Chapter 8

Design at the Register Transfer Level

- The behavior of many digital systems depends on the history of their inputs, and the conditions that determine their future actions depend on the results of previous actions. Such systems are said to have “memory.”

- A digital system is a sequential logic system constructed with flip-flops and gates.

- Sequential circuits can be specified by means of state tables as shown in Chapter 5 .

- To specify a large digital system with a state table is very difficult, because the number of states would be enormous.

- To overcome this difficulty, digital systems are designed via a **modular approach**.

- The system is partitioned into subsystems, each of which performs some function.

- The modules are constructed from digital devices such as registers, decoders, multiplexers, arithmetic elements, and control logic.

- The various modules are interconnected with datapaths and control signals to form a digital system.

- In this chapter, we will introduce a design methodology for describing and designing large, complex digital systems.

8.2 REGISTER TRANSFER LEVEL NOTATION

- The modules of a digital system are best defined by a set of registers and the operations that are performed on the binary information stored in them. Examples of register operations are *shift*, *count*, *clear*, and *load* .

- Registers are assumed to be the basic components of the digital system.

- A register is a connected group of flip-flops that stores binary information and has the capability of performing one or more elementary operations.

- The information flow and processing performed on the data stored in the registers are referred to as ***register transfer operations.***

A digital system is represented at the *register transfer level* (RTL) when it is specified by the following three components:

1. The set of registers in the system.
2. The operations that are performed on the data stored in the registers.
3. The control that supervises the sequence of operations in the system.

Register Transfer Level Notations

Information transfer from one register to another is designated in symbolic form by means of a replacement operator.

$$R2 \leftarrow R1$$

Normally, we want a register transfer operation to occur, not with every clock cycle, but only under a predetermined condition.

A conditional statement governing a register transfer operation is symbolized with an if–then statement such as

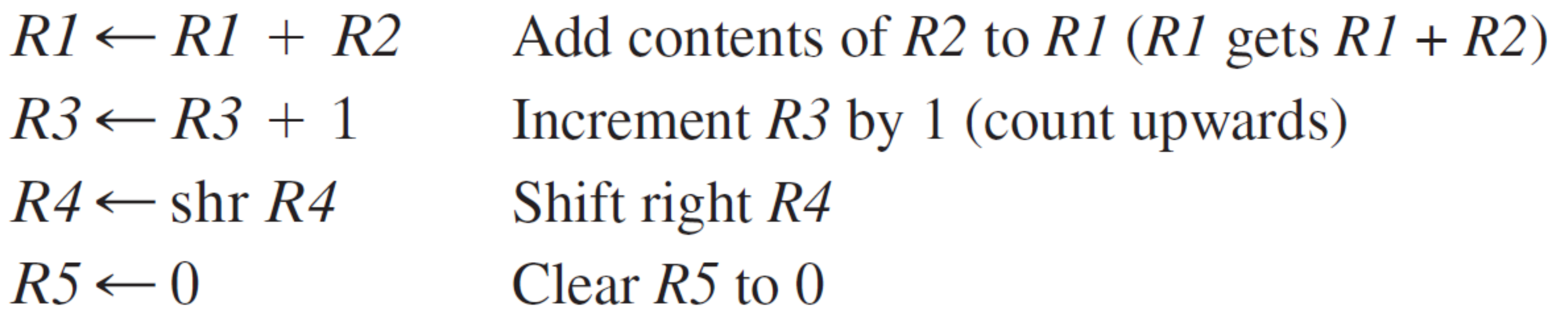
$$\text{If } (T1 \; = \; 1) \text{ then } (R2 \leftarrow R1)$$

Register Transfer Level Notations

A comma may be used to separate two or more operations that are executed at the same time (concurrently). Consider the statement

$$\text{If } (T3 = 1) \text{ then } (R2 \leftarrow R1,\ R1 \leftarrow R2)$$

Other examples of register transfers are



$R1 \leftarrow R1 + R2$	Add contents of $R2$ to $R1$ ($R1$ gets $R1 + R2$)
$R3 \leftarrow R3 + 1$	Increment $R3$ by 1 (count upwards)
$R4 \leftarrow \text{shr } R4$	Shift right $R4$
$R5 \leftarrow 0$	Clear $R5$ to 0

Register Transfer Level

In hardware,

- **addition is done with a binary parallel adder,**
- **incrementing is done with a counter, and**
- **shift operation is implemented with a shift register.**

The type of operations most often encountered in digital systems can be classified into four categories:

1. Transfer operations, which transfer (i.e., copy) data from one register to another.

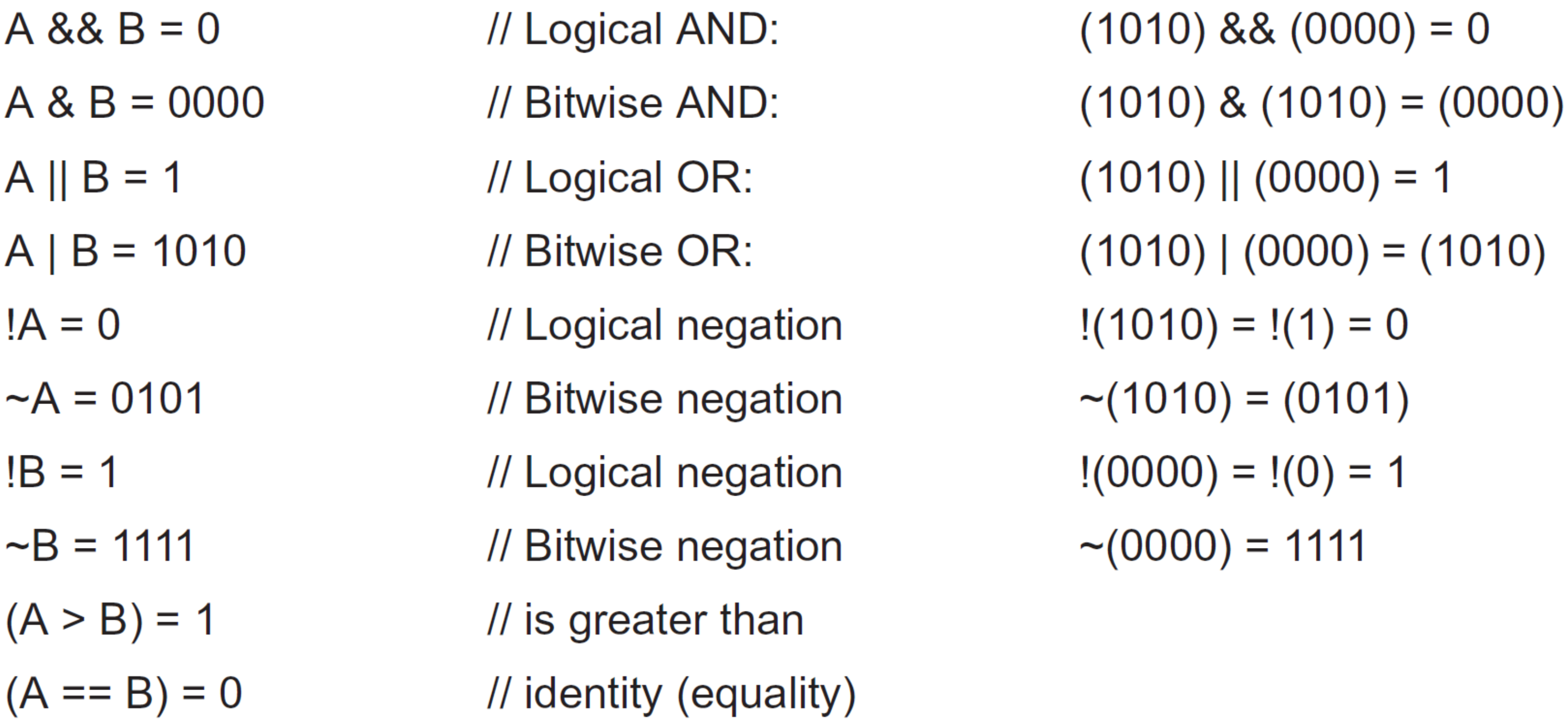
2. Arithmetic operations, which perform arithmetic (e.g., multiplication) on data in registers.

3. Logic operations, which perform bit manipulation (e.g., logical OR) of non-numeric data in registers.

4. Shift operations, which shift data between registers.

Register Transfer Level Notations

A = 1010 and *B* = 0000,



A && B = 0	// Logical AND:	(1010) && (0000) = 0
A & B = 0000	// Bitwise AND:	(1010) & (1010) = (0000)
A || B = 1	// Logical OR:	(1010) || (0000) = 1
A | B = 1010	// Bitwise OR:	(1010) | (0000) = (1010)
!A = 0	// Logical negation	!(1010) = !(1) = 0
~A = 0101	// Bitwise negation	~(1010) = (0101)
!B = 1	// Logical negation	!(0000) = !(0) = 1
~B = 1111	// Bitwise negation	~(0000) = 1111
(A > B) = 1	// is greater than	
(A == B) = 0	// identity (equality)	

8.3 REGISTER TRANSFER LEVEL IN HDL

Self Study

8.4 ALGORITHMIC STATE MACHINES (ASMs)

Binary information in Digital Systems:

Data
Control

Data processing tasks
Addition, decoding, counting etc

Control information provides command signals for monitoring data processing tasks.

Various modules are interconnected to form digital system

Logic Design

Design of Data processing circuits

Design of control circuits

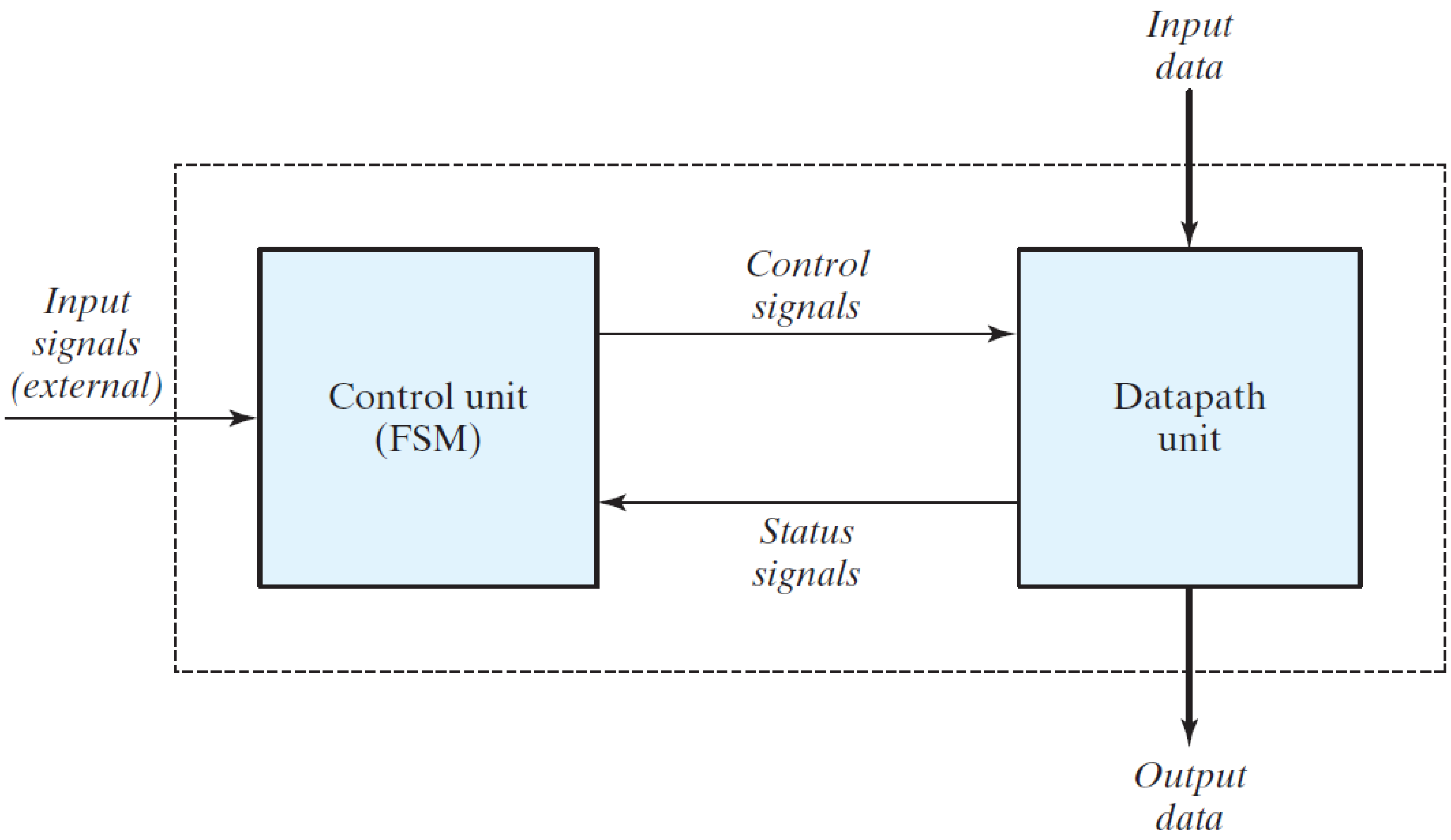


FIGURE 8.2
Control and datapath interaction

Control sequence and data processing tasks are specified by means of Hardware Algorithm.

A Special Flowchart developed to design Digital hardware algorithms is called Algorithmic State Machine (ASM) chart

A conventional flowchart describes sequence of procedural steps without concern for their time relationship

An ASM chart describes the sequence of events as well as the timing relationship between states of sequential controller

ASM CHART
Composed of three basic elements

The State box

The decision box

The conditional box

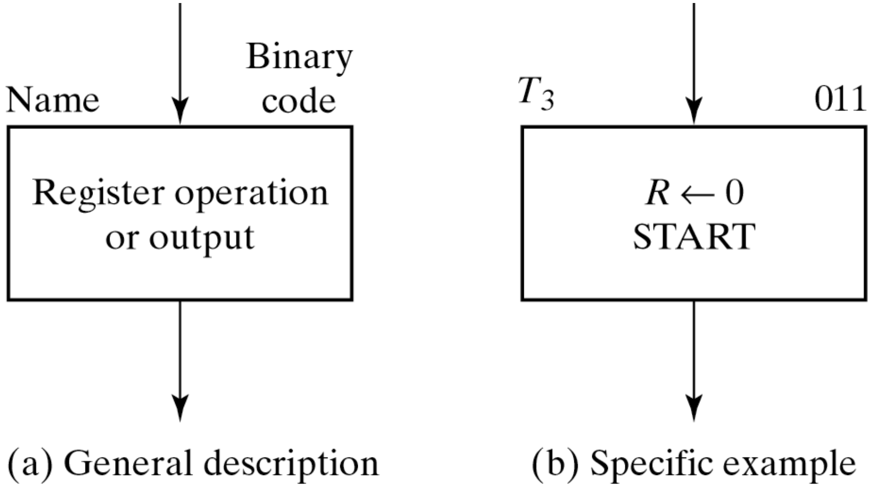
A state is indicated by the state box within which

register operations are written

Or

the output signal names that the control generates while being in the state

The state is given a symbolic name, binary code assigned to state



(a) General description (b) Specific example

Fig. 8-3 State Box

Start name may indicate an output signal that Starts an operation

- ✓ **Decision box indicates the effect of an input on the control subsystem**

- ✓ **Input condition to be tested is written inside the box**

- ✓ **One exit path is taken if the condition is true and another if false**

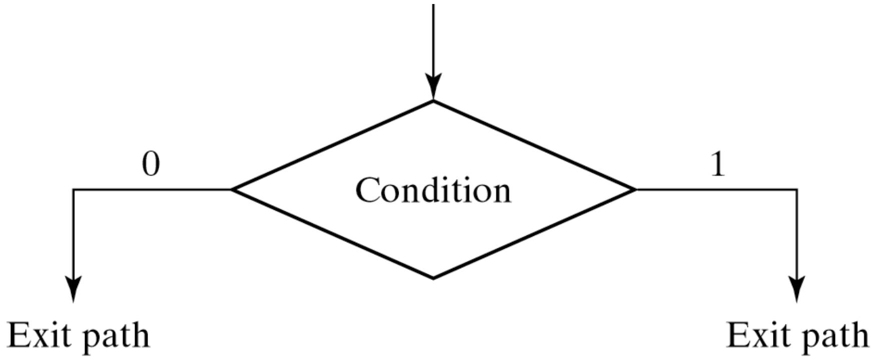
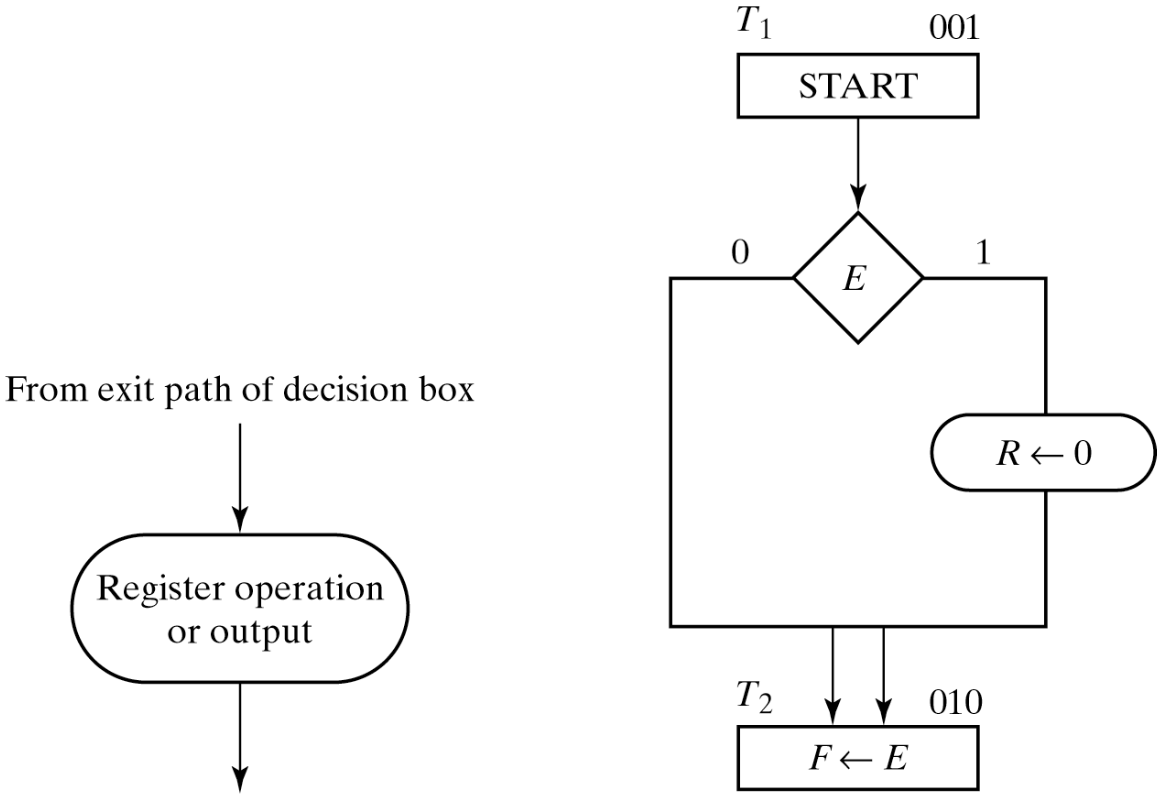


Fig. 8-4 Decision Box

M. Morris Mano
DIGITAL DESIGN, 3e.

- ✓ **The conditional box in unique to ASM chart**
- ✓ **Input path to conditional box comes from one of exit paths of a decision box**
- ✓ **The register operations or outputs listed inside a conditional box are generated during a given state provided the input condition is satisfied.**



(a) General description

(b) Example with conditional box

Fig. 8-5 Conditional Box

ASM Block

- 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

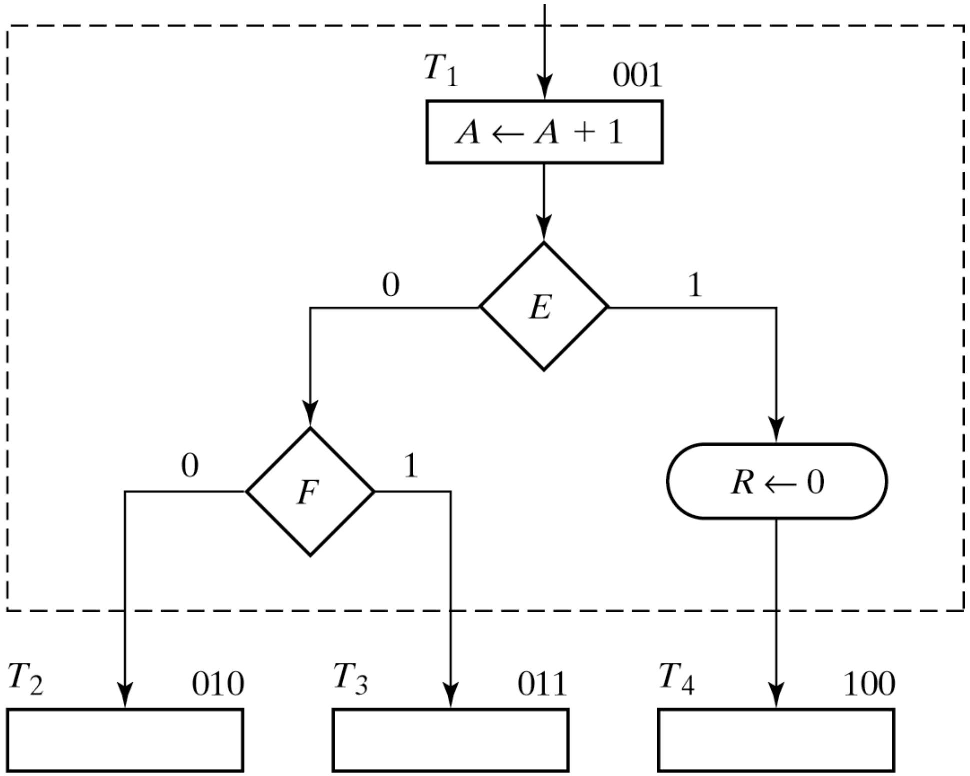


Fig. 8-6 ASM Block

M. Morris Mano
DIGITAL DESIGN, 3e.

- ✓ **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 are executed with a common clock pulse while the system is in state T1.**

- ✓ **The same clock pulse also transfers the system controller to one of the next states.**

- ✓ **ASM chart is very similar to state diagram.**

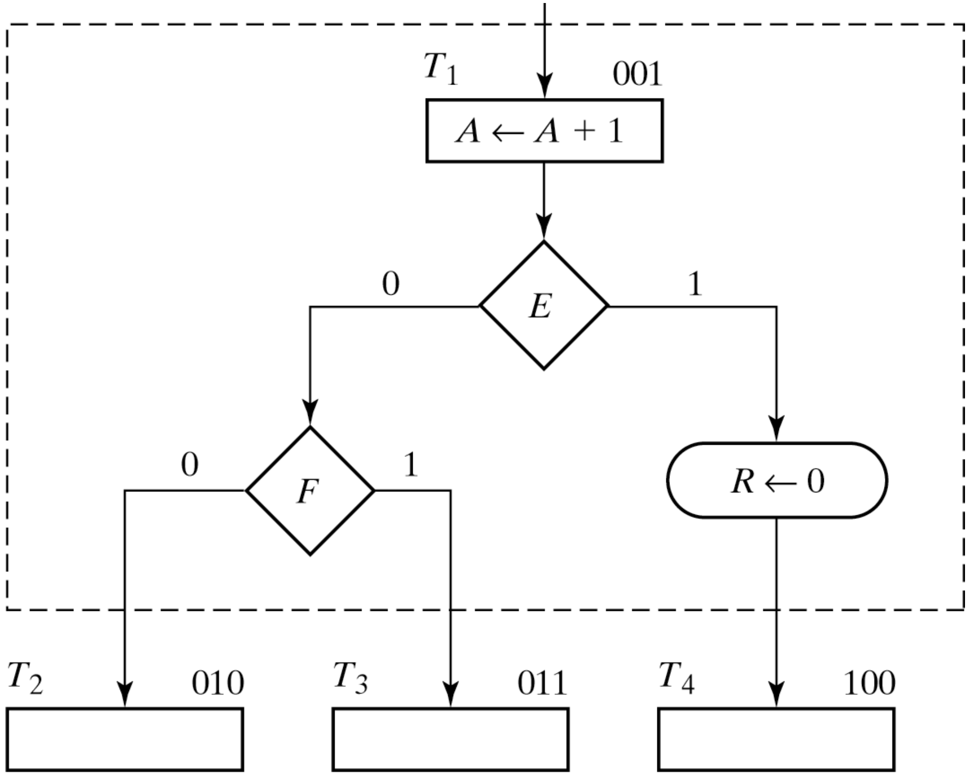


Fig. 8-6 ASM Block

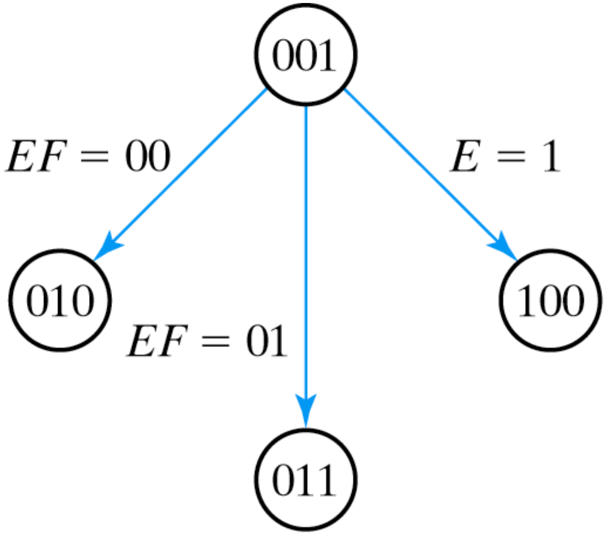


Fig. 8-7 State Diagram Equivalent to the ASM Chart of Fig. 8-6

M. Morris Mano
DIGITAL DESIGN, 3e.

- ✓Major difference between a conventional flowchart and an ASM chart is interpreting the timing relations.

