' ',S12);
                    writeln(MACHFIL,ZER,DC,BL5,NEXT3@.NUMBER:1);
                    writeln(MACHFIL,ONE,ZER,BL5,
                              NEXT4@.NEXT1@.NUMBER:1);
                    writeln(MACHFIL,ONE,ONE,BL5,
                              NEXT4@.NEXT2@.NUMBER:1)
                  end;
    end;
  end;
end;

```

```

DECSTA : begin
    NAM1:=ALL[NEXT5@.INPVAR].VNAME;
    writeln(MACHFIL,NAM0,' ',NAM1,' ',S12);
    writeln(MACHFIL,ZER,ZER,BL5,
            NEXT5@.NEXT1@.NUMBER:1);
    writeln(MACHFIL,ZER,ONE,BL5,
            NEXT5@.NEXT2@.NUMBER:1);
    writeln(MACHFIL,ONE,DC,BL5,NEXT6@.NUMBER:1)
end;
DECDEC : begin
    NAM1:=ALL[NEXT7@.INPVAR].VNAME;
    NAM2:=ALL[NEXT8@.INPVAR].VNAME;
    write(MACHFIL,NAM0,' ',NAM1,' ');
    if NAM1 <> NAM2 then write(MACHFIL,NAM2,' ');
    writeln(MACHFIL,S12); write(MACHFIL,ZER,ZER);
    if NAM1 <> NAM2 then write(MACHFIL,DC);
    writeln(MACHFIL,BL5,NEXT7@.NEXT1@.NUMBER:1);
    write(MACHFIL,ZER,ONE);
    if NAM1 <> NAM2 then write(MACHFIL,DC);
    writeln(MACHFIL,BL5,NEXT7@.NEXT2@.NUMBER:1);
    write(MACHFIL,ONE);
    if NAM1 <> NAM2 then write(MACHFIL,DC);
    writeln(MACHFIL,ZER,BL5,NEXT8@.NEXT1@.NUMBER:1);
    write(MACHFIL,ONE);
    if NAM1 <> NAM2 then write(MACHFIL,DC);
    writeln(MACHFIL,ONE,BL5,NEXT8@.NEXT2@.NUMBER:1)
    end
end; (* case *)
writeln(MACHFIL,S14)
end
end
end;
close(MACHFIL)
end
end; (* SAVEASM *)

```

```

procedure READASM;
(*****
* PROCEDURE READASM;                                1.4.5 *
* PURPOSE   : This procedure reads a machine from a disk file created *
*             by SAVEASM.                                     *
*             --PROCEDURES CALLED--                         *
* INTERNAL   : SHOWINDEX, GETFNAME, INITIALIZE, INITPTR.    *
* EXTERNAL   : WAIT, ERRORMSG, SET$ACNM.                   *
*****)

var  FNAME,DUMY      :  FILENAME;
     NAME1,NAME2,NAME3 :  STRING8;
     MNAME           :  SCISTRING;
     IER             :  boolean;
     MACHFIL         :  text;
     I,J,N,NUM       :  integer;
     S               :  SPTR;
     D               :  DPTR;
     DUMY20          :  packed array[ 1..20 ] of char;
     LINE            :  packed array[ 1..62 ] of char;

begin (* READASM *)
  SHOWINDEX(IER,RD);
  if IER then WAIT(BLOCK) else
  begin
    GETFNAME(FNAME); reset(MACHINX);
    while not eof(MACHINX) do
    begin
      readln(MACHINX,DUMY);  IER:=IER or DUMY = FNAME
    end;
    close(MACHINX);
    if not IER then ERRORMSG(BLOCK,6,FNAME,true) else
    begin
      for I:=1 to 8 do MACHNAME[I]:=FNAME[I+16];
      MNAME[0]:=chr(MAXLENFIL);
      for I:=1 to MAXLENFIL do MNAME[I]:=FNAME[I];
      SET$ACNM(MACHFIL,MNAME); reset(MACHFIL); INITIALIZE(false);
      for I:=1 to 3 do readln(MACHFIL);
      readln(MACHFIL,DUMY20,N);  MXSTNUM:=N-1;
      with ASMVAR do
      begin
        readln(MACHFIL,DUMY20,VNUMIN);  readln(MACHFIL,DUMY20,VNUMOUT);
        for I:=1 to 4 do readln(MACHFIL);
        for I:=1 to VNUMIN+VNUMOUT do
        begin
          readln(MACHFIL,LINE);
          N:=10*(ord(LINE[8]-ord('0'))+ord(LINE[9])-ord('0'));

```

```

with ALL[N] do
begin
  for J:=1 to 8 do
  begin
    VNAME[J]:=LINE[J+20];  NAME1[J]:=LINE[J+39];
    NAME2[J]:=LINE[J+53]
  end;
  if LEVTABLE[HIGH] = NAME1 then VACTIVE:=HIGH
  else VACTIVE:=LOW;
  if TYPETABLE[INPV] = NAME2 then VUSE:=INPV
  else VUSE:=OUTV
end
end
end;
if MXSTNUM <> 0 then
begin
  readln(MACHFIL);  readln(MACHFIL);
  while not eof(MACHFIL) do
  begin
    readln(MACHFIL);  readln(MACHFIL);  readln(MACHFIL,DUMY20,N);
    INITPTR(STATESET[N],D,false,N);
    with STATESET[N]@,DECS@ do
    begin
      readln(MACHFIL,DUMY20,NAME);
      readln(MACHFIL,DUMY20,NUM);  read(MACHFIL,DUMY20);
      if NUM <> 0 then for I:=1 to NUM do
      begin
        read(MACHFIL,N);  VARLIST[N]:=1
      end;
      readln(MACHFIL);  readln(MACHFIL,DUMY20,NAME1);
      readln(MACHFIL);
      if NAME1 = 'NONE'      ' then BRANCH:=NON;
      if NAME1 = 'SINGLE'    ' then BRANCH:=STASTA;
      if NAME1 = 'DUAL SD'  ' then BRANCH:=STADEC;
      if NAME1 = 'DUAL DS'  ' then BRANCH:=DECSTA;
      if NAME1 = 'TRIPLE'   ' then BRANCH:=DECDEC;
      case BRANCH of
        NON      : begin
                      readln(MACHFIL,DUMY20,NUM);
                      INITPTR(STATESET[ NUM],D,false,NUM);
                      NEXT0:=STATESET[ NUM]
                    end;
        STASTA   : begin
                      readln(MACHFIL,NAME1);
                      for I:=1 to MAXNUMVAR do with ASMVVAR.ALL[I] do
                        if NAME1 = VNAME then INPVAR:=I;
                      for J:=1 to 2 do

```

```

begin
  readln(MACHFIL,I,NUM);
  INITPTR(STATESET[NUM],D,false,NUM);
  if J = 1 then NEXT1:=STATESET[NUM] else
  NEXT2:=STATESET[NUM]
end
end;
STADec,DECSTA
: begin
  INITPTR(S,D,true,0);
  readln(MACHFIL,NAME1,CH,NAME2);
  for I:=1 to MAXNUMVAR do with ASMVAR.ALL[I] do
  begin
    if NAME1 = VNAME then INPVAR:=I;
    if NAME2 = VNAME then if BRANCH=STADec then
    begin
      NEXT4:=D; NEXT4@.INPVAR:=I
    end else
    begin
      NEXST5:=D; NEXT5@.INPVAR:=I
    end
  end;
  for J:=1 to 3 do
  begin
    readln(MACHFIL,DUMY20,NUM);
    INITPTR(STATESET[NUM],D,false,NUM);
    if BRANCH = STADec then
    case J of
      1 : NEXT3:=STATESET[NUM];
      2 : NEXT4@.NEXT1:=STATESET[NUM];
      3 : NEXT4@.NEXT2:=STATESET[NUM]
    end else
    case J of
      1 : NEXT5@.NEXT1:=STATESET[NUM];
      2 : NEXT5@.NEXT2:=STATESET[NUM];
      3 : NEXT6:=STATESET[NUM]
    end
  end
end
end;
DECDEC : begin
  readln(MACHINX,NAME1,CH,NAME2,CH,NAME3);
  INITPTR(S,NEXT7,true,0);
  INITPTR(S,NEXT8,true,0);
  for I:=1 to MAXNUMVAR do with ASMVAR.ALL[I] do
  begin
    if NAME1 = VNAME then INPVAR:=I;
    if NAME2 = VNAME then NEXT7@.INPVAR:=I;
    if NAME3 = VNAME then NEXT8@.INPVAR:=I
  end;
end;

```

```

if NEXT8@.INPVAR = 0 then
NEXT8@.INPVAR:=NEXT7@.INPVAR;
for J:=1 to 4 do
begin
  readln(MACHFIL,DUMY20,NUM);
  INITPTR(STATESET[NUM],D,false,NUM);
  case J of
    1 : NEXT7@.NEXT1:=STATESET[NUM];
    2 : NEXT7@.NEXT2:=STATESET[NUM];
    3 : NEXT8@.NEXT1:=STATESET[NUM];
    4 : NEXT8@.NEXT2:=STATESET[NUM]
  end (* case *)
end
end; (* case *)
readln(MACHFIL)
end
end
end;
close(MACHFIL)
end
end
end; (* READASM *)

```

```

procedure DELASM;
(*****
*PROCEDURE DELASM;                                     1.4.6 *
* PURPOSE   : This procedure deletes an ASM from memory.      *
* INPUT     : User Machine Name.                               *
*           : --PROCEDURES CALLED--                             *
* INTERNAL  : SHOWINDEX, GETFNAME, UPDATEIX.                   *
* EXTERNAL  : WAIT, ERRORMSG, DISPLAY, ACCEPT, DELFILE.        *
*****)

var  FNAME,DUMY :  FILENAME;
     PATHNAME   :  SCISTRING;
     IER        :  boolean;
     USERIN    :  STRING2;
     I          :  integer;

begin (* DELASM *)
  SHOWINDEX(IER,DD);  PATHNAME[0]:=chr(MAXLENFIL);
  if IER then WAIT(BLOCK) else
  begin
    USERIN:='N ';  GETFNAME(FNAME);  reset(MACHINX);
    while not eof(MACHINX) do
      begin
        readln(MACHINX,DUMY);  IER:=IER or FNAME = DUMY
      end;
    close(MACHINX);
    if not IER then ERRORMSG(BLOCK,6,FNAME,true) else
    begin
      repeat
        DISPLAY(BLOCK,MAXNUMACH+7,1,'Are You Sure (Y or N) ?  N',25);
        ACCEPT(BLOCK,MAXNUMACH+7,25,USERIN,1,ENDMES)
      until USERIN[1] in ['Y','N'];
      if USERIN[1] = 'Y' then
        begin
          for I:=1 to MAXLENFIL do PATHNAME[I]:=FNAME[I];
          DELFILE(PATHNAME);  UPDATEIX(FNAME,true,IER);  MACHNAME:=BLANK
        end
      end
    end
  end
end;  (* DELASM *)

```

