AMERICAN INTERNAIONAL UNIVERSITY – BANGLADESH

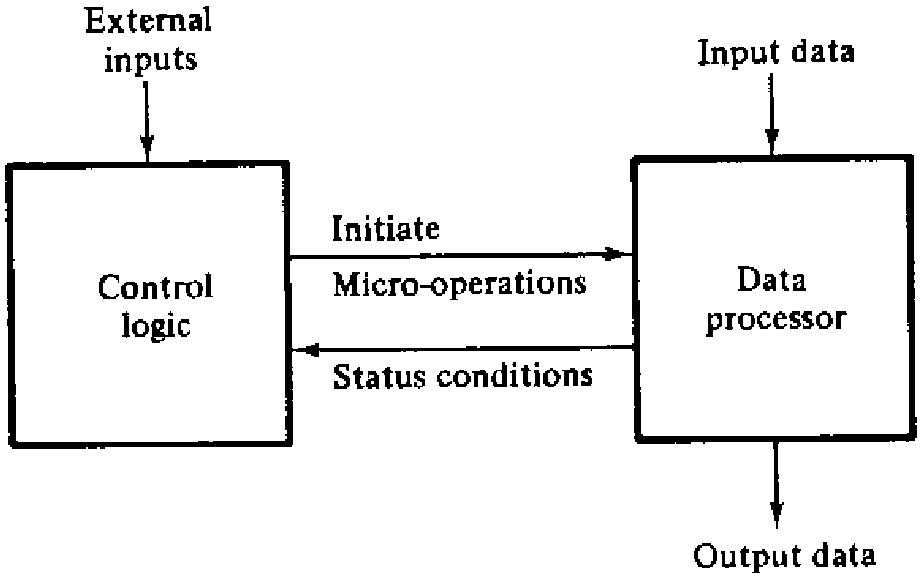
Where leaders are created

Lecture-3 (Final)
Control Logic Design

Department of EEE

Introduction

- The process of logic design is a complex undertaking.
- The data processor part may be general purpose processor unit.

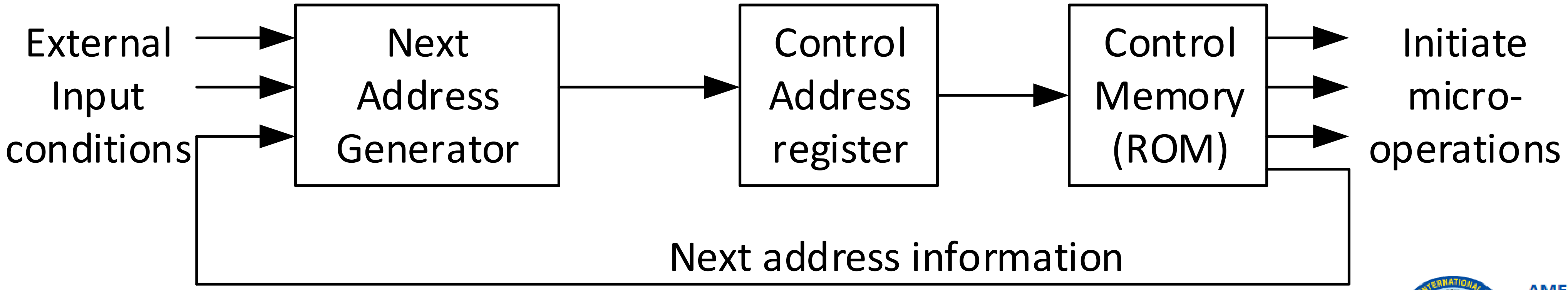


Control Organization

- Methods of Control Organization
  - One flip-flop per state method
  - Sequence register and decoder method
  - PLA control
  - **Microprogram control**
- The first two method must use SSI and MSI circuits for the implementations.
- PLA or microprogram which uses an LSI device.

