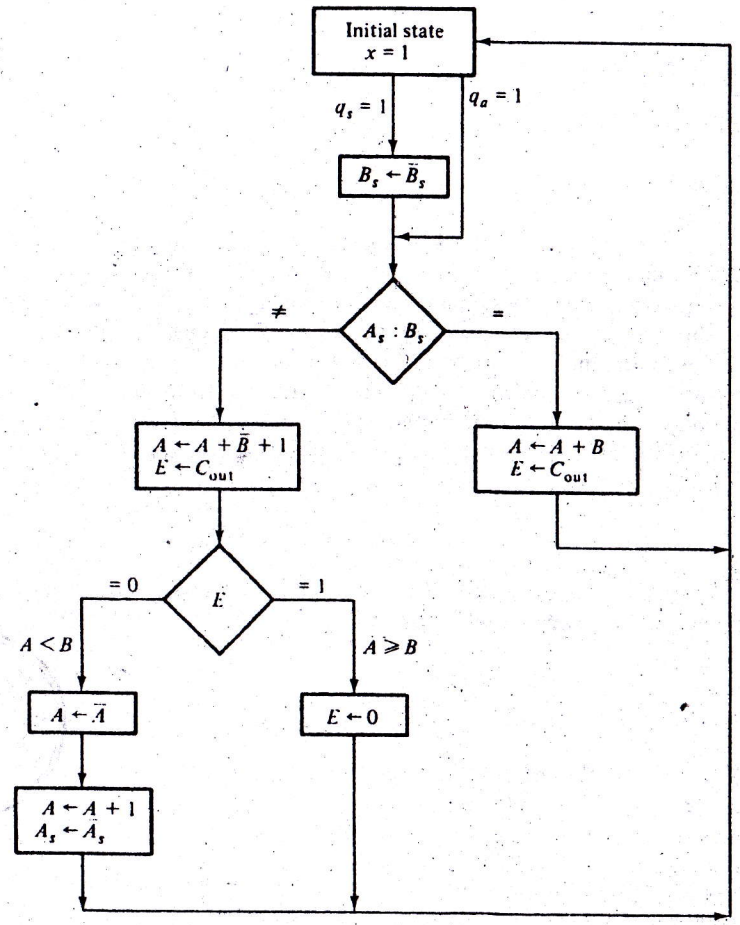
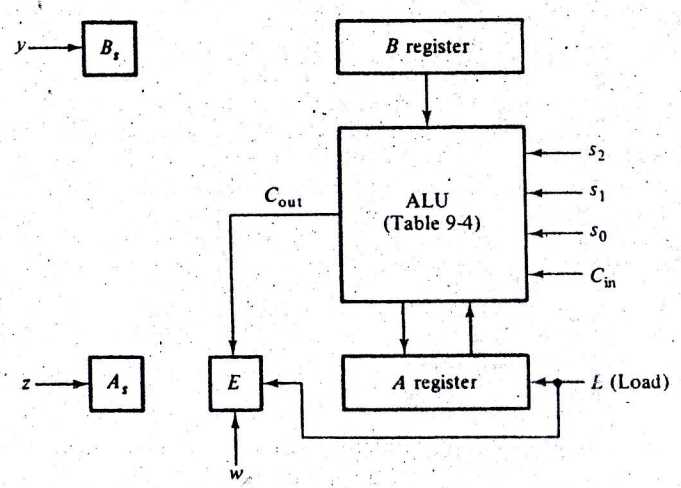


