AN OPTIMIZED SEQUENTIAL CONTROLLER FOR IMPLEMENTING STATE TRANSITION TECHNIQUES

By

ROBERT M. LAURIE

A THESIS

Submitted in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

MICHIGAN TECHNOLOGICAL UNIVERSITY

1986

This thesis, "An Optimized Sequential Controller for Implementing State Transition Techniques" is hereby approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE IN ELECTRICAL ENGINEERING.

DEPARTMENT: Electrical Engineering

Paul H Lewis
Thesis Advisor

E.K. Stanek
Head of Department

4/15/86
Date

ABSTRACT

AN OPTIMIZED SEQUENTIAL CONTROLLER FOR
IMPLEMENTING STATE TRANSITION TECHNIQUES

By
ROBERT M. LAURIE

A THESIS
Submitted in partial fulfillment of the requirements
for the degree of
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

MICHIGAN TECHNOLOGICAL UNIVERSITY
1986

Sequential control is currently implemented on programmable controllers using relay ladder logic to describe the sequential behavior of a system. State transition techniques have many advantages over relay ladder logic for synthesizing sequential control algorithms. Presented in this thesis are descriptions of the state transition diagram, the Petri net, and the state transition table for synthesizing sequential control algorithms. A sequential controller is designed using a state machine and state table architecture, and the design is implemented on a Motorola 6809 microcomputer system. The sequential controller is capable of implementing multiple active state systems, such as those requiring global control.

ACKNOWLEDGMENTS

I wish to thank Brian Staten for his help in debugging the hardware of the MC6809 microcomputer board. I thank Pam for giving me something great to look forward to after graduation. I wish to express my gratitude to Ed Heikkila and the rest of the EERC night-janitorial crew for their moral support and encouragement on all those late nights throughout my graduate career.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A person walks into an elevator and pushes the button of the desired floor. A sequential controller then takes over. It checks to see that there is no obstruction of the door; closes the door; accelerates to a specified transit speed; decelerates to a final stop at the desired floor; and opens the door. With this sequence complete, control is then passed from the sequential controller to the next person to request service.

Control of elevators, washing machines, and many automated manufacturing machines and processes are all applications of a class of automatic control theory called sequential control. Most operator-machine interfaces are also examples [17]. The increasing need of industry to automate production to remain competitive has generated many new demands for improved techniques in the field of sequential control [8].

Initially, relay logic was the primary means of implementing sequential control. Today, programmable controllers have replaced electromechanical relays in most applications to reduce the cost of implementation and

improve system reliability [38] [29]. The programmable controller uses a graphical language called Relay Ladder Logic to program the sequential control algorithm. Although this language resulted in quick acceptance by industry, it did not improve design methods for sequential control [23].

Much theory and methodology has been developed for continuous control; however sequential control has developed with a minimum of theory or standardized synthesis methods [17]. As sequential systems become more complex and ambitious, improved synthesis methods and new sequential controllers must be developed.

State transition techniques have shown the greatest potential as a synthesis technique for sequential control. They have been used for design and analysis in wholly discrete digital computer systems and have been adapted to sequential control. To reap the real benefits of state transition techniques, a new type of architecture must be developed for programmable controllers. Response times are expected to decrease to one hundredth of what is currently available for scanning-type programmable controllers [23].

This thesis examines sequential control, and how it is

implemented on currently available programmable controllers. Various synthesis methods are compared, and a general purpose sequential controller is designed capable of implementing state transition algorithms with multi-active states. A Motorola 6809 based microcomputer is used to implement this design using state machine and state table architecture. The sequential controller operation is demonstrated for a drilling station example.

CHAPTER 2

SEQUENTIAL CONTROL

This chapter defines sequential control and presents the general case sequential control problem. A semi-automatic drill press is described as an example of sequential control. This example is used in the following two chapters to compare various synthesis techniques.

2.1 DEFINITION

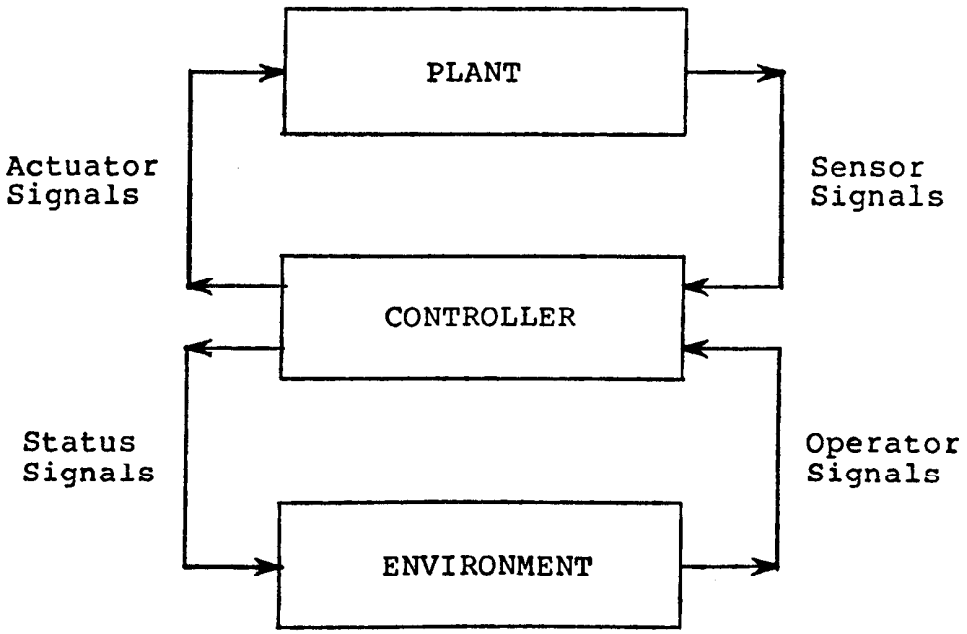
Sequential control is characterized by current events being dependent on past events. Control is determined not only by the value of control signals but also by their order of occurrence. Conceptually, sequential control is viewed as a discrete (noncontinuous) process. Actuation signals are discrete in nature, with a finite number of possible values. Generally, the control algorithm can be expressed as a sequence of actions and conditions; rather than algebraic differential equations as in continuous control. Sequential control is usually characterized by complex cyclical behavior, while continuous control tends to act in response to continuous input signals.

2.2 GENERAL CASE PROBLEM

The generalized sequential control problem consists of

three basic components and four groups of signals as illustrated in Figure 2.1. The plant is the entity to be controlled. The controller implements the control algorithm by transmitting "actuator" signals to the plant dependent on "sensor" signals received from the plant and environmental inputs. The environment is usually an operator or another controller that transmits "operator" signals to the controller and receives "status" signals from the controller. All signals are discrete, having a finite number of possible values.

Figure 2.1: General Case Sequential Control Problem



2.3 SEMI-AUTOMATIC DRILL PRESS EXAMPLE

The semi-automatic drill press shown in Figure 2.2 is an example of a sequential control application. The control signals are grouped into actuator, sensor, and operator signal catagories as described by Section 2.2, in Table 2.1. Note that no status signals exist for this example, therefore this column has been omitted from Table 2.1. Each signal is also described by a letter symbol and this letter is used to represent the signal in the figures to follow. This example will be used in the next two chapters to compare various synthesis techniques.

Start and stop signals are considered to be the only operator signals coming from the environment. When the start button is pushed, the drilling operation begins. When the stop button is pushed all outputs are deactivated.

The drilling operation begins by extending the gripper until the "piece gripped" sensor signals that the workpiece has been gripped. The drill is then lowered until the "drill down" sensor signals that the hole has been drilled. Then the drill is raised until the "drill up" sensor signals that the drill has been returned. The gripper is retracted until the "workpiece free" sensor is activated which defines the end of the drilling operation. This

sequence is repeated when the start button is pushed again.

Figure 2.2: Semi-Automatic Drill Press Example

Actuator Signals: E = Extend gripper
L = Lower drill
R = Raise drill
T = Retract gripper

Sensor Signals: G = Workpiece gripped
D = Drill down
U = Drill up
F = Workpiece free

Operator Signals: S = Start button
P = Stop button

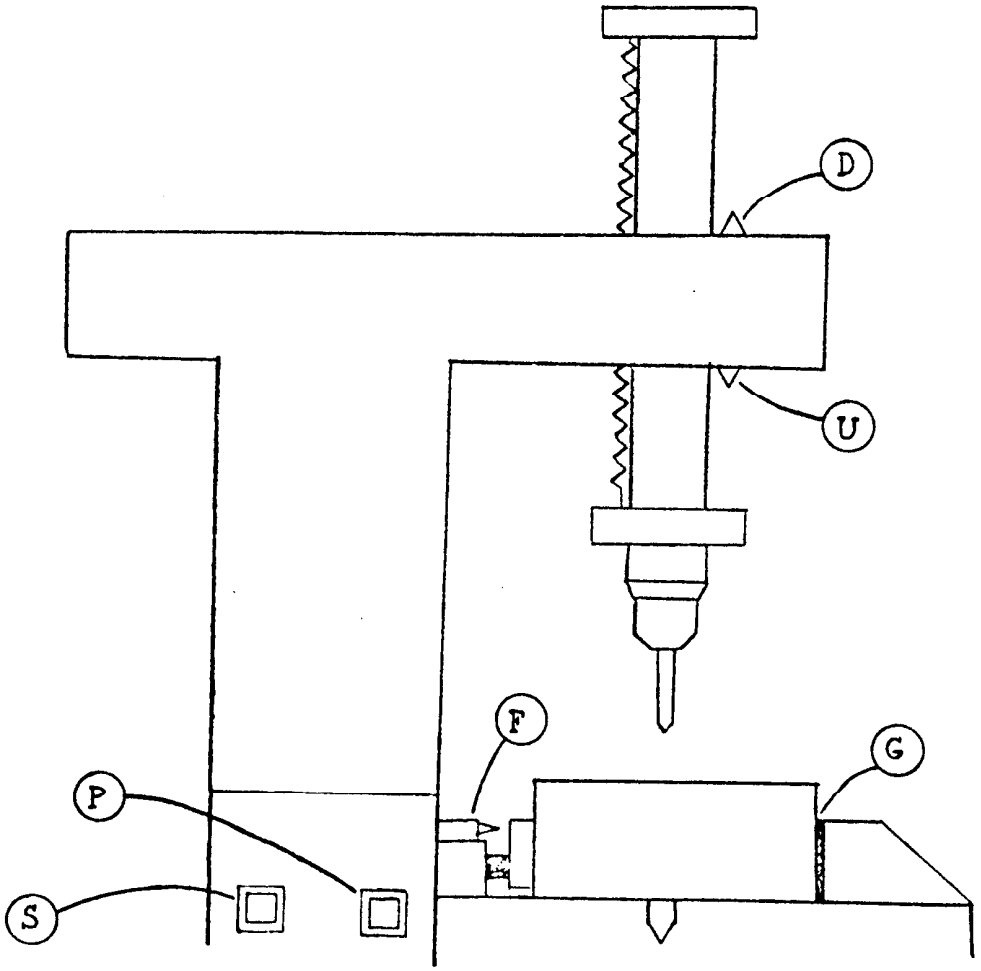


Table 2.1: Control Signal Classification For The Semi-Automatic Drill Press Example

SIGNAL SOURCE	ACTUATOR SIGNAL	SENSOR SIGNAL	OPERATOR SIGNAL
ENVIRONMENT Control Panel			Start button =S Stop button =P
PLANT Drill Press Station	Extend gripper =E Lower drill =L Raise drill =R Retract gripper =T	Workpiece gripped =G Drill down =D Drill up =U Workpiece free =F	

CHAPTER 3

THE PROGRAMMABLE CONTROLLER

The primary implementation method of sequential control is the programmable controller. Described in this chapter are the programmable controller's evolution, industrial acceptance, future trends, and the relay ladder logic programming language.

3.1 EVOLUTION

Sequential control has always been an important part of automation, especially in the automotive industry.

In the beginning, sequential control was implemented using many interconnecting relays. Using electromechanical relays, simple controllers were easily designed, noise immunity was high, and loads could be driven directly. However, difficulties in changeability, maintainability, size, slow speed, and high power consumption, prompted the need for a new solution.

Solid state logic brought much improvement over the shortcomings of relays, but changeability was still a problem. In the automotive industry, literally thousands of relays or logic gates would have to be rewired every

time a car model was changed on the production line. Software-based control algorithms were recognized as the best way to accomplish changes quickly and inexpensively. Programmable controllers were developed to meet these requirements.

3.2 DESCRIPTION

Programmable controllers are implemented using microprocessor-based computer systems. Operation in the industrial environment requires that they be rugged and immune to high levels of heat, electro-magnetic interference, and vibration. A major requirement of a programmable controller is to reliably interface to the plant and environment. Unlike a personal computer, the programmable controller must operate in very harsh industrial environments and be easily interfaced to typical industrial situations. Relatively few mathematical capabilities are provided by programmable controllers; however, extensive mathematical computations are not required to implement sequential control algorithms.

The programmable controller functions as the sequential controller discussed in section 2.2. It accepts discrete inputs from the plant such as push buttons, limit switches, and set points. The controller than determines the proper response to these inputs from the programmed sequential

control algorithm. The control algorithm is then implemented by activating outputs such as relays, motor contactors, and lights.

Most programmable controllers operate as a program scanning translator. This architecture has the advantage of direct programming and implementation by the user. However, it has the disadvantage of slow speed and large program size as compared with compiler type architectures.

The programmable controller has been described as a device that hides a computer in a box in such a way that a designer can use it without knowing anything about computer programming [29]. Since its introduction, the programmable controller has been programmed using a graphical language called relay ladder logic. It allowed for quick acceptance by industry with minimal additional training of personnel to implement existing relay control algorithms.

3.3 INDUSTRIAL ACCEPTANCE

Since its introduction, the programmable controller has sustained very rapid growth. In 1984 sales of programmable controller products reached a high of $800 million, with an expected market growth of 53 percent in 1985 [11]. With the recent demands that automation has placed on industry, this growth should be sustained for many years to come.

3.4 FUTURE TRENDS

Sequential control designers are demanding much more of programmable controllers than what they were originally designed for. Applications have become more complex, and so has the need for improved design and implementation methods. In this section, future trends are discussed as expected for programmable controllers. These include: control algorithm partitioning; distributed control; fault diagnosis; redundancy; faster response times; and higher mathematical capabilities.

As the size of the application increases, so does the need to partition the problem into smaller and more manageable modules [14] [38] [29] [35] [17]. Just as computer programmers modularize large computer programs into smaller subroutines, sequential control designers would like to partition the application software into segments. The basic rule of partitioning is to divide the plant into smaller segments such that dependence between segments is kept to a minimum. Generally, larger plants consist of several stations. Simplification of the plant control algorithm can be accomplished by partitioning control into local segments for the stations and a global segment to control all local segments. This master/slave relationship minimized dependence between segments.

Distributed control through multiprocessing has been shown to reduce implementation costs and improve system availability [9] [4] [30] [22] [31]. As software is simplified by partitioning the control algorithm into segments, the hardware implementation may also be simplified by using several microcomputers instead of a larger mainframe computer. Multiprocessor communication is accomplished through the use of a data highway with benefits of increased throughput and faster response time [4] [30]. By using this distributed control scheme for automated manufacturing, production lines may be added easily and system faults may be localized to specific lines resulting in only partial production shut down. To have all control accomplished through one large computer is like putting all eggs in one basket.

Fault diagnostics are important in decreasing the down time of a plant and debugging the control system during its development [5]. Before a fault can be diagnosed it must be detected. Abnormal behavior must be described for such fault detection to occur [38]. Anticipatory diagnostics are also desired to predict such things as tool wear, bearing temperatures, and overheating.

Plants requiring high reliability such as medical and

aerospace systems, have fault recovery capability through redundant systems [16]. Redundancy must occur in both hardware and software to ensure high reliability [21]. Typical redundant structures are triplex voting and duplex standby.

Slow response times and minimal mathematical capabilities have prevented programmable controller use in many applications including computerized numerical control [31]. These capabilities must be improved for advanced applications.

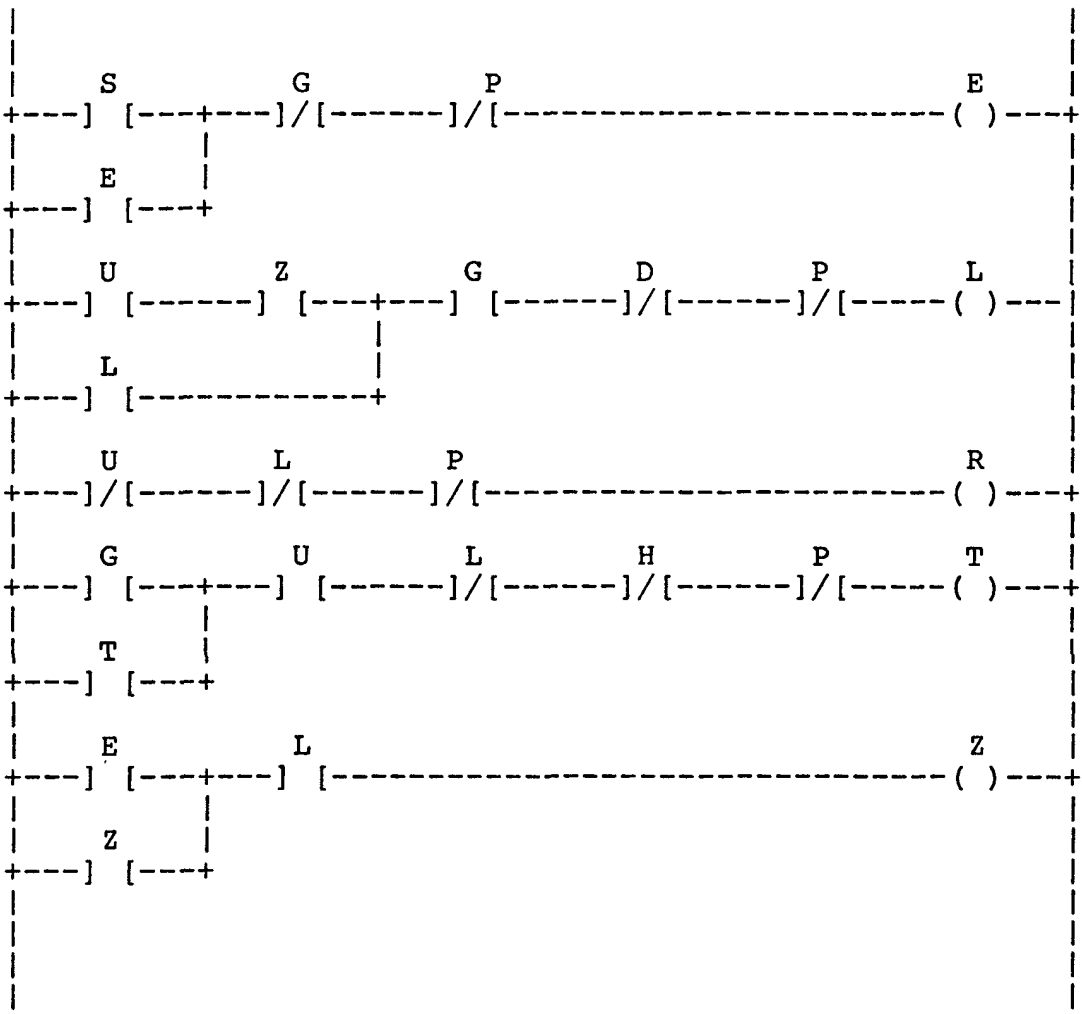
3.5 RELAY LADDER LOGIC PROGRAMMING LANGUAGE

Relay ladder logic is currently the most popular language for programmable controllers [7]. It is a direct implementation of the relay ladder logic design method used by designers for many years [33]. All relays described by the control algorithm program are simply replaced by the programmable controller. The graphical language is desirable in terms of testing and monitoring operations; the symbols are easily recognized, and syntax requirements are minimized. Thus, despite inherent problems relay ladder logic programming is certainly one of the major reasons for quick industrial acceptance of the programmable controller [23].

The relay ladder logic diagram for the example problem of Section 2.3 is shown in Figure 3.1. Each rung of the ladder diagram describes input conditions required to activate an output. Conditions are symbolized by either "--] [--" (examine on) or "--]/[--" (examine off). Outputs are symbolized by the symbol "--( )--". Timers and counters may also be used in the design with properly labeled symbols.