```

procedure VARASIGN;
(*****
* PROCEDURE VARASIGN;                                1.5.0 *
* PURPOSE : This procedure maintains the variable table contained *
*           in ASMVAR according to the user's response.          *
* INPUT    : User commands.                                     *
* OUTPUT   : ASMVAR -- Contains the variable table.             *
*           --PROCEDURES CALLED--                               *
* INTERNAL : VARIABLE, MODVARS, HELP.                           *
* EXTERNAL : MENU, GETCMD.                                       *
*****)

var CHOSEN : CMD;
    CHOICES : CSET;
    DONE : boolean;

procedure VARIABLE; FORWARD;
procedure MODVARS(CH:CMD); FORWARD;

begin (* VARASIGN *)
  DONE:=false; CHOICES:=[IV,MV,VT,RM,HF];
  repeat
    MENU(BLOCK,VA); GETCMD(BLOCK,CHOICES,CHOSEN);
    case CHOSEN of
      IV,MV : if (CHOSEN = IV) and (ASMVAR.VNUMIN+ASMVAR.VNUMOUT =
        MAXNUMVAR) then ERRORMSG(BLOCK,7,PREFIX,true) else
        if NOVARS and (CHOSEN = MV) then
          ERRORMSG(BLOCK,4,PREFIX,true) else MODVARS(CHOSEN);
      VT : if NOVARS then ERRORMSG(BLOCK,4,PREFIX,true) else
        VARIABLE;
      HF : HELP;
      RM : DONE:=true
    end (* case *)
  until DONE
end; (* VARASIGN *)

```

```

procedure VARIABLE;
(*****
* PROCEDURE VARIABLE;                                     1.5.1 *
* PURPOSE : This procedure displays the contents of the variable *
*           ASMVAR in tabular form.                             *
* INPUT    : Global variable ASMVAR.                           *
* OUTPUT   : Variable table displayed on the VDT.               *
*           --PROCEDURES CALLED--                               *
* INTERNAL : NONE.                                              *
* EXTERNAL : CLEARSCREEN, DISPLAY, GRAPHICS, IDISPLAY, WAIT, SCROLL. *
*****)

const HEADER =
  'Variable Number   Variable Name   Active Level   Input/Output';

var  LIN : BYTE;
     VARI : VARNUM;

begin (* VARIABLE *)
  CLEARSCREEN(BLOCK);  DISPLAY(BLOCK,1,8,HEADER,80);
  GRAPHICS(BLOCK,'V',1,1);  GRAPHICS(BLOCK,'H',2,1);  LIN:=3;
  for VARI:=1 to MAXNUMVAR do with ASMVAR.ALL[VARI] do
    if VNAME <> BLANK then
      begin
        IDISPLAY(BLOCK,LIN,15,VARI);
        DISPLAY(BLOCK,LIN,28,VNAME,8);
        DISPLAY(BLOCK,LIN,48,LEVTABLE[VACTIVE],8);
        DISPLAY(BLOCK,LIN,60,TYPETABLE[VUSE],8);
        GRAPHICS(BLOCK,'V',LIN,1);
        if LIN > 20 then
          begin
            DISPLAY(BLOCK,24,32,'Press Return to Continue:',80);
            WAIT(BLOCK);  SCROLL(BLOCK,3,21,10);  LIN:=LIN-10
          END ELSE LIN:=LIN+1
        end;
        GRAPHICS(BLOCK,'H',LIN,1);
        DISPLAY(BLOCK,22,10,'Total Inputs:',13);
        IDISPLAY(BLOCK,22,24,ASMVAR.VNUMIN);
        DISPLAY(BLOCK,22,31,'Total Outputs:',14);
        IDISPLAY(BLOCK,22,46,ASMVAR.VNUMOUT);  WAIT(BLOCK)
      end; (* VARIABLE *)
    end;
  end;
end; (* VARIABLE *)

```

```

procedure MODVARS(* CH:CMD *);
(*****
* PROCEDURE MODVARS;                                     1.5.2 *
* PURPOSE : This procedure creates or modifies the variable table. *
*           By specifying the variable name as a blank, the *
*           variable is deleted from the table. *
* INPUT : CH -- Determines insert or modify mode. *
* OUTPUT : ASMVAR -- Modified variable table. *
*           --PROCEDURES CALLED-- *
* INTERNAL : NONE. *
* EXTERNAL : DISPLAY, IACCEPT, MENU, ACCEPT, IDISPLAY. *
*****)

var NAME : VARNAME;
    DONE : boolean;
    CNTR : VARNUM0;
    LEV : VARLEV;
    TYP : VARTYPE;
    CON : STRING2;
    REQ : integer;

begin (* MODVARS *)
    CNTR:=0; DONE:=false; CON:='Y '; MENU(BLOCK,CH);
    repeat
        CNTR:=CNTR+1;
        with ASMVAR do
            begin
                if CH = IV then while (ALL[CNTR].VNAME <> BLANK)
                    and (CNTR <> MAXNUMVAR) do CNTR:=CNTR+1 else
                    begin
                        repeat
                            IDISPLAY(BLOCK,7,37,CNTR); IACCEPT(BLOCK,7,37,REQ)
                        until REQ <= MAXNUMVAR;
                        CNTR:=REQ
                    end;
                    IDISPLAY(BLOCK,7,37,CNTR);
                    with ALL[CNTR] do
                        begin
                            DISPLAY(BLOCK,8,35,VNAME,MAXLENVAR);
                            DISPLAY(BLOCK,9,50,LEVTABLE[VACTIVE],8);
                            DISPLAY(BLOCK,10,53,TYPETABLE[VUSE],8);
                            DISPLAY(BLOCK,12,48,CON,1);
                            ACCEPT(BLOCK,8,35,VNAME,MAXLENVAR,ENDMES);
                            repeat
                                ACCEPT(BLOCK,9,50,NAME,8,ENDMES);
                                for LEV:=HIGH to LOW do if LEVTABLE[LEV] = NAME then
                                    begin
                                        VACTIVE:=LEV; DONE:=true
                                    end
                                until DONE;
                                DONE:=false;

```

```

repeat
  ACCEPT(BLOCK,10,53,NAME,8,ENDMES);
  for TYP:=INPV to OUTV do if TYPETABLE[TYP] = NAME then
    begin
      VUSE:=TYP;  DONE:=true
    end
  until DONE;
  ACCEPT(BLOCK,12,48,CON,1,ENDMES)
end
end
until CON[1] = 'N';
with ASMVAR do
begin
  VNUMIN:=0;  VNUMOUT:=0;
  for CNTR:=1 to MAXNUMVAR do with ALL[CNTR] do
    if VNAME <> BLANK then
      if VUSE = INPV then VNUMIN:=VNUMIN+1 else VNUMOUT:=VNUMOUT+1;
      if VNUMIN + VNUMOUT <> 0 then NOVARS:=false
    end
  end
end;  (* MODVARS *)

```

```

procedure ASMCHART;
(*****
* PROCEDURE ASMCHART;                                1.6.0 *
* PURPOSE : This procedure constructs the machine according to the *
*           user commands. The user may create states or modify *
*           existing states. The user may also delete states by *
*           specifying a smaller number of states than the default *
*           value (if any). In this case, the machine is scanned *
*           and references to the deleted state(s) are updated. *
* INPUTS : User commands. *
* OUTPUT : Set of states contained in STATESSET. *
*           --PROCEDURES CALLED-- *
* INTERNAL : MODSTATES, HELP. *
* EXTERNAL : MENU, GETCMD, ERRORMSG. *
*****)

```

```

var CHOICES : CSET;
    CHOSEN  : CMD;
    DONE    : boolean;

```

```

procedure DRAWVARS(CH:VARTYPE); FORWARD;
procedure GETSTR(L,C:BYTE; var INP:VARNUM0; CH:char); FORWARD;
procedure GETSPTR(L,C:BYTE; NS:SPTR); FORWARD;
procedure INVARS(var L,C:BYTE; var DEC:DECISION); FORWARD;
procedure OTVARS(var L,C:BYTE; var LW:LONGWORD); FORWARD;
procedure DELSTATE(N:integer); FORWARD;
procedure MODSTATES(CH:CMD); FORWARD;

```

```

begin (* ASMCHART *)
  DONE:=false; CHOICES:=[AS,MS,RM,HF];
  repeat
    MENU(BLOCK,AC); GETCMD(BLOCK,CHOICES,CHOSEN);
    case CHOSEN of
      AS : if NOVARS then ERRORMSG(BLOCK,4,PREFIX,true)
            else MODSTATES(CHOSEN);
      MS : if NOVARS then ERRORMSG(BLOCK,4,PREFIX,true)
            else if MXSTNUM = 0 then ERRORMSG(BLOCK,9,PREFIX,true)
            else MODSTATES(CHOSEN);
      HF : HELP;
      RM : DONE:=true;
    end (* case *)
  until DONE
end; (* ASMCHART *)

```

```

PROCEDURE DRAWVARS(* CH:VARTYPE *);
(*****
* PROCEDURE DRAWVARS;                                     1.6.1 *
* PURPOSE   : This procedure displays the variable names on the VDT. *
* INPUT     : CH -- Requests input or output type variables.      *
* OUTPUT    : Variable list on the screen.                         *
*           : --PROCEDURES CALLED--                                *
* INTERNAL  : NONE.                                                *
* EXTERNAL  : DISPLAY.                                             *
*****)

var I,LIN,COL : BYTE;

begin (* DRAWVARS *)
  LIN:=3; COL:=63; DISPLAY(BLOCK,1,COL+4,TYPETABLE[CH],8);
  DISPLAY(BLOCK,2,COL,'-----',16);
  for I:=1 to MAXNUMVAR do with ASMVAR,ALL[I] do
    if (VNAME <> BLANK) and (VUSE = CH) then
      begin
        DISPLAY(BLOCK,LIN,COL,VNAME,MAXLENVAR); LIN:=LIN+1;
        if LIN > 24 then
          begin
            LIN:=3; COL:=COL+9
          end
        end;
      while LIN < 25 do
        begin
          DISPLAY(BLOCK,LIN,COL,BLANK,8); LIN:=LIN+1
        end;
        if COL = 63 then for I:=3 to 24 do DISPLAY(BLOCK,I,COL+9,BLANK,8)
      end; (* DRAWVARS *)

```