- ✓Conventional flowchart the listed operations follow one after the other in time sequence.

- ✓ASM chart consider the entire block as one unit.

- ✓All operations within a block must occur in synchronism during clock edge.

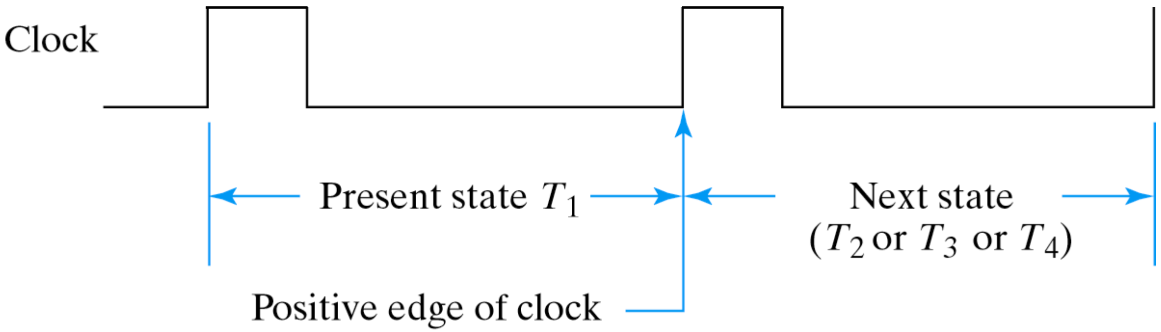


Fig. 8-8 Transition Between States

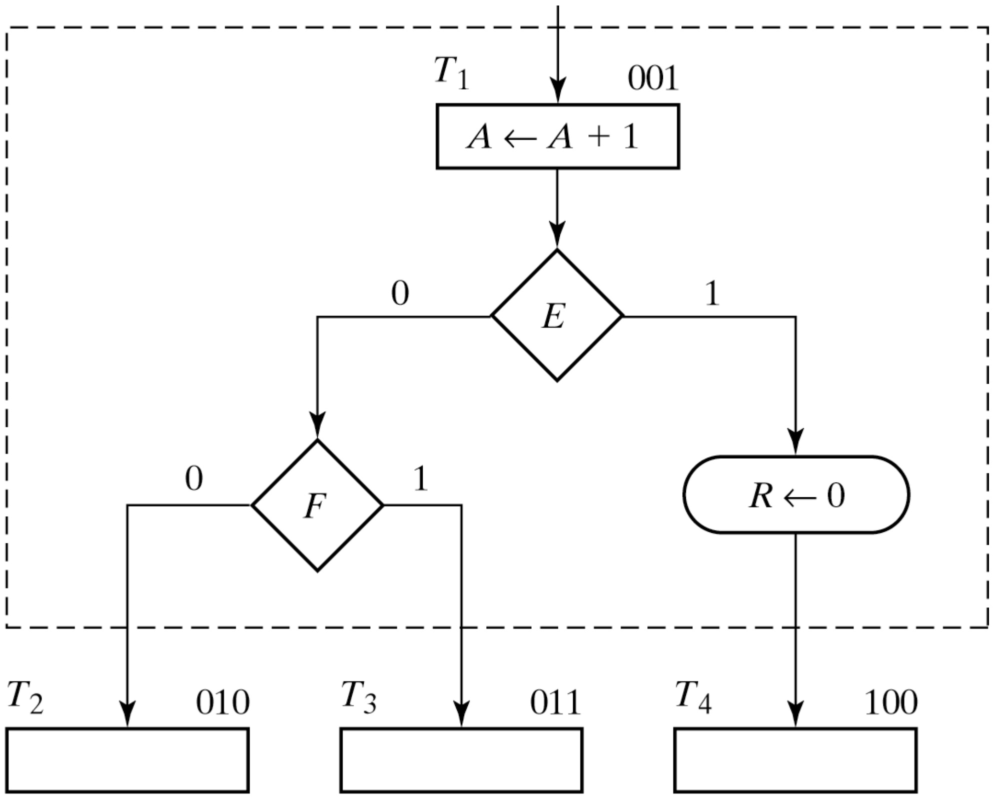


Fig. 8-6 ASM Block

During transition
A is incremented, If E = 1 , R is cleared
Depending on values of EF control transferred to T2 or T3 or T4

Design Example

Design a Digital system with two FFs E and F, one 4-bit binary counter A (A4A3A2A1). A start signal S initiates the system operation by clearing counter A and flip-flop F. Counter incremented by one starting from next clock pulse, continues to increment until operations stop.

If A3 = 0, E is cleared to 0 and count continues

If A3 = 1, E set to 1; then if A4=0, count continues, but if A4 = 1, F set to 1 on next clock pulse and system stops counting. If S = 0 system remains in initial state, but if S = 1 operation cycle repeats.

Design a Digital system with two FFs E and F, one 4-bit binary counter A (A4A3A2A1). A start signal S initiates the system operation by clearing counter A and flip-flop F.

Counter incremented by one starting from next clock pulse, continues to increment until operations stop.

If A3 = 0, E is cleared to 0 and count continues

If A3 = 1, E set to 1; then if A4 = 0, count continues,

but if A4 = 1, F set to 1 on next clock pulse and system stops counting. If S = 0 system remains in initial state, but if S = 1 operation cycle repeats.

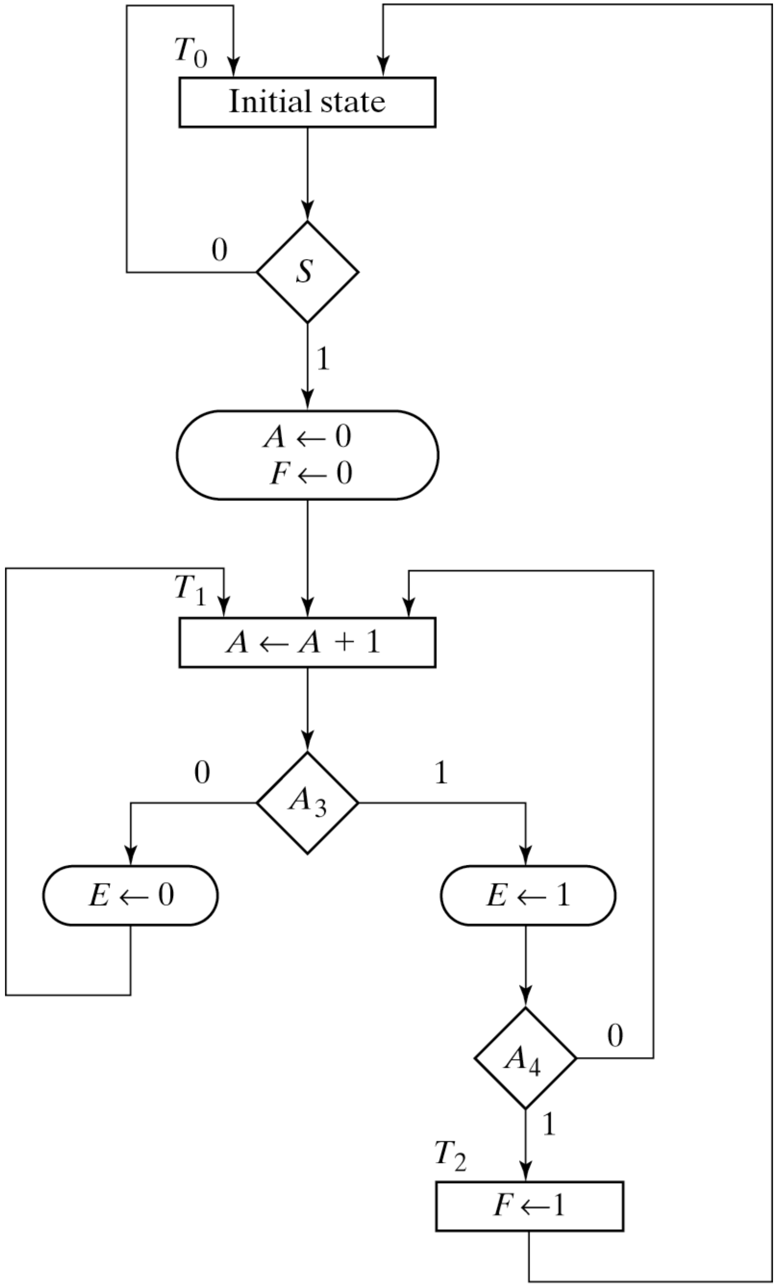


Fig. 8-9 ASM Chart for Design Example

When no operations the system in initial state T0 waiting for S

When input S = 1, counter A and flip-flop F cleared
Counter goes to state T1

Block with T1 has two decision and two conditional boxes

Counter incremented with every clock pulse, at the same time
One of three operations occur during clock transition

Either E cleared and control stays in T1 (A3 = 0); or
E is set and control stays in T1 (A3A4 = 10); or
E is set and control goes to T2 (A3A4 = 11).

When in T2 F is set and circuit goes to T0.

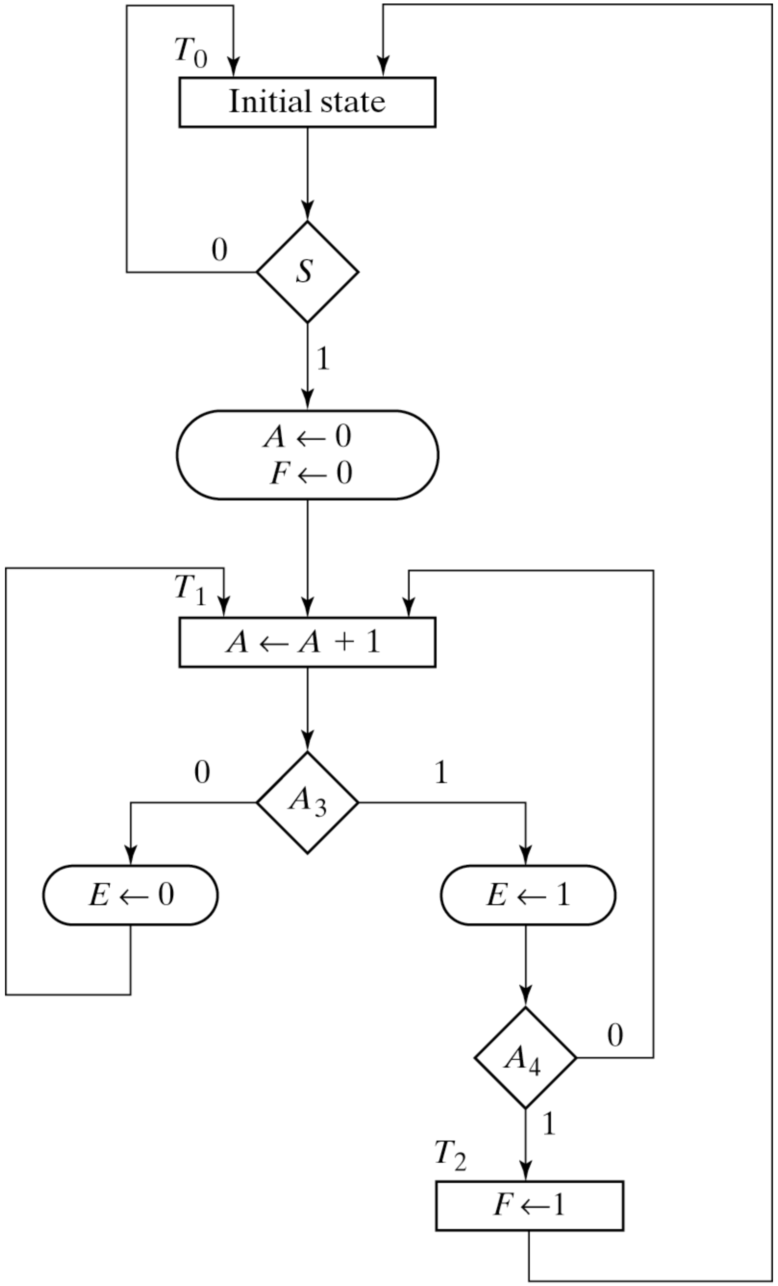


Fig. 8-9 ASM Chart for Design Example

- ✓**Every block in the ASM chart specifies the operations performed during one common clock pulse.**

- ✓**Operations specified within the state and conditional boxes in the block are performed in the datapath section.**

- ✓**The change from one state to next is performed in the control section.**

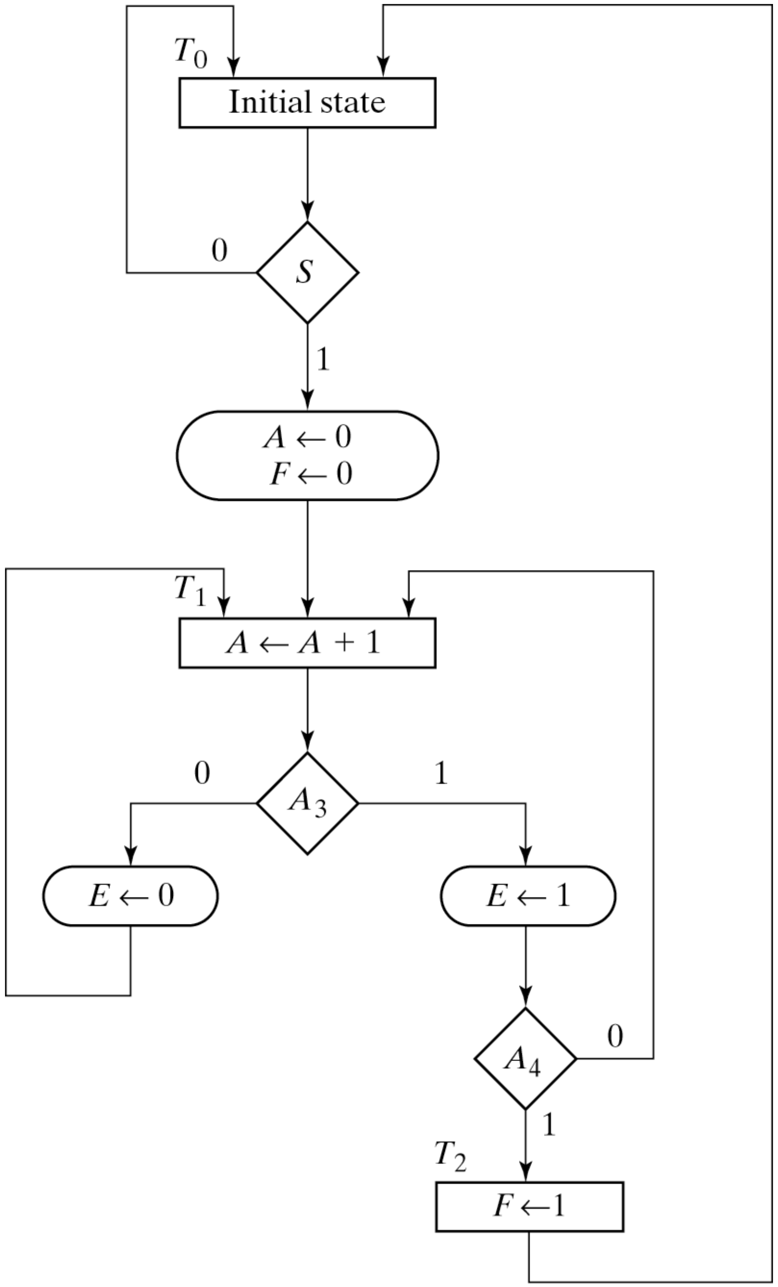


Fig. 8-9 ASM Chart for Design Example

Sequence of Operations for Design Example

Counter				Flip-Flops			
A4	A3	A2	A1	*E*	*F*	Conditions	State
0	0	0	0	1	0	A3 = 0, A4 = 0	T1
0	0	0	1	0	0		
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	A3 = 1, A4 = 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	A3 = 0, A4 = 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	A3 = 1, A4 = 1	
1	1	0	1	1	0		T2
1	1	0	1	1	1		T0

- ✓ **The requirements of design of the datapath are specified inside the state and conditional boxes**

- ✓ **Control logic is determined by the decision boxes and required state transitions.**

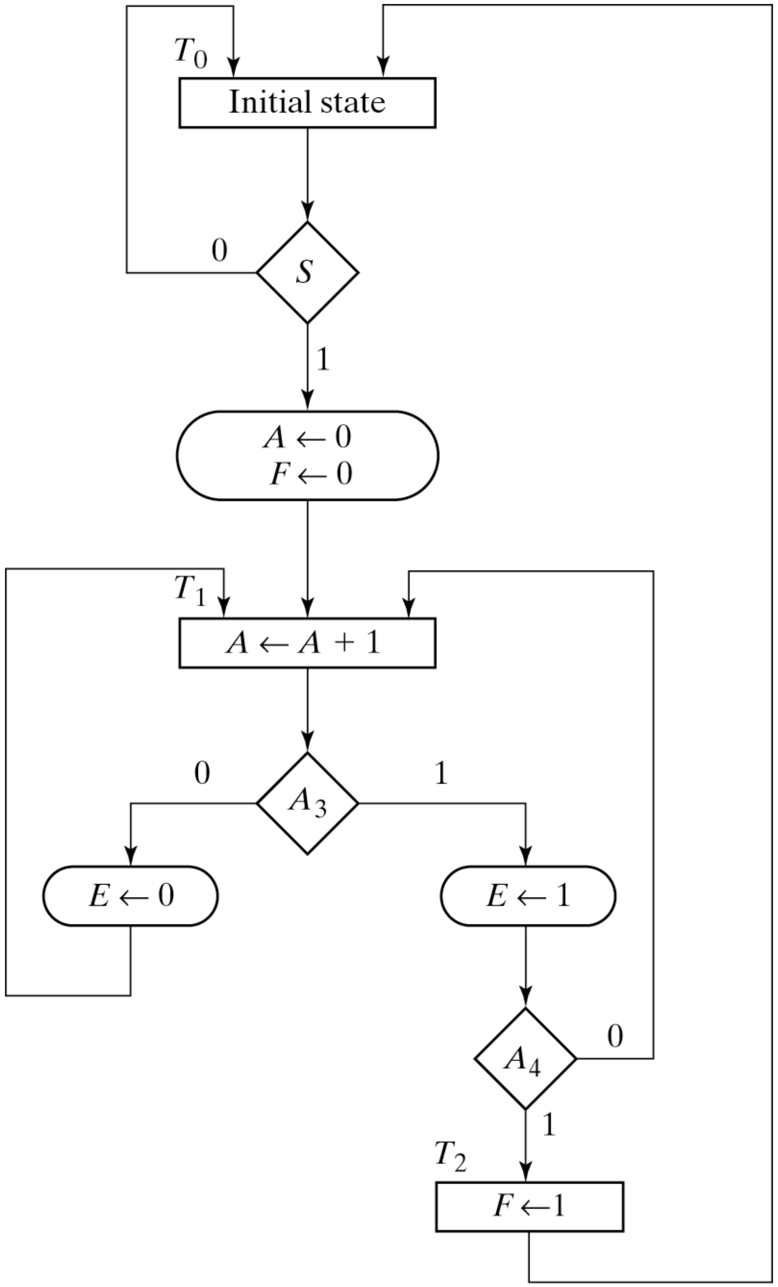


Fig. 8-9 ASM Chart for Design Example

Data path consists of
-4 bit binary counter
-Two flip-flops
-Gates

Counter is incremented with every clock cycle when control is in state T1.
Cleared when control is in state T0 and S is equal to 1. - uses an AND gate

The other two conditional operations use two other AND gates for setting or clearing E.
F is set unconditionally during T2.

