# Algorithmic State Machines (ASM)

## 8-1 INTRODUCTION

The binary information stored in a digital system can be classified as either data or control information. Data are discrete elements of information that are manipulated to perform arithmetic, logic, shift, and other similar data-processing tasks. These operations are implemented with digital components such as adders, decoders, multiplexers, counters, and shift registers. Control information provides command signals that supervise the various operations in the data section in order to accomplish the desired data-processing tasks. The logic design of a digital system can be divided into two distinct parts. One part is concerned with the design of the digital circuits that perform the data-processing operations. The other part is concerned with the design of the control circuit that supervises the operations and their sequence.

The relationship between the control logic and the data processor in a digital system is shown in Fig. 8-1. The data processor subsystem manipulates data in registers according to the system's requirements. The control logic initiates properly sequenced commands to the data processor. The control logic uses status conditions from the data processor to serve as decision variables for determining the sequence of control signals.

The control logic that generates the signals for sequencing the operations in the data processor is a sequential circuit whose internal states dictate the control commands for the system. At any given time, the state of the sequential control initiates a prescribed set of commands. Depending on status conditions and other external inputs, the sequential control goes to the next state to initiate other operations. The digital circuits

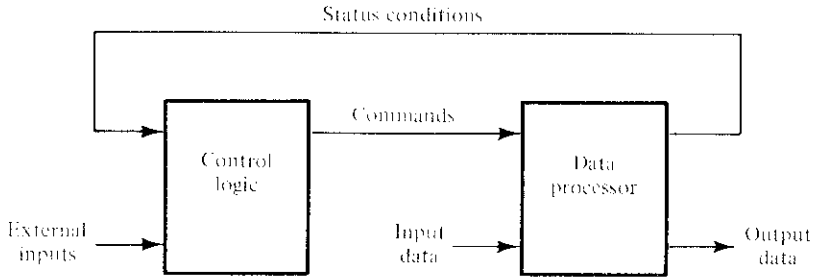



**FIGURE 8-1**
*Control and data-processor interaction*

that act as the control logic provide a time sequence of signals for initiating the operations in the data processor and also determine the next state of the control subsystem itself.

The control sequence and data-processing tasks of a digital system are specified by means of a hardware algorithm. An algorithm consists of a finite number of procedural steps that specify how to obtain a solution to a problem. A hardware algorithm is a procedure for implementing the problem with a given piece of equipment. The most challenging and creative part of digital design is the formulation of hardware algorithms for achieving required objectives.

A flow chart is a convenient way to specify the sequence of procedural steps and decision paths for an algorithm. A flow chart for a hardware algorithm translates the word statement to an information diagram that enumerates the sequence of operations together with the conditions necessary for their execution. A special flow chart that has been developed specifically to define digital hardware algorithms is called an *algorithmic state machine* (ASM) chart. A *state machine* is another term for a sequential circuit, which is the basic structure of a digital system.

The ASM chart resembles a conventional flow chart, but is interpreted somewhat differently. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship. The ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next. It is specifically adapted to specify accurately the control sequence and data-processing operations in a digital system, taking into consideration the constraints of digital hardware.

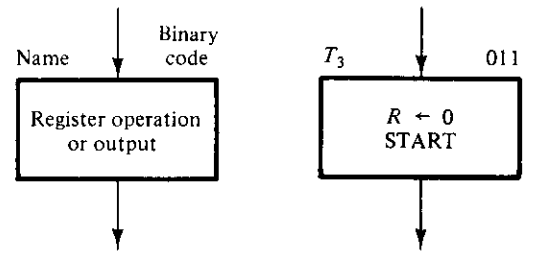
This chapter presents a method of digital logic design using the ASM chart. The various blocks that make up the chart are first defined. The timing relationship between the blocks is then explained by example. Various ways of implementing the control logic are discussed together with examples of ASM charts and the corresponding digital systems that they represent.

## 8-2 ASM CHART

The ASM chart is a special type of flow chart suitable for describing the sequential operations in a digital system. The chart is composed of three basic elements: the state box, the decision box, and the conditional box. A state in the control sequence is indicated by a state box, as shown in Fig. 8-2. The shape of the state box is a rectangle within which are written register operations or output signal names that the control generates while being in this state. The state is given a symbolic name, which is placed at the upper left corner of the box. The binary code assigned to the state is placed at the upper right corner. Figure 8-2(b) shows a specific example of a state box. The state has the symbolic name $T_3$, and the binary code assigned to it is 011. Inside the box is written the register operation $R \leftarrow 0$, which indicates that register $R$ is to be cleared to 0 when the system is in state $T_3$. The START name inside the box may indicate, for example, an output signal that starts a certain operation.

The decision box describes the effect of an input on the control subsystem. It has a diamond-shaped box with two or more exit paths, as shown in Fig. 8-3. The input condition to be tested is written inside the box. One exit path is taken if the condition is true and another when the condition is false. When an input condition is assigned a binary value, the two paths are indicated by 1 and 0.

The state and decision boxes are familiar from use in conventional flow charts. The third element, the conditional box, is unique to the ASM chart. The oval shape of the conditional box is shown in Fig. 8-4. The rounded corners differentiate it from the state box. The input path to the conditional box must come from one of the exit paths of a decision box. The register operations or outputs listed inside the conditional box are generated during a given state provided that the input condition is satisfied. Figure 8-5 shows an example with a conditional box. The control generates a START output signal when in state $T_1$. While in state $T_1$, the control checks the status of input $E$. If $E = 1$, then $R$ is cleared to 0; otherwise, $R$ remains unchanged. In either case, the next state is $T_2$.




(a) General description (b) Specific example

**FIGURE 8-2**
State box

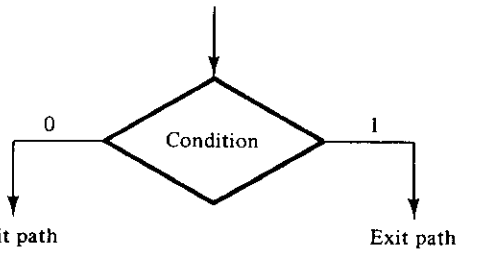



**FIGURE 8-3**
Decision box

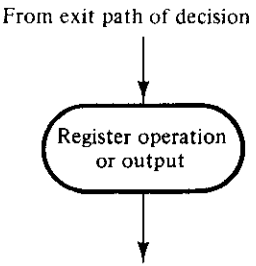



**FIGURE 8-4**
Conditional box

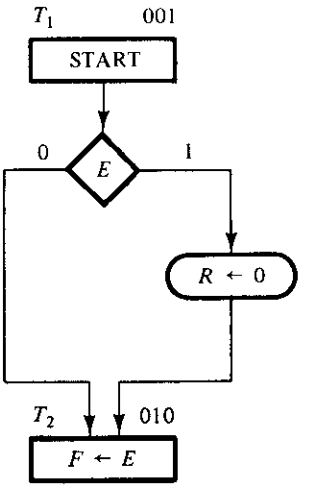



**FIGURE 8-5**
Example with conditional box

## ASM Block

An ASM block is a structure consisting of one state box and all the decision and conditional boxes connected to its exit path. An ASM block has one entrance and any number of exit paths represented by the structure of the decision boxes. An ASM chart consists of one or more interconnected blocks. An example of an ASM block is shown in Fig. 8-6. Associated with state $T_1$ are two decision boxes and one conditional box. The diagram distinguishes the block with dashed lines around the entire structure, but

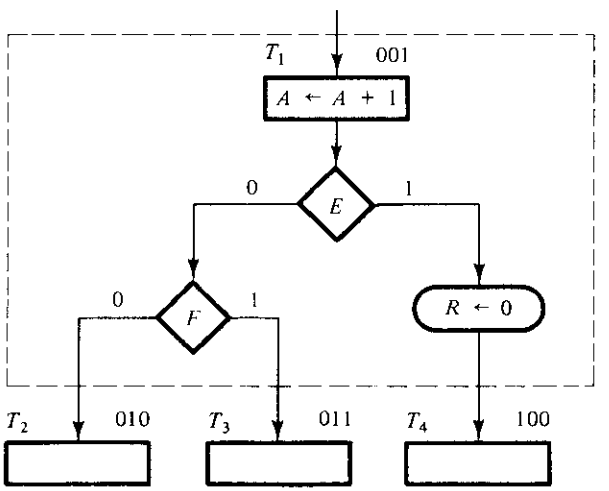



**FIGURE 8-6**
ASM block

this is not usually done, since the ASM chart uniquely defines each block from its structure. A state box without any decision or conditional boxes constitutes a simple block.

Each block in the ASM chart describes the state of the system during one clock-pulse interval. The operations within the state and conditional boxes in Fig. 8-6 are executed with a common clock pulse while the system is in state $T_1$. The same clock pulse also transfers the system controller to one of the next states, $T_2$, $T_3$, or $T_4$, as dictated by the binary values of $E$ and $F$.

The ASM chart is very similar to a state diagram. Each state block is equivalent to a state in a sequential circuit. The decision box is equivalent to the binary information written along the directed lines that connect two states in a state diagram. As a consequence, it is sometimes convenient to convert the chart into a state diagram and then use sequential-circuit procedures to design the control logic. As an illustration, the ASM chart of Fig. 8-6 is drawn as a state diagram in Fig. 8-7. The three states are symbolized by circles, with their binary values written inside the circles. The directed lines indicate the conditions that determine the next state. The unconditional and conditional operations that must be performed are not indicated in the state diagram.

## Register Operations

A digital system is quite often defined by the registers it contains and the operations that are performed on the data stored in them. A *register* in its broader sense includes storage registers, shift registers, counters, and single flip-flops. Examples of register operations are shift, increment, add, clear, and data transfer. It is sometimes convenient to adopt a suitable notation to describe the operations performed among the registers.

Table 8-1 gives examples of symbolic notation for some register operations. A register is designated by one or more capital letters such as $A$, $B$, or $RA$. The individual cells or flip-flops within an $n$-bit register are numbered in sequence from 1 to $n$ or from 0 to $n - 1$. A single flip-flop is considered a 1-bit register. The transfer of data from one register to another is symbolized by a directed arrow that denotes a transfer of contents

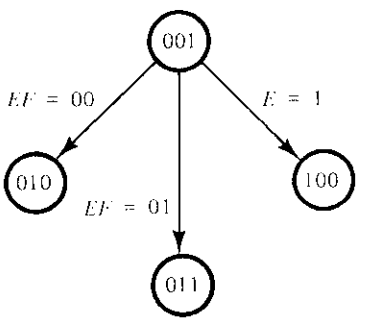



**FIGURE 8-7**
State-diagram equivalent to the ASM chart of Fig. 8-6

**TABLE 8-1**
**Symbolic Notation for Register Operations**

| Symbolic Notation | Description |
|---|---|
| $A \leftarrow B$ | Transfer contents of register $B$ into register $A$ |
| $R \leftarrow 0$ | Clear register $R$ |
| $F \leftarrow 1$ | Set flip-flop $F$ to 1 |
| $A \leftarrow A + 1$ | Increment register $A$ by 1 (count-up) |
| $A \leftarrow A - 1$ | Decrement register $A$ by 1 (count-down) |
| $A \leftarrow A + B$ | Add contents of register $B$ to register $A$ |

from the source register to the destination register. The register-clear operation is symbolized by a transfer of 0 into the register. A single flip-flop can be set to 1 or cleared to 0. To increment a register by 1, it is necessary that the register be able to count up as in a binary counter. The decrement operation requires a count-down counter. The contents of two registers can be added by means of an adder circuit. Some operations, such as the shift operation, do not have a known symbol. In such a case, we will use the words "shift right $R$" to denote a shift right of register $R$.