```

procedure GETSTR(* L,C:BYTE; var INP:VARNUM0; CH:char *);
(*****
* PROCEDURE GETSTR;                                     1.6.2 *
* PURPOSE : This procedure determines input variables to be used *
*           in the decision boxes. If the value of INP is not zero *
*           a default value is displayed.                      *
* INPUTS  : L,C -- Screen line and column for input.          *
*           CH -- Character used to specify which type of decision *
*           ('D' or 'E') to draw.                             *
* OUTPUT  : INP -- Variable number representing the user response. *
*           --PROCEDURES CALLED--                             *
* INTERNAL : DRAWVARS.                                         *
* EXTERNAL : GRAPHICS, DISPLAY, ACCEPT, ERRORMSG.             *
*****)

const ST1 = '...Enter Input Variable...';

var GOOD      : boolean;
    HALF1,HALF2 : STRING4;
    NAMEV,DEF   : VARNAME;
    I           : integer;

begin (* GETSTR *)
    GRAPHICS(BLOCK,CH,L,C); DRAWVARS(INPV);
    if INP = 0 then DEF:=BLANK else DEF:=ASMVAR.ALL[INP].VNAME;
    repeat
        DISPLAY(BLOCK,24,1,ST1,40); GOOD:=false;
        for I:=1 to 8 do if I < 5 then HALF1[I]:=DEF[I]
            else HALF2[I-4]:=DEF[I];
        DISPLAY(BLOCK,L+2,C-1,HALF1,4);
        DISPLAY(BLOCK,L+3,C-1,HALF2,4);
        ACCEPT(BLOCK,L+2,C-1,HALF1,4,ENDMES);
        ACCEPT(BLOCK,L+3,C-1,HALF2,4,ENDMES);
        for I:=1 to 8 do if I < 5 then NAMEV[I]:=HALF1[I]
            else NAMEV[I]:=HALF2[I-4];
        for I:=1 to MAXNUMVAR do with ASMVAR,ALL[I] do if not GOOD then
            begin
                GOOD:=(NAMEV = VNAME) and (NAMEV <> BLANK) and (VUSE = INPV);
                if GOOD then INP:=I
            end;
        if not GOOD then ERROEMSG(BLOCK,12,PREFIX,true)
    until GOOD
end; (* GETSTR *)

```

```

procedure GETSPTR(* L,C:BYTE; var NS:SPTR *);
(*****
* PROCEDURE GETSPTR;                                     1.6.3 *
* PURPOSE   : This procedure returns a state pointer corresponding to *
*             the users input request.                             *
* INPUTS    : L,C -- Screen line and column coordinates.          *
* OUTPUT    : NS -- Next state pointer.                            *
*             --PROCEDURES CALLED--                                *
* INTERNAL  : INITPTR.                                             *
* EXTERNAL  : GRAPHICS, DISPLAY, IDISPLAY, IACCEPT, ERRORMSG.     *
*****)

const ST1 = '...Enter Next State...' ;

var   N : integer;
      D : DPTR;

begin (* GETSPTR *)
  GRAPHICS(BLOCK,'X',L,C);
  repeat
    DISPLAY(BLOCK,24,1,ST1,40);
    if NS <> nil then IDISPLAY(BLOCK,L,C,NS@.NUMBER) else
      DISPLAY(BLOCK,L,C,' ',4);
    IACCEPT(BLOCK,L,C,N);
    if N > MXSTNUM then ERRORMSG(BLOCK,10,PREFIX,true)
  until N <= MXSTNUM;
  INITPTR(STATESET[N],D,false,N);  NS:=STATESET[N]
end; (* GETSPTR *)

```

```

procedure INVARS(* var L,C:BYTE; var DEC:DECISION *);
(*****
* PROCEDURE INVARS;                                     1.6.4 *
* PURPOSE : This procedure accepts input to define the decision *
*           structure. Defaults are displayed if the DECISION *
*           record already exists.                             *
* INPUT    : User commands.                                   *
* OUTPUTS  : L,C -- Screen line and column coordinates.      *
*           DEC -- Completed decision structure.              *
*           --PROCEDURES CALLED--                             *
* INTERNAL : GETSTR, DRAWVARS, GETSPTR, INITPTR.              *
* EXTERNAL : DISPLAY, ACCEPT, GRAPHICS, SCROLL.               *
*****)

```

```

const ST1 = '...Enter State or Decision (S or D)... ';

```

```

var I,J      : BYTE;
    ZERO,ONE : STRING2;
    DUMY     : SPTR;
    TEMPO,TEMP1 : DPTR;

```

```

begin (* INVARS *)
  with DEC,ASMVAR do
    begin
      ZERO:=' '; ONE:=' ';
      if BRANCH <> NON then
        begin
          if BRANCH in [STASTA,STADDEC] then ZERO:='S ' else ZERO:='D ';
          if BRANCH in [STASTA,DECSTA] then ONE:='S ' else ONE:='D ';
        end;
      GETSTR(L,C,INPVAR,'D'); DISPLAY(BLOCK,24,1,ST1,40);
      repeat
        DISPLAY(BLOCK,L+6,C-10,ZERO,1);
        ACCEPT(BLOCK,L+6,C-10,ZERO,1,ENDMES)
      until ZERO[1] in ['S','D'];
      repeat
        DISPLAY(BLOCK,L+6,C+10,ONE,1);
        ACCEPT(BLOCK,L+6,C+10,ONE,1,ENDMES)
      until ONE[1] in ['S','D'];
      if (L > 8) and ((ZERO[1] = 'D') or (ONE[1] = 'D')) then
        begin
          I:=L-9; SCROLL(BLOCK,1,23,I); L:=L-I; DRAWVARS(INPV)
        end;
      if (ZERO = 'S ') and (ONE = 'S ') then
        begin
          if BRANCH <> STASTA then

```

```

begin
  case BRANCH of
    NON      :  ;
    STADEC   :  dispose(NEXT4);
    DECSTA   :  dispose(NEXT5);
    DECDEC   :  begin
                  dispose(NEXT7);  dispose(NEXT8)
                end
  end;  (* case *)
  BRANCH = STASTA;  NEXT1:=nil;  NEXT2:=nil
end;
GETSPTR(L+7,C-11,NEXT1);  GETSPTR(L+7,C+9,NEXT2)
end;
if (ZERO = 'S ') and (ONE = 'D ') then
begin
  if BRANCH <> STADEC then
  begin
    case BRANCH of
      NON,STASTA :  INITPTR(DUMY,TEMP1,true,0);
      DECSTA     :  TEMP1:=NEXT5;
      DECDEC     :  begin
                      TEMP1:=NEXT8;  dispose(NEXT7)
                    end
    end;  (* case *)
    BRANCH = STADEC;  NEXT3:=nil;  NEXT4:=TEMP1
  end;
  GETSTR(L+6,C+10,NEXT4@.INPVAR,'E');  GETSPTR(L+7,C-11,NEXT3);
  GETSPTR(L+13,C+3,NEXT4@.NEXT1);  GETSPTR(L+13,C+15,NEXT4@.NEXT2)
end;
if (ZERO = 'D ') and (ONE = 'S ') then
begin
  if BRANCH <> DECSTA then
  begin
    case BRANCH of
      NON,STASTA :  INITPTR(DUMY,TEMP0,true,0);
      STADEC     :  TEMP0:=NEXT4;
      DECDEC     :  begin
                      TEMP0:=NEXT7;  dispose(NEXT8)
                    end
    end;  (* case *)
    BRANCH:=DECSTA;  NEXT5:=TEMP0;  NEXT6:=nil
  end;
  GETSTR(L+6,C-10,NEXT5@.INPVAR,'E');  GETSPTR(L+7,C+9,NEXT6);
  GETSPTR(L+13,C-17,NEXT5@.NEXT1);  GETSPTR(L+13,C-5,NEXT5@.NEXT2)
end;

```

```

if (ZERO = 'D ') and (ONE = 'D ') then
begin
  if BRANCH <> DECDEC then
  begin
    case BRANCH of
      NON,STASTA : begin
        INITPTR(DUMY,TEMPO,true,0);
        INITPTR(DUMY,TEMP1,true,0)
      end;
      STADEC      : begin
        INITPTR(DUMY,TEMPO,true,0);
        TEMP1:=NEXT4
      end;
      DECSTA      : begin
        TEMPO:=NEXT5;
        INITPTR(DUMY,TEMP1,true,0)
      end
    end;
    (* case *)
    BRANCH = DECDEC; NEXT7:=TEMPO; NEXT8:=TEMP1
  end;
  GETSTR(L+6,C-10,NEXT7@.INPVAR,'E');
  GETSPTR(L+13,C-17,NEXT7@.NEXT1);
  GETSPTR(L+13,C-5,NEXT7@.NEXT2);
  GETSTR(L+6,C+10,NEXT8@.INPVAR,'E');
  GETSPTR(L+13,C+3,NEXT8@.NEXT1);
  GETSPTR(L+13,C+15,NEXT8@.NEXT2)
end
end
end; (* INVARS *)

```

```

procedure OTVARS(* var L,C:BYTE; var LW:LONGWORD *);
(*****
* PROCEDURE OTVARS;                                     1.6.5 *
* PURPOSE   : This procedure accepts output variables for the state *
*             boxes.                                           *
* INPUTS    : L,C -- Initial screen line and column.          *
*             LW  -- List of existing active outputs.          *
* OUTPUTS   : L,C -- Final screen line and column.            *
*             LW  -- List of current active outputs.           *
*             --PROCEDURES CALLED--                             *
* INTERNAL  : DRAWVARS.                                         *
* EXTERNAL  : GRAPHICS, DISPLAY, ACCEPT, ERRORMSG.             *
*****)

```

```

const ST ='...Enter Active Variables...';

```

```

var   NVAR,I,J : integer;
      GOOD      : boolean;
      NAMEV     : VARNAME;
      VSET      : array[ VARNUM ] of VARNUM;

```

```

begin (* OTVARS *)
  DRAWVARS(OUTV); L:=2; C:=23; GRAPHICS(BLOCK,'T',L-1,C-1);
  DISPLAY(BLOCK,24,1,ST,40); NVAR:=0;
  with ASMVAR do
    begin
      for I:=1 to MAXNUMVAR do with ALL[I] do if LW[I] = 1 then
        begin
          DISPLAY(BLOCK,L,C,VNAME,MAXLENVAR); NVAR:=NVAR+1;
          C:=C+9; VSET[NVAR]:=I;
          if C > 41 then
            begin
              C:=C-27; GRAPHICS(BLOCK,'S',L,C-1); L:=L+1
            end
          end;
          L:=2; C:=23; NAMEV:='*****';
          for I:=1 to NVAR do
            begin
              repeat
                ACCEPT(BLOCK,L,C,NAMEV,MAXLENVAR,ENDMES); GOOD:=false;
                if NAMEV = BLANK then
                  begin
                    LW[VSET[I]]:=0; C:=C+9; GOOD:=true
                  end;
                if NAMEV = ALL[VSET[I]].VNAME then
                  begin
                    C:=C+9; GOOD:=true
                  end;
                if not GOOD then for J:=1 to MAXNUMVAR do with ALL[J] do
                  if (NAMEV = VNAME) and (VUSE = OUTV) and (NAMEV <> BLANK) then

```

```

begin
  LW[J]:=1; LW[VSET[I]]:=0; C:=C+9; GOOD:=true
end
until GOOD;
if C > 41 then
begin
  C:=C-27; L:=L+1
end
end;
if NAMEV <> BLANK then
repeat
  repeat
    DISPLAY(BLOCK,L,C,BLANK,8); GOOD:=false;
    ACCEPT(BLOCK,L,C,NAMEV,MAXLENVAR,ENDMES);
    for I:=1 to MAXNUMVAR do with ALL[I] do
      if (NAMEV = VNAME) and (VUSE = OUTV) and (LW[I] = 0)
        and (VNAME <> BLANK) then
        begin
          GOOD:=true; LW[I]:=1; C:=C+9
        end
      until GOOD or (NAMEV = BLANK);
      if C > 41 and (NAMEV <> BLANK) then
        begin
          C:=C-27; GRAPHICS(BLOCK,'S',L,C-1); L:=L+1
        end
      until NAMEV = BLANK
    end;
    GRAPHICS(BLOCK,'S',L,22); GRAPHICS(BLOCK,'B',L+1,22);
    L:=L+4; C:=34
  end; (* OTVARS *)

```

