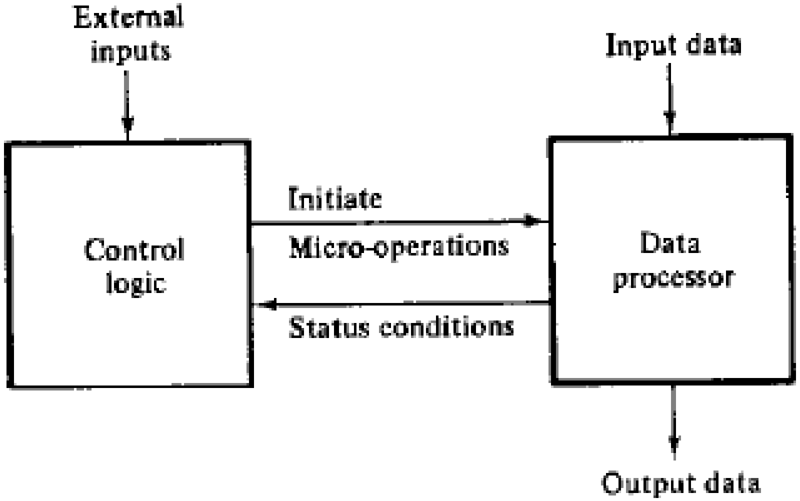
# Control logic design

Module 6



- The control logic initiates all microoperations in data processor
- To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence
- Done using sequential circuit whose internal states dictate the control functions of the system
- Ie, at any time, state of sequential control initiates a set of microoperations

# Control organization

1. Hardwired control
   1. One flipflop per stage method
   2. Sequence register and decoder method

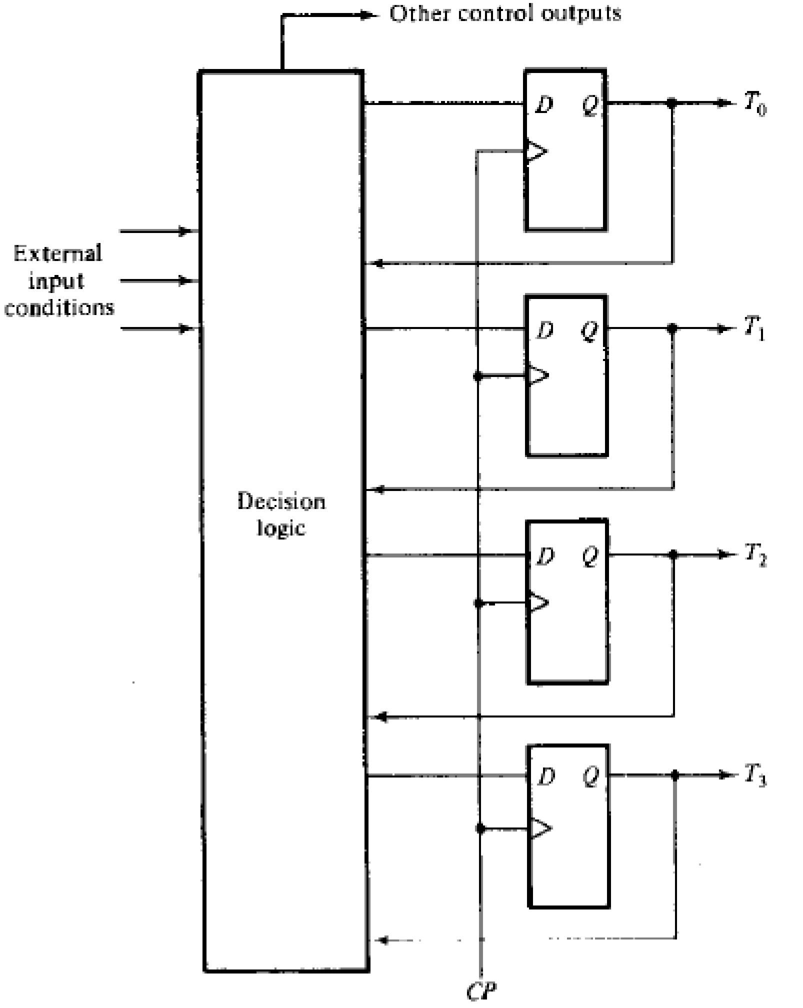
2. PLA control
3. Microprogrammed control

# HARD WIRED CONTROL

# 1. One flip flop per stage

- One flip flop set at a time
- Single bit propagates  under the control of design logic
- Each represents a state
- Transition: function of

  input and present Ti.

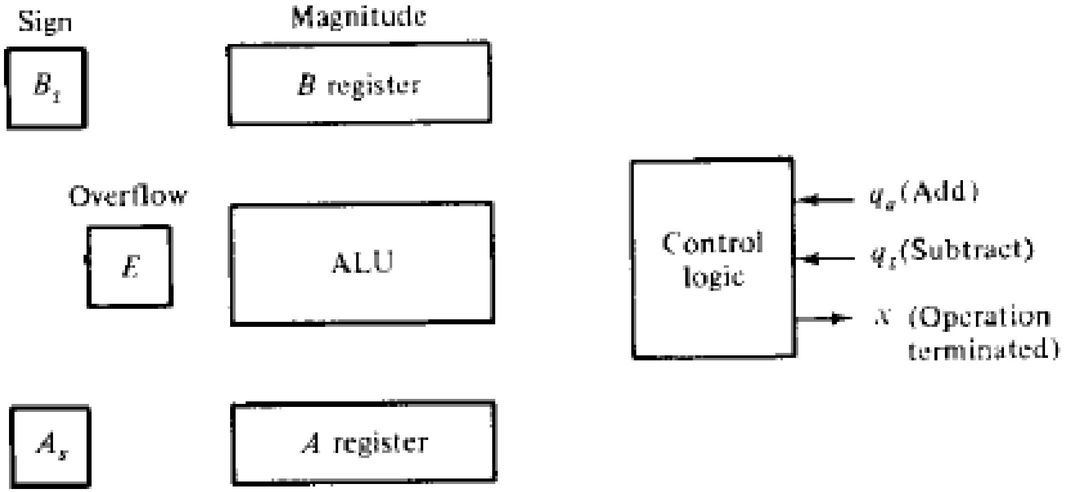


# Example[Hardwired control-example1]

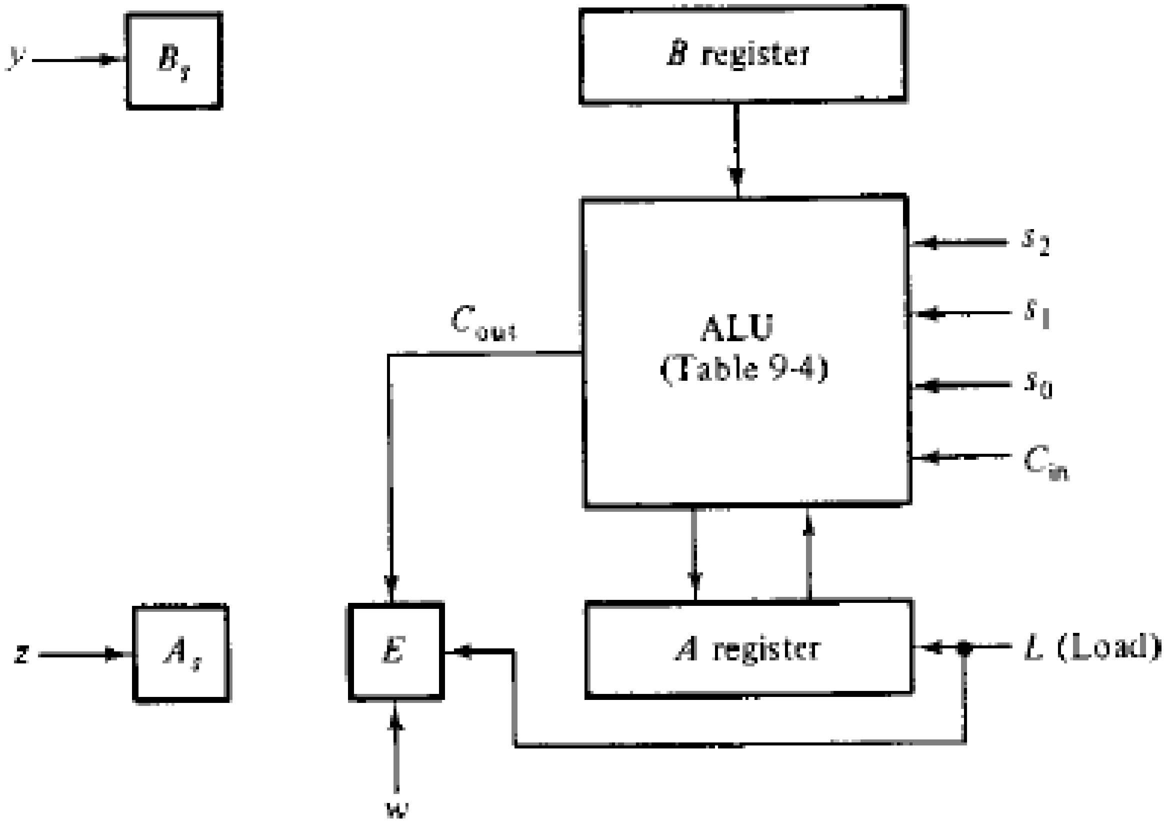
- Design carried out with 5 steps:
    1. Problem statement
    2. Initial equipment configuration
    3. Algorithm formulation
    4. Data processor part specification
    5. Design of control logic

- Problem statement: Addition and subtraction of binary fixed point numbers, negative numbers in sign's 2 complement form

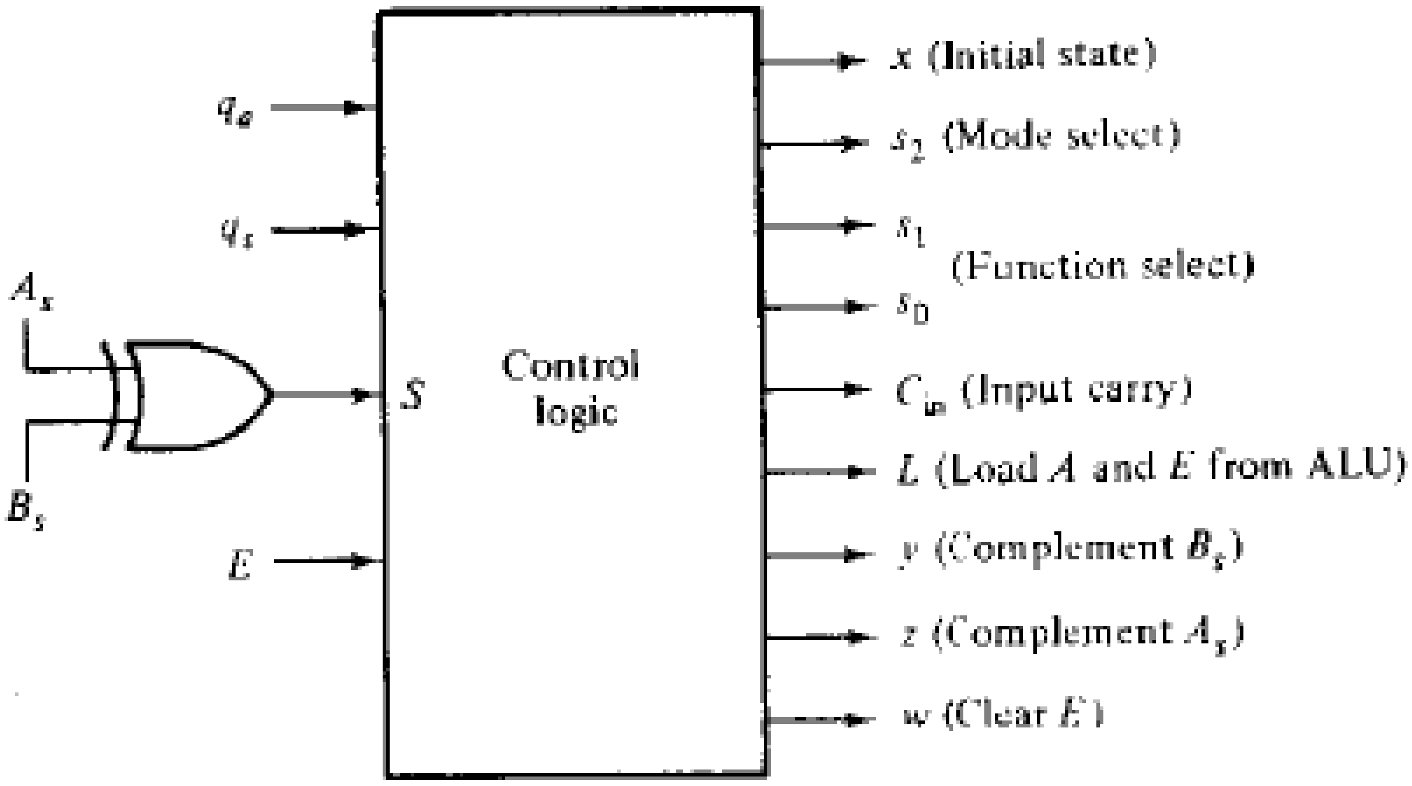
- Equipment configuration:
  - Registers
  - A,B: numbers to be added/ subtracted stored in these reg
  - $A_s, B_s$: sign bits
  - E: 1 bit register storing overflow info
  - Input signals to control logic
  - $q_a$ and $q_b$: add/subtract operation to be performed
  - Output signal
  - x : indicates end of operation