Figure 10-7 Flowchart for sign-magnitude addition and subtraction

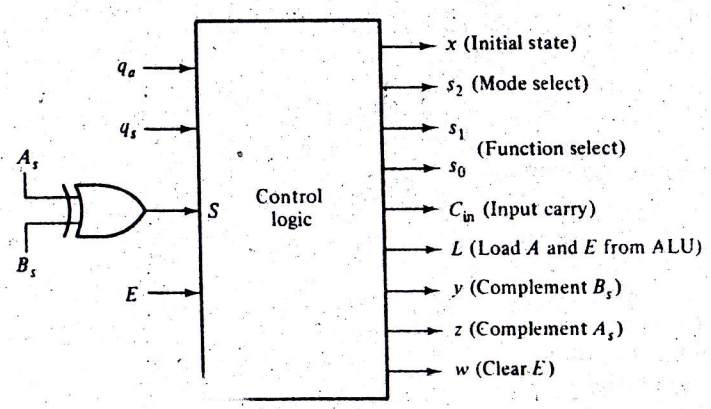
of B is complemented. Input q_a initiates an add operation, and the sign of B is left unchanged. The next step is to compare the two signs. The decision block marked with $A_s : B_s$ symbolizes this decision. If the signs are equal, we take the path marked by the symbol = ; otherwise, we take the path marked by the symbol ≠. For equal signs, the content of A is added to the content of B and the sum is transferred to A. The value of the end carry in this case is an overflow; so the E flip-flop is made equal to the output carry C_{out}. The circuit then goes to its initial state and output x becomes 1. The sign of the result in this case is the same as the original sign of A_s; so the sign bit is left unchanged.

The two magnitudes are subtracted if the signs are not the same. The subtraction of the magnitudes is done by adding A to the 2's complement of B. No overflow can occur if the numbers are subtracted; so E is cleared to 0. A 1 in E

indicates that $A \geqslant B$ and the number in A is the correct result. The sign of the result again is equal to the original value of A_s. A 0 in E indicates that $A < B$. For this case, it is necessary to form the 2's complement of the value in A and complement the sign in A_s. The 2's complement of A can be done with one microoperation, $A \leftarrow \bar{A} + 1$. However, we want to use the ALU of Chapter 9 and this ALU does not have the 2's complement operation. For this reason, the 2's complement is obtained from the complement and increment operations which are available in the ALU.



(a) Data processor registers and ALU



(b) Control block diagram

Figure 10-8 System block diagram

Data Processor Specification

The flowchart algorithm lists all the microoperations for the data-processor part of the system. The operations between A and B can be done with the ALU. The operations with A_s, B_s, and E must be initiated with separate control variables. Figure 10-8(a) shows the data-processor with the required control variables. As mentioned before, the ALU is from Chapter 9 and its function is specified in Table 9-4. This ALU has four selection variables, as shown in the diagram. The variable L loads the output of the ALU into register A and also the output carry into E. Variables y, z, and w complement B_s and A_s and clear E, respectively.

The block diagram of the control logic is shown in Fig. 10-8(b). The control receives five inputs: two from the external environment and three from the data-processor. To simplify the design, we define a new variable S:

$$S = A_s \oplus B_s$$

This variable gives the result of the comparison between the two sign bits. The exclusive-OR operation is equal to 1 if the two signs are not the same, and it is equal to 0 if the signs are both positive or both negative.

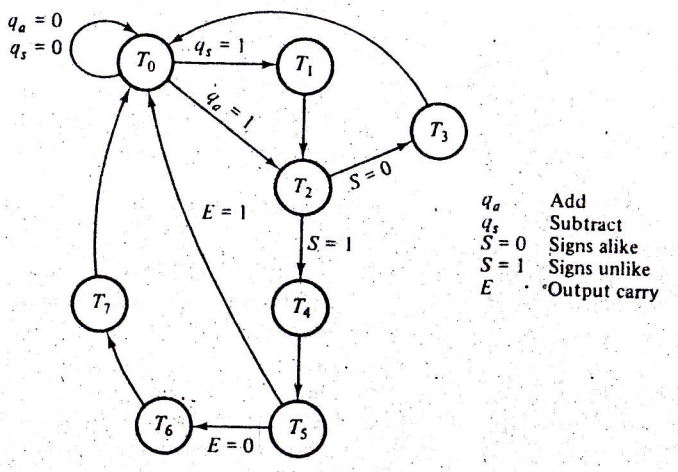
The control provides an output x for the external circuit. It also selects the operations in the ALU through the four selection variables s_2, s_1, s_0, and C_{in}. The other four outputs go to registers in the data-processor as specified in the diagram. Although not shown in the diagram, the outputs of the control logic should be connected to the corresponding inputs in the data-processor. Now that the data-processor is specified, we can design the control logic for the system.