The primary disadvantage of relay ladder logic is that it is a combinational design method and does not describe the sequential progression of the operation [33]. The current state of the system is not given explicitly [14]. This makes designing and debugging very difficult for complex systems. For larger programs the code is difficult to read and modify. Larger control algorithms contain many pages of relay ladder logic symbols. If one symbol is changed, it may affect any other part of the program. There is no formal way to implement fault diagnostics or redundancy using relay ladder logic. As the program size increases, so does the response time. This problem arises because ladder diagrams are executed by rapidly scanning the entire program in an effort to approximate simultaneous execution of all operations [23].

Figure 3.1: Relay Ladder Logic Diagram For The Semi-Automatic Drill Press Example



Note: Z is a "dummy" output (It exists only within the programmed simulation to provide a desired sequence of operations.)

CHAPTER 4

STATE TRANSITION TECHNIQUES

State transition techniques have been widely used as a design and analysis method for digital computer systems. They have recently been adapted as a design and analysis method for sequential control. Unlike relay ladder logic diagrams which use combinational methods to implement sequential control, state transition techniques show sequential behavior explicitly. All information about past behavior, which is required to determine future behavior, is defined by the currently active state. For all state transition techniques, the current state of the system is explicitly given. Therefore, state transition techniques are a natural and highly structured synthesis method for implementing sequential control.

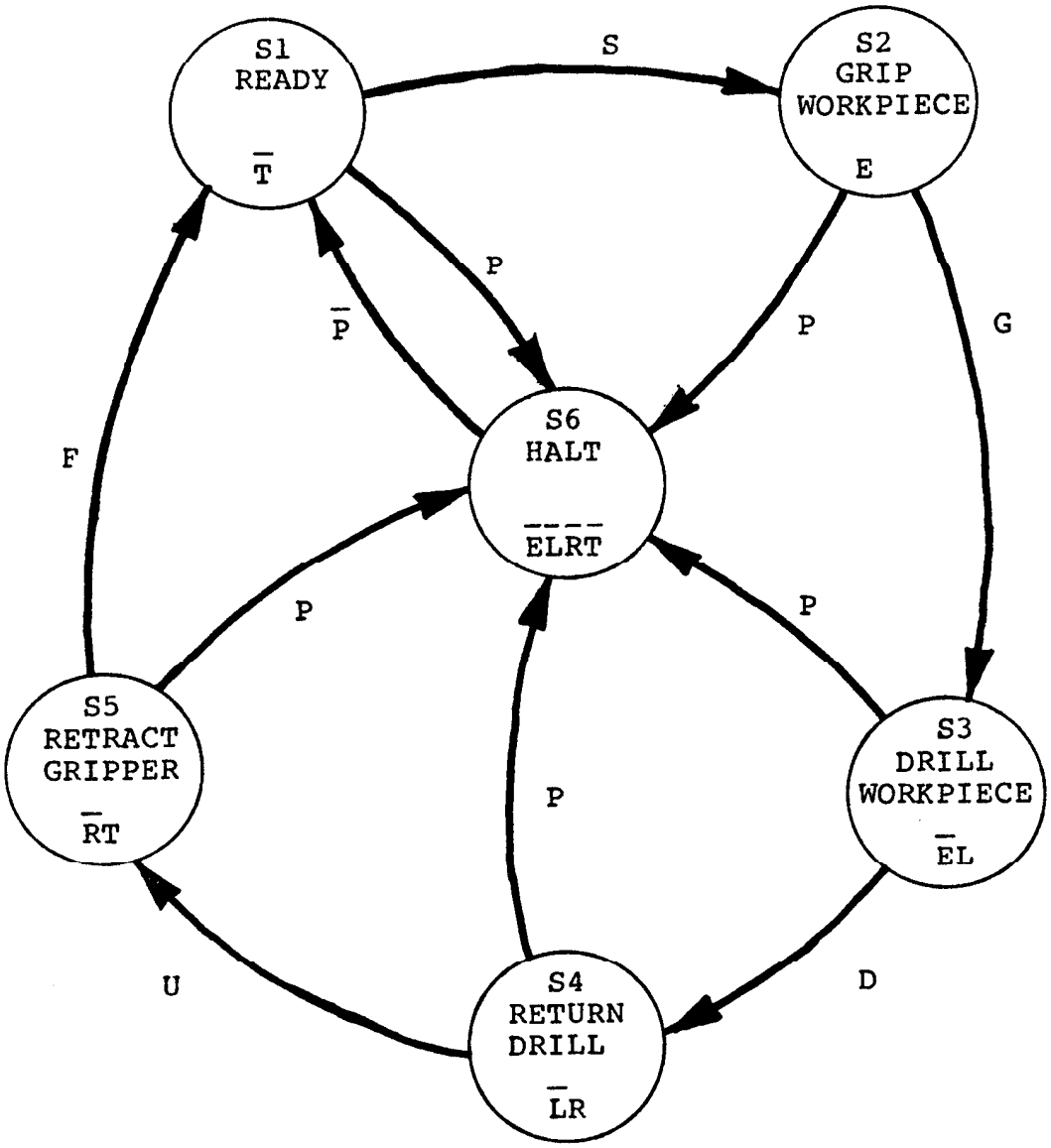
This chapter describes the three state transition techniques applied to sequential control; state transition diagrams, Petri nets, and state transition tables. Advantages and disadvantages of each technique are discussed, and a three station automated drilling system is presented to demonstrate global and local control requirements.

4.1 STATE TRANSITION DIAGRAMS

The state transition diagram for the example of Section 2.3 is shown in Figure 4.1. Each state is represented by a circle enclosing a state label and action list. Directed lines between circles represent possible transitions which occur if the conditions, given in the condition list next to the directed lines, are met. When a state is first entered, the device specified in the action list (such as device "A") is either turned on as indicated by "A" or turned off as indicated by "$\bar{A}$". Only one state may be active at any time.

An advantage of using the state transition diagram approach is that it is a very explicit method of describing sequential control. The desired sequence is easily designed and debugged. The graphical approach makes it easily interpreted, especially for cases which have several transitional paths. Since only one active state exists on a diagram the progression of control can be easily understood. If a specific state is active, only the conditions that are specified for leaving that state are polled by the controller. Fault diagnostics are also easily incorporated into the design.

Figure 4.1: State Transition Diagram For the Semi-Automatic Drill Press Example

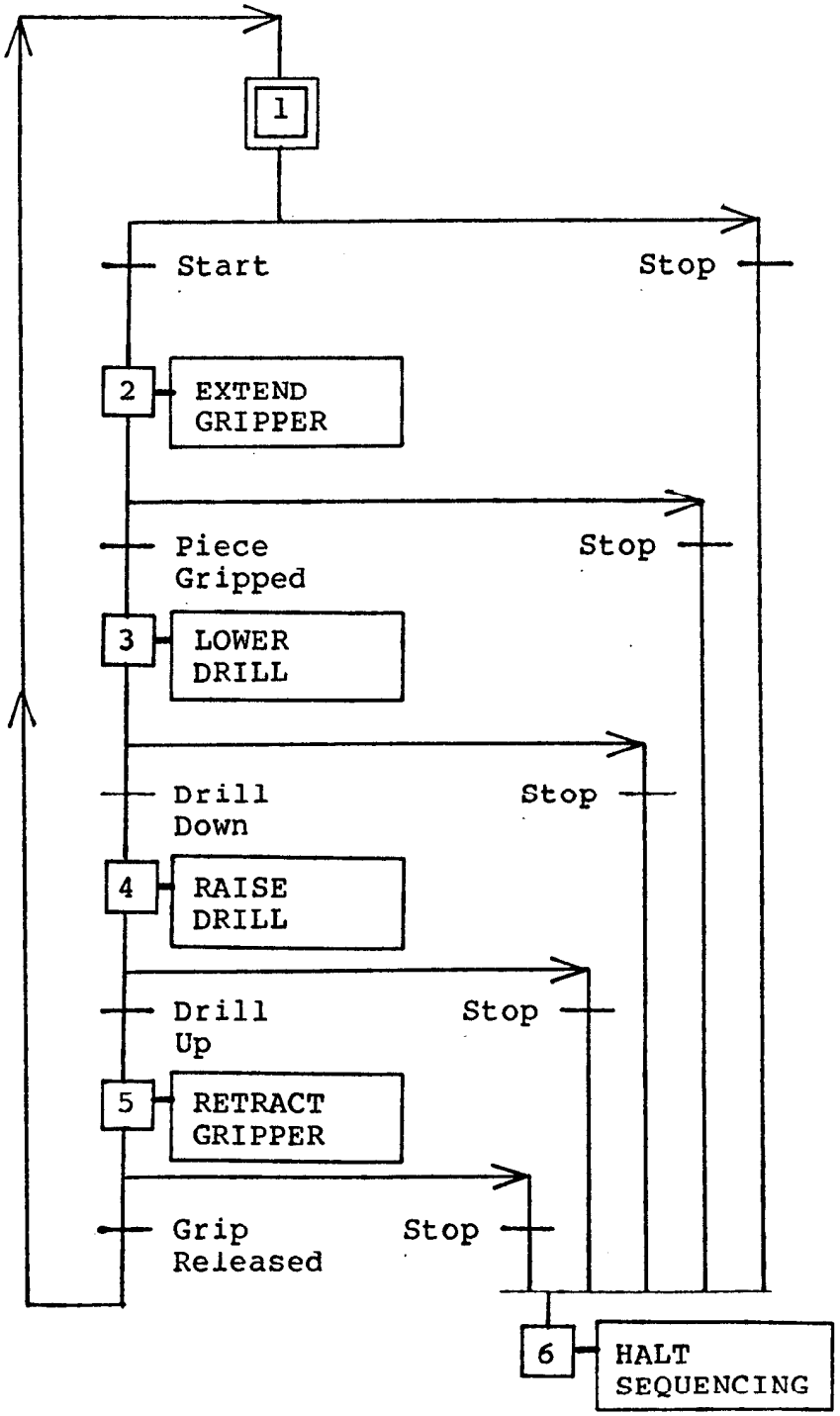


A disadvantage of using state transition diagrams is that no formal way of representing parallel asynchronous processes exists. It is not well suited for describing global control.

4.2 PETRI NET

The Petri net is also a graphical method in which the states of the system are defined by the position of tokens on steps in the net. The Petri net for the example of Section 2.3 is shown in Figure 4.2. Steps are represented by squares containing numbers with the initial step being symbolized by two concentric squares. A rectangle to the right of each square describes the actions which occur if the step is activated. Directed links indicate the flow of control between steps, which is from top to bottom unless otherwise indicated by arrows. Transitions through a link are represented by a short horizontal line with the condition list just to the right. The placement of a token on a step signifies that the step is active and the action list is executed. The token remains on the step until transition conditions are satisfied and it is transferred to a new step. For the relatively simple example of figure 4.2, only one token is on the net at all times. The placement of this token corresponds to the active state.

Figure 4.2: Petri Net Diagram For Semi-Automatic Drill Press Example



For the more complicated case shown in Figure 4.4, several steps may be activated at one time. When transition occurs from step 1, steps 2, 5, and 10 are simultaneously activated, with tokens being placed at steps 2, 5, and 10. This expansion of tokens is symbolized by the double horizontal line. The three tokens progress independently through each of the three step sequences until steps 4, 9, and 14 are all activated. When this occurs the token count is contracted to one and transition occurs to step 17. This reduction in token count is also symbolized by the double horizontal line.

A state exists for each unique placement of tokens on the net. As calculated in Equation 4.1, this Petri net with 17 steps is equivalent to 122 states.

Equation 4.1:

Total Number of States
= [Step1]
+ [{Steps2,3,4} x {Steps5,6,7,8,9}
x {(Step10) + (Steps11,12,13 x Steps 15,16)
+ (Step14)}] + [Step17]

= 1 + [3 x 5 x {1 + (3 x 2) + 1}] + 1

= 122 States

The advantage of using Petri nets is that parallel asynchronous processes can be represented directly. As

shown by Equation 4.1, it reduces the complexity of a problem with parallel processes over a state transition diagram description. The Petri net is a highly structured and logical design method, and a natural choice for the design of global controllers.

A disadvantage of using Petri nets is that nets become confusing with several transitional paths. Petri nets are slightly more confusing than state transition diagrams for local control (single active state) applications.

4.3 STATE TRANSITION TABLE

A state transition table is a tabular representation of the control algorithm. The state transition table for the example of Section 2.3 is shown in Table 4.1. Four columns exist on a state transition table: the state label; an action list, which is executed when the state is entered; a conditions list, which defines conditions for a transition to occur; and the next state, which is entered if a transition occurs. A state transition table can be constructed easily for any sequential control application. The state transition table clearly answers the question "what will happen if...?" and is easily modified if an answer is unacceptable. Global control can be implemented on a state transition table by using an "expand active states" command. This will have the same effect as

Table 4.1: State Transition Table For The Semi-Automatic Drill Press Example

STATE LABEL	ACTIONS LIST	CONDITIONS LIST	NEW STATE
S1: Ready	$\overline{T}$	S	2
		P	6
S2: Grip Workpiece	E	G	3
		P	6
S3: Drill Workpiece	$\overline{E}$ L	D	4
		P	6
S4: Return Drill	$\overline{L}$ R	U	5
		P	6
S5: Retract Gripper	$\overline{R}$ T	F	1
		P	6
S6: Halt	$\overline{E}$ $\overline{L}$ $\overline{R}$ $\overline{T}$	$\overline{P}$	1

expanding active steps for the Petri net. A contract active states command must also be included to reduce the number of active states. The state transition table for such a multi-active state system is shown in Table 4.3. The advantage of using a state transition table is that any sequential control problem can be synthesized, no matter how many parallel processes or transition paths exist.

The disadvantage of using state transition tables is that the transfer of control between states is not as obvious as with graphical methods.

4.4 THREE STATION AUTOMATED DRILLING SYSTEM EXAMPLE

This example is used to demonstrate the applicability of the three state transition techniques described to a system with parallel asynchronous processes. It will also be used to test the sequential controller design of this thesis. This example is presented in reference [38].

As shown in Figure 4.3, the drilling station consists of a turntable with three stations positioned 120 degees apart. The control signals are grouped into the four catagories described by Section 2.2, in Table 4.2. Signals are labeled in Table 4.2, with a letter and number.

Figure 4.3: Three Station Automated Drilling System

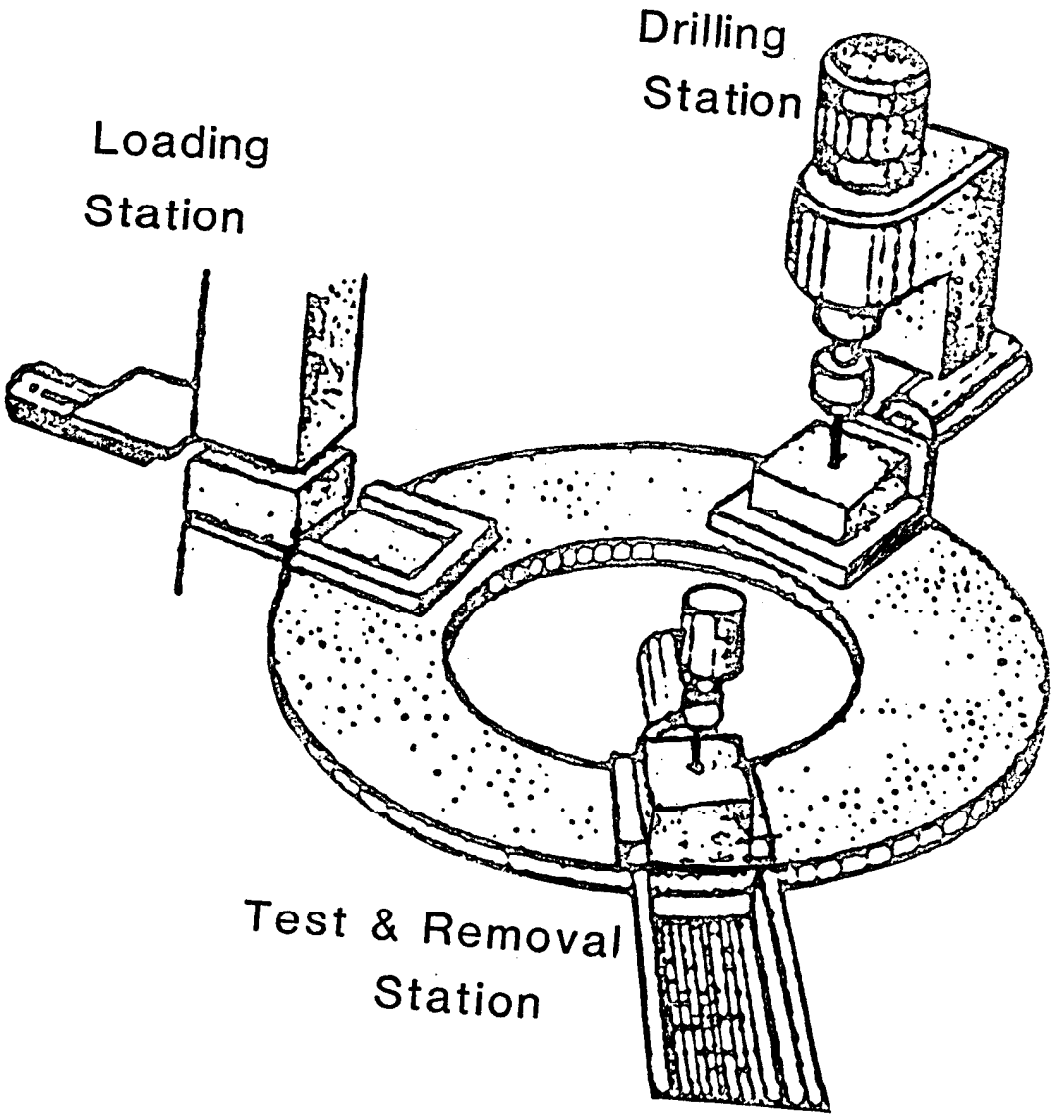


Table 4.2: Control Signal Classification For Three Station Automated Drilling System Example

SIGNAL SOURCE	ACTUATOR SIGNAL	SENSOR SIGNAL	OPERATOR SIGNAL	STATUS SIGNAL
ENVIRONMENT Control Panel			Start button =C1 Stop button =C2	Request Removal =C3
PLANT Station 1 Workpiece Loader	Extend loader =L1 Retract loader =L3	Piece loaded =L2 Loader return =L4		Loading done =L5
PLANT Station 2 Drilling Station	Extend gripper=D1 Retract gripper=D3 Lower drill =D5 Raise drill =D7	Workpiece gripped=D2 Grip release=D4 Drill down =D6 Drill up =D8		Drilling done =D9
PLANT Station 3 Testing & Unloading Station	Lower tester =T1 Raise tester =T3 Extend remover=T5 Retract remover=T7	Tester down =T2 Tester up =T4 Workpiece removed=T6 Remover return =T8	Manually removed=T9	Testing done =T10
PLANT	Rotate Plate =R1	Plate Rotated=R2		

Start and stop signals are considered to be the only operator signals coming from the environment. When the start button is pushed, the sequential controller will begin implementation of the control algorithm. When the stop button is pushed, power to the plant is shut off.

At Station 1, workpieces are loaded onto the turntable one at a time. The loader arm extends until the "piece loaded" sensor is activated and then retracts until the "loader return" sensor is activated. The next workpiece then positions itself on the loader driven by gravity.

Station 2 is the drilling station. The gripper extends until the "workpiece gripped" sensor is activated. The drill is then lowered until the "drill down" sensor is activated which signals that the hole has been drilled. The drill is then raised until the "drill up" sensor signals its return and grip is released.

Station 3 is the testing station. The tester is lowered and a timer is activated. If the "tester down" sensor activates prior to an elapsed time of two seconds, the workpiece passes the test. The tester is raised until the "tester up" sensor signals its return, the remover is extended until the "workpiece removed" sensor signals removal, and the remover is retracted until the "remover

returned" sensor signals its return. If an elapsed time of two seconds occurs without the "tester down" sensor activating, the workpiece fails the test. The tester is raised until the "tester up" sensor signals its return and manual removal is requested until the "manually removed" sensor signals that it has been removed.

When the tasks of all three stations have been completed, only then is the turntable rotated 120 degrees. After rotation, the entire sequence is repeated. This cyclical behavior continues until the stop button is activated.

As shown in Table 4.2, this example requires eleven sensor inputs, three operator inputs, eleven actuator outputs, and four status outputs.

4.5 EVALUATION OF METHODS

State transition techniques are very useful in synthesizing sequential control algorithms. They are well adapted for sequential control purposes and show the sequential behavior of the system explicitly.