Microprogram Control

- The Purpose of the control unit is to initiate a series of sequential steps of microoperations.
- 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.**

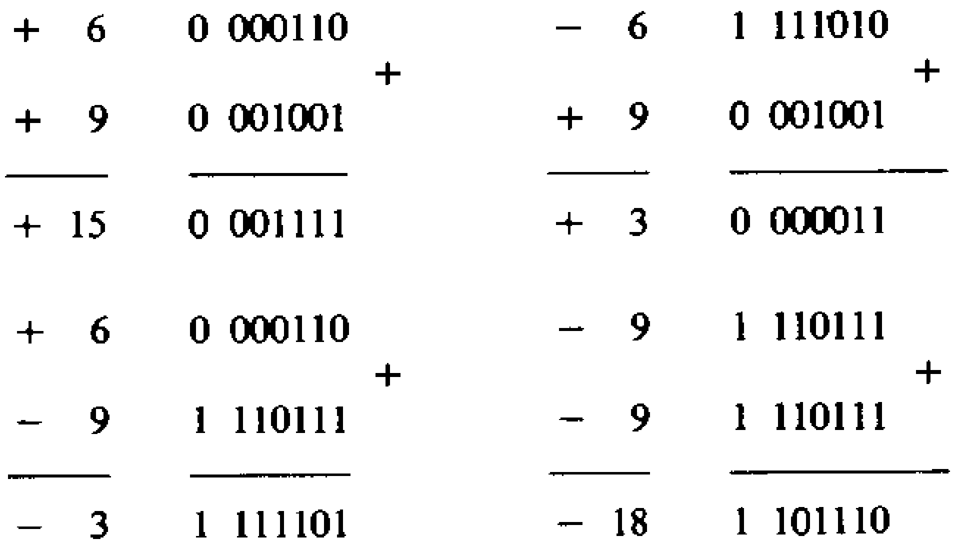


Hard-wired Control -Example

- The design is carried out in five consecutive steps
  1. The problem is stated
  2. An initial equipment configuration is assumed
  3. An algorithm is formulated
  4. The data-processor part is specified
  5. The control logic is designed