Control State Diagram

The design of a hard-wired control is a sequential-logic problem. As such, it may be convenient to formulate the state diagram of the sequential control. The function boxes in a flowchart may be considered as states of the sequential circuit, and the decision boxes as next-state conditions. The microoperations that must be executed at a given state are specified within the function box. The conditions for the next state transition are specified inside the decision box or in the directed lines between two function boxes. Although one can formulate this relationship between a flowchart and a state diagram, the conversion from one form to the other is not unique. Consequently, different designers may produce different state diagrams for the same flowchart, and each may be a correct representation of the system.

We start by assigning an initial state, T_0, to the sequential controller. We then determine the transition to other states T_1, T_2, T_3, and so on. For each state, we determine the microoperations that must be initiated by the control circuit. This procedure produces the state diagram for the controller, together with a list of register-transfer operations which are to be initiated while the control circuit is in each and every state.

The control state diagram and the corresponding register-transfer operations are derived in Fig. 10-9. The information for this design is taken directly from the



(a) State diagram

	Control outputs								
	x	s_2	s_1	s_0	C_{in}	L	y	z	w
T_0: Initial state $x = 1$	1	0	0	0	0	0	0	0	0
T_1: $B_s \leftarrow \overline{B}_s$	0	0	0	0	0	0	1	0	0
T_2: nothing	0	0	0	0	0	0	0	0	0
T_3: $A \leftarrow A + B$, $E \leftarrow C_{out}$	0	0	0	1	0	1	0	0	0
T_4: $A \leftarrow A + \overline{B} + 1$, $E \leftarrow C_{out}$	0	0	1	0	1	1	0	0	0
T_5: $E \leftarrow 0$	0	0	0	0	0	0	0	0	1
T_6: $A \leftarrow \overline{A}$	0	1	1	1	0	1	0	0	0
T_7: $A \leftarrow A + 1$, $A_s \leftarrow \overline{A}_s$	0	0	0	0	1	1	0	1	0

(b) Sequence of register transfers

Figure 10-9 Control state diagram and sequence of microoperations

flowchart of Fig. 10-7 and the variables defined in the block diagram of Fig. 10-6. The initial control state is T_0. While the control is in this state, variable x is made equal to 1. This variable is 0 in all other states. As long as q_a and q_s are 0, the control stays in its initial state. If q_s becomes 1, the control performs a subtraction operation by going to state T_1. In this state, sign bit B_s is complemented. Control then goes to state T_2 to add the two numbers. If q_a becomes 1, control goes directly to state T_2.

The next state after T_2 depends on the relative values of the sign bits which are determined from input variable S. If the signs are alike, S is 0 and control goes to state T_3. In this state, the two magnitudes are added and the overflow bit set. Once this is done, control goes back to the initial state. If the signs are unlike, S is 1 and control goes from state T_2 to state T_4. In this state, the two magnitudes are subtracted by taking the 2's complement of B. The end carry is transferred to E during the subtraction, and control then goes to state T_5.

It must be realized that the end carry from the ALU is transferred to E with a clock pulse. This happens with the same clock pulse that causes the control to go from state T_4 to T_5. Although we show the microoperation:

$$E \leftarrow C_{out}$$

with timing variable T_4 this operation is not executed until a clock pulse occurs. Once this clock pulse executes the operation, control finds itself in state T_5. Therefore, the value of E for an end carry should not be checked until control reaches state T_5. The value of E is checked to determine the relative magnitudes of A and B. If $E = 1$, it indicates that $A \geq B$. For this case, E must be cleared and the operation is completed. If $E = 0$, it indicates that $A < B$. Control then goes to states T_6 and T_7 to complement A and A_s. Note that E is cleared while the control is in state T_5. This is done whether E is 1 or 0, since trying to clear a flip-flop that is already 0 leaves the flip-flop in the 0 state anyway. Note also that E is cleared with the clock pulse that causes control to go out of state T_5. It must be realized that clearing E and transferring control to state T_0 or T_6 is done with one common clock pulse without a conflict. The original value of E at time T_5 determines the next state even though this flip-flop is cleared while the clock pulse goes through its edge transition.