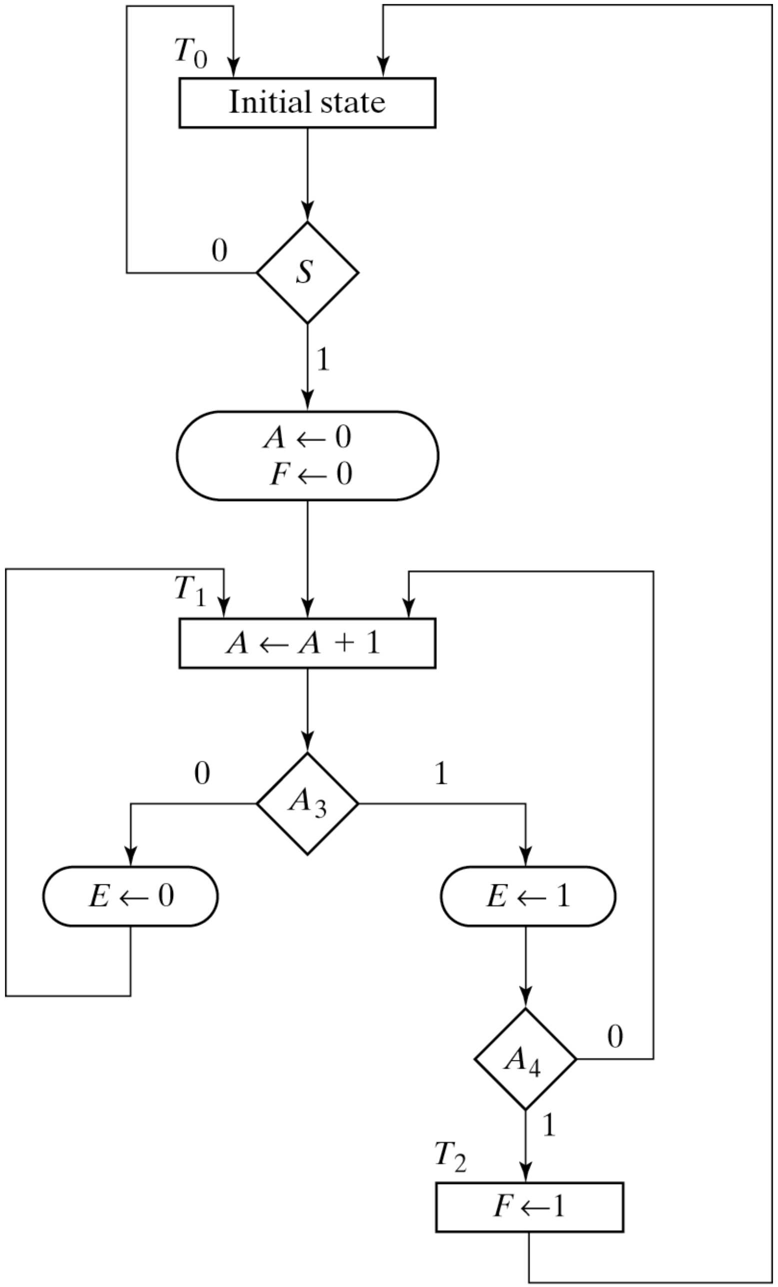


Fig. 8-9 ASM Chart for Design Example

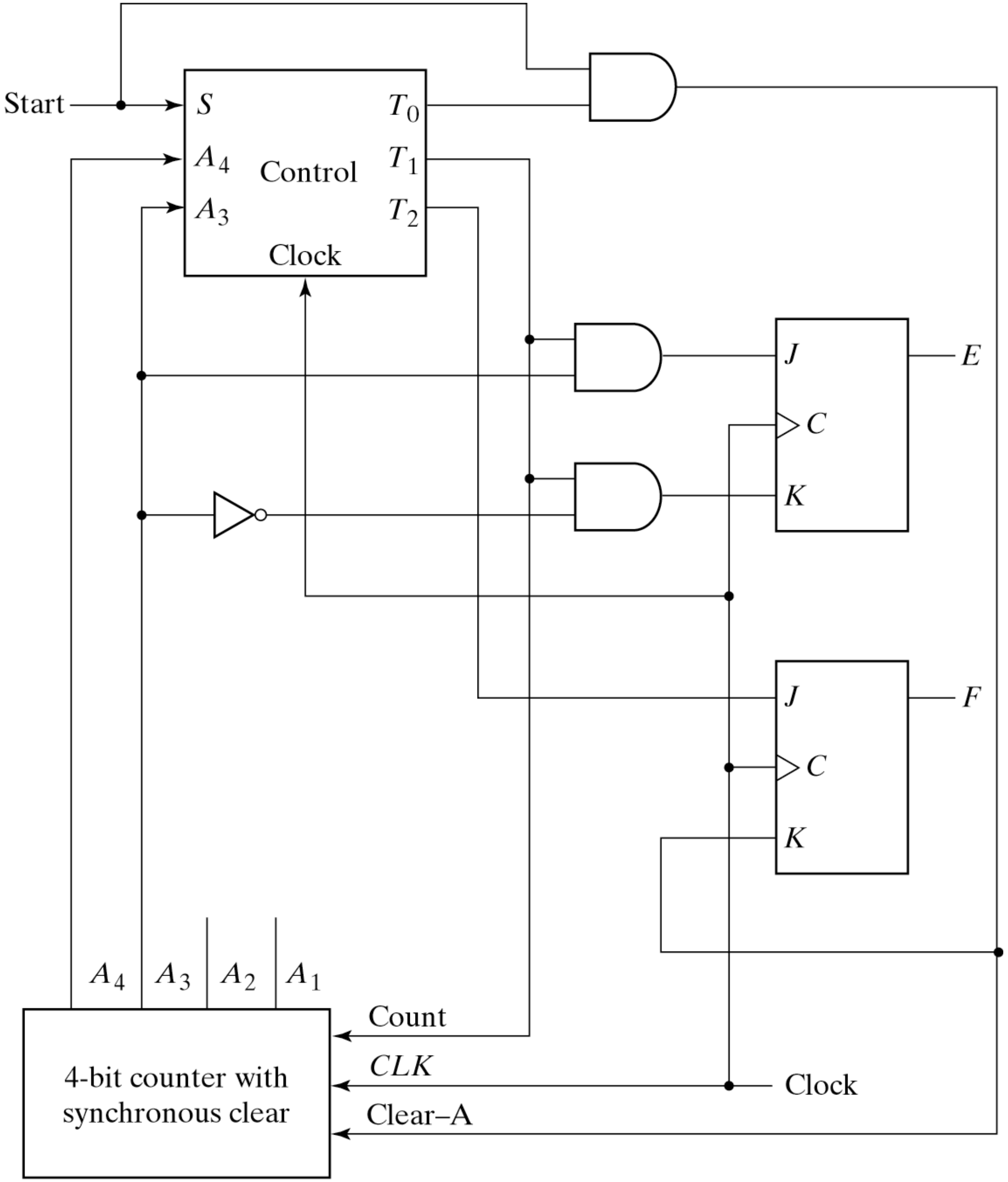


Fig. 8-10 Datapath for Design Example

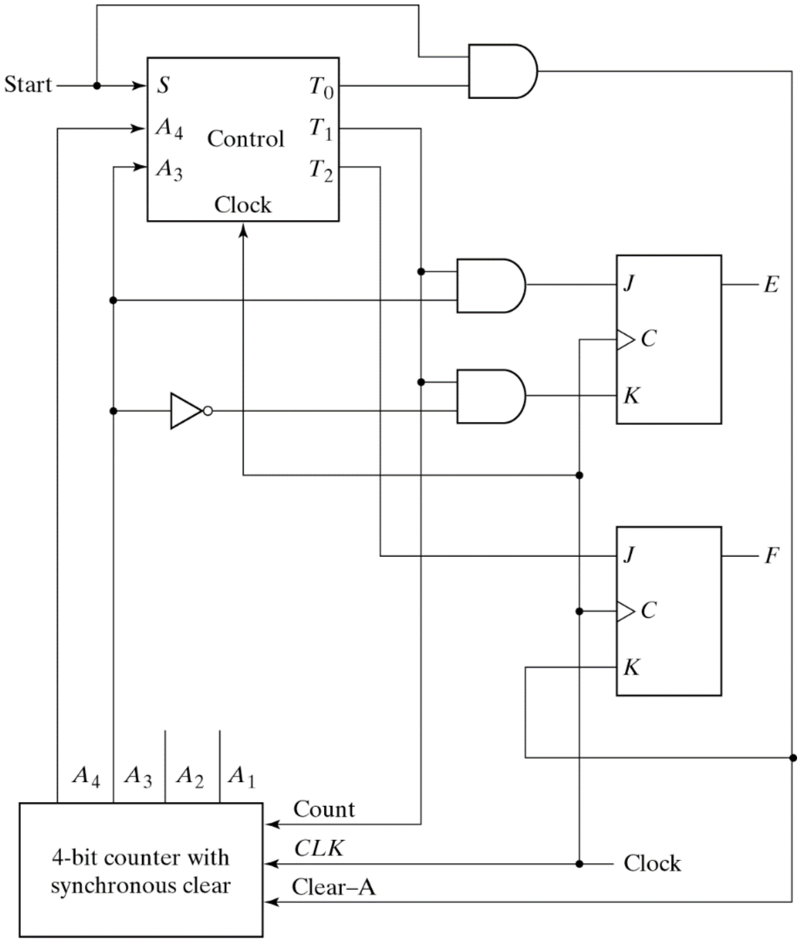


Fig. 8-10 Datapath for Design Example

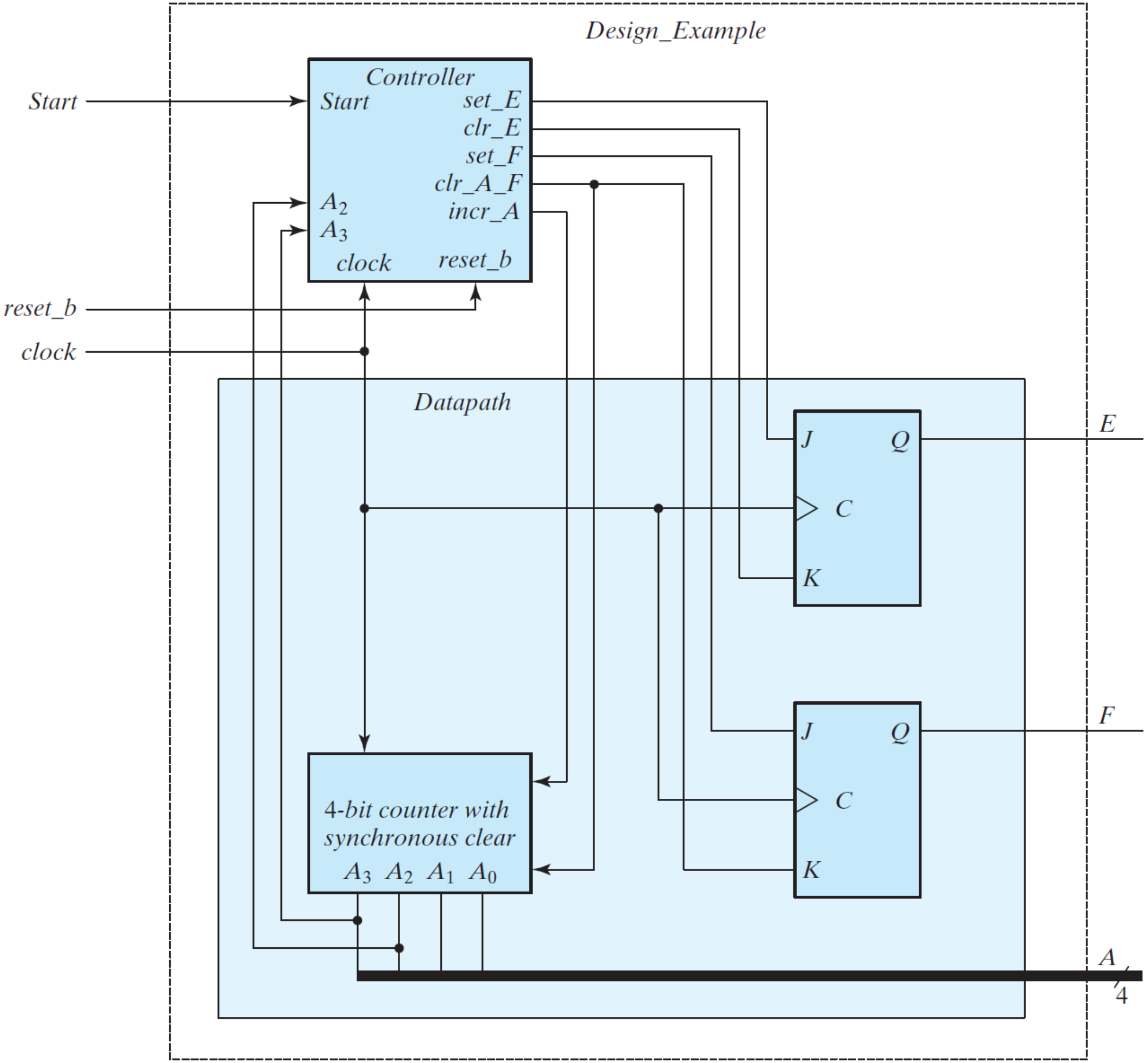


FIGURE 8.10

Datapath and controller for design example

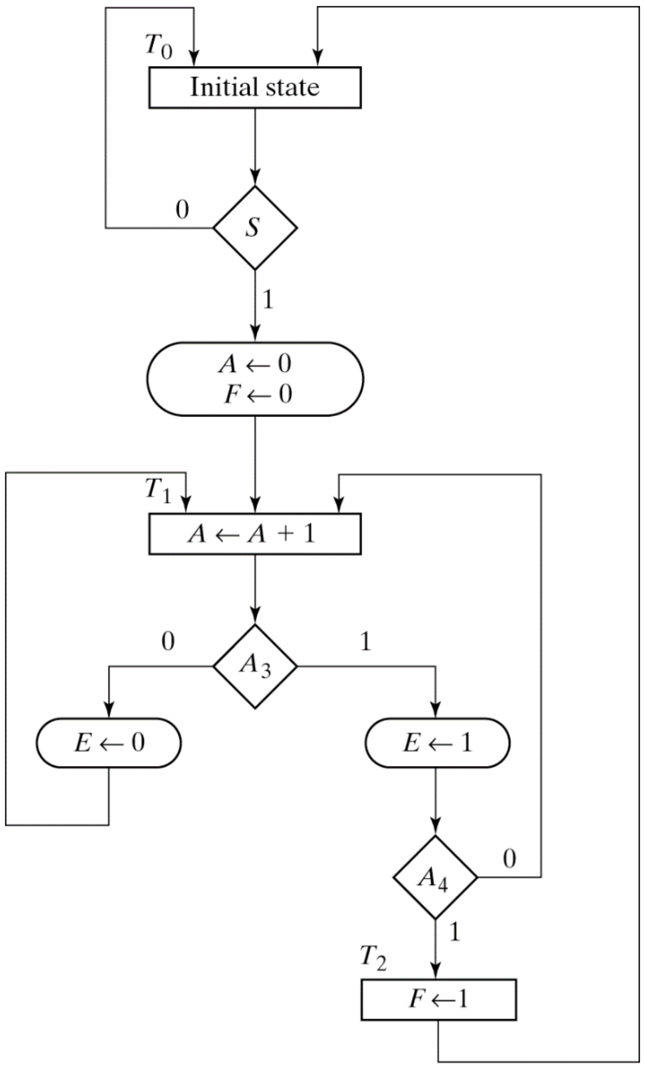
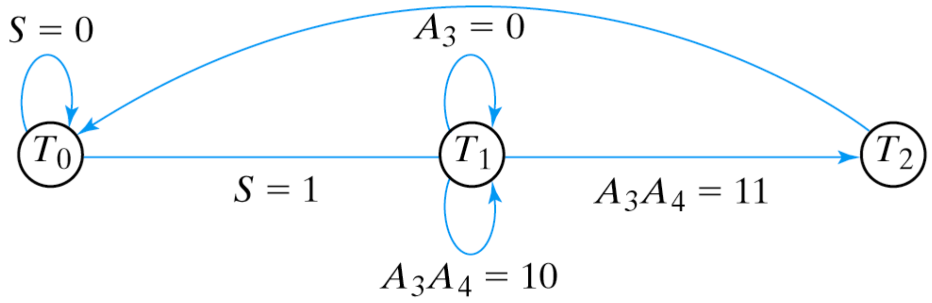


Fig. 8-9 ASM Chart for Design Example



(a) State diagram for control

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

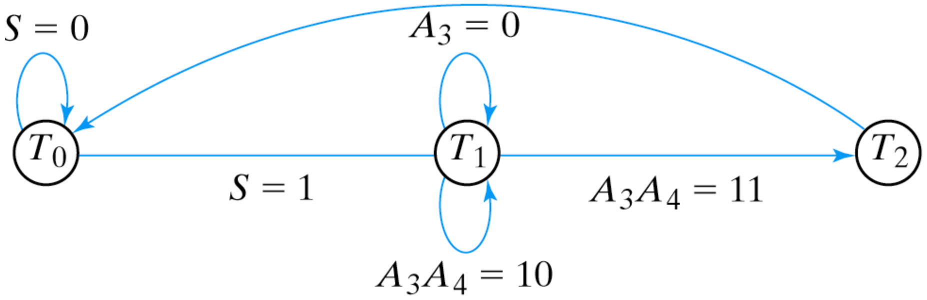
if $(A_3 = 1)$ then $E \leftarrow 1$

if $(A_3 = 0)$ then $E \leftarrow 0$

T_2: $F \leftarrow 1$

(a) Register transfer operations

Fig. 8-11 Register Transfer Level Description of Design Example



(a) State diagram for control

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

if $(A_3 = 1)$ then $E \leftarrow 1$

if $(A_3 = 0)$ then $E \leftarrow 0$

T_2: $F \leftarrow 1$

(a) Register transfer operations

Fig. 8-11 Register Transfer Level Description of Design Example

State Table:

Present-State	Present		Inputs			Next		Outputs		
	State					State				
Symbol	**G1**	**G0**	**S**	**A3**	**A4**	**G1**	**G0**	**T0**	**T1**	**T2**
T0	0	0	0	X	X	0	0	1	0	0
T0	0	0	1	X	X	0	1	1	0	0
T1	0	1	X	0	X	0	1	0	1	0
T1	0	1	X	1	0	0	1	0	1	0
T1	0	1	X	1	1	1	1	0	1	0
T2	1	1	X	X	X	0	0	0	0	1

State Table:

Present-State Symbol	Present State G1	 G0	Inputs S	 A3	 A4	Next State G1	 G0	Outputs T0	 T1	 T2
T0	0	0	0	X	X	0	0	1	0	0
T0	0	0	1	X	X	0	1	1	0	0
T1	0	1	X	0	X	0	1	0	1	0
T1	0	1	X	1	0	0	1	0	1	0
T1	0	1	X	1	1	1	1	0	1	0
T2	1	1	X	X	X	0	0	0	0	1

Design Using D FFs.

Flip-flop Inputs

DG1 = T1A3A4

DG0 = T0S +T1

Output Functions

T0 = G0’

T1 = G1’G0

T2 = G1

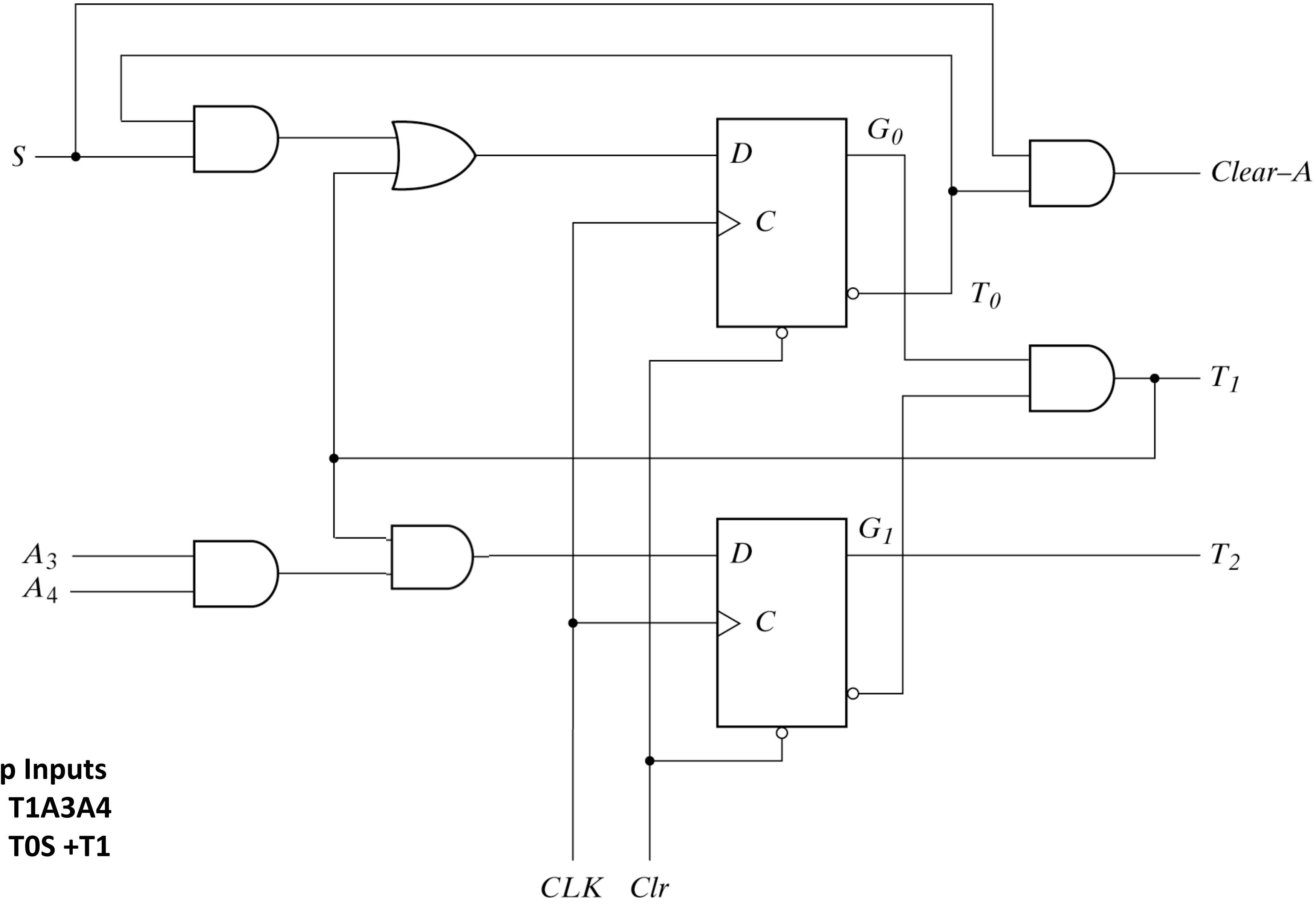


Fig. 8-12 Logic Diagram of Control

Flip-flop Inputs

DG1 = T1A3A4

DG0 = T0S +T1

Output Functions

T0 = G0’

T1 = G1’G0

T2 = G1

Thank You

8.7 SEQUENTIAL BINARY MULTIPLIER

- This section introduces a second design example.
- It presents a hardware algorithm for binary multiplication, proposes the register configuration for its implementation, and
- then shows how to use an ASMD chart to design its datapath and its controller.
- The system we will examine multiplies two unsigned binary numbers.
- The hardware developed in Section 4.7 to execute multiplica resulted in a combinational circuit multiplier with many adders and AND gates, and requires large area of silicon as an integrated circuit.

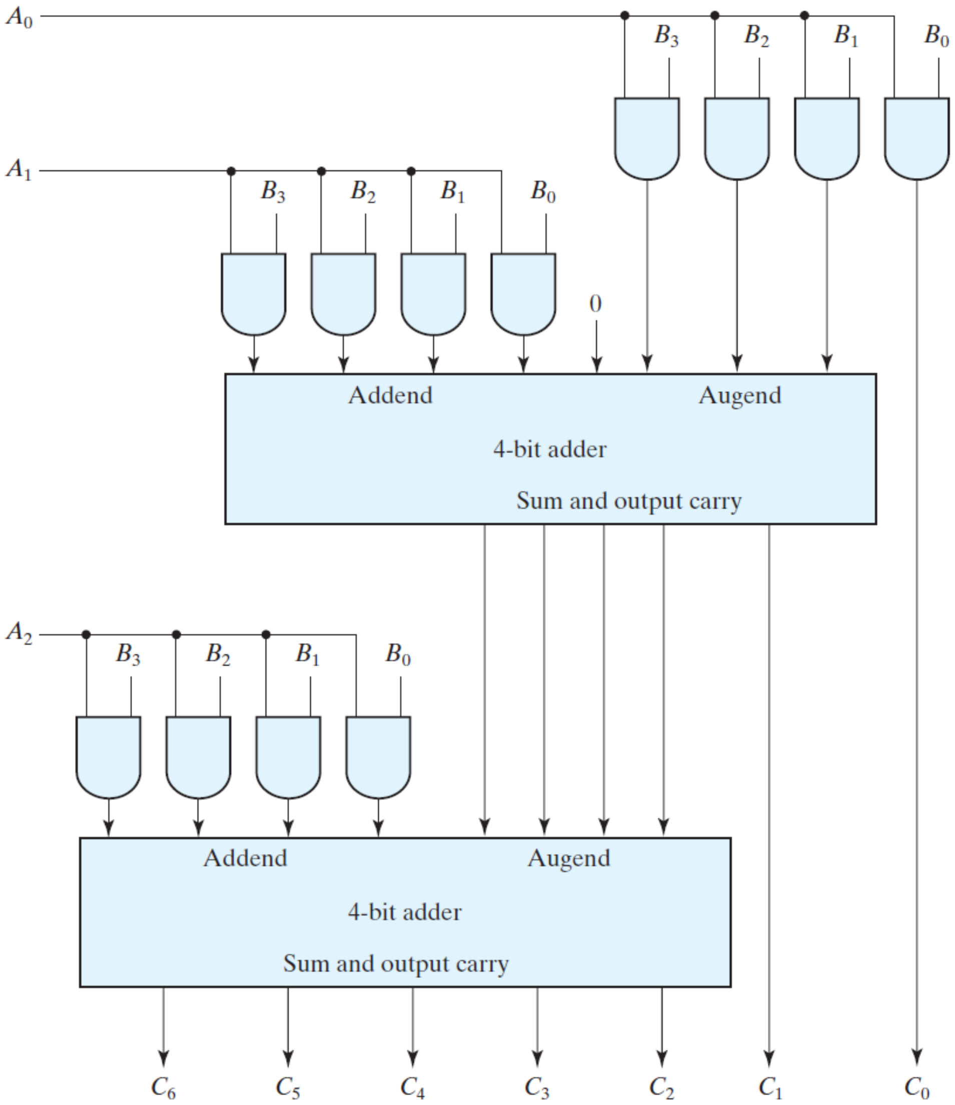
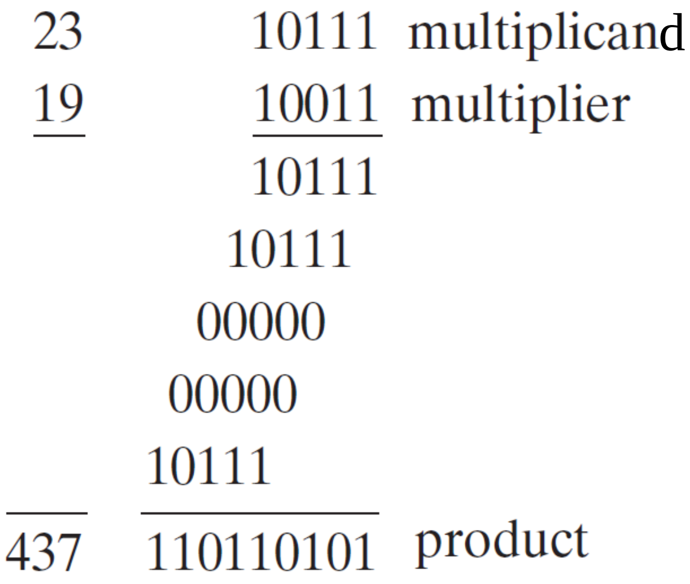


FIGURE 4.16
Four-bit by three-bit binary multiplier

- The multiplication of two binary numbers is done with paper and pencil by successive (i.e., sequential) additions and shifting.

- The 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 successively adding and shifting copies of the multiplicand.

- Successive bits of the multiplier are examined, least significant bit first.

- If the multiplier bit is 1, the multiplicand is copied down; otherwise, 0’s are copied down.

- The numbers copied 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.

- The product obtained from the multiplication of two binary numbers of *n* bits each can have up to 2 *n* bits.

- It is apparent that the operations of addition and shifting are executed by the algorithm.

- When the multiplication process is implemented with digital hardware, it is convenient to change the process slightly.

- First, we note that, in the context of synthesizing a sequential machine, the add-and-shift algorithm for binary multiplication can be executed in a single clock cycle or over multiple clock cycles.

- A choice to form the product in the time span of a single clock cycle will synthesize the circuit of a parallel multiplier like the one discussed in Section 4.7.

- On the other hand, an RTL model of the algorithm adds shifted copies of the multiplicand to an accumulated partial product.

- The values of the multiplier, multiplicand, and partial product are stored in registers, and the operations of shifting and adding their contents are executed under the control of a state machine.

