Digital System Design

Lecture – 10

Control Logic Design

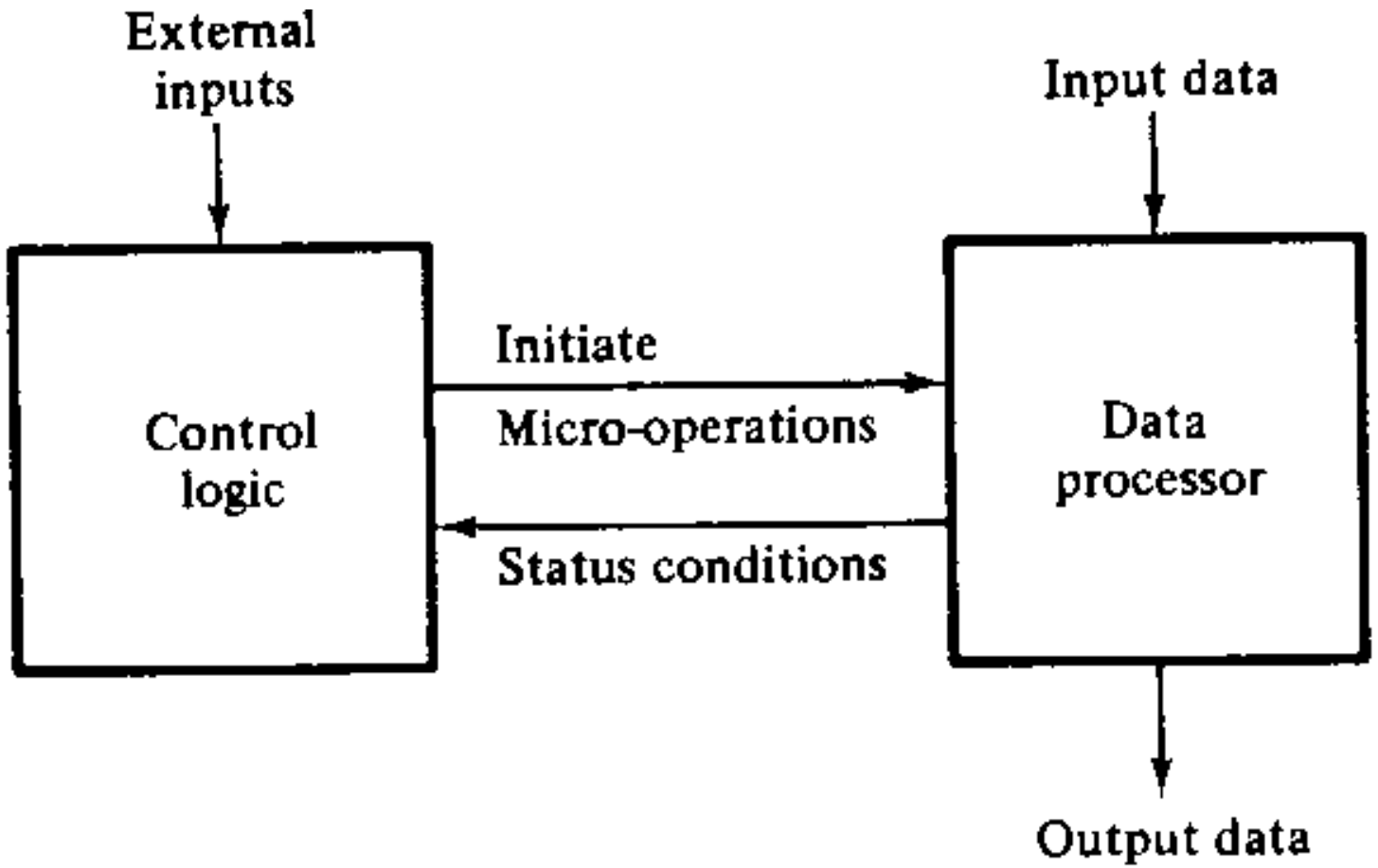
Introduction

- Binary information stored in processor or memory registers
- Can be either data or control information
- Data – Discrete elements of information that are manipulated by microoperations
- Control information provides command signals for specifying the sequence of microoperations
- **Logic Design of a digital system is a process for deriving the digital circuits that perform data processing and provide control signals**

What does a Control Unit do?

- Binary variables controlling the selection variables and enabling inputs of registers are generated in the control unit
- Outputs of the control unit selects and enables the data-processor part and determines the next state of the control unit itself
- Provides a time sequence of signals that initiates all microoperations in the data processor
- Generates signals for sequencing the microoperations
- Internal state dictates the control functions for the system
- Next state depends on status conditions or other inputs
- Directly related to the data processor part

Control and Data Processor Interactions



Representations in Control System Design

- **Timing Diagram:** A timing diagram clarifies the timing sequence and other relationships among the various control signals in the system.

- **Flowchart:** A flowchart is a convenient way to specify the sequence of procedural steps and decision paths for an algorithm.

Control Organization

- Major goal should be the development of a circuit that implements the desired control sequence in a logical and straightforward manner

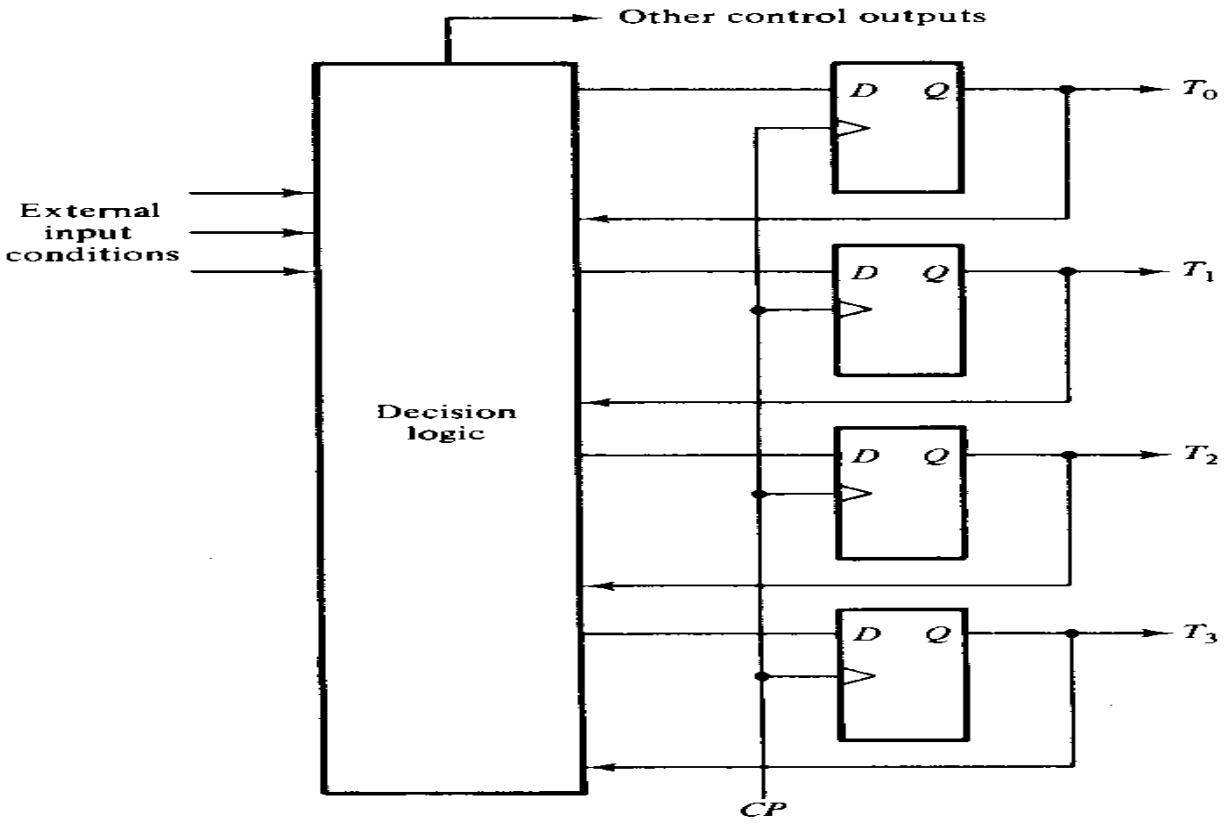
- **Four methods:**
  I. One flip-flop per state method
  II. Sequence register and decoder method
  III. PLA control
  IV. Microprogram control

Differences

- First two methods must use SSI and MSI circuits. Control unit implemented with SSI and MSI devices are called a hard-wired control. The circuits must be rewired to fulfill the new requirements if any modifications are needed in this case.

- PLA or microprogram control uses an LSI device (e.g. programmable logic array or a read-only memory). Any alterations in a microprogram control can be easily achieved without wiring changes by removing the ROM from its socket and inserting another programmed ROM

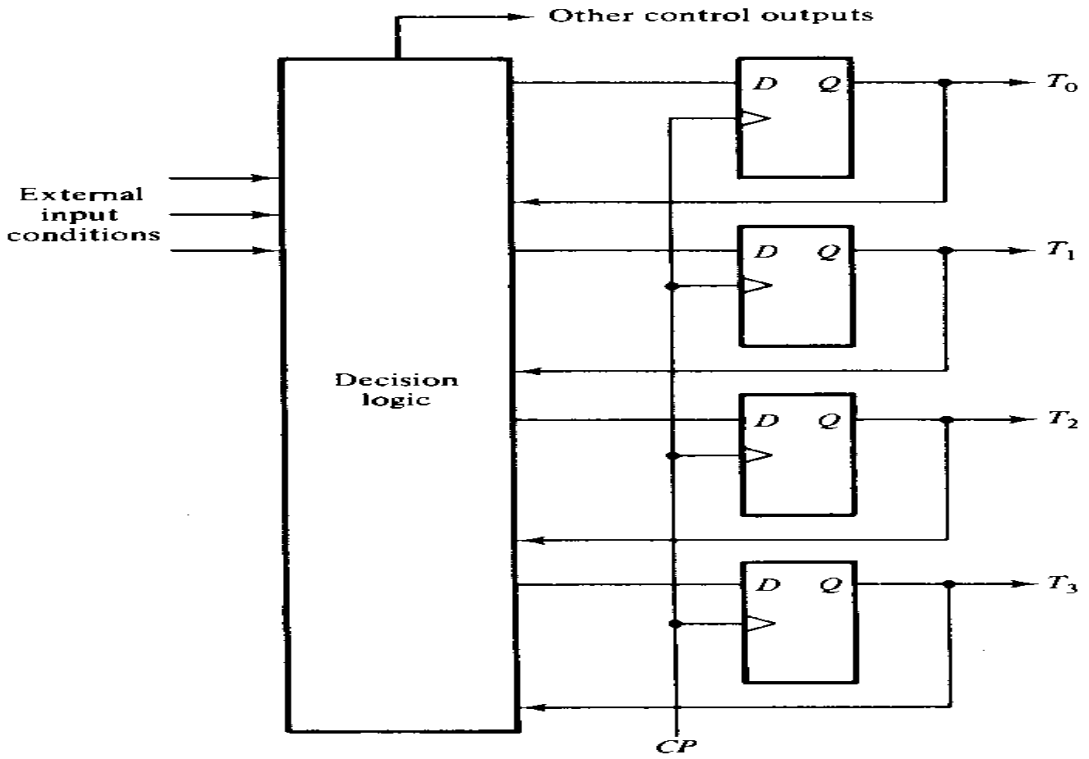
One Flip-Flop per State Method



- Uses one flip-flop per state
- Only one flip-flop set at any particular time, all others cleared
- A single bit is made to propagate from one flip-flop to the other under the control of decision logic
- Each flip-flop represents a state and is activated only when the control bit is transferred to 1

One Flip-Flop per State Method (Contd.)