## 8-3 TIMING CONSIDERATIONS

The timing for all registers and flip-flops in a digital system is controlled by a master-clock generator. The clock pulses are applied not only to the registers of the data-processor subsection, but also to all the flip-flops in the control logic. Inputs are also synchronized with the clock pulses because they are normally generated as outputs of another circuit that uses the same clock signals. If the input signal changes at an arbitrary time independent of the clock, we call it an asynchronous input. Asynchronous inputs may cause a variety of problems, as discussed in Chapter 9. To simplify the design, we will assume that all inputs are synchronized with the clock and change state in response to an edge transition of the clock pulse. Similarly, any output that is a function of the present state and a synchronous input will also be synchronous.

The major difference between a conventional flow chart and an ASM chart is in interpreting the time relationship among the various operations. For example, if Fig. 8-6 were a conventional flow chart, then the listed operations would be considered to follow one after another in time sequence: Register $A$ is first incremented and only then is $E$ evaluated. If $E = 1$, then register $R$ is cleared and control goes to state $T_4$. Otherwise, if $E = 0$, the next step is to evaluate $F$ and go to state $T_2$ or $T_3$. In contrast, an ASM chart considers the entire block as one unit. All the operations that are specified within the block must occur in synchronism during the edge transition of the same clock pulse while the system changes from $T_1$ to the next state. This is presented pictorially in Fig. 8-8. We assume positive-edge triggering of all flip-flops. The first positive transition of the clock transfers the control circuit into state $T_1$. While in state $T_1$, the control circuits check inputs $E$ and $F$ and generate appropriate signals accordingly. The

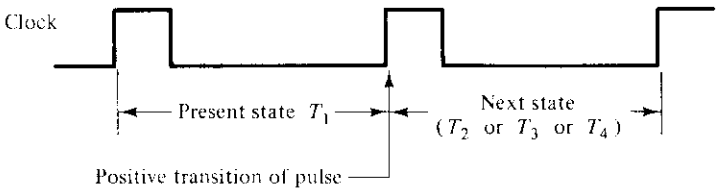



**FIGURE 8-8**
Transition between states

following operations occur simultaneously during the next positive transition of the clock pulse:

1. Register $A$ is incremented.
2. If $E = 1$, register $R$ is cleared.
3. Depending on the values of $E$ and $F$, control is transferred to next state, $T_2$ or $T_3$ or $T_4$.

Note that the operations in the data-processor subsection and the change of state in the control logic occur at the same time.

We will now demonstrate the time relationship between the components of an ASM chart by going over a specific design example. The example does not have any known application and is merely formulated to show the usefulness of the ASM Chart. We start from the initial specifications and proceed with the development of an appropriate ASM chart from which the digital hardware can be derived.

### Design Example

We wish to design a digital system with two flip-flops, $E$ and $F$, and one 4-bit binary counter, $A$. The individual flip-flops in $A$ are denoted by $A_4$, $A_3$, $A_2$, and $A_1$, with $A_4$ holding the most significant bit of the count. A start signal $S$ initiates the system operation by clearing the counter $A$ and flip-flop $F$. The counter is then incremented by 1 starting from the next clock pulse and continues to increment until the operations stop. Counter bits $A_3$ and $A_4$ determine the sequence of operations:

If $A_3 = 0$, $E$ is cleared to 0 and the count continues.

If $A_3 = 1$, $E$ is set to 1; then if $A_4 = 0$, the count continues, but if $A_4 = 1$, $F$ is set to 1 on the next clock pulse and the system stops counting.

### ASM Chart

The ASM chart is shown in Fig. 8-9. When no operations are performed, the system is in the initial state $T_0$, waiting for the start signal $S$. When input $S$ is equal to 1, counter $A$ and flip-flop $F$ are cleared to 0 and the controller goes to state $T_1$. Note the conditional box that follows the decision box for $S$. This means that the counter and flip-flop will be cleared during $T_0$ if $S = 1$, and at the same time, control transfers to state $T_1$.

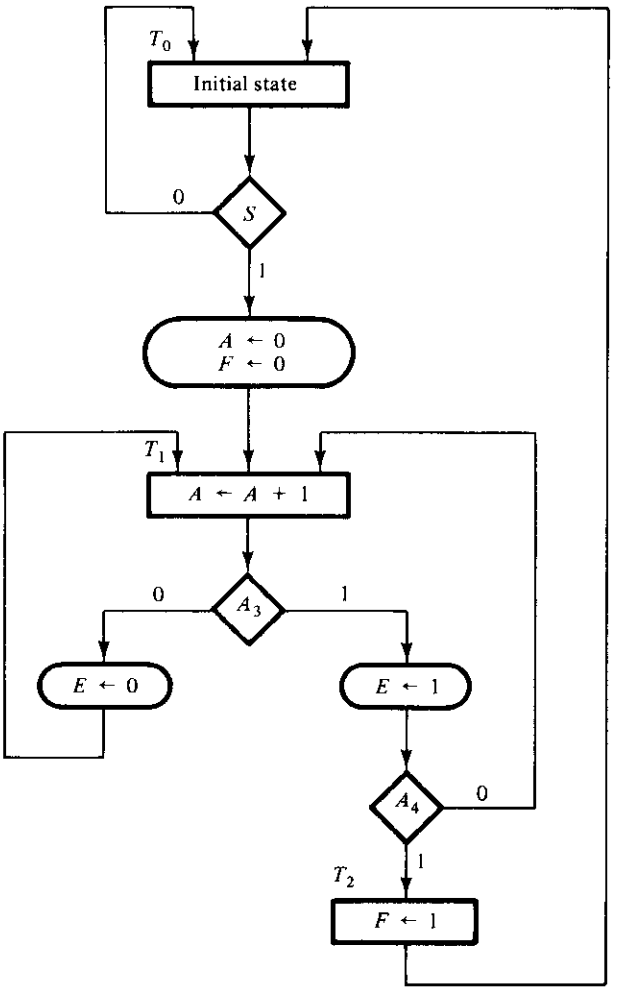



**FIGURE 8-9**
ASM chart for design example

The block associated with state $T_1$ has two decision boxes and two conditional boxes. The counter is incremented with every clock pulse. At the same time, one of three possible operations occur during the same clock pulse transition:

Either $E$ is cleared and control stays in state $T_1$ ($A_3 = 0$); or
$E$ is set and control stays in state $T_1$ ($A_3A_4 = 10$); or
$E$ is set and control goes to state $T_2$ ($A_3A_4 = 11$).

When the control is in state $T_2$, flip-flop $F$ is set to 1 and the circuit goes back to its initial state, $T_0$.

The ASM chart consists of three states and three blocks. The block associated with $T_0$ consists of the state box, one decision box, and one conditional box. The block associated with $T_2$ consists of only the state box. The control logic has one external input, $S$, and two status inputs, $A_3$ and $A_4$.

## Timing Sequence

Every block in an ASM chart specifies the operations that are to be performed during one common clock pulse. The operations specified within the state and conditional boxes in the block are performed in the data-processor subsection. The change from one state to the next is performed in the control logic. In order to appreciate the timing relationship involved, we will list the step-by-step sequence of operations after each clock pulse from the time the start signal occurs until the system goes back to its initial state.

Table 8-2 shows the binary values of the counter and the two flip-flops after every clock pulse. The table also shows separately the status of $A_3$ and $A_4$ as well as the present state of the controller. We start with state $T_1$ right after the input signal $S$ has caused the counter and flip-flop $F$ to be cleared. The value of $E$ is assumed to be 1, because $E$ is equal to 1 at $T_0$ (as shown at the end of the table) and because $E$ does not change during the transition from $T_0$ to $T_1$. The system stays in state $T_1$ during the next thirteen clock pulses. Each pulse increments the counter and either clears or sets $E$. Note the relationship between the time at which $A_3$ becomes a 1 and the time at which $E$

**TABLE 8-2**
**Sequence of Operations for Design Example**

| Counter | | | | Flip-Flops | | | |
|---|---|---|---|---|---|---|---|
| $A_4$ | $A_3$ | $A_2$ | $A_1$ | $E$ | $F$ | Conditions | State |
| 0 | 0 | 0 | 0 | 1 | 0 | $A_3 = 0, A_4 = 0$ | $T_1$ |
| 0 | 0 | 0 | 1 | 0 | 0 | | |
| 0 | 0 | 1 | 0 | 0 | 0 | | |
| 0 | 0 | 1 | 1 | 0 | 0 | | |
| 0 | 1 | 0 | 0 | 0 | 0 | $A_3 = 1, A_4 = 0$ | |
| 0 | 1 | 0 | 1 | 1 | 0 | | |
| 0 | 1 | 1 | 0 | 1 | 0 | | |
| 0 | 1 | 1 | 1 | 1 | 0 | | |
| 1 | 0 | 0 | 0 | 1 | 0 | $A_3 = 0, A_4 = 1$ | |
| 1 | 0 | 0 | 1 | 0 | 0 | | |
| 1 | 0 | 1 | 0 | 0 | 0 | | |
| 1 | 0 | 1 | 1 | 0 | 0 | | |
| 1 | 1 | 0 | 0 | 0 | 0 | $A_3 = 1, A_4 = 1$ | |
| 1 | 1 | 0 | 1 | 1 | 0 | | $T_2$ |
| 1 | 1 | 0 | 1 | 1 | 1 | | $T_0$ |

is set to 1. When $A = 0011$, the next clock pulse increments the counter to 0100, but that same clock pulse sees the value of $A_3$ as 0, so $E$ is cleared. The next pulse changes the counter from 0100 to 0101, and now $A_3$ is initially equal to 1, so $E$ is set to 1. Similarly, $E$ is cleared to 0 not when the count goes from 0111 to 1000, but when it goes from 1000 to 1001, which is when $A_3$ is 0 in the present value of the counter.

When the count reaches 1100, both $A_3$ and $A_4$ are equal to 1. The next clock pulse increments $A$ by 1, sets $E$ to 1, and transfers control to state $T_2$. Control stays in $T_2$ for only one clock period. The pulse transition associated with $T_2$ sets flip-flop $F$ to 1 and transfers control to state $T_0$. The system stays in the initial state $T_0$ as long as $S$ is equal to 0.