```

procedure DELSTATE(* N:integer *);
(*****
* PROCEDURE DELSTATE;                                1.6.6 *
* PURPOSE   : This procedure deletes states from the machine in *
*             memory. The state numbers are shifted after deletion. *
* INPUT      : N -- The number of states to delete.                *
*             --PROCEDURES CALLED--                                *
* INTERNAL   : GETSPTR.                                             *
* EXTERNAL   : CLEARSCREEN, DISPLAY, ACCEPT, IDISPLAY, IACCEPT,    *
*             ERRORMSG.                                             *
*****)

```

```

const S1 = 'Unresolved References to State      ';
      S2 = 'State Number      State Name: ';
      S3 = 'Variable          is      ';
      S4 = 'Next State ---->';

```

```

var   I,J,K,NUM : integer;
      BAD       : set of STATENUM;
      STAT      : packed array[ 1..13 ] of boolean;
      NAMEV     : VARNAME;
      GOOD      : boolean;
      TEMP      : SPTR;
      S         : STATENUM;

```

```

begin (* DELSTATE *)
  BAD:=[];
  for I:=1 to N do
    begin
      CLEARSCREEN(BLOCK);
      repeat
        DISPLAY(BLOCK,24,1,'Enter State to be Deleted (0 to      ):',37);
        IDISPLAY(BLOCK,24,33,MXSTNUM); IACCEPT(BLOCK,24,39,NUM);
        GOOD:=(NUM in [0..MXSTNUM]) and not (NUM in BAD);
        if not GOOD then ERRORMSG(BLOCK,10,PREFIX,true) else S:=NUM
      until GOOD;
      BAD:=BAD+[S]; DISPLAY(BLOCK,5,8,S1,40); IDISPLAY(BLOCK,5,39,NUM);
      DISPLAY(BLOCK,8,1,S2,40); DISPLAY(BLOCK,20,1,S4,40);
      for J:=0 to MXSTNUM do
        if (STATESET[J] <> nil) and not (J in BAD) then
          with STATESET[J]@,DECS@ do
            begin
              STAT[1] :=(BRANCH = NON      ) and (STATESET[ NUM] = NEXT0);
              STAT[2] :=(BRANCH = DECSTA) and (STATESET[ NUM] = NEXT5@.NEXT1);
              STAT[3] :=(BRANCH = STADec) and (STATESET[ NUM] = NEXT4@.NEXT2);
              STAT[4] :=(BRANCH = DECSTA) and (STATESET[ NUM] = NEXT5@.NEXT2);
              STAT[5] :=(BRANCH = DECDEC) and (STATESET[ NUM] = NEXT8@.NEXT2);
              STAT[6] :=(BRANCH = DECDEC) and (STATESET[ NUM] = NEXT7@.NEXT1);
              STAT[7] :=(BRANCH = STADec) and (STATESET[ NUM] = NEXT4@.NEXT1);
              STAT[8] :=(BRANCH = STASTA) and (STATESET[ NUM] = NEXT1);
              STAT[9] :=(BRANCH = STASTA) and (STATESET[ NUM] = NEXT2);
            end
          end
    end

```

```

STAT[10]:= (BRANCH = DECDEC) and (STATESET[NUM] = NEXT7@.NEXT2);
STAT[11]:= (BRANCH = DECSTA) and (STATESET[NUM] = NEXT6);
STAT[12]:= (BRANCH = STADDEC) and (STATESET[NUM] = NEXT3);
STAT[13]:= (BRANCH = DECDEC) and (STATESET[NUM] = NEXT8@.NEXT1);
for K:= 1 to 13 do if STAT[K] then
begin
  DISPLAY(BLOCK,8,30,NAME,20);  IDISPLAY(BLOCK,8,14,NUMBER);
  if K > 1 then with ASMVAR do
  begin
    DISPLAY(BLOCK,10,1,S3,40);  IDISPLAY(BLOCK,10,22,K mod 2);
    DISPLAY(BLOCK,10,10,ALL[INPVAR].VNAME,MAXLENVAR);
    if K in [2,3,4,5,6,7,10,13] then
    begin
      DISPLAY(BLOCK,12,1,S3,40);
      IDISPLAY(BLOCK,12,22,(K div 3) mod 2);
      if K in [3,7] then NAMEV:=ALL[NEXT4@.INPVAR].VNAME;
      if K in [2,4] then NAMEV:=ALL[NEXT5@.INPVAR].VNAME;
      if K in [6,10] then NAMEV:=ALL[NEXT7@.INPVAR].VNAME;
      if K in [5,13] then NAMEV:=ALL[NEXT8@.INPVAR].VNAME;
      DISPLAY(BLOCK,12,10,NAMEV,MAXLENVAR)
    end
  end;
  repeat
    TEMP:=nil;  GETSPTR(20,16,TEMP)
  until not (TEMP@.NUMBER in BAD);
  case K of
    1 : NEXT0:=TEMP;
    2 : NEXT5@.NEXT1:=TEMP;
    3 : NEXT4@.NEXT2:=TEMP;
    4 : NEXT5@.NEXT2:=TEMP;
    5 : NEXT8@.NEXT2:=TEMP;
    6 : NEXT7@.NEXT1:=TEMP;
    7 : NEXT4@.NEXT1:=TEMP;
    8 : NEXT1:=TEMP;
    9 : NEXT2:=TEMP;
    10 : NEXT7@.NEXT2:=TEMP;
    11 : NEXT6:=TEMP;
    12 : NEXT3:=TEMP;
    13 : NEXT8@.NEXT1:=TEMP
  end (* case *)
end
end
end;

```

```

for I in BAD do
begin
  if STATESET[I] <> nil then with STATESET[I]@,DECS@ do
  begin
    case BRANCH of
      NON,STASTA :   ;
      STADDEC    :  dispose(NEXT4);
      DECSTA     :  dispose(NEXT5);
      DECDEC     :  begin
                      dispose(NEXT7);  dispose(NEXT8)
                    end
    end;  (* case *)
    dispose(DECS);  dispose(STATESET[I])
  end;
  for J:=I to MXSTNUM-1 do
  begin
    STATESET[J]:=STATESET[J+1];  STATESET[J]@.NUMBER:=J
  end;
  STATESET[MXSTNUM]:=nil;  MXSTNUM:=MXSTNUM-1
end
end;  (* DELSTATE *)

```

```

procedure MODSTATES(* CH:CMD *);
(*****
* PROCEDURE MODSTATES;                                1.6.7 *
* PURPOSE   : This procedure modifies existing states in the machine. *
* INPUT     : User commands.                                         *
* OUTPUT    : Modified states.                                       *
*           --PROCEDURES CALLED--                                    *
* INTERNAL  : DELSTATE, OTVARS, INVARS, GETSPTR.                    *
* EXTERNAL  : ERRORMSG, CLEARSCREEN, DISPLAY, ACCEPT, IDISPLAY,    *
*           IACCEPT.                                                 *
*****)

const ST1 = '...Enter State Name...';
      ST2 = 'Continue (Y or N) ?';
      ST3 = 'Is There a Decision Block (Y or N) ?';

var PRES,NUM : integer;
    LIN,COL  : BYTE;
    GOOD     : boolean;
    ANS      : STRING2;
    D        : DPTR;

begin (* MODSTATES *)
  CLEARSCREEN(BLOCK); PRES:=0;
  repeat
    DISPLAY(BLOCK,24,1,'Total Number of States:',24);
    IDISPLAY(BLOCK,24,26,MXSTNUM+1); IACCEPT(BLOCK,24,26,NUM);
    GOOD:=(NUM > 1) and (NUM < MAXNUMST + 2);
    if not GOOD then ERRORMSG(BLOCK,10,PREFIX,true)
  until GOOD;
  if NUM < MXSTNUM+1 then DELSTATE(MXSTNUM+1-NUM) else
  begin
    MXSTNUM:=NUM-1;
    repeat
      CLEARSCREEN(BLOCK);
      if CH = AS then
        repeat
          GOOD:=STATASET[PRES] = nil;
          if not GOOD then with STATASET[PRES]@,DECS@ do
            GOOD:=(BRANCH = NON) and (NEXT0 = nil);
          if not GOOD then PRES:=PRES+1
        until GOOD or (PRES = MXSTNUM) else
          repeat
            DISPLAY(BLOCK,24,1,'Enter State Number (0 to ): ',31);
            IDISPLAY(BLOCK,24,26,MXSTNUM); IDISPLAY(BLOCK,24,33,PRES);
            IACCEPT(BLOCK,24,33,NUM); GOOD:=NUM <= MXSTNUM;
            if not GOOD then ERRORMSG(BLOCK,10,PREFIX,true)
            else PRES:=NUM
          until GOOD;
          if not GOOD then ERRORMSG(BLOCK,11,PREFIX,true) else

```

```

begin
  INITPTR(STATESET[PRES],D,false,PRES);
  DISPLAY(BLOCK,1,4,'---State Name---',14);
  DISPLAY(BLOCK,5,1,'State Number:',13);
  DISPLAY(BLOCK,24,1,ST1,40);
  with STATESET[PRES]@,DECS@ do
  begin
    DISPLAY(BLOCK,3,1,NAME,MAXLENST); IDISPLAY(BLOCK,5,15,NUMBER);
    ACCEPT(BLOCK,3,1,NAME,MAXLENST,ENDMES);
    OTVARS(LIN,COL,VARLIST); DISPLAY(BLOCK,24,1,ST3,40);
    repeat
      if BRANCH = NON then ANS:='N ' else ANS:='Y ';
      DISPLAY(BLOCK,24,38,ANS,1);
      ACCEPT(BLOCK,24,38,ANS,1,ENDMES)
    until ANS[1] in ['Y','N'];
    if ANS = 'Y ' and (ASMVAR.VNUMIN > 0) then
      INVARS(LIN,COL,DECS@) else
      begin
        case BRANCH of
          NON,STASTA : ;
          STADEC      : dispose(NEXT4);
          DECSTA      : dispose(NEXT5);
          DECDEC      : begin
                        dispose(NEXT7); dispose(NEXT8)
                      end
        end; (* case *)
        BRANCH:=NON; GETSPTR(LIN+1,COL-1,DECS@.NEXT0)
      end
    end;
    PRES:=PRES+1;
    if PRES > MXSTNUM then ANS:='N ' else ANS:='Y ';
    DISPLAY(BLOCK,24,1,ST2,40);
    repeat
      DISPLAY(BLOCK,24,21,ANS,1); ACCEPT(BLOCK,24,21,ANS,1,ENDMES)
    until ANS[1] in ['Y','N']
  end
  until (ANS = 'N ') or (PRES > MXSTNUM) or not GOOD
end
end; (* MODSTATES *)

```