- Requires a maximum number of flip-flops. E.g. 12 states require a minimum number of 4 flip-flops since $2^3<12<2^4$



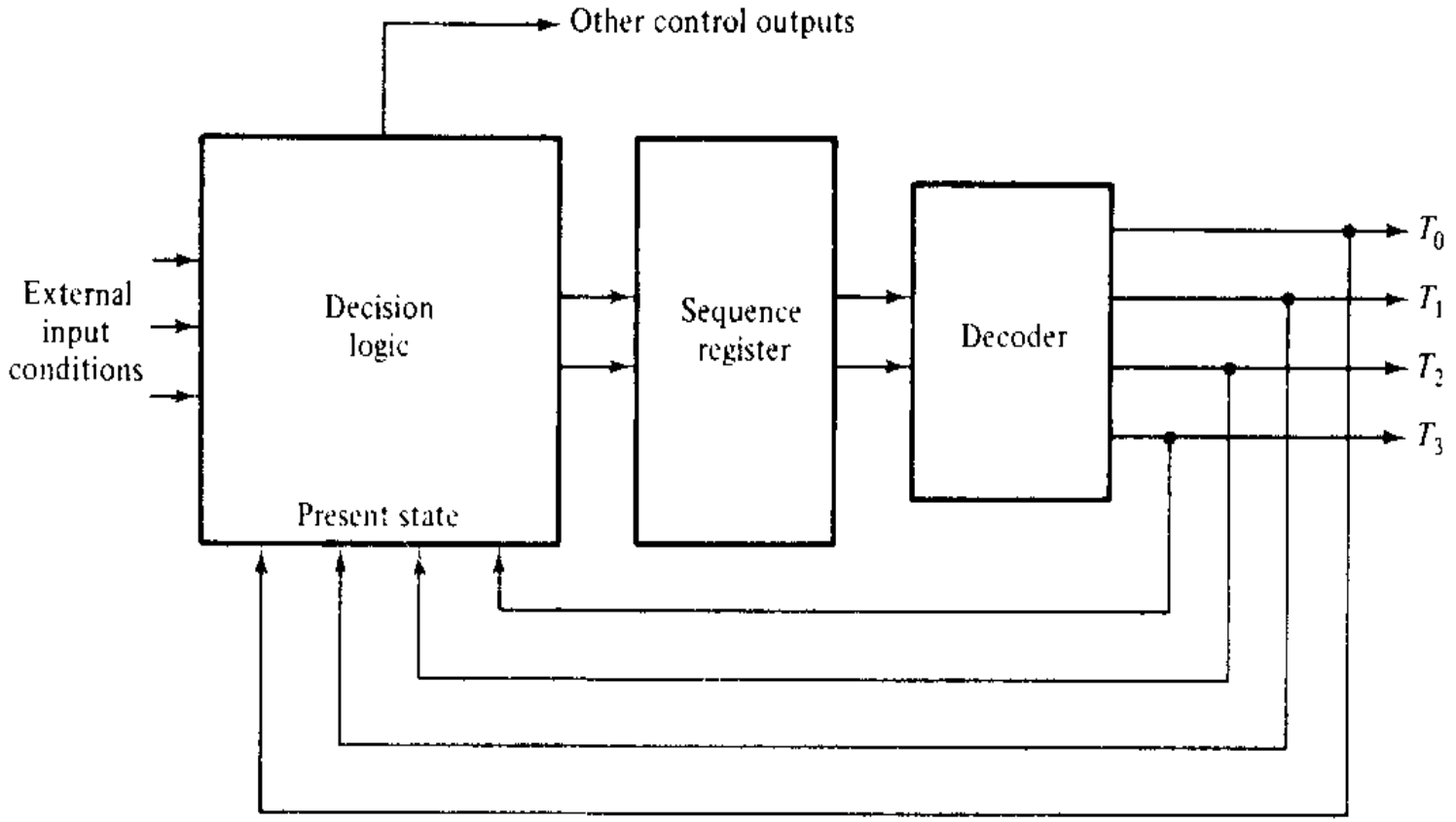
- Transition from present to next state is a function of T_i which is 1 and certain input conditions
- Each flip-flop connected to the data processing section
- Other Control outputs are a function of the T's and external inputs, which can also initiate microoperations

One Flip-Flop per State Method (Contd.)

- **Advantages:** Operational simplicity, saving the design effort, a potential decrease in the combinational circuits required to implement the complete sequential circuit
- **Disadvantages:** Increased system cost as more flip-flops are used

If the control circuit does not need external inputs, it reduces to a straight single bit shift register. And if the control sequence must be repeated over and over again, the control reduces to a ring counter. For this reason, the One FF per State Method is sometimes called a ring counter controller.

Sequence Register and Decoder Method

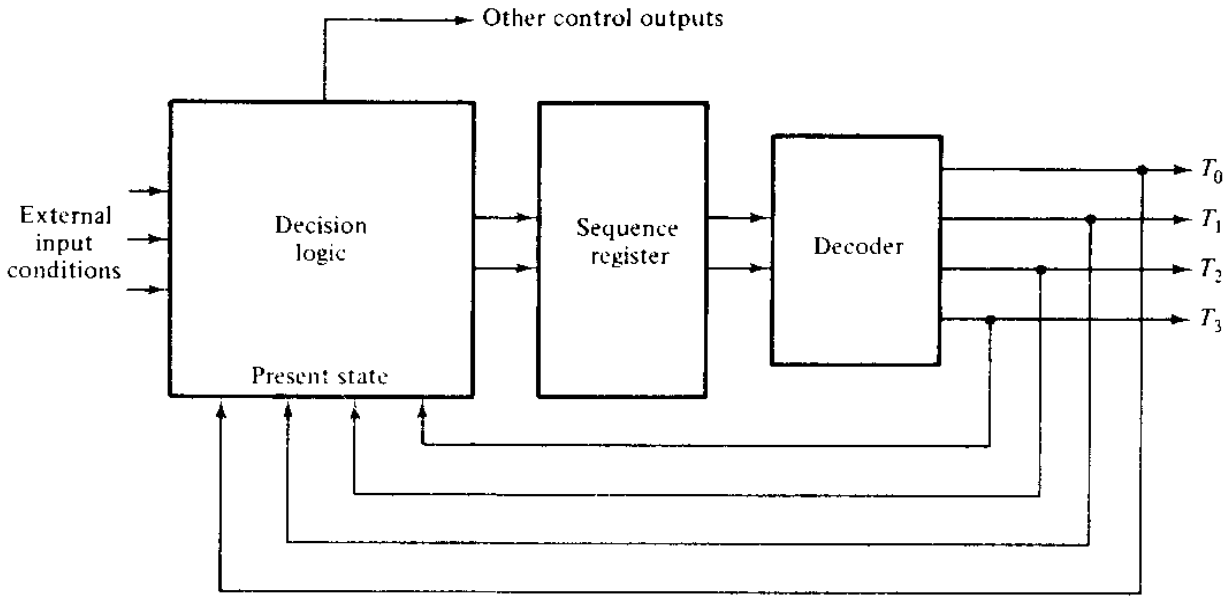


- Uses a register to sequence the control states
- Register is decoded to provide one output for each state
- Register and Decoder both are MSI Devices

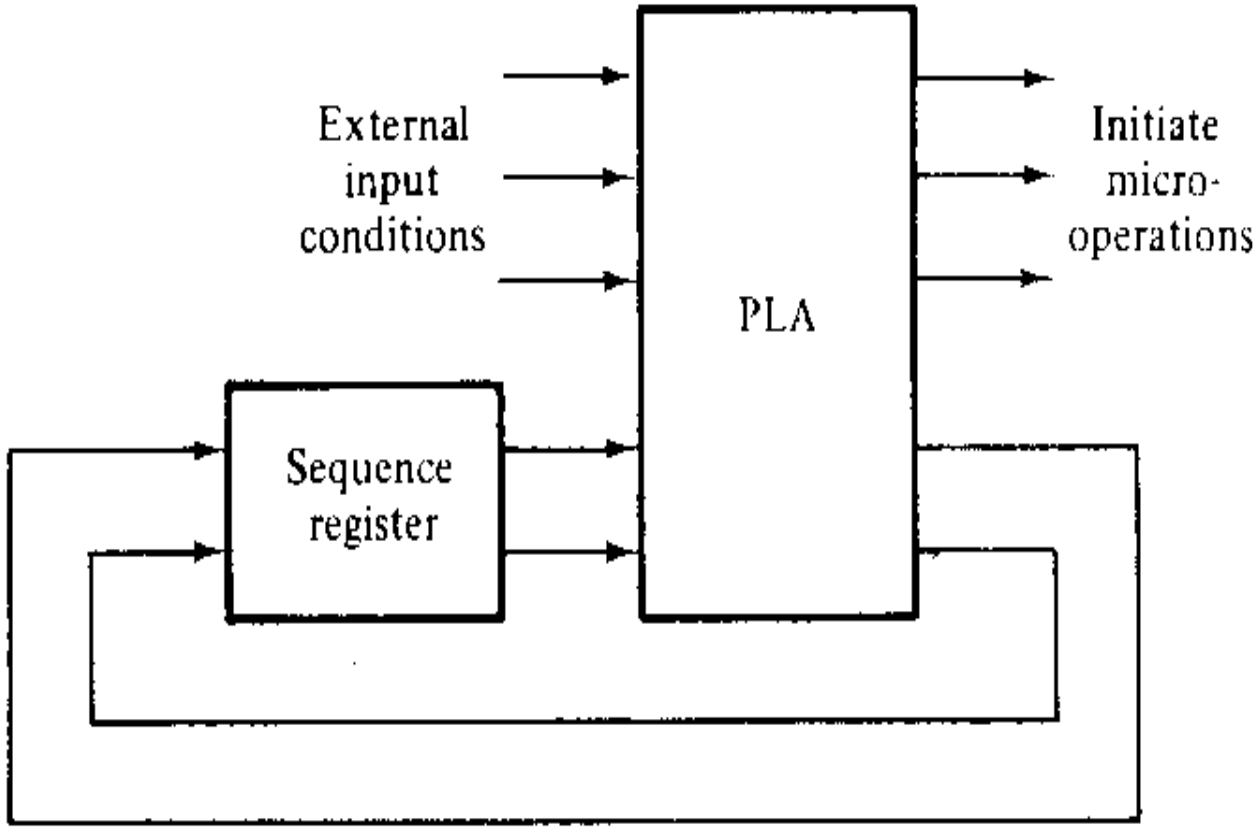
Sequence Register and Decoder Method (Contd.)

- For n flip-flops in the sequence register, 2^n circuit states and 2^n outputs for the decoder
- 4-bit register can be any of 16 states. A 4 X 16 decoder will have 16 outputs, one for each state of the register
- Next state transition in the sequence register is a function of the present state and the external input conditions
- Other outputs may also initiate microoperations

Ques: Why is it called a counter-decoder method?



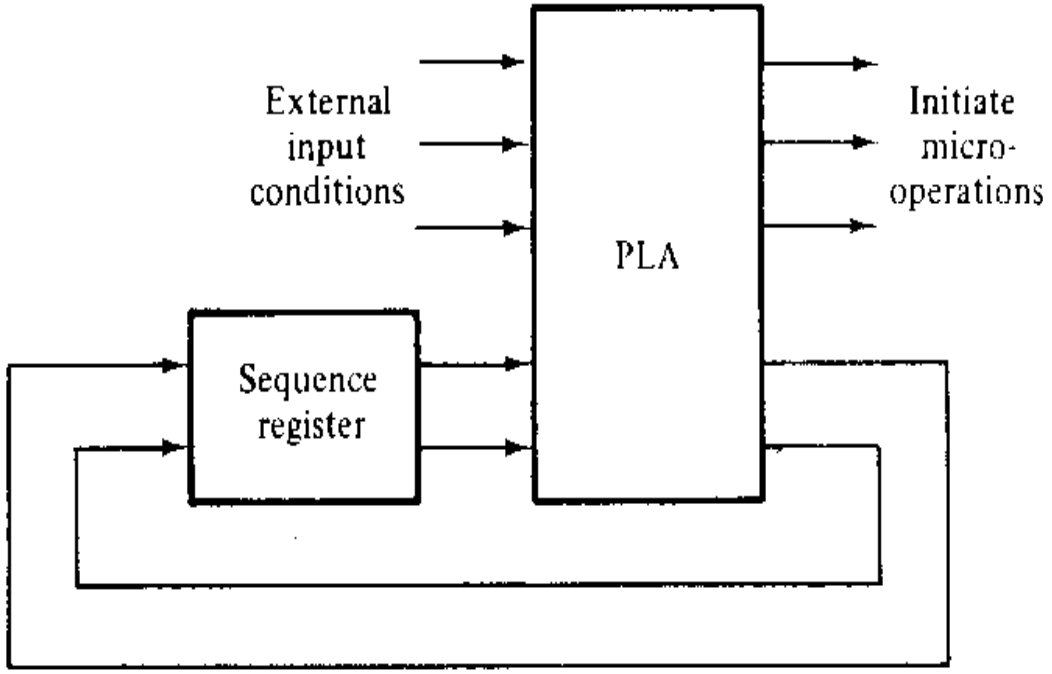
PLA Control



- Similar to the Sequence register and Decoder Method
- All combinational circuits including the decoder and the decision logic are implemented by PLA
- Using PLA reduces number of ICs and interconnecting wires

PLA Control (Contd.)

- An external sequence register establishes the present state
- PLA determines which microoperations should be initiated, depending on present state of the sequence register and external input conditions
- Other PLA outputs determines the next state of the sequence register



Microprogram Control

- A control unit whose control variables are stored in a memory is called a **microprogrammed control unit**
- Each control word of memory is called a **microinstruction**
- A sequence of microinstructions is called a **microprogram**
- Control memory can be a ROM
- Content of the word in ROM at a given address specifies the microoperations
- **Dynamic microprogram** permits a microprogram to be loaded initially from computer console or from auxiliary memory such as a magnetic disk

Microprogram Control (Contd.)