From observation of Table 8-2, it may seem that the operations performed on $E$ are delayed by one clock pulse. This is the difference between an ASM chart and a conventional flow chart. If Fig. 8-9 were a conventional flow chart, we would assume that $A$ is first incremented and the incremented value would have been used to check the status of $A_3$. The operations that are performed in the digital hardware as specified by a block in the ASM chart occur during the same clock period and not in a sequence of operations following each other in time, as is usually interpreted in a conventional flow chart. Thus, the value of $A_3$ to be considered in the decision box is taken from the value of the counter in the present state and before it is incremented. This is because the decision box for $E$ belongs with the same block as state $T_1$. The digital circuits in the control generate the signals for all the operations specified in the present block prior to the arrival of the next clock pulse. The next clock transition executes all the operations in the registers and flip-flops, including the flip-flops in the controller that determine the next state.

### Data Processor

The ASM chart gives all the information necessary to design the digital system. The requirements for the design of the data-processor subsystem are specified inside the state and conditional boxes. The control logic is determined from the decision boxes and the required state transitions. A diagram showing the hardware for the design example is shown in Fig. 8-10. The control subsystem is shown with only its inputs and outputs. The detailed design of the control is considered in the next section. The data processor consists of a 4-bit binary counter, two flip-flops, and a number of gates. The counter is similar to the one shown in Fig. 7-17 except that additional gates are required for the synchronous clear operation. The counter is incremented with every clock pulse when control is in state $T_1$. It is cleared only when control is at state $T_0$ and $S$ is equal to 1. This conditional operation requires an AND gate to guarantee that both conditions are present. The other two conditional operations use two other AND gates for setting or clearing flip-flop $E$. Flip-flop $F$ is set unconditionally during state $T_2$. Note that all flip-flops and registers including the flip-flops in the control use a common clock-pulse source.

This example demonstrates a method of digital design using the ASM chart. The design of the data-processor subsystem requires an interpretation of the register operations and their implementation by means of the components introduced in Chapters 5 and 7

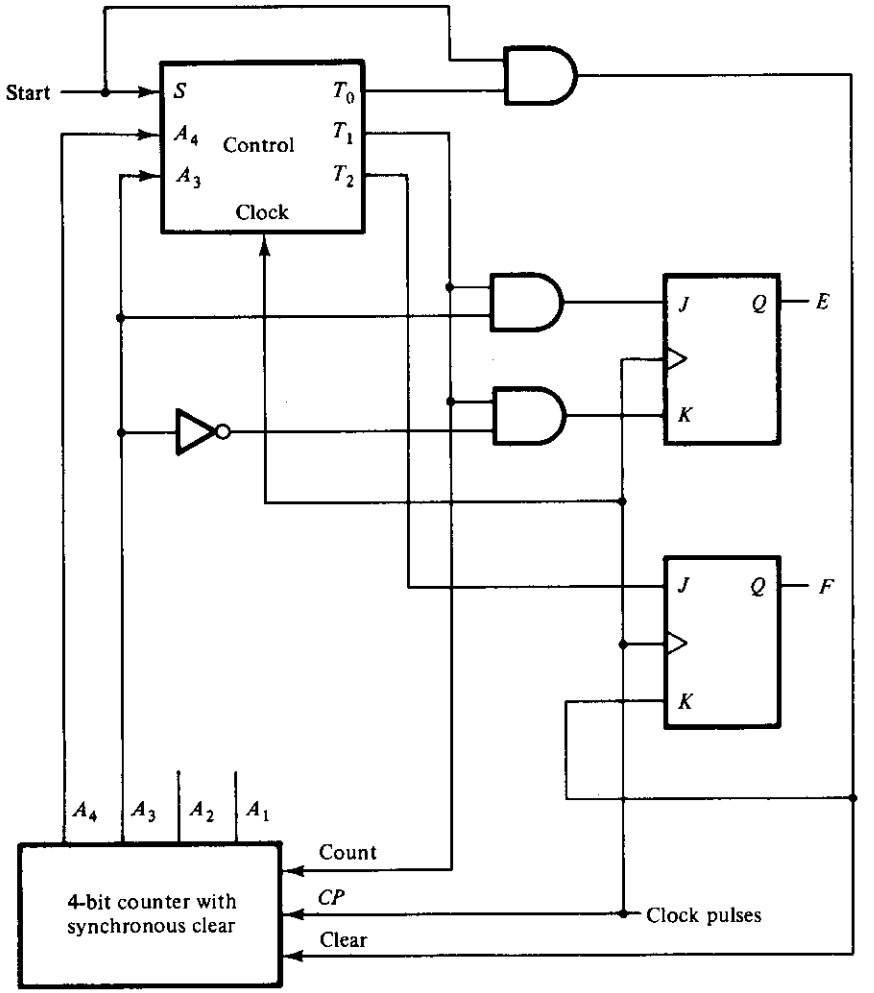



**FIGURE 8-10**
Data processor for design example

such as registers, counters, multiplexers, and adders. The design of the control subsystem requires the application of design procedures based on the theory of sequential logic. The next three sections present some of the alternatives that are available for designing the control logic.

## 8-4 CONTROL IMPLEMENTATION

The control section of a digital system is essentially a sequential circuit that can be designed by the procedure outlined in Chapter 6. However, in most cases, this method is impractical because of the large number of states and inputs that a typical control cir-

cuit may have. Except for very simple controllers, the design method that uses state and excitation tables is cumbersome and difficult to manage. Experienced digital designers use specialized methods for control logic design that may be considered an extension of the classical sequential method combined with other simplified assumptions. Two of these specialized methods are presented in this section, and a third method is explained in Section 8-5. Another alternative is to use a ROM or PLA to design the control logic. This is covered in Section 8-6.

## State Table

As mentioned previously, the ASM chart resembles a state diagram, with each state box representing a state. The state diagram can be converted into a state table from which the sequential circuit of the controller can be designed. First, we must assign binary values to each state in the ASM chart. For $n$ flip-flops in the control sequential circuit, the ASM chart can accommodate up to $2^n$ states. A chart with three or four states requires a sequential circuit with two flip-flops. With five to eight states, there is a need for three flip-flops. Each combination of flip-flop values represents a binary number for one of the states.

A state table for a controller is a list of present states and inputs and their corresponding next states and outputs. In most cases, there are many don't-care input conditions that must be included, so it is advisable to arrange the state table to take this into consideration. In order to clarify the procedure, we will illustrate by obtaining the state table of the controller defined in the example of the previous section.

The ASM chart of the design example is shown in Fig. 8-9. We assign the following binary values to the three states: $T_0 = 00$, $T_1 = 01$, $T_2 = 11$. Binary state 10 is not used and will be treated as a don't-care condition. The state table corresponding to the ASM chart is shown in Table 8-3. Two flip-flops are needed, and they are labeled $G_1$ and $G_2$. There are three inputs and three outputs. The inputs are taken from the conditions in the decision boxes. The outputs are equivalent to the present state of the control. Note that there is a row in the table for each possible transition between states. Initial state 00 goes to state 01 or stays in 00, depending on the value of input $S$. The

**TABLE 8-3**
**State Table for Control of Fig. 8-10**

| Present-State Symbol | Present State | | Inputs | | | Next State | | Outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $G_1$ | $G_2$ | $S$ | $A_3$ | $A_4$ | $G_1$ | $G_2$ | $T_0$ | $T_1$ | $T_2$ |
| $T_0$ | 0 | 0 | 0 | $X$ | $X$ | 0 | 0 | 1 | 0 | 0 |
| $T_0$ | 0 | 0 | 1 | $X$ | $X$ | 0 | 1 | 1 | 0 | 0 |
| $T_1$ | 0 | 1 | $X$ | 0 | $X$ | 0 | 1 | 0 | 1 | 0 |
| $T_1$ | 0 | 1 | $X$ | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| $T_1$ | 0 | 1 | $X$ | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| $T_2$ | 1 | 1 | $X$ | $X$ | $X$ | 0 | 0 | 0 | 0 | 1 |

other two inputs are marked with don't-care $X$'s, as they do not determine the next state in this case. While the system is in binary state 00, the control provides an output labeled $T_0$ to initiate the required register operations. The transition from binary state 01 depends on inputs $A_3$ and $A_4$. The system goes to binary state 11 only if $A_3A_4 = 11$; otherwise, it remains in binary state 01. Finally, binary state 11 goes to 00 independently of the input variables.

This example demonstrates a state table for a sequential controller. Note again the large number of don't-care conditions under the inputs. The number of rows in the state table is equal to the number of distinct paths between the states in the ASM chart.

### Logic Diagram with *JK* Flip-Flops

The procedure for designing a sequential circuit starting from a state table is presented in Section 6-7. This procedure requires that we obtain the excitation table of the flip-flop inputs and then simplify the combinational-circuit part of the sequential circuit. If we apply this procedure to Table 8-3, we will need to use five-variable maps (see Fig. 3-12) to simplify the input functions. This is because there are five variables listed under the "present-state" and "input" columns. Since this procedure was explained in Chapter 6, we will not show the detail work here. The flip-flop input functions obtained by this method, if we assume *JK* flip-flops, are

$$JG_1 = G_2A_3A_4 \qquad JG_2 = S$$
$$KG_1 = 1 \qquad KG_2 = G_1$$

To derive the three output functions, we can utilize the fact that binary state 10 is not used and obtain the following simplified functions:

$$T_0 = G_2'$$
$$T_1 = G_1'G_2$$
$$T_2 = G_1$$

The logic diagram of the control is shown in Fig. 8-11. This circuit replaces the control block in Fig. 8-10.

### *D* Flip-Flops and Decoder

When the number of flip-flops plus inputs in a state table is greater than 5, it is necessary to use large maps to simplify the input functions. This is cumbersome and difficult to achieve, as explained in Chapter 3. Therefore, it is necessary to find alternative ways to design controllers except when they are very simple. One possibility is to use *D*-type flip-flops and obtain the input functions directly from the state table without the need of an excitation table. This is because the next state is the same as the input requirement for the *D* flip-flops (see Section 6-7). To design the sequential circuit with *D* flip-flops,

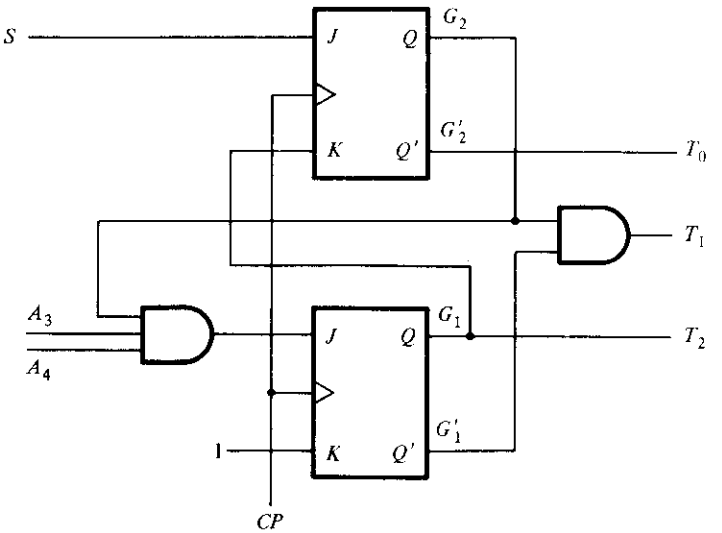



**FIGURE 8-11**
Logic diagram of control using *JK* flip-flops

it is necessary to go over the next state column in the state table and derive all the conditions that must set each flip-flop to 1. From Table 8-3, we note that the next state column of $G_1$ has a single 1 in the fifth row. The $D$ input of flip-flop $G_1$ must be equal to 1 during present state $T_1 = G_1' G_2$ when both inputs $A_3$ and $A_4$ are equal to 1. This is expressed with the $D$ flip-flop input function

$$DG_1 = G_1' G_2 A_3 A_4$$

Similarly, the next state column of $G_2$ has four 1's, and the condition for setting this flip-flop is

$$DG_2 = G_1' G_2' S + G_1' G_2$$

We can go one step further and insert a decoder at the output of the flip-flops to obtain the necessary three outputs, $T_0$, $T_1$, and $T_2$. Then, instead of using the flip-flop outputs as the present-state condition, we might as well use the outputs of the decoder to supply this information. The input functions to the $D$ flip-flops can now be expressed as follows:

$$DG_1 = A_3 A_4 T_1$$

$$DG_2 = ST_0 + T_1$$

The alternative logic diagram is shown in Fig. 8-12. The decoder provides the three control outputs, and these outputs are also used to determine the next state of each flip-flop. The second control circuit requires more components than the first, but it has the advantage that it can be derived by inspection from the state table.