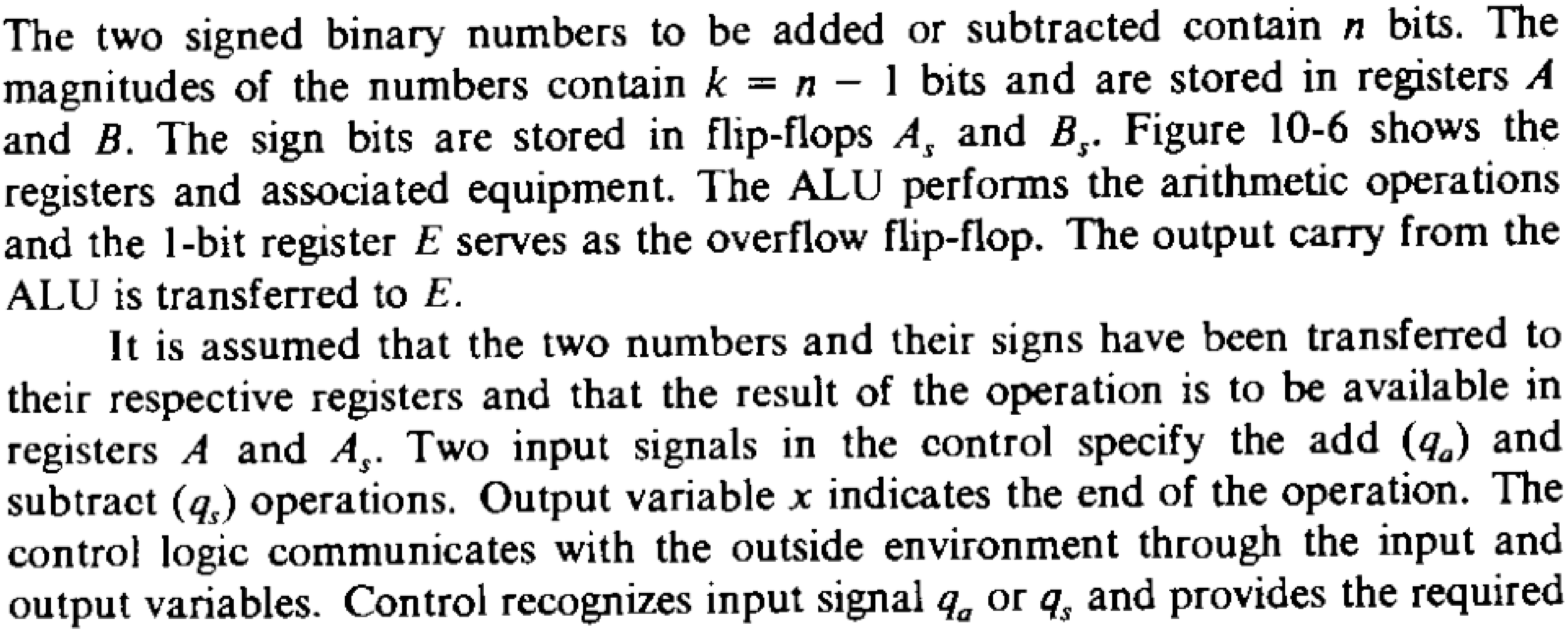
1. Statement of the problem

- Addition and subtraction of binary fixed point numbers when negative numbers are in sign 2’s complement 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.



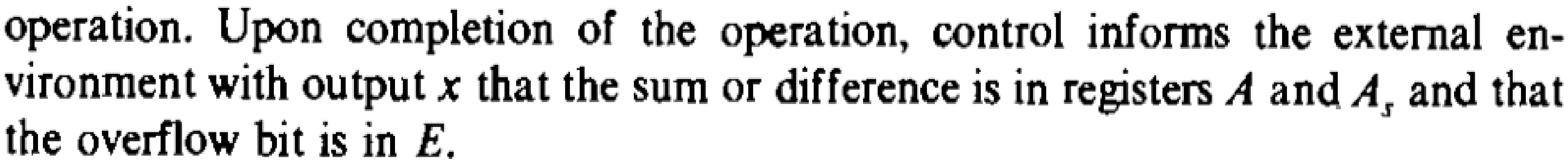
```
+  6     0 000110          -  6     1 111010
                    +                          +
+  9     0 001001          +  9     0 001001
-----    --------          -----    --------
+ 15     0 001111          +  3     0 000011

+  6     0 000110          -  9     1 110111
                    +                          +
-  9     1 110111          -  9     1 110111
-----    --------          -----    --------
-  3     1 111101          - 18     1 101110
```

Equipment Configuration



The two signed binary numbers to be added or subtracted contain n bits. The magnitudes of the numbers contain $k = n - 1$ bits and are stored in registers A and B. The sign bits are stored in flip-flops A_s and B_s. Figure 10-6 shows the registers and associated equipment. The ALU performs the arithmetic operations and the 1-bit register E serves as the overflow flip-flop. The output carry from the ALU is transferred to E.