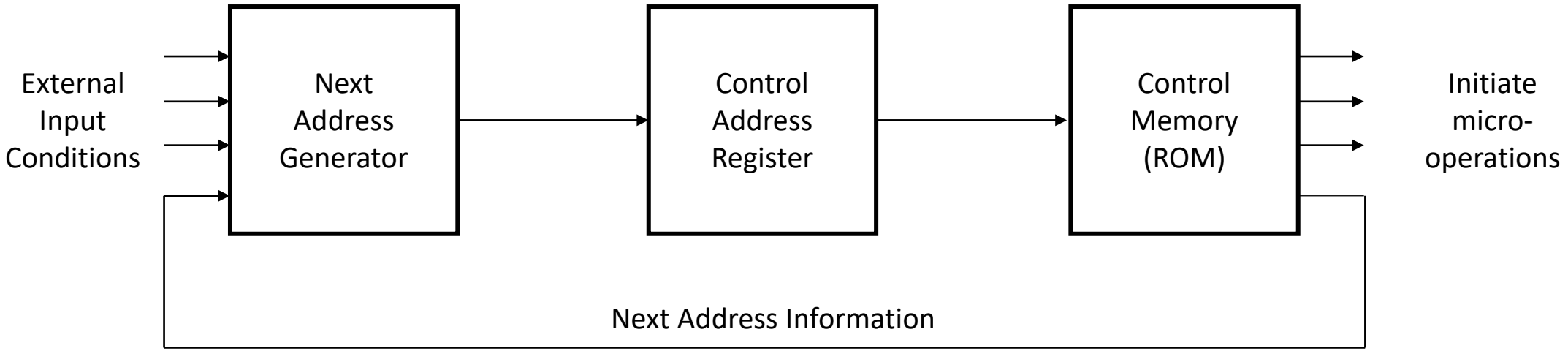


Figure: Microprogram Control Logic

- ROM operates with the address value as input and the corresponding word as the output
- Word remains on the output wire as long as the address remains in the address register
- ROM word should be transferred to a buffer register in case of changes

Microprogram Control (Contd.)

- Microoperations executed → CU determines next address
- Next microinstruction location can be sequential/located somewhere else in the control memory
- So, some bits of the microinstruction needed to control the next microinstruction's address generation
- Next address may be a function of external input conditions
- And computed in the next address generator circuit while the microinstructions being executed
- Then transferred with the next clock pulse into the control address register to read the next microinstruction
- Next address generator circuit construction depends on particular application

Hard-Wired Control - Example 1

This example demonstrates the one flip-flop per state method. The design is carried out in five consecutive steps:

1. The problem is stated
2. An initial equipment configuration is assumed
3. The algorithm is formulated
4. The data-processor part is specified
5. The control logic is designed

Hard-Wired Control - Example 1 (Contd.)

1. **Statement of the problem:**

The problem here is to implement with hardware the addition and subtraction of two fixed-point binary numbers in sign-magnitude form.

The addition of two numbers stored in registers of finite length may result in a sum that exceeds the storage capacity of the register by one bit. The extra bit is said to cause an overflow. The circuit must provide a flip-flop for storing a possible overflow bit.

Hard-Wired Control - Example 1 (Contd.)

2. Equipment Configuration:

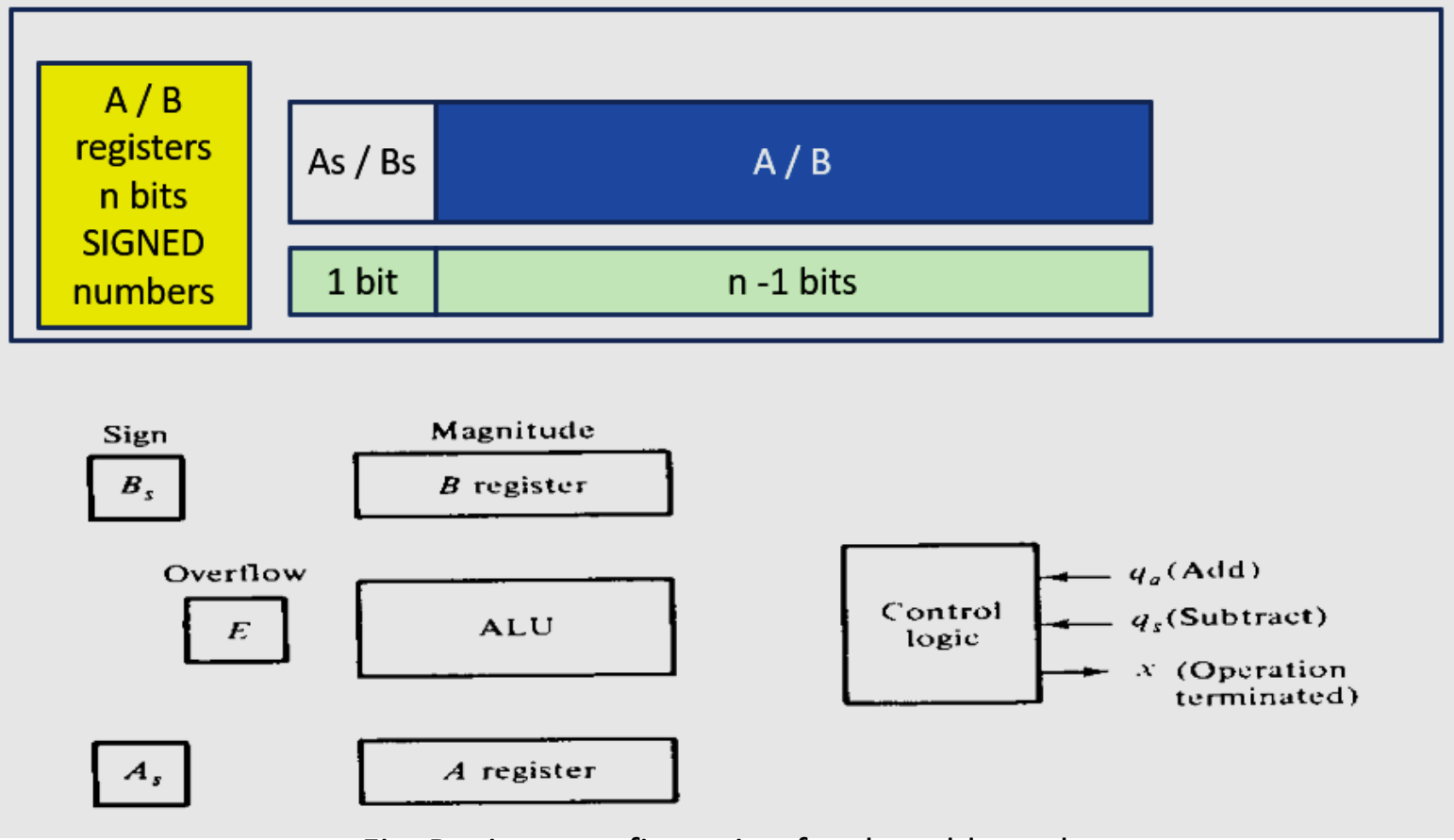


Fig: Register configuration for the adder-subtractor

Hard-Wired Control - Example 1 (Contd.)

3. Derivation of the Algorithm:

When two numbers A and B are added or subtracted, eight possible combinations can occur:

1. (+A) + (+B)
2. (+A) + (-B)
3. (-A) + (+B)
4. (-A) + (-B)
5. (+A) – (-B)
6. (+A) – (+B)
7. (-A) – (-B)
8. (-A) – (+B)

These eight combinations can be expressed as:

$(\pm A) \pm (\pm B)$

Hard-Wired Control - Example 1 (Contd.)

3. Derivation of the Algorithm:

If the arithmetic operation specified is subtraction, we change the sign of B and add. This is evident from the relations:

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

This reduces the number of possible conditions to four, namely:

$$(\pm A) + (\pm B)$$

When the signs of A and B are the same, we add the two magnitudes and the sign of the result is the same as the common sign. When the signs of A and B are not the same, we subtract the smaller number from the larger and the sign of the result is the sign of the larger number. This is evident from the following relationships:

	if $A \geq B$	if $A < B$
$(+A) + (+B) =$	$+(A + B)$	
$(+A) + (-B) =$	$+(A - B) = -(B - A)$	
$(-A) + (+B) =$	$-(A - B) = +(B - A)$	
$(-A) + (-B) =$	$-(A + B)$	

Hard-Wired Control - Example 1 (Contd.)

Flow Chart:

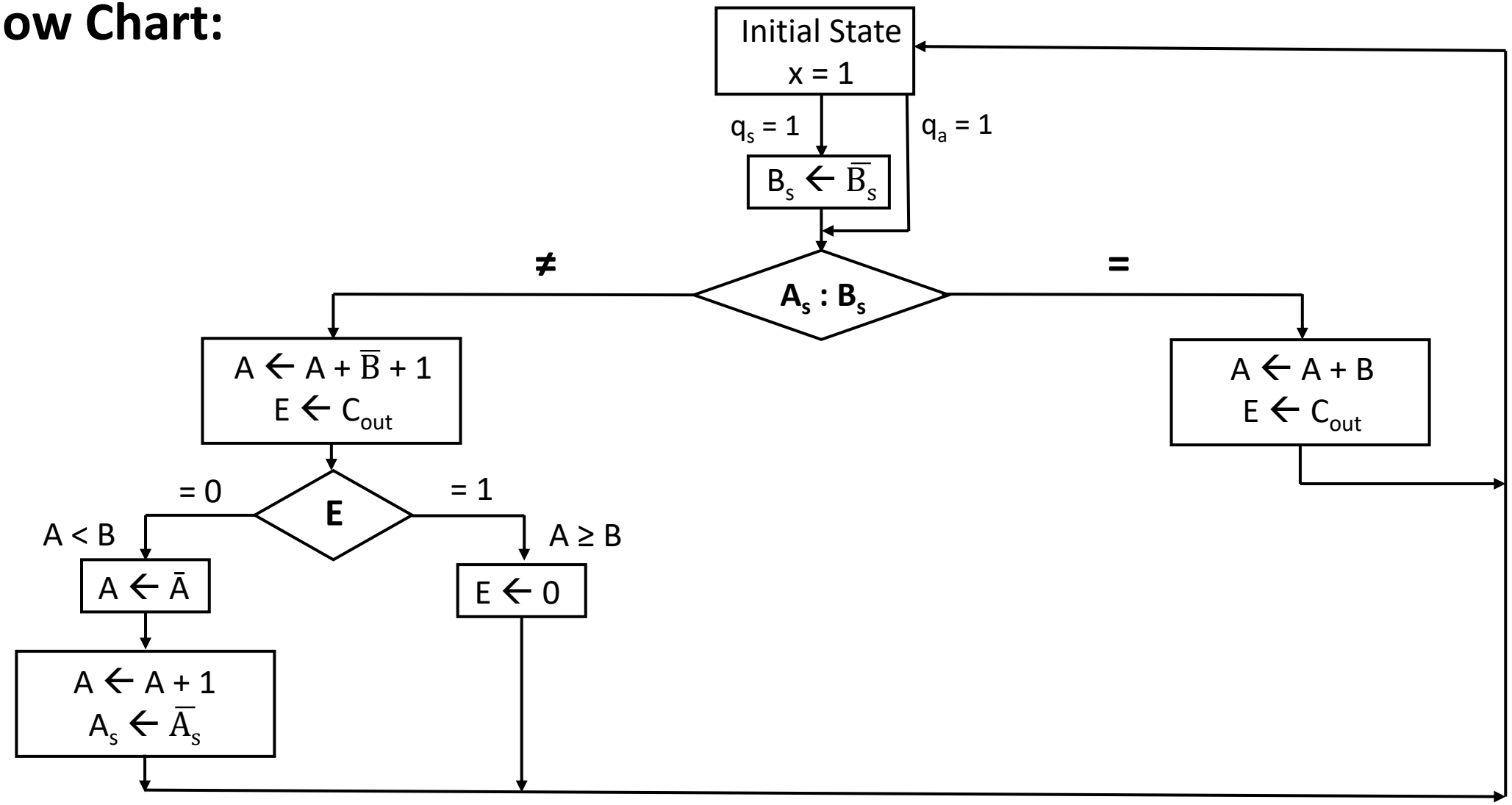


Fig: Flowchart for Sign Magnitude Addition and Subtraction

Hard-Wired Control - Example 1 (Contd.)