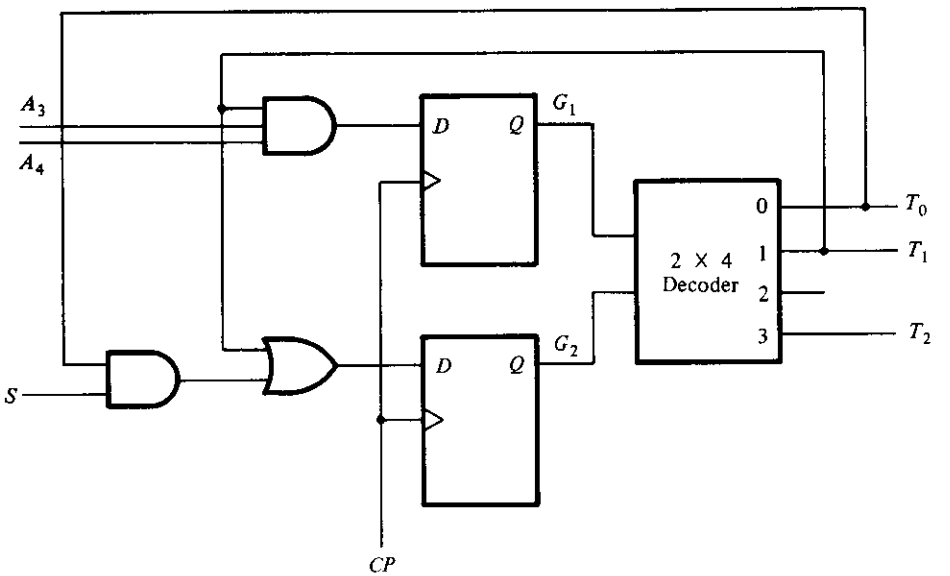



**FIGURE 8-12**
Alternate logic diagram of control using *D* flip-flops and a decoder

## One Flip-Flop per State

Another possible method of control logic design is to use one flip-flop per state in the sequential circuit. Only one flip-flop is set at any particular time; all others are cleared to 0. The single bit is made to propagate from one flip-flop to the other under the control of decision logic. In such an array, each flip-flop represents a state that is activated only when the control bit is transferred to it.

It is obvious that this method does not use a minimum number of flip-flops for the sequential circuit. In fact, it uses a maximum number of flip-flops. For example, a sequential circuit with 12 states requires a minimum of four flip-flops. Yet by this method, the control circuit needs 12 flip-flops, one for each state.

A control organization that uses one flip-flop per state has the convenient characteristic that the circuit can be derived directly from the state diagram without the need of state or excitation tables. Consider, for example, the state diagram of Fig. 8-13. This diagram is equivalent to the ASM chart of the design example from Fig. 8-9 as far as the control-state transitions are concerned. Since the diagram has three states, we assign three flip-flops to the circuit and label them $T_0$, $T_1$, and $T_2$. The controller can be designed by inspection from the state diagram if *D*-type flip-flops are used. The Boolean function for setting the flip-flop is determined from the present-state and the input conditions along the directed lines. For example, flip-flop $T_0$ is set with the next clock pulse if present state $T_2 = 1$ or if present state $T_0 = 1$ and input $S = 0$. This condition is defined by the flip-flop input function:

$$DT_0 = T_2 + S'T_0$$

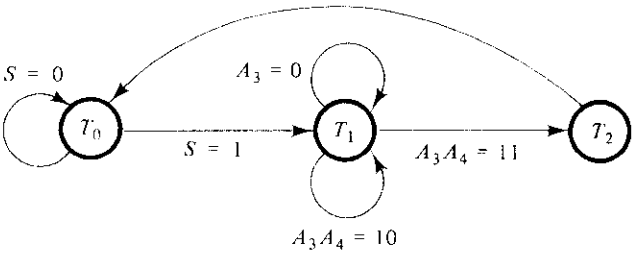



**FIGURE 8-13**
State diagram of a controller

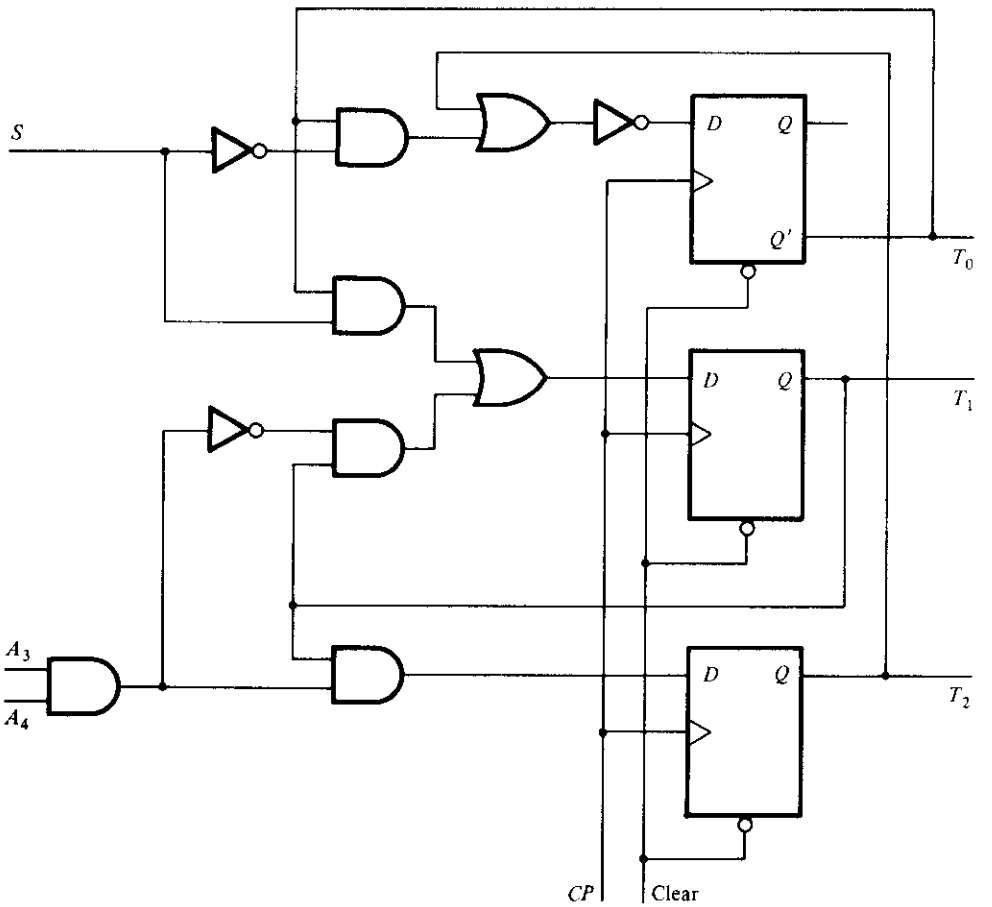



**FIGURE 8-14**
Third alternate logic diagram of control using one flip-flop per state

where $DT_0$ designates the $D$ input of flip-flop $T_0$. In fact, the condition for setting a flip-flop to 1 is obtained directly from the state diagram from the condition specified in the directed lines going into the corresponding flip-flop state ANDed with the previous flip-flop state. If there is more than one directed line going into a state, all conditions must be ORed. Using this procedure for the other two flip-flops, we obtain the input functions:

$$DT_1 = ST_0 + A_3'T_1 + A_3A_4'T_1 = ST_0 + (A_3A_4)'T_1$$

$$DT_2 = A_3A_4T_1$$

The logic diagram is shown in Fig. 8-14. It consists of three $D$ flip-flops, $T_0$, $T_1$, and $T_2$, and the associated gates specified by the input functions listed before.

Initially, flip-flop $T_0$ must be set to 1 and all other flip-flops cleared to 0 so that the flip-flop representing the initial state is equal to 1 and all other states equal to 0. Once started, the one-flip-flop-per-state controller will propagate itself from state to state in the proper manner. For a register with a common asynchronous clear input, as shown in Fig. 8-14, all flip-flops, including the $Q$ output of $T_0$, are cleared to 0. Taking the output of $T_0$ from the complement output $Q'$ provides the required initial 1 signal for $T_0$. In order to keep $Q'$ as the output of $T_0$, it is necessary that the input function to the $D$ input be complemented. This is done by the extra inverter that is placed at the $D$ input of flip-flop $T_0$.

## 8-5 DESIGN WITH MULTIPLEXERS

One major goal of control-logic design is the development of a circuit that implements the desired control sequence in a logical and straightforward manner. The attempt to minimize the number of gates tends to produce an irregular network, making it difficult for anyone but the designer to identify the sequence of events the control undergoes. As a consequence, it is difficult to alter, service, or maintain the equipment after the initial design. The sequence of states in the control should be clearly evident from the circuit configuration even if this requires additional components and results in a non-minimal circuit. The multiplexer method is such an implementation.

The control circuit shown in Fig. 8-12 consists of three components: the flip-flops that hold the binary state value, the decoder that generates the control outputs, and the gates that determine the next state. We now replace the gates with multiplexers and use a register for the individual flip-flops. This design method results in a regular pattern of three levels of components. The first level consists of multiplexers that determine the next state of the register. The second level contains a register that holds the present binary state. The third level has the decoder that provides a separate output for each control state.