- Among the many possibilities for distributing the effort of multiplication over multiple clock cycles, we will consider that in which only one partial product is formed and accumulated in a single cycle of the clock. (

- Instead of providing digital circuits to store and add simultaneously as many binary numbers as there are 1’s in the multiplier, it is less expensive to provide only the hardware needed to sum two binary numbers and accumulate the partial products in a register.

- Second, instead of shifting the multiplicand to the left, the partial product being formed is shifted to the right. This leaves the partial product and the multiplicand in the required relative positions.

- Third, when the corresponding bit of the multiplier is 0, there is no need to add all 0’s to the partial product, since doing so will not alter its resulting value.

Register Configuration

- A block diagram for the sequential binary multiplier is shown in Fig. 8.14 (a), and the register configuration of the datapath is shown in Fig. 8.14 (b).

- The multiplicand is stored in register *B*, the multiplier is stored in register *Q*, and the partial product is formed in register *A* and stored in *A* and *Q* .

- A parallel adder adds the contents of register *B* to register *A* .

- The *C* flip-flop stores the carry after the addition. The counter *P* 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.

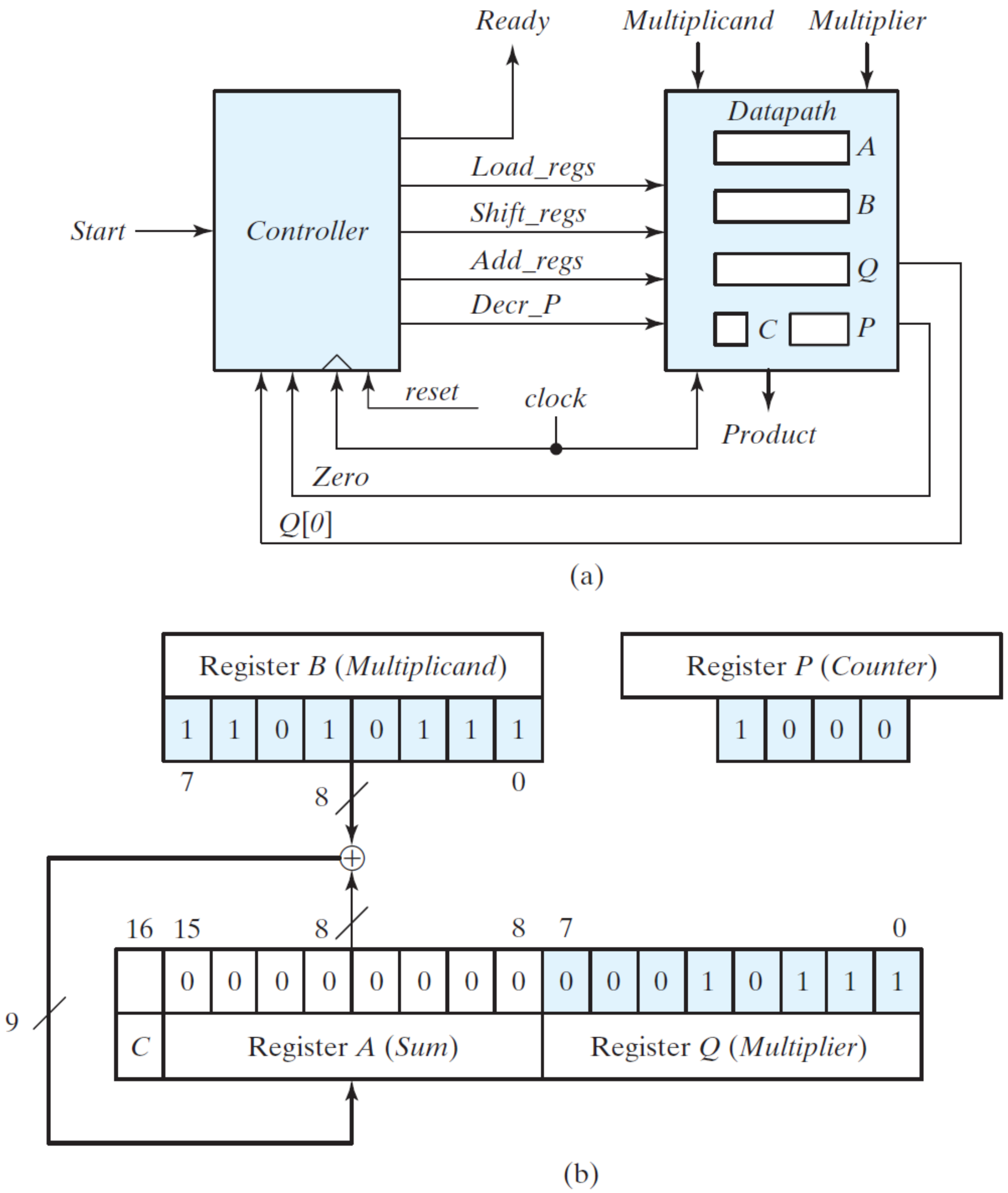
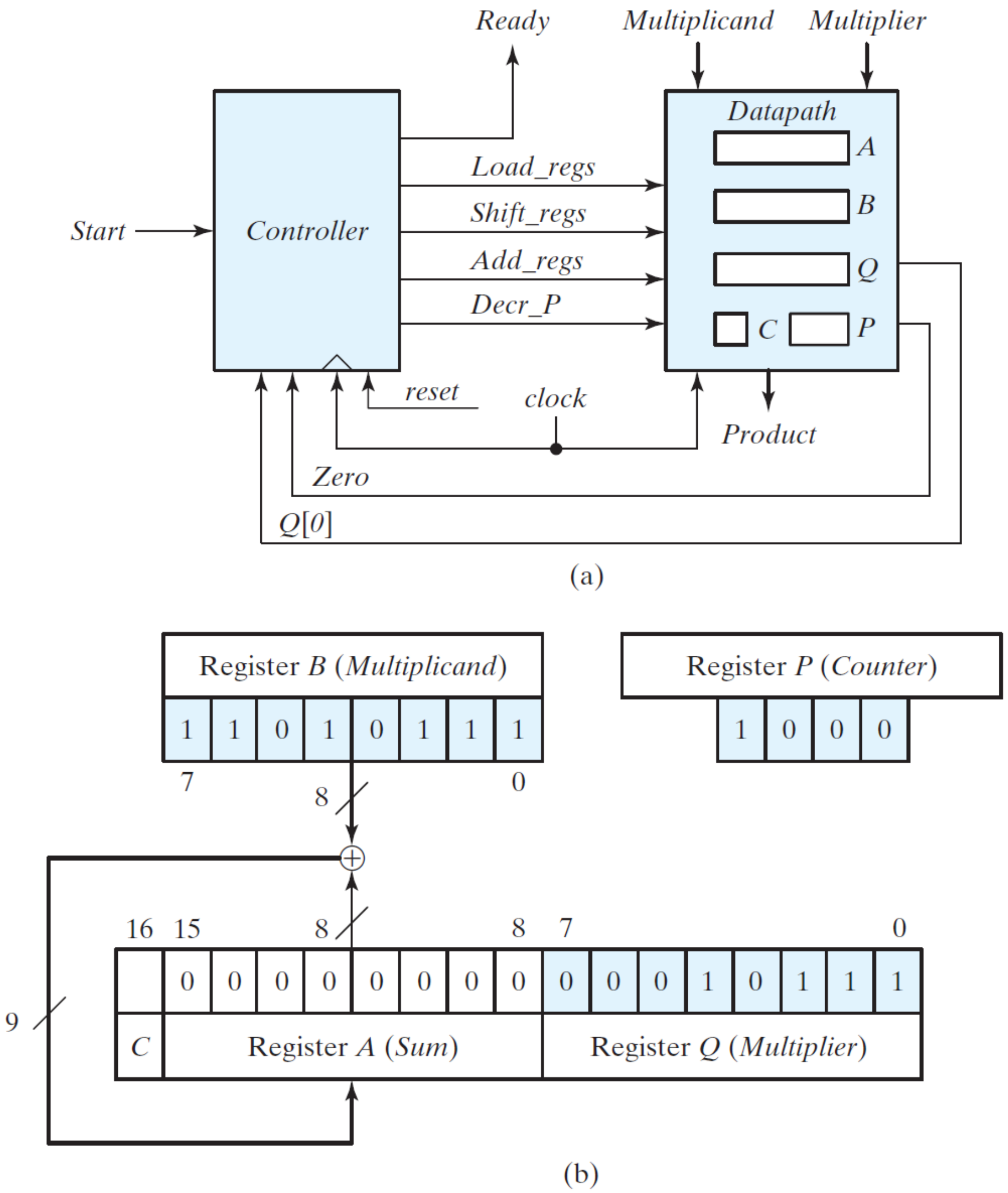


FIGURE 8.14
(a) Block diagram and (b) datapath of a binary multiplier

- The control logic stays in an initial state until *Start* becomes 1.
- The system then performs the multiplication. The sum of *A* and *B* forms the *n* most significant bits of the partial product, which is transferred to *A* .
- The output carry from the addition, whether 0 or 1, is transferred to *C* .
- 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 carry from *C* is shifted into the most significant position of *A*, and 0 is shifted into *C* .
- 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 least significant bit of register *Q*, designated by *Q[0]*, holds the bit of the multiplier that must be inspected next.



(a)

(b)

FIGURE 8.14
(a) Block diagram and (b) datapath of a binary multiplier

- The control logic determines whether to add or not on the basis of this input bit.
- The control logic also receives a signal, *Zero*, from a circuit that checks counter *P* for zero.
- *Q[0]* and *Zero* are status inputs for the control unit.
- The input signal *Start* is an external control input.
- The outputs of the control logic launch the required operations in the registers of the datapath unit.
- The interface between the controller and the datapath consists of the status signals and the output signals of the controller.
- The control signals govern the synchronous register operations of the datapath.
- Signal *Load_regs* loads the internal registers of the datapath, *Shift_regs* causes the shift register to shift, *Add_regs* forms the sum of the multiplicand and register *A*, and *Decr_P* decrements the counter.

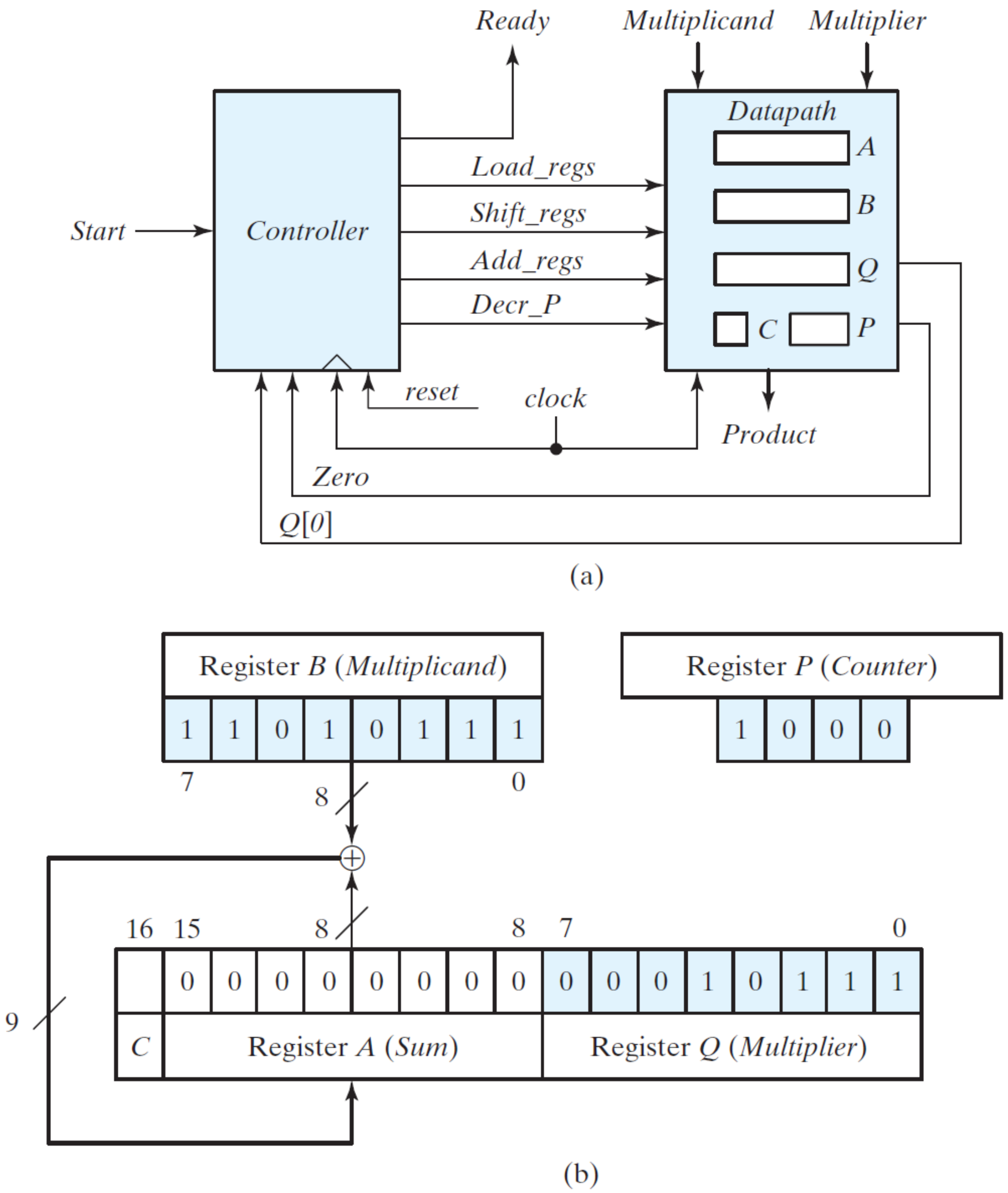


FIGURE 8.14
(a) Block diagram and (b) datapath of a binary multiplier

- The controller also forms output *Ready* to signal to the host environment that the machine is ready to multiply.

- The contents of the register holding the product vary during execution, so it is useful to have a signal indicating that its contents are valid.

- Note, again, that the state of the control is not an interface signal between the control unit and the datapath.

- Only the signals needed to control the datapath are included in the interface.

- Putting the state in the interface would require a decoder in the datapath, and would require a wider and more active bus than the control signals alone. Not good.

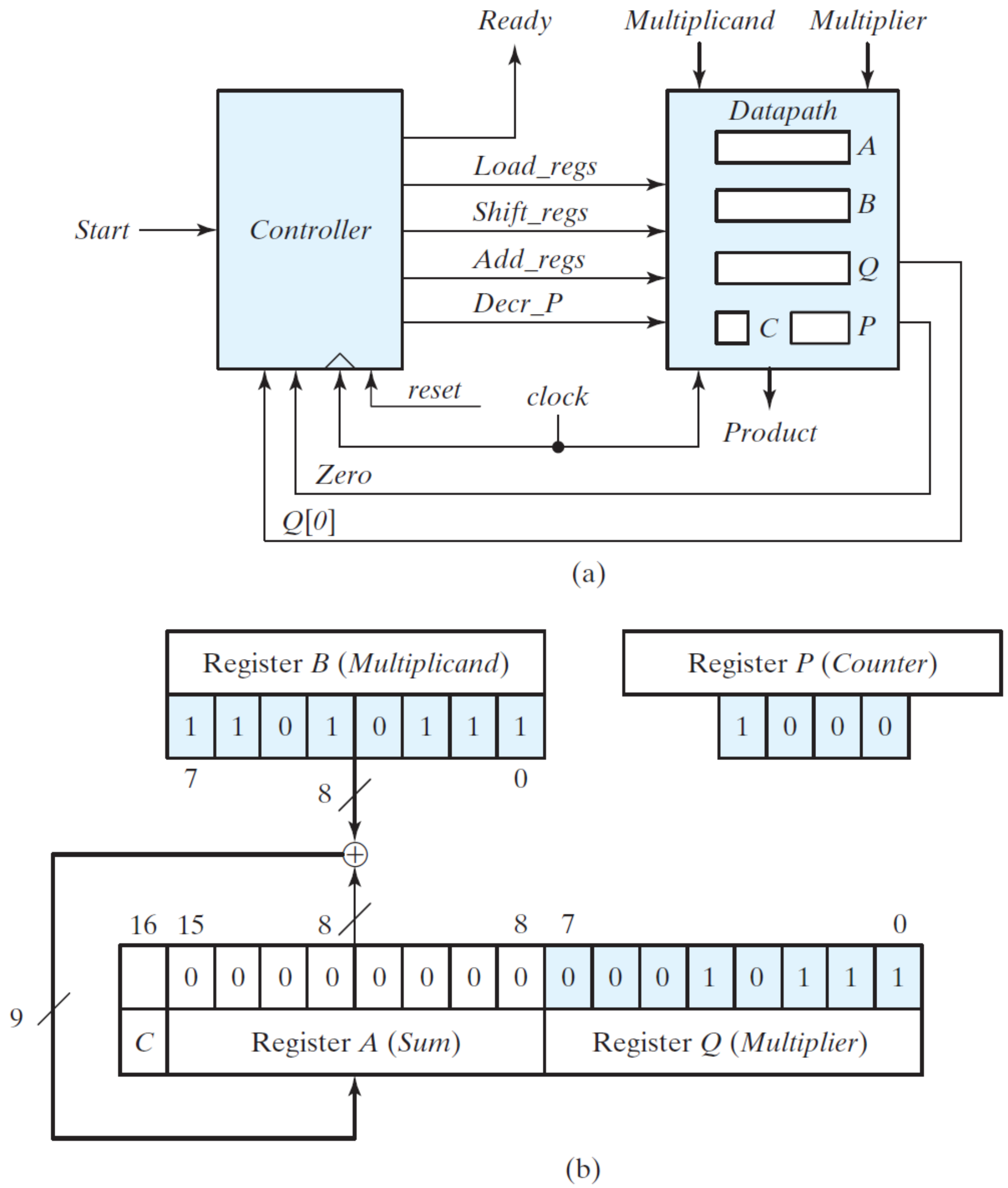


FIGURE 8.14
(a) Block diagram and (b) datapath of a binary multiplier

ASMD Chart

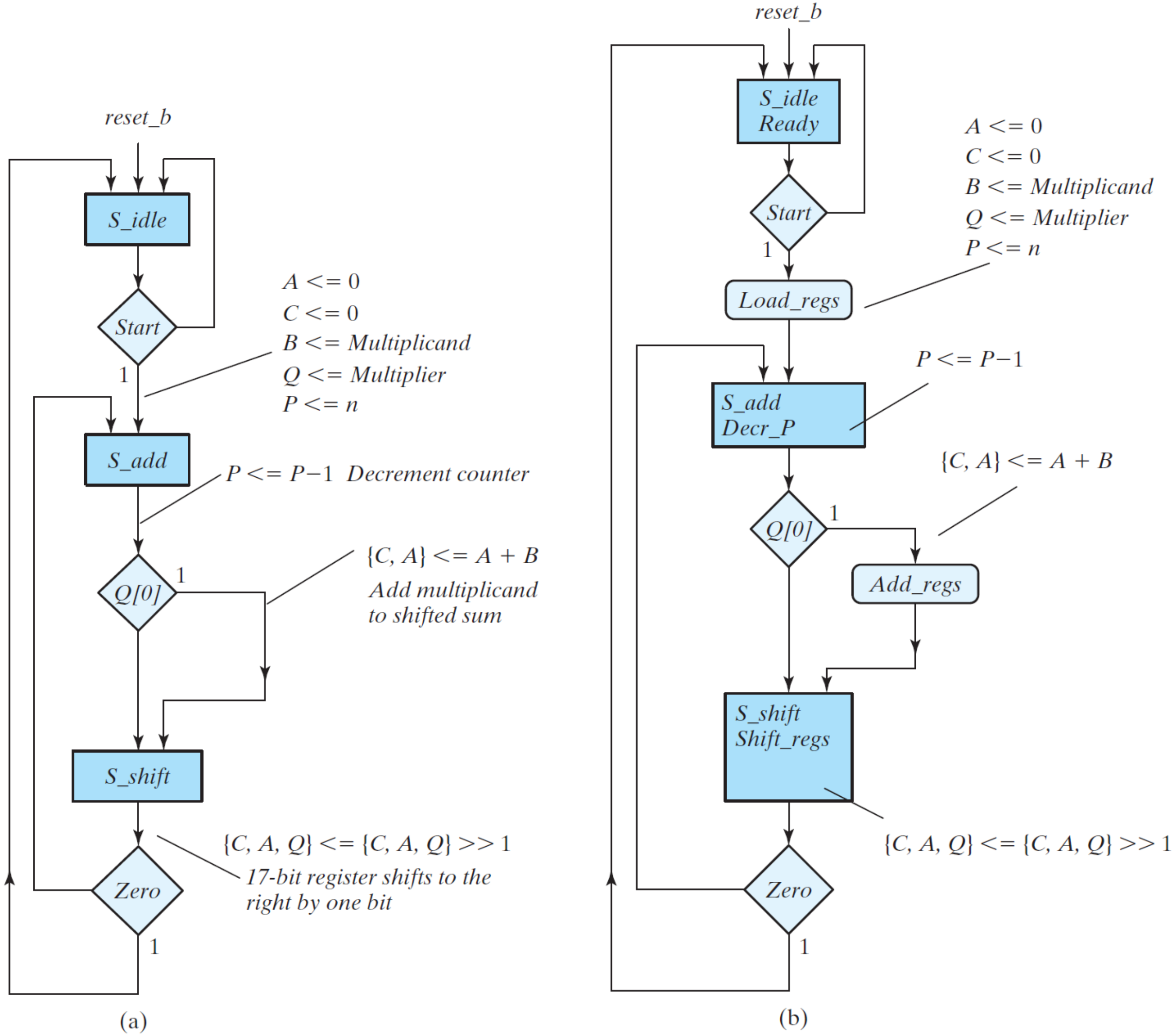


FIGURE 8.15
ASMD chart for binary multiplier

- The ASMD chart for the binary multiplier is shown in Fig. 8.15 .

- The intermediate form in Fig. 8.15 (a) annotates the ASM chart of the controller with the register operations, and

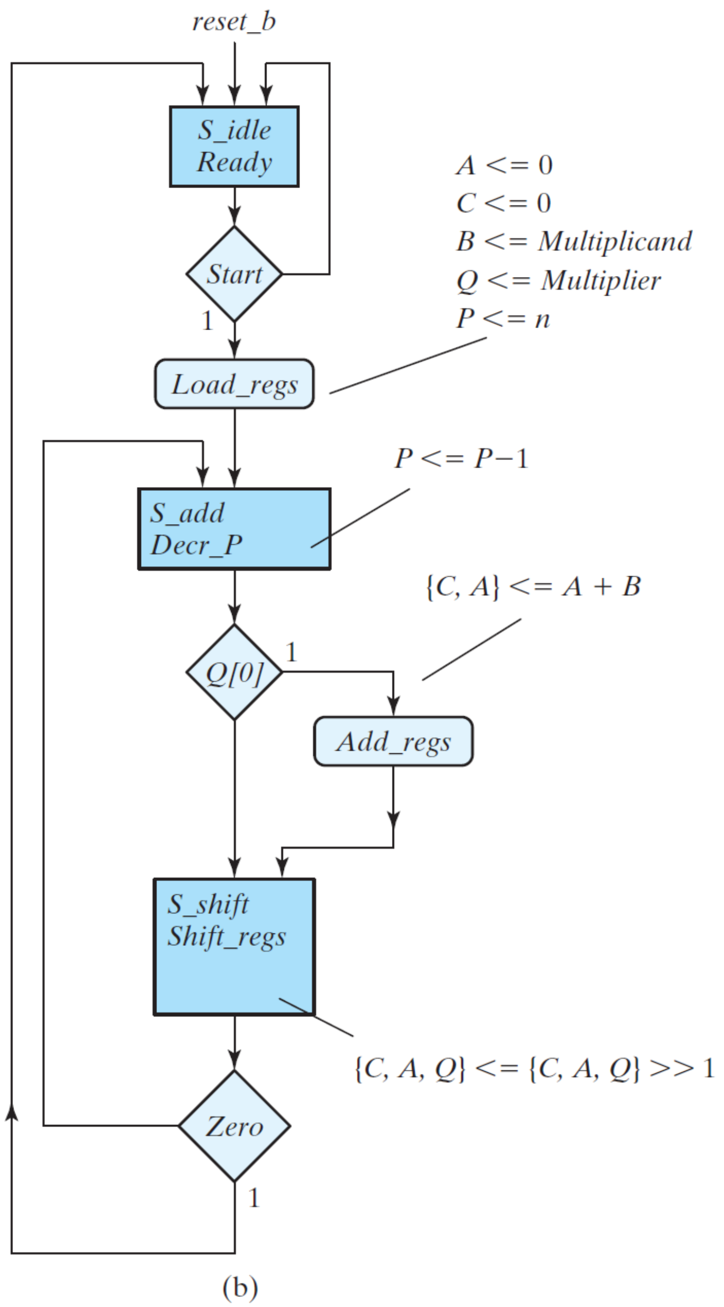
- the completed chart in Fig. 8.15 (b) identifies the Moore and Mealy outputs of the controller.