- Data processor part specification

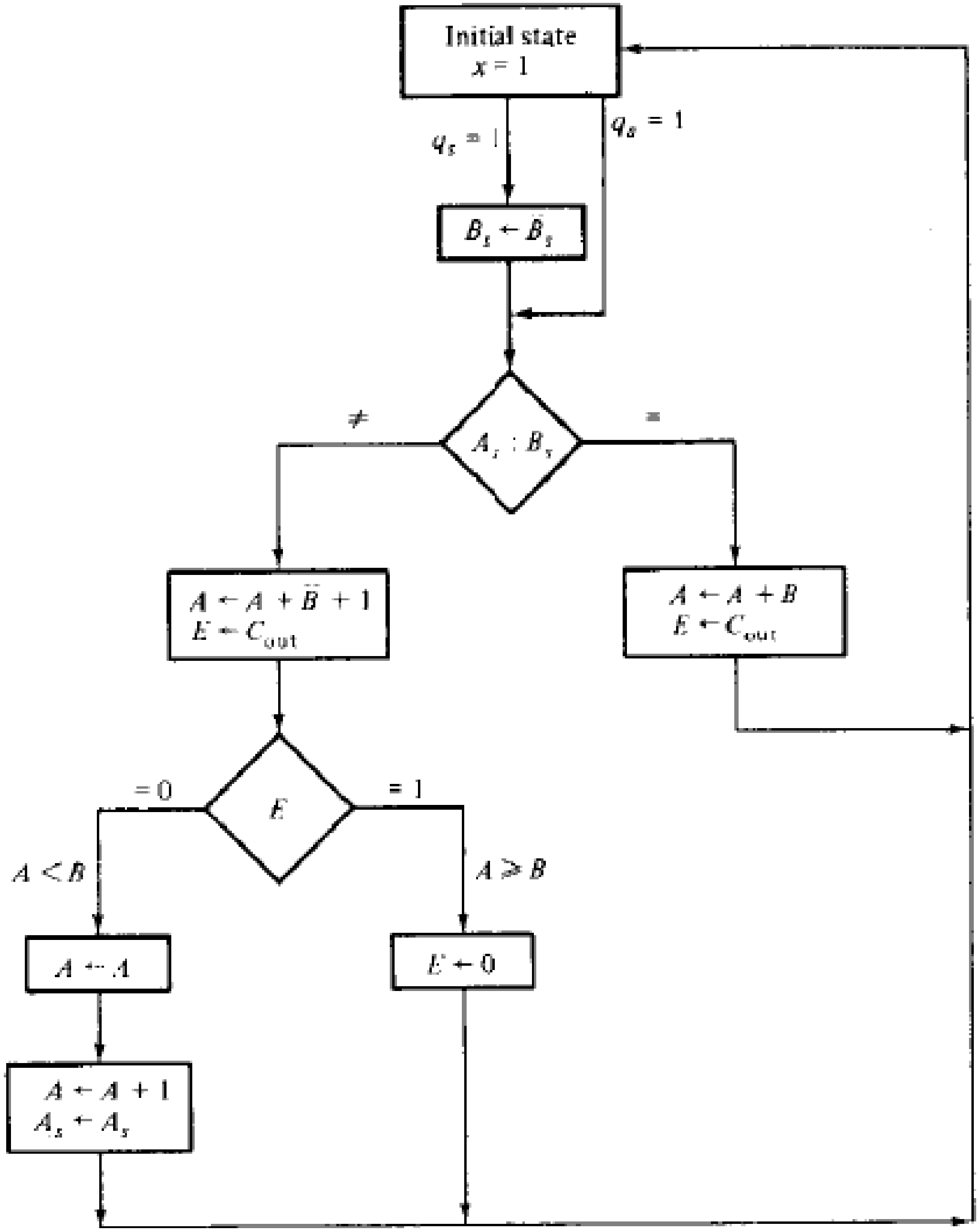


(a) Data processor registers and ALU

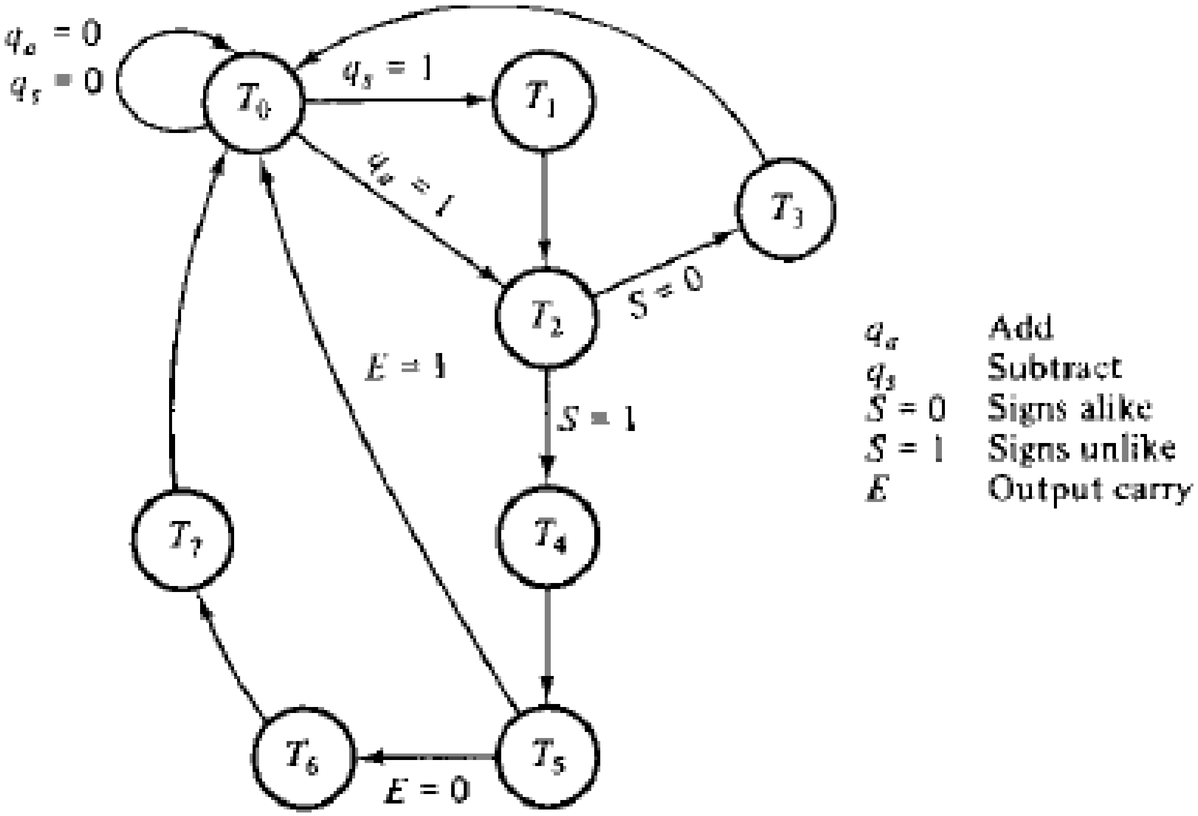


(b) Control block diagram

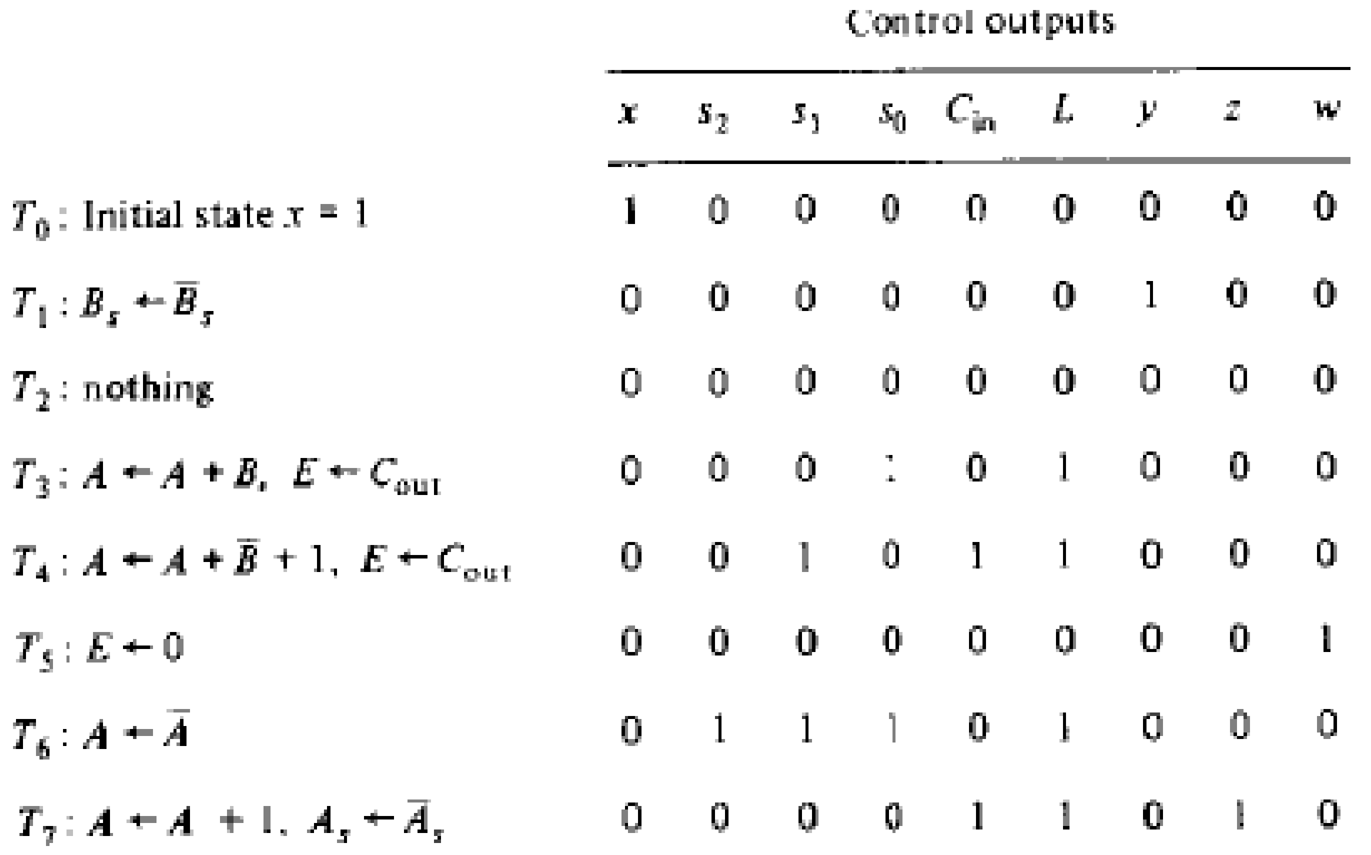
- Algorithm:


Initial state
$x = 1$
$q_s = 1$
$q_a = 1$
$B_s \leftarrow \bar{B}_s$
$A_s : B_s$
$\neq$
$=$
$A \leftarrow A + \bar{B} + 1$
$E \leftarrow C_{out}$
$A \leftarrow A + B$
$E \leftarrow C_{out}$
$E$
$= 0$
$= 1$
$A < B$
$A \geqslant B$
$A \leftarrow \bar{A}$
$E \leftarrow 0$
$A \leftarrow A + 1$
$A_s \leftarrow \bar{A}_s$

- Control state diagram



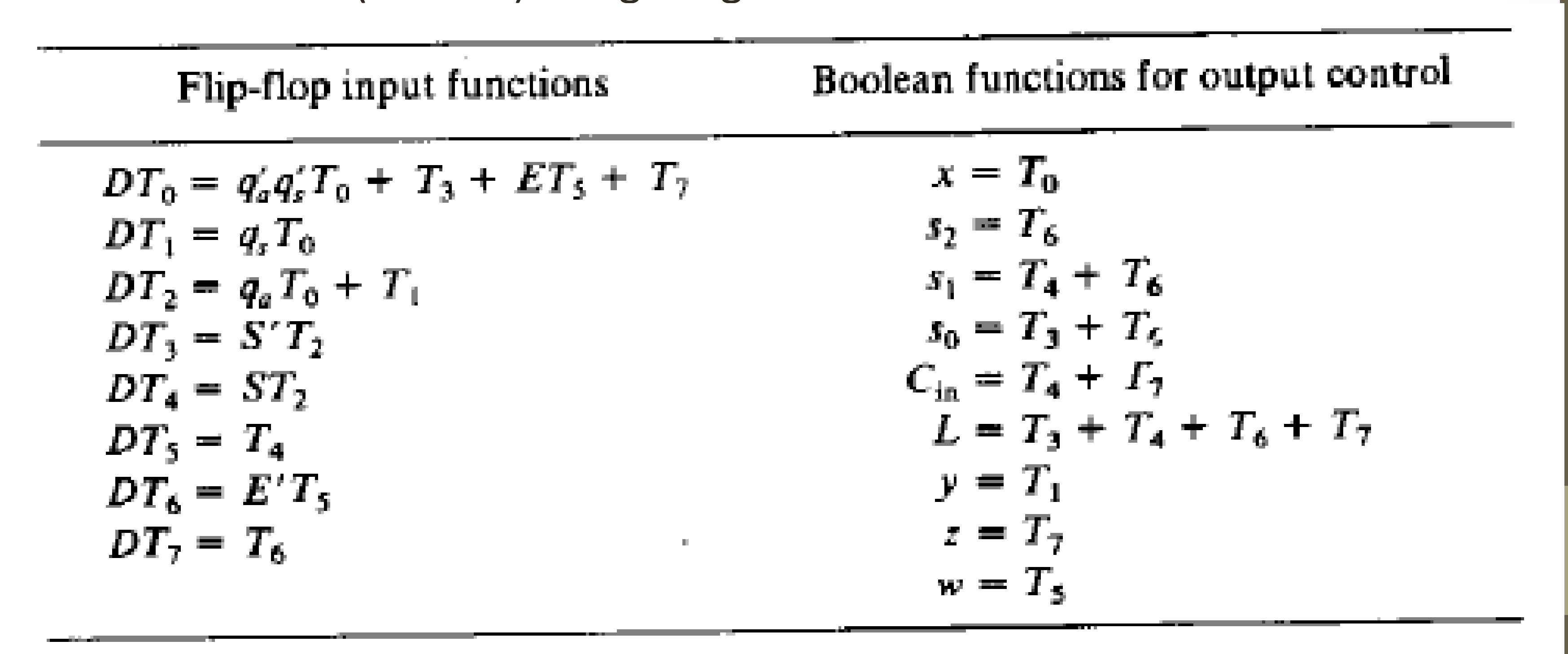
(a) State diagram



| | 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

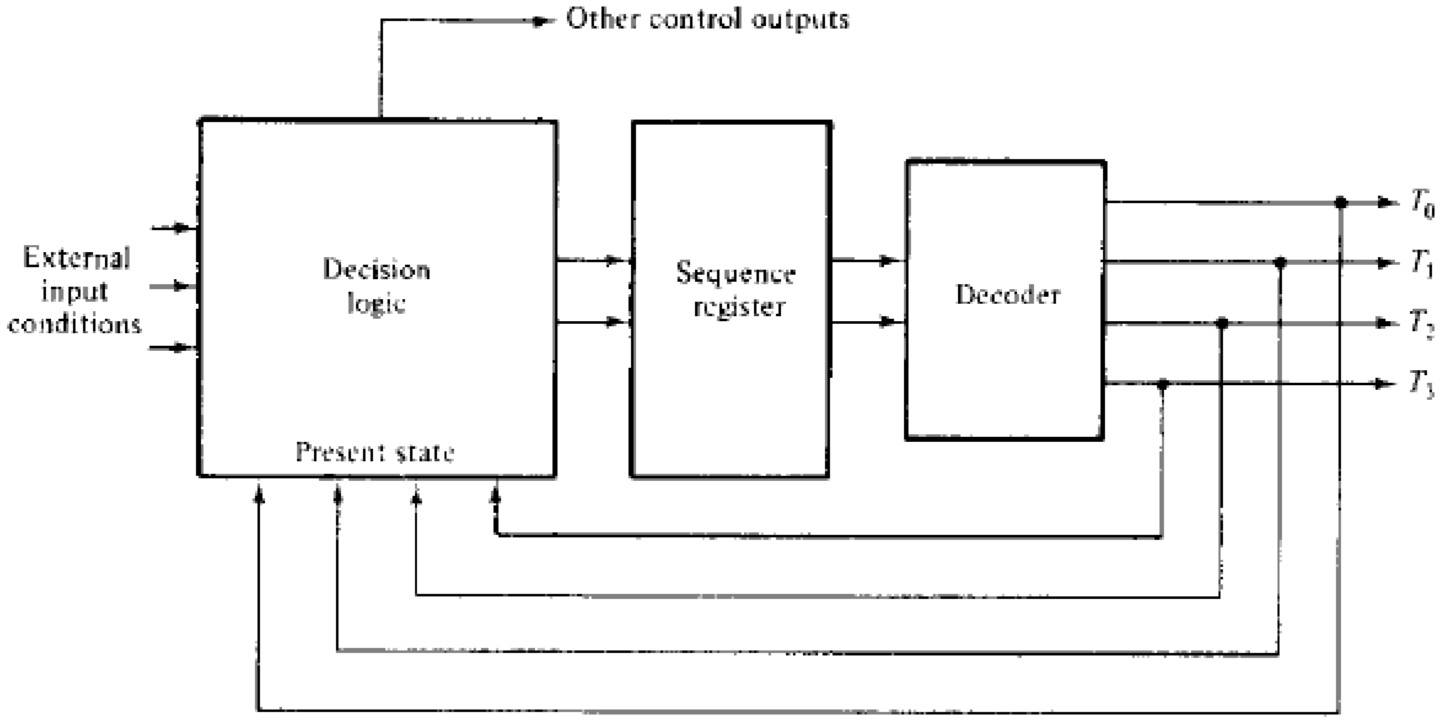
- Design of hard wired control
  - 8 states are required as we have seen.
  - So 8 D flipflops used with the following input functions(implemented in decision logic)
  - At a time, any of the state will be active and corresponding D flipflop will be activated
  - The output signals are generated according to the booean function(in table) using OR gate.