Consider, for example, the ASM chart of Fig. 8-15. It consists of four states and four control inputs. The state boxes are left empty in this case because we are interested only in the control sequence, which is independent of the register operations. The

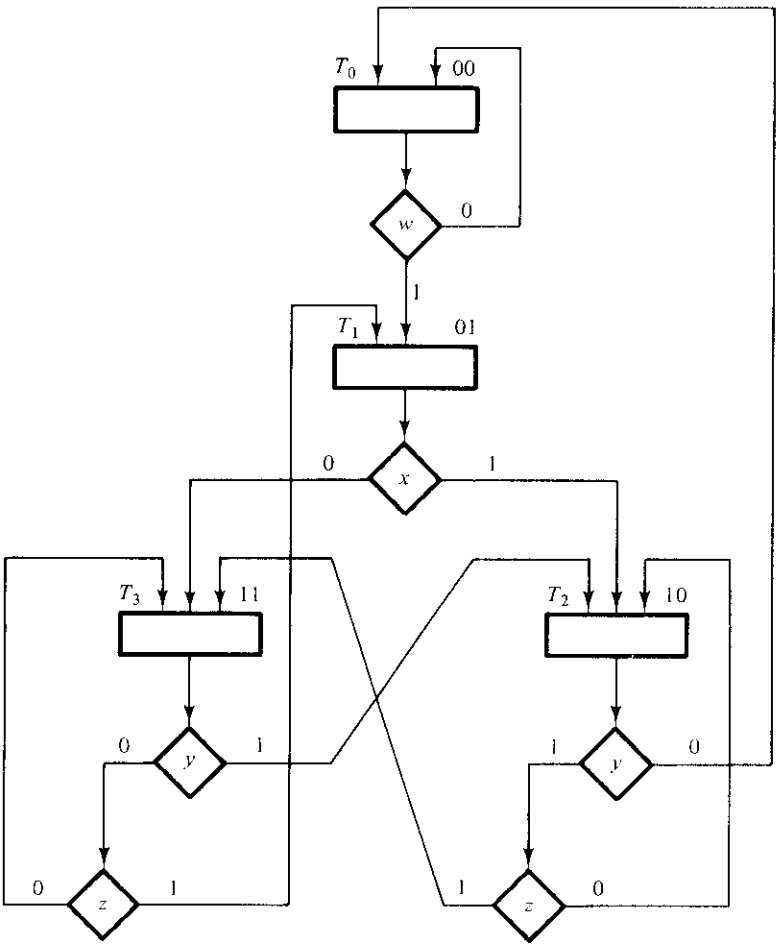



**FIGURE 8-15**
Example of ASM chart with four control inputs

binary assignment for each state is indicated at the upper right corner of the state boxes. The decision boxes specify the state transitions as a function of the four control inputs, $w$, $x$, $y$, and $z$. The three-level control implementation is shown in Fig. 8-16. It consists of two multiplexers, MUX1 and MUX2; a register with two flip-flops, $G_1$ and $G_2$; and a decoder with four outputs. The outputs of the register are applied to the decoder inputs and also to the select inputs of the multiplexers. In this way, the present state of the register is used to select one of the inputs from each multiplexer. The outputs of the multiplexers are then applied to the $D$ inputs of $G_1$ and $G_2$. The purpose of each multiplexer is to produce an input to its corresponding flip-flop equal to the binary value of the next state.

The inputs of the multiplexers are determined from the decision boxes and state

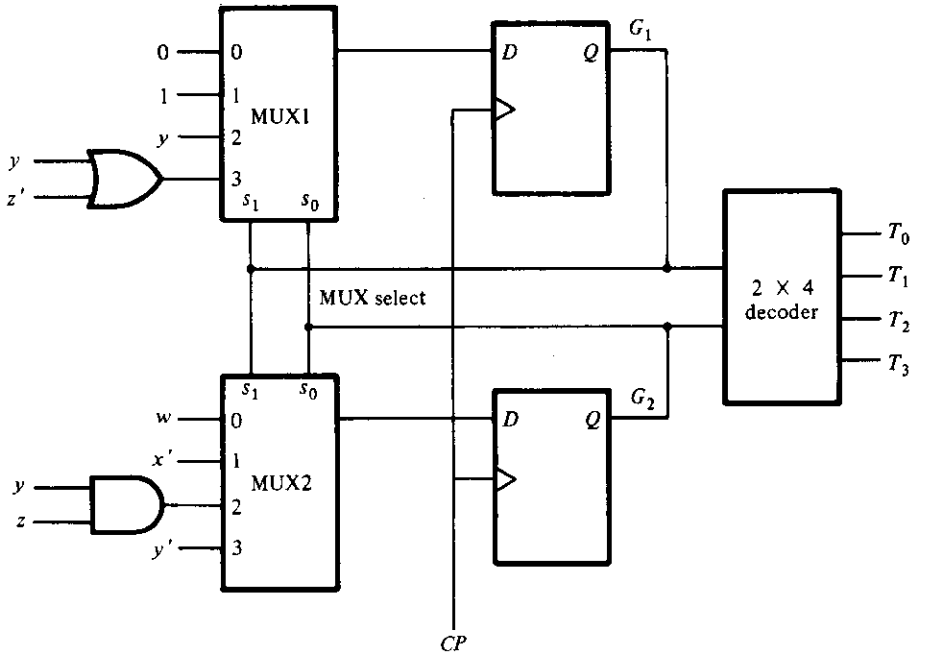



**FIGURE 8-16**
Control implementation with multiplexers

transitions given in the ASM chart. For example, state 00 stays at 00 or goes to 01, depending on the value of input $w$. Since the next state of $G_1$ is 0 in either case, we place a signal equivalent to logic-0 in MUX1 input 0. The next state of $G_2$ is 0 if $w = 0$ and 1 if $w = 1$. Since the next state of $G_2$ is equal to $w$, we apply control intput $w$ to MUX2 input 0. What this means is that when the select inputs of the multiplexers are equal to present state 00, the outputs of the multiplexers provide the binary value that is transferred to the register during the next clock pulse.

To facilitate the evaluation of the multiplexer inputs, we prepare a table showing the input conditions for each possible transition in the ASM chart. Table 8-4 gives this information for the ASM chart of Fig. 8-15. There are two transitions from present state 00 or 01 and three transitions from present state 10 or 11. These are separated by horizontal lines across the table. The input conditions listed in the table are obtained from the decision boxes in the ASM chart. For example, from Fig. 8-15, we note that present state 01 will go to next state 10 if $x = 1$ or to next state 11 if $x = 0$. In the table, we mark these input conditions as $x$ and $x'$, respectively. The two columns under "multiplexer inputs" in the table specify the input values that must be applied to MUX1 and MUX2. The multiplexer input for each present state is determined from the input conditions when the next state of the flip-flop is equal to 1. Thus, after present state 01, the next state of $G_1$ is always equal to 1 and the next state of $G_2$ is equal to the complement value of $x$. Therefore, the input of MUX1 is made equal to 1 and that of MUX2

**TABLE 8-4**
**Multiplexer Input Conditions**

| Present State | | Next State | | | Multiplexer Inputs | |
|---|---|---|---|---|---|---|
| $G_1$ | $G_2$ | $G_1$ | $G_2$ | Input Conditions | MUX1 | MUX2 |
| 0 | 0 | 0 | 0 | $w'$ | | |
| 0 | 0 | 0 | 1 | $w$ | 0 | $w$ |
| 0 | 1 | 1 | 0 | $x$ | | |
| 0 | 1 | 1 | 1 | $x'$ | 1 | $x'$ |
| 1 | 0 | 0 | 0 | $y'$ | | |
| 1 | 0 | 1 | 0 | $yz'$ | | |
| 1 | 0 | 1 | 1 | $yz$ | $yz' + yz = y$ | $yz$ |
| 1 | 1 | 0 | 1 | $y'z$ | | |
| 1 | 1 | 1 | 0 | $y$ | | |
| 1 | 1 | 1 | 1 | $y'z'$ | $y + y'z' = y + z'$ | $y'z + y'z' = y'$ |

to $x'$ when the present state of the register is 01. As another example, after present state 10, the next state of $G_1$ must be equal to 1 if the input conditions are $yz'$ or $yz$. When these two Boolean terms are ORed together and then simplified, we obtain the single binary variable $y$, as indicated in the table. The next state of $G_2$ is equal to 1 if the input conditions are $yz = 11$. If the next state of $G_1$ remains at 0 after a given present state, we place a 0 in the multiplexer input as shown in present state 00 for MUX1. If the next state of $G_1$ is always 1, we place a 1 in the multiplexer input as shown in present state 01 for MUX1. The other entries for MUX1 and MUX2 are derived in a similar manner. The multiplexer inputs from the table are then used in the control implementation of Fig. 8-16. Note that if the next state of a flip-flop is a function of two or more control variables, the multiplexer may require one or more gates in its input. Otherwise, the multiplexer input is equal to the control variable, or the complement of the control variable, or 0, or 1.

## Design Example

We will demonstrate the multiplexer control implementation by means of a second design example. The example will also demonstrate the formulation of the ASM chart and the implementation of the data-processor subsystem.

The digital system to be designed consists of two registers, $R1$ and $R2$, and a flip-flop, $E$. The system counts the number of 1's in the number loaded into register $R1$ and sets register $R2$ to that number. For example, if the binary number loaded into $R1$ is 10111001, the circuit counts the five 1's in $R1$ and sets register $R2$ to the binary count 101. This is done by shifting each bit from register $R1$ one at a time into flip-flop $E$. The value in $E$ is checked by the control, and each time it is equal to 1, register $R2$ is incremented by 1.

The control subsystem uses one external input $S$ to start the operation and two status inputs $E$ and $Z$ from the data processor. $E$ is the output of the flip-flop. $Z$ is the output of a circuit that checks the contents of register $R1$ for all 0's. The circuit produces an output $Z = 1$ when $R1$ is equal to 0.

The ASM chart for the design example is shown in Fig. 8-17. The binary number is loaded into $R1$, and register $R2$ is set to an all-1's value. Note that a number with all 1's in a register when incremented produces a number with all 0's. In state $T_1$, register $R2$ is incremented and the content of $R1$ is examined. If the content is zero, then $Z = 1$, and it signifies that there are no 1's stored in the register; so the operation terminates with $R2$ equal to 0. If the content of $R1$ is not zero, then $Z = 0$, and it indicates that

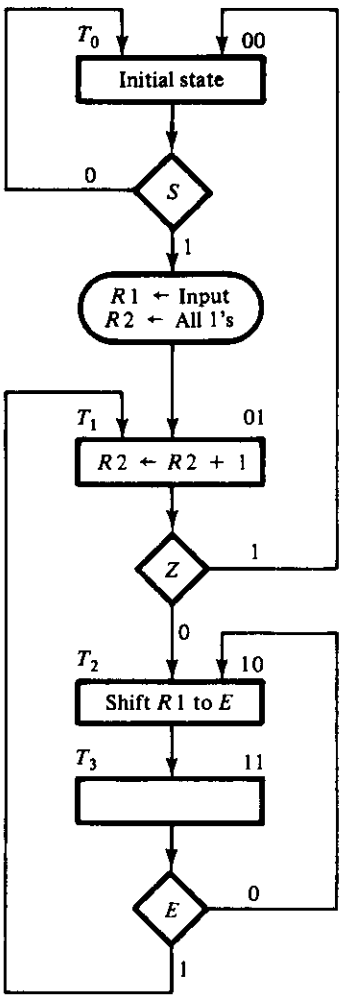



**FIGURE 8-17**
ASM chart for design example

there are some 1's stored in the register. The number in $R1$ is shifted and its leftmost bit transferred into $E$. This is done as many times as necessary until a 1 is transferred into $E$. For every 1 detected in $E$, register $R2$ is incremented and register $R1$ is checked again for more 1's. The major loop is repeated until all the 1's in $R1$ are counted. Note that the state box of $T_3$ has no register operations, but the block associated with it contains the decision box for $E$. Also note that the serial input to shift register $R1$ must be equal to 0 because we don't want to shift external 1's into $R1$.