The three state transition techniques discussed have many advantages over relay ladder logic methods [14]. They force logical, structured, and accurate designs. Record keeping is convenient and complete. The exact state of

the system is explicitly shown. Partitioning of the control algorithm is easily accomplished. Diagnostics and redundancy can be incorporated in the design by considering anominal behavior [38] [16]. The algorithm can be easily modified and debugged. All sequential functions used by programmable controllers are included with state transition techniques.

All three techniques discussed can be used to synthesize sequential control algorithms; however the best choice depends on the requirements of a particular application. For global control either the Petri net or state transition table is the better choice. Figure 4.4 shows the Petri net for the example of Section 4.4. The transition of control is more obvious with the Petri net than with the state transition table description of the same application as shown in Table 4.3. The state transition diagram would become unmanageable for this application, since it requires 122 states. For local control or algorithms requiring many branching transition paths, the state transition diagram would be less confusing than the Petri net. Graphical methods are preferred for simple applications, since the transfer of control is more obvious than with a state transition table. However, as applications become more complex, graphical methods become cumbersome and state transition tables are

Figure 4.4: Petri Net Diagram For Three Station Automated Drilling System Example

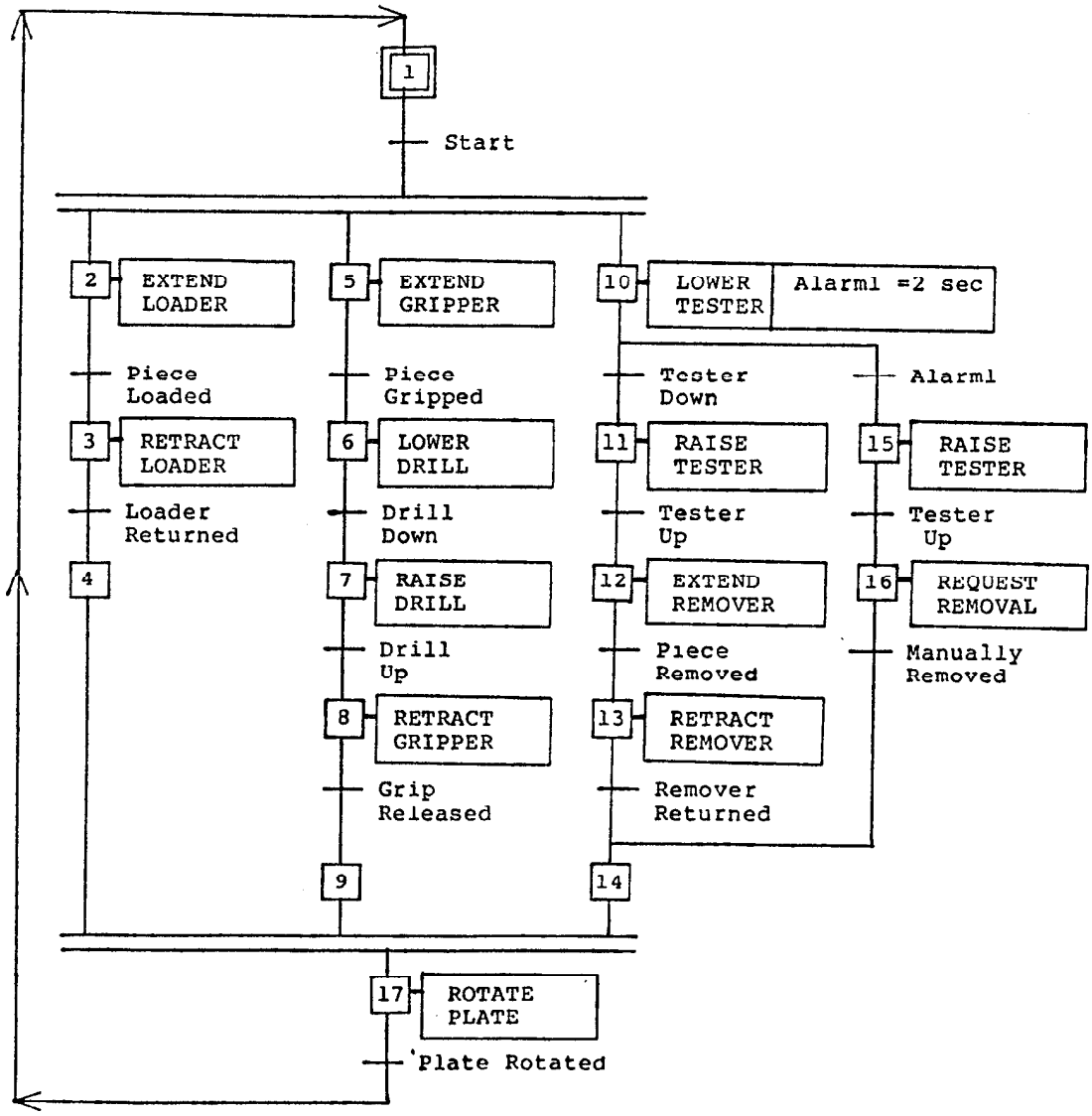


Table 4.3: State Transition Table For The Three Station Automated Drilling System Example

STATE LABEL	ACTIONS LIST	CONDITIONS LIST	NEW STATE
S1: Ready	$\overline{R1}$	C1	2,5,10
S2: Load Workpiece	L1	L2	3
S3: Return Loader	$\overline{L1}$ L3	L4	4
S4: Loading Done	$\overline{L3}$ L5	States 9,14	17
S5: Grip Workpiece	D1	D2	6
S6: Drill Workpiece	$\overline{D1}$ D5	D6	7
S7: Return Drill	$\overline{D5}$ D7	D8	8
S8: Release Grip	$\overline{D7}$ D3	D4	9
S9: Drilling Done	$\overline{D3}$		

(State transition table continued on next page)

Table 4.3: State Transition Table (Continued)

S10: Test Workpiece	Alarm1= 2sec T1	T2 Alarm1	11 15
S11: Test Pass	$\overline{T1}$ T3	T4	12
S12: Remove Workpiece	$\overline{T3}$ T5	T6	13
S13: Return Remover	$\overline{T5}$ T7	T8	14
S14: Testing & Removal Done	$\overline{T7}$ T10		
S15: Test Fail	$\overline{T1}$ T3	T4	16
S16: Request Removal	$\overline{T3}$ C3	T9	14
S17: Rotate Plate	$\overline{L5}$ $\overline{D9}$ $\overline{T10}$	R2	1

the preferred approach [24]. It is always easy to construct a state transition table from a state transition diagram or Petri net representation, but it is often difficult to construct a graphical representation of a complex system from the state transition table. Consider the system with 50 states and 30 transitional paths between each of the states. For this case either graphical approach would be confusing, and the state transition table description should be used.

CHAPTER 5

SEQUENTIAL CONTROLLER ARCHITECTURE

State transition techniques are far superior to relay ladder logic for synthesizing sequential control algorithms. Programmable controllers use a scanning-translator architecture for implementing relay ladder logic; however, state transition techniques require a new approach with respect to sequential controller architecture. Discussed in this chapter are functional requirements of a sequential controller and a sequential controller architecture based on the state machine and state table concept.

5.1 FUNCTIONAL REQUIREMENTS

The sequential controller must execute the basic sequential control functions common among programmable controllers. These sequential functions include discrete I/O checking and manipulation, counter operations, and timer alarms.

All functions required by state transition techniques must be executable on the sequential controller. Transferring from a state transition technique representation of the control algorithm to the required application data must be

easily accomplished. The application data should be compact and require minimal processing time. Data for implementing several application algorithms should be accessible to the sequential controller.

The sequential controller design should be implemented on microprocessor-based systems, and its program should be written in assembly language to generate fast and compact machine code. Position independent machine code is desired, so that the sequential controller can be executed from anywhere within the microprocessor's memory map.

A terminal display of discrete I/O and currently active states is necessary for debugging the control algorithm and monitoring the plant. However, once the control algorithm has been verified it is often desirable to disable the display so that a costly terminal is not required in the final design. This is particularly true of product oriented applications, such as sewing machines [20], vehicles [34], washing machines, or elevators.

5.2 STATE MACHINE AND STATE TABLE ARCHITECTURE

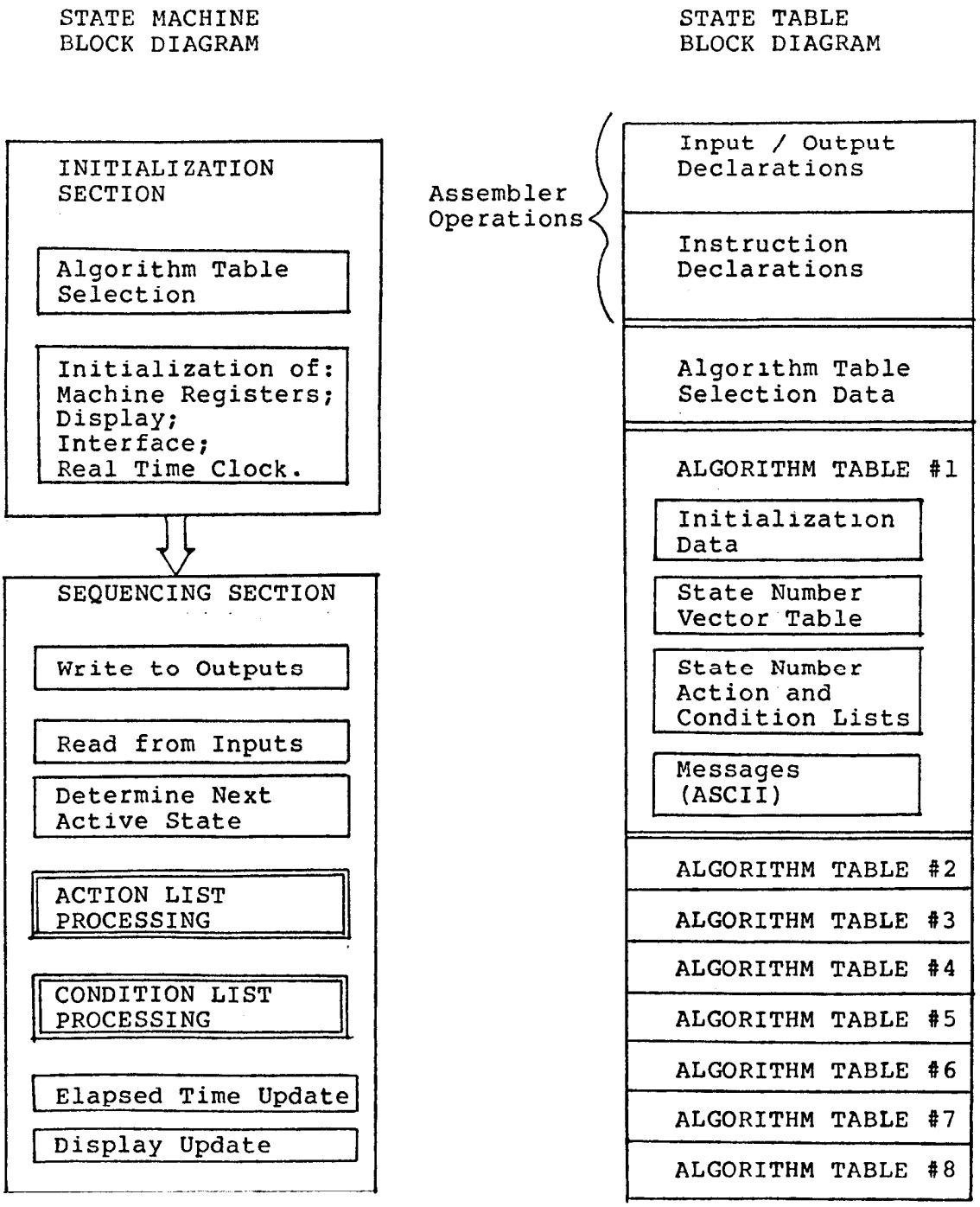
The state machine and state table architecture have been mentioned in several references as the best architecture for implementing a sequential controller based on state transition techniques [16] [20] [9] [23] [29]. However, no

references were found detailing an actual implementation of this architecture. The state machine is the sequential controller which operates on the state table to implement a particular control algorithm. Block diagrams of both the state machine and state table architectures presented in this thesis are shown in Figure 5.1.

The state machine consists of the following four sections: the initialization section; the sequencing section; the action list processing section; and the condition list processing section. The initialization section performs all actions required to initialize the state machine for a particular control application. The sequencing section performs the sequencing required to implement the control algorithm. Called as subroutines within the sequencing section are the action list and condition list processing sections. When a state has just been activated, action list processing occurs; otherwise condition list processing occurs with polling of specified conditions.

The state table contains the machine code required to select and implement several control algorithms. Each control algorithm is assembled into machine code to form an algorithm table, and several algorithm tables may be assembled together to form a state table. The algorithm table may be constructed from a state transition table,

Figure 5.1: Sequential Controller Architecture



Petri net, or state transition diagram representation of the control algorithm.

As discussed in Section 4.5, a state transition table can be easily constructed from either a state transition diagram or Petri net representation. Graphical representations often become too cumbersome for complex systems; however a state transition table can always be designed no matter how complicated the application. Therefore, it is best to use a state transition table format to generate the algorithm table. Graphical methods may be used to clarify an algorithm; however, to base sequential controller input data on a graphical method would limit the sequential controller's capabilities.

Global control may be implemented by the state machine if the capability exists for executing multiple active states. The state machine described in this thesis provides this capability, and the number of active states can be increased or decreased, just as token count can be increased or decreased for the Petri net. Therefore, the state machine as described is based on the token player concepts of the Petri net.

State machine registers are RAM locations required for operation of the state machine. These RAM locations are

used to represent flags, pointers, input and output data buffers, currently active states, and user registers. User registers are used for counters, timer alarms, and data manipulation. State machine registers are the only RAM locations required for a sequential controller implementation, since the state machine and state table are usually stored in ROM.

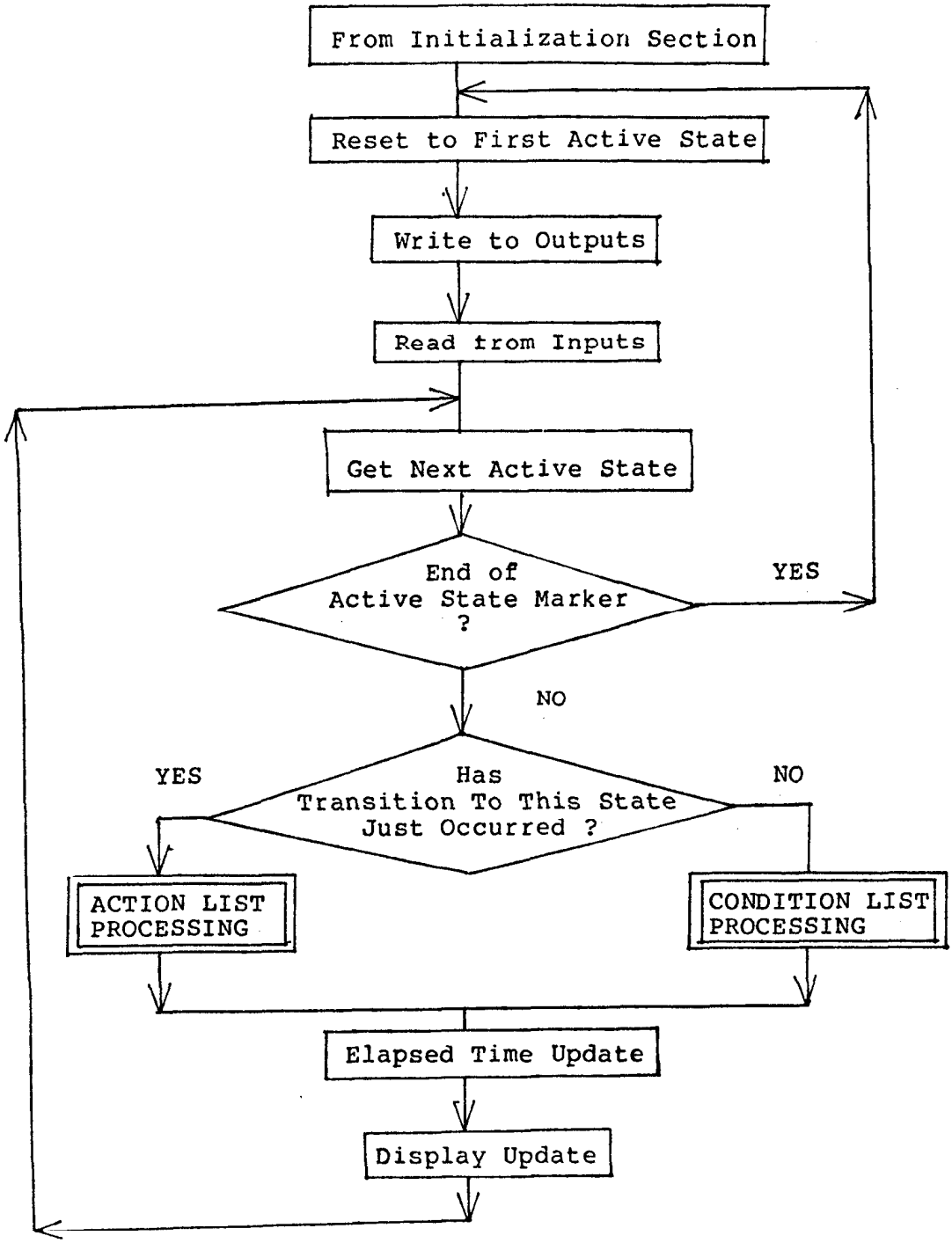
5.3 STATE MACHINE AND STATE TABLE OPERATION

The state table "input/output declarations" block, as shown in Figure 5.1, defines the I/O device associated with each discrete I/O bit. The state table "instruction declaration" block defines the operation codes for all action and condition functions. These blocks are used to define mnemonic labels when assembling a state table for a particular application.

Up to eight algorithm tables may be contained in the state table, with each algorithm table representing a sequential control algorithm for a different application. The state machine selects the algorithm table based on data contained in the state table or by user prompts entered on the terminal. Then the state machine registers, interface, display, and real time clock are initialized based on information contained in the chosen algorithm table.

Sequencing begins as the state machine enters the sequencing section. A flow chart of the sequencing operation is shown in figure 5.2. Outputs are activated as specified by the output data buffer, and input conditions are stored in the input data buffer. The next active state is determined by incrementing the pointer to the next active state register. If a transition to this active state has just occurred, the state's action list is processed; otherwise the state's condition list is processed. The location of the state's action list or condition list is determined from the state number vector table. Action list processing can manipulate outputs, increment counters, set timer alarms, increase the number of active states, or manipulate user registers. These actions must occur only once, when the state is entered. After an action list has been executed, transition conditions are checked every time this active state is called again by the state machine. These transition conditions may be based on inputs, timer alarms, counters, the value of user registers, or the number of other active states. After action or condition list processing has occurred for an active state and if appropriate flags are enabled, the elapsed time is updated and the display is updated. After all active states have been executed, inputs are read and outputs are activated, and the active

Figure 5.2: Sequencing Section Flow Chart



state pointer is reset to the first active state.

The state machine described in this thesis uses polling to check inputs. Polling was chosen over an interrupt-driven structure for several reasons. There is little else for the processor to do except check input conditions. The interrupt service routine would have the same polling structure as the sequencing section. It is desirable to minimize the external hardware requirements. The polling structure as described eliminates potential race conditions.

CHAPTER 6

SEQUENTIAL CONTROLLER IMPLEMENTATION

The sequential controller architecture described in Chapter 5 has been implemented on a microcomputer system based on the Motorola MC6809 microprocessor. This microcomputer was chosen because it was available and interfaces easily to peripherals [36]. The eight bit microprocessor system represents an inexpensive implementation which would be especially important for consumer product applications.

Discussed in this chapter are: the features of the sequential controller; memory and hardware considerations; descriptions of the initialization, sequencing, condition list processing and action list processing sections; and a description of the construction of the state table for a particular application.

6.1 SEQUENTIAL CONTROLLER FEATURES

The sequential controller design of this thesis can access 64 discrete I/O bits. Each bit can be programmed as an input or output, with input bits connected to on/off sensor signals and output bits connected to on/off actuator signals.

Up to 127 states can be programmed for each algorithm table. Initially state #1 alone is activated, however the number of active states can be expanded to 16 of the 127 states. With this multi-active state capability, global control can be implemented easily.