```

procedure ASMTABLE;
(*****
* PROCEDURE ASMTABLE;                                1.7.0 *
* PURPOSE   : This procedure generates the truth table for a machine *
*             stored in a disk file. The file name is the same as *
*             the machine name except that a dollar sign is inserted *
*             as the second character. *
* INPUT     : Machine file. *
* OUTPUT    : Truth table file. *
*             --PROCEDURES CALLED-- *
* INTERNAL  : FILEIO, BINARY. *
* EXTERNAL  : SET$ACNM, DISPLAY, WAIT, ERRORMSG, OVLY$. *
*****)

```

```

const S1 = 'ASMDSIGN Machine Table Listing';
      S2 = 'Machine Name:      ' ;
      S3 = 'Number of States:  ' ;
      S4 = 'State Name        ' ;
      S5 = 'Active Level      ' ;
      S6 = 'Inputs/PS|NS/Outputs';
      S7 = '                  ' ;
      S8 = '-----' ;

```

```

type TABLENTRY = packed array[ 1..48 ] of char;

```

```

var  MNAME      : SCISTRING;
     NAME1      : STRING8;
     POS,I,J,NBIT,CP,N : integer;
     LINE       : TABLENTRY;
     ENTRY      : array[ 1..4 ] of TABLENTRY;
     TABLE     : text;
     INS,OUTS   : set of VARNUM;

```

```

procedure BINARY(var LIN:TABLENTRY; P,NUM:integer); FORWARD;

```

```

begin (* ASMTABLE *)
  OVLY$(1); FILEIO(RD); INS:=[]; OUTS:=[]; POS:=2;
  if MACHNAME <> BLANK then
    begin
      NAME1:=MACHNAME; NAME1[2]:='$'; MNAME[0]:=chr(MAXLENFIL);
      for I:=1 to MAXLENFIL do if I < 17 then
        MNAME[I]:=PREFIX[I] else MNAME[I]:=NAME1[I-16]; DATE(NAME1);
      SET$ACNM(TABLE,MNAME); rewrite(TABLE);
      writeln(TABLE,S1,BLANK,NAME1); writeln(TABLE);
      writeln(TABLE,S2,MACHNAME);
      writeln(TABLE,S3,MXSTNUM+1:1); writeln(TABLE);
      NBIT:=round(ln(MXSTNUM)/ln(2.0)+0.5);
      with ASMVAR do
        begin
          if VNUMIN > 2 then CP:=VNUMIN+9 else CP:=11;
          for I:=1 to 48 do ENTRY[1,I]:=' '; LINE:=ENTRY[1];

```

```

for I:=2 to CP+VNUMOUT+7 do LINE[I]:='-';
LINE[1]:='+'; LINE[CP]:='+'; ENTRY[1,1]:='|'; ENTRY[1,CP]:='|';
for I:=2 to 4 do ENTRY[I]:=ENTRY[1];
for I:=1 to MAXNUMVAR do with ALL[I] do
if VNAME <> BLANK then if VUSE = INPV then
INS:=INS+[I] else OUTS:=OUTS+[I];
write(TABLE,S7,'|');
for I:=3 to VNUMIN do write(TABLE,' ');
writeln(TABLE,S6); writeln(TABLE,S7,LINE);
for I in INS do
begin
ENTRY[1,POS]:=chr(I div 10 + ord('0'));
ENTRY[2,POS]:=chr(I mod 10 + ord('0')); POS:=POS+1
end;
for I:=1 to 6 do
begin
ENTRY[1,CP-7+I]:='S'; ENTRY[1,CP+I]:='S';
ENTRY[2,CP-7+I]:=chr(6 - I + ord('0'));
ENTRY[2,CP+I]:=ENTRY[2,CP-7+I]
end;
POS:=CP+8;
for I in OUTS do
begin
ENTRY[1,POS]:=chr(I div 10 + ord('0'));
ENTRY[2,POS]:=chr(I mod 10 + ord('0')); POS:=POS+1
end;
writeln(TABLE,S7,ENTRY[1]); writeln(TABLE,S4,ENTRY[2]);
writeln(TABLE,S8,LINE);
for I:=1 to 3 do ENTRY[I]:=ENTRY[4];
for J:=0 to MXSTNUM do if STATESET[J] <> nil then
with STATESET[J]@,DECS@ do
if (BRANCH <> NON) or (NEXT0 <> nil) then
begin
BINARY(ENTRY[1],CP-NBIT,NUMBER); POS:=CP+8;
for I in OUTS do with ALL[I] do
begin
if ((VACTIVE = HIGH) and (VARLIST[I] = 1)) or
((VACTIVE = LOW) and (VARLIST[I] = 0))
then ENTRY[1,POS]:='1' else ENTRY[1,POS]:='0';
POS:=POS+1
end;
end;
for I:=1 to VNUMIN do ENTRY[1,I+1]:='-'; POS:=2;
for I:=2 to 4 do ENTRY[I]:=ENTRY[1];
case BRANCH of
NON : begin
BINARY(ENTRY[1],CP+7-NBIT,NEXT0@.NUMBER); N:=1
end;
STASTA : begin
BINARY(ENTRY[1],CP+7-NBIT,NEXT1@.NUMBER); N:=2;
BINARY(ENTRY[2],CP+7-NBIT,NEXT2@.NUMBER);
for I in INS do with ALL[I] do

```

```

begin
  if INPVAR = I then if VACTIVE = HIGH then
    begin
      ENTRY[1,POS] := '0'; ENTRY[2,POS] := '1'
    end else
      begin
        ENTRY[1,POS] := '1'; ENTRY[2,POS] := '0'
      end;
      POS := POS + 1
    end
  end;
STADEC : begin
  BINARY(ENTRY[1], CP+7-NBIT, NEXT3@.NUMBER); N:=3;
  BINARY(ENTRY[2], CP+7-NBIT, NEXT4@.NEXT1@.NUMBER);
  BINARY(ENTRY[3], CP+7-NBIT, NEXT4@.NEXT2@.NUMBER);
  for I in INS do with ALL[I] do
    begin
      if INPVAR = I then if VACTIVE = HIGH then
        begin
          ENTRY[1,POS] := '0'; ENTRY[2,POS] := '1';
          ENTRY[3,POS] := '1'
        end else
          begin
            ENTRY[1,POS] := '1'; ENTRY[2,POS] := '0';
            ENTRY[3,POS] := '0'
          end;
          if NEXT4@.INPVAR = I then if VACTIVE = HIGH then
            begin
              ENTRY[2,POS] := '0'; ENTRY[3,POS] := '1'
            end else
              begin
                ENTRY[2,POS] := '1'; ENTRY[3,POS] := '0'
              end;
              POS := POS + 1
            end
          end;
        end;
      end;
    end;
  end;
DECSTA : begin
  BINARY(ENTRY[1], CP+7-NBIT, NEXT5@.NEXT1@.NUMBER);
  BINARY(ENTRY[2], CP+7-NBIT, NEXT5@.NEXT2@.NUMBER);
  BINARY(ENTRY[3], CP+7-NBIT, NEXT6@.NUMBER); N:=3;
  for I in INS do with ALL[I] do
    begin
      if INPVAR = I then if VACTIVE = HIGH then
        begin
          ENTRY[1,POS] := '0'; ENTRY[2,POS] := '0';
          ENTRY[3,POS] := '1'
        end else
          begin
            ENTRY[1,POS] := '1'; ENTRY[2,POS] := '1';
            ENTRY[3,POS] := '0'
          end;
        end;
      end;
    end;
  end;
end;

```

```

        if NEXT5@.INPVAR = I then if VACTIVE = HIGH then
        begin
            ENTRY[1,POS] := '0';  ENTRY[2,POS] := '1'
        end else
        begin
            ENTRY[1,POS] := '1';  ENTRY[2,POS] := '0'
        end;
        POS:=POS+1
    end
end;
DECDEC : begin
    BINARY(ENTRY[1],CP+7-NBIT,NEXT7@.NEXT1@.NUMBER);
    BINARY(ENTRY[2],CP+7-NBIT,NEXT7@.NEXT2@.NUMBER);
    BINARY(ENTRY[3],CP+7-NBIT,NEXT8@.NEXT1@.NUMBER);
    BINARY(ENTRY[4],CP+7-NBIT,NEXT8@.NEXT2@.NUMBER);
    N:=4;
    for I in INS do with ALL[I] do
    begin
        if INPVAR = I then if VACTIVE = HIGH then
        begin
            ENTRY[1,POS] := '0';  ENTRY[2,POS] := '0';
            ENTRY[3,POS] := '1';  ENTRY[4,POS] := '1'
        end else
        begin
            ENTRY[1,POS] := '1';  ENTRY[2,POS] := '1';
            ENTRY[3,POS] := '0';  ENTRY[4,POS] := '0'
        end;
        if NEXT7@.INPVAR = I then if VACTIVE = HIGH then
        begin
            ENTRY[1,POS] := '0';  ENTRY[2,POS] := '1'
        end else
        begin
            ENTRY[1,POS] := '1';  ENTRY[2,POS] := '0'
        end;
        if NEXT8@.INPVAR = I then if VACTIVE = HIGH then
        begin
            ENTRY[3,POS] := '0';  ENTRY[4,POS] := '1'
        end else
        begin
            ENTRY[3,POS] := '1';  ENTRY[4,POS] := '0'
        end;
        POS:=POS+1
    end
    end
end; (* case *)
writeln(TABLE,NAME,ENTRY[1]);
for I:=2 to N do writeln(TABLE,S7,ENTRY[I]);
end;
writeln(TABLE);  writeln(TABLE,S8,LINE);  POS:=2;

```

```

for I in INS do with ALL[I] do
begin
  if VACTIVE = HIGH then ENTRY[1,POS]:='H' else ENTRY[1,POS]:='L';
  POS:=POS+1
end;
for I:= CP-NBIT to CP-1 do
begin
  ENTRY[1,I]:='H'; ENTRY[1,I+7]:='H'
end;
POS:=CP+8;
for I in OUTS do with ALL[I] do
begin
  if VACTIVE = HIGH then ENTRY[1,POS]:='H' else ENTRY[1,POS]:='L';
  POS:=POS+1
end
end;
writeln(TABLE,S5,ENTRY[1]);
writeln(TABLE,S8,LINE);
close(TABLE)
end
end; (* ASMTABLE *)

```

```

procedure BINARY(* var LIN:TABLEENTRY; P,NUM:integer *);
(*****
* PROCEDURE BINARY;                                     1.7.1 *
* PURPOSE   : This procedure converts a decimal state number into a *
*             binary string. The characters are entered in the *
*             string 'LIN' starting at position 'P'.           *
* INPUTS    : P -- Starting position for character entry.      *
*             NUM -- State number to be converted.              *
*             --PROCEDURES CALLED--                             *
* INTERNAL  : NONE.                                           *
* EXTERNAL  : NONE.                                           *
*****)

var I,NUM,D : integer;

begin (* BINARY *)
  INUM:=NUM; D:=1;
  for I:=1 to NBIT-1 do D:=D*2;
  for I:=1 to NBIT do
    begin
      if INUM div D = 1 then
        begin
          LIN[P+I-1]:='1'; INUM:=INUM-D
        end else LIN[P+I-1]:='0';
        D:=D div 2
      end
    end;
end; (* BINARY *)

```