4. Data Processor Part:

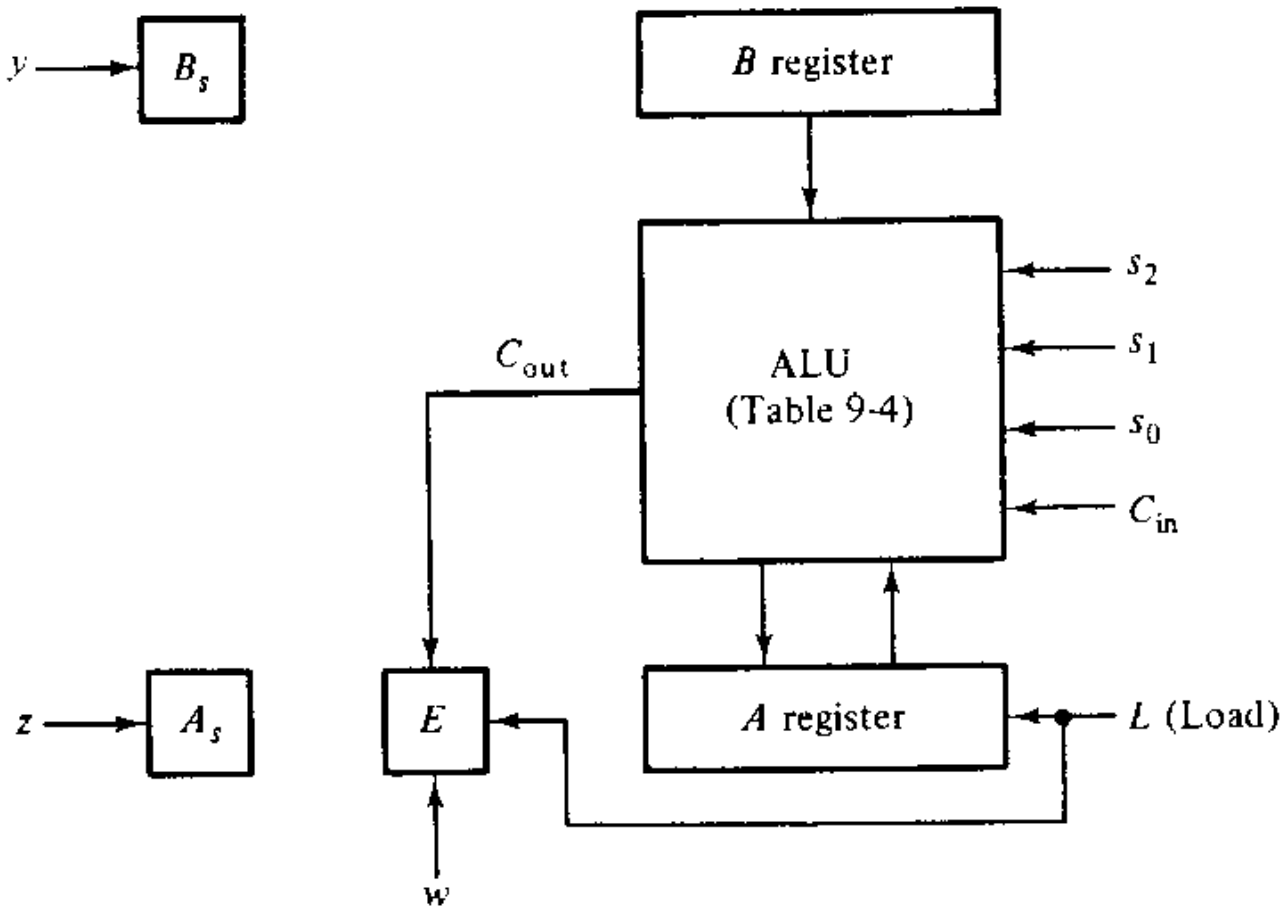


Fig: Data Processor Registers and ALU

Hard-Wired Control - Example 1 (Contd.)

5. Control Logic Design:

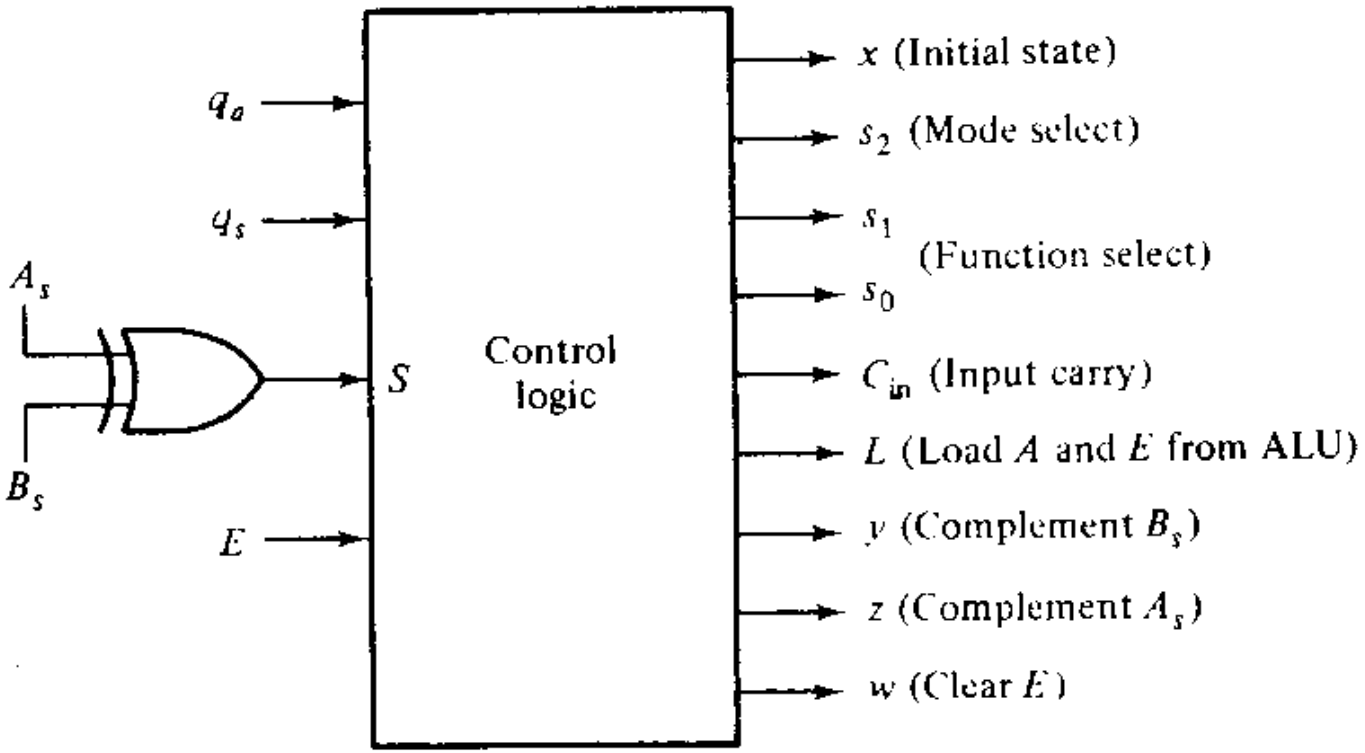


Fig: Control Block Diagram

The outputs of the control logic should be connected too the corresponding inputs in the data-processor.

Hard-Wired Control - Example 1 (Contd.)

Control State Diagram:

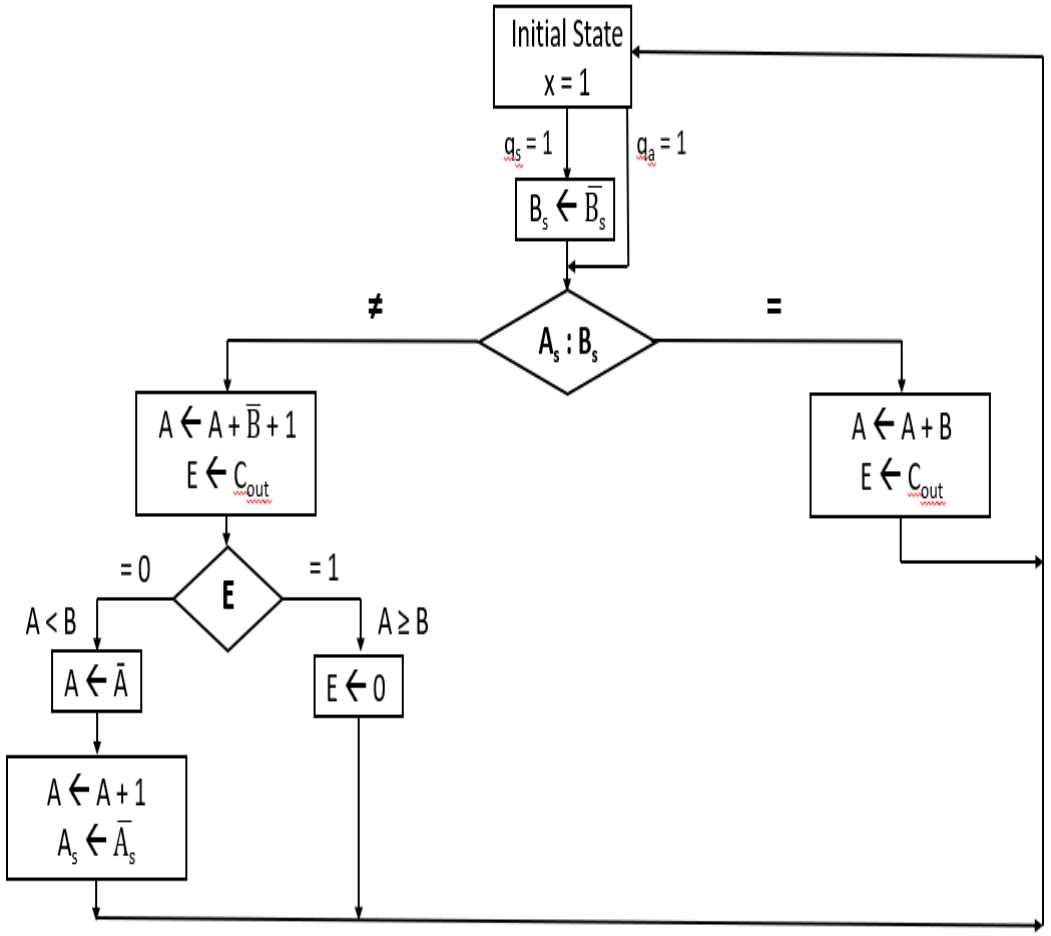


Fig: Flowchart for Sign Magnitude Addition and Subtraction

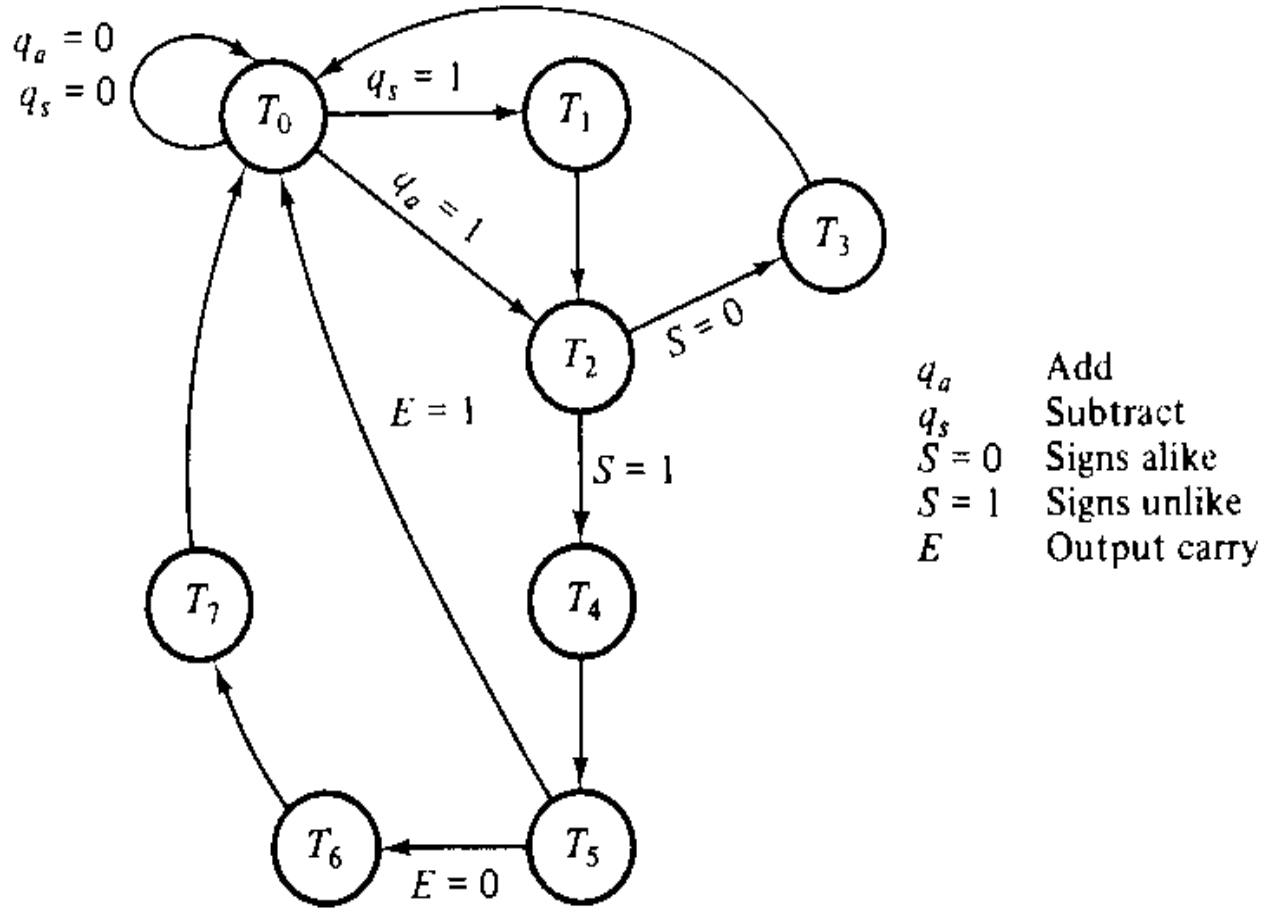
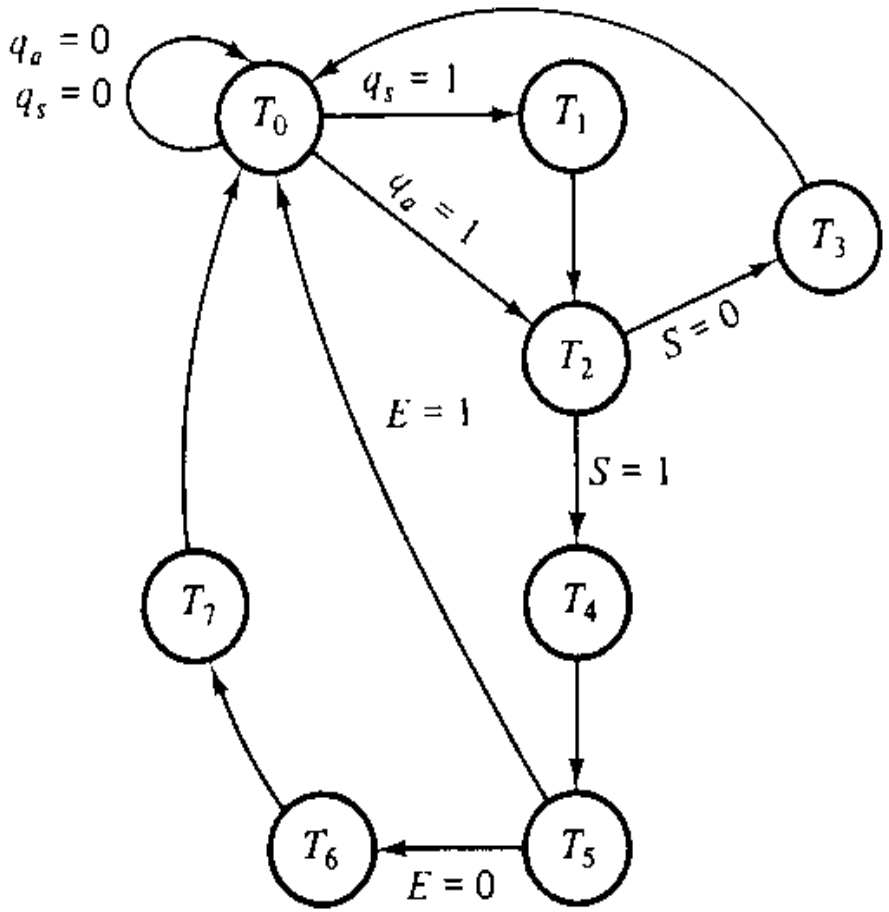