A status line is transmitted periodically to the terminal, which can display one of four formats. Format #1 displays the elapsed time and the 16 possible active states. Format #2 displays the elapsed time and the status of all 64 I/O bits. Format #3 displays the elapsed time, the status of the first 32 I/O bits, and the first eight active states. Format #4 is no status line display.

The terminal display and user prompts can be disabled by changing the first byte of the state table to a number between one and eight. This number tells the state machine which algorithm table to execute. It also allows the final design to be implemented without using a costly terminal.

The state table is constructed and assembled on a development system. After assembly the state table can either be downloaded into RAM from a cassette tape or programmed into EPROMs. If the state table is less than

2K bytes it may be executed from system RAM starting at hexadecimal address D200. The state table can always be executed from external memory starting at address 2800 hex using either ROM or RAM memory.

6.2 MEMORY REQUIREMENTS

The memory map for the sequential controller implementation of this thesis is shown in Table 6.1. All peripherals use memory mapped I/O and must be defined in Table 6.1.

The system monitor is based on Motorola ASSIST09 software [28] and is located at hexadecimal addresses E800 through FFFF. System peripherals are located between E000 and E7FF. System RAM memory, located from D000 through DFFF, contains: the system use RAM; hardware and software stacks; a 2K byte allocation for state table execution; and user and state machine registers. User registers are defined by the state table and are used for implementing counters, flags, timer alarms, and data storage. Table 6.2 shows the state machine register memory map. These RAM locations are required for state machine operation and represent flags, pointers, data input and output buffers, the real-time clock counter, active state numbers, and status line display ASCII characters.

Table 6.1: Memory Map of MC6809 System for Sequential Controller Implementation.

MEMORY ADDRESS	DESCRIPTION	MEMORY BLOCK
FFFF E000	System Monitor Program ASSIST09 Extended	System ROM
DFFF DF52	System Use	System RAM
DF51 DC00	Hardware Stack	
DBFF DA00	User Stack	
D9FF D200	State Table for 2K Byte RAM Execution	
D1FF D100	User Registers Defined by State Table	
D0FF D000	State Machine Registers	
CFFF 8000	Unaddressable Due To System Design	
7FFF 3000	State Table 20K Bytes Expansion Area	External Memory
2FFF 2800	State Table 2K Byte ROM Execution	
27FF 2000	State Machine	
1FFF 1000	Discrete I/O Ports Expandable to 64 I/O bits	PIA's
0FFF 0000	Other Hardware	PTM

Table 6.2: State Machine Register Memory Map.

MEMORY ADDRESS	DESCRIPTION
D0FF D0B0	Status Display Line (ASCII Characters)
D0AF D050	Not Allocated For Any Purpose
D04F D040	Active State Registers Contains the Number of Currently Active States
D03F D03A	Pointers: State Table and Active State
D035 D030	Real Time Clock Counters
D02F D028	Standby Data Output Information
D027 D020	Discrete Data Output Buffer
D01F D018	Data Direction Information
D017 D010	Discrete Data Input Buffer
D00F D004	Address Information
D003 D000	Flags

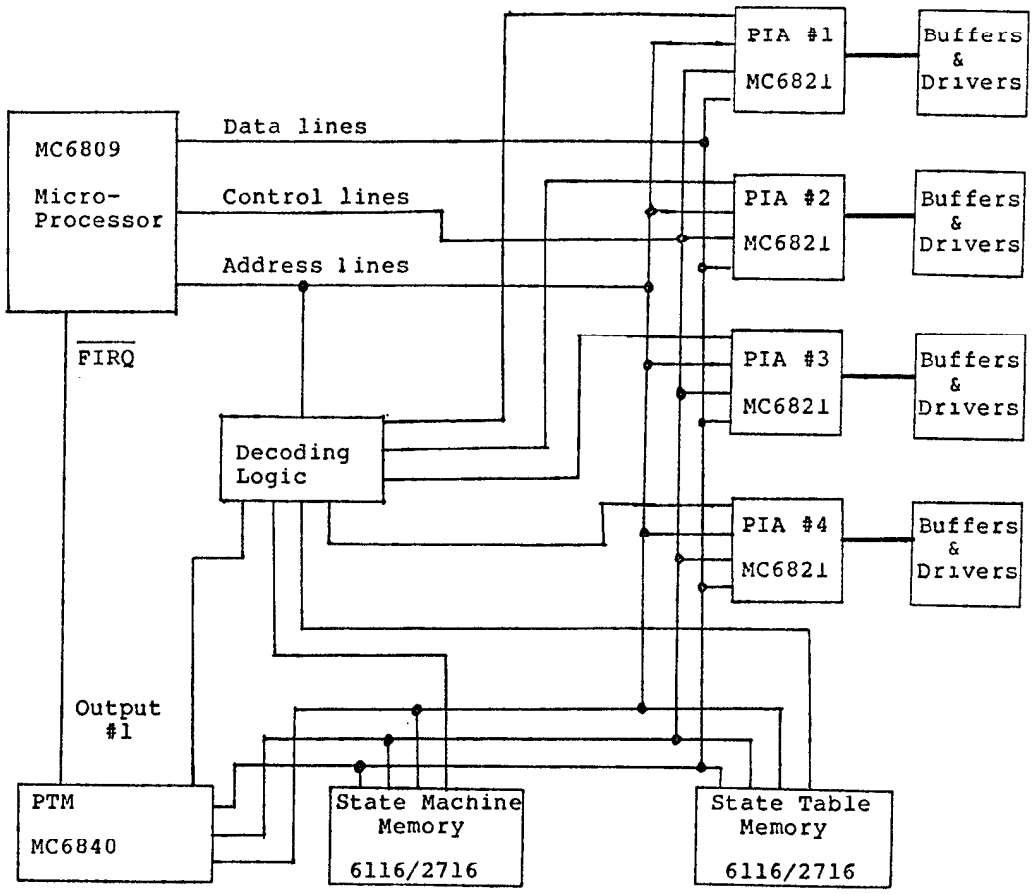
External memory and peripheral addresses are located at hexadecimal addresses 0000 through 7FFF. The state machine can be programmed into one 2716 EPROM chip, since it requires less than 2K bytes of memory. The state machine is located at addresses 2000 through 27FF; however, it is position independent and can be executed from anywhere in memory. The state table is expandable to 22K bytes when contained in external memory. It must always begin at external memory address 2800. Discrete I/O ports are located from 1000 through 1FFF. Other peripheral chips are located at addresses 0000 through 0FFF.

6.3 HARDWARE REQUIREMENTS

External hardware is required for the discrete I/O interface, external memory, generating the real time clock, and address decoding logic for each of these devices. A block diagram describing external hardware is shown in Figure 6.1.

The discrete I/O interface is implemented using four MC6821 PIAs (Peripheral Interface Adapters). Each PIA contains 16 discrete I/O bits, with each bit programmable as an input or output. Buffers and drivers are connected to the respective inputs and outputs of the PIAs to control peripheral devices. The realtime clock is generated using one timer of an MC6840 PTM (Programmable Timer Module).

Figure 6.1: External Hardware Block Diagram



The timer output is connected to the FIRQ (Fast Interrupt Request) pin of the MC6809 microprocessor. External memory may use either 6116 Static RAM or 2716 EPROM chips which are interchangeable since they have the same pin assignments. Address decoding for each of these devices is accomplished by using a 3 to 8 decoder and several NAND gates and inverters.

Descriptions of each of these peripheral chips can be found in references [27] and [1].

6.4 STATE MACHINE INITIALIZATION SECTION

The initialization section performs all actions required by the state machine prior to executing a control algorithm. The reader is referred to the Appendix A program listing of the state machine initialization section.

The program begins by resetting the external PTM chip. The location of the state table is determined by reading the external address at 2800 hex. The data bus will float high if no device is accessed. Should this location read FF hex, then no memory chip exists containing the state table and the state table is assumed to exist at D200 hex.

Eight algorithm tables are contained in each state table.

Algorithm selection occurs by reading the first byte of the state table. Should this number be a zero then algorithm table selection is performed by prompting the user via the terminal. Should this be a number 1 through 8 then the state table pointer points to the prescribed algorithm table, and the terminal display is disabled.

The real time clock, active state, and user registers are all zeroed, with the first active state register set to state one action list processing. Each state has both an action list and condition list. When the active state is read from an active state register, "bit 0" defines whether it is an action or condition list. This format is described in Figure 6.2. Note that only seven bits remain to describe the state numbers 1 through 127. A state number of zero marks the end of the active state. The input and output data buffers and state table pointers are initialized with information contained in the algorithm table.

The real time clock uses a one millisecond pulse from the external PTM chip to generate a fast interrupt request. The fast interrupt is used to minimize servicing time and to have a higher priority over normal interrupt requests. The interrupt service routine increments a millisecond counter and when it reaches 1000 it then resets the counter to zero

Figure 6.2: Data Formats For The Sequential Controller

ACTIVE STATE REGISTER FORMAT

```
Bit # 76543210
     ::::::::
     00000000 = End of Active States Marker
     xxxxxxx1 = Action List Processing
     xxxxxxx0 = Condition List Processing

                xxxxxxx = State Number (1 - 127)
```

ACTION LIST FORMAT

```
Bit # 76543210
     ::::::::
     00rrrrrr = Force Off Discrete Output
     01rrrrrr = Force On Discrete Output
     1aaaaaaa = Action List Function

                rrrrrr = Output Bit Number
                aaaaaaa = Action List Function
                          Operation Code
```

CONDITION LIST FORMAT

```
Bit # 76543210
     ::::::::
     00uuuuuu = Check for Off Discrete Input
     01uuuuuu = Check for On Discrete Input
     1ccccccc = Condition List Function

                uuuuuu = Input Bit Number
                ccccccc = Condition List Function
                          Operation Code
```

and sets a seconds flag. When all initialization is completed the sequencing section is entered.

6.5 STATE MACHINE SEQUENCING SECTION

The state machine sequencing section scans through each of the active states while polling inputs and activating outputs. The reader should refer to Appendix B for the program listing of the sequencing section. The flow chart for the sequencing section is given in Figure 5.2.

It is not possible to use the terminal output routines of the monitor, because hardware interrupts are disabled during execution. This would prevent the real time clock from operating. Therefore characters must be sent to the terminal one at a time during sequencing.

Sequencing begins by writing data contained in the data output buffer to the PIAs, and entering data from the PIAs into the data input buffer. The active state is determined by reading the active state register pointed to by the active state pointer. Should this state number be zero, the active state number pointer is initialized to the first active state register, and the PIAs are again accessed. Otherwise the pointer is simply incremented to the next active state register. Action list processing occurs if "bit 0" of the state number equals one, and

condition list processing occurs if "bit 0" equals zero.

After each state list has been executed, the system ACIA is checked to determine if a character has been sent. If a character has been sent, then the next character of the status line is sent. The seconds flag is then checked to determine if 1000 milliseconds has elapsed, and if this flag is set the real time clock registers are updated. After the real time clock has been updated, the display flag is checked to determine if the entire status line has been sent, and if it has then a new status line is generated. Control is then returned to the address where the active state register is read, and sequencing continues.

6.6 STATE MACHINE ACTION LIST PROCESSING

The action list processing section is called by the sequencing section to operate on state action lists contained in the state table. The program listing of the action list processing section is contained in Appendix C. The action list may include output manipulation and any of the state action list functions described in Table 6.3. The format of action list data is described in Figure 6.2.

Action list processing begins by reading a byte pointed at in the state table. If "bit 7" of this byte equals zero,

Table 6.3: State Action List Functions

MNEMONIC SYMBOL	OP. CODE	DESCRIPTION
LOADR	A0 hex	Loads a user register with immediate data. Operands (2): [Reg#] [Immediata data]
MOVER	A1 hex	Move data from first user register to second. Operands (2): [Reg#1] [Reg#2]
INCRG	A2 hex	Increment data in user register. Operands (1): [Reg#]
DECRG	A3 hex	Decrement data in user register. Operands (1): [Reg#]
ADDIM	A4 hex	Add immediate data to user register. Operands (2): [Reg#] [Immediate data]
SUBIM	A5 hex	Subtract immediate data to user register. Operands (2): [Reg#] [Immediate data]
ADDRG	A6 hex	Add two register values together storing sum in first. Operands (2): [Reg#1] [Reg#2]
SUBRG	A7 hex	Subtract second register from first storing difference in the first register. Operands (2): [Reg#1] [Reg#2]
TIMES	A8 hex	Set timer alarm. Note the data format and maximum value of operands. Operands (7): [Starting Reg#] [Hours:9000,900;BCD] [Hours:90,9;BCD] [Minutes:59;BCD] [Seconds:59;BCD] [mSeconds:03;HEX] [mSeconds:E7;HEX]
EXPAC	A9 hex	Expands the number of active states. Operands (Variable): [# of States Expanded] [State #] ...
HLTSQ	AA hex	Halt sequencing, outputs to standby, message. Operands (1): [Message #]
EXTFN	AB	Transfers control to State Table via SWI2.
ACNOP	AC	No operation.
ACEND	AD	Marks the End of the action list.

then a discrete output bit is turned on or off dependent on the value of "bit 6". If "bit 7" equals one, an action list function is called as specified in Table 6.3. After this action list command is executed, control is returned to the beginning of action list processing and the next byte is read from the state table.

Each of the state action list fuctions of Table 6.3, is specified by an operation code which is decoded by the action list processing section. Operands may also be required by functions and they are contained in the bytes of the state table immediately following the operation code of the function. The functions LOADR, MOVER, INCRG, DECRG, ADDIM, SUBIM, ADDRG, and SUBRG are all user register/arithmetic functions. The function TIMES is used to set timer alarms using six consecutive user registers. This function adds the contents of six operands representing the alarm time to the real time clock registers and stores the sum in six consecutive user registers specified by an operand. The function EXPAC is used to expand the number of active states. The number of active states to be expanded and the new active states are contained in the operand list, and each of these new states is entered in the active state registers. HLTSQ terminates sequencing of the state machine, returns outputs to their standby mode, transmits a specified

message to terminal, and returns control of the microcomputer to the system monitor. HLTSQ is used to respond to a fault when detected and it is the only function that will terminate sequencing of the state machine. The fault is identified for the user by the terminal message. When a function is required for an application and not contained in the state machine action list, it may be executed as machine code from the state table by using the EXTFN function. All external functions contained in the state table must end with a "Return From Interrupt" command. ACNOP is a no-operation command, and it is used in the development of a state table to occupy bytes. All state action lists must end with an ACEND command. The function ACEND changes "bit 0" of the Active State Register to a zero, which causes condition list testing to occur until transition conditions are satisfied. ACEND returns control to the sequencing section.

6.7 STATE MACHINE CONDITION LIST PROCESSING

The condition list processing section is called by the sequencing section to operate on state condition lists contained in the state table. The program listing of the condition list processing section is contained in Appendix D. The condition list may include input checking and any of the state condition list functions described in Table 6.4.

The format of condition list data is described in Figure 6.2.

Condition list processing begins by enabling a pass flag. Then the byte as selected by the pointer for the state table is read. If "bit 7" of this byte equals zero, then a discrete input bit is checked for on or off dependent on the value of "bit 6". If "bit 7" equals one, a condition list function is called as specified in Table 6.4. The pass flag is disabled if either a condition function or the discrete input condition is false. Then the state table pointer is incremented and control is returned to the address where the next byte is read.

The functions CMPGE, CMPGT, CMPEQ, and CMPNE are all register to register comparisons; and CIMGE, CIMGT, CIMEQ, CIMNE, CIMLE, and CIMLT are all register to immediate data comparisons. The operands for these functions specify the register or immediate data to be compared. The function TIMET directly supports the TIMES function of the action list. TIMET tests to see if an alarm condition occurs and disables the pass flag if it has not. A transition may occur if either of two sets of conditions are true. The BLKOR function is used to separate two condition blocks, and if either is true the transition to a new state occurs. The TRANS function marks the end of a condition

Table 6.4: State Condition List Functions

MNEMONIC SYMBOL	OP. CODE	DESCRIPTION
CMPGE	80 hex	Compares two registers and checks that the first is greater than or equal to the second. Operands (2): [Reg#1] [Reg#2]
CMPGT	81	Checks for greater than. Operands (2)
CMPEQ	82	Checks for equal. Operands (2)
CMPNE	83	Checks for not equal. Operands (2)
CIMGE	84 hex	Compares a register with immediate data and checks that the register is greater than or equal to the immediate data. Operands (2): [Reg#] [Immediate data]
CIMGT	85	Checks for greater than. Operands (2)
CIMEQ	86	Checks for equal. Operands (2)
CIMNE	87	Checks for not equal. Operands (2)
CIMLE	88	Checks for less than or equal. Operands (2)
CIMLT	89	Checks for less than. Operands (2)
TIMET	8A hex	Checks the consecutive registers for alarm. Operands (1): [First Reg#]
BLKOR	8B hex	Marks the end of a conditions block which is to be OR'ed with next condition Block
TRANS	8C hex	Marks the end of a condition block and if true transition occurs to the new state. Operands (1): [New State#]
CONAC	8D	Checks to see if specified states are active and if true contracts number of active states and transition occurs to new state. Operands (Variable): [Total Active States To Be Contacted] [State#]... [New State#]
CNNOP	8E	No operation.
CNEND	8F	Marks the End of the condition list.

block. If all transition conditions are true, the active state number is changed to the number specified in the operand, and action list processing will occur on the next sequencing for this state. CONAC examines the state numbers contained in consecutive active state registers, and if they match the state numbers contained in the operands, the total number of states is contracted and transition occurs to a new state. The CNNOP function results in no-operation and it is used to occupy bytes in a state table. All state condition lists must end with the CNEND command, which returns control to the sequencing section.

6.8 CONSTRUCTING THE STATE TABLE

The state table is constructed and assembled on a development system. Using the assembler pseudo-instructions, the source listing of the state table can be as readable as the state transition table discussed in Section 4.3. The state table constructed in this section is for the semi-automatic drill press example described in Section 2.3. All state table data contained in the figures to follow represent actual state table entries for this example. The reader is urged to compare the constructed state table in the figures to follow with the state transition technique descriptions shown in Table 4.1, Figure 4.1, and Figure 4.2. The source listing of

the state table, for the three station automated drilling system example discussed in Section 4.4, is contained in Appendix E.

The first step in setting up a state table is to declare which discrete I/O ports are allocated to each device. This is accomplished in the I/O Declaration Block as shown in Figure 6.3, for the example of Section 2.3. The data format described in Figure 6.2 must be followed, therefore 64 must be added to each port number when describing the port in the ON state, so that "bit 6" will be set to one. The pseudo-instruction EQU equates a symbolic label to a number. EQU is also used to equate the operation codes with the mnemonic symbols of the action and condition list functions as shown in Figure 6.4.