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

# 2. Sequence register and decoder method

- Register: to sequence control states
- Register decoded to provide one output for each state



# Example[Hardwired control-example2]

- Problem statement: To design an arithmetic circuit that multiplies two fixed point binary numbers in sign magnitude representation
- Equipment configuration:
  - <u>Registers</u>
  - B: to store multiplicand, Q-multiplier, A-partial product
  - $A_s, B_s, Q_s$: sign bits
  - P: counter(no of bits in the multiplier, decremented after formation of each partial product)
  - <u>Input signals to control logic</u>
  - $q_m$: operation to be performed
  - <u>Output signal</u>
  - $Q_1$: rightmost bit in register Q

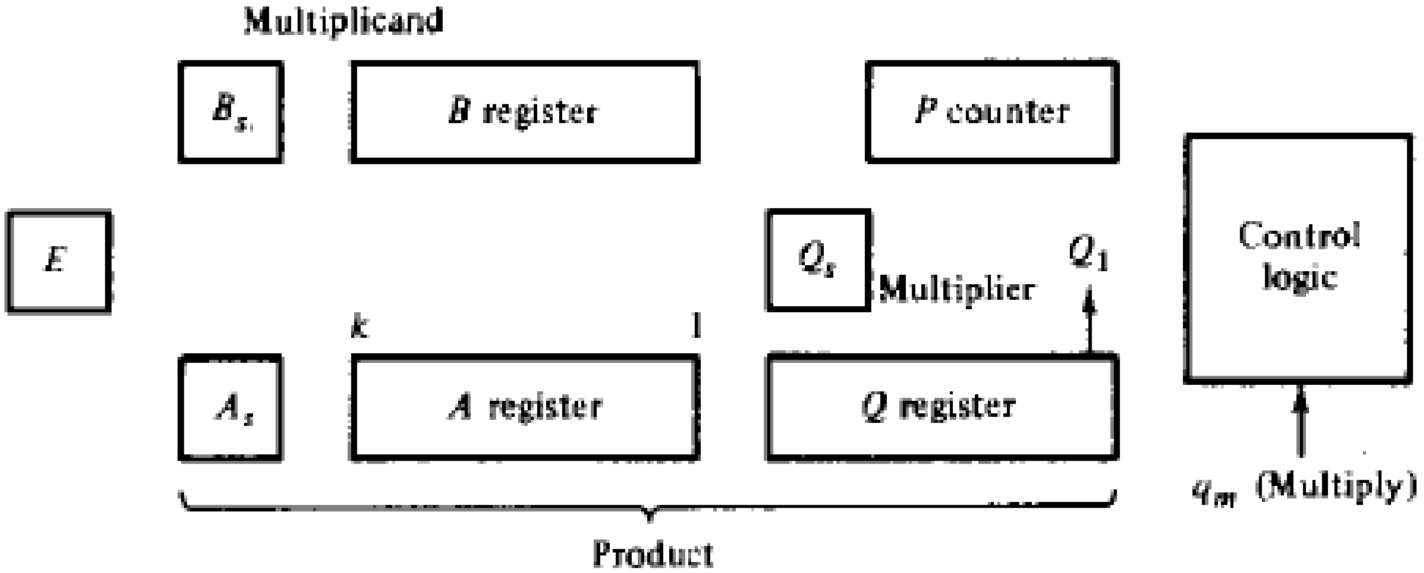
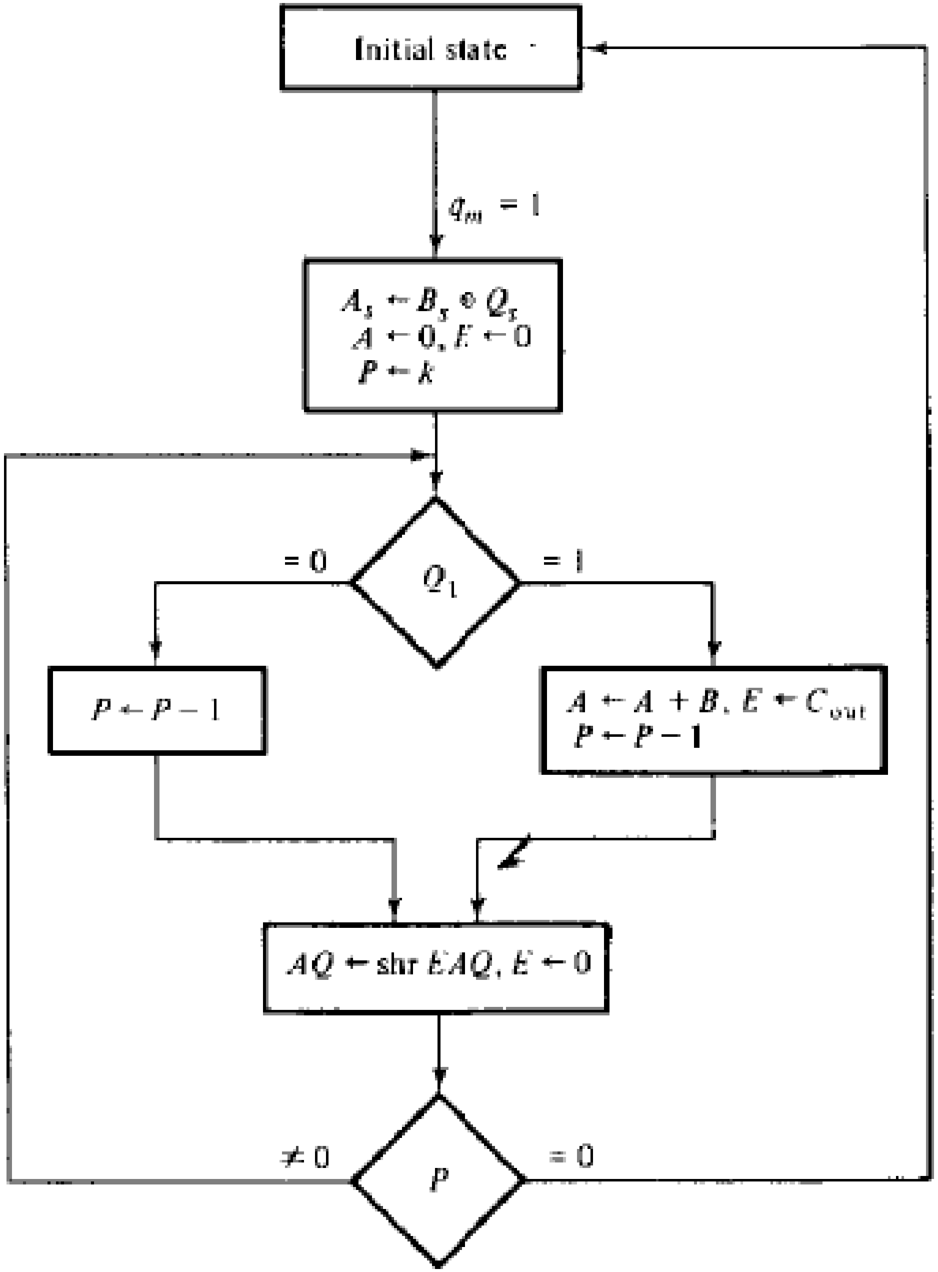
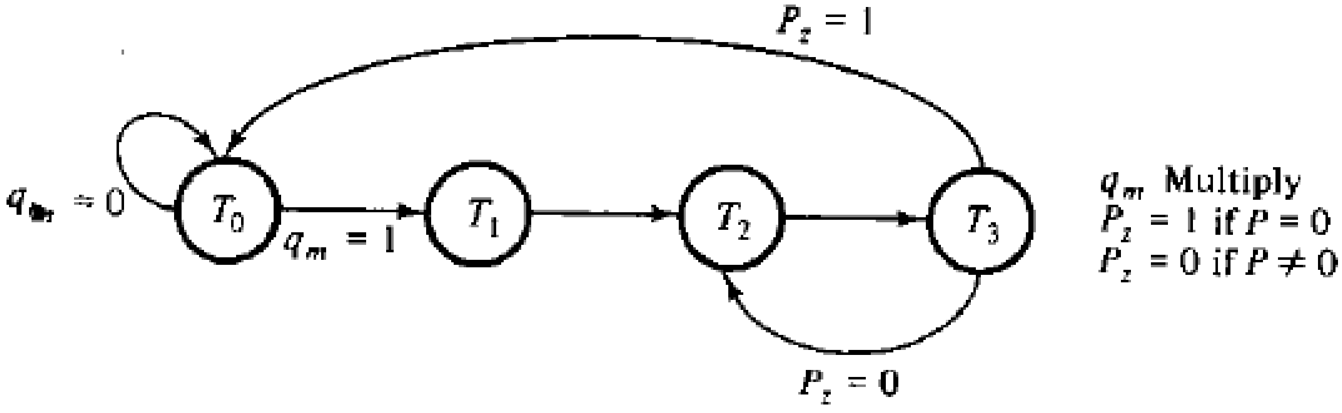


Figure 10-13 Registers for binary multiplier


Initial state
$q_m = 1$
$A_s \leftarrow B_s \oplus Q_s$
$A \leftarrow 0, E \leftarrow 0$
$P \leftarrow k$
$Q_1$
$= 0$
$= 1$
$P \leftarrow P - 1$
$A \leftarrow A + B, E \leftarrow C_{out}$
$P \leftarrow P - 1$
$AQ \leftarrow \text{shr } EAQ, E \leftarrow 0$
$P$
$\neq 0$
$= 0$



(a) State diagram

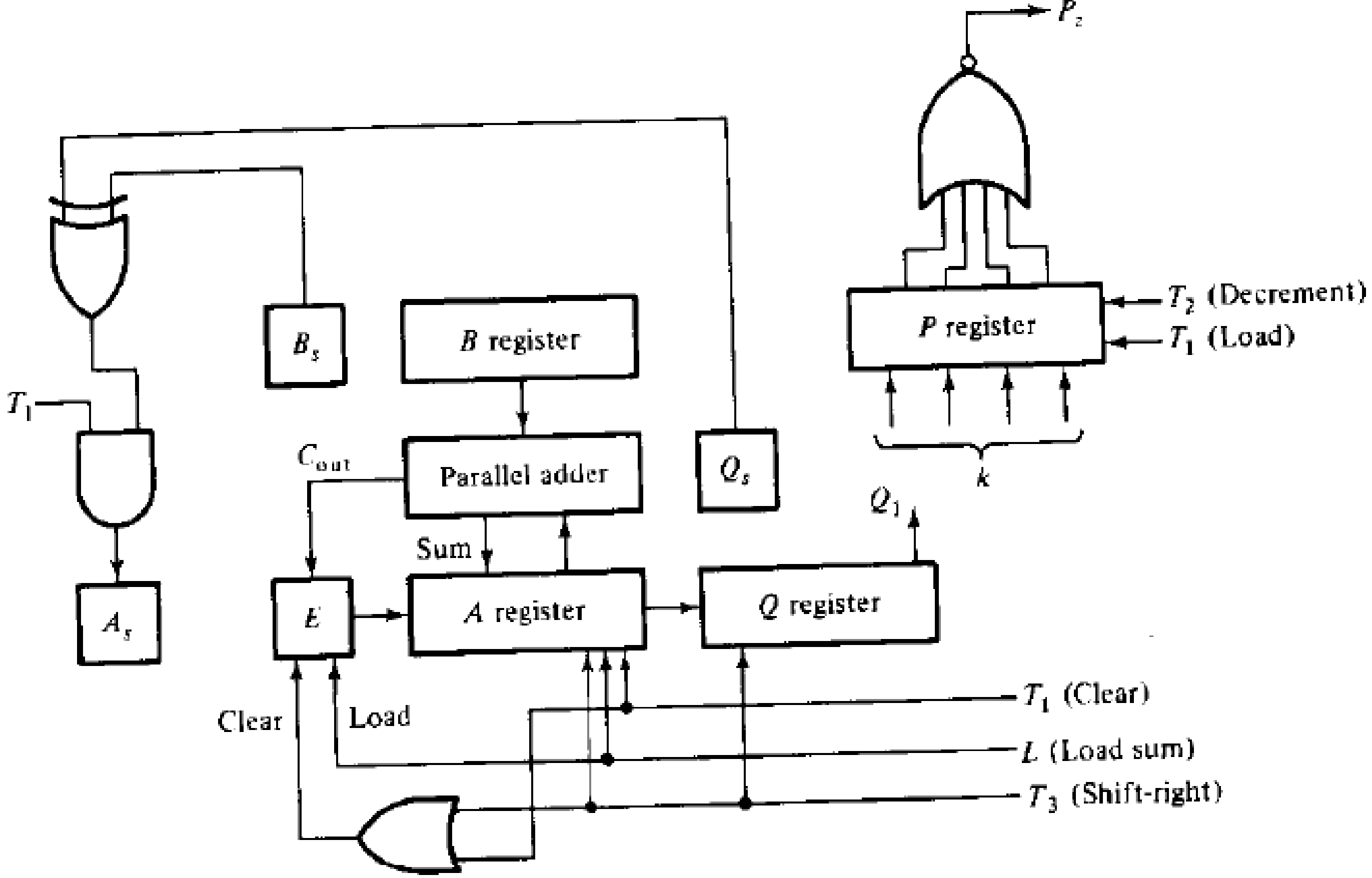
$T_0$: Initial state

$T_1$: $A_s \leftarrow B_s \oplus Q_s, A \leftarrow 0, E \leftarrow 0, P \leftarrow k$

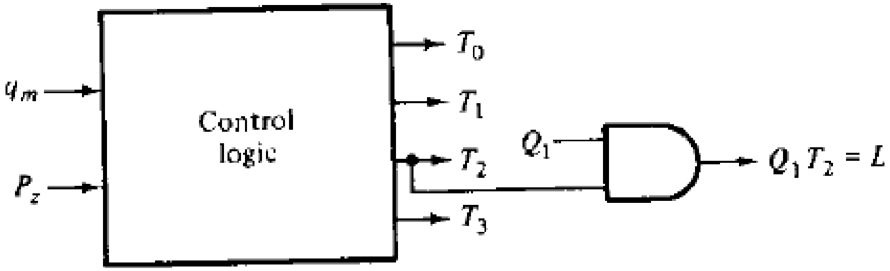
$Q_1 T_2$: $A \leftarrow A + B, E \leftarrow C_{out}$