Fig: Control State Diagram

Hard-Wired Control - Example 1 (Contd.)

Control State Diagram:



q_a	Add
q_s	Subtract
$S = 0$	Signs alike
$S = 1$	Signs unlike
E	Output carry

In T4: E = Cout is NOT executed until a clock pulse. But after the clock pulse, it goes to T5 state. So E SHOULD BE CHECKED at T5 state.

Fig: Control State Diagram

Hard-Wired Control - Example 1 (Contd.)

Control State Diagram:

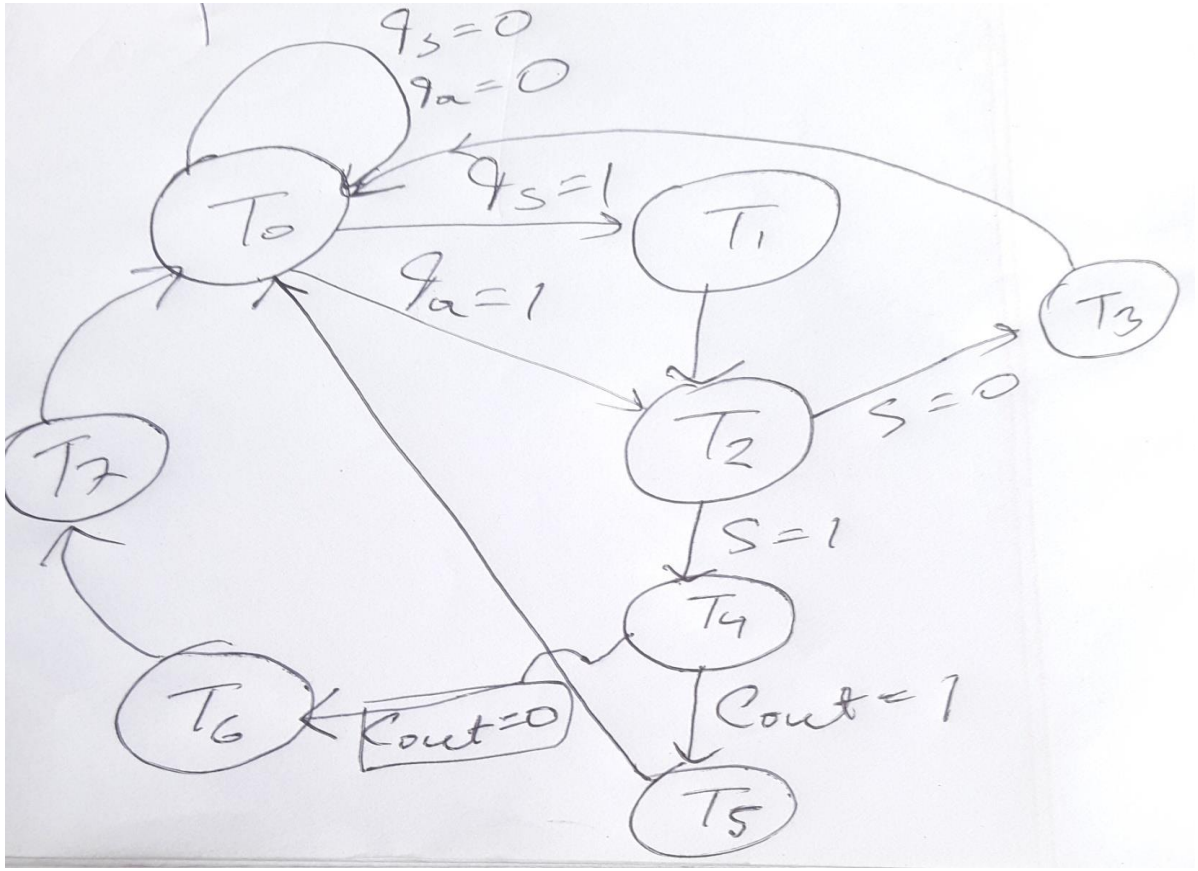
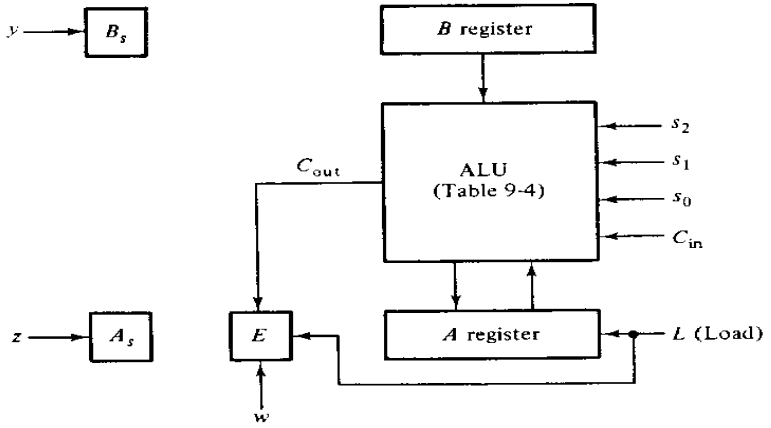


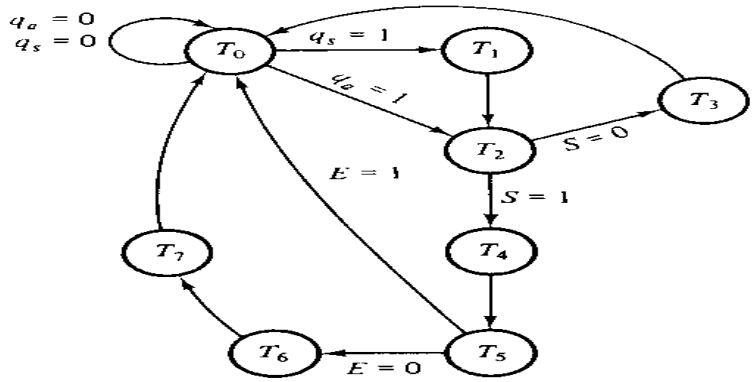
Fig: Control State Diagram

Flowchart can result in a different state diagram as long as it follows the hardware constraints and the system functions according to the specification. For example, we can check Cout at T4 state instead of checking E at T5:
If Cout = = 1: Clear E @ T5 state
If Cout = = 0: Go directly to T6 state

Hard-Wired Control - Example 1 (Contd.)

Sequence of Microoperations:



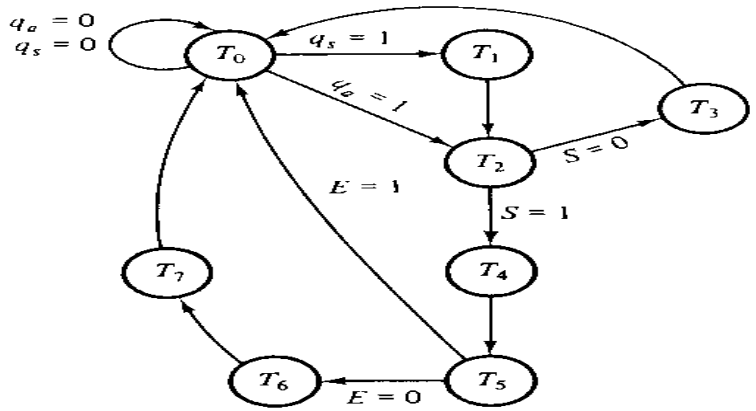


	Control outputs							
	x	s_2	s_1	s_0	C_{in}	L	y	z	w
T_0: Initial state $x = 1$	1	0	0	0	0	0	0	0	0
$T_1: B_s \leftarrow \overline{B}_s$	0	0	0	0	0	0	1	0	0
T_2: nothing	0	0	0	0	0	0	0	0	0
$T_3: A \leftarrow A + B,\ E \leftarrow C_{out}$	0	0	0	1	0	1	0	0	0
$T_4: A \leftarrow A + \overline{B} + 1,\ E \leftarrow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

Hard-Wired Control - Example 1 (Contd.)

Design of Hardwired Control:

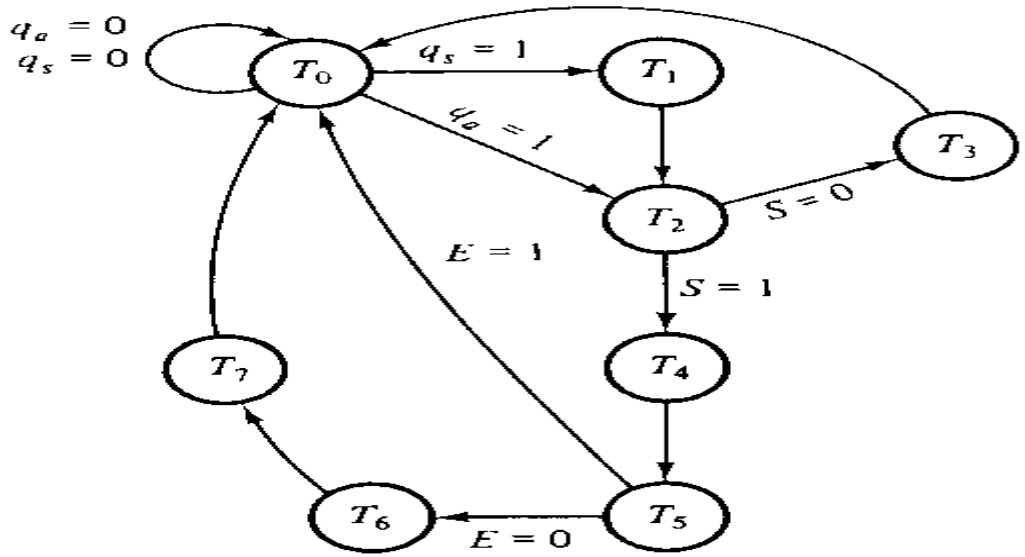


	Control outputs								
	x	s_2	s_1	s_0	C_{in}	L	y	z	w
T_0: Initial state $x = 1$	1	0	0	0	0	0	0	0	0
$T_1: B_s \leftarrow \overline{B}_s$	0	0	0	0	0	0	1	0	0
T_2: nothing	0	0	0	0	0	0	0	0	0
$T_3: A \leftarrow A + B,\ E \leftarrow C_{out}$	0	0	0	1	0	1	0	0	0
$T_4: A \leftarrow A + \overline{B} + 1,\ E \leftarrow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

TABLE 10-1 Boolean functions for control

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

Microprogram Control:



1. Provision for loading an external address as a result of the occurrence of external signals q_a and q_s.
2. Provision for sequencing consecutive addresses.
3. Provision for choosing between two addresses as a function of present values of the status variables S and E.