It should be apparent from this example that the interpretation of a flowchart may result in a different state diagram for the same control logic. This is acceptable as long as the hardware constraints are taken into consideration and the system functions according to the specifications. For example, instead of checking E at time T_5, we could have chosen to check C_{out} at time T_4. If C_{out} is 1, control goes to state T_5 to clear E. If it is 0, control can go directly to state T_6, bypassing state T_5 in this case.

Design of Hard-wired Control

The control outputs are a function of the control states and are listed in Table 10-9(b). These outputs are defined in the block diagram of Fig. 10-8(b). The values for the ALU selection variables are determined from Table 9-4. The L (load)

variable must be made equal to 1 every time the output of the ALU is transferred to register A. Otherwise, L is 0 and the ALU outputs have no effect on the register. To design the control for this system, we need to design the state diagram of Fig. 10-9(a) and provide the control outputs as specified in Fig. 10-9(b).

The control can be designed using the classical sequential-logic procedure. This procedure requires a state table with eight states, four inputs, and nine outputs. The sequential circuit to be derived from such a state table will not be easy to obtain because of the large number of variables. The circuit obtained by using this method may have a minimum number of gates, but it will have an irregular pattern and will be difficult to analyze if a malfunction occurs. These difficulties are removed if the control is designed by the one flip-flop per state method.

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 by inspection. No state or excitation tables are needed if D flip-flops are employed. Remember that the next state of a D flip-flop is a function of the D input and is independent of the present state. Since the method requires one flip-flop for each state, we choose eight D flip-flops and label their outputs $T_0, T_1, T_2, \ldots, T_7$. The condition for setting a given flip-flop is specified in the state diagram. For example, flip-flop T_2 is set with the next clock pulse if $T_1 = 1$ or if $T_0 = 1$ and $q_a = 1$. This condition can be defined with the Boolean function:

$$DT_2 = q_a T_0 + T_1$$

where DT_2 designates the D input of flip-flop T_2. In fact, the condition for setting a flip-flop to 1 is obtained from the condition specified in the directed lines going into a given 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 flip-flops, we obtain the input functions given in Table 10-1.

Initially, flip-flop T_0 is set and all others are cleared. At any given time, only one D input is in the 1 state while all others are maintained at 0. The next clock pulse sets the flip-flop whose D input is 1 and clears all others. For example, if

TABLE 10-1 Boolean functions for control

Flip-flop input functions	Boolean functions for output control
$DT_0 = q_a' q_s' T_0 + T_3 + ET_5 + T_7$	$x = T_0$
$DT_1 = q_s T_0$	$s_2 = T_6$
$DT_2 = q_a T_0 + T_1$	$s_1 = T_4 + T_6$
$DT_3 = S' T_2$	$s_0 = T_3 + T_6$
$DT_4 = ST_2$	$C_{in} = T_4 + T_7$
$DT_5 = T_4$	$L = T_3 + T_4 + T_6 + T_7$
$DT_6 = E' T_5$	$y = T_1$
$DT_7 = T_6$	$z = T_7$
	$w = T_5$