The data-processor subsystem is shown in Fig. 8-18. The control has three inputs

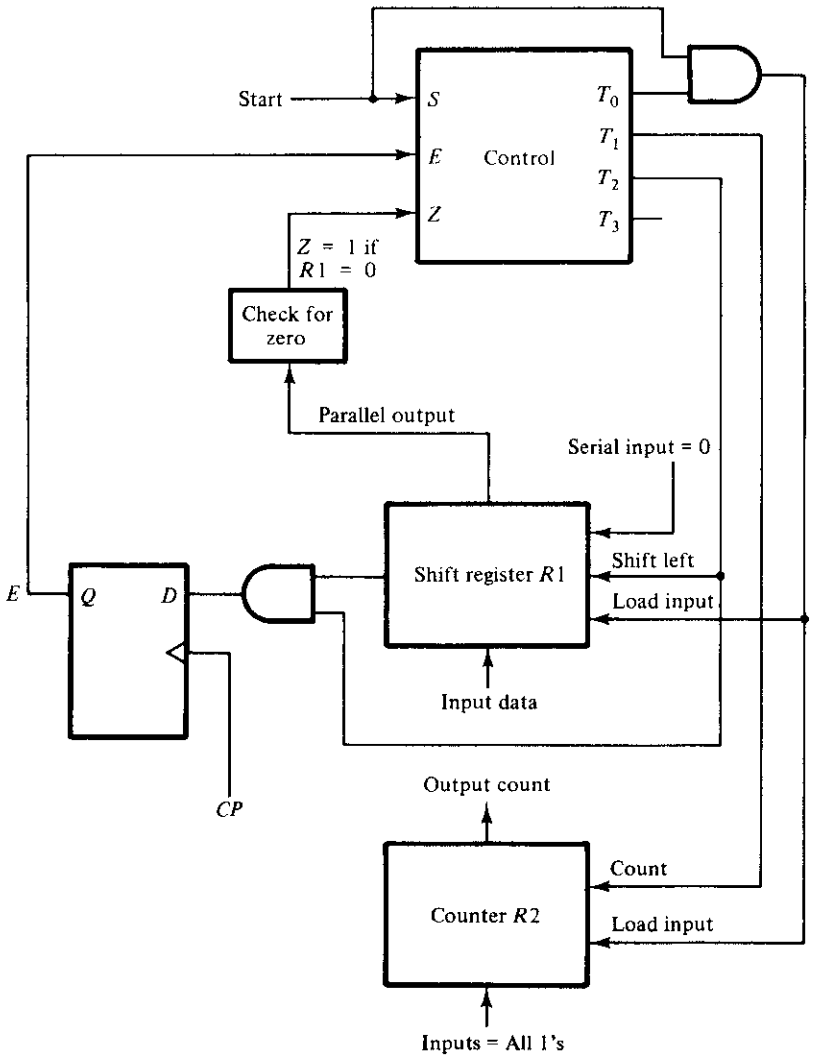



**FIGURE 8-18**
Data-processor subsystem for design example

**TABLE 8-5**
**Multiplexer Input Conditions for Design Example**

| Present State | | Next State | | Input Conditions | Multiplexer Inputs | |
|---|---|---|---|---|---|---|
| $G_1$ | $G_2$ | $G_1$ | $G_2$ | | MUX1 | MUX2 |
| 0 | 0 | 0 | 0 | $S'$ | | |
| 0 | 0 | 0 | 1 | $S$ | 0 | $S$ |
| 0 | 1 | 0 | 0 | $Z$ | | |
| 0 | 1 | 1 | 0 | $Z'$ | $Z'$ | 0 |
| 1 | 0 | 1 | 1 | None | 1 | 1 |
| 1 | 1 | 1 | 0 | $E'$ | | |
| 1 | 1 | 0 | 1 | $E$ | $E'$ | $E$ |

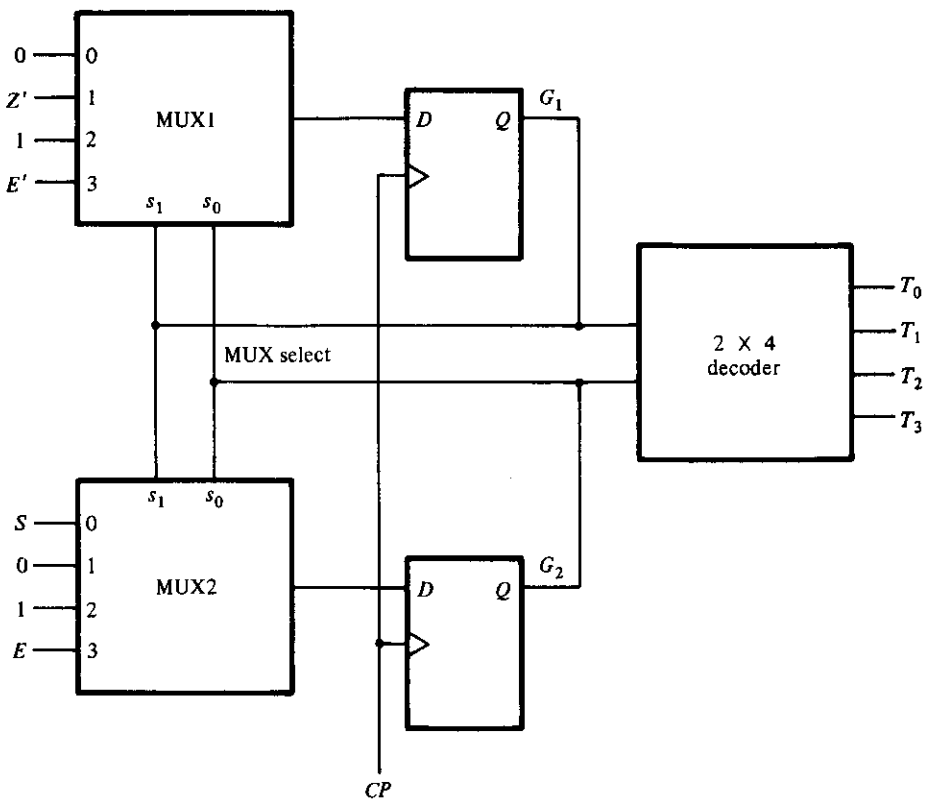



**FIGURE 8-19**
Control implementation of design example

and four outputs. Only three outputs are used by the data processor. Register $R1$ is a shift register similar to the one shown in Fig. 7-9. Register $R2$ is a counter with parallel load similar to the one shown in Fig. 7-19. In order not to complicate the diagram, the clock pulses are not shown, but they must be applied to the two registers, the $E$ flip-flop, and the flip-flops in the control. The circuit that checks for zero is a NOR gate. For example, if $R1$ is a four-bit register with outputs $R_1$, $R_2$, $R_3$, $R_4$, then $Z$ is generated with the Boolean function

$$Z = R_1' R_2' R_3' R_4' = (R_1 + R_2 + R_3 + R_4)'$$

which is the NOR function of all bits in the register.

The multiplexer input conditions for the control are determined from Table 8-5. The input conditions are obtained from the ASM chart for each possible binary state transition. The binary assignment to each state is written at the upper right corner of the state boxes. The transition from present state 00 depends on $S$, from present state 01 depends on $Z$, and from present state 11 on $E$. Present state 10 goes to next state 11 unconditionally. The values under MUX1 and MUX2 in the table are determined from the input Boolean conditions for the next state of $G_1$ and $G_2$, respectively.

The control implementation of the design example is shown in Fig. 8-19. This is a three-level implementation with the multiplexers in the first level. The inputs to the multiplexers are obtained from Table 8-5.

## 8-6 PLA CONTROL

We have seen from the examples presented in this chapter that the design of a control circuit is essentially a sequential-logic problem. In Section 7-2, we showed that a sequential circuit can be constructed by means of a register connected to a combinational circuit. In Section 5-8, we investigated the programmable logic array (PLA) and showed that it can be used to implement any combinational circuit. Since the control logic is a sequential circuit, it is then possible to design the control circuit with a register connected to a PLA. The design of a PLA control requires that we obtain the state table of the circuit. The PLA method should be used if the state table contains many don't-care entries; otherwise, it may be advantageous to use a ROM instead of a PLA.

The PLA control will be demonstrated by means of a third design example. This example is an arithmetic circuit that multiplies two unsigned binary numbers and produces their binary product.

### Binary Multiplier

The multiplication of two binary numbers is done with paper and pencil by successive additions and shifting. This process is best illustrated with a numerical example. Let us multiply the two binary numbers 10111 and 10011.

| | | |
|---|---|---|
| 23 | 10111 | multiplicand |
| 19 | 10011 | multiplier |
| | 10111 | |
| | 10111 | |
| | 00000 | |
| | 00000 | |
| | 10111 | |
| 437 | 110110101 | product |

The process consists of looking at successive bits of the multiplier, least significant bit first. If the multiplier bit is a 1, the multiplicand is copied down; otherwise, 0's are copied down. The numbers copied down in successive lines are shifted one position to the left from the previous number. Finally, the numbers are added and their sum forms the product. Note that the product obtained from the multiplication of two binary numbers of $n$ bits each can be up to $2n$ bits long.

When the above process is implemented with digital hardware, it is convenient to change the process slightly. First, instead of providing digital circuits to store and add simultaneously as many binary numbers as there are 1's in the multiplier, it is convenient to provide circuits for the summation of only two binary numbers and successively accumulate the partial products in a register. Second, instead of shifting the multiplicand to the left, the partial product is shifted to the right, which results in leaving the partial product and the multiplicand in the required relative positions. Third, when the corresponding bit of the multiplier is a 0, there is no need to add all 0's to the partial product, since this will not alter its value.