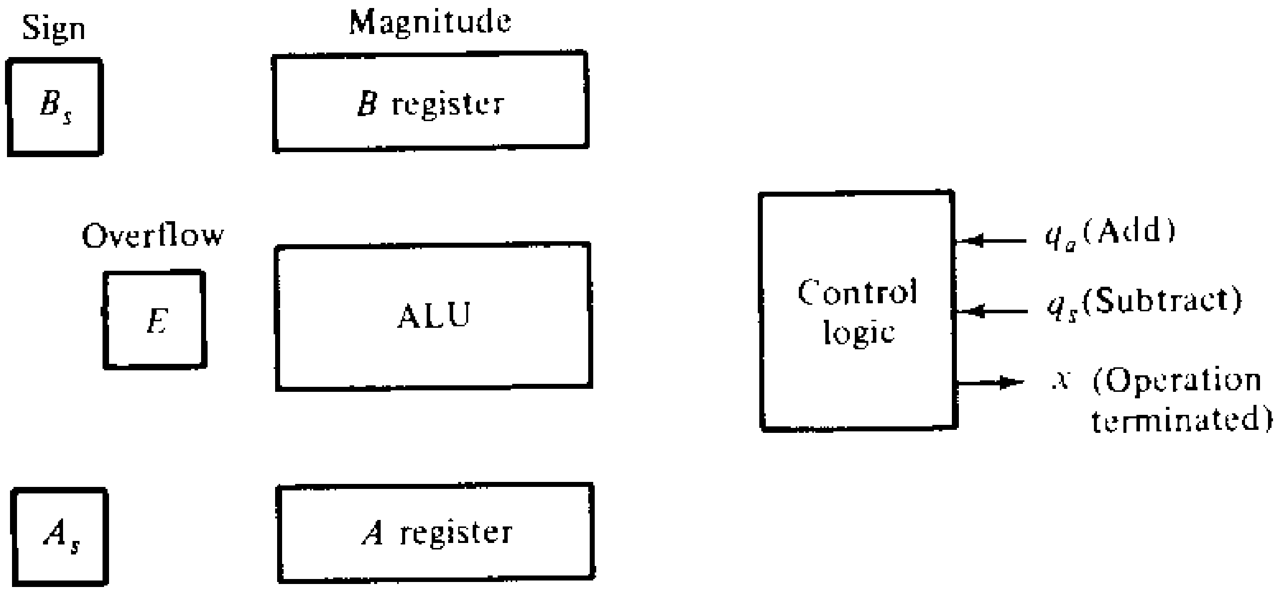
It is assumed that the two numbers and their signs have been transferred to their respective registers and that the result of the operation is to be available in registers A and A_s. Two input signals in the control specify the add (q_a) and subtract (q_s) operations. Output variable x indicates the end of the operation. The control logic communicates with the outside environment through the input and output variables. Control recognizes input signal q_a or q_s and provides the required



operation. Upon completion of the operation, control informs the external environment with output x that the sum or difference is in registers A and A_s and that the overflow bit is in E.

2.Equipment Configuration

- The two signed binary numbers to be added or subtracted contains n bits.
- The magnitudes of numbers contain k=n-1 bits are stored in registers A and B.
- The sign bits are stored in FFs As and Bs.



Register configuration for the Adder-subtractor

3.Derivation of the Algorithm

- When the numbers are added or subtracted algebraically, there are eight different conditions to be consider and may be expressed as compact form as follows:

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

- In arithmetic operation specified in subtraction, we change the sign of B and add. So the relationships :

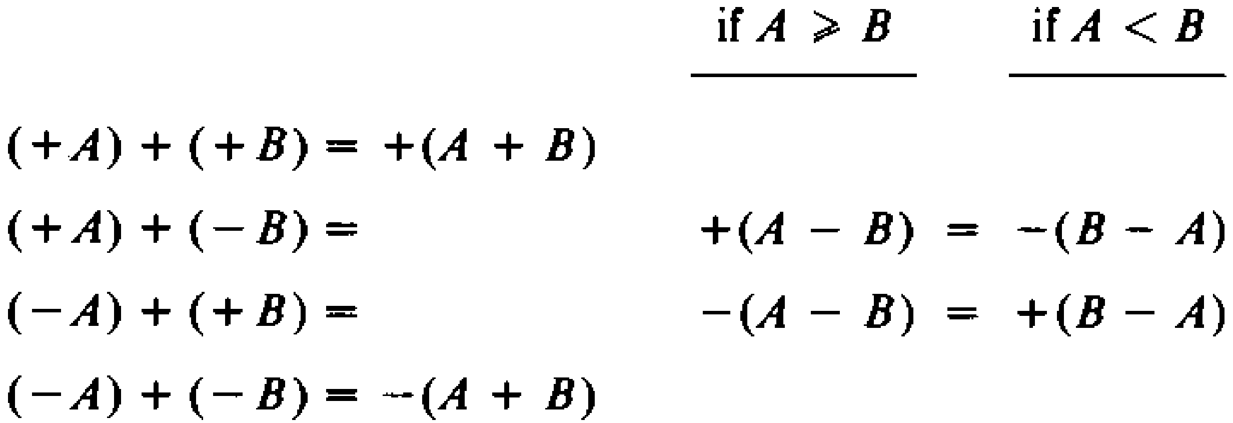
$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