```

program ASM1;
(*****
* TITLE      : ASM1                      BY : Thomas Tomcik          2.0.0 *
* LANGUAGE   : TI PASCAL                  *
* COMPUTER   : TI 990/12 with DX10 3.6.0 *
* ABSTRACT   : This dummy program contains seven general purpose *
*               procedures used in the ASMDESIGN program. They are *
*               accessed with the EXTERNAL statement from the main *
*               program. The link control file for ASMDESIGN must also *
*               include the object code generated by the compiler. *
* FILES      : .RES.RESS48.MASTER.ASM1 -- Source code.             *
*               .RES.RESS48.OBJLIB.ASM1 -- Object code.             *
*               --PROCEDURES CALLED--                                *
* INTERNAL   : NONE.                                                  *
* EXTERNAL   : CLEARSCREEN, DISPLAY, ACCEPT.                         *
*****)

const MAXLENFIL = 24; (* Maximum Length of File Name. *)
      DUMY       = '          '; (* 24 Spaces. *)

type SCISTRING = packed array[ 0..MAXLENFIL ] of char;
      FILENAME  = packed array[ 1..MAXLENFIL ] of char;
      STRING2   = packed array[ 1..2 ] of char;
      STRING8   = packed array[ 1..8 ] of char;
      BTYPE     = array[ 1..16 ] of integer;
      CMD       = (SD,DD,RD,VA,AC,MT,HF,QU,NU,IV,MV,VT,AS,MS,RM);
      CSET      = set of CMD;

procedure CLEARSCREEN(var BLOCK:BTYPE); EXTERNAL;
procedure DISPLAY(var BLOCK:BTYPE; LINE,COLUMN:integer;
      BUFFER:packed array[ 1..? ] of char; LENGTH:integer); EXTERNAL;
procedure ACCEPT(var BLOCK:BTYPE; LINE,COLUMN:integer;
      var BUFFER:packed array[ 1..? ] of char;
      LENGTH:integer; var TC:char); EXTERNAL;

procedure WAIT(var BLOCK:BTYPE);
(*****
* PROCEDURE WAIT;                                                  2.1.0 *
* PURPOSE   : This procedure suspends program execution until a key *
*               is entered.                                          *
*               --PROCEDURES CALLED--                                *
* INTERNAL   : NONE.                                                  *
* EXTERNAL   : DISPLAY, ACCEPT.                                       *
*****)

var USERIN : STRING2;
      TC    : char;

begin (* WAIT *)
      DISPLAY(BLOCK,24,70,'<RETURN>',8); ACCEPT(BLOCK,24,80,USERIN,1,TC)
end; (* WAIT *)

```

```

procedure IDISPLAY(var BLOCK:BTYP; LIN,COL,NUM:integer);
(*****
* PROCEDURE IDISPLAY;                                2.2.0 *
* PURPOSE   : This procedure displays integers in [ 0..99 ] at screen *
*             coordinates LIN and COL.                      *
* INPUTS    : LIN -- Screen line.                          *
*             COL -- Screen column.                        *
*             NUM -- Displayed integer.                    *
*             --PROCEDURES CALLED--                        *
* INTERNAL  : NONE.                                         *
* EXTERNAL  : DISPLAY.                                     *
*****
)

```

```

var   SNUM : STRING2;
      ZERO : integer;

```

```

begin (* IDISPLAY *)
  SNUM:=' '; ZERO:=ord('0');
  if NUM in [0..9] then SNUM:=chr(NUM+ZERO) else
  begin
    SNUM[1]:=chr(NUM div 10 + ZERO);
    SNUM[2]:=chr(NUM mod 10 + ZERO)
  end;
  DISPLAY(BLOCK,LIN,COL,SNUM,2)
end; (* IDISPLAY *)

```

```

procedure IACCEPT(var BLOCK:BTYP; LIN,COL:integer; var NUM:integer);
(*****
* PROCEDURE IACCEPT;                                2.3.0 *
* PURPOSE   : This procedure accepts integers in [ 0..99 ] at screen *
*             coordinates LIN and COL.                      *
* INPUTS    : LIN -- Screen line.                          *
*             COL -- Screen column.                        *
* OUTPUT    : NUM -- Accepted integer.                    *
*             --PROCEDURES CALLED--                        *
* INTERNAL  : NONE.                                         *
* EXTERNAL  : ACCEPT.                                       *
*****
)

```

```

var   USER      : STRING2;
      TC        : char;
      DIG1,DIG2 : boolean;
      ZERO      : integer;

```

```

begin (* IACCEPT *)
  ZERO:=ord('0');
  repeat
    ACCEPT(BLOCK,LIN,COL,USER,2,TC);
    DIG1:=(USER[1] in ['0'..'9']) and (USER[2] = ' ');
    DIG2:=(USER[1] in ['0'..'9']) and (USER[2] in ['0'..'9'])
  until DIG1 or DIG2;

```

```

    if DIG1 then NUM:=ord(USER[1])-ZERO else
    NUM:=10*(ord(USER[1])-ZERO)+ord(USER[2])-ZERO
end;  (* IACCEPT *)

```

```

procedure MENU(var BLOCK:BTYP; MODE:CMD);
(*****
* PROCEDURE MENU;                                     2.4.0 *
* PURPOSE   : This procedure displays the ASMDESIGN menus.      *
* INPUT     : MODE -- Menu selector.                            *
*           : --PROCEDURES CALLED--                             *
* INTERNAL  : NONE.                                             *
* EXTERNAL  : CLEARSCREEN, DISPLAY.                             *
*****)

```

```

begin (* MENU *)
  CLEARSCREEN(BLOCK);
  DISPLAY(BLOCK,1,31,'ASM Design Program.',20);
  case MODE of
    NU      : begin
      DISPLAY(BLOCK,3,31,'Main Menu.',11);
      DISPLAY(BLOCK,5,33,'I/O Commands:',13);
      DISPLAY(BLOCK,6,25,'RD <--> Read From Disk File.',28);
      DISPLAY(BLOCK,7,25,'SD <--> Save to Disk File.',26);
      DISPLAY(BLOCK,8,25,'DD <--> Delete From Disk File.',30);
      DISPLAY(BLOCK,10,33,'General Commands:',17);
      DISPLAY(BLOCK,11,25,'VA <--> Variable Assignment.',29);
      DISPLAY(BLOCK,12,25,'AC <--> ASM Charts.',20);
      DISPLAY(BLOCK,13,25,'MT <--> Machine Tables.',24);
      DISPLAY(BLOCK,14,25,'HF <--> Help Files.',20);
      DISPLAY(BLOCK,15,25,'QU <--> QUIT Program Execution.',32)
    end;
    VA      : begin
      DISPLAY(BLOCK,3,31,'Variable Assignment.',21);
      DISPLAY(BLOCK,5,25,'IV <--> Insert Variables.',25);
      DISPLAY(BLOCK,6,25,'VT <--> Variable Table.',24);
      DISPLAY(BLOCK,7,25,'MV <--> Modify Variables.',25);
      DISPLAY(BLOCK,8,25,'HF <--> Help Files.',20);
      DISPLAY(BLOCK,9,25,'RM <--> Return to Main Menu.',29)
    end;
    IV,MV   : begin
      if MODE=IV then DISPLAY(BLOCK,5,25,'Insert Variables.',18)
      else DISPLAY(BLOCK,5,25,'Modify Variables.',18);
      DISPLAY(BLOCK,7,20,'Variable Number:',18);
      DISPLAY(BLOCK,8,20,'Variable Name:',15);
      DISPLAY(BLOCK,9,20,'Active Level (HIGH or LOW) :',29);
      DISPLAY(BLOCK,12,20,'Another Variable (Y or N) :',28);
      DISPLAY(BLOCK,10,20,'Variable Use (INPUT or OUTPUT) :',34)
    end;
    AC      : begin
      DISPLAY(BLOCK,3,34,'ASM Charts.',12);
      DISPLAY(BLOCK,5,25,'AS <--> Assign States.',22);

```

```

        DISPLAY(BLOCK,6,25,'MS <--> Modify States.',22);
        DISPLAY(BLOCK,7,25,'HF <--> Help Files.',20);
        DISPLAY(BLOCK,8,25,'RM <--> Return to Main Menu.',29)
    end
end (* case *)
end; (* MENU *)

procedure ERRORMSG(var BLOCK:BTYP; ER:integer; F:FILENAME;
    PAUSE:boolean);
(*****
* PROCEDURE ERRORMSG;                                2.5.0 *
* PURPOSE   : This procedure displays error messages in the VDT. *
* INPUT     : PAUSE -- Procedure WAIT is called if true.         *
*           : F      -- Invalid file name.                       *
*           : ER     -- Error number.                             *
*           : --PROCEDURES CALLED--                               *
* INTERNAL  : WAIT.                                              *
* EXTERNAL  : DISPLAY.                                           *
*****
)

var I : integer;

begin (* ERRORMSG *)
    DISPLAY(BLOCK,24,1,'ERROR: ',8);
    for I:=1 to 9 do DISPLAY(BLOCK,24,8*I,' ',8);
    case ER of
        1 : DISPLAY(BLOCK,24,8,'RESERVED FILE NAME !',21);
        2 : DISPLAY(BLOCK,24,8,'THE FIRST CHARACTER MUST BE A LETTER !',38);
        3 : DISPLAY(BLOCK,24,8,'ILLEGAL CHARACTER IN THE FILE NAME !',36);
        4 : DISPLAY(BLOCK,24,8,'VARIABLE TABLE EMPTY !',22);
        5 : DISPLAY(BLOCK,24,8,'MACHINE INDEX FULL !',20);
        6 : begin
            DISPLAY(BLOCK,24,8,F,MAXLENFIL);
            DISPLAY(BLOCK,24,MAXLENFIL+9,'NOT FOUND !',11)
        end;
        7 : DISPLAY(BLOCK,24,8,'VARIABLE TABLE FULL !',21);
        8 : DISPLAY(BLOCK,24,8,'OPTION NOT AVAILABLE !',22);
        9 : DISPLAY(BLOCK,24,8,'THERE IS NO ASM IN MEMORY !',27);
        10 : DISPLAY(BLOCK,24,8,'THE NUMBER IS OUT OF RANGE !',28);
        11 : DISPLAY(BLOCK,24,8,'STATE TABLE FULL !',18);
        12 : DISPLAY(BLOCK,24,8,'VARIABLE NOT FOUND !',20);
        13 : DISPLAY(BLOCK,24,8,'STATE NOT INITIALIZED !',22)
    end; (* case *)
    if PAUSE then
        begin
            WAIT(BLOCK); for I:=1 to 3 do DISPLAY(BLOCK,24,24*I-15,DUMY,24)
        end
    end; (* ERRORMSG *)
end;

```