Microprogram Control:

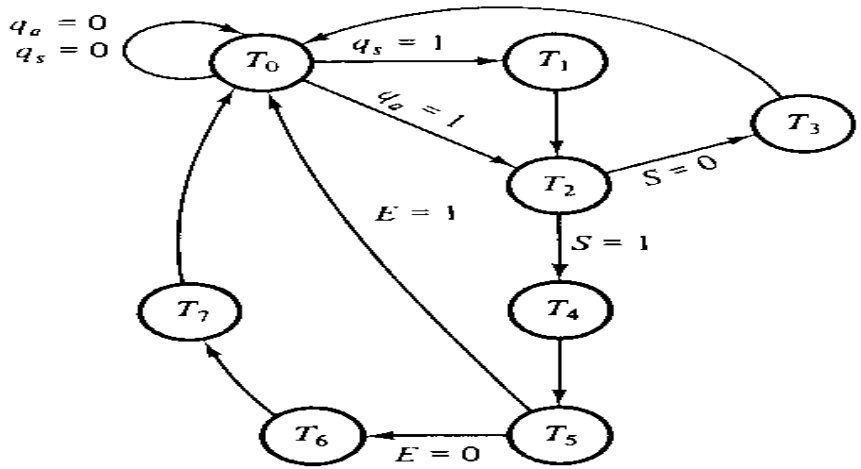
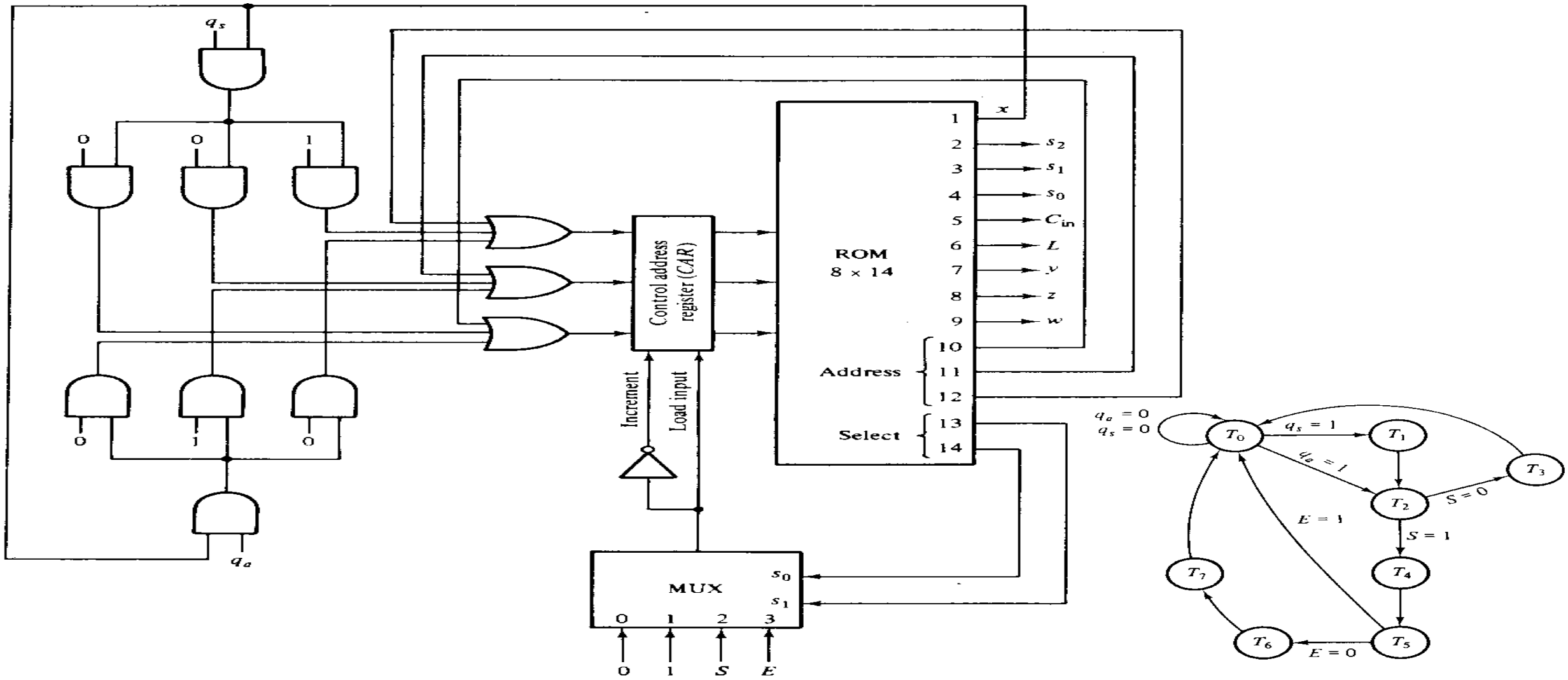


TABLE 10-2 Symbolic microprogram for control memory

ROM address	Microinstruction	Comments
0	$x = 1$, if $(q_s = 1)$ then (go to 1), if $(q_a = 1)$ then (go to 2), if $(q_s \wedge q_a = 0)$ then (go to 0)	Load 0 or external address
1	$B_s \leftarrow \overline{B}_s$	$q_s = 1$, start subtraction
2	If $(S = 1)$ then (go to 4)	$q_a = 1$, start addition
3	$A \leftarrow A + B$, $E \leftarrow C_{out}$, go to 0	Add magnitudes and return
4	$A \leftarrow A + \overline{B} + 1$, $E \leftarrow C_{out}$	Subtract magnitudes
5	If $(E = 1)$ then (go to 0), $E \leftarrow 0$	Operation terminated if $E = 1$
6	$A \leftarrow \overline{A}$	$E = 0$, complement A
7	$A \leftarrow A + 1$, $A_s \leftarrow \overline{A}_s$, go to 0	Done, return to address 0

Microprogram Hardware Configuration:



Microprogram Hardware Configuration:

ROM bits 13	14	MUX select function
0	0	Increment CAR
0	1	Load input to CAR
1	0	Load input to CAR if $S = 1$, increment CAR if $S = 0$
1	1	Load inputs to CAR if $E = 1$, increment CAR if $E = 0$

The Microprogram:

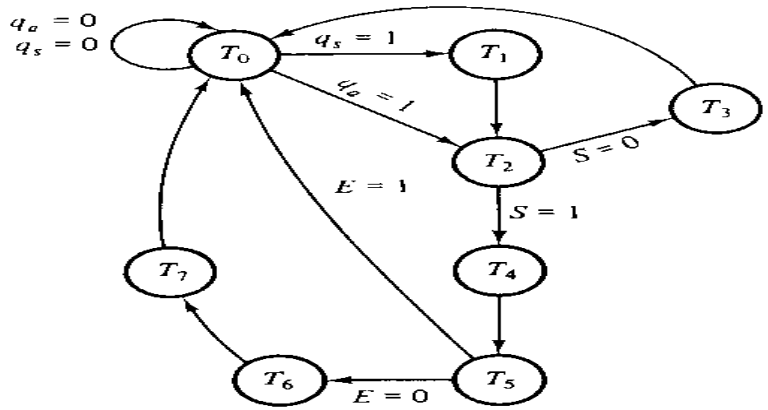


TABLE 10-3 Binary microprogram for control memory

ROM address			ROM outputs												
			x	s_2	s_1	s_0	C_{in}	L	y	z	w	*Address*			*Select*	
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	1
1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1
1	1	1	0	0	0	0	1	1	0	1	0	0	0	0	0	1

Hard-Wired Control - Example 2 (Section 10.6)

23		10111	multiplicand
	×		
19		10011	multiplier
		10111	
		10111	
		00000	+
		00000	
		10111	
437		110110101	product

Hard-Wired Control - Example 2 (Contd.)

1. Problem Statement:

We wish to design an arithmetic circuit that multiplies two fixed-point binary numbers in sign-magnitude representation.

The control organization chosen for this example is the sequence register and decoder method. But the control logic is designed by means of a PLA.

If last bit of Multiplier is 1: Add Multiplicand with the partial product and Right Shift the partial product

If last bit of Multiplier is 0: Just Right Shift the partial product.

Hard-Wired Control - Example 2 (Contd.)

1. Problem Statement:

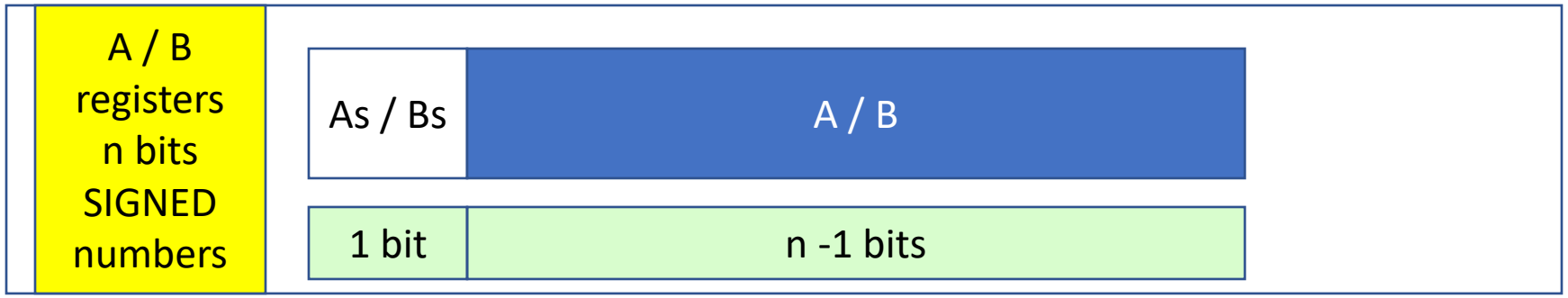
If last bit of Multiplier is 1: Add Multiplicand with the partial product and Right Shift the partial product

If last bit of Multiplier is 0: Just Right Shift the partial product.

multiplicand:	10111
multiplier:	10011
1st multiplier bit = 1, copy multiplicand	10111
shift right to obtain 1st partial product	010111
2nd multiplier bit = 1, copy multiplicand	10111
add multiplicand to previous partial product	1000101
shift right to obtain 2nd partial product	1000101
3rd multiplier bit = 0, shift right to obtain 3rd partial product	01000101
4th multiplier bit = 0, shift right to obtain 4th partial product	001000101
5th multiplier bit = 1, copy multiplicand	10111
add multiplicand to previous partial product	110110101
shift right to obtain 5th partial product = final product	0110110101

Hard-Wired Control - Example 2 (Contd.)

2. Equipment Configuration:



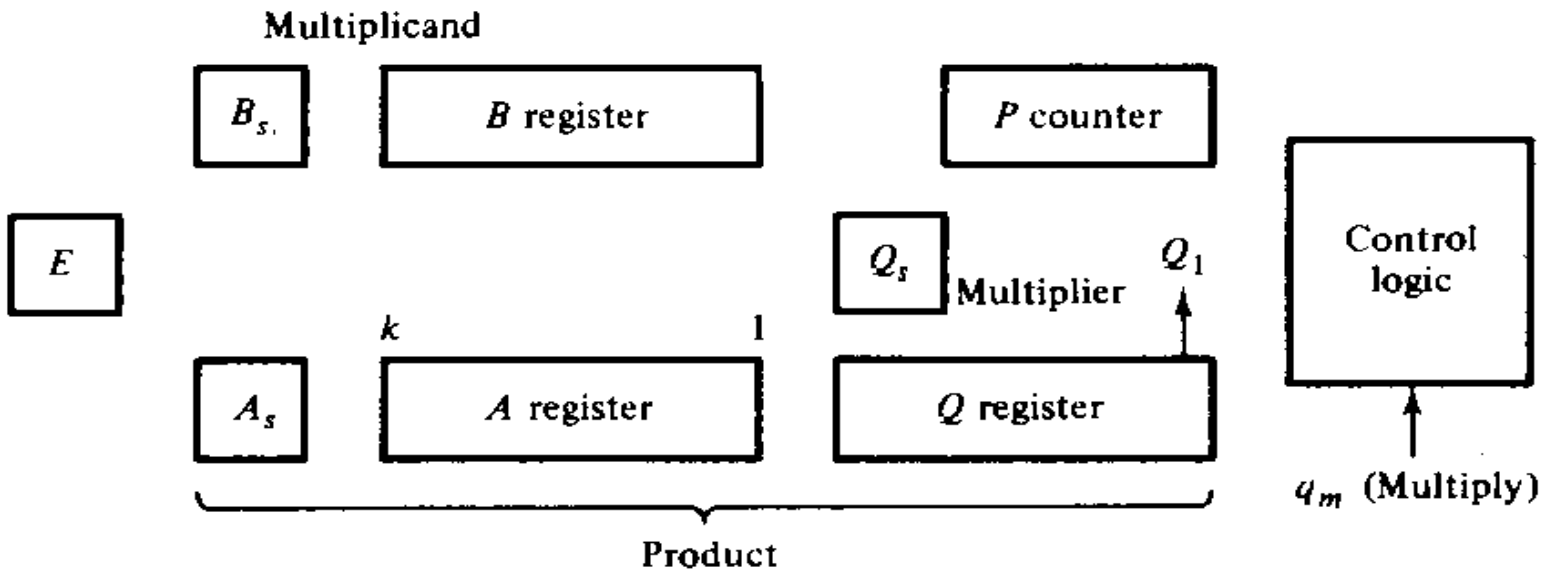


Figure 10-13 Registers for binary multiplier

Hard-Wired Control - Example 2 (Contd.)