$T_2$: $P \leftarrow P - 1$

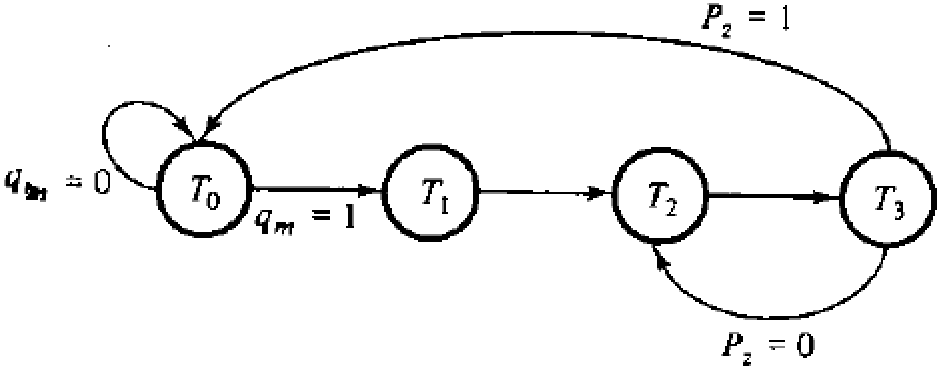
$T_3$: $AQ \leftarrow \text{shr } EAQ, E \leftarrow 0$

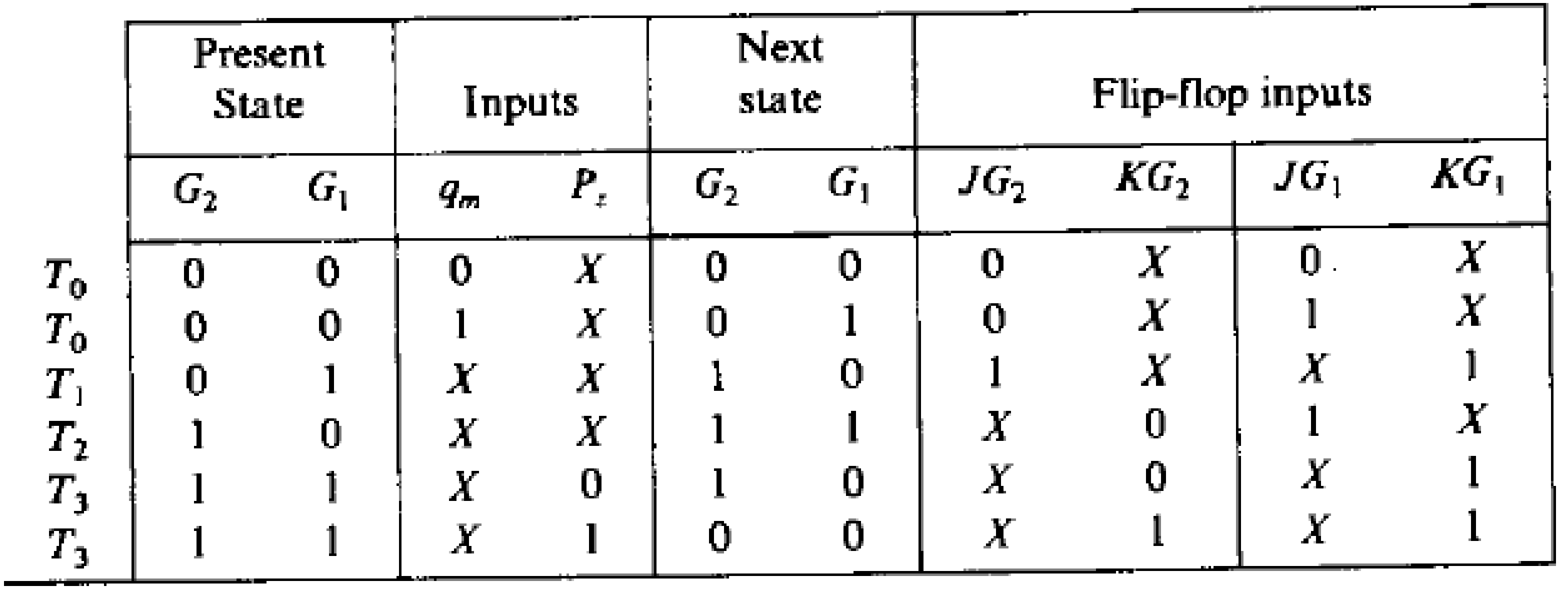


- Output from control logic:
  - **T0**-indicates that system in initial state
  - **T1**-loads sign to As and k into P
  - **T2**-decrements P,
    - If Q=1, L generated- loads A (sum)and E
  - **T3**-Shifts A,Q, clears E



- Design of hard wired control
  - 2 JK flipflops and 1 2x4 decoder can be used as there are 4 states

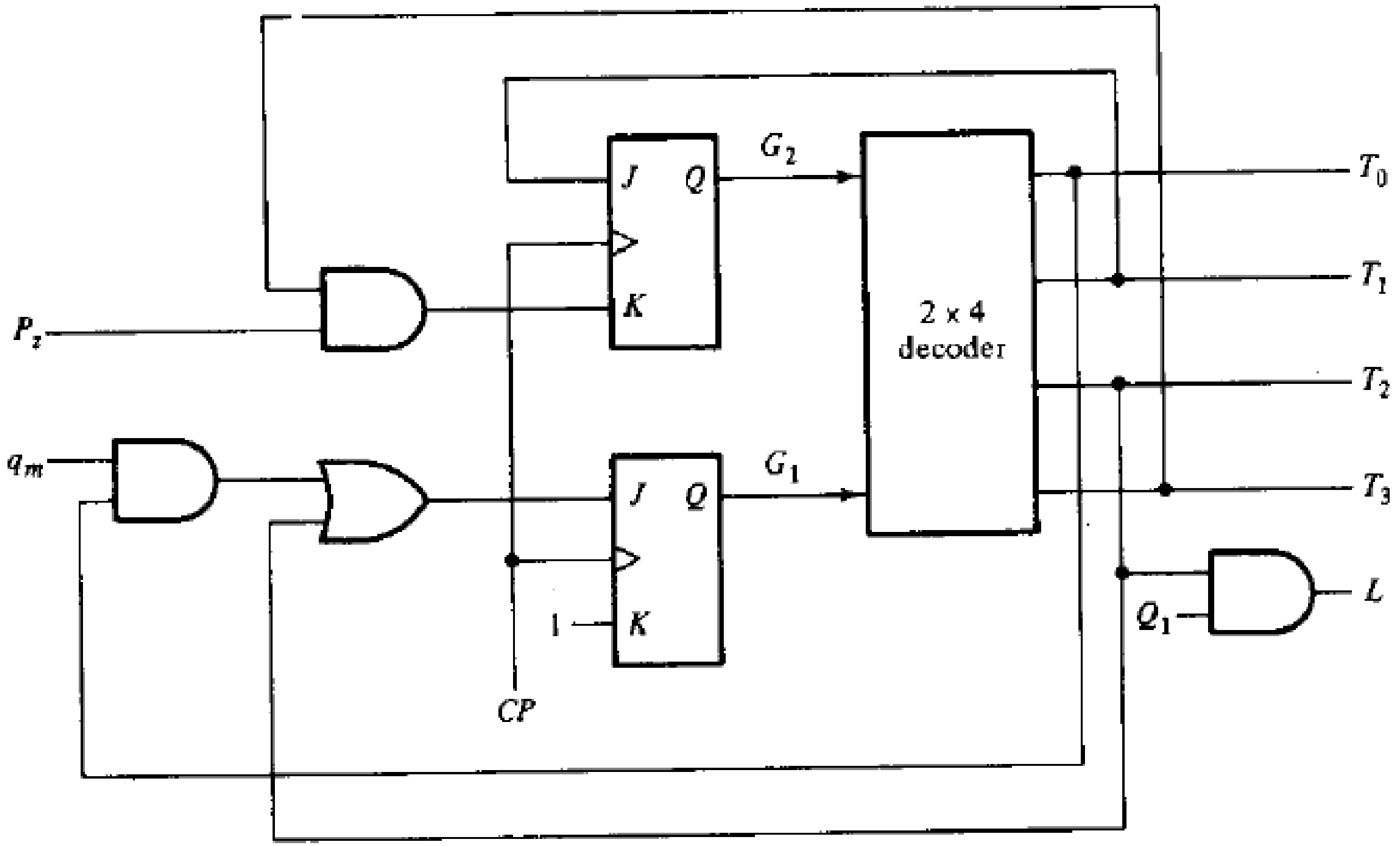




| | Present State | | Inputs | | Next state | | Flip-flop inputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $G_2$ | $G_1$ | $q_m$ | $P_z$ | $G_2$ | $G_1$ | $JG_2$ | $KG_2$ | $JG_1$ | $KG_1$ |
| $T_0$ | 0 | 0 | 0 | X | 0 | 0 | 0 | X | 0 | X |
| $T_0$ | 0 | 0 | 1 | X | 0 | 1 | 0 | X | 1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

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


G₂
J
Q
K
2 x 4
decoder
T₀
T₁
T₂
T₃
P_z
q_m
G₁
J
Q
1
K
CP
Q₁
L

# 3. Programmable Logic Array

- PLA replaces decoder and decision logic circuits
- Internal paths inside PLA are programmed according to program table
- Similar to sequence register and decoder method, except that all combinational circuits are implemented with PLA

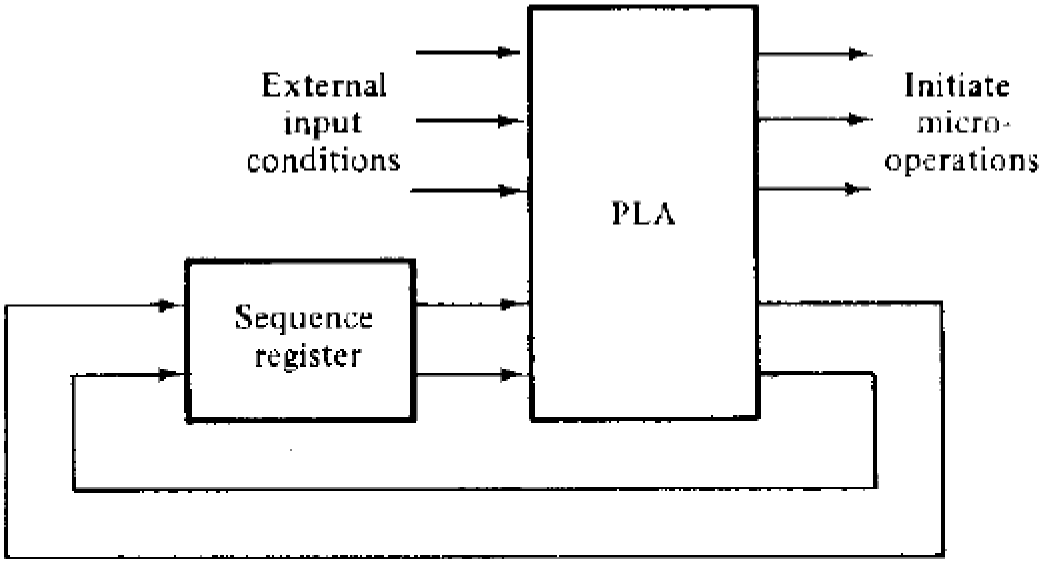
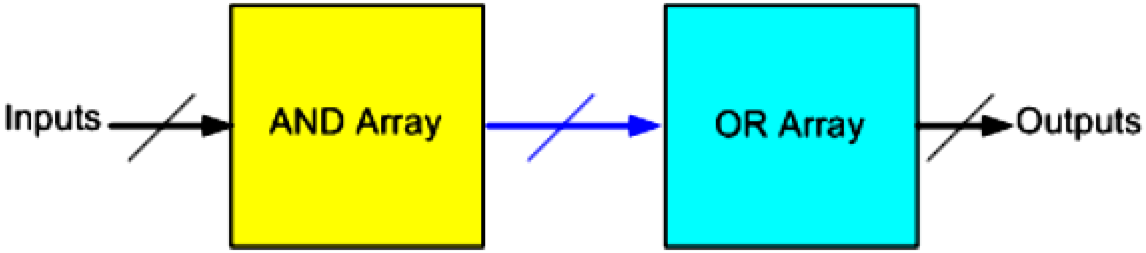


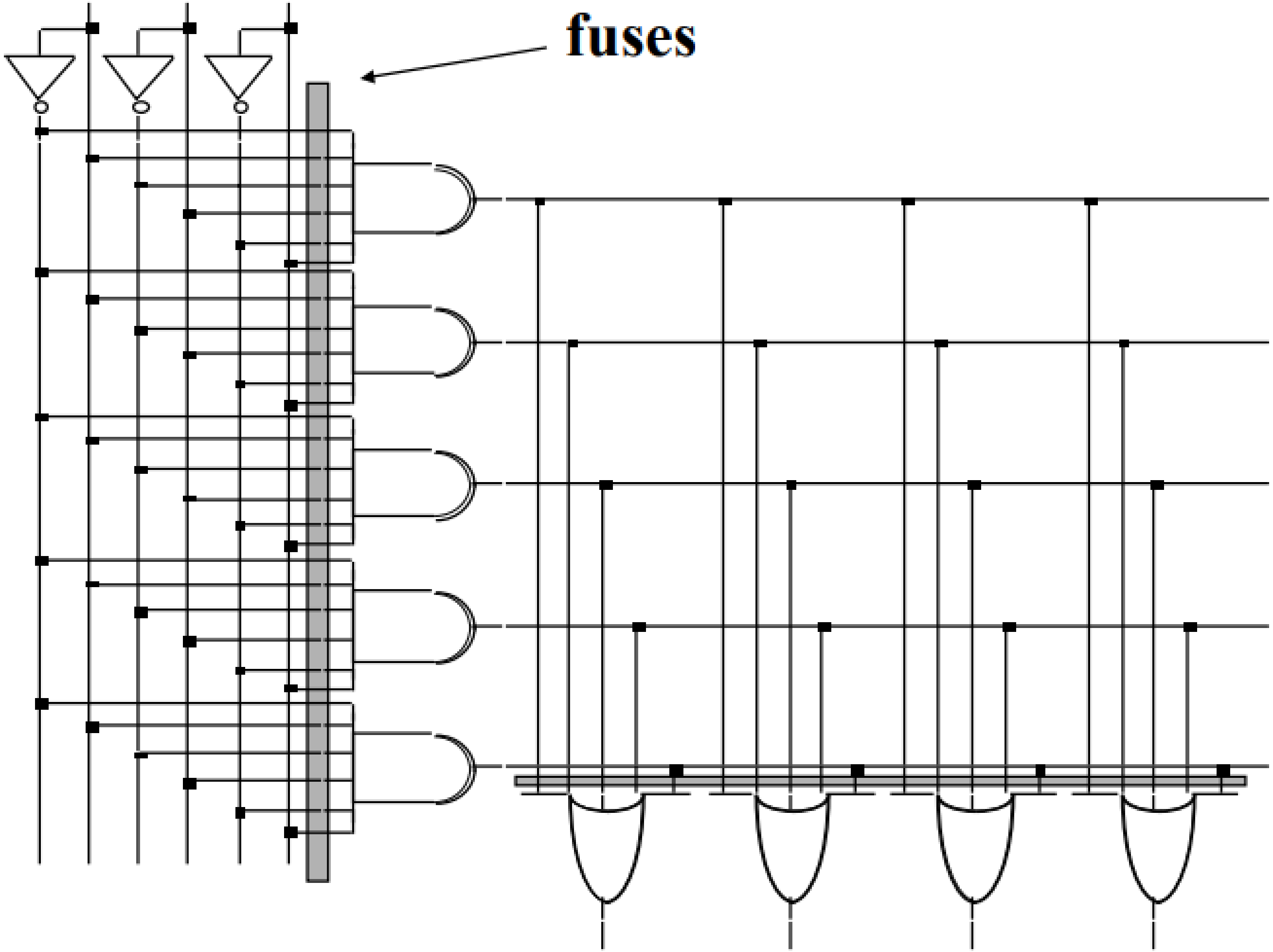
**Figure 10-4** PLA control logic

# PLA basics



- The AND and OR gates inside the PLA are initially fabricated with the links (fuses) among them