The data-processor subsystem for the binary multiplier is shown in Fig. 8-20. The

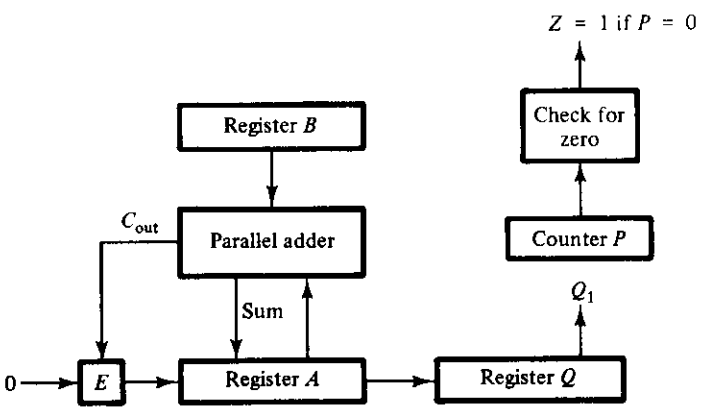



**FIGURE 8-20**
Data processor for binary multiplier

multiplicand is stored in register $B$, the multiplier is stored in register $Q$, and the partial product is formed in register $A$. A parallel adder similar to the circuit shown in Fig. 5-2 is used to add the contents of register $B$ to register $A$. The $E$ flip-flop stores the carry after the addition. The $P$ counter is initially set to hold a binary number equal to the number of bits in the multiplier. This counter is decremented after the formation of each partial product. When the content of the counter reaches zero, the product is formed in the double register $A$ and $Q$ and the process stops.

The control logic stays in an initial state until the start signal $S$ becomes a 1. The system then performs the multiplication. The sum of $A$ and $B$ forms a partial product, which is transferred to $A$. The output carry from the addition, whether 0 or 1, is transferred to $E$. Both the partial product in $A$ and the multiplier in $Q$ are shifted to the right. The least significant bit of $A$ is shifted into the most significant position of $Q$; the

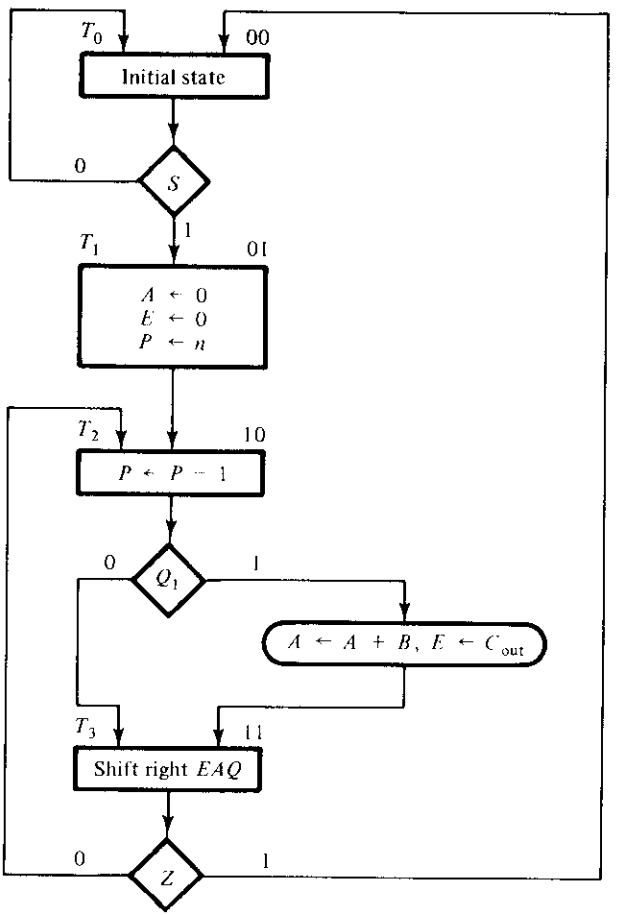



**FIGURE 8-21**
ASM chart for binary multiplier

carry from $E$ is shifted into the most significant position of $A$; and 0 is shifted into $E$. After the shift-right operation, one bit of the partial product is transferred into $Q$ while the multiplier bits in $Q$ are shifted one position to the right. In this manner, the rightmost bit of register $Q$, designated by $Q_1$, always holds the bit of the multiplier that must be inspected next.

### ASM Chart

The ASM chart for the binary multiplier is shown in Fig. 8-21. Initially, the multiplicand is in $B$ and the multiplier in $Q$. The multiplication process is initiated when $S = 1$. Register $A$ and flip-flop $E$ are cleared and the sequence counter $P$ is set to a binary number $n$, which is equal to the number of bits in the multiplier.

Next we enter a loop that keeps forming the partial products. The multiplier bit in $Q_1$ is checked, and if it is equal to 1, the multiplicand in $B$ is added to the partial product in $A$. The carry from the addition is transferred to $E$. The partial product in $A$ is left unchanged if $Q_1 = 0$. The $P$ counter is decremented by 1 regardless of the value of $Q_1$. Registers $E$, $A$, and $Q$ are combined into one composite register $EAQ$, which is then shifted once to the right to obtain a new partial product.

The value in the $P$ counter is checked after the formation of each partial product. If the content of $P$ is not zero, control input $Z$ is equal to 0 and the process is repeated to form a new partial product. The process stops when the $P$ counter reaches 0 and the control input $Z$ is equal to 1. Note that the partial product formed in $A$ is shifted into $Q$ one bit at a time and eventually replaces the multiplier. The final product is available in $A$ and $Q$, with $A$ holding the most significant bits and $Q$ the least significant bits.

The previous numerical example is repeated in Fig. 8-22 to clarify the multiplication process. The procedure follows the steps outlined in the ASM chart.

Multiplicand $B = 10111$

| | $E$ | $A$ | $Q$ | $P$ |
|---|---|---|---|---|
| Multiplier in $Q$ | 0 | 00000 | 10011 | 101 |
| $Q_1 = 1$; add $B$ | | 10111 | | |
| First partial product | 0 | 10111 | | 100 |
| Shift right $EAQ$ | 0 | 01011 | 11001 | |
| $Q_1 = 1$: add $B$ | | 10111 | | |
| Second partial product | 1 | 00010 | | 011 |
| Shift right $EAQ$ | 0 | 10001 | 01100 | |
| $Q_1 = 0$; shift right $EAQ$ | 0 | 01000 | 10110 | 010 |
| $Q_1 = 0$; shift right $EAQ$ | 0 | 00100 | 01011 | 001 |
| $Q_1 = 1$; add $B$ | | 10111 | | |
| Fifth partial product | 0 | 11011 | | 000 |
| Shift right $EAQ$ | 0 | 01101 | 10101 | |
| Final product in $AQ = 0110110101$ | | | | |

**FIGURE 8-22**
Example of binary multiplication

## PLA Control

The control for the binary multiplier has four states and three inputs. The binary state assignment is shown in the ASM chart over each state box. The three control inputs are $S$, $Q_1$, and $Z$. The design of a control unit with a PLA is similar to the design using $D$ flip-flops and a decoder. The only difference is in the way the combinational circuit part of the control is implemented. The PLA essentially replaces the decoder and all the gates in the inputs of the flip-flops.

The block diagram of the PLA control is shown in Fig. 8-23. The PLA is connected to a register with two flip-flops, $G_1$ and $G_2$. The inputs to the PLA are the values of the present state of the register and the three control inputs. The outputs of the PLA provide the values for the next state in the register and the control output variables. There is one output for each present state and an additional output for the conditional operation $D = Q_1T_2$. Since the PLA implements the control combinational circuit, we might as well include within it the gates for all conditional operations. In the binary multiplier, there is a conditional operation to add $B$ to $A$ during state $T_2$ provided that $Q_1 = 1$. PLA output $D$ will then activate this operation in the data processor.

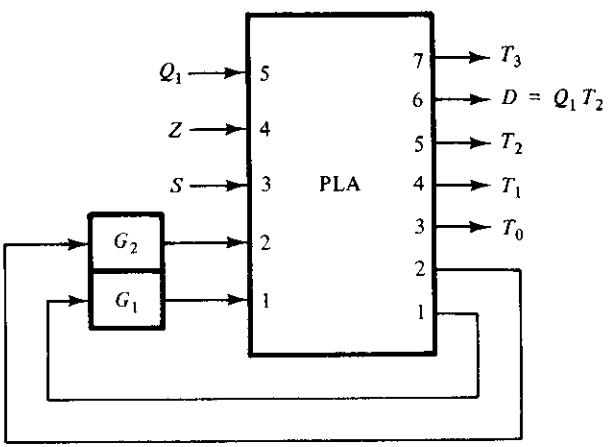



**FIGURE 8-23**
PLA control block diagram

At any given time, the present state of the register together with the input conditions determine the output values and the next state for the register. The next clock pulse initiates the register operations specified by the PLA outputs and transfers the next-state value into the register. This provides a new control state and possibly different input values. Thus, the PLA acts as the combinational-circuit part of the sequential circuit to generate the control outputs and the next-state values for the register.

### A Program Table

The internal organization of the PLA was presented in Section 5-8. It was also shown there how to obtain the PLA program table. The reader is advised to review this section to make sure that the meaning of a PLA program table is understood. The internal paths inside the PLA are constructed according to the specifications given in the program table. The design of a PLA control requires that we obtain the state table for the circuit. The state table gives essentially all the information required for obtaining the PLA program table.

The state table for the control subsystem of the binary multiplier is shown in Table 8-6. The present state is determined from flip-flops $G_1$ and $G_2$. The input variables for the control are $S$, $Z$, and $Q_1$. The next state of $G_1$ and $G_2$ may be a function of one of the input control variables or it may be independent of any inputs. If an input variable does not influence the next state, we mark it with a don't-care condition $X$. If there are two different transitions from the same present state, the present state is repeated in the table, but the next states are assigned different binary values. The table also lists all control outputs as a function of the present state. Note that input $Q_1$ does not affect the next state, but only determines the value of output $D$ during state $T_2$.

The PLA program table can be obtained directly from the state table without the need for simplification procedures. The PLA program table listed in Table 8-7 specifies seven product terms, one for each row in the state table. The input and output terminals are marked with numbers, and the variables applied to these numbered terminals are indicated in the block diagram of Fig. 8-23. The comments are not part of the table, but are included for clarification.

According to the rules established in Section 5-8, a no connection for a PLA path is indicated by a dash (–) in the table. The $X$'s in the state table designate don't-care conditions and imply no connection for the PLA. The 0's in the output columns also indicate no connections to the OR gates within the PLA. The translation from the state table to a PLA program table is very simple. The $X$'s in the "input" columns and the 0's in the "next-state" and "output" columns are changed to dashes, and all other en-

**TABLE 8-6**
**State Table for Control Circuit**

| Present State | | Inputs | | | Next State | | Outputs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $G_1$ | $G_2$ | $S$ | $Z$ | $Q_1$ | $G_1$ | $G_2$ | $T_0$ | $T_1$ | $T_2$ | $D$ | $T_3$ |
| 0 | 0 | 0 | $X$ | $X$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | $X$ | $X$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | $X$ | $X$ | $X$ | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | $X$ | $X$ | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | $X$ | $X$ | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | $X$ | 0 | $X$ | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | $X$ | 1 | $X$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**TABLE 8-7**
**PLA Program Table**

| Product Term | Inputs 1 | 2 | 3 | 4 | 5 | Outputs 1 | 2 | 3 | 4 | 5 | 6 | 7 | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | – | – | – | – | 1 | – | – | – | – | $T_0 = 1$, $S = 0$ |
| 2 | 0 | 0 | 1 | – | – | – | 1 | 1 | – | – | – | – | $T_0 = 1$, $S = 1$ |
| 3 | 0 | 1 | – | – | – | 1 | – | – | 1 | – | – | – | $T_1 = 1$ |
| 4 | 1 | 0 | – | – | 0 | 1 | 1 | – | – | 1 | – | – | $T_2 = 1$, $Q_1 = 0$ |
| 5 | 1 | 0 | – | – | 1 | 1 | 1 | – | – | 1 | 1 | – | $T_2 = 1$, $D = 1$ |
| 6 | 1 | 1 | – | 0 | – | 1 | – | – | – | – | – | 1 | $T_3 = 1$, $Z = 0$ |
| 7 | 1 | 1 | – | 1 | – | – | – | – | – | – | – | 1 | $T_3 = 1$, $Z = 1$ |

tries remain the same. The inputs to the PLA are the same as the present state and inputs in the state table. The outputs of the PLA are the same as the next state and outputs in the state table.