- Initially, the multiplicand is in *B* and the multiplier in *Q* .

- As long as the circuit is in the initial state and *Start* = 0, no action occurs and the system remains in state *S_idle* with *Ready* asserted.

- The multiplication process is launched when *Start* = 1.

ASMD Chart



(b)

- The multiplication process is launched when *Start* = 1.

- Then, (1) control goes to state *S_add*, (2) register A and carry flip-flop C are cleared to 0, (3) registers *B* and *Q* are loaded with the multiplicand and the multiplier, respectively, and (4) the sequence counter *P* is set to a binary number *n*, equal to the number of bits in the multiplier.

- In state *S_add*, the multiplier bit in *Q[0]* 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 *C* . The partial product in *A* and *C* is left unchanged if *Q[0]* = 0.

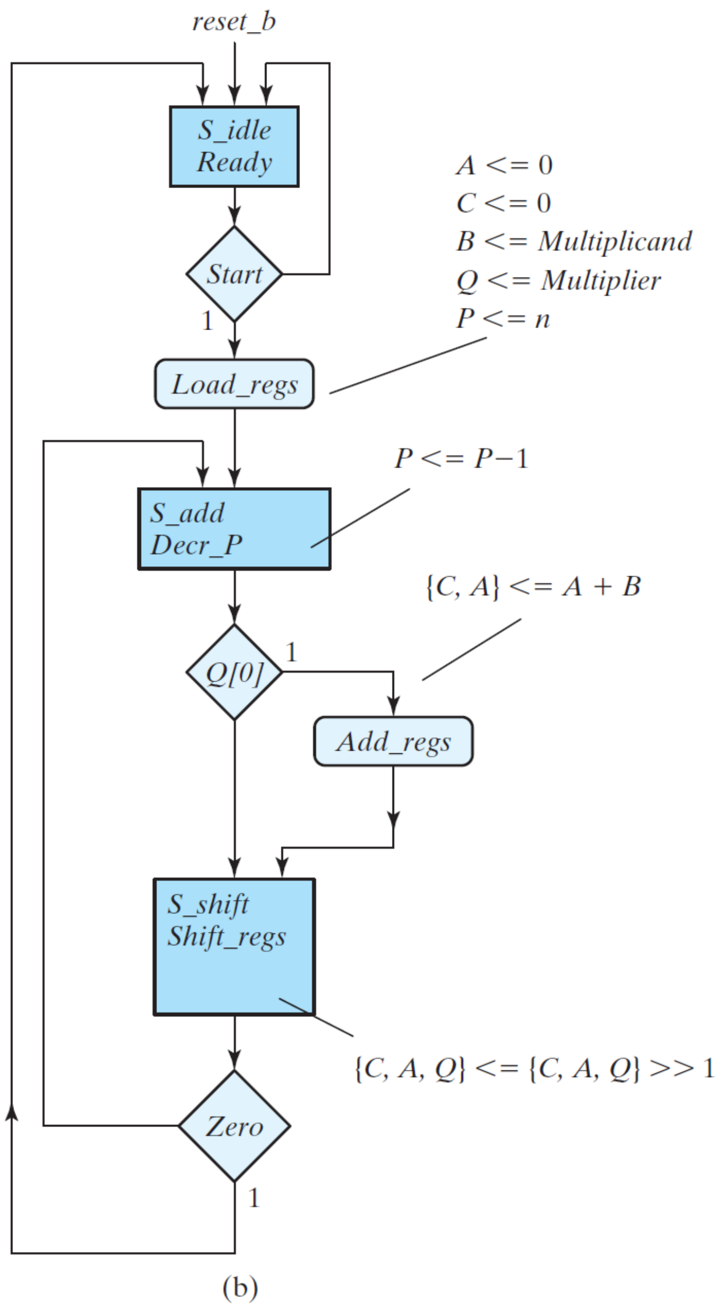
- The counter *P* is decremented by 1 regardless of the value of *Q[0]*, so *Decr_P* is formed in state *S_add* as a Moore output of the controller. In both cases, the next state is *S_shift* .

- Registers *C*, *A*, and *Q* are combined into one composite register *CAQ*, denoted by the concatenation { *C*, *A*, *Q* }, and its contents are shifted once to the right to obtain a new partial product.

- This shift operation is symbolized in the flowchart with the Verilog logical right-shift operator, >>. It is equivalent to the following statement in register transfer notation:

$$\text{Shift right } CAQ, C \leftarrow 0$$

ASMD Chart



- In terms of individual register symbols, the shift operation can be described by the following register operations:

$$A \leftarrow \text{shr } A, A_{n-1} \leftarrow C$$
$$Q \leftarrow \text{shr } Q, Q_{n-1} \leftarrow A_0$$
$$C \leftarrow 0$$

- Both registers *A* and *Q* are shifted right. The leftmost bit of *A*, designated by $A_{n\text{-}1}$, receives the carry from *C* . The leftmost bit of *Q*, $Q_{n\text{-}1}$, receives the bit from the rightmost position of *A* in A_0 , and *C* is reset to 0.

- In essence, this is a long shift of the composite register *CAQ* with 0 inserted into the serial input, which is at *C* .

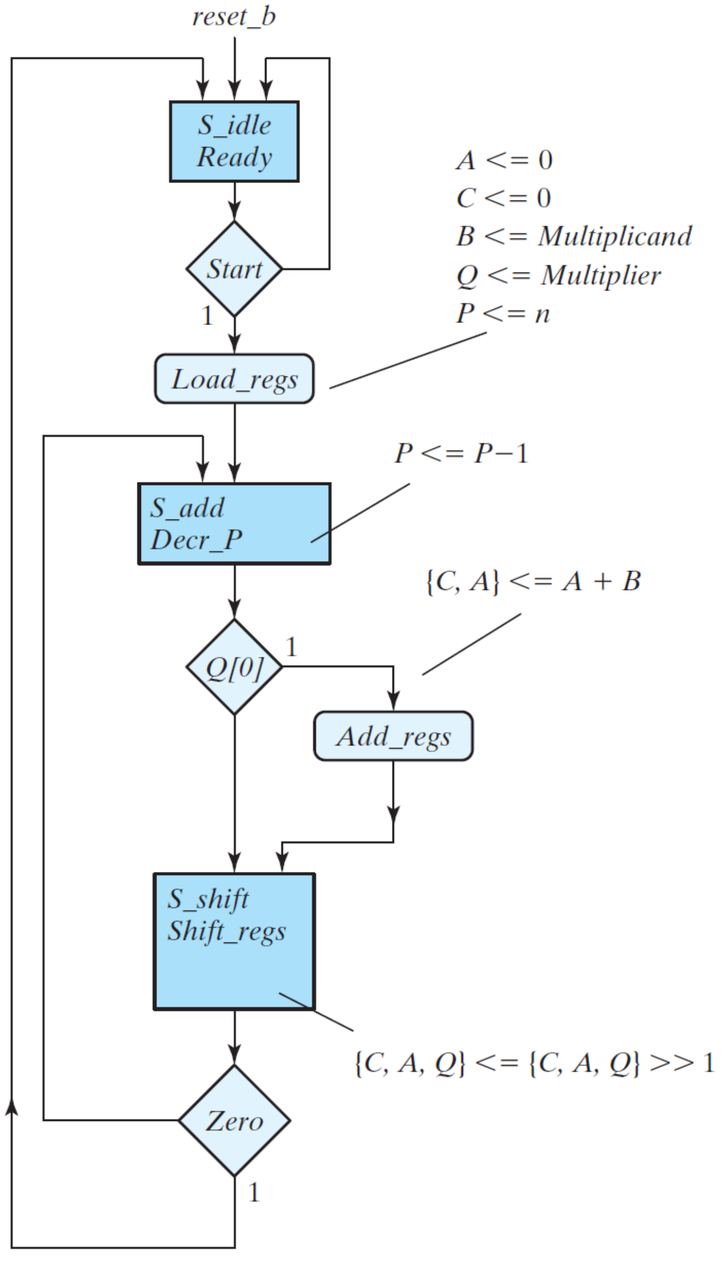
- The value in counter *P* is checked after the formation of each partial product. If the contents of *P* are different from zero, status bit *Zero* is set equal to 0 and the process is repeated to form a new partial product.

- The process stops when the counter reaches 0 and the controller's status input *Zero* 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 of the product.

ASMD Chart



(b)

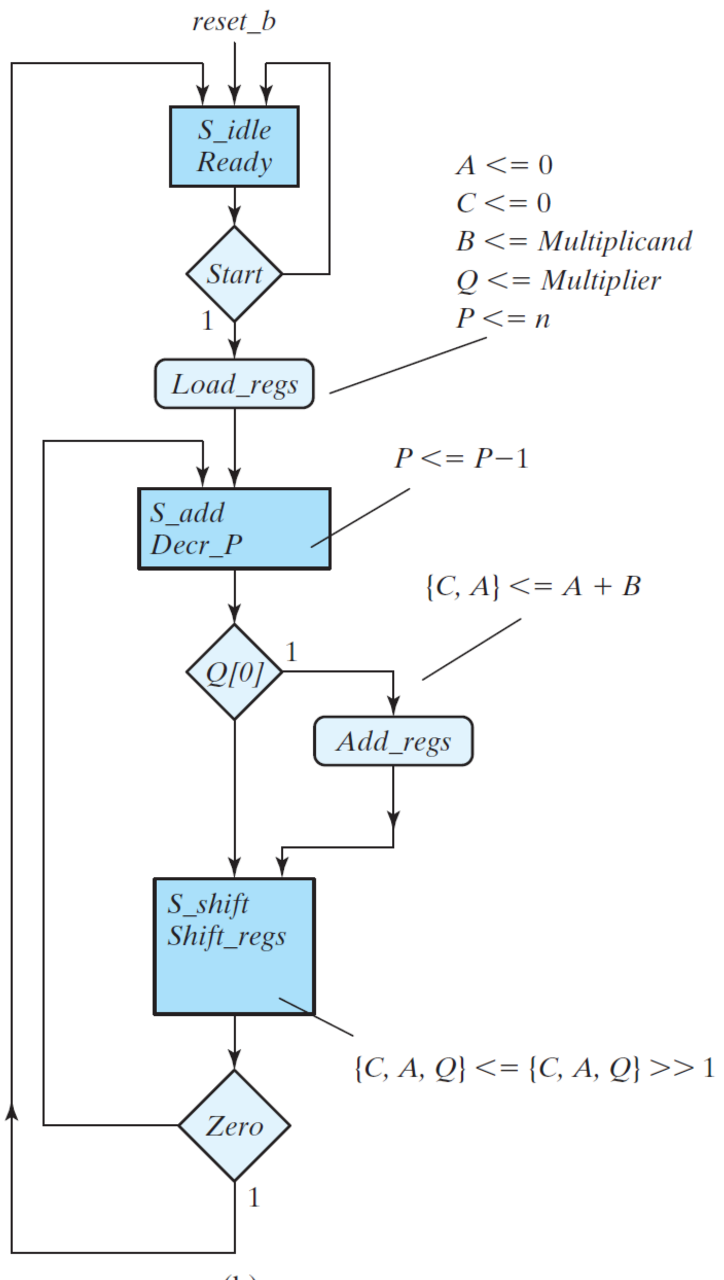
Table 8.5 to clarifies the multiplication process. The procedure follows the steps outlined in the ASMD chart.

Table 8.5
Numerical Example For Binary Multiplier

Multiplicand $B = 10111_2 = 17_H = 23_{10}$ **Multiplier $Q = 10011_2 = 13_H = 19_{10}$**

	C	A	Q	P
Multiplier in Q	0	00000	10011	101
$Q_0 = 1$; add B		10111		
First partial product	0	10111		100
Shift right CAQ	0	01011	11001	
$Q_0 = 1$; add B		10111		
Second partial product	1	00010		011
Shift right CAQ	0	10001	01100	
$Q_0 = 0$; shift right CAQ	0	01000	10110	010
$Q_0 = 0$; shift right CAQ	0	00100	01011	001
$Q_0 = 1$; add B		10111		
Fifth partial product	0	11011		
Shift right CAQ	0	01101	10101	000
Final product in $AQ = 0110110101_2 = 1b5_H$				

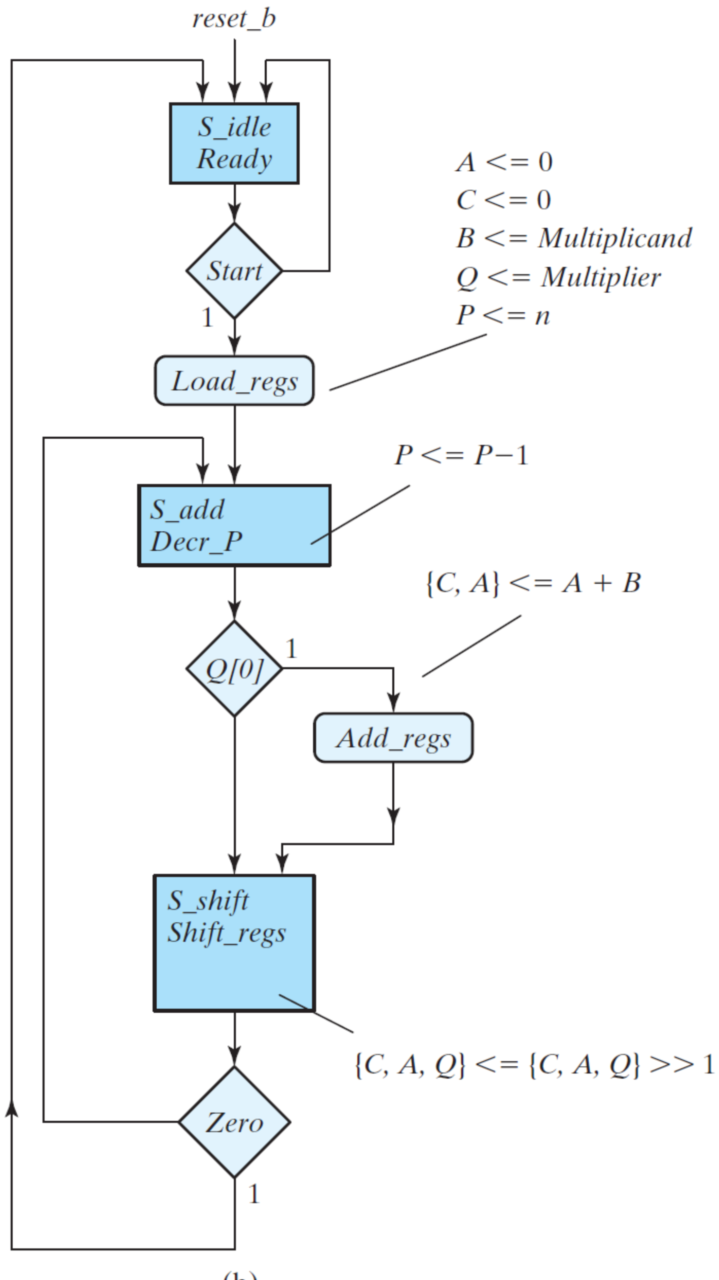
ASMD Chart



(b)

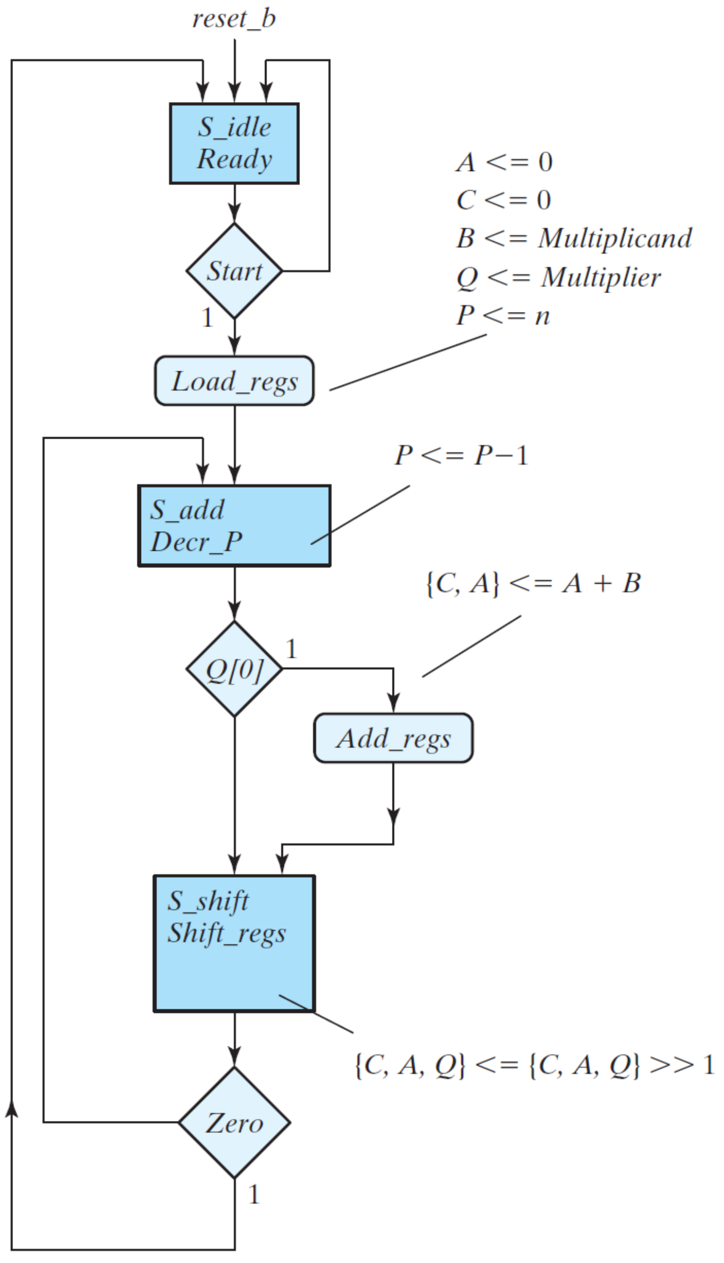
- The type of registers needed for the data processor subsystem can be derived from the register operations listed in the ASMD chart.
- Register *A* is a shift register with parallel load to accept the sum from the adder and must have a synchronous clear capability to reset the register to 0.
- Register *Q* is a shift register.
- The counter *P* is a binary down counter with a facility to parallel load a binary constant.
- The *C* flip-flop must be designed to accept the input carry and have a synchronous clear.
- Registers *B* and *Q* need a parallel load capability in order to receive the multiplicand and multiplier prior to the start of the multiplication process.

8.8 CONTROL LOGIC

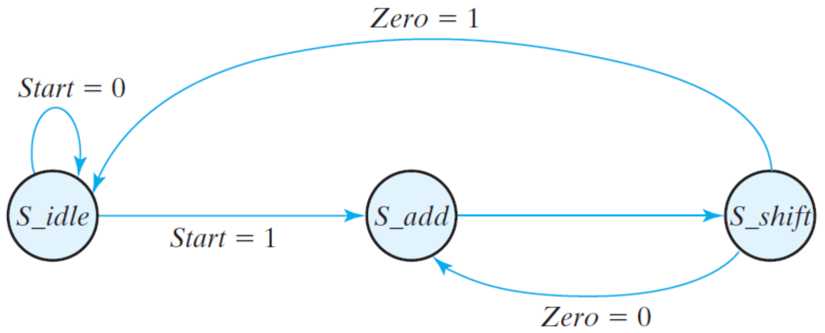


(b)

- The design of a digital system can be divided into two parts: (1) the design of the register transfers in the datapath unit and (2) the design of the control logic of the control unit.
- The control logic is a finite state machine; its Mealy- and Moore-type outputs control the operations of the datapath.
- The inputs to the control unit are the primary (external) inputs and the internal status signals fed back from the datapath to the controller.
- The design of the system can be synthesized from an RTL description derived from the ASMD chart.
- Alternatively, a manual design must derive the logic governing the inputs to the flip-flops holding the state of the controller.
- The information needed to form the state diagram of the controller is already contained in the ASMD chart, since the rectangular blocks that designate state boxes are the states of the sequential circuit.
- The diamond-shaped blocks that designate decision boxes determine the logical conditions for the next state transition in the state diagram and assertions of the conditional outputs.



(b)



(a)

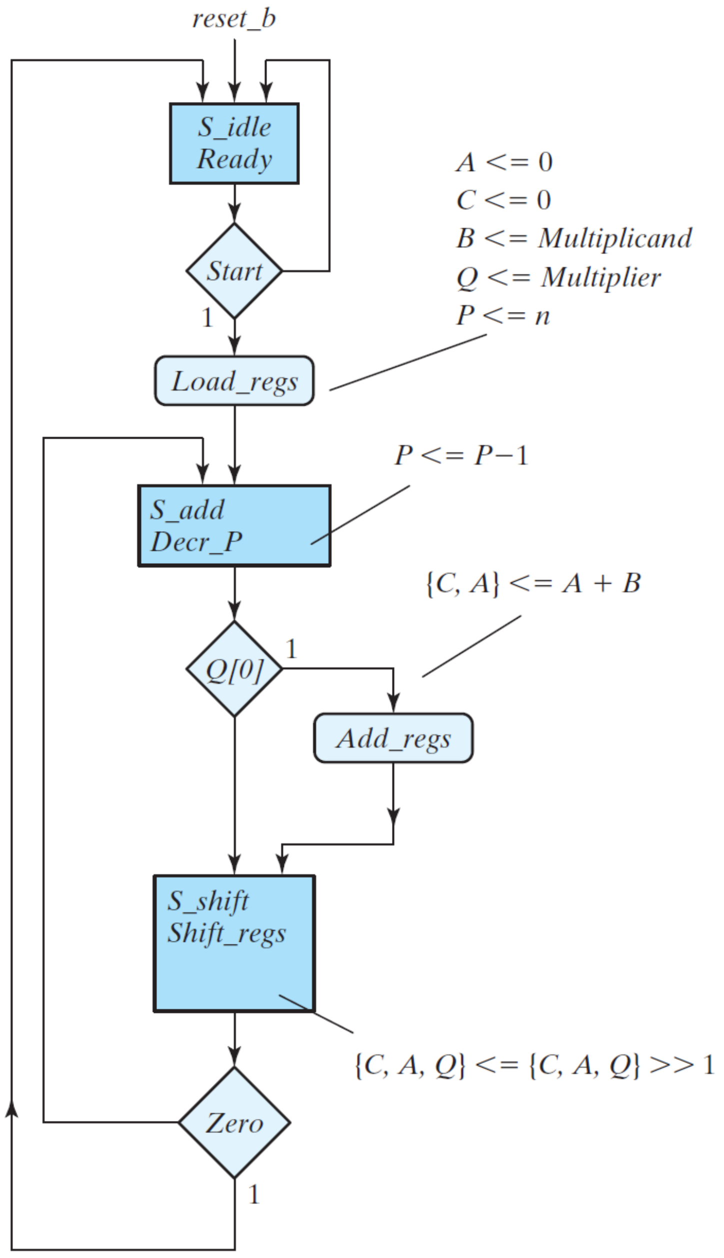
State Transition		Register Operations
<u>From</u>	<u>To</u>	
S_idle		Initial state
S_idle	*S_add*	$A <= 0, C <= 0, P <= dp\_width$
S_add	*S_shift*	$P <= P - 1$ if ($Q[0]$) then ($A <= A + B, C <= C_{out}$)
S_shift		shift right $\{CAQ\}, C <= 0$

(b)

FIGURE 8.16
Control specifications for binary multiplier

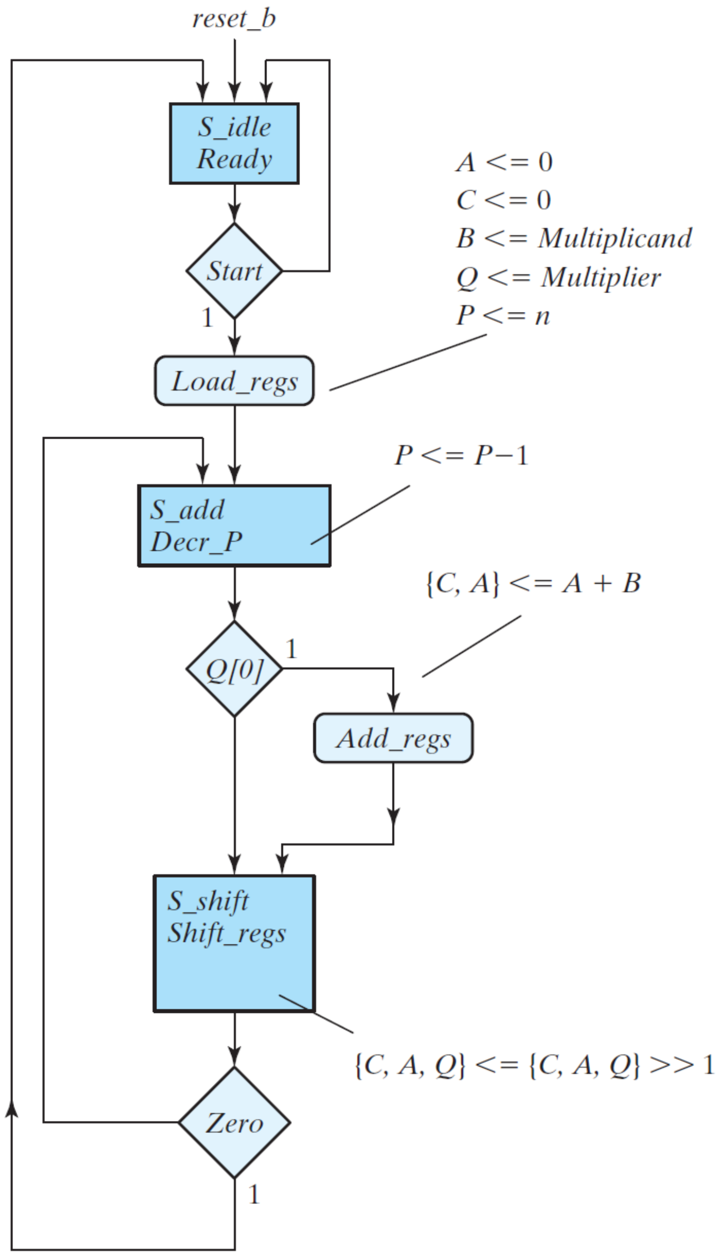
- ✓ The register transfer operations for each of the three states are listed in Fig. 8.16 (b) and are taken from the corresponding state and conditional boxes in the ASMD chart.