3. Derivation of the Algorithm:

multiplicand is in B and the multiplier in Q. Their corresponding signs are in B_s and Q_s. The multiplication process is initiated when $q_m = 1$. The two signs are compared by means of an exclusive-OR gate. If the two signs are alike, the exclusive-OR operation produces a 0 which is transferred to A_s to give a plus for the product. If the signs are unlike, a 1 is transferred to A_s to give a negative sign for the product. Registers A and E are cleared and the sequence counter P is set to a binary number k, 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 present partial product in A. Any 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 A, Q, and E are then shifted once to the right to obtain a new partial product. This shift operation is symbolized in the flowchart in compact form with the statement:

$$AQ \leftarrow \text{shr } EAQ, E \leftarrow 0$$

EAQ is a composite register made up of registers E, A, and Q. If we use the individual register symbols, the shift operation can be described by the following microoperations:

$$A \leftarrow \text{shr } A, Q \leftarrow \text{shr } Q, A_k \leftarrow E, Q_k \leftarrow A_1, E \leftarrow 0$$

Hard-Wired Control - Example 2 (Contd.)

3. Derivation of the Algorithm:

- B ← multiplicand magnitude, B_s ← sign
- Q ← multiplier magnitude, Q_s ← sign
- A ← A + B (Multiplicand + Partial Product)
- E ← C_{out}
- P ← k, k = number of bits in the multiplier

Shift Operation:

AQ ← shr EAQ, E← 0

Hard-Wired Control - Example 2 (Contd.)

Flowchart for binary multiplication:

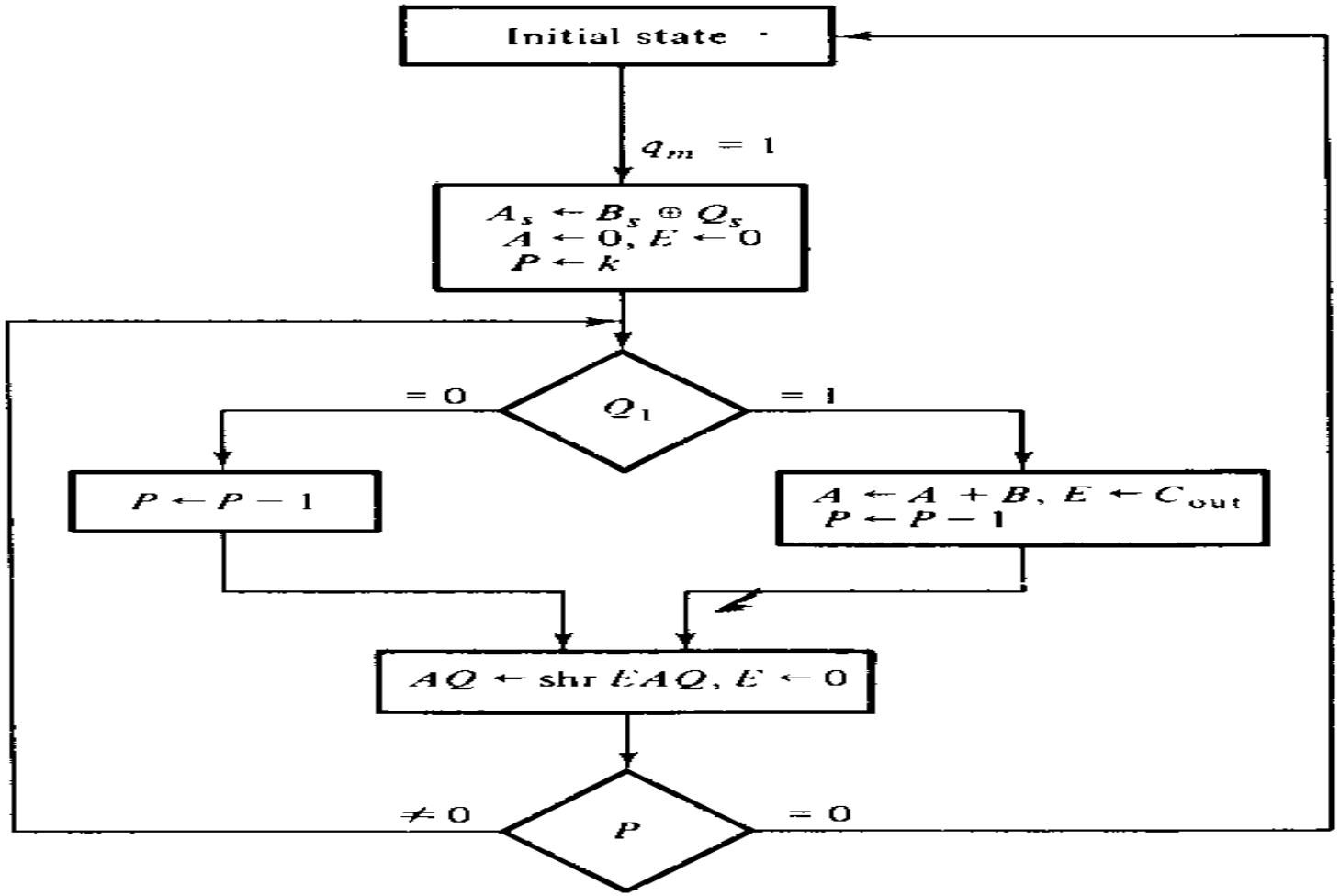


Figure 10-14 Flowchart for binary multiplier

Hard-Wired Control - Example 2 (Contd.)

Control State Diagram for multiplication:

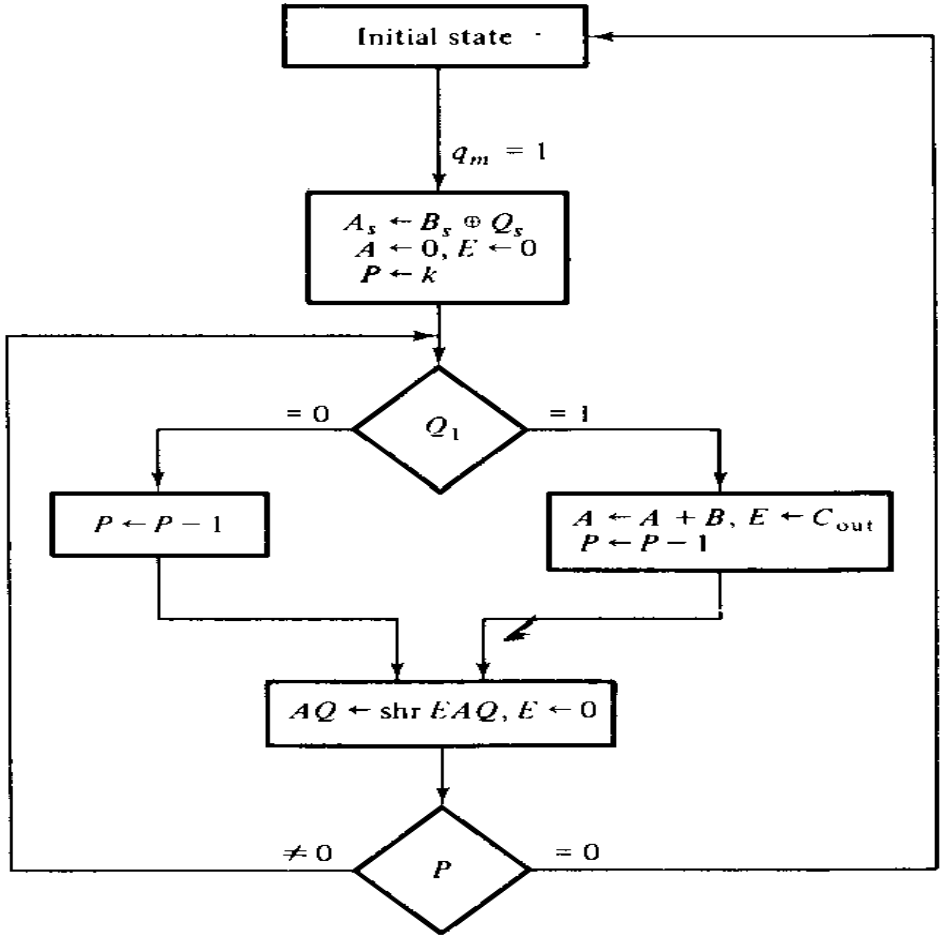
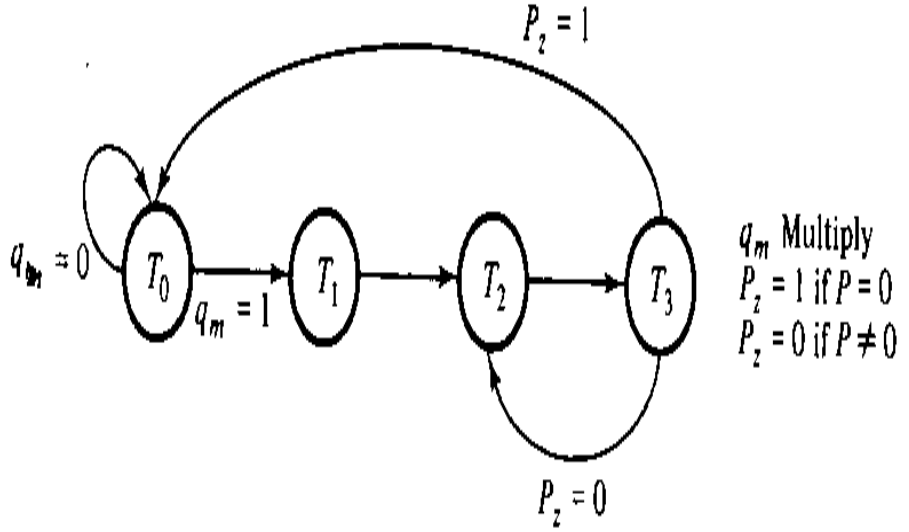


Figure 10-14 Flowchart for binary multiplier



(a) State diagram

T_0: Initial state
T_1: $A_s \leftarrow B_s \oplus Q_s, A \leftarrow 0, E \leftarrow 0, P \leftarrow k$
$Q_1 T_2$: $A \leftarrow A + B, E \leftarrow C_{out}$
T_2: $P \leftarrow P - 1$
T_3: $AQ \leftarrow \text{shr } EAQ, E \leftarrow 0$

Hard-Wired Control - Example 2 (Contd.)