- The specific Boolean functions are implemented in sum of products form by opening appropriate links and leaving the desired connections.


fuses

# Example(section 10.7)

- Design steps:

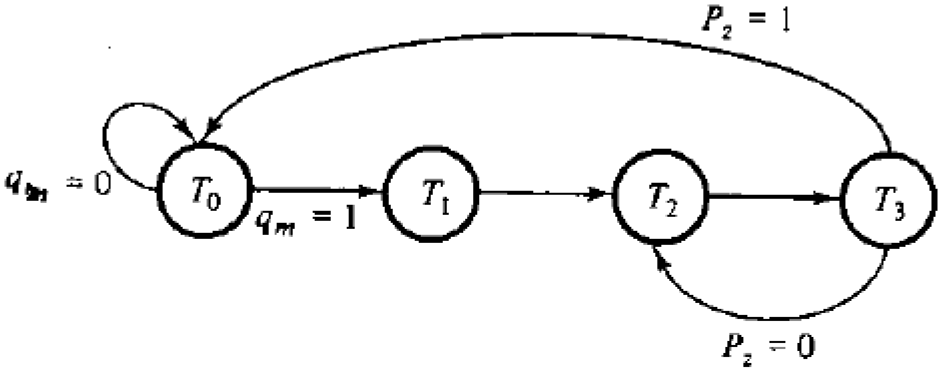
1. Obtain state table for controller

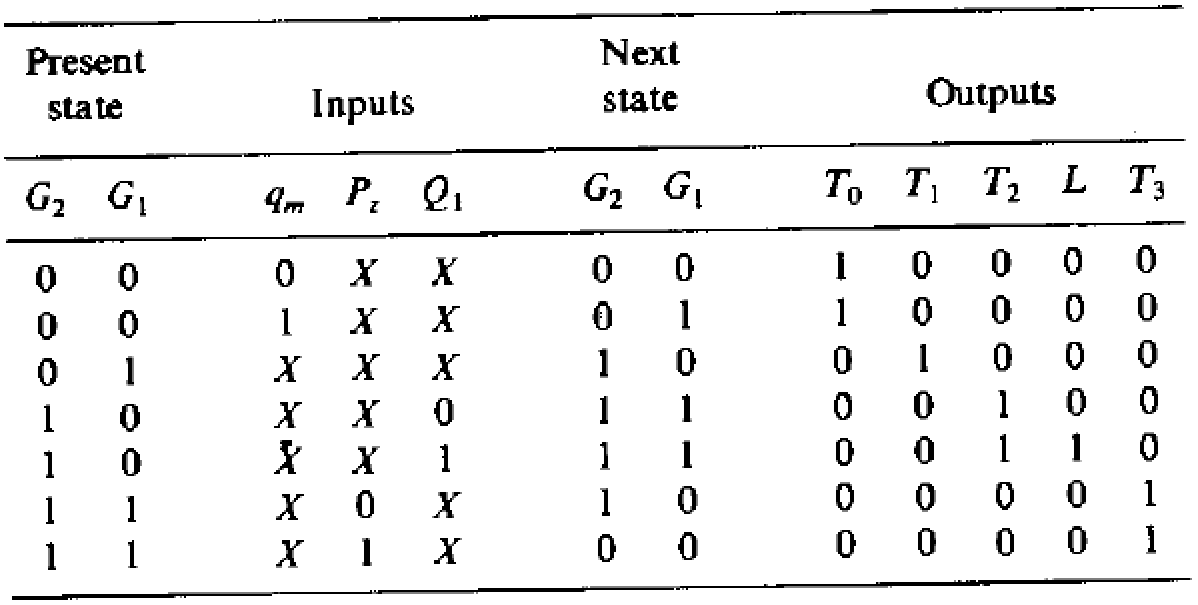
(No of states determine the number of flipflops for sequence reg)

2. Connect PLA to sequence register and to input and output variables

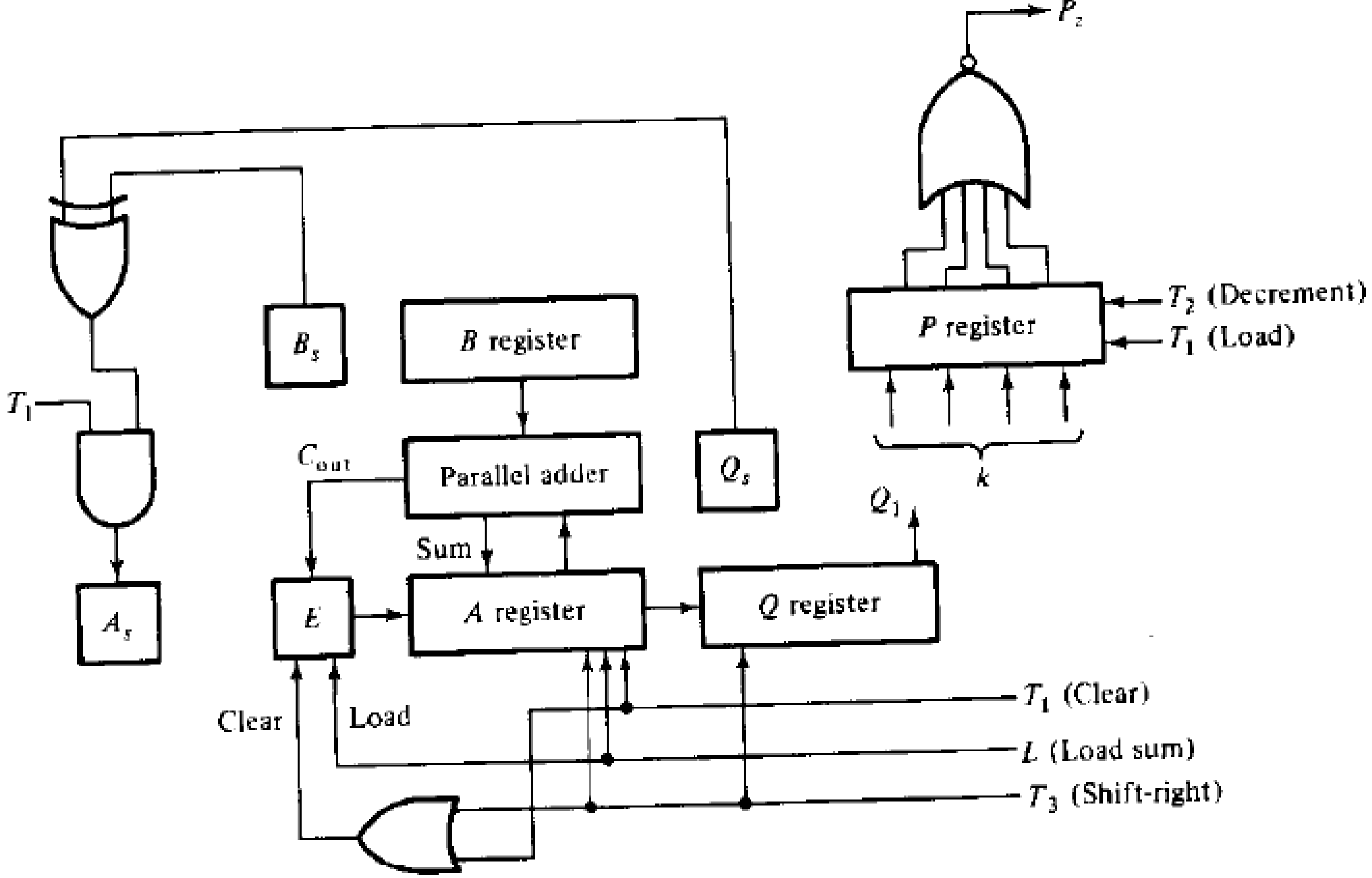
PLA program table obtained from state table

- Consider the example of binary multiplier

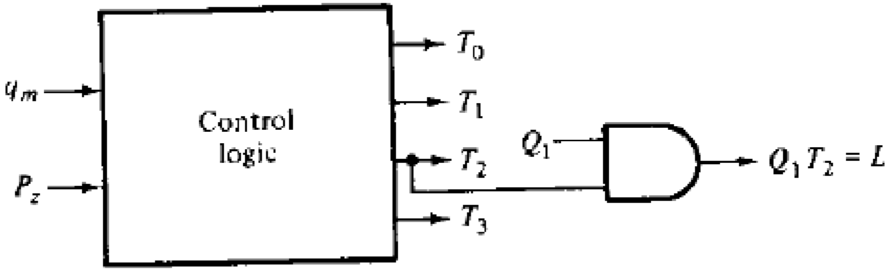


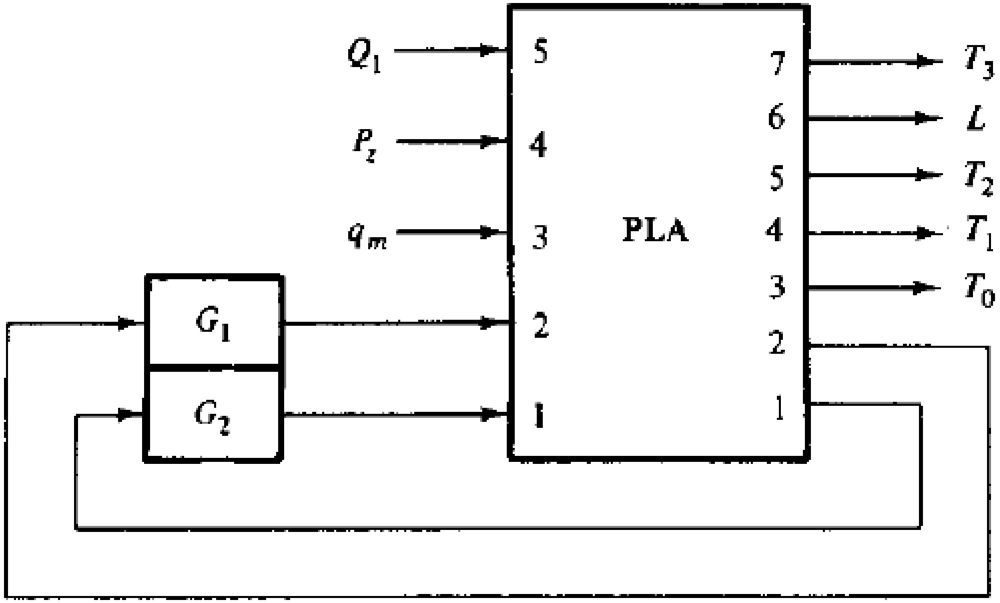


| Present state | | Inputs | | | Next state | | Outputs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $G_2$ | $G_1$ | $q_m$ | $P_z$ | $Q_1$ | $G_2$ | $G_1$ | $T_0$ | $T_1$ | $T_2$ | $L$ | $T_3$ |
| 0 | 0 | 0 | X | X | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | X | X | 0 | 1 | 1 | 0 | 0 | 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 |



- Output from control logic:
  - **T0**-indicates that system in initial state
  - **T1**-loads sign to As and k into P
  - **T2**-decrements P,
    - If Q=1, L generated- loads A (sum)and E
  - **T3**-Shifts A,Q, clears E





- **G1 and G2** –flipflops(sequence reg)
- **Input to PLA-** value of present state and 3 ext input

| Product term | Inputs |  |  |  |  | Outputs |  |  |  |  |  |  | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
| 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 | $T_3 = 1$, $P_z = 1$ |