The preceding example demonstrates the procedure for designing the control logic with a PLA. From the specifications of the system, we first obtain a state table for the controller. The number of states determines the number of flip-flops for the register. The PLA is then connected to the register and to the input and output variables. The PLA program table is obtained directly from the state table.

The examples introduced in this chapter demonstrate five methods of control-logic design. These should not be considered the only possible methods. A resourceful designer may be able to formulate a control configuration to suit a particular application. This configuration may consist of a combination of methods or may constitute a control organization other than the ones presented here.

## REFERENCES


1. CLARE, C. R., *Designing Logic Systems Using State Machines*. New York: McGraw-Hill, 1971.
2. WINKEL, D., and F. PROSSER, *The Art of Digital Design*, 2nd Ed. Englewood Cliffs, NJ: Prentice-Hall, 1987.
3. PEATMAN, J. B., *Digital Hardware Design*, New York: McGraw-Hill, 1980.
4. WIATROWSKI, C. A., and C. H. HOUSE, *Logic Circuits and Microcomputer Systems*. New York: McGraw-Hill, 1980.
5. ROTH, C. H., *Fundamentals of Logic Design*, 3rd Ed. New York: West, 1985.
6. MANO, M. M., *Computer System Architecture*, 2nd Ed. Englewood Cliffs, NJ: Prentice-Hall, 1982.
7. SHIVA, S. G., *Introduction to Logic Design*. Glenview, IL: Scott, Foresman, 1988.
8. FLETCHER, W. I., *An Engineering Approach to Digital Design*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

## PROBLEMS

**8-1** Draw the portion of an ASM chart that specifies a conditional operation to increment register $R$ during state $T_1$ and transfer to state $T_2$ if control inputs $z$ and $y$ are equal to 1 and 0, respectively.

**8-2** Show the eight exit paths in an ASM block emanating from the decision boxes that check the eight possible binary values of three control variables, $x$, $y$, and $z$.

**8-3** Obtain the ASM charts for the following state transitions:
(a) If $x = 0$, control goes from state $T_1$ to state $T_2$; if $x = 1$, generate a conditional operation and go from $T_1$ to $T_2$.
(b) If $x = 1$, control goes from $T_1$ to $T_2$ and then to $T_3$; if $x = 0$, control goes from $T_1$ to $T_3$.
(c) Start from state $T_1$; then: if $xy = 00$, go to $T_2$; if $xy = 01$, go to $T_3$; if $xy = 10$, go to $T_1$; otherwise, go to $T_3$.

**8-4** Construct an ASM chart for a digital system that counts the number of people in a room. People enter the room from one door with a photocell that changes a signal $x$ from 1 to 0 when the light is interrupted. They leave the room from a second door with a similar photocell with a signal $y$. Both $x$ and $y$ are synchronized with the clock, but they may stay on or off for more than one clock-pulse period. The data-processor subsystem consists of an up–down counter with a display of its contents.

**8-5** Explain how the ASM chart differs from a conventional flow chart. Using Fig. 8-5 as an illustration, show the difference in interpretation.

**8-6** Design the 4-bit counter with synchronous clear specified in Fig. 8-10.

**8-7** Using five-variable maps, derive the input Boolean functions to the *JK* flip-flops of Fig. 8-11.

**8-8** Design the control whose state table is given in Table 8-3 using two multiplexers, a register, and a decoder.

**8-9** Design the control of the design example of Section 8-5 by the method of one flip-flop per state.

**8-10** The state diagram of a control unit is shown in Fig. P8-10. It has four states and two inputs, $x$ and $y$.
(a) Draw the equivalent ASM chart, leaving the state boxes empty.
(b) Design the control with multiplexers.

**8-11** Assume that $R1$ in Fig. 8-18 is the 4-bit shift register shown in Fig. 7-9. Show how the shift and load inputs in Fig. 8-18 are to be connected to the $s_1$ and $s_0$ inputs of the shift register.

**8-12** Design a digital system with three 4-bit registers, $A$, $B$, and $C$, to perform the following operations:
1. Transfer two binary numbers to $A$ and $B$ when a start signal is enabled.
2. If $A < B$, shift left the contents of $A$ and transfer the result to register $C$.
3. If $A > B$, shift right the contents of $B$ and transfer the result to register $C$.
4. If $A = B$, transfer the number to register $C$ unchanged.

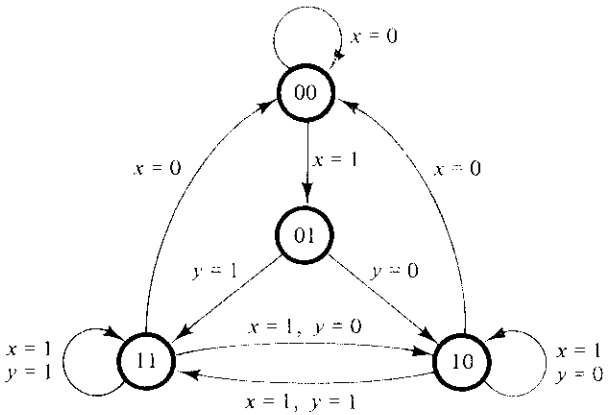



**FIGURE P8-10**
Control state diagram for Problem 8-10

**8-13** Design a digital system that multiplies two binary numbers by the repeated addition method. For example, to multiply $5 \times 4$, the digital system evaluates the product by adding the multiplicand four times: $5 + 5 + 5 + 5 = 20$. Let the multiplicand be in register *BR*, the multiplier in register *AR*, and the product in register *PR*. An adder circuit adds the contents of *BR* to *PR*. A zero-detection circuit *Z* checks when *AR* becomes 0 after each time that it is decremented.

**8-14** Prove that the multiplication of two $n$-bit numbers gives a product of length less than or equal to $2n$ bits.

**8-15** In Fig. 8-20, the *Q* register holds the multiplier and the *B* register holds the multiplicand. Assume that each number consists of 15 bits.
(a) How many bits can be expected in the product, and where is it available?
(b) How many bits are in the *P* counter, and what is the binary number loaded into it initially?
(c) Design the circuit that checks for zero in the *P* counter.

**8-16** List the contents of registers *E*, *A*, *Q*, and *P* similar to Fig. 8-22 during the process of multiplying the two numbers 11111 (multiplicand) and 10101 (multiplier).

**8-17** Determine the time it takes to process the multiplication operation in the binary multiplier described in Section 8-6. Assume that the *Q* register has $n$ bits and the clock period is $t$ nanoseconds.

**8-18** Design the control circuit of the binary multiplier specified by the ASM chart of Fig. 8-21 using each of the following methods:
(a) *JK* flip-flops and gates.
(b) *D* flip-flops and a decoder.
(c) Input multiplexers and a register.
(d) One flip-flop per state.

**8-19** Design the control whose state table is given in Table 8-3 using the PLA method.

**8-20** Consider the ASM chart of Fig. P8-20. The register operations are not specified, because we are interested only in designing the control logic.
(a) Draw the equivalent state diagram.
(b) Design the control with one flip-flop per state.

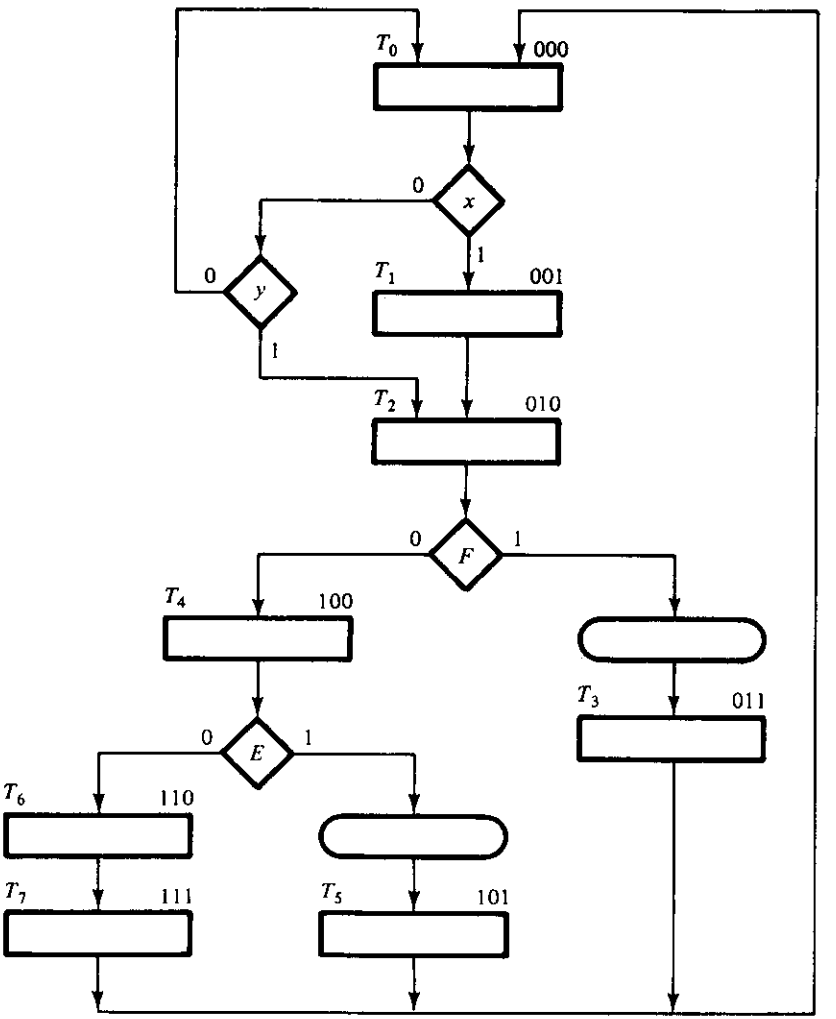



**FIGURE P8-20**
ASM chart for Problems 8-20 through 8-22

**8-21** (a) Derive the state table for the ASM chart of Fig. P8-20.
(b) Design the control with three $D$ flip-flops, a decoder, and gates.
(c) Design the control with a register and a PLA. List the PLA program table.

**8-22** (a) Derive a table showing the multiplexer input conditions for the control specified in the ASM chart of Fig. P8-20.
(b) Design the control with three multiplexers, a register with three flip-flops, and a $3 \times 8$ decoder.