- ✓ Establishing the state transitions is the initial focus, so the outputs of the controller are not shown.



(b)

- We must execute two steps when implementing the control logic: (1) establish the required sequence of states, and (2) provide signals to control the register operations.
- The sequence of states is specified in the ASMD chart or the state diagram.
- The signals for controlling the operations in the registers are specified in the register transfer statements annotated on the ASMD chart or listed in tabular format.
- For the multiplier, these signals are *Load_regs* (for parallel loading the registers in the datapath unit), *Decr_P* (for decrementing the counter), *Add_regs* (for adding the multiplicand and the partial product), and *Shift_regs* (for shifting register *CAQ*).
- The block diagram of the control unit is shown in Fig. 8.14 (a).
- The inputs to the controller are *Start*, *Q[0]*, and *Zero*, and the outputs are *Ready*, *Load_regs*, *Decr_P*, *Add_regs*, and *Shift_regs*, as specified in the ASMD chart.
- We note that *Q[0]* affects only the output of the controller, not its state transitions. The machine transitions from *S_add* to *S_shift* unconditionally.



(b)

- An important step in the design is the assignment of coded binary values to the states

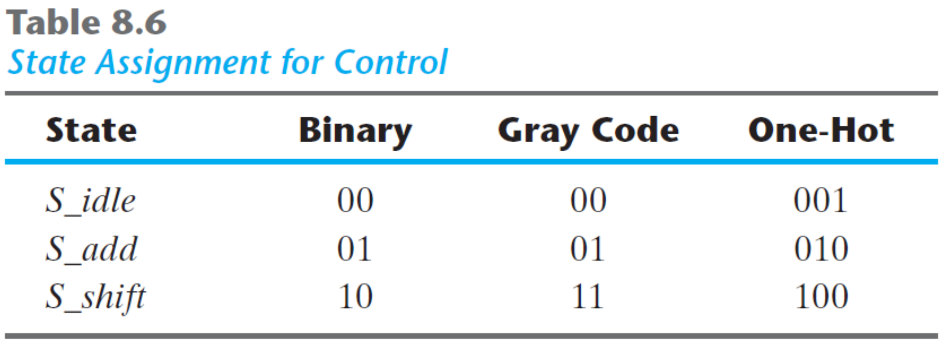


Table 8.6
State Assignment for Control

State	Binary	Gray Code	One-Hot
S_idle	00	00	001
S_add	01	01	010
S_shift	10	11	100

- Indeed, one-hot encoding uses more flip-flops than other types of coding, but it usually leads to simpler decoding logic for the next state and the output of the machine.

- Because the decoding logic does not become more complex as states are added to the machine, the speed at which the machine can operate is not limited by the time required to decode the state.

- Since the controller is a sequential circuit, it can be designed manually by the sequential logic procedure outlined in Chapter 5 .

- However, in most cases this method is difficult to carry out manually because of the large number of states and inputs that a typical control circuit may have.

- As a consequence, it is necessary to use specialized methods for control logic design that may be considered as variations of the classical sequential logic method.

- We will now present two such design procedures.
  - ✓ One uses a sequence register and decoder, and
  - ✓ the other uses one flip-flop per state.

- The method will be presented for a small circuit, but it applies to larger circuits as well.

- Of course, the need for these methods is eliminated if one has software that automatically synthesizes the circuit from an HDL description.

Sequence Register and Decoder

- The sequence-register-and-decoder (manual) method, as the name implies, uses a register for the control states and a decoder to provide an output corresponding to each of the states. (The decoder is not needed if a one-hot code is used.)

- A register with *n* flipflops can have up to 2^{n} states, and an *n* -to-2^{n} -line decoder has up to 2^{n} outputs.

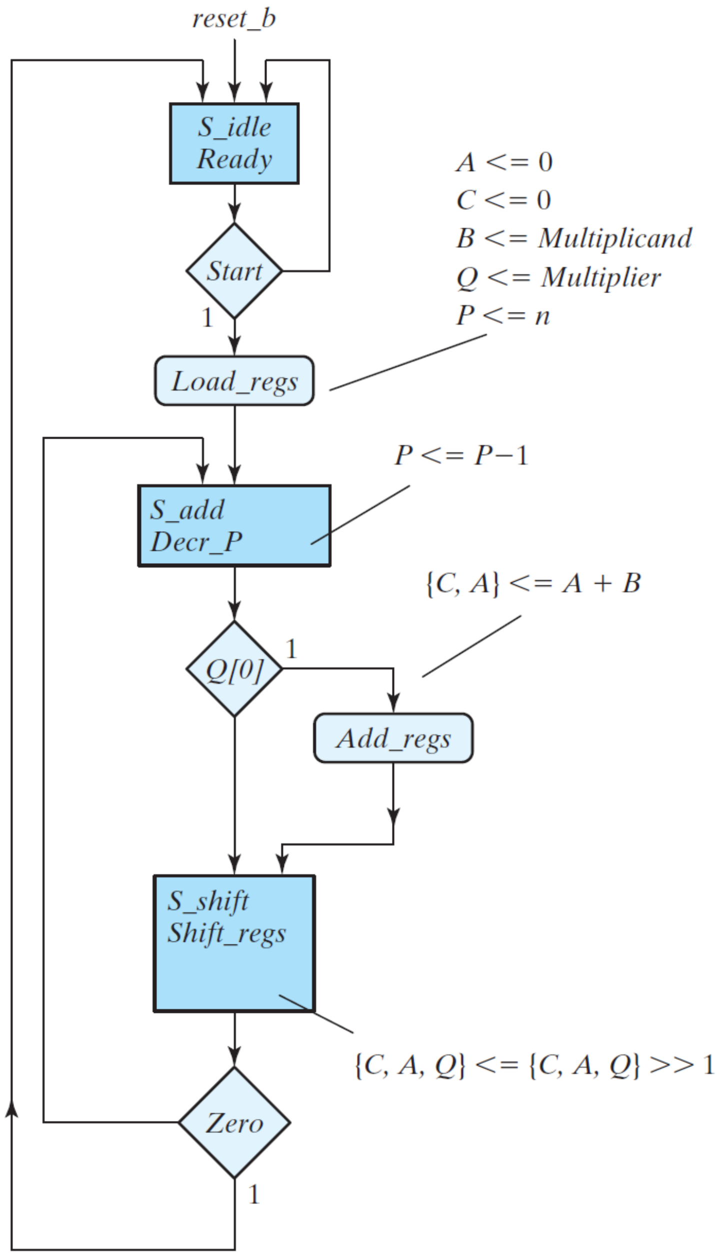
- An *n* -bit sequence register is essentially a circuit with *n* flip-flops, together with the associated gates that effect their state transitions.

- The ASMD chart and the state diagram for the controller of the binary multiplier have three states and two inputs. (There is no need to consider *Q[0]* .)

- To implement the design with a sequence register and decoder, we need two flip-flops for the register and a two-to-four-line decoder.

- The outputs of the decoder will form the Moore-type outputs of the controller directly.

- The Mealy-type outputs will be formed from the Moore outputs and the inputs.



(b)

- The state table for the finite state machine of the controller is shown below

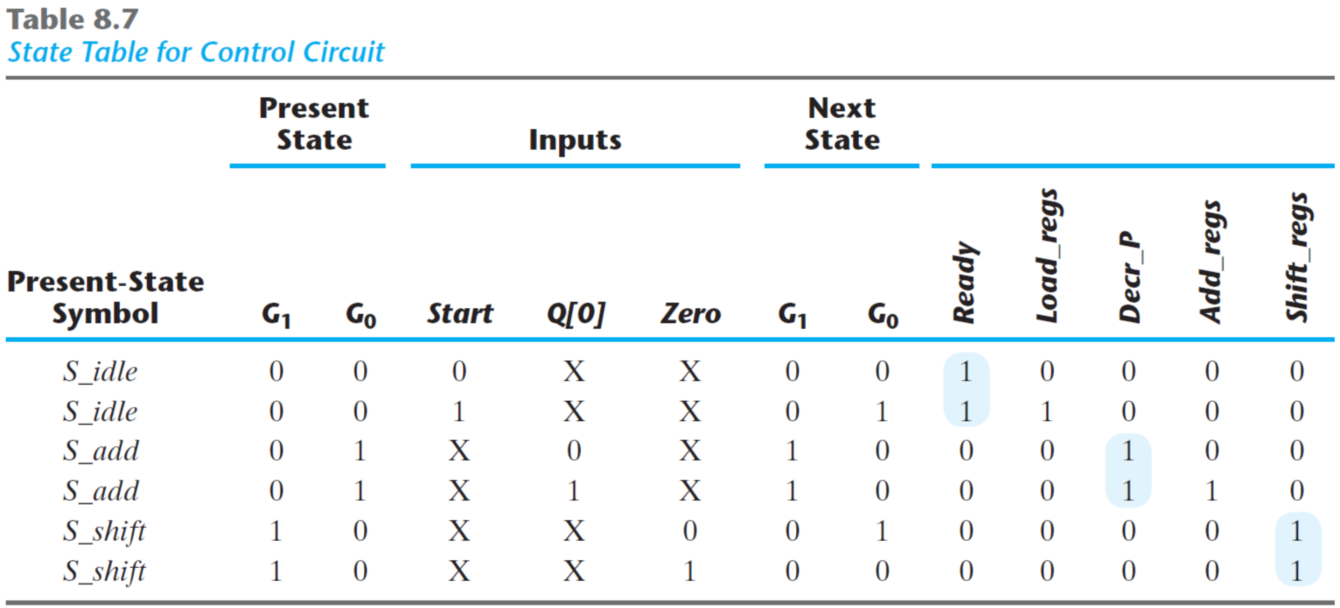


Table 8.7
State Table for Control Circuit

	Present State		Inputs			Next State						
Present-State Symbol	G_1	G_0	*Start*	*Q[0]*	*Zero*	G_1	G_0	*Ready*	*Load_regs*	*Decr_P*	*Add_regs*	*Shift_regs*
S_idle	0	0	0	X	X	0	0	1	0	0	0	0
S_idle	0	0	1	X	X	0	1	1	1	0	0	0
S_add	0	1	X	0	X	1	0	0	0	1	0	0
S_add	0	1	X	1	X	1	0	0	0	1	1	0
S_shift	1	0	X	X	0	0	1	0	0	0	0	1
S_shift	1	0	X	X	1	0	0	0	0	0	0	1

- The particular Moore-type output variable that is equal to 1 at any given time is determined from the equivalent binary value of the present state.

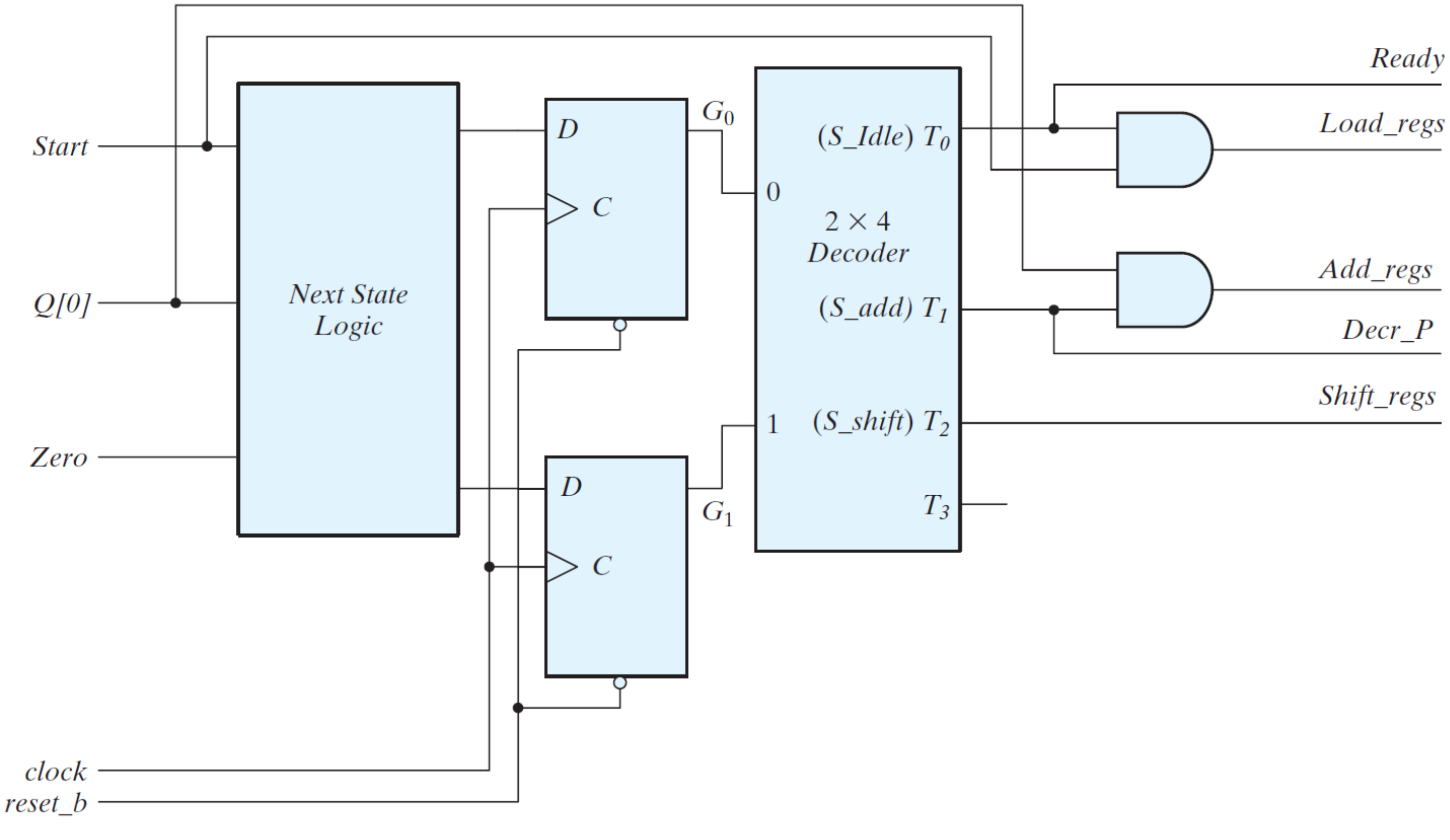
- Those output variables are shaded in the Table.

- Thus, when the present state is $G_1G_0 = 00$, output *Ready* must be equal to 1, while the other outputs remain at 0.

Since the Moore-type outputs are a function of only the present state, they can be generated with a decoder circuit having the two inputs G_1 and G_0 and using three of the decoder outputs T_0 through T_2, as shown in Fig. 8.17 (a), which does not include the wiring for the state feedback.

Table 8.7
State Table for Control Circuit

	Present State		Inputs			Next State						
Present-State Symbol	G_1	G_0	*Start*	*Q[0]*	*Zero*	G_1	G_0	*Ready*	*Load_regs*	*Decr_P*	*Add_regs*	*Shift_regs*
S_idle	0	0	0	X	X	0	0	1	0	0	0	0
S_idle	0	0	1	X	X	0	1	1	1	0	0	0
S_add	0	1	X	0	X	1	0	0	0	1	0	0
S_add	0	1	X	1	X	1	0	0	0	1	1	0
S_shift	1	0	X	X	0	0	1	0	0	0	0	1
S_shift	1	0	X	X	1	0	0	0	0	0	0	1



(a)

FIGURE 8.17
Logic diagram of control for binary multiplier using a sequence register and decoder

- The state machine of the controller can be designed from the state table by means of the classical procedure presented in Chapter 5 .

- This example has a small number of states and inputs, so we could use maps to simplify the Boolean functions. In most control logic applications, the number of states and inputs is much larger.

- In general, the application of the classical method requires an excessive amount of work to obtain the simplified input equations for the flip-flops and is prone to error.

- The design can be simplified if we take into consideration the fact that the decoder outputs are available for use in the design.

- Instead of using flip-flop outputs as the present-state conditions, *we use the outputs of the decoder to indicate the present-state condition of the sequential circuit* .

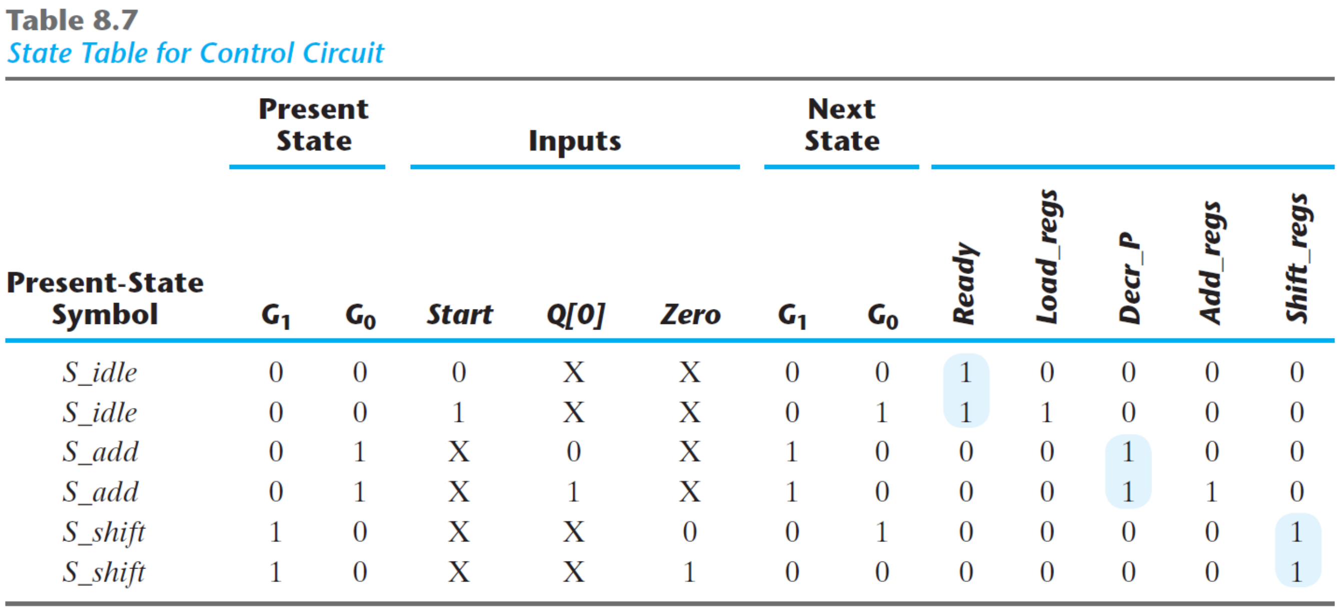


Table 8.7
State Table for Control Circuit

	Present State		Inputs			Next State						
Present-State Symbol	G_1	G_0	*Start*	*Q[0]*	*Zero*	G_1	G_0	*Ready*	*Load_regs*	*Decr_P*	*Add_regs*	*Shift_regs*
S_idle	0	0	0	X	X	0	0	1	0	0	0	0
S_idle	0	0	1	X	X	0	1	1	1	0	0	0
S_add	0	1	X	0	X	1	0	0	0	1	0	0
S_add	0	1	X	1	X	1	0	0	0	1	1	0
S_shift	1	0	X	X	0	0	1	0	0	0	0	1
S_shift	1	0	X	X	1	0	0	0	0	0	0	1

- Moreover, instead of using maps to simplify the flip-flop equations, we can obtain them directly by inspection of the state table.

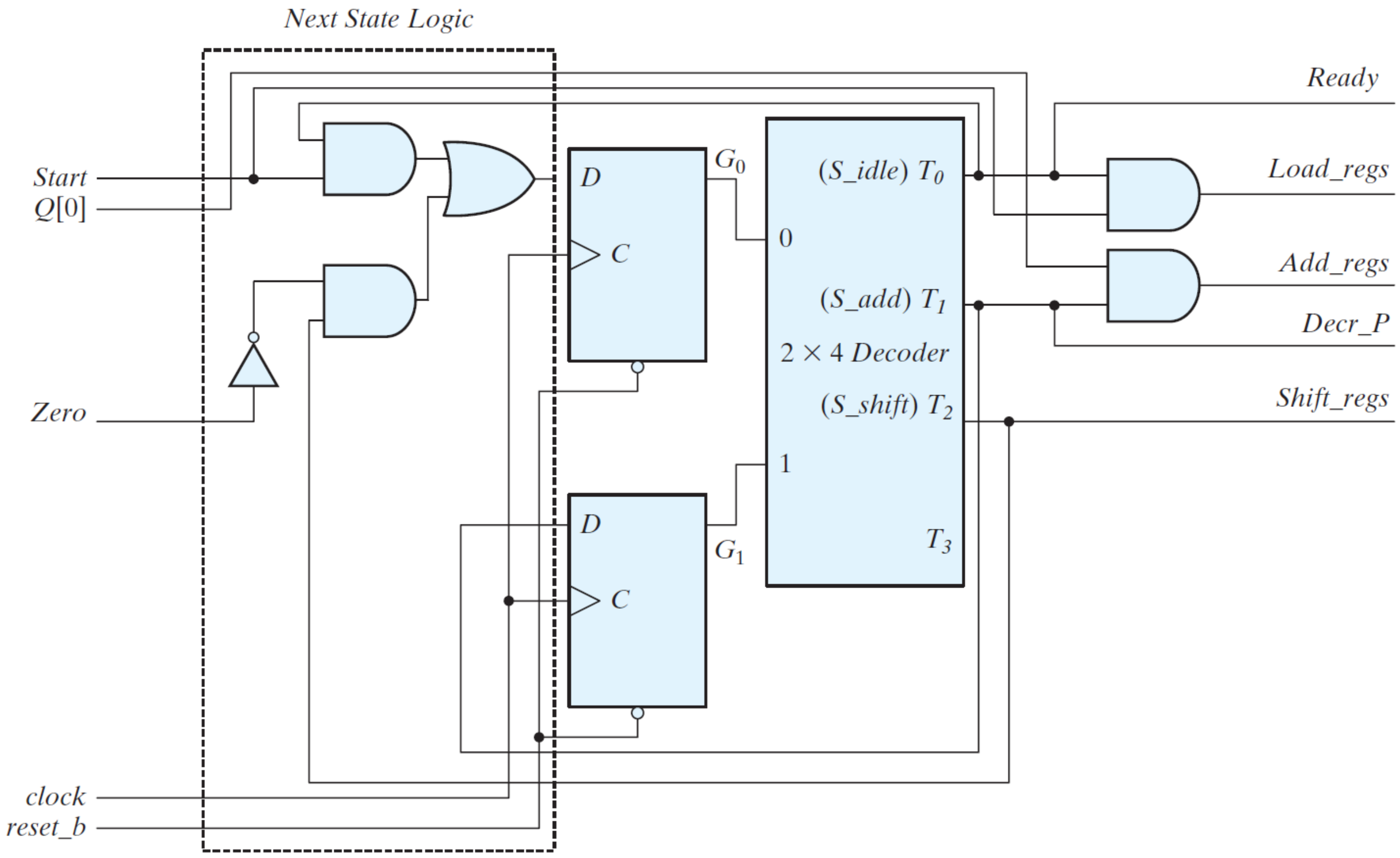
- For example, from the next-state conditions in the state table, we find that the next state of G_1 is equal to 1 when the present state is *S_add* and is equal to 0 when the present state is *S_idle* or *S_shift* . These conditions can be specified by the equation

$$D_{G1} = T_1$$

where D_{G1} is the D input of flip-flop G_1.