# Microprogrammed Control

- Control unit initiate a series of sequential steps of microoperations

- At any time certain operations are to be initiated and others idle

- So represented as series of 1’s and 0’s ->control word

- Microprogrammed-> CU whose control variables are stored in memory

- **Microinstruction**
- **Microprogram**

- **Control memory**

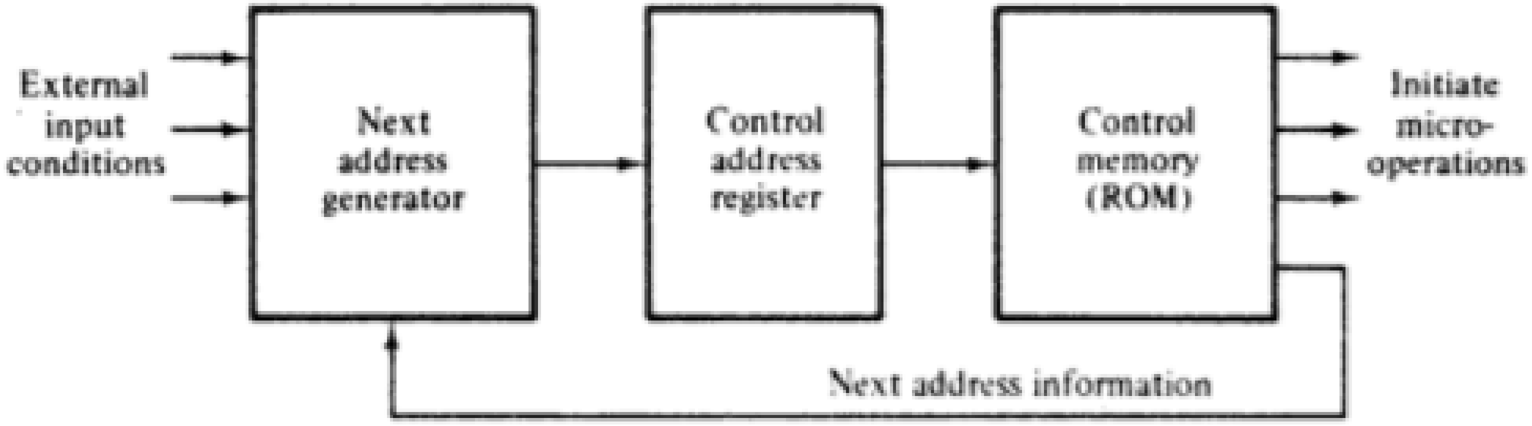
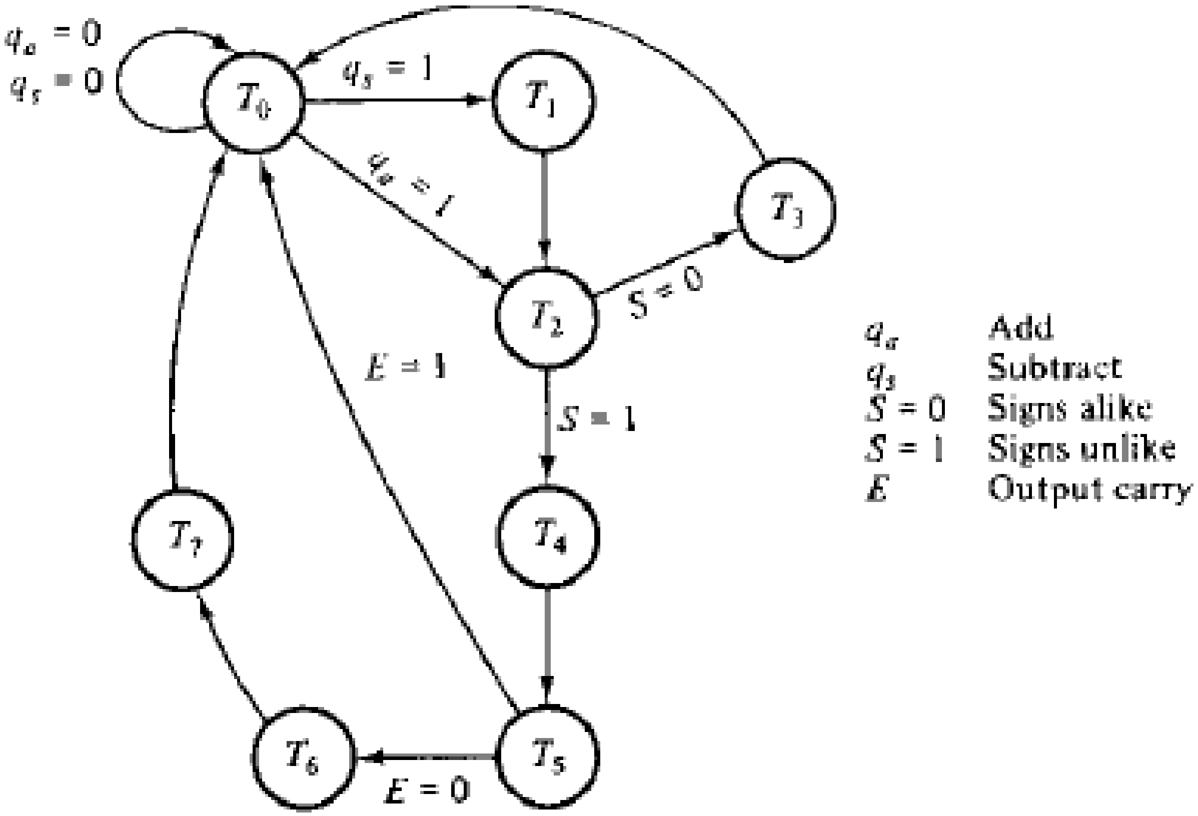


**Figure 10-5** Microprogram control logic

# Example(section 10.4)

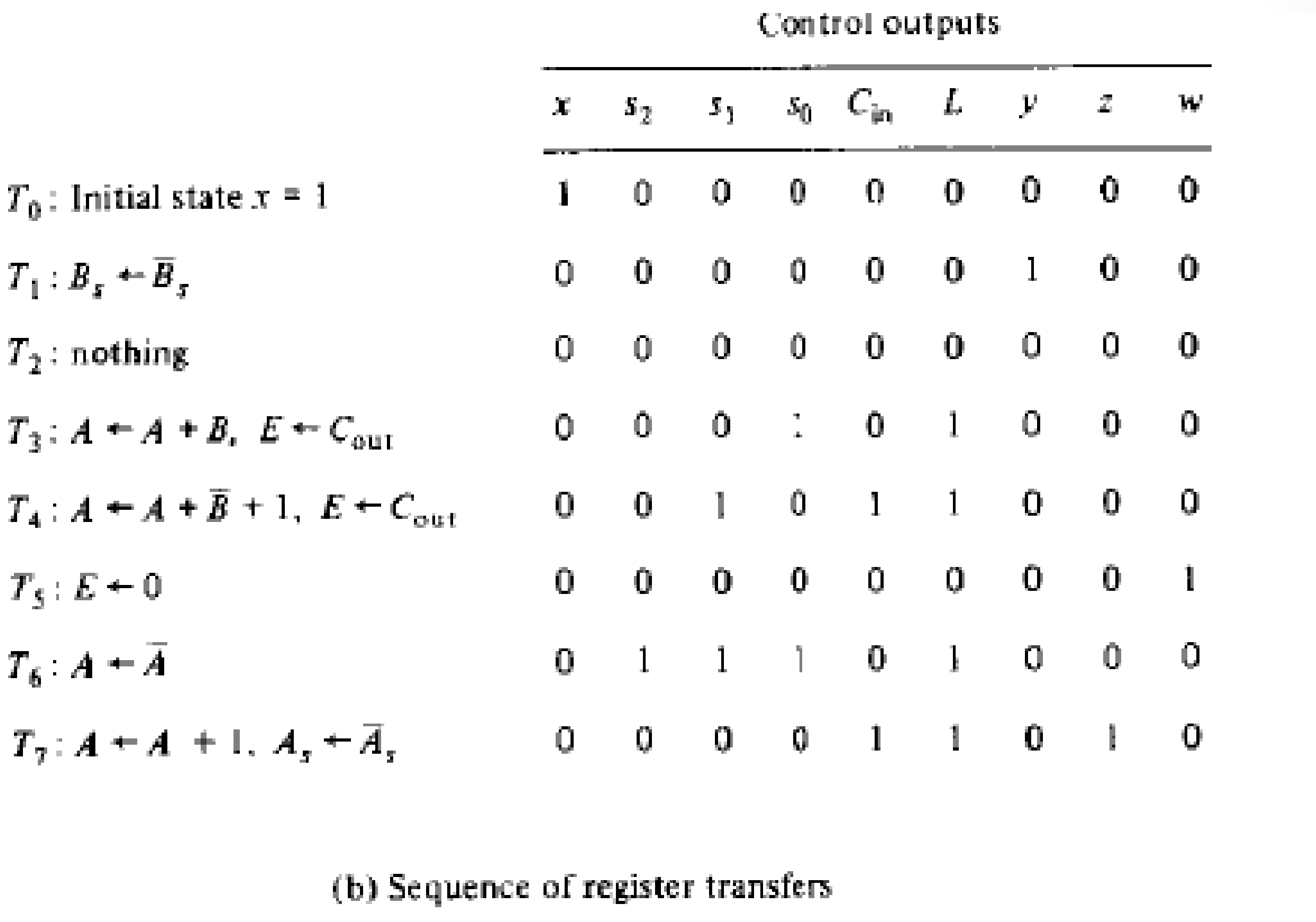
- Problem statement:  To design control logic for sign magnitude adder subtractor using microprogram control
- State in control memory- address of microinstruction

- Control state diagram



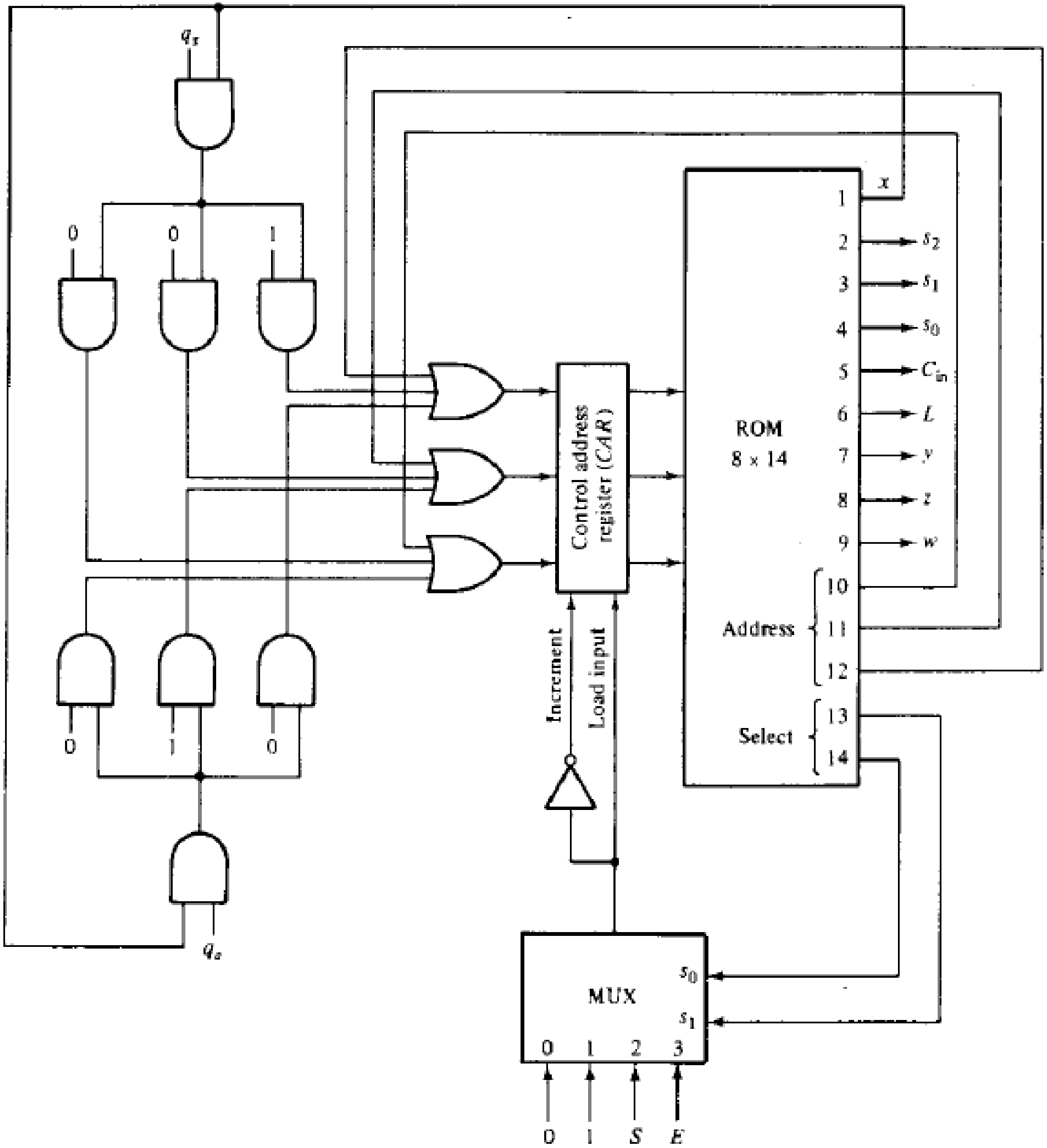
| | |
|---|---|
| $q_a$ | Add |
| $q_s$ | Subtract |
| $S = 0$ | Signs alike |
| $S = 1$ | Signs unlike |
| $E$ | Output carry |

(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


$q_x$
0
0
1
x
1
2
3
4
5
6
7
8
9
10
11
12
13
14
$s_2$
$s_1$
$s_0$
$C_{in}$
$L$
$y$
$z$
$w$
ROM
8 × 14
Control address register (CAR)
Address
Select
Increment
Load input
0
1
0
$q_a$
MUX
$s_0$
$s_1$
0
1
2
3
0
1
S
E

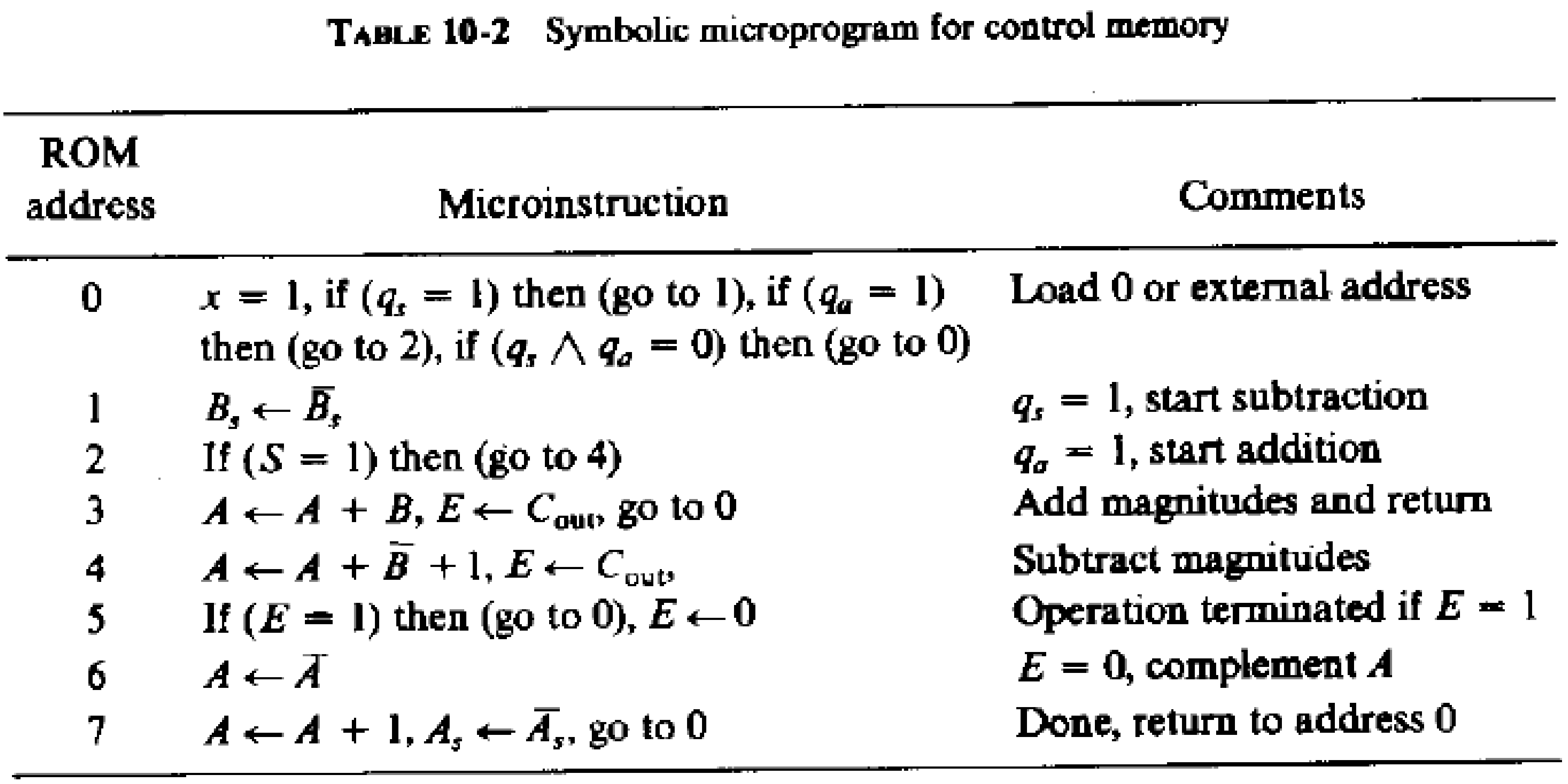


TABLE 10-2 Symbolic microprogram for control memory

| 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 | $B_s \leftarrow \bar{B}_s$ | $q_s = 1$, 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 | $A \leftarrow A + \bar{B} + 1, E \leftarrow C_{out}$ | Subtract magnitudes |
| 5 | If ($E = 1$) then (go to 0), $E \leftarrow 0$ | Operation terminated if $E = 1$ |
| 6 | $A \leftarrow \bar{A}$ | $E = 0$, complement $A$ |
| 7 | $A \leftarrow A + 1, A_s \leftarrow \bar{A}_s$, go to 0 | Done, return to address 0 |