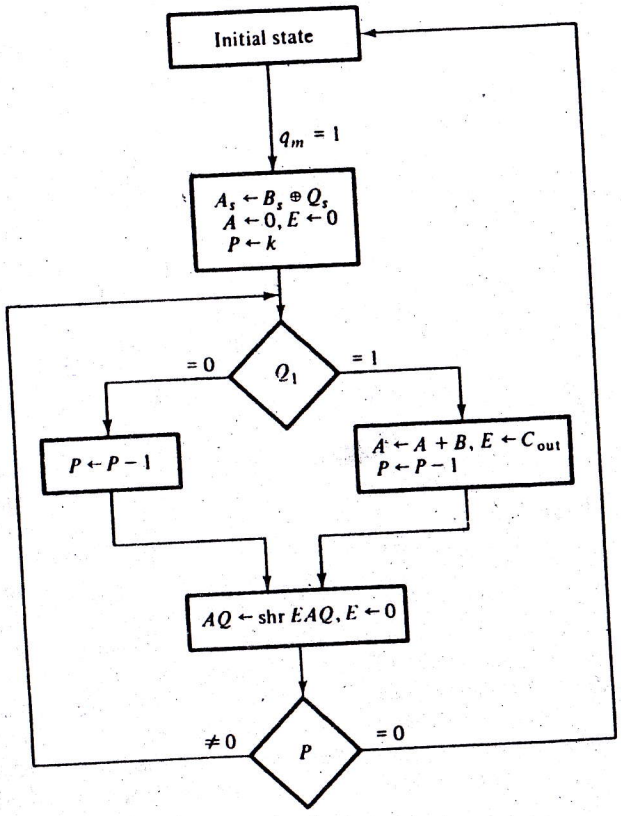


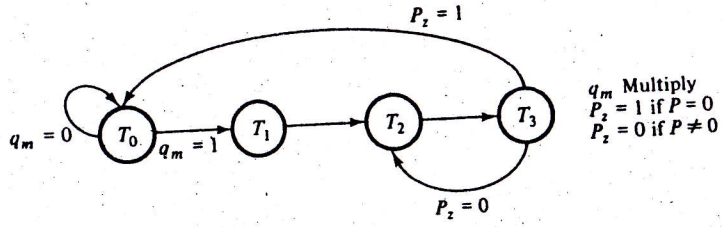
Figure 10-14 Flowchart for binary multiplier

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 holding the least significant bits. The sign of the product is in A_s.

Control Specifications

The design algorithm given in the flowchart can be specified more precisely by a state diagram and a list of register-transfer operations. It was mentioned previously that the conversion from a flowchart to a state diagram is not unique. The flowchart may be considered a preliminary formulation of the algorithm. The control state diagram, together with a list of microoperations, is more precise since it takes into consideration the hardware constraints of the system.

The control sequence of operations is defined in Fig. 10-15. The control has four states, and the register-transfer operations for each state are listed below the



(a) State diagram

T_0: Initial state
T_1: $A_s \leftarrow B_s \oplus Q_s, A \leftarrow 0, E \leftarrow 0, P \leftarrow k$
Q_1T_2: $A \leftarrow A + B, E \leftarrow C_{out}$
T_2: $P \leftarrow P - 1$
T_3: $AQ \leftarrow \text{shr } EAQ, E \leftarrow 0$

(b) Sequence of register transfers

Figure 10-15 Control state diagram and sequence of micro-operations for multiplier

state diagram. Control stays in an initial state T_0 until q_m becomes 1. It then goes to state T_1 to initialize registers A, E, and P and to form the sign of the product. Control then goes to state T_2. In this state, register P is decremented and the contents of B are added to A if $Q_1 = 1$; otherwise, A is left unchanged. The two control functions at time T_2 are:

$$Q_1T_2: \quad A \leftarrow A + B, E \leftarrow C_{out}$$
$$T_2: \quad P \leftarrow P - 1$$

The second statement is always executed when $T_2 = 1$. The first statement is executed at time T_2 only if $Q_1 = 1$. Thus, a status variable (here Q_1) can be included with a timing variable to form a control function. Note that it is convenient to decrement P at state T_2 so that its new value can be checked at state T_3.

Control goes to T_3 after T_2. At state T_3, the composite register EAQ is shifted to the right and the contents of P are checked for zeros. The binary variable P_z is 1 if the P register contains all 0's; otherwise P_z is 0. If $P_z = 1$, the operation is terminated and control goes to the initial state. If $P_z = 0$, control goes to state T_2 to form a new partial product. Note that P refers to the contents of the register, whereas P_z is a binary variable.