Similarly, the D input of G_0 is

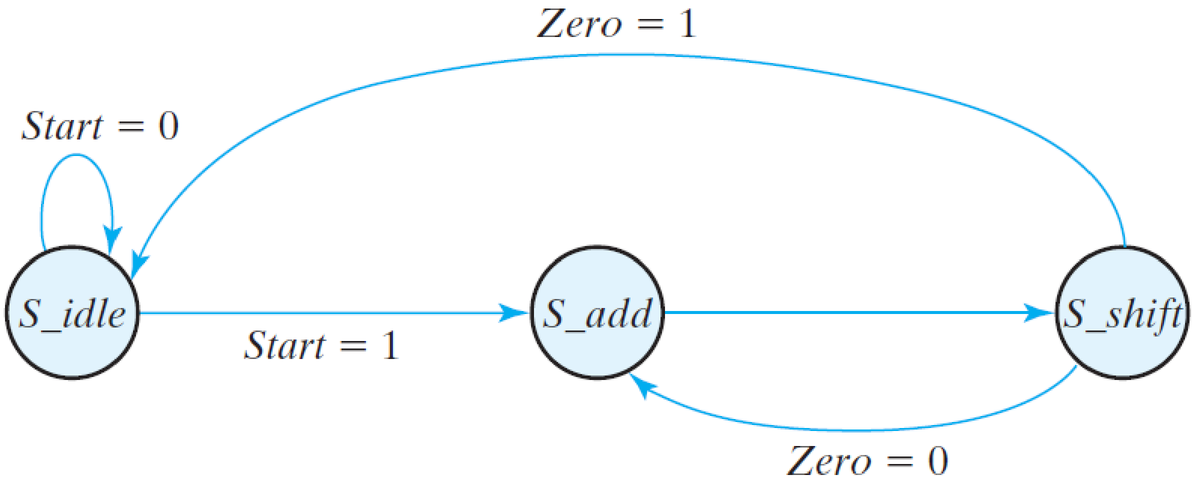
$$DG_0 = T_0\ Start + T_2 Zero'$$



(b)

FIGURE 8.17
Logic diagram of control for binary multiplier using a sequence register and decoder

One-Hot Design (One Flip-Flop per State)



(a)

- The design procedure for a one-hot state assignment will be demonstrated by obtaining the control circuit specified by the state diagram of Fig. 8.16 (a).
- Since there are three states in the state diagram, we choose three *D* flip-flops and label their outputs G_0, G_1, and G_2, *corresponding to S_idle*, *S_add*, and *S_shift*, respectively.
- The input equations for setting each flip-flop to 1 are determined from the present state and the input conditions along the corresponding directed lines going into the state.

- For example, D_{G0}, the input to flipflop G_0, is set to 1 if the machine is in state G_0 and *Start* is not asserted, or if the machine is in state G_2 and *Zero* is asserted.
- These conditions are specified by the input equation:

$$D_{G0} = G_0 Start' + G_2 Zero$$
$$D_{G1} = G_0 Start + G_2 Zero'$$
$$D_{G2} = G_1$$

One-Hot Design (One Flip-Flop per State)

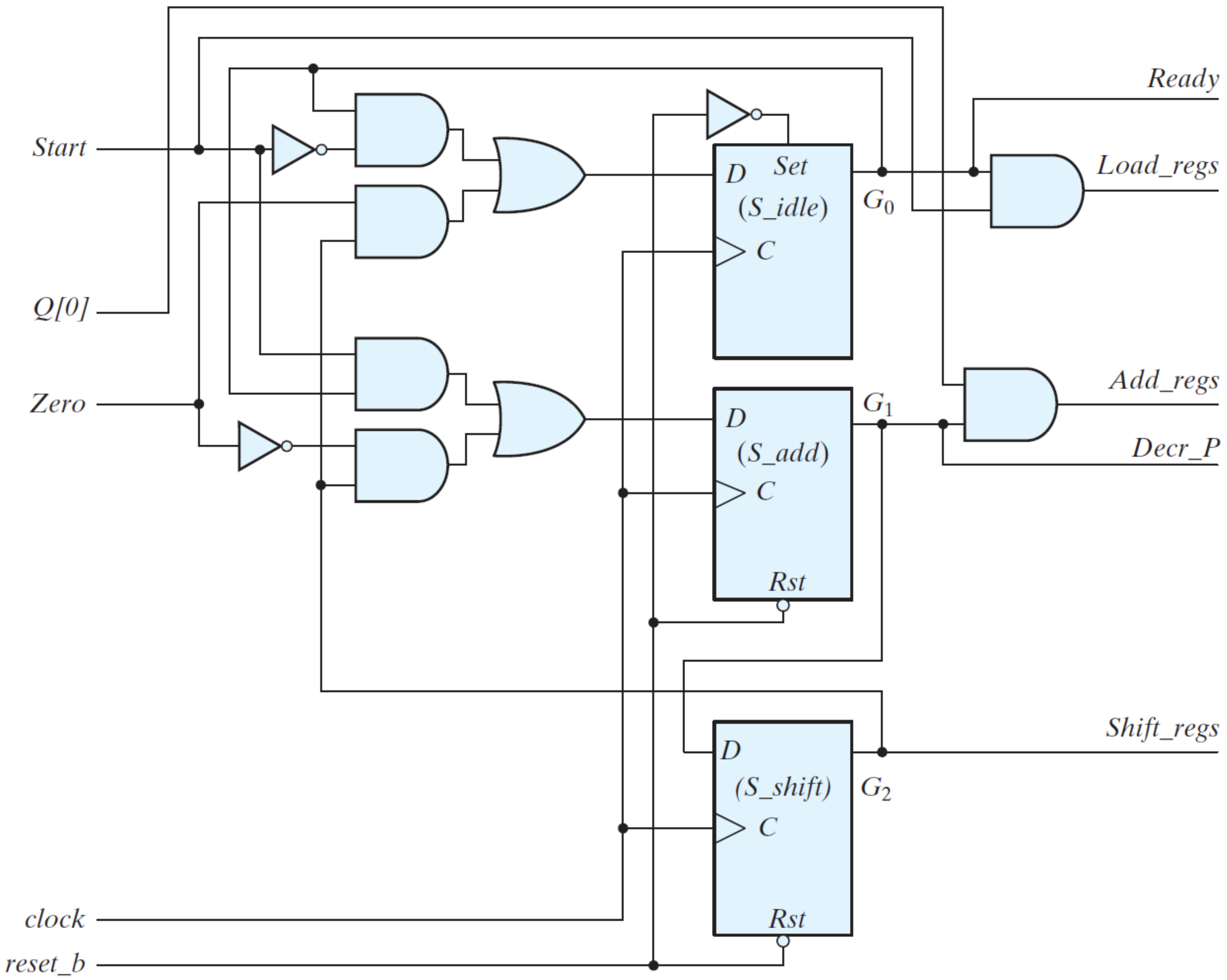


FIGURE 8.18
Logic diagram for one-hot state controller

- The circuit consists of three *D* flip-flops labeled G_0 through G_2, together with the associated gates specified by the input equations.
- Initially, flip-flop G_0 must be set to 1 and all other flip-flops must be reset to 0, so that the flip-flop representing the initial state is enabled.
- This can be done by using an synchronous preset on flip-flop G_0 and an asynchronous clear for the other flip-flops.
- Once started, the controller with one flip-flop per state will propagate from one state to the other in the proper manner.
- Only one flip-flop will be set to 1 with each clock edge; all others are reset to 0, because their *D* inputs are equal to 0.

8.9 HDL DESCRIPTION OF BINARY MULTIPLIER

Self Study Assignment

8.10 DESIGN WITH MULTIPLEXERS

8.10 DESIGN WITH MULTIPLEXERS

- The register-and-decoder scheme for the design of a controller has three parts:
  - ✓ the flip-flops that hold the binary state value,
  - ✓ the decoder that generates the control outputs, and
  - ✓ the gates that determine the next-state and output signals.

- In Section 4.11, it was shown that a combinational circuit can be implemented with multiplexers instead of individual gates.

- Replacing the gates with multiplexers 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 a decoder that asserts a unique output line for each control state.

  - ➢ *These three components are predefined standard cells in many integrated circuits.*

- Consider, for example, the ASM chart of Fig. 8.20 , consisting of four states and four control inputs.

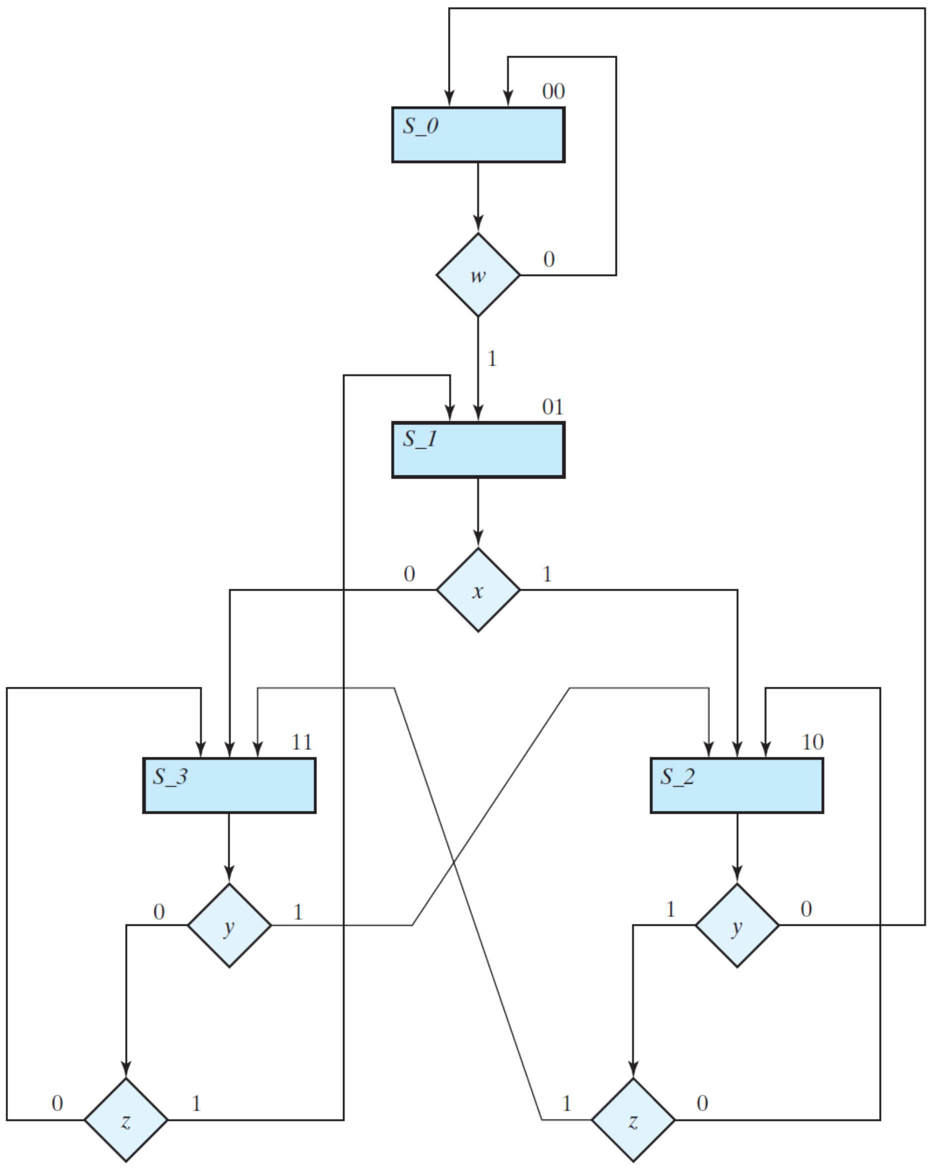


FIGURE 8.20
Example of ASM chart with four control inputs

- We are interested in only the control signals governing the state sequence.

  - State boxes left empty as it is only to design Control sequence
  - Binary assignments specified at upper right corner
  - Decision boxes specify state transition as a function Of four control inputs w, x, y, z
  - Three level control implementation includes

    MUX, registers and decoders

Multiplexer Input Conditions

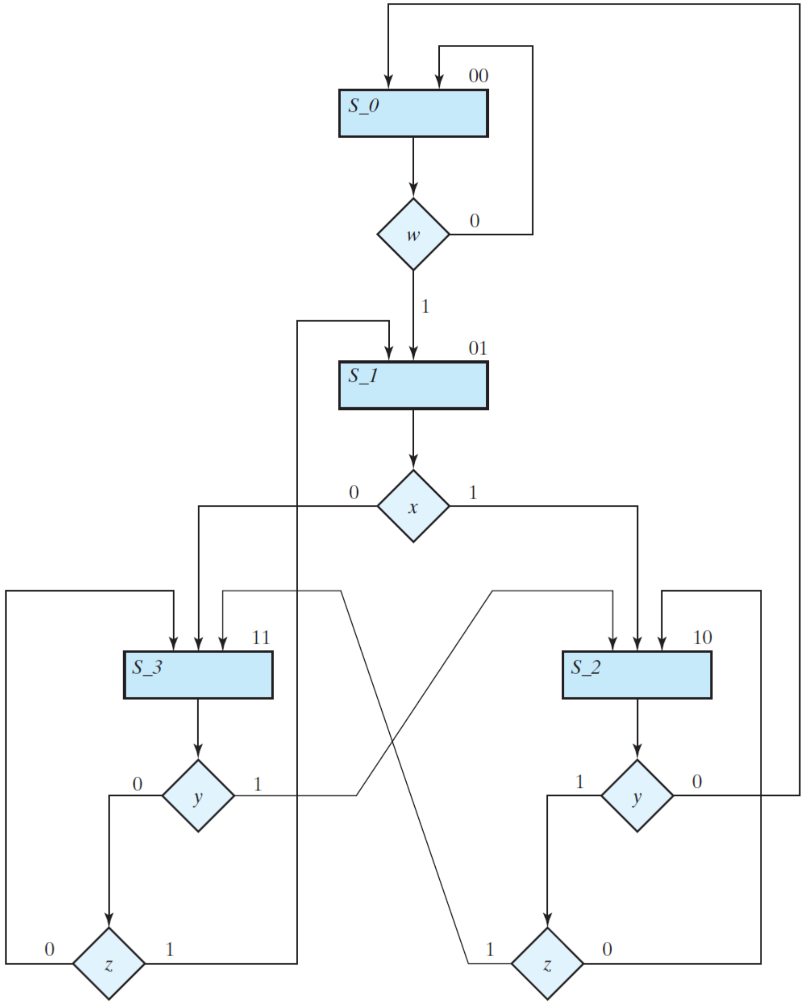


FIGURE 8.20
Example of ASM chart with four control inputs

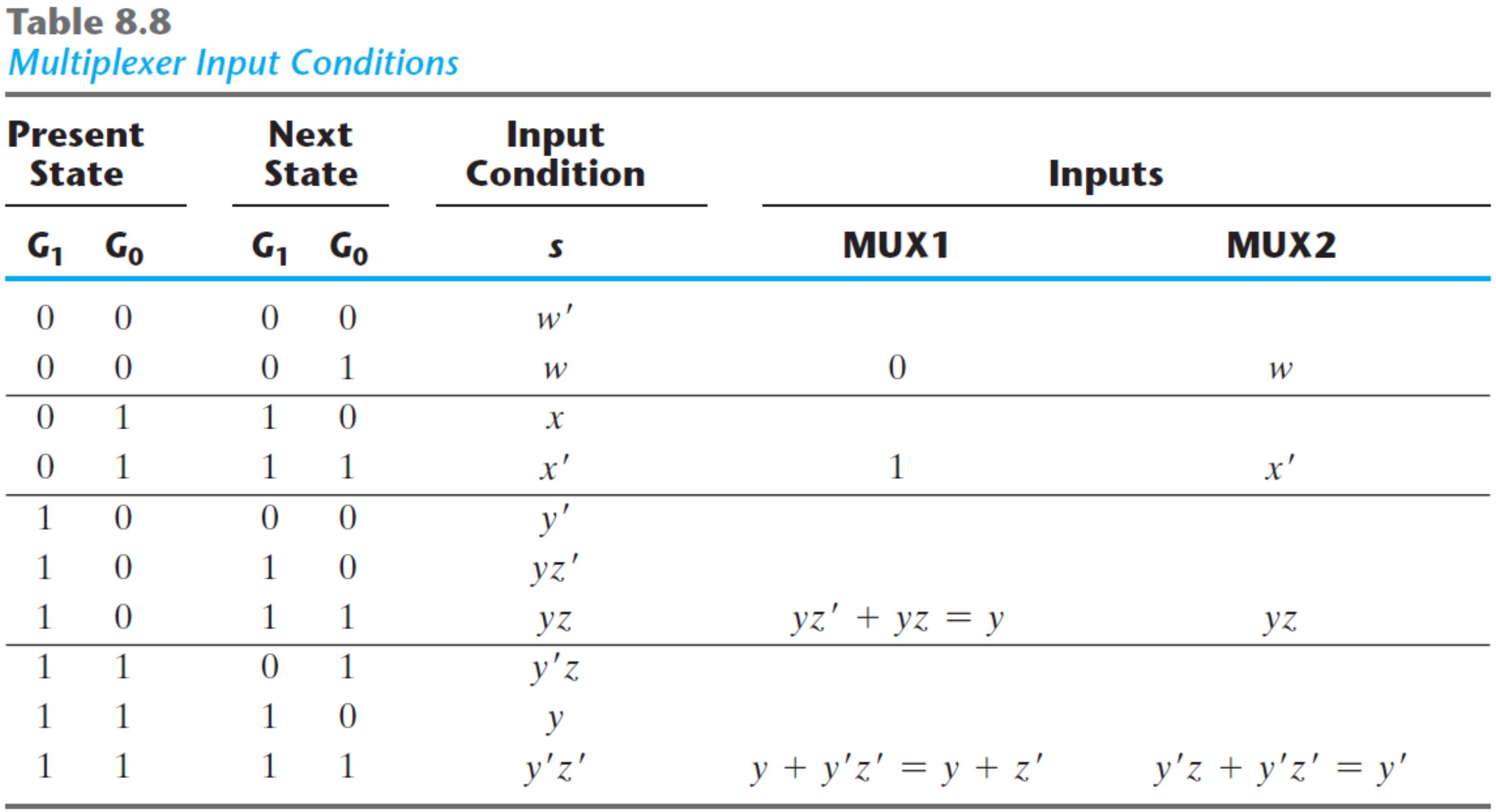


Table 8.8
Multiplexer Input Conditions

Present State		Next State		Input Condition	Inputs	
G_1	G_0	G_1	G_0	s	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	$y'z'$	$y + y'z' = y + z'$	$y'z + y'z' = y'$

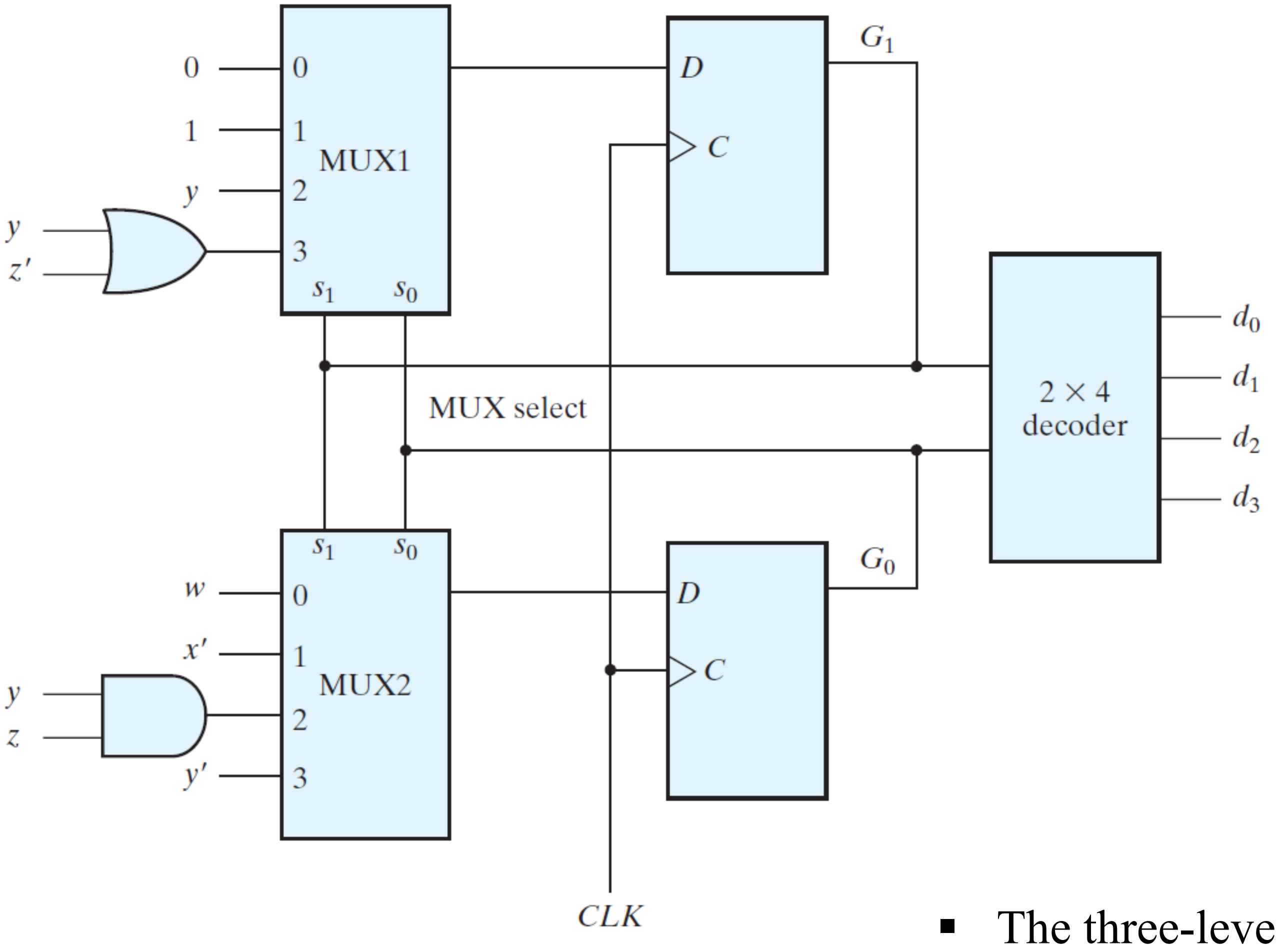


FIGURE 8.21
Control implementation with multiplexers

Table 8.8
Multiplexer Input Conditions

Present State		Next State		Input Condition	Inputs	
G_1	G_0	G_1	G_0	s	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	$y'z'$	$y + y'z' = y + z'$	$y'z + y'z' = y'$

- The three-level control implementation consists of two multiplexers, MUX1 and MUX2;
- a register with two flip-flops, G_1 and G_0; and
- a decoder with four outputs— d_0, d_1, d_2, and d_3, corresponding to *S_0*, *S_1*, *S_2*, and *S_3*, respectively.

Design Example: Count the Number of Ones in a Register

To demonstrate the multiplexer implementation of the logic for a control unit, along with formulation of the ASMD chart and the implementation of the datapath subsystem.

Design:

Two registers R1 and R2 and Flip-flop E

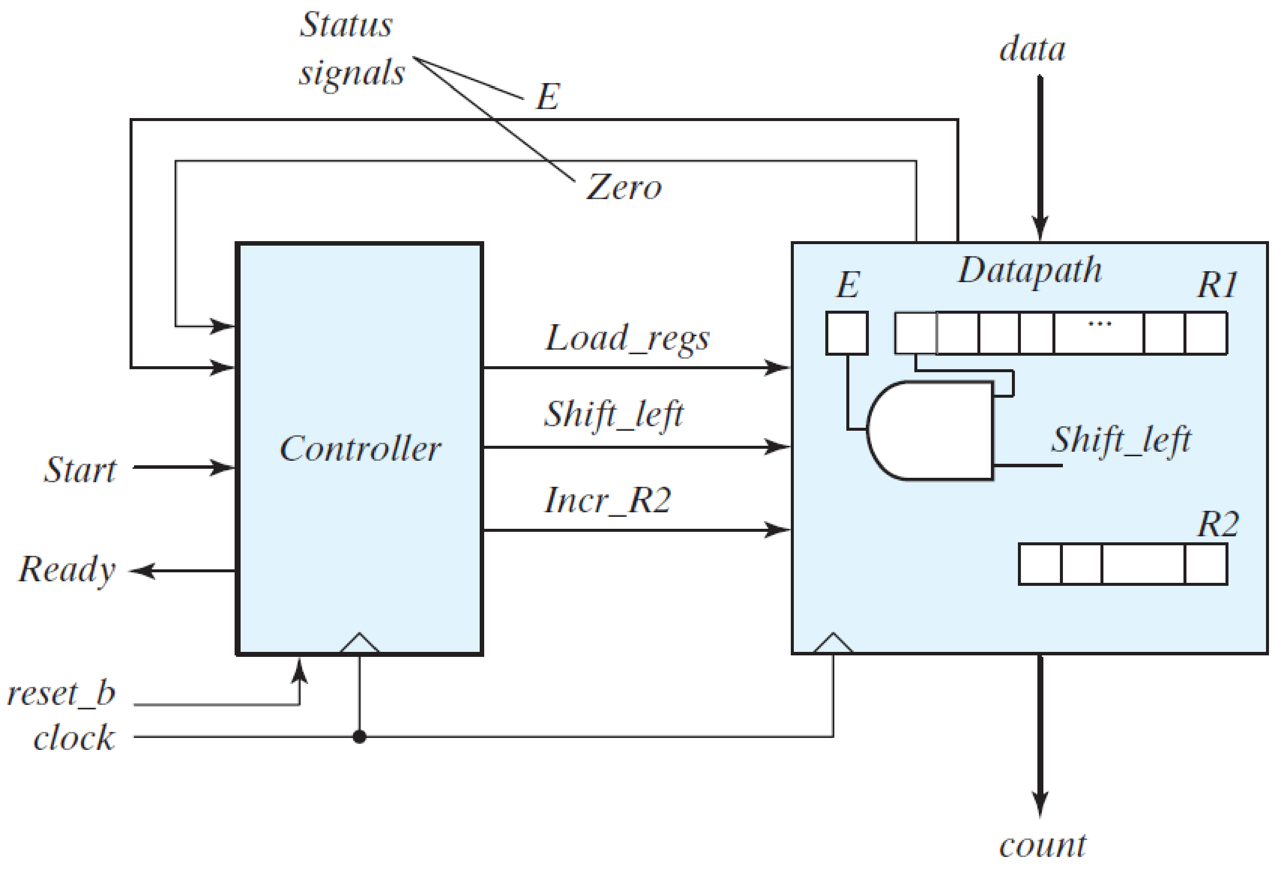
System count number of 1’s in R1 and sets R2 to that number. Done by shifting each bit in R1 one at a time into E, E checked by control and if 1 R2 is incremented.