The state table data required by the initialization part of the state machine is shown in Figure 6.5, for the example of Section 2.3. Pseudo-instructions used by this section are FDB (Fill Double Byte); ASC (Generate ASCII Characters); HEX (Number is in Hexadecimal); and FCB (Fill Constant Byte). The first byte of the state table is 0 if a terminal display with user prompts is required, or 1 through 8 if the state table is to be selected directly and the terminal display disabled. The next 16 bytes of the state table are the absolute addresses of each

Figure 6.3: State Table Input / Output Declarations
For Semi-Automatic Drill Press Example

```
*
***** I/O DECLARATION BLOCK   (USER DEFINABLE) *
*
EXTGRP_OFF          EQU                 0
EXTGRP_ON           EQU                 64
LWRDRL_OFF          EQU                 1
LWRDRL_ON           EQU                 65
RASDRL_OFF          EQU                 2
RASDRL_ON           EQU                 66
RTRGRP_OFF          EQU                 3
RTRGRP_ON           EQU                 67

PCGRP_OFF           EQU                 4
PCPRP_ON            EQU                 68
DRLDN_OFF           EQU                 5
DRLDN_ON            EQU                 69
DRLUP_OFF           EQU                 6
DRLUP_ON            EQU                 70
PCFRE_OFF           EQU                 7
PCFRE_ON            EQU                 71

START_OFF           EQU                 8
START_ON            EQU                 72
STOPS_OFF           EQU                 9
STOPS_ON            EQU                 73
```

Figure 6.4: State Table Action and Condition List Functions Declarations For Semi-Automatic Drill Press Example

```
*
***** ACTION LIST FUNCTIONS DECLARATION BLOCK *
*
LOADR                EQU                A0H
MOVER                EQU                A1H
INCRG                EQU                A2H
DECRG                EQU                A3H
ADDIM                EQU                A4H
SUBIM                EQU                A5H
ADDRG                EQU                A6H
SUBRG                EQU                A7H
TIMES                EQU                A8H
EXPAC                EQU                A9H
HLTSQ                EQU                AAH
EXTFN                EQU                ABH
ACNOP                EQU                ACH
ACEND                EQU                ADH
*
***** CONDITION LIST FUCTIONS DECLARATION BLOCK *
*
CMPGE                EQU                80H
CMPGT                EQU                81H
CMPEQ                EQU                82H
CMPNE                EQU                83H
CIMGE                EQU                84H
CIMGT                EQU                85H
CIMEQ                EQU                86H
CIMNE                EQU                87H
CIMLE                EQU                88H
CIMLT                EQU                89H
TIMET                EQU                8AH
BLKOR                EQU                8BH
TRANS                EQU                8CH
CONAC                EQU                8DH
CNNOP                EQU                8EH
CNEND                EQU                8FH
```

Figure 6.5: State Table Initialization Section
For Semi-Automatic Drill Press Example

```
*************************************************
STBEG             HEX               0
*
***** ALGORITHM TABLE STARTING ADDRESSES *
*
                  FDB               AT1
                  FDB               0
                  FDB               0
                  FDB               0
                  FDB               0
                  FDB               0
                  FDB               0
                  FDB               0
*
***** SELECTION MESSAGE *
*
                  HEX               1A
                  ASC    "PLEASE SELECT ONE OF THE FOLLOWING"
                  HEX               0A,0A,0D
                  ASC    "     1 = Semi-Automatic Drill Press"
                  HEX               0A,0D
                  ASC    "     2 = None "
                  HEX               0A,0D,04
*
***** ALGORITHM TABLE # 1 *
*
AT1               HEX               10,00    ;PIA Start Addr.
                  FCB               2        ;Total PIA's Used

                  HEX               F0,00    ;Unit 1 DDR/OR
                  HEX               00,00    ;Unit 2 DDR/OR

                  FCB               4        ;Display Format #
                  FCB               0        ;Heading Message#
                  FDB               MESG     ;Message Vec. Tab.
                  FDB               EXTF     ;Ext. Func. Addr.
```

algorithm table. Should no algorithm table exist for a number, zero must be entered. A selection message follows which is called when the user is prompted to select a state table. This message may be of any length, and it is terminated with the byte 04 hex. The algorithm table is then entered with the first three bytes representing the starting address of the PIA I/O Interface and the number of 8-bit ports required for the application. The next group of bytes contains the data direction and standby output information used to initialize the I/O interface. Following this is a byte representing the single line status display format number as described in Section 6.5. The sequence heading message number is contained in the next byte and the address of the message vector table follows. The address of the external functions routine is the last data to be required of the initialization section of the state machine.

The state table sequencing section is shown in Figure 6.6, for the example of Section 2.3. This section contains information required to implement the control algorithm as described by a state transition table. The state number vector table contains the addresses of all action and condition lists contained in the algorithm table. The symbolic label may be used with the FDB pseudo-instruction as shown. The state action and condition lists are

Figure 6.6: State Table -- Sequencing Section
For Semi-Automatic Drill Press Example

```
*
***** STATE NUMBER VECTOR TABLE *
*
                    FDB                 S1C
                    FDB                 S1A
                    FDB                 S2C
                    FDB                 S2A
                    FDB                 S3C
                    FDB                 S3A
                    FDB                 S4C
                    FDB                 S4A
                    FDB                 S5C
                    FDB                 S5A
                    FDB                 S6C
                    FDB                 S6A
*
***** STATE ACTION AND CONDITION LISTS *
*
S1A                 FCB                 RTRGRP_OFF
                    FCB                 ACEND
S1C                 FCB                 START_ON
                    FCB                 TRANS,2
                    FCB                 STOPS_ON
                    FCB                 TRANS,6
                    FCB                 CNEND

S2A                 FCB                 EXTGRP_ON
                    FCB                 ACEND
S2C                 FCB                 PCGRP_ON
                    FCB                 TRANS,3
                    FCB                 STOPS_ON
                    FCB                 TRANS,6
                    FCB                 CNEND

S3A                 FCB                 EXTGRP_OFF,LWRDRL_ON
                    FCB                 ACEND
S3C                 FCB                 DRLDN_ON
                    FCB                 TRANS,4
                    FCB                 STOPS_ON
                    FCB                 TRANS,6
                    FCB                 CNEND
```

(State Table Continued On Next Page)

Table 6.6: State Table -- Sequencing Section (Continued)

```
S4A             FCB             LWRDRL_OFF,RASDRL_ON
                FCB             ACEND
S4C             FCB             DRLUP_ON
                FCB             TRANS,5
                FCB             STOPS_ON
                FCB             TRANS,6
                FCB             CNEND

S5A             FCB             RASDRL_OFF,RTRGRP_ON
                FCB             ACEND
S5C             FCB             PCFRE_ON
                FCB             TRANS,1
                FCB             STOPS_ON
                FCB             TRANS,6
                FCB             CNEND

S6A             FCB             RTRGRP_OFF,RASDRL_OFF
                FCB             LWRDRL_OFF,EXTGRP_OFF
                FCB             ACEND
S6C             FCB             STOPS_OFF
                FCB             TRANS,1
                FCB             CNEND
*
***** EXTERNAL FUNCTIONS *
*
EXTF            RTI
*
***** MESSAGE VECTOR TABLE *
*
MESG            FDB             MESG0
*
***** MESSAGES *
*
MESG0           ASC    "To Begin Drilling Push Start"
                HEX             0A,0D,04
```

assembled using the labels defined in the declaration sections to represent I/O ports and action and condition list functions. List function operands may be appended to the function line and separated by commas.

Following the state number action and condition lists is the external functions block. An assembly language program may be contained here which is called by the EXTFN command of the action list processing section. The program must end with a RTI (Return from Interrupt) instruction. A message vector table and message ASCII character data are the last entries of the algorithm table. Each message is terminated by the hexadecimal byte 04.

Up to eight algorithm tables may be included with each state table.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

The primary objectives of this thesis were to demonstrate the applicability of state transition techniques to sequential control and to design a sequential controller based on the state machine and state table architecture.

7.1 CONCLUSIONS

State transition techniques are far superior to relay ladder logic methods for synthesis and design of sequential control. State transition techniques describe the sequential behavior of the system explicitly, are highly structured, force good record keeping, and can be easily modified and debugged. Partitioning of the control algorithm can be easily accomplished, and fault detection and diagnostics can be easily incorporated in the design. The two graphical methods discussed (the state transition diagram and the Petri net) are well suited for describing the sequential behavior of simple systems, but they become cumbersome for describing the sequential behavior of complex systems. The state transition table is the best method for describing the general case sequential control problem. Global control requirements are easily described by using either Petri net or state transition table

representations.

The sequential controller architecture discussed in this thesis is based on the state machine and state table concept. This architecture has the advantages of compact application code and high speed processing. Only those conditions required for transition to a new system state are polled by the sequential controller.

The sequential controller design of this thesis was implemented on a Motorola 6809 based microcomputer system. The state machine was programmed in assembly language to generate compact machine code with minimal execution time. The state table contains the data describing up to eight applications and is assembled on a development system. The control algorithm for each application is assembled from a state transition table format. Each control algorithm may contain up to 127 states, 64 discrete I/O ports, and 16 simultaneously active states.

The source listing of the state table is very explicit and readable for describing the sequential behavior of the system. Only two hours were required to assemble and debug the state table shown in Appendix E for the three station automated drilling system example. The machine code is very compact and requires minimal processing by the state

machine. Fault diagnostics can be incorporated in the state table by using one active state to monitor possible error conditions. Global control can be implemented in the state table by using multiple active states. The state table may be located in either RAM or ROM.

It is not possible to compare the actual response times of a programmable controller with the sequential controller presented in this thesis, because of the differences in architecture. The programmable controller response time is dependent on the size of the program. All conditions in the program are checked regardless of the current state of the system. For the sequential controller, only those conditions which are required for a transition to occur are checked. The sequential controller response time is also dependent on the system clock frequency which will vary for other microprocessor systems. For a large sequential control algorithm, the sequential controller is expected to be several hundred times faster than currently available programmable controllers [29].

7.2 RECOMMENDATIONS FOR FUTURE WORK

This thesis represents the first step in developing improved methods for implementing sequential control. Described in this section are recommendations for future research work.

Should a sequential controller need more than the 127 available states it is suggested that the same architecture be implemented on a 16-bit microprocesssor. Using the same concepts described in this thesis, the 16-bit machine could implement up to 32,767 states, and have 16,384 discrete I/O ports.

The sequential controller architecture could also be implemented on a bit-slice bipolar microprocessor. This technology has the advantage of high speed processing with clock frequencies typically ten times faster than MOS microprocessor technology. The state machine should be programmed using micro-instructions, and condition and action list functions should be programmed using macro-instructions.

The sequential controller implementation described in this thesis requires a development system to construct and assemble the state table. This implementation could be directly applied to products, but to be a direct replacement for the programmable controller, a state table assembler must be developed which can be implemented directly on the microcomputer system. The state table assembler should be user interactive, and a CRT controller chip should be interfaced to provide better graphical

capabilities for both the development and monitoring of control algorithms. It is suggested that the state table assembler be implemented using Pascal to minimize development time over an assembly language implementation.

Distributed control through multi-processing is of great interest for implementing large control algorithms, to reduce hardware costs and increase throughput. State machine interaction and communication paths are the primary concerns of distributed control. Additional controller functions and data highway bus hardware will be required to implement a multi-processing system.

Appendix A Program Listing Initialization Section

```
"5809"

*************************************************
*      GRADUATE THESIS                          *
*      State Machine                            *
*      Initialization Section                   *
*                                               *
*      AUTHOR: Robert M. Laurie                 *
*      DATE: February 9, 1986                   *
*                                               *
*************************************************


*
***** STATE MACHINE: MEMORY ALLOCATION   2000H - 27FFH *
*
***** STATE TABLE: MEMORY ALLOCATION  2800H - 5FFFH *
*               This area is used when state table is stored in ROM.
*
*
***** RAM STORAGE LOCATIONS  D000H - DFFFH *
*               DF50 - DFFF: Reserved for System Use.
*               DC00 - DF51: Reserved for Hardware Stack.
*               DA00 - DBFF: Reserved for User Stack.
*               D200 - D9FF: STATE TABLE (Located Here to Allow for Downloading).
*               D000 - D1FF: STATE MACHINE Registers.

FLGPRT          EQU             00H             ;Print enable flag.
FLGSEC          EQU             01H             ;Seconds flag.
FLGPAS          EQU             02H             ;Conditional pass flag.
FLGDIS          EQU             03H             ;Display flag.

FORNU           EQU             07H             ;Display format number.
DISAD           EQU             08H             ;Current display address.
MESAD           EQU             0AH             ;Message address look up table.
PIAAD           EQU             0CH             ;PIA starting address.
PIANU           EQU             0EH             ;Number of 8-bit PIA units required.

DATIN           EQU             0D010H          ;Data input buffer.
DATDDR          EQU             0D018H          ;Data direction information.
DATOUT          EQU             0D020H          ;Data output buffer.
SDATOUT         EQU             0D028H          ;Standby data output information.

STIME           EQU             0D030H
TIMERHH         EQU             30H             ;Hour timer high byte (BCD).
```

```
TIMERHL         EQU             31H             ;Hour timer low byte  (BCD).
TIMERM          EQU             32H             ;Minute timer (BCD).
TIMERS          EQU             33H             ;Second timer (BCD).
TIMERmS         EQU             34H             ;mSecond timer (16-bit up counter).

STTBAS          EQU             3AH             ;State table pointer base address.
ACTBAS          EQU             3CH             ;Active state register base address.
ACTPNT          EQU             3EH             ;Active state register pointer.

SACT            EQU             0D040H          ;Start of active state registers.
EACT            EQU             0D04FH          ;End of active state registers.

SDIS            EQU             0D0B0H          ;Start of display buffer (ASCII).
EDIS            EQU             0D0FFH          ;End of display buffer (ASCII).

SREG            EQU             0D100H          ;Start of user registers.
EREG            EQU             0D1FFH          ;End of user registers.

USTKPNT         EQU             0DC00H          ;Starting address of user stack.

STRAM           EQU             0D200H          ;Starting address of state table located in RAM.
STROM           EQU             02800H          ;Starting address of state table located in ROM.
*
***** SOFTWARE INTERRUPT ROUTINES *
*
INCHP           EQU             0               ;Input character from terminal.
OUTCH           EQU             1               ;Output character to terminal.
PDATA1          EQU             2               ;Send string to terminal.
PDATA           EQU             3               ;Send new line and string to terminal.
MONITR          EQU             8               ;Go to ASSIST09 Startup.
VCTRSW          EQU             9               ;Vector Swap.
BRKPT           EQU             10              ;User Breakpoint.
FIRQX           EQU             10              ;FIRQ Vector Number.
SWI2X           EQU             8               ;SWI2 Vector Number.
*
***** PERIPHERAL MEMORY LOCATIONS *
*
PTMCR1          EQU             0000H           ;External PTM Control Register 1.
PTMCR2          EQU             0001H           ;External PTM Control Register 2.
PTMLT2          EQU             0004H           ;External PTM Latch 2.
ACIACR          EQU             0E008H          ;System ACIA Control Register.
ACIASR          EQU             0E008H          ;System ACIA Status Register.
ACIATD          EQU             0E009H          ;System ACIA Transmit Data Register.
*
***** EXTERNAL / GLOBAL LABELS *
```

```
*
                EXT             SEQUENCER       ;Starting address of sequencer section.

                GLB             FLGPRT, FLGSEC, FLGPAS, FLGDIS
                GLB             FORNU, MESAD, DISAD, PIAAD, PIANU
                GLB             STIME,TIMERmS,TIMERS,TIMERM,TIMERHL,TIMERHH
                GLB             ACTPNT, ACTBAS, STTBAS, SACT, EACT
                GLB             DATIN, DATOUT, SDATOUT, DATDDR
                GLB             SDIS, EDIS, SREG, EREG
                GLB             PDATA, PDATA1, MONITR, BRKPT
                GLB             ACIACR, ACIASR, ACIATD

************************************************
****        PROGRAM LISTING                 ****
************************************************

                PROG

************************************************
*        INITIALIZATION SECTION                *
************************************************
*
***** PTM - TIMER CHIP RESET *
*
START           LDA             #01H
                STA             >PTMCR2         ;Access control register 1.
                STA             >PTMCR1         ;Reset PTM.
                CLRA
                STA             >PTMCR1         ;Enable PTM.
*
***** INITIALIZATION *
*
                LDA             #0D0H
                TFR             A,DP            ;Initialize base page register.

                LDA             >STROM
                CMPA            #0FFH           ;Is state table located in ROM?
                BEQ             BINI1           ;NO.
                LDY             #STROM          ;YES, load ST pointer with ROM starting address.
                BRA             BINI2
BINI1           LDY             #STRAM          ;Load ST pointer with starting address for RAM.
BINI2           LDA             Y               ;Get table access number.

                STA             <FLGPRT         ;Initialize Print Flag, and is output enabled?
                BNE             BREC3           ;NO, output disabled.
```

```
*
***** SEND INITIAL MESSAGE TO TERMINAL *
*
                LEAX            11H,Y           ;YES, point to start of fire up message.
                SWI
                FCB             PDATA           ;Call output routine.
*
***** RECEIVE CHARACTER FROM TERMINAL AND DECODE *
*
                SWI
                FCB             INCHP           ;Receive character from terminal.

                CMPA            #"1"            ;Is character = 1?
                BEQ             BREC3           ;YES, algorithm table 1 selected.

                CMPA            #"2"            ;NO, is character = 2?
                BEQ             BREC3           ;YES, algorithm table 2 selected.

                CMPA            #"3"            ;NO, is character = 3?
                BEQ             BREC3           ;YES, algorithm table 3 selected.

                CMPA            #"4"            ;NO, is character = 4?
                BEQ             BREC3           ;YES, algorithm table 4 selected.

                CMPA            #"5"            ;NO, is character = 5?
                BEQ             BREC3           ;YES, algorithm table 5 selected.

                CMPA            #"6"            ;NO, is character = 6?
                BEQ             BREC3           ;YES, algorithm table 6 selected.

                CMPA            #"7"            ;NO, is character = 7?
                BEQ             BREC3           ;YES, algorithm table 7 selected.

                CMPA            #"8"            ;NO, is character = 8?
                BEQ             BREC3           ;YES, algorithm table 8 selected.

                LEAX            ABORT1,PCR      ;NO, point to ABORT1 message.
                SWI
                FCB             PDATA           ;Transmit ABORT1 message.
BREC1           LEAX            ABORT2,PCR      ;Point to ABORT2 message.
                SWI
                FCB             PDATA           ;Transmit ABORT2 message.
BREC2           SWI
                FCB             MONITR          ;Return to monitor.
```

```
BREC3           ANDA            #0FH            ;Decode from ASCII.
                ASLA
                DECA                            ;Generate offset.
*
***** ACCESS SELECTED ALGORITHM TABLE *
*
                LDY             A,Y             ;Point to the algorithm table selected.
                                                ;Is algorithm table generated?
                BNE             BACT1           ;YES.
                TST             <FLGPRT         ;NO, is print flag disabled?
                BNE             BREC2           ;YES.
                LEAX            ABORT3,PCR      ;NO, point to ABORT3 message.
                SWI
                FCB             PDATA           ;Transmit ABORT3 message.
                BRA             BREC1
*
***** TIMER AND REGISTER INITIALIZATION *
*
BACT1           LDX             #STIME          ;Point to starting address for initialization.
BACT2           CLR             X+              ;Clear pointed to address.
                CMPX            #EREG           ;Has end address been cleared?
                BLE             BACT2           ;NO.
                LDA             #03H            ;YES.
                STA             >SACT           ;Set first active step to step 1 action.
*
***** PIA INITIALIZATION *
*
BPIA1           LDX             Y++             ;Get PIA starting address.
                LDU             #DATOUT         ;Point to data output buffer.
                LDA             Y+              ;Get total number of bytes of I/O required.
                STX             <PIAAD          ;Store PIA starting address.
                STA             <PIANU          ;Store total number of bytes of I/O.

BPIA2           CLRB
                STB             1,X             ;Access PIA data direction register.
                LDB             Y+
                STB             X               ;Load PIA data direction register.
                STB             -8,U            ;Store data direction information.
                LDB             #4
                STB             1,X             ;Access PIA output register.
                LDB             Y+
                STB             X++             ;Load PIA output register.
                STB             8,U             ;Store standby output configuration.
                STB             U+              ;Load data output buffer.
                DECA                            ;Have all PIA's been initialized?
```

```
                BNE             BPIA2           ;NO.
*
***** PRINT HEADING ON TERMINAL *
*
                LDA             Y+              ;YES, get display format number.
                STA             <FORNU          ;Store display format number.
                LDA             Y+              ;Get heading message number.
                LDX             Y               ;Get message base address.
                STX             <MESAD          ;Store message base address.
                ASLA
                LDX             A,X             ;Point to first character of message.

                TST             <FLGPRT         ;Is print flag enabled?
                BNE             BVAR1           ;NO.
                SWI
                FCB             PDATA1          ;YES, transmit heading.

                LEAX            MESG1,PCR       ;Point to MESG1 message.
                SWI
                FCB             PDATA1          ;Transmit message.
*
***** READY FOR SEQUENCING? *
*
BRED1           SWI
                FCB             INCHP           ;Receive character from terminal.
                CMPA            #"S"            ;Is character = S ?
                BNE             BRED1           ;NO.
*
***** VARIABLE INITIALIZATION *
*
BVAR1           LDU             #USTKPNT        ;YES, initialize user stack pointer.

                LDX             #SACT           ;)
                LEAX            -1,X            ;)) Initialize active state register
                STX             <ACTBAS         ;)       base address.

                LDX             2,Y             ;)
                LDA             #SWI2X          ;)
                SWI                             ;)) External functions called by SWI2.
                FCB             VCTRSW          ;)
                                                ;)

                STY             <STTBAS         ;Store algorithm table base address.

                LEAX            FIRQISR,PCR     ;]
```

```
                LDA             #FIRQX          ;)
                SWI                             ;)) When FIRQ is generated call
                FCB             VCTRSW          ;)       interrupt service routine.
                                                ;)

                LDA             #01H
                STA             <FLGDIS         ;Disable display flag.
                LDX             #SDIS
                STX             <DISAD          ;Point to start of display buffer.

                LDA             #03H
                STA             >ACIACR         ;Reset ACIA.
                LDA             #49H
                STA             >ACIACR         ;Initialize ACIA.

                LDA             #0DH
                STA             >ACIATD         ;Transmit Carriage Return.
*
***** REAL TIME CLOCK CONFIGURATION *
*
                LDA             #86H            ;)
                STA             >PTMCR2         ;)
                LDD             #1B1FH          ;)) Configure Timer #2 of PTM for
                STD             >PTMLT2         ;)       1 mSecond pulsed output.
                CLR             <FLGSEC         ;)

                ANDCC           #00H            ;Enable interrupts.


                LBRA            SEQUENCER       ;Begin sequencing.


************************************************
*     INTERRUPT SERVICE ROUTINE  -  FIRQISR    *
************************************************
*
*               This routine increments the milliseconds count of the
*               real time clock with each interrupt on the FIRQ input.
*               When the counter resets to zero the seconds flag is enabled.

FIRQISR         PSHS            D               ;Push used register.
                LDD             <TIMERmS        ;Get mSecond counter.
                ADDD            #1              ;Increment counter.
                CMPD            #1000           ;Is counter = 1000?
                BLT             BFRQ1           ;NO.
```

```
                INC             <FLGSEC         ;YES, increment seconds flag.
                LDD             #0              ;Reset counter.
BFRQ1           STD             <TIMERmS        ;Store counter.
                PULS            D               ;Pull used register.
                RTI                             ;Return.

************************************************
*        MESSAGES                              *
************************************************

ABORT1          ASC             "YOU MUST CHOOSE 1 - 8 ONLY"
                HEX             04
ABORT2          ASC             "YOU ARE RETURNED TO THE MONITOR"
                HEX             04
ABORT3          ASC             "NO ALGORITHM TABLE HAS BEEN GENERATED FOR THIS NUMBER"
                HEX             04
MESG1           ASC             "TO BEGIN SEQUENCING ENTER [S]"
                HEX             04
```