4. Data Processor Specification:

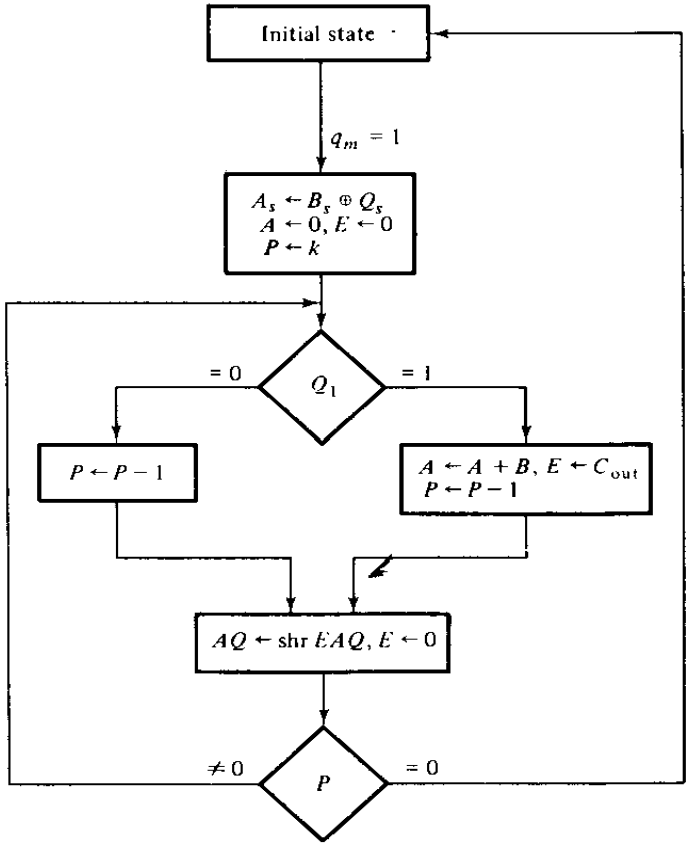


Figure 10-14 Flowchart for binary multiplier

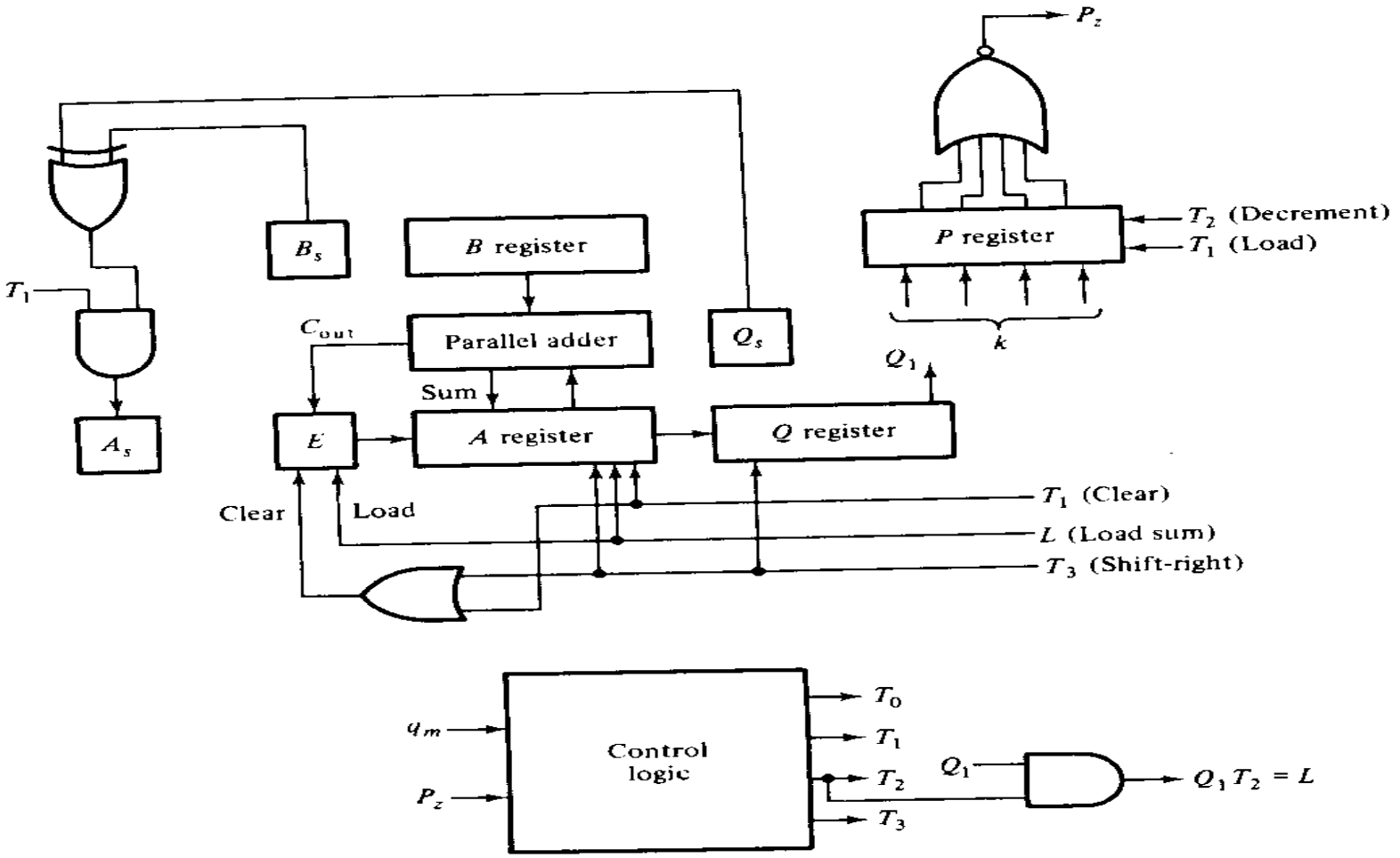
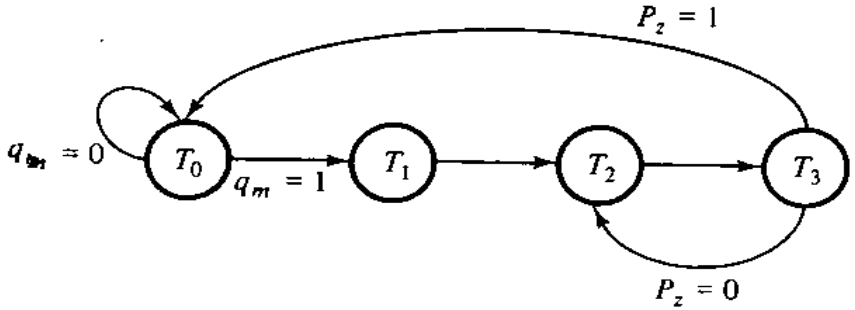


Figure 10-16 Data processor for binary multiplier

Hard-Wired Control - Example 2 (Contd.)

5. Hard-wired Control with JK FF:



(a) State diagram

q_m Multiply
$P_z = 1$ if $P = 0$
$P_z = 0$ if $P \neq 0$

T_0: Initial state
T_1: $A_s \leftarrow B_s \oplus Q_s, A \leftarrow 0, E \leftarrow 0, P \leftarrow k$
$Q_1 T_2$: $A \leftarrow A + B, E \leftarrow C_{out}$
T_2: $P \leftarrow P - 1$
T_3: $AQ \leftarrow \text{shr}\, EAQ, E \leftarrow 0$

	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

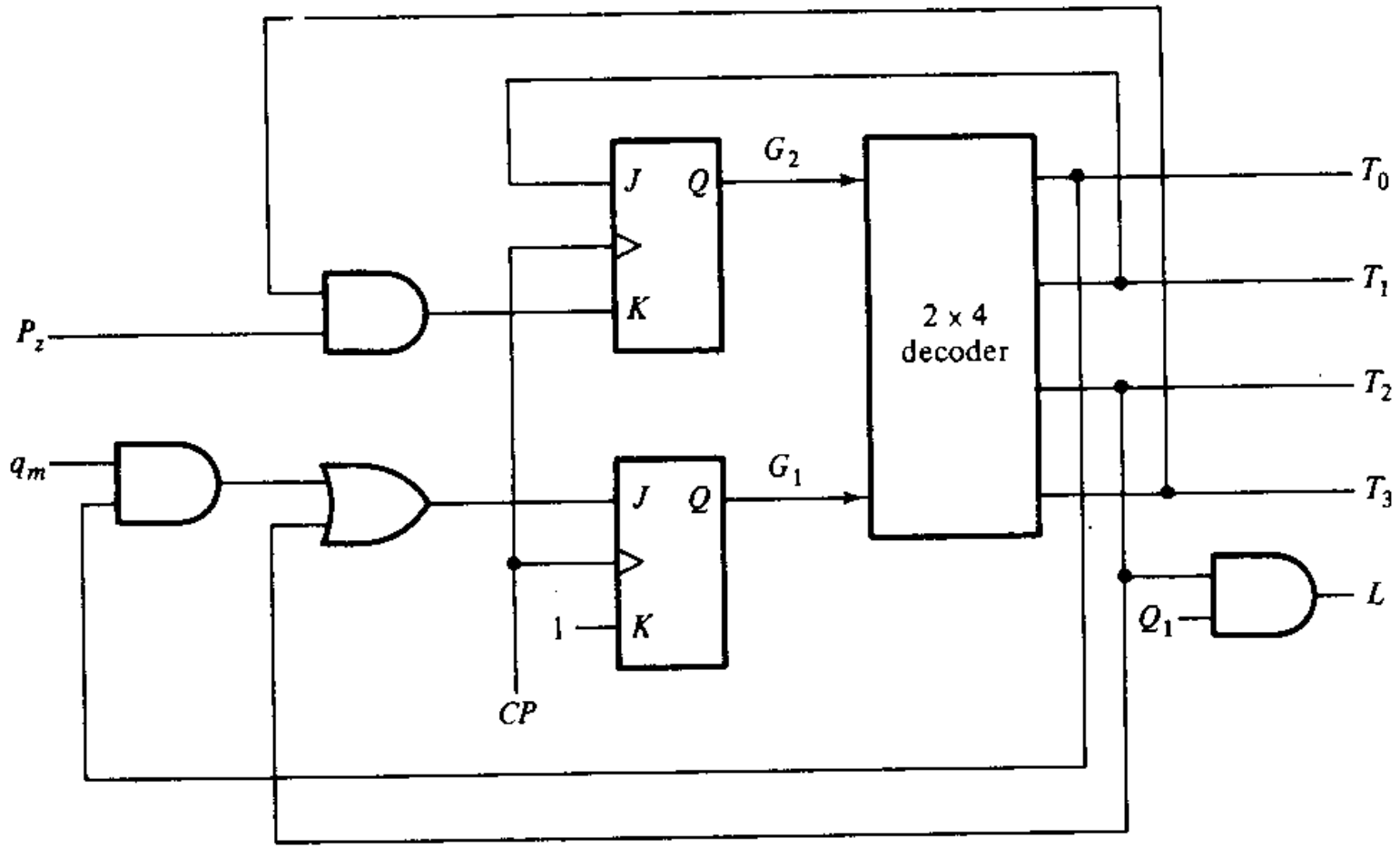
$$JG_2 = T_1 \qquad KG_2 = T_3 P_z$$
$$JG_1 = T_0 q_m + T_2 \qquad KG_1 = 1$$

(b) Flip-flop input functions

Q Output		Inputs	
Present State	Next State	J_n	K_n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Hard-Wired Control - Example 2 (Contd.)

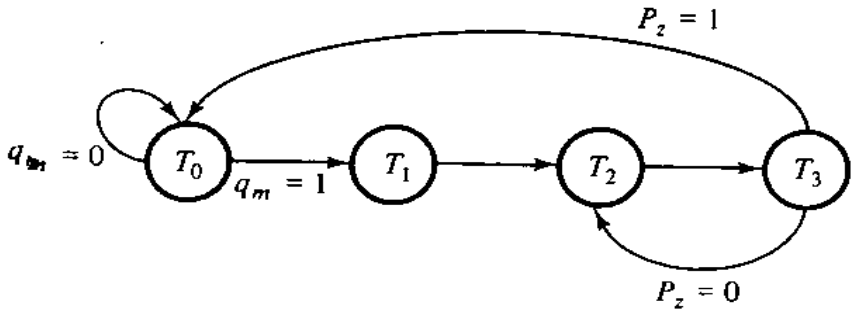
5. Hard-wired Control with JK FF:



(c) Logic diagram

Hard-Wired Control - Example 2 (Contd.)

PLA Control:



(a) State diagram

T_0: Initial state
T_1: $A_s \leftarrow B_s \oplus Q_s, A \leftarrow 0, E \leftarrow 0, P \leftarrow k$
$Q_1 T_2$: $A \leftarrow A + B, E \leftarrow C_{out}$
T_2: $P \leftarrow P - 1$
T_3: $AQ \leftarrow \text{shr}\, EAQ, E \leftarrow 0$

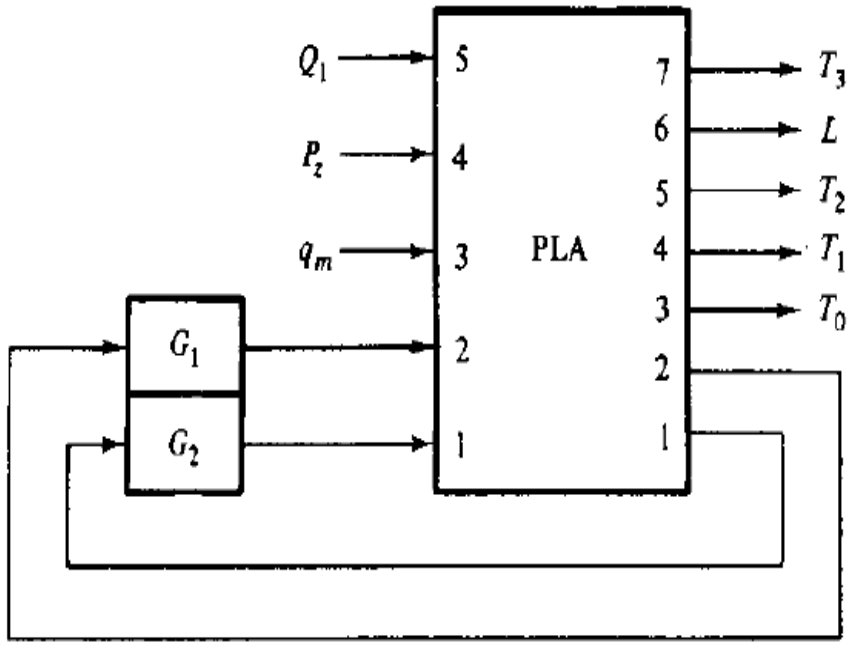
q_m Multiply
$P_z = 1$ if $P = 0$
$P_z = 0$ if $P \neq 0$

TABLE 10-6 State table for control circuit

Present state		Inputs			Next state		Outputs				
G_2	G_1	q_m	P_z	Q_1	G_2	G_1	T_0	T_1	T_2	L	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

Hard-Wired Control - Example 2 (Contd.)

PLA Control:



(a) Block diagram

Product term	Inputs 1	2	3	4	5	Outputs 1	2	3	4	5	6	7	Comments
1	0	0	0	–	–	–	–	1	–	–	–	–	$T_0 = 1$, $q_m = 0$
2	0	0	1	–	–	–	1	1	–	–	–	–	$T_0 = 1$, $q_m = 1$
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	–	$T_2 = 1$, $L = 1$, $Q_1 = 1$
6	1	1	–	0	–	1	–	–	–	–	–	1	$T_3 = 1$, $P_z = 0$
7	1	1	–	1	–	–	–	–	–	–	–	1	$T_3 = 1$, $P_z = 1$

(b) PLA program table

Figure 10-18 PLA control for binary multiplier

Best of Luck!