Control uses external input S to start and uses status input E and Z from datapath. Z = 1 when R1 = 0

Design Example: Count the Number of Ones in a Register

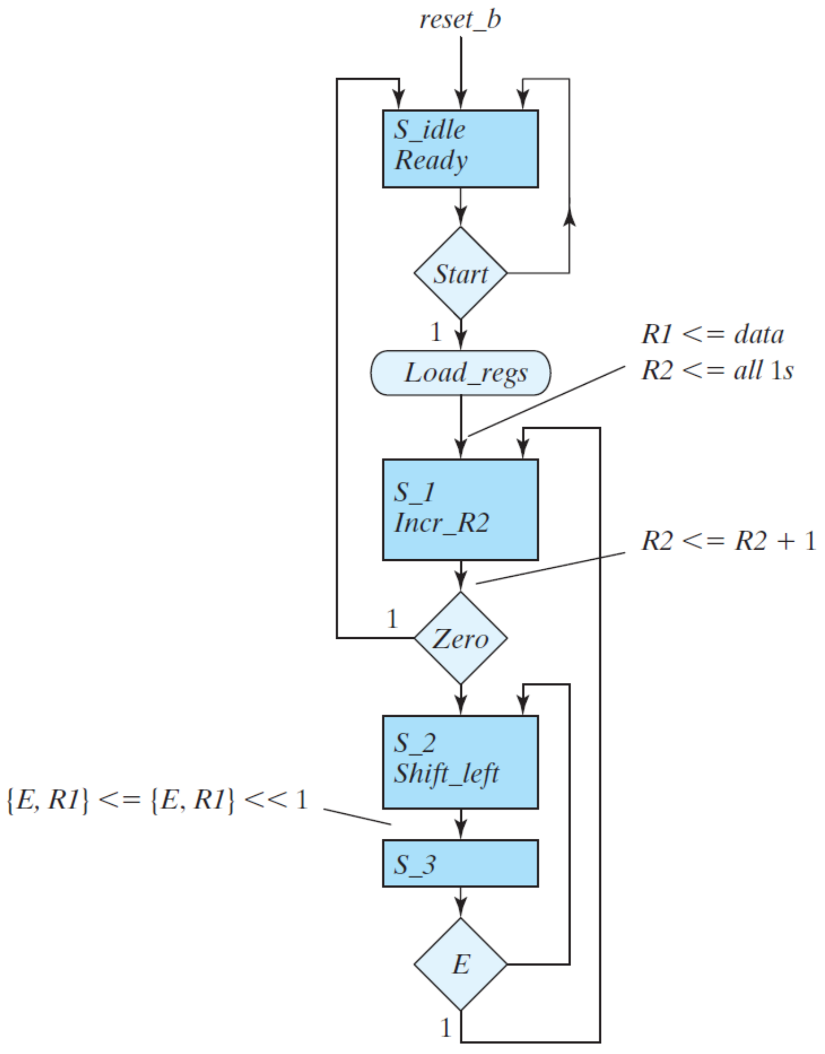
- Binary number loaded into R1, R2 set to all 1's
- R2 made to Zero in the next state
- Content of R1 not zero then Z = 0
- Content of R1 shifted and left most bit into E
- For every 1 detected in E, R2 incremented and R1 checked for more 1's

The block diagram of the datapath and controller are shown in Fig. 8.22 (a).



(a)

Design Example: Count the Number of Ones in a Register



(c)

Table 8.9
Multiplexer Input Conditions for Design Example

Present State		Next State		Input Conditions	Multiplexer Inputs	
G_1	G_0	G_1	G_0		MUX1	MUX2
0	0	0	0	*Start'*		
0	0	0	1	*Start*	0	*Start*
0	1	0	0	*Zero*		
0	1	1	0	*Zero'*	*Zero'*	0
1	0	1	1	*None*	1	1
1	1	1	0	*E'*		
1	1	0	1	*E*	*E'*	*E*

Design Example: Count the Number of Ones in a Register

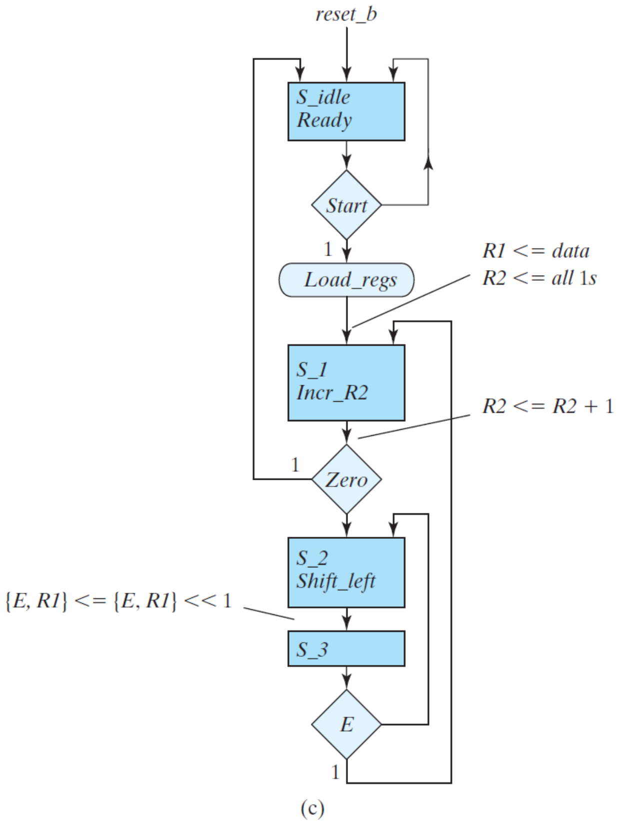


Table 8.9
Multiplexer Input Conditions for Design Example

Present State		Next State		Input Conditions	Multiplexer Inputs	
G_1	G_0	G_1	G_0		MUX1	MUX2
0	0	0	0	*Start'*		
0	0	0	1	*Start*	0	*Start*
0	1	0	0	*Zero*		
0	1	1	0	*Zero'*	*Zero'*	0
1	0	1	1	*None*	1	1
1	1	1	0	*E'*		
1	1	0	1	*E*	*E'*	*E*

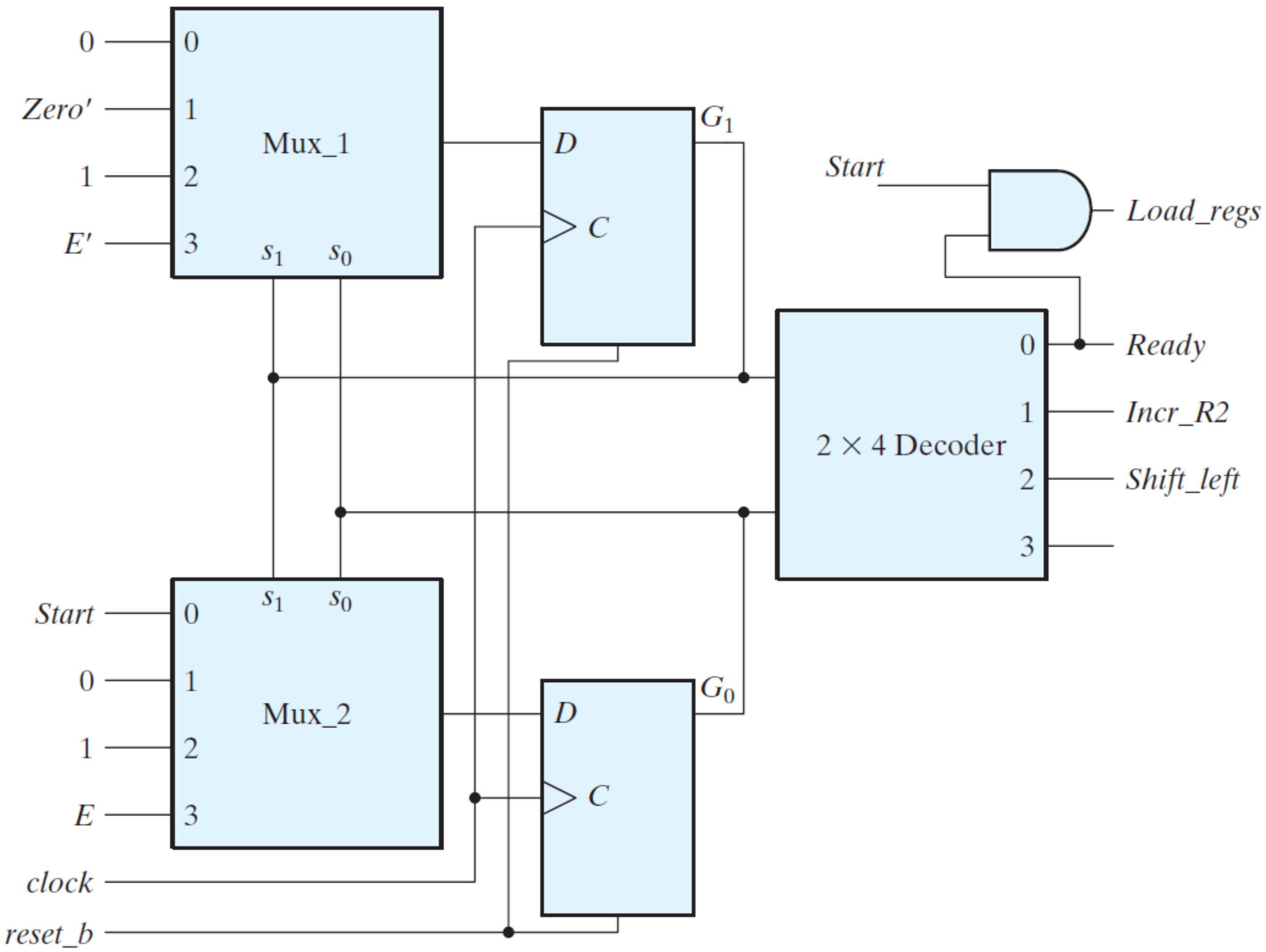


FIGURE 8.23
Control implementation for count-of-ones circuit

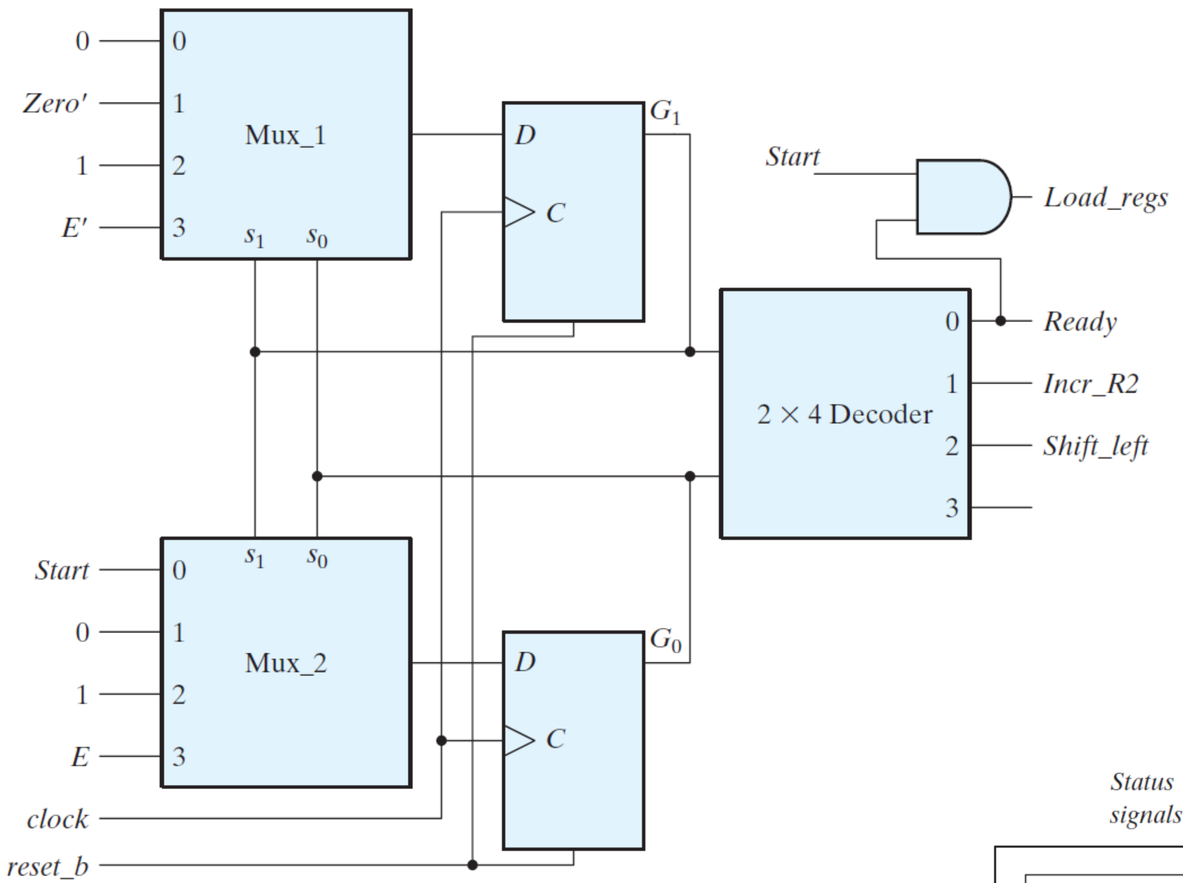
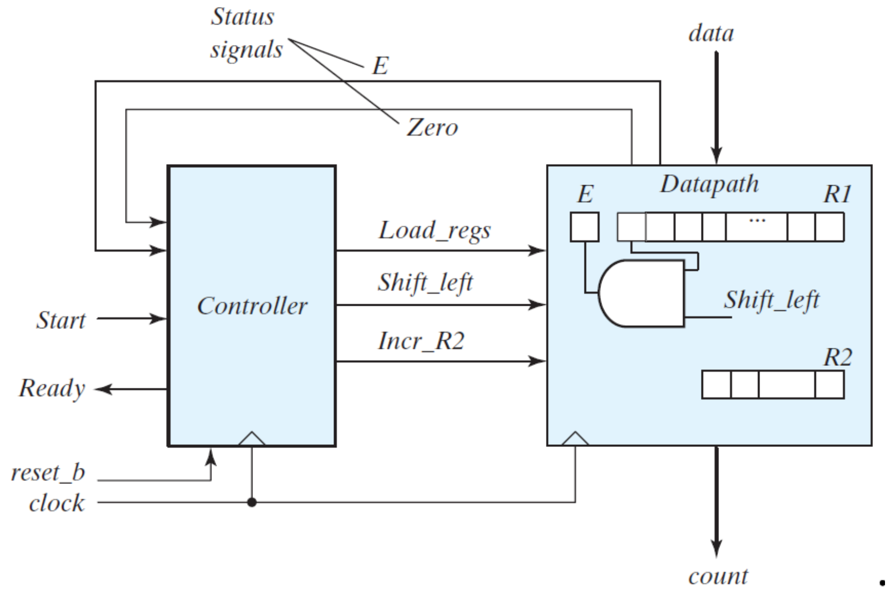
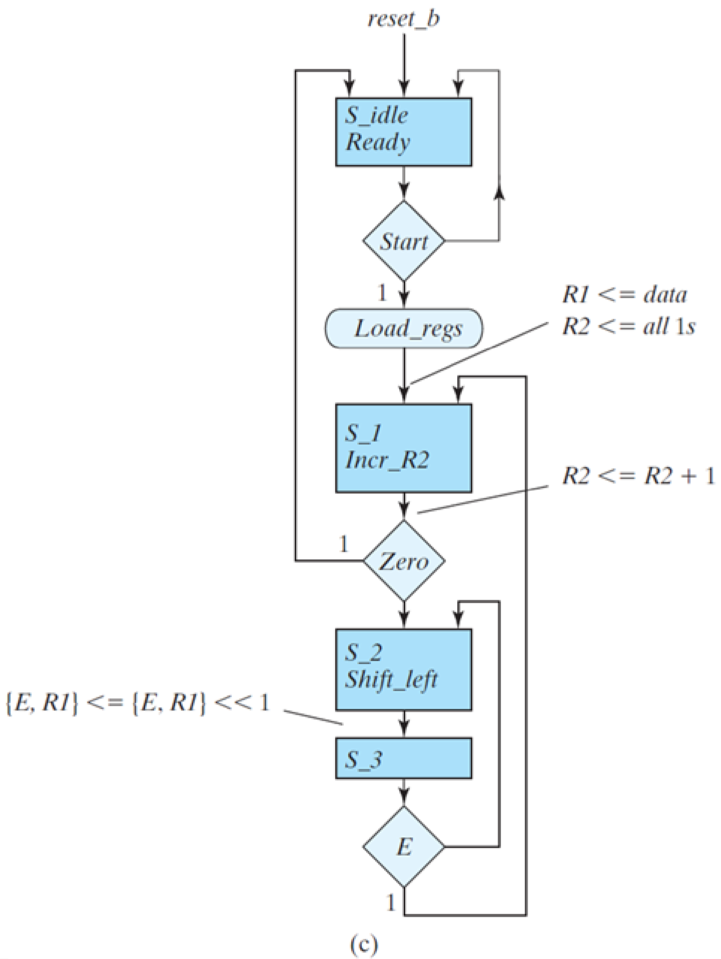


FIGURE 8.23
Control implementation for count-of-ones circuit





Draw the ASM chart leaving state boxes empty

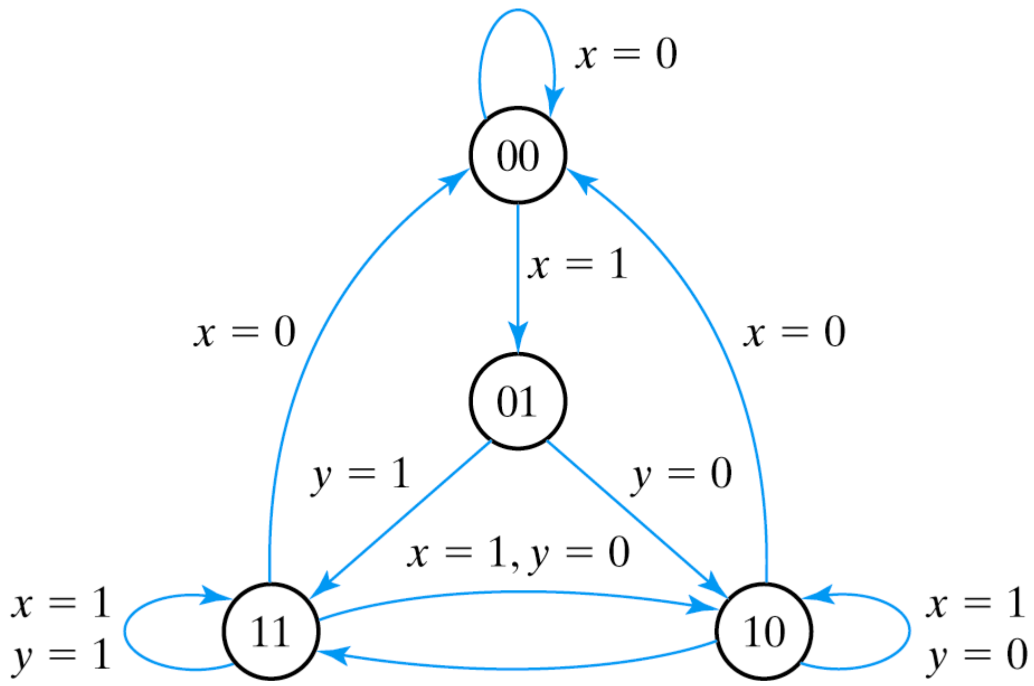
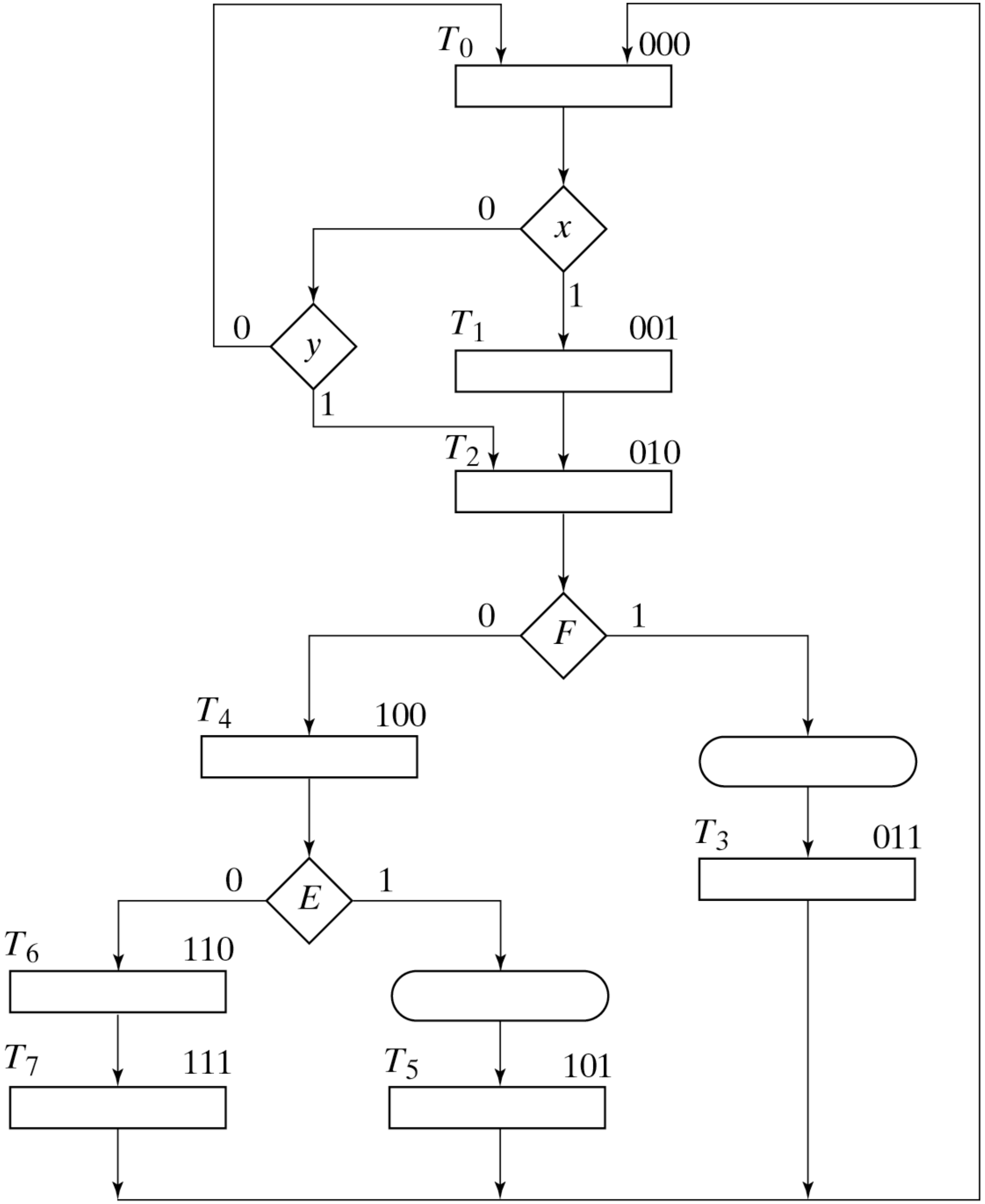


Fig. P8-10 Control State Diagram for Problems 8-10 and 8-11

Draw the State box



M. Morris Mano
DIGITAL DESIGN, 3e.

Thank You