- This reduce the number of possible conditions to four, namely:

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

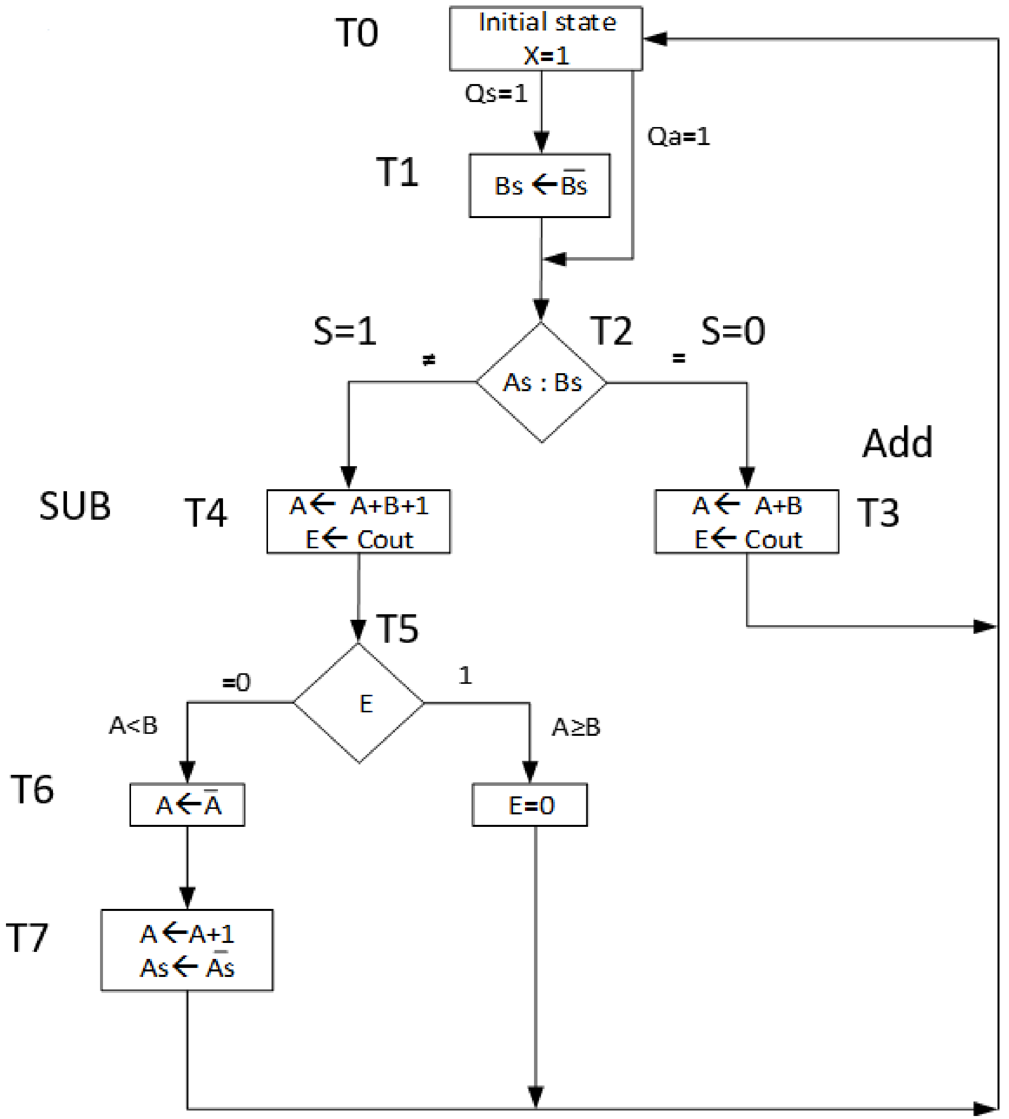
3. Derivation of the Algorithm cont….