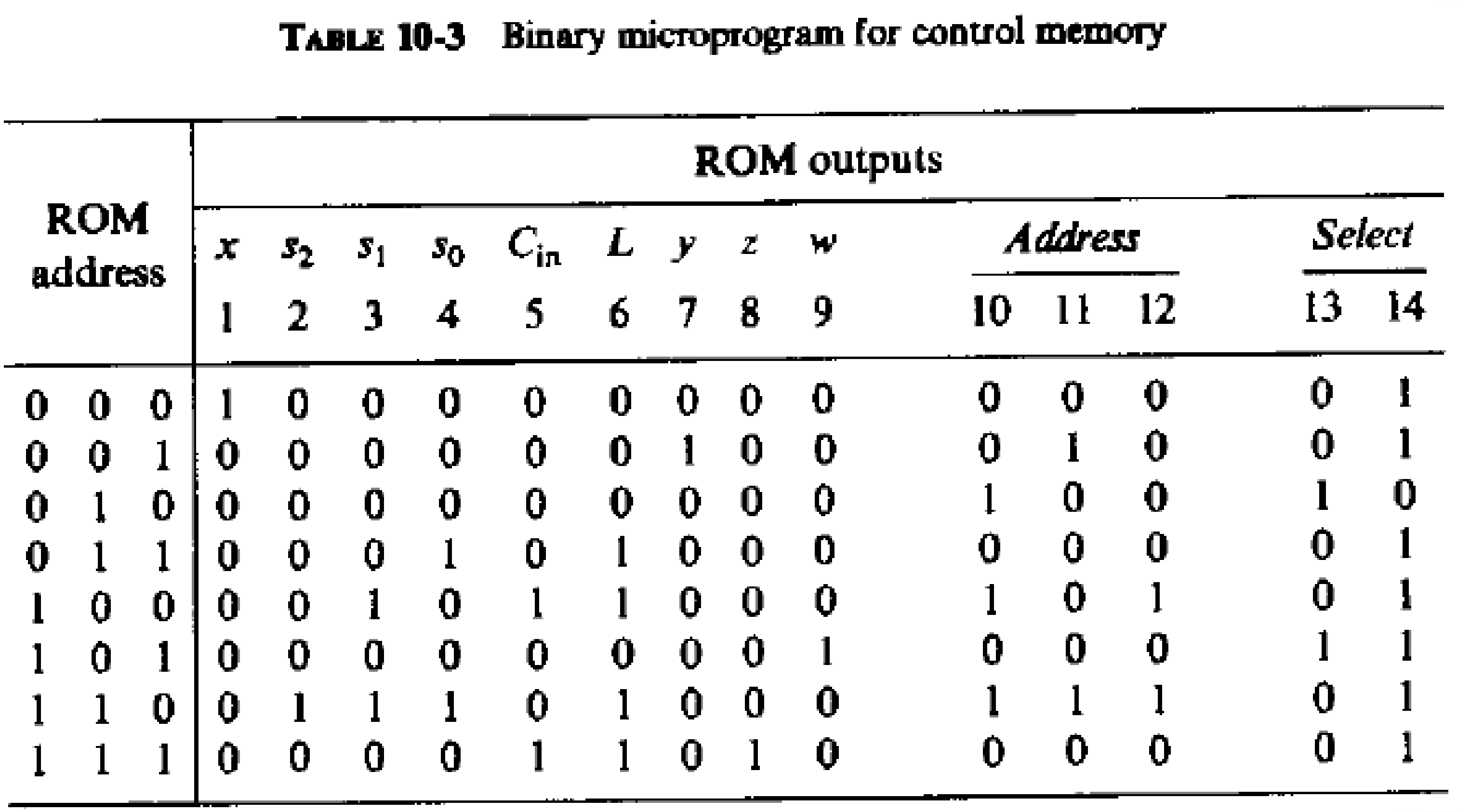
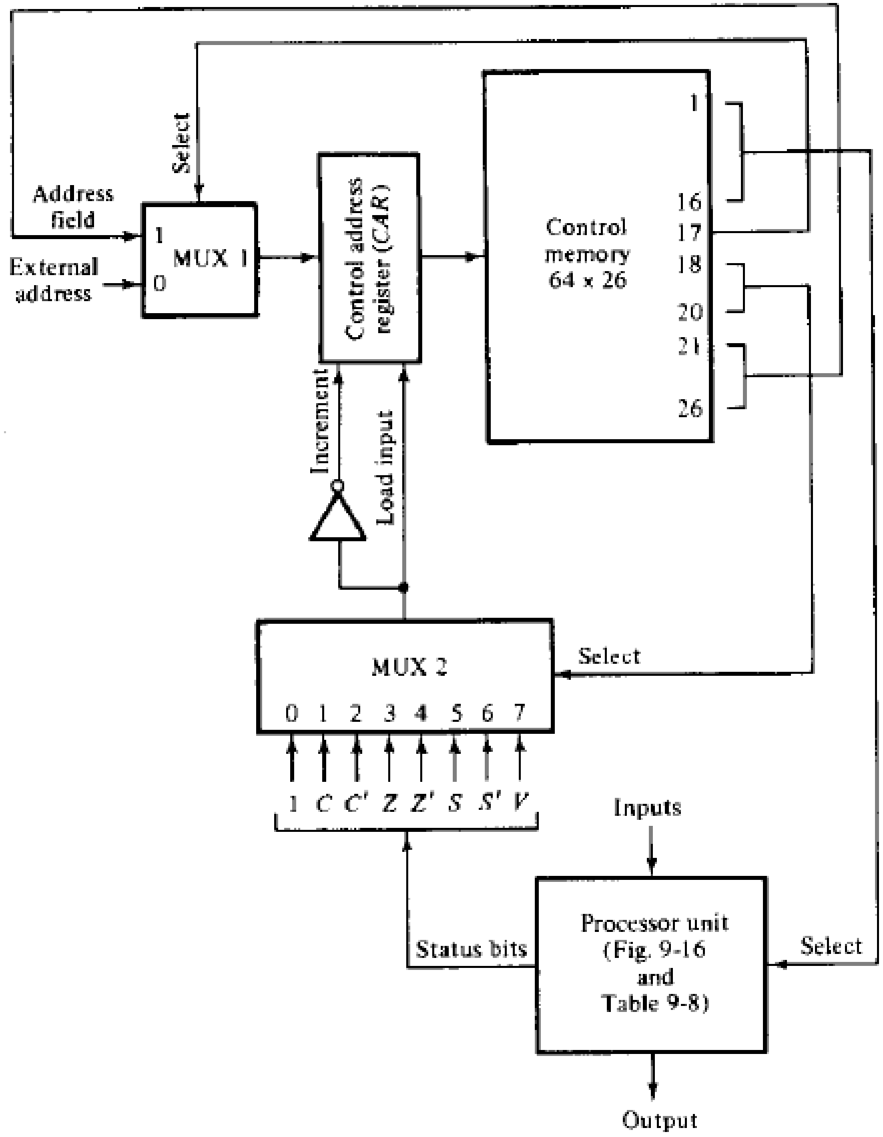


TABLE 10-3 Binary microprogram for control memory

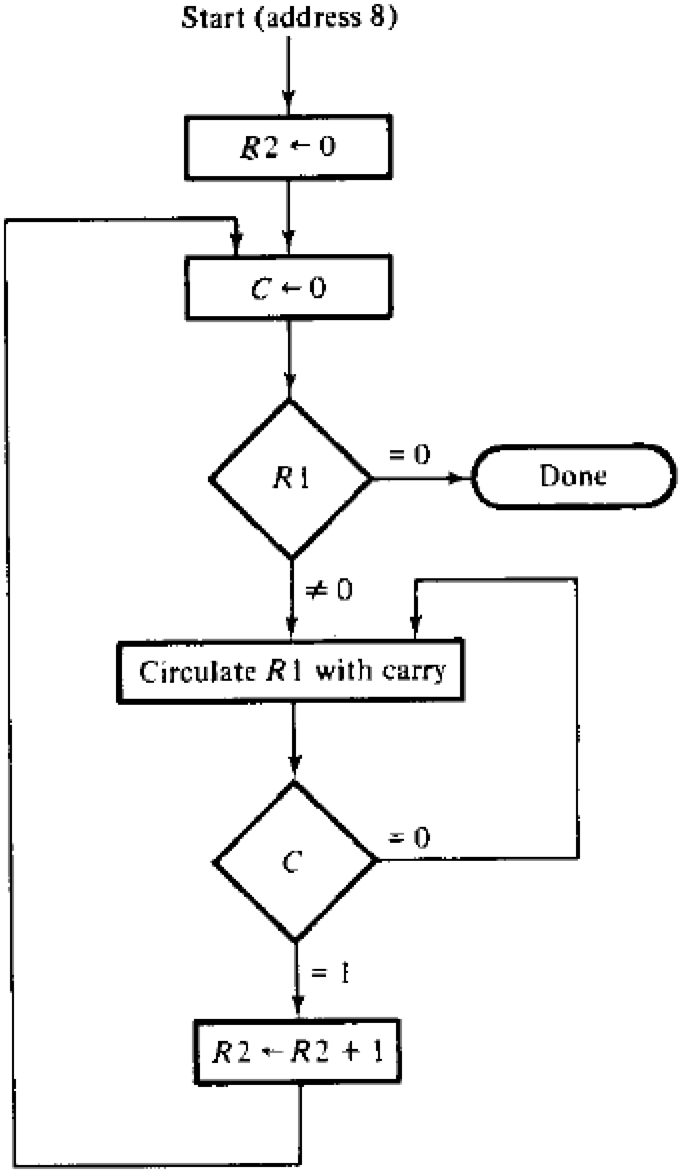
| ROM address | ROM outputs | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $x$ | $s_2$ | $s_1$ | $s_0$ | $C_{in}$ | $L$ | $y$ | $z$ | $w$ | *Address* | | | *Select* | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 0 0 0 | 1 | 0 | 0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

# Control of processor unit



# Example2

- Problem: counting no of 1’s in register R1
- Algorithm:



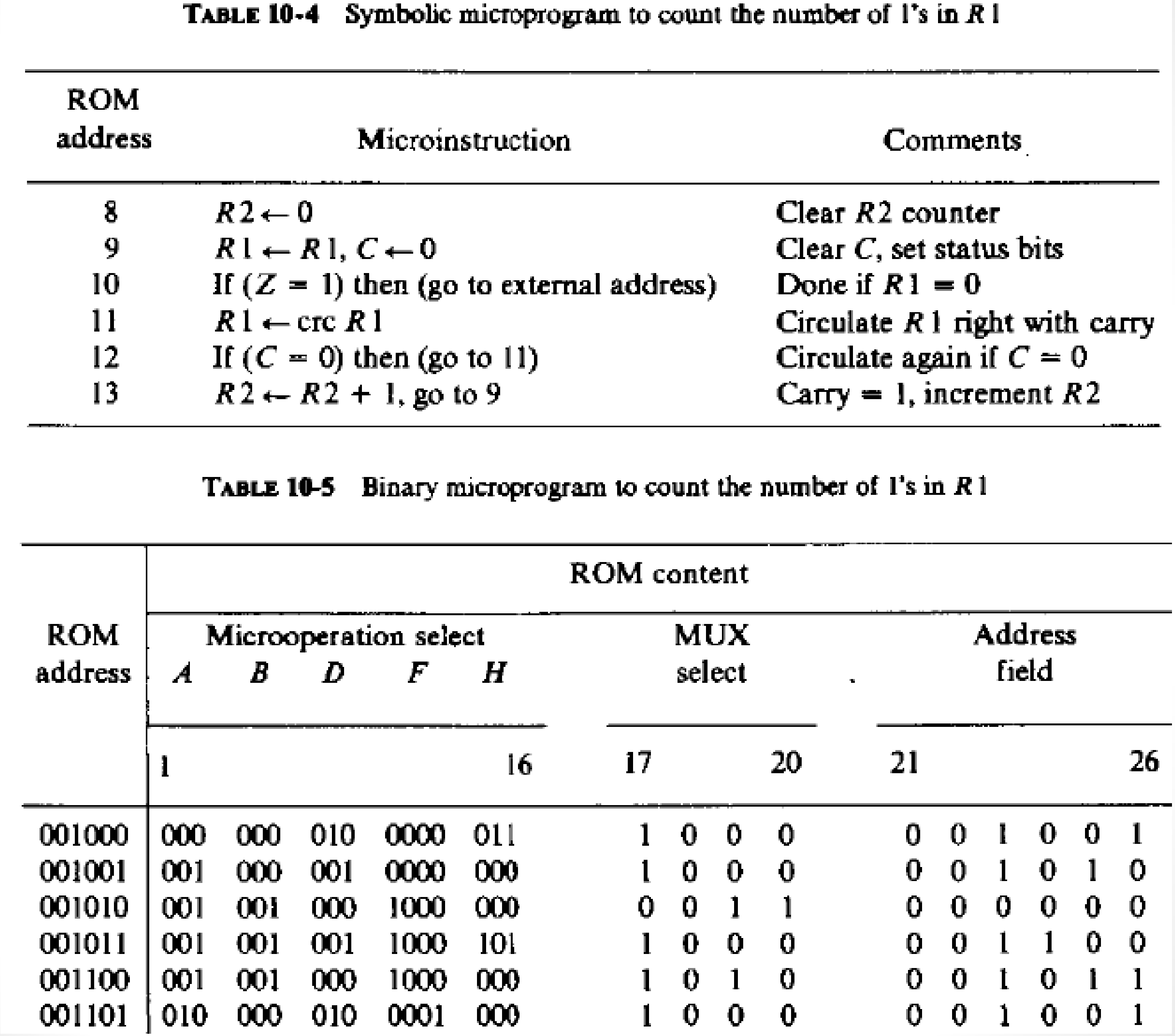


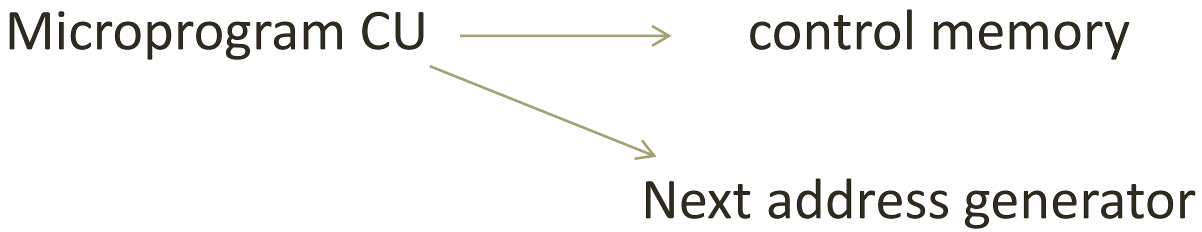
**Table 10-4** Symbolic microprogram to count the number of 1's in *R* 1

| ROM address | Microinstruction | Comments |
|---|---|---|
| 8 | $R2 \leftarrow 0$ | Clear *R*2 counter |
| 9 | $R1 \leftarrow R1, C \leftarrow 0$ | Clear *C*, set status bits |
| 10 | If ($Z = 1$) then (go to external address) | Done if $R1 = 0$ |
| 11 | $R1 \leftarrow$ crc $R1$ | Circulate *R* 1 right with carry |
| 12 | If ($C = 0$) then (go to 11) | Circulate again if $C = 0$ |
| 13 | $R2 \leftarrow R2 + 1$, go to 9 | Carry = 1, increment *R*2 |