Appendix B Program Listing Sequencing Section

```
"6809"

*************************************************
*      GRADUATE THESIS                          *
*      State Machine                            *
*      Sequencer Section                        *
*                                               *
*      AUTHOR: Robert M. Laurie                 *
*      DATE: February 9, 1986                   *
*                                               *
*************************************************

*
***** EXTERNAL / GLOBAL LABELS *
*
                GLB             SEQUENCER

                EXT             ACTPROC         ;Action List Processing.
                EXT             CONPROC         ;Conditional List Processing.

                EXT             FLGPRT, FLGSEC, FLGPAS, FLGDIS
                EXT             FORNU, MESAD, DISAD, PIAAD, PIANU
                EXT             STIME,TIMERmS,TIMERS,TIMERM,TIMERHL,TIMERHH
                EXT             ACTPNT, ACTBAS, STTBAS, SACT, EACT
                EXT             DATIN, DATOUT, SDATOUT, DATDDR
                EXT             SDIS, EDIS, SREG, EREG
                EXT             PDATA, PDATA1, MONITR, BRKPT
                EXT             ACIACR, ACIASR, ACIATD

*************************************************
*        SEQUENCER SECTION                      *
*************************************************
                PROG
*
***** SEND OUTPUT DATA TO PIA'S *
*
SEQUENCER       LDX             <PIAAD          ;)
                LDA             <PIANU          ;)) Load registers for PIA write data.
                LDY             #DATOUT         ;)

BSND1           LDB             Y+              ;Get data from DATOUT buffer.
                STB             X++             ;Load data into PIA.
                DECA                            ;Have all PIA's been loaded?
                BNE             BSND1           ;NO.
```

```
*
***** RECEIVE INPUT DATA FROM PIA *
*
                LDX             <PIAAD          ;)
                LDA             <PIANU          ;)) YES, load registers for PIA read data.
                LDY             #DATIN          ;)

BRCV1           LDB             X++             ;Get data from PIA.
                STB             Y+              ;Store data in DATIN buffer.
                DECA                            ;Have all PIA's been read?
                BNE             BRCV1           ;NO, read the next.
*
***** POINT TO THE ACTIVE STATE *
*
                CLR             <ACTPNT         ;YES, reset active step pointer to zero.
BPNT1           INC             <ACTPNT         ;Increment pointer for next active step.
                LDA             <ACTPNT         ;Get pointer.
                LDX             <ACTBAS         ;Get pointer base address.
                LDB             A,X             ;Get the active step number.
                CMPB            #00H            ;Is step number = 0?
                BEQ             SEQUENCER       ;YES.
*
***** BRANCH FOR ACTION LIST OR CONDITIONAL LIST PROCESSING *
*
*     ACTPROC and CONPROC
*     Passed Out Register Variables: X = Address of Active State List
*
                CLRA                            ;)
                ASLB                            ;)
                ROLA                            ;)) NO, point to algorithm table list for
                LDX             <STTBAS         ;)       given state number.
                LDX             D,X             ;)

                BITB            #02H            ;Is state list an action list?
                BEQ             BACE1           ;NO.
                LBSR            ACTPROC         ;YES, perform action list processing.
                BRA             BCHR1
BACE1           LBSR            CONPROC         ;Perform conditional list processing.
*
***** CHARACTER OUTPUT *
*
BCHR1           TST             <FLGDIS         ;Is display flag enabled?
                BNE             BTIM1           ;NO.
```

```
                LDA             >ACIASR         ;YES.
                ANDA            #02H            ;Has character been sent?
                BEQ             BTIM1           ;NO.

                LDX             <DISAD          ;YES, get current display address.
                LDA             X+              ;Get next ASCII character.
                STA             >ACIATD         ;Transmit character to display.
                CMPA            #0DH            ;Was character a carriage return?
                BNE             BCHR2           ;NO.

                INC             <FLGDIS         ;YES, disable display flag.
                LDX             #SDIS           ;Point to start of display buffer.
BCHR2           STX             <DISAD          ;Save, display address.
*
***** ELAPSED TIME UPDATE *
*
BTIM1           TST             <FLGSEC         ;Seconds flag enabled?
                BEQ             BPNT1           ;NO.
                LDA             <TIMERS         ;YES.
                ADDA            #1              ;)
                DAA                             ;)) Increment seconds timer.
                STA             <TIMERS         ;)
                CMPA            #60H            ;Is seconds timer = 60?
                BLT             BTIM2           ;NO.

                CLR             <TIMERS         ;YES, clear seconds timer.
                LDA             <TIMERM         ;)
                ADDA            #1              ;)
                DAA                             ;)) Increment minutes timer.
                STA             <TIMERM         ;)
                                                ;)
                CMPA            #60H            ;Is minutes timer = 60?
                BLT             BTIM2           ;NO.

                CLR             <TIMERM         ;YES, clear minutes timer.
                LDD             <TIMERHH        ;)
                EXG             A,B             ;)
                ADDA            #1              ;)
                DAA                             ;)) Increment hours timer.
                EXG             A,B             ;)
                ADCA            #0              ;)
                DAA                             ;)
                STD             <TIMERHH        ;)

BTIM2           DEC             <FLGSEC          ;Clear seconds flag.
```

```
*
***** STATUS DISPLAY LINE UPDATE *
*
                TST             <FLGPRT         ;Is print flag enabled?
                LBNE            BPNT1           ;NO.
                TST             <FLGDIS         ;YES, is display flag enabled?
                LBEQ            BPNT1           ;YES.

                CLR             <FLGDIS         ;NO, enable display flag.
                LDX             <DISAD          ;Point to start of display buffer.

                LDA             <FORNU          ;Get format number for display.
                CMPA            #1              ;)
                BEQ             BFOR1           ;)
                CMPA            #2              ;)
                BEQ             BFOR2           ;)) Branch to desired format routine.
                CMPA            #3              ;)
                BEQ             BFOR3           ;)

                INC             <FLGDIS         ;Disable display.
                LBRA            BPNT1


88  BFOR1       LBSR            TIME            ;Display elapsed time.

                LDA             #" "            ;)
                STA             X+              ;)) Display two spaces.
                STA             X+              ;)

                LDY             #SACT           ;)
        BF101   LDB             Y+              ;)
                LDA             #" "            ;)
                STA             X+              ;)
                LBSR            HEXBCD          ;)
                LBSR            CRYDIS          ;)) Display 16 active states.
                LBSR            BCDDIS          ;)
                CMPY            #EACT           ;)
                BLE             BF101           ;)
                                                ;)
                LDA             #0DH            ;)
                STA             X               ;)

                LBRA            BPNT1           ;Done.
```

```
BFOR2           LBSR            TIME            ;Display elapsed time.

                LDA             #" "            ;)
                STA             X+              ;)
                STA             X+              ;)) Display three spaces.
                STA             X+              ;)
                                                ;)

                LDY             #DATIN          ;)
BF201           LDA             Y+              ;)
                LBSR            BITDIS          ;)
                CMPY            #DATIN+8        ;)) Display status of 64 I/O points.
                BNE             BF201           ;)
                                                ;)
                LDA             #0DH            ;)
                STA             X               ;)

                LBRA            BPNT1           ;Done.


BFOR3           LBSR            TIME            ;Display elapsed time.

                LDA             #" "            ;)
                STA             X+              ;)
                STA             X+              ;)) Display three spaces
                STA             X+              ;)
                                                ;)

                LDY             #DATIN          ;)
BF301           LDA             Y+              ;)
                LBSR            BITDIS          ;)) Display status of 32 I/O points.
                CMPY            #DATIN+4        ;)
                BNE             BF301           ;)

                LDY             #SACT           ;)
BF302           LDB             Y+              ;)
                LDA             #" "            ;)
                STA             X+              ;)
                LBSR            HEXBCD          ;)
                LBSR            CRYDIS          ;)) Display 8 active states.
                LBSR            BCDDIS          ;)
                CMPY            #SACT+7         ;)
                BLE             BF302           ;)
```

```
                LDA             #0DH            ;)
                STA             X               ;)

                LBRA            BPNT1           ;Done.


*************************************************
*          SUBROUTINES                          *
*************************************************
*
***** TIME *
*
*               TIME is used to convert the hours, minutes, and seconds into ASCII code
*               stored at consecutive memory locations pointed to by register X.
*
*               X=Incremented Pointer;   A=Altered
*
TIME            LDA             <TIMERHH        ;Get high byte hours counter.
                BSR             BCDDIS          ;Convert to ASCII characters for display.
                LDA             <TIMERHL        ;Get low byte hours counter.
                BSR             BCDDIS          ;Convert to ASCII characters for display.
                LDA             #":"            ;Generate ASCII character.
                STA             X+              ;Insert in status line.
                LDA             <TIMERM         ;Get minutes counter.
                BSR             BCDDIS          ;Convert to ASCII characters for display.
                LDA             #":"            ;Generate ASCII character.
                STA             X+              ;Insert in status line.
                LDA             <TIMERS         ;Get seconds counter.
                BSR             BCDDIS          ;Convert to ASCII characters for display.
                RTS
*
***** HEXBCD *
*
*               HEXBCD converts the state between 1 and 127 to BCD.
*               B=HEX Entered;    A=BCD Returned
*
HEXBCD          CLRA                            ;Clear counter.
                LSRB
                LSRB                            ;Is bit 1 = 1?
                BCC             BHEX1           ;NO
                INCA                            ;YES, add 1 to counter.
BHEX1           LSRB                            ;Is bit 2 = 1?
                BCC             BHEX2           ;NO.
                ADDA            #02H            ;YES, add 2 to counter.
```

```
BHEX2           LSRB                            ;Is bit 3 = 1?
                BCC             BHEX3           ;NO.
                ADDA            #04H            ;YES, add 4 to counter.
BHEX3           LSRB                            ;Is bit 4 = 1?
                BCC             BHEX4           ;NO.
                ADDA            #08H            ;YES, add 8 to counter.
                DAA
BHEX4           LSRB                            ;Is bit 5 = 1?
                BCC             BHEX5           ;NO.
                ADDA            #16H            ;YES, add 16 to counter.
                DAA
BHEX5           LSRB                            ;Is bit 6 = 1?
                BCC             BHEX6           ;NO.
                ADDA            #32H            ;YES, add 32 to counter.
                DAA
BHEX6           LSRB                            ;Is bit 7 = 1?
                BCC             BHEX7           ;NO.
                ADDA            #64H            ;YES, add 64 to counter.
                DAA
BHEX7           RTS
*
***** BITDIS *
*
*               BITDIS stores the binary bits of a byte in the display buffer.
*
*               X=Pointer;  B=Byte
*
BITDIS          PSHU            B
                LDB             #8              ;Set counter.
BBIT1           LSRA                            ;Get bit value.
                BSR             CRYDIS          ;Send ASCII character for bit to display.
                DECB                            ;Have all bits been processed?
                BNE             BBIT1           ;NO.
                PULU            B               ;YES.
                RTS
*
***** CRYDIS *
*
*               CRYDIS converts the value of the carry to ASCII code, and stores it in X+.
*
*                       X=POINTER
*
CRYDIS          PSHU            B               ;Is carry flag set?
                BCC             BCRY1           ;NO.
                LDB             #"1"            ;YES, Generate ASCII 1.
```

```
                STB             X+              ;Insert in status line.
                BRA             BCRY2
BCRY1           LDB             #"0"
                STB             X+              ;Generate ASCII 0.
BCRY2           PULU            B               ;Insert in status line.
                RTS
*
***** BCDDIS *
*
*               BCDDIS is used to convert a two digit BCD number stored in register A
*               into a two digit ASCII code stored an two consecutive memory locations
*               pointed at by X.  Register X is incremented by two after this subroutine.
*                    A=BCD NUMBER
BCDDIS          PSHU            A               ;Save BCD number.
                LSRA                            ;)
                LSRA                            ;)) Mask out tens digit.
                LSRA                            ;)
                LSRA                            ;)
                ORA             #30H            ;Generate ASCII character.
                STA             X+              ;Insert in status line.
                PULU            A               ;Get BCD number.
                ANDA            #0FH            ;Mask out ones digit.
                ORA             #30H            ;Generate ASCII character.
                STA             X+              ;Insert in status line.
                RTS
```

Appendix C Program Listing Action List Processing

```
*************************************************
*      GRADUATE THESIS                          *
*      State Machine                            *
*      Action Table Processor                   *
*                                               *
*      AUTHOR: Robert M. Laurie                 *
*      DATE: February 9, 1986                   *
*                                               *
*************************************************

*
***** EXTERNAL / GLOBAL LABELS *
*
                GLB             ACTPROC

                EXT             FLGPRT, FLGSEC, FLGPAS, FLGDIS
                EXT             FORNU, MESAD, DISAD, PIAAD, PIANU
                EXT             STIME,TIMERmS,TIMERS,TIMERM,TIMERHL,TIMERHH
                EXT             ACTPNT, ACTBAS, STTBAS, SACT, EACT
                EXT             DATIN, DATOUT, SDATOUT, DATDDR
                EXT             SDIS, EDIS, SREG, EREG
                EXT             PDATA, PDATA1, MONITR, BRKPT
                EXT             ACIACR, ACIASR, ACIATD
*
***** ACTION FUNCTIONS *
*
LOADR           EQU             0A0H            ;Load register with immediate data.
MOVER           EQU             0A1H            ;Move data from first register to second.
INCRG           EQU             0A2H            ;Increment register.
DECRG           EQU             0A3H            ;Decrement register.
ADDIM           EQU             0A4H            ;Add immediate data to register.
SUBIM           EQU             0A5H            ;Subtract immediate data from register.
ADDRG           EQU             0A6H            ;Add two registers placing result in first.
SUBRG           EQU             0A7H            ;Subtract registers placing result in first.

TIMES           EQU             0A8H            ;Set timer alarm.

EXPAC           EQU             0A9H            ;Expand active states.
HLTSQ           EQU             0AAH            ;Sequencing is halted, outputs to standby mode.
EXTFN           EQU             0ABH            ;External function.
ACNOP           EQU             0ACH            ;No operation.
ACEND           EQU             0ADH            ;End of active state block.
```

```
                PROG

*************************************************
*     ACTION TABLE PROCESSOR -- ACTPROC         *
*************************************************
*
*    Passed In Register Variables: X = Address of Active State List
*
ACTPROC         LDA             X+              ;Get action list entry.
                BITA            #80H            ;Is it a function?
                BNE             BFNA1           ;YES.
*
***** DISCRETE OUTPUT ALTERED *
*
                TFR             A,B             ;NO.
                ANDB            #38H            ;Mask out PIA #.
                LSRB
                LSRB
                LSRB
                PSHU            B               ;Save PIA #.

                TFR             A,B
                ANDB            #07H            ;Mask out bit #.
                ROLA
                ROLA                            ;Check for on?
                LDA             #0FFH
                BCC             BOUT1           ;NO.
                COMA
BOUT1           ROLA                            ;YES.
                TSTB
                BEQ             BOUT2
                DECB
                BRA             BOUT1

BOUT2           PULU            B
                LDY             #DATOUT
                BCS             BOUT3

                ORA             B,Y             ;Turn output bit off.
                STA             B,Y
                BRA             ACTPROC

BOUT3           ANDA            B,Y             ;Turn output bit on.
                STA             B,Y
                BRA             ACTPROC
```

```
*
***** FUNCTION DECODER *
*
                CMPA            #LOADR
                LBEQ            BLOADR
BFNA1           CMPA            #MOVER
                LBEQ            BMOVER

                CMPA            #INCRG
                LBEQ            BINCRG
                CMPA            #DECRG
                LBEQ            BDECRG
                CMPA            #ADDIM
                LBEQ            BADDIM
                CMPA            #SUBIM
                LBEQ            BSUBIM
                CMPA            #ADDRG
                LBEQ            BADDRG
                CMPA            #SUBRG
                LBEQ            BSUBRG

                CMPA            #TIMES
                LBEQ            BTIMES
                CMPA            #EXPAC
                LBEQ            BEXPAC
                CMPA            #HLTSQ
                LBEQ            BHLTSQ
                CMPA            #EXTFN
                LBEQ            BEXTFN
                CMPA            #ACNOP
                LBEQ            BACNOP
                CMPA            #ACEND
                LBEQ            BACEND
*
***** LOADR *
*
*               Load register with immediate data.
*
*               Operands (2):   [Reg#] [Immediate data]
*
BLOADR          LDA             X+              ;Get register #.
                LDB             X+              ;Get immediate data.
                LDY             #SREG           ;Point to register.
                STB             A,Y             ;Store value.
                LBRA            ACTPROC
```

```
*
***** MOVER *
*
*               Moves data from first register to second register.
*
*               Operands (2):  [Reg#] [Reg#]
*
BMOVER          LDA             X+              ;Get first register #.
                LDB             X+              ;Get second register #.
                LDY             #SREG
                LDA             A,Y             ;Transfer data from first to second.
                STA             B,Y
                LBRA            ACTPROC
*
***** INCRG *
*
*               Increment register.
*
*               Operands (1):  [Reg#]
*
BINCRG          LDA             X+              ;Get register #.
                LDY             #SREG
                INC             A,Y             ;Increment value in register.
                LBRA            ACTPROC
*
***** DECRG *
*
*               Decrement register.
*
*               Operands (1):  [Reg#]
*
BDECRG          LDA             X+              ;Get register #.
                LDY             #SREG
                DEC             A,Y             ;Decrement value in register.
                LBRA            ACTPROC
*
***** ADDIM *
*
*               Add immediate data to a register.
*
*               Operands (2):  [Reg#] [Immediate data]
*
BADDIM          LDA             X+              ;Get register #.
                LDY             #SREG
                LDB             A,Y
```

```
                ADDB            X+              ;Add data to register value.
                STB             A,Y
                LBRA            ACTPROC
        *
        ***** SUBIM *
        *
        *               Subtract immediate data from a register.
        *
        *               Operands (2):  [Reg#] [Immediate data]
        *
        BSUBIM          LDA             X+              ;Get register #.
                        LDY             #SREG
                        LDB             A,Y
                        SUBB            X+              ;Subtract data from register value.
                        STB             A,Y
                        LBRA            ACTPROC
        *
        ***** ADDRG *
        *
        *               Add two register values together storing sum in first.
        *
        *               Operands (2):  [Reg#] [Reg#]
        *
        BADDRG          LDA             X+              ;Get first register #.
                        LDB             X+              ;Get second register #.
                        LDY             #SREG
                        LDB             B,Y
                        ADDB            A,Y             ;Add register values together.
                        STB             A,Y             ;Store sum in first register.
                        LBRA            ACTPROC
        *
        ***** SUBRG *
        *
        *               Subtract first register value from second placing difference in the first.
        *
        *               Operands (2):  [Reg#] [Reg#]
        *
        BSUBRG          LDA             X+              ;Get first register #.
                        LDB             X+              ;Get second register #.
                        LDY             #SREG
                        LDB             B,Y
                        NEGB
                        ADDB            A,Y             ;Subtract first value from second.
                        STB             A,Y             ;Store difference in the second.
                        LBRA            ACTPROC
```

```
*
***** TIMES *
*
*               Set timer alarm.  Note the data form of operands.
*               Maximum operand values contained in ( ).
*
*               Operands (7):   [Reg#]  [Hours:thousands,hundreds;BCD] (99)
*                                       [Hours:tens,ones;BCD] (99)
*                                       [Minutes:tens,ones;BCD] (59)
*                                       [Seconds:tens,ones;BCD] (59)
*                                       [mSeconds:high byte:HEX] (03)
*                                       [mSeconds: low byte; HEX] (E7)
*
BTIMES          LDA             X+              ;Get register number.
                LDY             #SREG
                LEAY            A,Y             ;Point to register.

                LDD             <TIMERmS        ;Get milliseconds time.
                ADDD            4,X             ;Add to get alarm time.
                CMPD            #1000           ;Value to high?
                BLO             BTIM1           ;NO.
                SUBD            #1000           ;Reduce by 1 second.
                STD             4,Y             ;Store alarm.
                COMA                            ;Set carry.
                BRA             BTIM2

BTIM1           STD             4,Y             ;Store alarm
                CLRA                            ;Clear carry.

BTIM2           LDA             <TIMERS         ;Get seconds time.
                ADCA            3,X             ;Add to get alarm time.
                DAA                             ;Convert to BCD.
                CMPA            #60H            ;Value to high?
                BLO             BTIM3           ;NO.
                SUBA            #60H            ;YES, Reduce by 1 minute.
                STA             3,Y             ;Store alarm.
                COMA                            ;Set carry.
                BRA             BTIM4

BTIM3           STA             3,Y             ;Store alarm.
                CLRA                            ;Clear carry.

BTIM4           LDA             <TIMERM         ;Get minutes time.
                ADCA            2,X             ;Add to get alarm.
                DAA                             ;Convert to BCD.
```

```
                CMPA            #60H            ;Value to high?
                BLO             BTIM5           ;NO.
                SUBA            #60H            ;YES, Reduce by 1 hour.
                STA             2,Y             ;Store alarm.
                COMA                            ;Set carry.
                BRA             BTIM6

BTIM5           STA             2,Y             ;Store alarm.
                CLRA                            ;Clear carry.

BTIM6           LDA             <TIMERHL        ;Get hours time (low byte).
                ADCA            1,X             ;Add to get alarm.
                DAA                             ;Convert to BCD.
                STA             1,Y             ;Store alarm.
                LDA             <TIMERHH        ;Get hours time (high byte).
                ADCA            X               ;Add to get alarm.
                DAA                             ;Convert to BCD.
                STA             Y               ;Store alarm.

                LEAX            6,X             ;Increment list pointer.
                LBRA            ACTPROC
*
***** EXPAC *
*
*               Expands the number of active states to a specified number.
*               The state numbers are given as operands calling up the action
*               list for each state number.
*
*               Operands (Variable):  [Total # of states expanded]
*                                       [State#] ...
*
BEXPAC          LDY             <ACTBAS
                LDA             <ACTPNT
                LEAY            A,Y             ;Point to currently active state.

                CLRA
BEXP1           INCA
                TST             A,Y             ;Determine end of active states.
                BNE             BEXP1
                LDB             X+
                PSHU            B,X,Y
                LEAY            A,Y             ;Load pointers.
                LEAX            B,Y
```

```
BEXP2           DECA
                BEQ             BEXP3
                LDB             -Y              ;Transfer data in active state registers.
                STB             -X
                BRA             BEXP2

BEXP3           PULU            B,X,Y
                LEAY            1,Y

BEXP4           LDA             X+
                ASLA                            ;Load active state registers with new action states.
                ORA             #01H
                STA             Y+
                DECB
                BNE             BEXP4

                LBRA            ACTPROC
*
***** HLTSQ *
*
*               Sequencing is halted, outputs are returned to the standby
*               mode, a message is sent to the terminal, and control is
*               returned to the system monitor.
*
BHLTSQ          LDA             X+              ;Get message #.
                TST             <FLGPRT         ;Print flag enabled?
                BNE             BHLT2           ;NO.
                LDX             <MESAD          ;YES.
                ASLA
                LDX             A,X             ;Get message starting address.
BHLT1           LDA             >ACIASR
                ANDA            #02H            ;Has last character been sent?
                BEQ             BHLT1           ;NO.

                CMPB            #04H            ;YES, has end of message been reached?
                BEQ             BHLT2           ;YES.

                LDB             X+              ;NO.
                STB             >ACIATD         ;Transmit Character.
                BRA             BHLT1

BHLT2           LDA             #03H
                STA             >ACIACR         ;Reset ACIA.
                LDA             #49H
                STA             >ACIACR
```

```
                LDX             <PIAAD
                LDA             <PIANU          ;Load PIA outputs with standby data.
                LDY             #SDATOUT
BHLT3           LDB             Y+
                STB             X++
                DECA
                BNE             BHLT3

                SWI
                FCB             MONITR          ;Return to monitor.
*
***** ACNOP
*
BACNOP          LBRA            ACTPROC         ;No operation.
*
***** EXTFN *
*
*               Used to transfer control over to the state table for execution
*               on an external function.
*
BEXTFN          SWI2                            ;Execute external function.
                LBRA            ACTPROC
*
***** ACEND *
*
*               End of a state action list.
*
BACEND          LDY             <ACTBAS
                LDA             <ACTPNT
                LDB             A,Y
                ANDB            #0FEH           ;Alter active state # for condition mode.
                STB             A,Y

                RTS
```