```

procedure GETCMD(var BLOCK:BTYP; OKSET:CSET; var CH:CMD);
(*****
* PROCEDURE GETCMD;                                     2.6.0 *
* PURPOSE   : This procedure accepts a CMD type variable from the *
*             user's terminal.                                   *
* INPUT     : OKSET -- Set of valid commands.                  *
* OUTPUT    : CH    -- Requested command.                      *
*             --PROCEDURES CALLED--                             *
* INTERNAL  : ERRORMSG.                                         *
* EXTERNAL  : DISPLAY, ACCEPT.                                  *
*****
)

var  CNTR   :  CMD;
     GOOD   :  boolean;
     USER   :  STRING2;
     TC     :  char;
     CTABLE :  array[ CMD ] of STRING2;

begin (* GETCMD *)
  CTABLE[SD] := 'SD'; CTABLE[DD] := 'DD'; CTABLE[RD] := 'RD';
  CTABLE[VA] := 'VA'; CTABLE[AC] := 'AC'; CTABLE[MT] := 'MT';
  CTABLE[HF] := 'HF'; CTABLE[QU] := 'QU'; CTABLE[NU] := 'NU';
  CTABLE[IV] := 'IV'; CTABLE[MV] := 'MV'; CTABLE[VT] := 'VT';
  CTABLE[AS] := 'AS'; CTABLE[MS] := 'MS'; CTABLE[RM] := 'RM';
  CH:=NU;
  DISPLAY(BLOCK,21,21,'Select an Option:',20);
  repeat
    DISPLAY(BLOCK,21,39,' ',2);
    ACCEPT(BLOCK,21,39,USER,2,TC);
    for CNTR:=SD to RM do
      if USER = CTABLE[CNTR] then CH:=CNTR;
    GOOD:=CH in OKSET;
    if not GOOD then ERRORMSG(BLOCK,8,DUMY,false)
  until GOOD
end; (* GETCMD *)

procedure GRAPHICS(var BLOCK:BTYP; CH:char; L,C:integer);
(*****
* PROCEDURE GRAPHICS;                                     2.7.0 *
* PURPOSE   : This procedure displays the graphics primitives for the *
*             ASMDESIGN program.                                   *
* INPUTS    : CH -- Graphics selection character.                *
*             L  -- Screen line.                                  *
*             C  -- Screen column.                                *
*             --PROCEDURES CALLED--                             *
* INTERNAL  : NONE.                                              *
* EXTERNAL  : DISPLAY.                                           *
*****
)

var  DSTR :  array[ 0..5 ] of packed array[ 1..22 ] of char;
     I,J  :  integer;
     S    :  packed array[ 1..68 ] of char;

```

```

begin (* GRAPHICS *)
  if CH = 'D' then
    begin
      DSTR[0] := 'HHHHHHHHHH6FHHHHHHHHHH';
      DSTR[1] := 'HHHHHHHXXH6HHFHYHHHHHHHH';
      DSTR[2] := '2>>>>>>>6HHHHF>>>>>>>B';
      DSTR[3] := '12>>>>>>>FHHHH6>>>>>>>B1';
      DSTR[4] := '11HHHHHHHFFHH6HHHHHHHH11';
      DSTR[5] := 'F6HHHHHHHFF6HHHHHHHFF6';
    end;
  if CH = 'E' then
    begin
      DSTR[0] := 'HHHHHH6FHHHHHHHHHHHHHHHH';
      DSTR[1] := 'HHXXH6HHFHYHHHHHHHHHHHHHHHH';
      DSTR[2] := '2>>>6HHHHF>>>BHHHHHHHHHH';
      DSTR[3] := '12>>FHHHH6>>B1HHHHHHHHHH';
      DSTR[4] := '11HHHFHH6HH11HHHHHHHHHH';
      DSTR[5] := 'F6HHHFF6HHHFF6HHHHHHHHHH';
    end;
  if CH in ['D','E'] then for I:=0 to 5 do
    for J:=1 to 22 do DSTR[I,J] := chr(ord(DSTR[I,J]) - 40);
  case CH of
    'V' : begin
      S[1] := chr(9);
      for I in [6,24,56,72] do DISPLAY(BLOCK,L,I,S,1)
      end;
    'H' : begin
      for I:=1 to 67 do if I in [1,19,35,51,67] then
        S[I] := chr(24) else S[I] := chr(22);
        DISPLAY(BLOCK,L,6,S,67)
      end;
    'T' : begin
      S[1] := chr(10); S[28] := chr(26);
      for I:=2 to 27 do S[I] := chr(22);
      DISPLAY(BLOCK,L,C,S,28)
      end;
    'S' : begin
      S[1] := chr(9); DISPLAY(BLOCK,L,C,S,1);
      DISPLAY(BLOCK,L,C+27,S,1)
      end;
    'B' : begin
      S[1] := chr(13); S[28] := chr(29);
      for I:=2 to 27 do if I in [13,14] then
        S[I] := chr(12) else S[I] := chr(22);
        DISPLAY(BLOCK,L,C,S,28); S[1] := chr(9); S[2] := chr(9);
        DISPLAY(BLOCK,L+1,C+12,S,2);
        S[1] := chr(30); S[2] := chr(14); DISPLAY(BLOCK,L+2,C+12,S,2)
      end;
  end;

```

```

'D' : for I:=0 to 5 do DISPLAY(BLOCK,L+I,C-10,DSTR[I],22);
'E' : for I:=0 to 5 do DISPLAY(BLOCK,L+I,C-6,DSTR[I],14);
'X' : begin
      S[1]:=chr(10); S[6]:=chr(26);
      for I:=2 to 5 do S[I]:=chr(22);
      DISPLAY(BLOCK,L-1,C-1,S,6); S[1]:=chr(13); S[6]:=chr(29);
      DISPLAY(BLOCK,L+1,C-1,S,6); S[1]:=chr(9); S[6]:=chr(9);
      for I:=2 to 5 do S[I]:=' ';
      DISPLAY(BLOCK,L,C-1,S,6)
    end
  end (* case *)
end; (* GRAPHICS *)

begin (* ASM1 *)
  (*$NO OBJECT *)
end. (* ASM1 *)

```

```

program VDTP;
(*****
* TITLE      : VDTP -- VDT PROCEDURES                      3.0.0 *
* LANGUAGE   : TI PASCAL                                    *
* COMPUTER   : TI 990/12 with DX10 3.6.0                    *
* ABSTRACT   : This dummy program defines a set of procedures which *
*               can be invoked as external Pascal procedures to do I/O *
*               with the user's VDT terminal. All procedures require *
*               access to an array 'B' which is defined as:      *
*               B : array [ 1..16 ] of integer;                *
*               The procedures are:                             *
* ---> INITSCREEN(B)      Initializes the data structure 'B' *
*               for the VDT to which the ACCEPT and DISPLAY calls will *
*               be made. This call must be made before any other calls. *
* ---> CLEARSCREEN(B)     Blanks the screen.                  *
* ---> DISPLAY(B,L,C,S,N) Displays string S beginning at line *
*               L, column C. The length of the field is the lesser of *
*               N or the length of S.                          *
* ---> ACCEPT(B,L,C,S,N,T) Read into string variable S from *
*               the field beginning at line L, column C. The length of *
*               the field is the lesser of N or the length of the *
*               field. Character variable T is set to the termination *
*               character.                                     *
*               On both ACCEPT and DISPLAY, if the line and column are *
*               given as zero, the field will begin at the end of the *
*               previous field.                                *
* FILES      : .RES.RES548.MASTER.DISPLAY -- Source code.      *
*               .RES.RES548.OBJLIB.DISPLAY -- Object code.     *
*****
)

```

```
const IO = 0;
```

```

type BYTE = 0..#FF;
    WORD = 0..#7FFF; (* NOTE: This should be #FFFF, but the TIP
                        compiler would then allocate two
                        words instead of one word. *)

```

```

SCREENPOSITION = packed record
    ROW      : BYTE;
    COLUMN   : BYTE
end;

```

```

OSSTRING = packed array[ 0..80 ] of char;
IOFLAGS = ( BUSY, ERROR, EOF, EVENT, SF4, SF5, SF6, SF7,
             INITIATE, OUTPUTWITHREPLY, UF2, UF3, UF4, UF5,
             EXTENDEDSCALLBLOCK, BLANKOREVENT );
VDTFLAGS = ( SETFIELDSTART, HIGHINTENSITY, BLINKCURSOR,
             GRAPHICS, EIGHTBIT, CARRIAGECONTROL, BEEP,
             RIGHTBOUNDARY, SETCURSORPOSITION,
             USEFILLCHARACTER, DONTINITIALIZEFIELD,
             RETURNONTERMINATIONCHARACTER,
             VF12, VF13, VF14, WARNINGBEEP );

```

```
UTILITYFLAGS = ( AF0, PROGRAMFILE, DIRECTORYFILE,
                  GLOBALLUNO, TERMINALLOCALLUNO, GENERATELUNO,
                  AUTOCREATE, AF7, LRL, TEMPORARYFILE,
                  IMMEDIATEWRITE, AF11, BLANKSUPPRESSED,
                  EXTENDABLE, RELREC, SEQUENTIAL );
```

```
IOCALLBLOCK = packed record
```

```
(* 0 *) SVCCODE : BYTE;
(* 1 *) STATUSCODE : BYTE;
(* 2 *) OPCODE : BYTE;
(* 3 *) LUNO : BYTE;
(* 4, 5 *) FLAGWORD : set of IOFLAGS;
(* 6, 7 *) BUFFERADDRESS : WORD;
(* 8, 9 *) LOGICALRECORDLENGTH : WORD;
(* 10, 11 *) CHARACTERCOUNT : WORD;
(* 12, 13 *) REPLYBLOCKADDRESS : WORD;
```

```
case (* OPCODE: *) integer of
```

```
0..#0F : (
```

```
(* 14, 15 *) EXTENDEDUSERFLAGS : set of VDTFLAGS;
(* 16 *) FILLCHARACTER : char;
(* 17 *) EVENTCHARACTER : char;
(* 18, 19 *) CURSORPOSITION : SCREENPOSITION;
(* 20, 21 *) FIELDBEGINNING : SCREENPOSITION );
```

```
#90..#9F : (
```

```
(* 14, 15 *) BYTE1415 : WORD;
(* 16, 17 *) UTILITYFLAGWORD : set of UTILITYFLAGS;
(* 18, 19 *) DEFINEDLRL : WORD;
(* 20, 21 *) DEFINEDPRL : WORD;
(* 22, 23 *) PATHNAMEPOINTER : @OSSTRING;
(* 24, 25 *) BYTE2425 : WORD;
(* 26, 27 *) BYTE2627 : WORD )
```