**Table 10-5** Binary microprogram to count the number of 1's in *R* 1

| ROM address | ROM content | | | | | | |
|---|---|---|---|---|---|---|---|
| | Microoperation select | | | | | MUX select | Address field |
| | *A* | *B* | *D* | *F* | *H* | | |
| | 1 | | | | 16 | 17 20 | 21 26 |
| 001000 | 000 | 000 | 010 | 0000 | 011 | 1 0 0 0 | 0 0 1 0 0 1 |
| 001001 | 001 | 000 | 001 | 0000 | 000 | 1 0 0 0 | 0 0 1 0 1 0 |
| 001010 | 001 | 001 | 000 | 1000 | 000 | 0 0 1 1 | 0 0 0 0 0 0 |
| 001011 | 001 | 001 | 001 | 1000 | 101 | 1 0 0 0 | 0 0 1 1 0 0 |
| 001100 | 001 | 001 | 000 | 1000 | 000 | 1 0 1 0 | 0 0 1 0 1 1 |
| 001101 | 010 | 000 | 010 | 0001 | 000 | 1 0 0 0 | 0 0 1 0 0 1 |

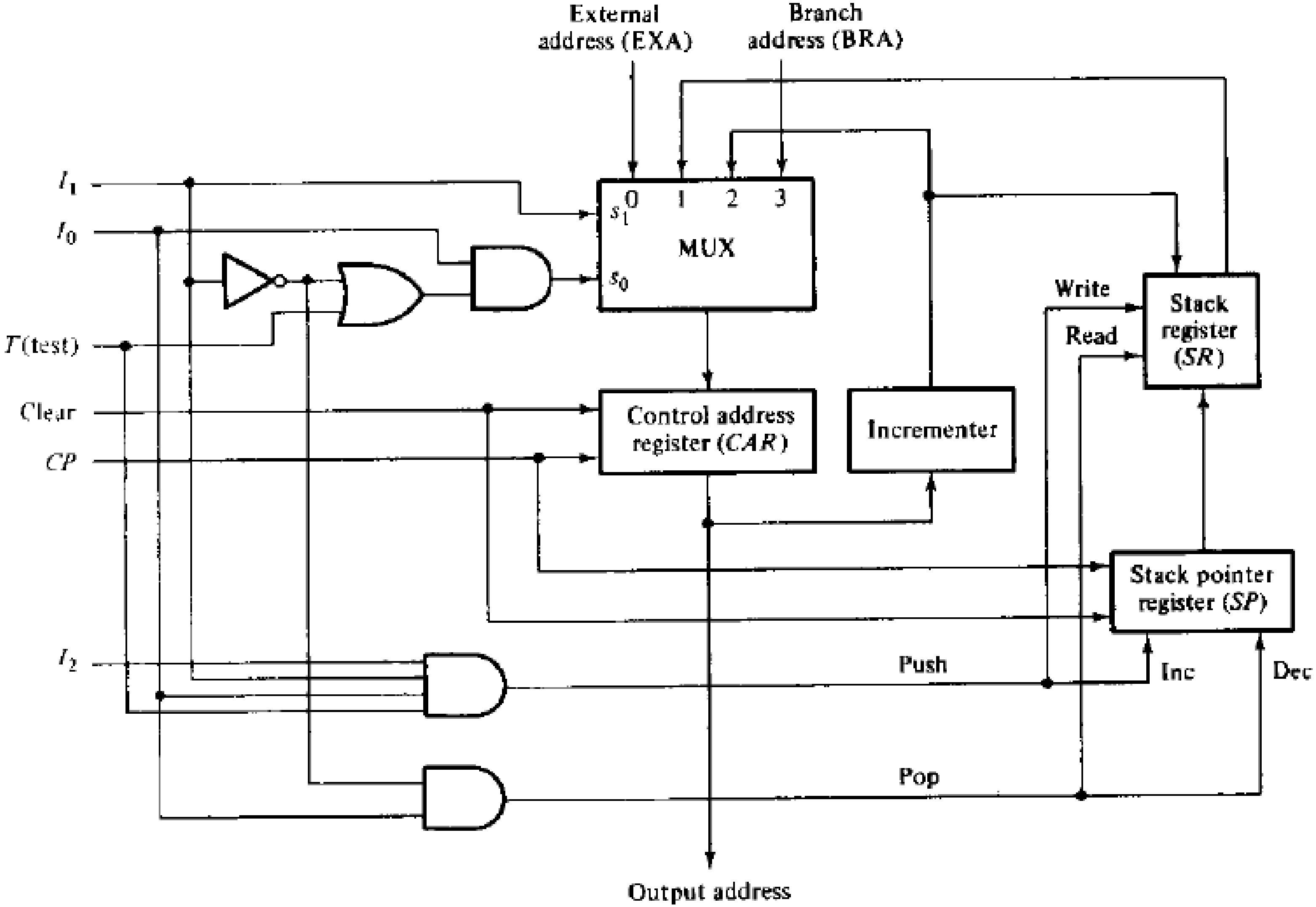
# Microprogram sequencer



- Microprogram CU → control memory
  → Next address generator

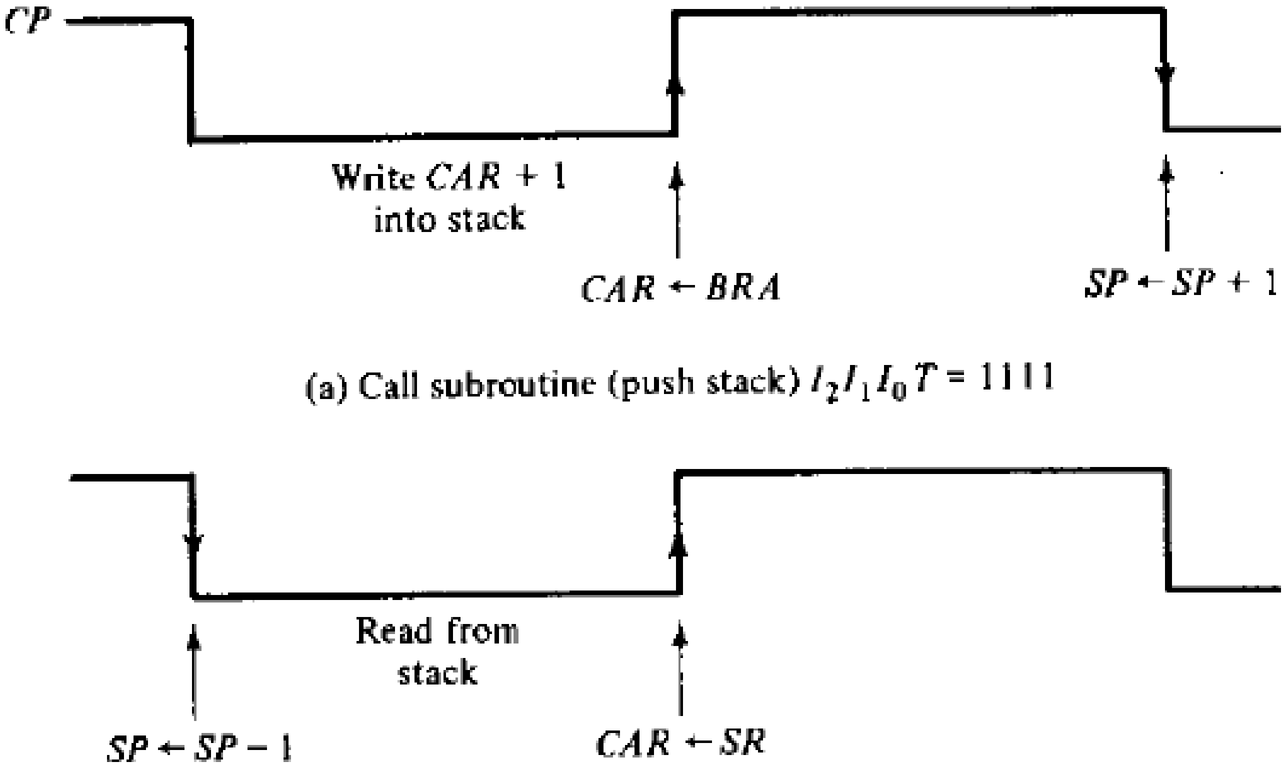
- Address generation part- sequencer

- Sequencer provide the following capabilities
  1. Increments the present address
  2. Branches to address specified in address field of microinstruction
  3. Branches to address if specified status bit is 1
  4. Transfers to new address specified by external source
  5. Has facility for subroutine calls and return



| $I_2$ | $I_1$ | $I_0$ | $T$ | $s_1$ | $s_0$ | Operation | Comments |
|---|---|---|---|---|---|---|---|
| $X$ | 0 | 0 | $X$ | 0 | 0 | $CAR \leftarrow EXA$ | Transfer external address |
| $X$ | 0 | 1 | $X$ | 0 | 1 | $CAR \leftarrow SR$ | Transfer from register stack |
| $X$ | 1 | 0 | $X$ | 1 | 0 | $CAR \leftarrow CAR + 1$ | Increment address |
| 0 | 1 | 1 | 0 | 1 | 0 | $CAR \leftarrow CAR + 1$ | Increment address |
| 0 | 1 | 1 | 1 | 1 | 1 | $CAR \leftarrow BRA$ | Transfer branch address |
| 1 | 1 | 1 | 0 | 1 | 0 | $CAR \leftarrow CAR + 1$ | Increment address |
| 1 | 1 | 1 | 1 | 1 | 1 | $CAR \leftarrow BRA,\ SR \leftarrow CAR + 1$ | Branch to subroutine |

- Output from CAR- address of control memory
- $I_2$,T have no effect when $I_1$ $I_0$=00,01,10
- Conditional branch operation:
- When $I_1$ $I_0$=11, $I_2$=0
  - If T=1, sequencer branches to BRA(branch address from microinstruction)
  - If T=0, increments CAR
- Conditional subroutine call:
- When $I_2 I_1$ $I_0$=111
  - If T=1, CAR+1 pushed to stack,sequencer branches to BRA
  - If T=0, increments CAR
- Return from subroutine:
- When $I_1$ $I_0$=01
  - Pop stack – branch to address on top of stack



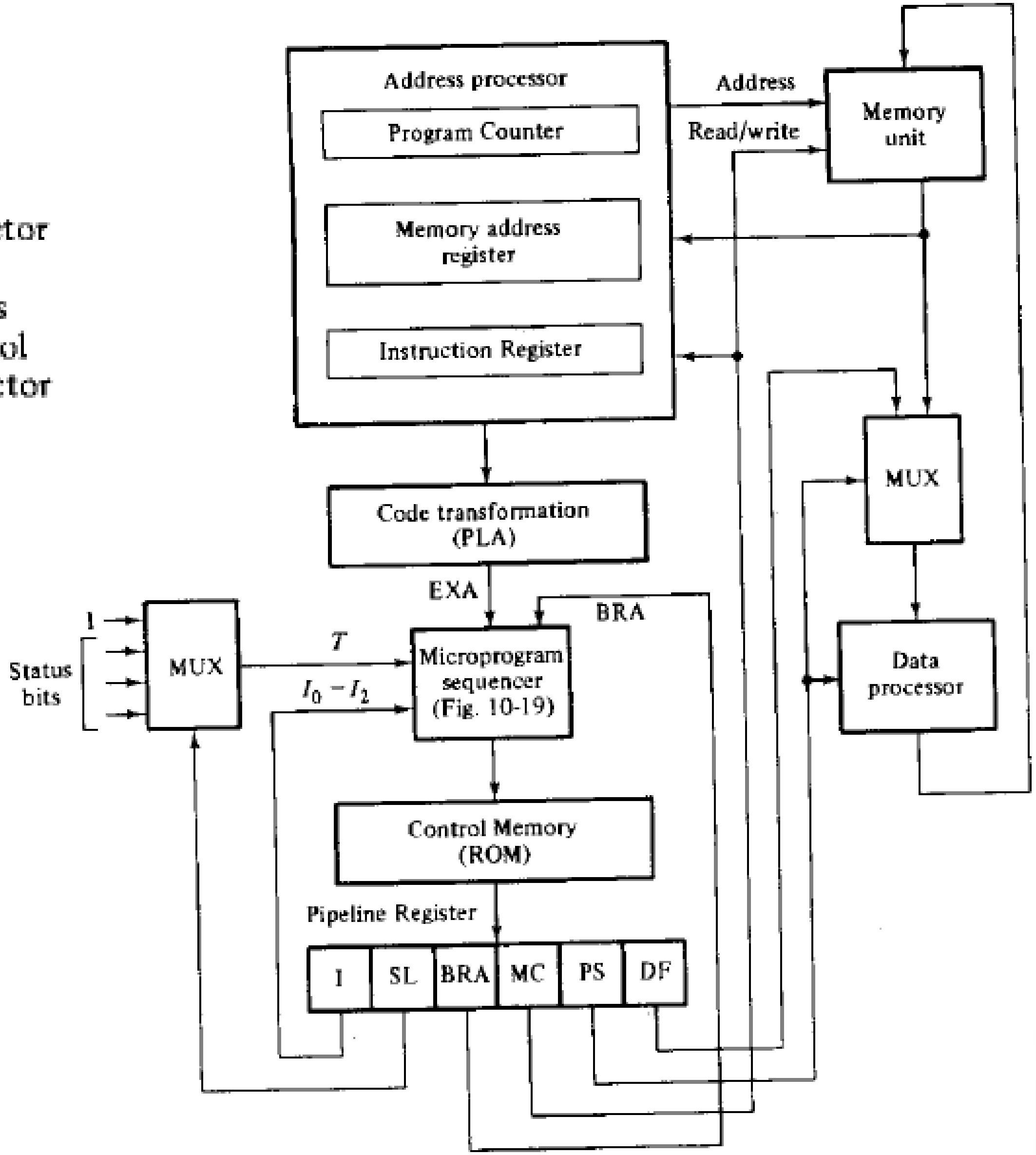
(a) Call subroutine (push stack) $I_2 I_1 I_0 T = 1111$

(b) Return from subroutine (pop stack) $I_1 I_0 = 01$

**Figure 10-20** Stack operations in microprogram sequencer

# Microprogrammed CPU organization

| | |
|---|---|
| I | - Sequence selector |
| SL | - MUX selector |
| BRA | - Branch address |
| MC | -- Memory control |
| PS | – Processor selector |
| DF | - Data field |


Address processor
Program Counter
Memory address register
Instruction Register
Address
Read/write
Memory unit
MUX
Data processor
Code transformation (PLA)
EXA
BRA
1
Status bits
MUX
$T$
$I_0 - I_2$
Microprogram sequencer (Fig. 10-19)
Control Memory (ROM)
Pipeline Register
I
SL
BRA
MC
PS
DF