Appendix D Program Listing Condition List Processing

```
"6809"

*************************************************
*      GRADUATE THESIS                          *
*      State Machine                            *
*      Conditional Table Processor              *
*                                               *
*      AUTHOR: Robert M. Laurie                 *
*      DATE: February 9, 1986                   *
*                                               *
*************************************************

*
***** EXTERNAL / GLOBAL LABELS *
*
                GLB             CONPROC

                EXT             FLGPRT, FLGSEC, FLGPAS, FLGDIS
                EXT             FORNU, MESAD, DISAD, PIAAD, PIANU
                EXT             STIME,TIMERmS,TIMERS,TIMERM,TIMERHL,TIMERHH
                EXT             ACTPNT, ACTBAS, STTBAS, SACT, EACT
                EXT             DATIN, DATOUT, SDATOUT, DATDDR
                EXT             SDIS, EDIS, SREG, EREG
                EXT             PDATA, PDATA1, MONITR, BRKPT
                EXT             ACIACR, ACIASR, ACIATD

***** CONDITIONAL FUNCTIONS *
*
CMPGE           EQU             80H             ;Compare registers greater than or equal.
CMPGT           EQU             81H             ;Compare registers greater than.
CMPEQ           EQU             82H             ;Compare registers equal.
CMPNE           EQU             83H             ;Compare registers not equal.

CIMGE           EQU             84H             ;Compare register with data greater than or equal.
CIMGT           EQU             85H             ;Compare register with data greater than.
CIMEQ           EQU             86H             ;Compare register with data equal.
CIMNE           EQU             87H             ;Compare register with data not equal.
CIMLE           EQU             88H             ;Compare register with data less than or equal.
CIMLT           EQU             89H             ;Compare register with data less than.

TIMET           EQU             8AH             ;Compares six registers with the elapsed timer.

BLKOR           EQU             8BH             ;Either the previous OR next conditional block.
TRANS           EQU             8CH             ;Transfer to new state if conditions satisfied.
CONAC           EQU             8DH             ;Contract active states.
```

```
CNNOP           EQU             8EH             ;No operation.
CNEND           EQU             8FH             ;End of a conditional state block.

                PROG

*************************************************
*     CONDITIONAL TABLE PROCESSOR -- CONPROC    *
*************************************************
*
*    Passed In Register Variables: X = Address of Active State List
*
CONPROC         CLR             <FLGPAS         ;Enable contitional pass flag.
BINT1           LDA             X+              ;Get state table entry and increment pointer.
*
***** DISCRETE INPUT CHECK *
*
                BITA            #80H            ;Bit 7?: 0=Input check, 1=Function.
                BNE             BFNC1           ;1, go to function decoder.
                TST             <FLGPAS         ;0, test pass flag enabled?
                BNE             BINT1           ;NO

                TFR             A,B             ;YES, B=table entry.
                ANDA            #38H            ;)
                LSRA                            ;)
                LSRA                            ;)) Determine byte number.
                LSRA                            ;)
                                                ;)
                PSHU            A,B             ;Push byte number and entry.
                TFR             B,A             ;A=Entry.
                ANDA            #07H            ;Mask out bit number.
                LDB             #0FFH
                COMB                            ;C=1,B=00H.
BINP1           ROLB                            ;)
                TSTA                            ;)
                BEQ             BINP2           ;)) Generate mask.
                DECA                            ;)
                BRA             BINP1           ;)

BINP2           PULU            A               ;Pull byte number.
                LDY             #DATIN          ;Get input data pointer.
                ANDB            A,Y             ;What is input bit value?
                BNE             BINP3           ;1.

                PULU            B               ;0, pull entry.
                BITB            #40H            ;What should bit be?
```

```
                BEQ             BINT1           ;0.
                INC             <FLGPAS         ;1, condition not satisfied.
                BRA             BINT1

BINP3           PULU            B               ;Pull entry.
                BITB            #40H            ;What should bit be?
                BNE             BINT1           ;1.
                INC             <FLGPAS         ;0, condition not satisfied.
                BRA             BINT1
*
***** FUNCTION DECODER *
*
BFNC1           CMPA            #CMPGE
                BEQ             BCMPGE
                CMPA            #CMPGT
                BEQ             BCMPGT
                CMPA            #CMPEQ
                LBEQ            BCMPEQ
                CMPA            #CMPNE
                LBEQ            BCMPNE

                CMPA            #CIMGE
                LBEQ            BCIMGE
                CMPA            #CIMGT
                LBEQ            BCIMGT
                CMPA            #CIMEQ
                LBEQ            BCIMEQ
                CMPA            #CIMNE
                LBEQ            BCIMNE
                CMPA            #CIMLE
                LBEQ            BCIMLE
                CMPA            #CIMLT
                LBEQ            BCIMLT

                CMPA            #TIMET
                LBEQ            BTIMET

                CMPA            #BLKOR
                LBEQ            BBLKOR
                CMPA            #TRANS
                LBEQ            BTRANS
                CMPA            #CONAC
                LBEQ            BCONAC
                CMPA            #CNNOP
                LBEQ            BCNNOP
```

```
                CMPA            #CNEND
                LBRA            BCNEND
*
***** CMPGE *
*
*               Compares a pair of registers and checks that the first
*               is greater than or equal to the second.
*
*               Operands (2):  [Reg#] [Reg#]
*
BCMPGE          BSR             CMPRG           ;Compare registers, does condition apply?
                LBGE            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
*
***** CMPGT *
*
*               Compares a pair of registers and checks that the first
*               is greater than the second.
*
*               Operands (2):  [Reg#] [Reg#]
*
BCMPGT          BSR             CMPRG           ;Compare registers, does condition apply?
                LBGT            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
*
***** CMPEQ *
*
*               Compares a pair of registers and checks that the first
*               is equal to the second.
*
*               Operands (2):  [Reg#] [Reg#]
*
BCMPEQ          BSR             CMPRG           ;Compare registers, does condition apply?
                LBEQ            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
*
***** CMPNE *
*
*               Compares a pair of registers and checks that the first
*               is not equal to the second.
*
*               Operands (2):  [Reg#] [Reg#]
```

```
*
BCMPNE          BSR             CMPRG           ;Compare registers, does condition apply?
                LBNE            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1

************************************************
*               SUBROUTINES                    *
************************************************
*
*               CMPRG is called by all register comparison programs.
*               Placed here to save bytes for short branches.
*
CMPRG           LDD             X++             ;Get first register number.
                TST             <FLGPAS         ;Test pass flag enabled?
                LBNE            BINT1           ;NO.
                LDY             #SREG           ;YES, point to start of user registers.
                LDA             A,Y             ;Get first register value.
                CMPA            B,Y             ;First value greater than or equal to second?
                RTS
*
*               CMPIM is called by all register to immediate data comparison routines.
*               Place here to save bytes for short branches.
*
CMPIM           LDD             X++             ;Get register number.
                TST             <FLGPAS         ;Test pass flag enabled?
                LBNE            BINT1           ;NO.
                LDY             #SREG           ;YES, point to start of user registers.
                LDA             A,Y
                CMPA            -1,X            ;Is register value greater than immedate data?
                RTS
*
*
***** CIMGE *
*
*               Compares a register with immediate data and checks that the register
*               is greater than or equal to the immediate data value.
*
*               Operands (2):  [Reg#] [Immediate data]
*
BCIMGE          BSR             CMPIM           ;Compare register with data, does condition apply?
                LBGE            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
```

```
*
***** CIMGT *
*
*               Compares a register with immediate data and checks that the register
*               is greater than to the immediate data value.
*
*               Operands (2):  [Reg#] [Immediate data]
*
BCIMGT          BSR             CMPIM           ;Compare register with data, does condition apply?
                LBGT            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
*
***** CIMEQ *
*
*               Compares a register with immediate data and checks that the register
*               is equal to the immediate data value.
*
*               Operands (2):  [Reg#] [Immediate data]
*
BCIMEQ          BSR             CMPIM           ;Compare register with data, does condition apply?
                LBEQ            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
*
***** CIMNE *
*
*               Compares a register with immediate data and checks that the register
*               is not equal to the immediate data value.
*
*               Operands (2):  [Reg#] [Immediate data]
*
BCIMNE          BSR             CMPIM           ;Compare register with data, does condition apply?
                LBNE            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
*
***** CIMLE *
*
*               Compares a register with immediate data and checks that the register
*               is less than or equal to the immediate data value.
*
*               Operands (2):  [Reg#] [Immediate data]
*
BCIMLE          BSR             CMPIM           ;Compare register with data, does condition apply?
```

```
                LBLE            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
*
***** CIMLT *
*
*               Compares a register with immediate data and checks that the register
*               is less than the immediate data value.
*
*               Operands (2):   [Reg#] [Immediate data]
*
BCIMLT          BSR             CMPIM           ;Compare register with data, does condition apply?
                LBLT            BINT1           ;YES.
                INC             <FLGPAS         ;NO, condition not satisfied.
                LBRA            BINT1
*
***** TIMET *
*
*               This instruction checks that the values of six
*               consecutive user registers are less than or equal
*               to the six elapsed time registers.  Only the first
*               register number is required as an operand.
*
*               Operands (1):   [Reg#]
*
BTIMET          LDA             X+              ;Get register number.
                TST             <FLGPAS         ;Test pass flag enabled?
                LBNE            BINT1           ;NO.
                LDY             #SREG           ;YES.
                LEAY            A,Y             ;Point to first register.
                LDD             4,Y             ;Get milliseconds value.
                CMPD            <TIMERmS        ;Is it greater than time?
                BGT             BTIM1           ;YES.
                LDD             2,Y             ;NO, get minutes/seconds value.
                CMPD            <TIMERM         ;Is it greater than time?
                BGT             BTIM2           ;YES.
                LDD             Y               ;NO, get hours value.
                CMPD            <TIMERHH        ;Is it greater than time?
                BGT             BTIM2           ;YES.
                LBRA            BINT1           ;NO.
BTIM1           LDD             2,Y             ;Get minutes/seconds value.
                CMPD            <TIMERM         ;Is it less than time?
                LBLT            BINT1           ;YES.
BTIM2           INC             <FLGPAS         ;NO, disable pass flag.
                LBRA            BINT1
```

```
*
***** BLKOR *
*
*
*               This instruction is use to mark the end of a conditional
*               which is to be OR'ed with the next conditional block.
*
BBLKOR          TST             <FLGPAS         ;Test pass flag enabled?
                LBNE            CONPROC         ;NO.

BBOR1           LDA             X+              ;YES, get next byte.
                CMPA            #TRANS          ;Is it equal to TRANS?
                BNE             BBOR1           ;NO.

                BRA             BTRANS          ;YES
*
***** TRANS *
*
*               This instruction marks the end of a set of conditional statements
*               and transfers control to a new state should the previous conditions
*               be satified.
*
*               Operands (1):  [New state #]
*
BTRANS          LDA             X+              ;Get new state #.
                TST             <FLGPAS         ;Test pass flag enabled?
                LBNE            CONPROC         ;NO.
                ASLA                            ;YES.
                ORA             #01H            ;Set for action list of new state.
                LDB             <ACTPNT         ;)
                LDY             <ACTBAS         ;)) Change active state to new state.
                STA             B,Y             ;)
                RTS
*
***** CONAC *
*
*               If the following states are conditionally active, the total number of
*               states is reduced by a specified number, and transition occurs to a new
*               state.  This instruction must be the last statement of a conditional list.
*
*               Operands (Variable):  [Total active states to be contracted]
*                                     [State # conditional] ...
*                                     [New state #]
*
BCONAC          TST             <FLGPAS         ;Test pass flag enabled?
```

```
                BNE             BCNEND          ;NO.

                LDA             X+              ;YES, get total states to be contracted.
                LDY             <ACTBAS
                LDB             <ACTPNT         ; Point to currently active state.
                LEAY            B,Y
                PSHU            A,Y
                LEAY            1,Y

BCON1           LDB             X+
                ASLB
                CMPB            Y+              ;Are conditional states active?
                BEQ             BCON2           ;YES.
                INC             <FLGPAS         ;NO, fails conditional test.
BCON2           DECA
                BNE             BCON1
                PULU            A,Y

                TST             <FLGPAS         ;Is pass flag enabled?
                BNE             BCNEND          ;NO.

                LDB             X               ;YES.
                ASLB
                ORB             #01H            ;Get new state #.
                STB             Y+
                LEAX            A,Y
BCON3           LDB             X+              ;Contract the required amount of states.
                STB             Y+
                CMPX            #EACT
                BLT             BCON3

                RTS
*
***** CNNOP *
*
BCNNOP          LBRA            BINT1           ;No operation.
*
***** CNEND *
*
*               This instruction marks the end of a conditional state.
*
BCNEND          RTS
```