```
end;
```

```
BLOCK = record
```

```
OPENFLAG : boolean;
FILLERCHAR : char;
CALLBLOCK : IOCALLBLOCK
```

```
end;
```

```
procedure SVC$(var SUPERVISORCONTROLBLOCK:IOCALLBLOCK); EXTERNAL;
procedure INITSCREEN(var VDT$:BLOCK); FORWARD;
procedure OPENTERMINAL(var VDT$:BLOCK); FORWARD;
procedure DISPLAY(var VDT$:BLOCK; LINE,COLUMN:integer;
var BUFFER:packed array[ 1..? ] of char;
LENGTH:integer); FORWARD;
procedure ACCEPT(var VDT$:BLOCK; LINE,COLUMN:integer;
var BUFFER:packed array[ 1..? ] of char;
LENGTH:integer;
var TERMINATIONCHARACTER:char); FORWARD;
procedure CLEARSCREEN(var VDT$:BLOCK); FORWARD;
```

```

procedure ACCEPT;
(*****
 * PROCEDURE ACCEPT;                                     3.1.0 *
 *****)

const READASCII = #09;

begin (* ACCEPT *)
  with VDT$.CALLBLOCK do
    begin
      SVCCODE:=I0;
      STATUSCODE:=0;
      OPCODE:=READASCII;
      FLAGWORD:=[ EXTENDEDCALLBLOCK ];
      EXTENDEDUSERFLAGS:=[ CARRIAGECONTROL, SETCURSORPOSITION,
        RETURNONTERMINATIONCHARACTER, WARNINGBEEP,
        HIGHINTENSITY ];
      if LINE > 0 then
        begin
          FIELDBEGINNING.ROW:=LINE-1;
          EXTENDEDUSERFLAGS:=EXTENDEDUSERFLAGS + [ SETFIELDSTART ]
        end;
      if COLUMN > 0 then
        begin
          FIELDBEGINNING.COLUMN:=COLUMN-1;
          EXTENDEDUSERFLAGS:=EXTENDEDUSERFLAGS + [ SETFIELDSTART ]
        end;
      CURSORPOSITION:=FIELDBEGINNING;
      if VDT$.FILLERCHARACTER = '#00' then
        EXTENDEDUSERFLAGS:=
          EXTENDEDUSERFLAGS + [ DONTINITIALIZEFIELD ] else
        begin
          EXTENDEDUSERFLAGS:=EXTENDEDUSERFLAGS + [ USEFILLCHARACTER ];
          FILLCHARACTER:=VDT$.FILLERCHAR;
          FLAGWORD:=FLAGWORD + [ BLANKOREVENT ]
        end;
      BUFFERADDRESS:=LOCATION(BUFFER);
      if LENGTH <= UB(BUFFER) then
        LOGICALRECORDLENGTH:=LENGTH
      else LOGICALRECORDLENGTH:=UB(BUFFER);
      SVC$(VDT$.CALLBLOCK);
      if STATUSCODE <> 0 then
        MESSAGE('I/O ERROR ON ACCEPT');
      TERMINATIONCHARACTER:=EVENTCHARACTER
    end
  end; (* ACCEPT *)

```

```

procedure CLEARSCREEN;
(*****
 * PROCEDURE CLEARSCREEN;                                     3.2.0 *
 *****)

const REWIND = #0E;

begin (* CLEARSCREEN *)
  with VDT$.CALLBLOCK do
    begin
      SVCCODE:=IO;
      OPCODE:=REWIND;
      SVC$(VDT$.CALLBLOCK);
      if STATUSCODE <> 0 then
        begin
          MESSAGE('REWIND ERROR');  HALT
        end
      end
    end;
  end; (* CLEARSCREEN *)

procedure DISPLAY;
(*****
 * PROCEDURE DISPLAY;                                       3.3.0 *
 *****)

const WRITEASCII = #0B;

begin (* DISPLAY *)
  with VDT$.CALLBLOCK do
    begin
      SVCCODE:=IO;
      STATUSCODE:=0;
      OPCODE:=WRITEASCII;
      FLAGWORD:=[ EXTENDEDCALLBLOCK ];
      EXTENDEDUSERFLAGS:=[ GRAPHICS ];
      if LINE >0 then
        begin
          FIELDBEGINNING.ROW:=LINE-1;
          EXTENDEDUSERFLAGS:=EXTENDEDUSERFLAGS + [ SETFIELDSTART ]
        end;
      if COLUMN > 0 then
        begin
          FIELDBEGINNING.COLUMN:=COLUMN-1;
          EXTENDEDUSERFLAGS:=EXTENDEDUSERFLAGS + [ SETFIELDSTART ]
        end;
      BUFFERADDRESS:=LOCATION(BUFFER);
      if LENGTH <0 then
        begin
          LENGTH:=-LENGTH;
          EXTENDEDUSERFLAGS:=EXTENDEDUSERFLAGS - [ GRAPHICS ]
        end;
    end;
  end;
end;

```

```

    if LENGTH <= UB(BUFFER) then
        CHARACTERCOUNT:=LENGTH
    else CHARACTERCOUNT:=UB(BUFFER);
    SVC$(VDT$.CALLBLOCK);
    if STATUSCODE <> 0 then
        MESSAGE('I/O ERROR ON DISPLAY')
    end
end; (* DISPLAY *)

procedure INITSCREEN;
(*****
 * PROCEDURE INITSCREEN;                                     3.4.0 *
 *****)

const ASSIGNLUNO = #91;

var  STID : BYTE;

function STATIONID():BYTE; EXTERNAL;

begin (* INITSCREEN *)
    STID:=STATIONID; VDT$.OPENFLAG:=false;
    VDT$.FILLERCHAR:=chr(0);
    with VDT$.CALLBLOCK do
        begin
            SVCCODE:=IO; STATUSCODE:=0; OPCODE:=ASSIGNLUNO;
            REPLYBLOCKADDRESS:=0;
            UTILITYFLAGWORD:=[ GENERATELUNO ];
            DEFINEDLRL:=0;
            DEFINEDPRL:=0;
            new(PATHNAMEPOINTER);
            PATHNAMEPOINTER@[0]:=chr(4);
            PATHNAMEPOINTER@[1]:='S';
            PATHNAMEPOINTER@[2]:='T';
            PATHNAMEPOINTER@[3]:=chr(STID div 10 + ord('0'));
            PATHNAMEPOINTER@[4]:=chr(STID mod 10 + ord('0'));
            BYTE2425:=0; BYTE2627:=0;
            SVC$(VDT$.CALLBLOCK); dispose(PATHNAMEPOINTER)
        end;
        OPENTERMINAL(VDT$)
    end; (* INITSCREEN *)

procedure OPENTERMINAL;
(*****
 * PROCEDURE OPENTERMINAL;                                     *
 *****)

const OPEN          = 0;
    OPENREWIND      = 3;

```

```

begin (* OPENTERMINAL *)
  if not VDT$.OPENFLAG then
    with VDT$.CALLBLOCK do
      begin
        SVCCODE:=IO;
        STATUSCODE:=0;
        OPCODE:=OPENREWIND;
        FLAGWORD:=[ BLANKOREVENT ];
        BUFFERADDRESS:=0;
        LOGICALRECORDLENGTH:=0;
        REPLYBLOCKADDRESS:=0;
        SVC$(VDT$.CALLBLOCK);
        if STATUSCODE <> 0 then
          begin
            MESSAGE('UNABLE TO OPEN TERMINAL'); HALT
          end;
        if BUFFERADDRESS <> 5 then
          begin
            MESSAGE('TERMINAL IS NOT A VDT'); HALT
          end;
        VDT$.OPENFLAG:=true
      end
    end; (* OPENTERMINAL *)

procedure SCROLL(var VDT$:BLOCK; TOP,BOTTOM,NLINES:BYTE);
(*****
* PROCEDURE SCROLL;                                     3.5.0 *
* PURPOSE   : This procedure scrolls the VDT screen.      *
* INPUTS    : TOP    -- Top line to scroll.                *
*            BOTTOM  -- Bottom line to scroll.              *
*            NLINES  -- Number of lines to scroll.          *
*            --PROCEDURES CALLED--                        *
* INTERNAL  : NONE.                                       *
* EXTERNAL  : SVC$.                                       *
*****
)

CONST READDIRECT = #0A;
      WRITEDIRECT = #0C;

var  BUFFER : packed array[ 1..80 ] of char;
      I      : BYTE;

begin (* SCROLL *)
  with VDT$.CALLBLOCK do
    begin
      SVCCODE:=IO; STATUSCODE:=0;
      FLAGWORD:=[ EXTENDEDSCALLBLOCK ];
      BUFFERADDRESS:=LOCATION(BUFFER);
      EXTENDEDUSERFLAGS:=[ SETFIELDSTART, HIGHINTENSITY, GRAPHICS,
        EIGHTBIT, CARRIAGECONTROL, DONTINITIALIZEFIELD ];
      FIELDBEGINNING.COLUMN:=0;

```

```

    for I:=0 to BOTTOM-1-NLINES do
    begin
        OPCODE:=READDIRECT;  FIELDBEGINNING.ROW:=TOP+I+NLINES-1;
        LOGICALRECORDLENGTH:=UB(BUFFER);  SVC$(VDT$.CALLBLOCK);
        OPCODE:=WRITEDIRECT;  FIELDBEGINNING.ROW:=TOP+I-1;
        CHARACTERCOUNT:=UB(BUFFER);  SVC$(VDT$.CALLBLOCK)
    end;
    for I:=1 to 80 do BUFFER[I]:=' ';
    for I:=BOTTOM-NLINES to BOTTOM-1 do
    begin
        FIELDBEGINNING.ROW:=TOP+I-1;
        CHARACTERCOUNT:=UB(BUFFER);  SVC$(VDT$.CALLBLOCK)
    end
    end
end;  (* SCROLL *)

procedure DELFILE(PATH:packed array[ 0..? ] of char);
(*****
* PROCEDURE DELFILE;                                     3.6.0 *
* PURPOSE   : This procedure deletes files from the ASM directory. *
* INPUT     : PATH -- Pathname to delete.                      *
*           : --PROCEDURES CALLED--                             *
* INTERNAL  : NONE.                                             *
* EXTERNAL  : SVC$.                                             *
*****)

const DELETEDFILE = #92;

var  SVCB : IOCALLBLOCK;
     I    : BYTE;

begin (* DELFILE *)
    with SVCB do
    begin
        SVCCODE:=IO;  STATUSCODE:=0;
        OPCODE:=DELETEDFILE;  new(PATHNAMEPOINTER);
        for I:=0 to UB(PATH) do PATHNAMEPOINTER@[I]:=PATH[I];
        SVC$(SVCB);  dispose(PATHNAMEPOINTER)
    end
END;  (* DELFILE *)

begin (* VDTP *)
    (*$NO OBJECT *)
end.  (* VDTP *)

```

VITA

Thomas Michael Tomcik was born in South Bend, Indiana on January 28, 1952. He attended elementary school in Buchanan, Michigan; high school in Warwick, Rhode Island and graduated from Warwick Veterans Memorial High School in June 1970. The following September he entered the University of Rhode Island, and in June 1974 he recieved a Bachelor of Science degree in Electrical Engineering.

After working in the audio electronics field, he entered the Graduate School of the University of Tennessee, Knoxville, in March 1981. He recieved a Master of Engineering degree with a major in Electrical Engineering in June 1984.