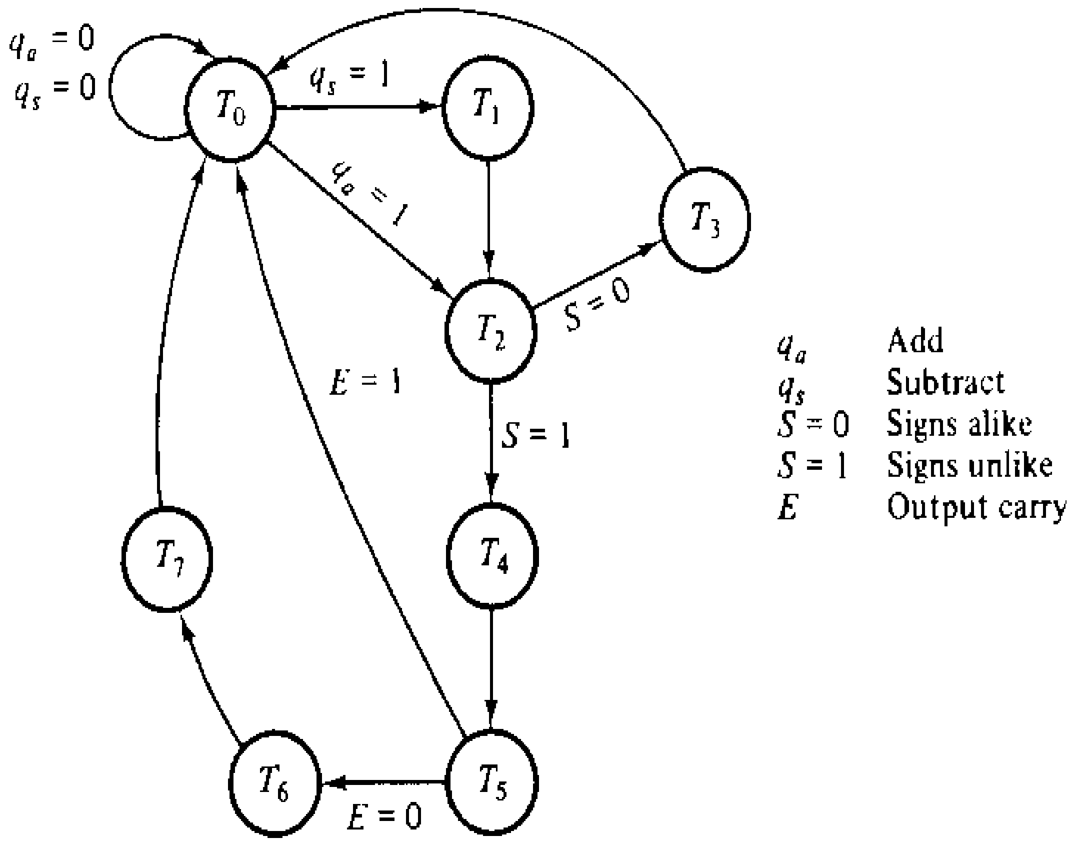
- When the signs of A and B are the same, we add the two magnitudes and the sign of the result in the same as the common sign.
- When the sign of A and B are not same, we subtract the smaller number from the larger and the sign of the result is the sign of the larger number.