Appendix E State Table Assembler Listing

```
"6809"

*************************************************
*      GRADUATE THESIS                          *
*      State Table                              *
*                                               *
*      AUTHOR: Robert M. Laurie                 *
*      DATE: February 9, 1986                   *
*                                               *
*************************************************
*
***** SYMBOL DEFINITION *
*
*               @ = The Prescribed Order Must Be Followed
*               ~ = Upper Block Marker - Data may be of variable length in this block.
*               ^ = Lower Block Marker
*
***** I/O DECLARATION BLOCK (USER DEFINEABLE) *
*
READYLT_OFF     EQU             0
READYLT_ON      EQU             64
EXTLDER_OFF     EQU             1
EXTLDER_ON      EQU             65
RTRLDER_OFF     EQU             2
RTRLDER_ON      EQU             66
LODDONE_OFF     EQU             3
LODDONE_ON      EQU             67
EXTGRIP_OFF     EQU             4
EXTGRIP_ON      EQU             68
LWRDRIL_OFF     EQU             5
LWRDRIL_ON      EQU             69
RASDRIL_OFF     EQU             6
RASDRIL_ON      EQU             70
RTRGRIP_OFF     EQU             7
RTRGRIP_ON      EQU             71
DRLDONE_OFF     EQU             8
DRLDONE_ON      EQU             72

START_OFF       EQU             9
START_ON        EQU             73
PLOAD_OFF       EQU             10
PLOAD_ON        EQU             74
LDRET_OFF       EQU             11
LDRET_ON        EQU             75
PGRIP_OFF       EQU             12
```

```
PGRIP_ON        EQU             76
DRLDN_OFF       EQU             13
DRLDN_ON        EQU             77
DRLUP_OFF       EQU             14
DRLUP_ON        EQU             78
GRPRL_OFF       EQU             15
GRPRL_ON        EQU             79

LWRTEST_OFF     EQU             16
LWRTEST_ON      EQU             80
RASTS11_OFF     EQU             17
RASTS11_ON      EQU             81
EXTREMV_OFF     EQU             18
EXTREMV_ON      EQU             82
RTRREMV_OFF     EQU             19
RTRREMV_ON      EQU             83
RMVDONE_OFF     EQU             20
RMVDONE_ON      EQU             84
RASTS15_OFF     EQU             21
RASTS15_ON      EQU             85
RGRMVMN_OFF     EQU             22
RGRMVMN_ON      EQU             86
ROTPLAT_OFF     EQU             23
ROTPLAT_ON      EQU             87
STOPS_OFF       EQU             24
STOPS_ON        EQU             88
TSTDN_OFF       EQU             25
TSTDN_ON        EQU             89
TUP11_OFF       EQU             26
TUP11_ON        EQU             90
PCRMV_OFF       EQU             27
PCRMV_ON        EQU             91
RMVRT_OFF       EQU             28
RMVRT_ON        EQU             92
TUP15_OFF       EQU             29
TUP15_ON        EQU             93
RMVMN_OFF       EQU             30
RMVMN_ON        EQU             94
ROTAT_OFF       EQU             31
ROTAT_ON        EQU             95
*
***** ACTION LIST FUNCTIONS DECLARATION BLOCK *
*
LOADR           EQU             0A0H            ;Load register with immediate data.
MOVER           EQU             0A1H            ;Move data from first register to second.
```

```
INCRG           EQU             0A2H            ;Increment register.
DECRG           EQU             0A3H            ;Decrement register.
ADDIM           EQU             0A4H            ;Add immediate data to register.
SUBIM           EQU             0A5H            ;Subtract immediate data from register.
ADDRG           EQU             0A6H            ;Add two registers and place result in first.
SUBRG           EQU             0A7H            ;Subtract two registers and place result in first.

TIMES           EQU             0A8H            ;Set timer alarm.

EXPAC           EQU             0A9H            ;Expand active states.
HLTSQ           EQU             0AAH            ;Sequencing is halted, outputs put in standby mode.
EXTFN           EQU             0ABH            ;External function.
ACNOP           EQU             0ACH            ;No operation.
ACEND           EQU             0ADH            ;End of active state block.
*
*****  CONDITION LIST FUNCTIONS DECLARATION BLOCK *
*
CMPGE           EQU             80H             ;Compare register pair for greater than or equal.
CMPGT           EQU             81H             ;Compare register pair for greater than.
CMPEQ           EQU             82H             ;Compare register pair for equal.
CMPNE           EQU             83H             ;Compare register pair for not equal.

CIMGE           EQU             84H             ;Compare register with data greater than or equal.
CIMGT           EQU             85H             ;Compare register with data greater than.
CIMEQ           EQU             86H             ;Compare register with data equal.
CIMNE           EQU             87H             ;Compare register with data not equal.
CIMLE           EQU             88H             ;Compare register with data less than or equal.
CIMLT           EQU             89H             ;Compare register with data less than.

TIMET           EQU             8AH             ;Compares six registers with the elapsed timer.

BLKOR           EQU             8BH             ;Either the previous OR next conditional block.
TRANS           EQU             8CH             ;Transfer to new state if conditions satisfied.
CONAC           EQU             8DH             ;Contract active states.
CNNOP           EQU             8EH             ;No operation.
CNEND           EQU             8FH             ;End of a conditional state block.

                ORG             0D200H          ;@ Starting address given in linker.

************************************************
STBEG           HEX             0               ;@ 0=Terminal Print Out/1-8=Algorithm Table Number.
*
***** ALGORITHM TABLE STARTING ADDRESSES *
*
                FDB             AT1             ;@ Algorithm Table 1 Starting Address
```

```
                FDB             0               ;@ Algorithm Table 2 Starting Address
                FDB             0               ;@ Algorithm Table 3 Starting Address
                FDB             0               ;@ Algorithm Table 4 Starting Address
                FDB             0               ;@ Algorithm Table 5 Starting Address
                FDB             0               ;@ Algorithm Table 6 Starting Address
                FDB             0               ;@ Algorithm Table 7 Starting Address
                FDB             0               ;@ Algorithm Table 8 Starting Address
*
***** SELECTION MESSAGE *
*
*               ~
                HEX             1A
                ASC     "THIS STATE MACHINE WILL IMPLEMENT ONE OF THE EIGHT ALGORITHM TABLES"
                HEX             0A,0D
                ASC     "LISTED BELOW. PLEASE SELECT ONE OF THE FOLLOWING (1 - 8):"
                HEX             0A,0A,0D
                ASC     "     1 = Thesis Example of Three Station Automatic Drilling Station"
                HEX             0A,0D
                ASC     "     2 = None"
                HEX             0A,0D
                ASC     "     3 = None"
                HEX             0A,0D
                ASC     "     4 = None"
                HEX             0A,0D
                ASC     "     5 = None"
                HEX             0A,0D
                ASC     "     6 = None"
                HEX             0A,0D
                ASC     "     7 = None"
                HEX             0A,0D
                ASC     "     8 = None"
                HEX             0A,0D
                HEX             04
*               ^
*
***** ALGORITHM TABLE # 1 *
*
AT1             HEX             10,00           ;@ PIA Starting Address
                HEX             4               ;@ Number of 8-Bit PIA Units Required
*               ~
                HEX             FF,01           ;@ Unit 1 DDR/OR
                HEX             01,00           ;@ Unit 2 DDR/OR
                HEX             FF,00           ;@ Unit 3 DDR/OR
                HEX             00,00           ;@ Unit 4 DDR/OR
*               ^
```

```
                FCB             3               ;@ Display format number
                FCB             0               ;@ Heading number
                FDB             MESG            ;@ Address of message number vector table
                FDB             EXTFUNC         ;@ External Functions Address
*
***** STATE NUMBER VECTOR TABLE *
*
*               ~
                FDB             S1C1            ;State 1 Condition List Address
                FDB             S1A1            ;State 1 Action List Address
                FDB             S2C1            ;State 2 Condition List
                FDB             S2A1            ;State 2 Action List
                FDB             S3C1            ;State 3 Condition List
                FDB             S3A1            ;State 3 Action List
                FDB             S4C1            ;State 4 Condition List
                FDB             S4A1            ;State 4 Action List
                FDB             S5C1            ;State 5 Condition List
                FDB             S5A1            ;State 5 Action List
                FDB             S6C1            ;State 6 Condition List
                FDB             S6A1            ;State 6 Action List
                FDB             S7C1            ;State 7 Condition List
                FDB             S7A1            ;State 7 Action List
                FDB             S8C1            ;State 8 Condition List
                FDB             S8A1            ;State 8 Action List
                FDB             S9C1            ;State 9 Condition List
                FDB             S9A1            ;State 9 Action List.
                FDB             S10C1           ;State 10 Condition List
                FDB             S10A1           ;State 10 Action List
                FDB             S11C1           ;State 11 Condition List
                FDB             S11A1           ;State 11 Action List
                FDB             S12C1           ;State 12 Condition List
                FDB             S12A1           ;State 12 Action List
                FDB             S13C1           ;State 13 Condition List
                FDB             S13A1           ;State 13 Action List
                FDB             S14C1           ;State 14 Condition List
                FDB             S14A1           ;State 14 Action List
                FDB             S15C1           ;State 15 Condition List
                FDB             S15A1           ;State 15 Action List
                FDB             S16C1           ;State 16 Condition List
                FDB             S16A1           ;State 16 Action List
                FDB             S17C1           ;State 17 Condition List
                FDB             S17A1           ;State 17 Action List
*               ^
```

```
*
***** STATE ACTION AND CONDITIONAL LISTS *
*
*                ~
S1A1             FCB             ROTPLAT_OFF     ;State 1 Action List
                 FCB             READYLT_ON
                 FCB             ACEND

S1C1             FCB             START_ON        ;State 1 Condition List
                 FCB             TRANS,2
                 FCB             CNEND

S2A1             FCB             READYLT_OFF     ;State 2 Action List
                 FCB             EXTLDER_ON
                 FCB             EXPAC,2,5,10
                 FCB             ACEND

S2C1             FCB             PLOAD_ON        ;State 2 Conditional List
                 FCB             TRANS,3
                 FCB             CNEND

S3A1             FCB             EXTLDER_OFF     ;State 3 Action List
                 FCB             RTRLDER_ON
                 FCB             ACEND

S3C1             FCB             LDRET_ON        ;State 3 Conditional List
                 FCB             TRANS,4
                 FCB             CNEND

S4A1             FCB             RTRLDER_OFF     ;State 4 Action List
                 FCB             LODDONE_ON
                 FCB             ACEND

S4C1             FCB             LDRET_OFF       ;State 4 Conditional List
                 FCB             CONAC,2,9,14,17
                 FCB             CNEND


S5A1             FCB             EXTGRIP_ON      ;State 5 Action List
```

```
                FCB             ACEND

S5C1            FCB             PGRIP_ON        ;State 5 Conditional List
                FCB             TRANS,6
                FCB             CNEND

S6A1            FCB             LWRDRIL_ON      ;State 6 Action List
                FCB             EXTGRIP_OFF
                FCB             ACEND

S6C1            FCB             DRLDN_ON        ;State 6 Conditional List
                FCB             TRANS,7
                FCB             CNEND

S7A1            FCB             LWRDRIL_OFF     ;State 7 Action List
                FCB             RASDRIL_ON
                FCB             ACEND

S7C1            FCB             DRLUP_ON        ;State 7 Conditional List
                FCB             TRANS,8
                FCB             CNEND

S8A1            FCB             RASDRIL_OFF     ;State 8 Action List
                FCB             RTRGRIP_ON
                FCB             ACEND

S8C1            FCB             GRPRL_ON        ;State 8 Conditional List
                FCB             TRANS,9
                FCB             CNEND

S9A1            FCB             RTRGRIP_OFF     ;State 9 Action List
                FCB             DRLDONE_ON
                FCB             ACEND

S9C1            FCB             CNEND           ;State 9 Conditional List

S10A1           FCB             LWRTEST_ON      ;State 10 Action List
                FCB             TIMES,10,0,0,0,2,0,0
                FCB             ACEND
```

```
S10C1       FCB     TSTDN_ON        ;State 10 Condition List
            FCB     TRANS,11
            FCB     TIMET,10
            FCB     TRANS,15
            FCB     CNEND

S11A1       FCB     LWRTEST_OFF     ;State 11 Action List
            FCB     RASTS11_ON
            FCB     ACEND

S11C1       FCB     TUP11_ON        ;State 11 Condition List
            FCB     TRANS,12
            FCB     CNEND

S12A1       FCB     RASTS11_OFF     ;State 12 Action List
            FCB     EXTREMV_ON
            FCB     ACEND

S12C1       FCB     PCRMV_ON        ;State 12 Condition List
            FCB     TRANS,13
            FCB     CNEND

S13A1       FCB     EXTREMV_OFF     ;State 13 Action List
            FCB     RTRREMV_ON
            FCB     ACEND

S13C1       FCB     RMVRT_ON        ;State 13 Condition List
            FCB     TRANS,14
            FCB     CNEND

S14A1       FCB     RTRREMV_OFF     ;State 14 Action List
            FCB     RQRMVMN_OFF
            FCB     RMVDONE_ON
            FCB     ACEND

S14C1       FCB     CNEND           ;State 14 Condition List

S15A1       FCB     LWRTEST_OFF     ;State 15 Action List
            FCB     RASTS15_ON
            FCB     ACEND
```

```
S15C1           FCB             TUP15_ON        ;State 15 Condition List
                FCB             TRANS,16
                FCB             CNEND

S16A1           FCB             RASTS15_OFF     ;State 16 Action List
                FCB             RQRMVMN_ON
                FCB             ACEND

S16C1           FCB             RMVMN_ON        ;State 16 Condition List
                FCB             TRANS,14
                FCB             CNEND

S17A1           FCB             LODDONE_OFF     ;State 17 Action List
                FCB             DRLDONE_OFF
                FCB             RMVDONE_OFF
                FCB             ROTPLAT_ON
                FCB             ACEND

S17C1           FCB             ROTAT_ON        ;State 17 Condition List
                FCB             TRANS,1
                FCB             CNEND
*               ^
*
***** EXTERNAL FUNCTIONS *
*
*               SWI2 Programs are placed here.  These programs are called
*               by the EXTFNC instruction of Step Lists.  All programs must
*               terminate with an RTI instruction as they are interrupt driven.
*
*               ~
EXTFUNC         LDX             10,S
                LDA             X

***** EXTERNAL FUNCTIONS VECTOR TABLE *

                CMPA            #1
                BEQ             FUNC1

FUNC1           RTI

*               ^
```

```
*
***** MESSAGE VECTOR TABLE *
*
*                ~
MESG             FDB             MESG0
*                ^
*
***** MESSAGES *
*                    0000000000111111111122222222223333333333444444444455555555556666666666777777777
*                ~   1234567890123456789012345678901234567890123456789012345678901234567890123456789
MESG0            HEX             1A,0D,0A,0A
                 ASC"                  Algorithm Table 1: Thesis Example "
                 HEX             0A,0D
                 ASC"                  ---------------------------------"
                 HEX             0A,0A,0D
                 ASC"  ELAPSED |||          I/O Status             |   CURRENTLY  ACTIVE  STATES"
                 HEX             0A,0D
                 ASC"   TIME   |||0000000000111111111122222222233|"
                 HEX             0A,0D
                 ASC"HOUR:MN:SC|||01234567890123456789012345678901|S00 S01 S02 S03 S04 S05 S06 S07"
                 HEX             0A,0D
                 ASC"    |  |  |||:::::::::::::::::::::::::::::::|   |   |   |   |   |   |   |   |"
                 HEX             0A,0D,04
*                ^
```

LIST OF REFERENCES

1. Andrews, Michael, "Programming Microprocessor Interfaces for Control and Instrumentation", Prentice Hall Inc., Englewood Cliffs, N.J., 1982.

2. Bartlett, Peter G.,"Advantages of Controlling Transfer Lines Using Distributed Control", Proceedings of the 1980 Joint Automatic Control Conference, Vol 1, San Francisco, CA., Paper No WA7-D, August 13-15, 1980.

3. Bartlett, Peter G.,"Installing a PC",Instruments & Control Systems, Vol 53, No 8, pp.45-47, August 1980.

4. Brown, Lyman F., "A Role For Programmable Controllers In Factory Distributed Control", IEEE Industry Applications Society Meeting, pp. 31-35, October 1982.

5. Christensen, J. H., O. J. Struger, "Programmable Controller Software Architectures for Advanced Machine Diagnostics", IEEE Transactions on Industry Applications, Vol IA-21, No 1, pp.112-115, January/February 1985.

6. Courvoisier, M., R. Valette, J. M. Bigou, and P. Esteban, " A Programmable Logic Controller Based On A High Level Specification Tool", Laboratoire d'Automatique, Toulouse, France, 1983.

7. Delato, Dave, "Programming Your PC", Instruments & Control Systems, Vol 53, No 7, pp.37-40, July 1980.

8. Dornfeld, D. A., D. M. Auslander, and P. Sagues, "Microprocessor - Controlled Manufacturing Processes", Mechanical Engineering, Vol 102, No 13, pp.34-41, December 1981.

9. Dornfeld, D. A., D. M. Auslander, and P. Sagues, "Software for Microprocessor Control of Mechanical Equipment", Proceedings: 1979 Joint Automatic Control Conference, Denver, CO., pp.71-77, June 17-21, 1979.

10. Flynn, W. R., "New PC Introductions Are Located At Opposite Ends of Spectrum", Control Engineering, Vol 32, No 1, pp.77-78, January 1985.

11. Flynn, W. R., "1985 PC Update", Control Engineering, Vol 32, No 1, pp.79-83, January 1985.

12. Gander, J. G. and H. U. Liechti, "State Language for Real-Time Process Control", Microprocessors and Microsystems, Vol 5, No1, pp.27-28, January/February 1981.

13. Grzelak, T. A., P. H. Lewis, M. J. Gilpin, and R. O. Shelton, "Identification and Classification of State Transition Techniques for Industrial Machine Control", Michigan Tech Univ, Houghton, MI, March 1, 1983.

14. Grzelak, T. A., P. H. Lewis, M. J. Gilpin, and R. O. Shelton, "Comparison of State Transition Techniques and Relay Ladder Logic Techniques for Industrial Machine Control", Michigan Tech Univ, Houghton, MI, May 31, 1983.

15. Henry, Don, "PC I/O Considerations", Instruments & Control Systems, Vol 53, No 6, pp.37-40, June 1980.

16. Hopkins, Albert L., Jr., "Software Issues in Redundant Sequential Control", IEEE Transactions on Industrial Electronics, Vol IE-29, No 4, pp.273-278, November 1982.

17. Hopkins, Albert L., Jr. and Louis J. Quagliata, "State Transition Synthesis for Sequential Control of Discrete Machines", Presented at the Second International Conference, Applications of Electronics in Industry, Torino, Italy, October 2-3, 1973.

18. Jasany, L. C., "Tying the Factory Together With PC's and Networks", Production Engineering, Vol 31, No 4, pp.52-62, April 1984.

19. Johnsonbaugh, Richard and Tadao Murata, "Petri Nets and Marked Graphs - Mathematical Models of Concurrent Computation", The American Mathematical Monthly, Vol 89, No 8, pp.552-566, October 1982.

20. Landau, Jack V., "State Description Techniques Applied to Industrial Machine Control", Computer, Vol 12, No 2, pp.32-40, February 1979.

21. Laduzinsky, A. J., "Programmable Controller Delivers Fault Tolerance Using Twelve Microprocessors", Control Engineering, Vol 32, No 2, pp.76-77, February 1985.

22. Laduzinsky, A. J., "STD Bus Performance Increases With Multiple Processors, Multitasking Software, and VLSI", Control Engineering, Vol 31, No 13, pp.65-70, December 1984.

23. Lloyd, Mike, "Graphical Function Chart Programming for Programmable Controllers", Control Engineering, Vol 32, No 10, pp.73-76, October 1985.

24. McPhillips, A. Scott, "Designing Logic with Software", Proceedings IEEE 1978 MICRO-DELCON, The Delaware Bay Microcomputer Conference, pp.7-11, March 1978.

25. Mical, Dr. Richard D. and Stephen C. Schwarm, "Implementing State Diagrams in a High Level Language", Proceedings IEEE 1979 MICRO-DELCON, The Delaware Bay Microcomputer Conference, pp.89-97, March 1979.

26. Morris, Henry M., "Fast Growing PC Market Encourages Wide Range of Product Offerings", Control Engineering, Vol 30, No 1, PP.57-61, January 1983.

27. Motorola Inc., 8-Bit Microprocessor & Peripheral Data, Motorola Inc., 1983.

28. Motorola Inc., MC6809-MC6809E Microprocessor Programming Manual, Motorola Inc., 1983.

29. O'Connor, Joseph Allan, "A CRT-Based State Transition Language for Industrial Sequential Control", M. S. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA., June 1981.

30. Ramamoorthy, C. V. and Gary S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets", IEEE Transactions on Software Engineering, Vol SE-6, No 5, pp.440-449, September 1980.

31. Schmitt, L. E., "PC versus CNC - Which Do You Choose?", IEEE Transactions On Industry Applications, Vol IA-20, No 5, pp.1141-1145, September/October 1984.

32. Smith, Gordon H., "Converting Relay Logic to Software", Machine Design, Vol 50, No 19, pp.93-99, August 24, 1978.

33. Struger, O. J. and J. H. Christensen, "Languages for Programmable Controllers", IEEE Industry Applications Society Meeting, pp. 1261-1265, October 1982.

34. Sutherland, H. A., B. K. Bose, and C.B. Somuah, "A State Language for the Sequencing in a Hybrid Electric Vehicle", IEEE 1982 IECON Proceedings, 1982 Industrial Electronics Society Annual Conference, Palo Alto, CA., pp.15-19, November 15-19, 1982.

35. Takahashi, Hideyuki, "An Automatic-Controller Description Language", IEEE Transactions on Software Engineering, Vol SE-6, No 1, pp.53-64, January 1980.

36. Thomas, David W., "EE 467 Computer Project", Michigan Technological University, Houghton, MI., Advisor: T. A. Grzelak, February 1985.

37. Waite, Ralph, "PC Study Reveals User Wants and Needs",Instruments & Control Systems, Vol 56, No 2, pp.59-60, February 1983.

38. Walker, Geoffry Robert, "The Formal Specification and Control of Sequential Processes Using the State Method", M. S. Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA., June 1978.

39. Wolfe, Peter DeMarques, "A Computer Language for Control of Autonomous Industrial Equipment", M. S. Thesis, Department of Electrical and Computer Science, Massachusetts Institute of Technology, Cambridge, MA., June 1973.