Data-Processor Specification

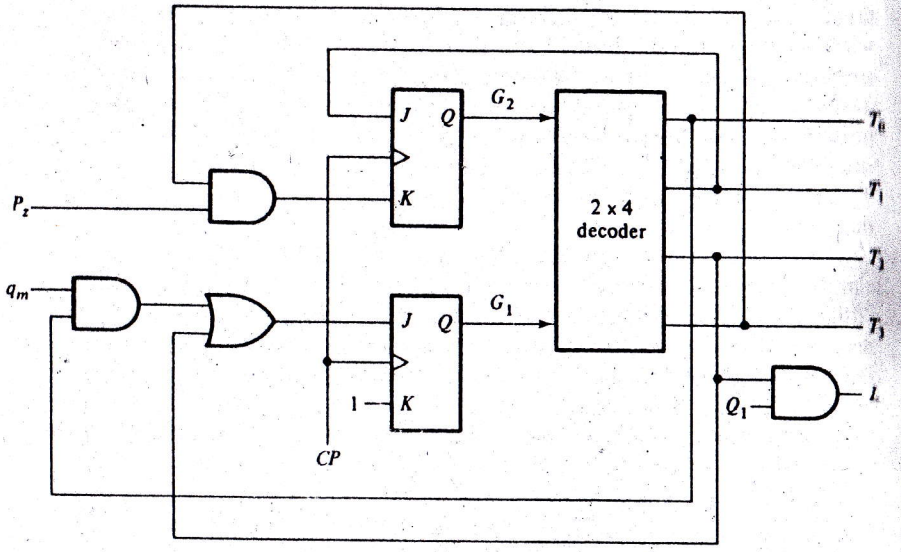
The data-processor part of the system can be derived from the microoperations list of Fig. 10-15(b). A block diagram of the data processor is shown in Fig. 10-16. A parallel adder is inserted between registers A and B to form the sum, which is

	Present State		Inputs		Next state		Flip-flop inputs			
	G_2	G_1	q_m	P_z	G_2	G_1	JG_2	KG_2	JG_1	KG_1
T_0	0	0	0	X	0	0	0	X	0	X
T_0	0	0	1	X	0	1	0	X	1	X
T_1	0	1	X	X	1	0	1	X	X	1
T_2	1	0	X	X	1	1	X	0	1	X
T_3	1	1	X	0	1	0	X	0	X	1
T_3	1	1	X	1	0	0	X	1	X	1

(a) Excitation table

$$JG_2 = T_1 \qquad KG_2 = T_3P_z$$
$$JG_1 = T_0q_m + T_2 \qquad KG_1 = 1$$

(b) Flip-flop input functions



(c) Logic diagram

Figure 10-17 Design of control for binary multiplier

The reason for KG_1 being always 1 is that all entries in the table for this input variable are either 1's or X's.

When deriving input functions by inspection from the excitation table, we cannot be sure that the functions have been simplified in the best way possible. For this reason, one should always analyze the circuit to ensure that the derived equations do indeed produce the required state transitions as specified in the state table.

The logic diagram of the control logic is drawn in Fig. 10-17(c). It consists of two flip-flops, G_1 and G_2, and a decoder. The outputs of the decoder are used to obtain the next state of the circuit according to the Boolean functions listed in Fig. 10-17(b). The outputs of the controller should be connected to the data-processor part of the system as shown in Fig. 10-16.

10-7 PLA CONTROL

We have seen from the two examples presented in this chapter that the design of a control circuit is essentially a sequential-logic design 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 and showed that it can be used to implement any combinational circuit. By replacing the combinational circuit with a PLA, it is then possible to design a control circuit with a register connected to a PLA. The register operates as a sequence register that determines the state of the control. The PLA is programmed to provide the control outputs and the next state for the sequence register.

The design of a control unit with a PLA is very similar to the design using the sequence register and decoder methods. In fact, the sequence register in both methods is the same. The difference in the methods is in the way the combinational-logic part of the control is implemented. The PLA essentially replaces the decoder and all other decision logic circuits required in the hard-wired implementation.

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 "programmed" 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 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 state table gives essentially all the information required for obtaining the PLA program table (or the ROM truth table).