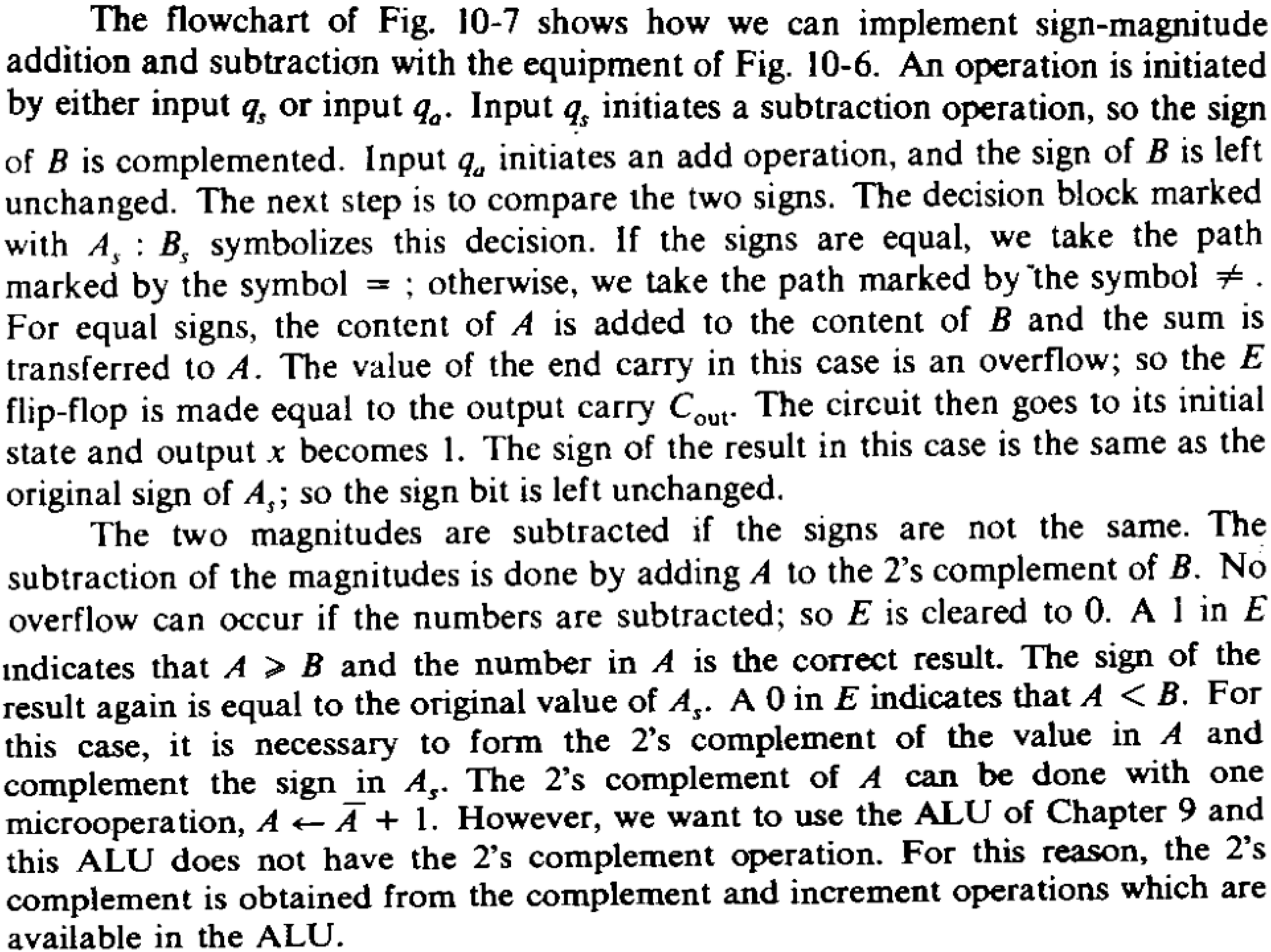


	if $A \geqslant B$	if $A < B$
$(+A) + (+B) = +(A + B)$		
$(+A) + (-B) =$	$+(A - B)$	$= -(B - A)$
$(-A) + (+B) =$	$-(A - B)$	$= +(B - A)$
$(-A) + (-B) = -(A + B)$		

Flowchart for sign magnitude addition and subtraction





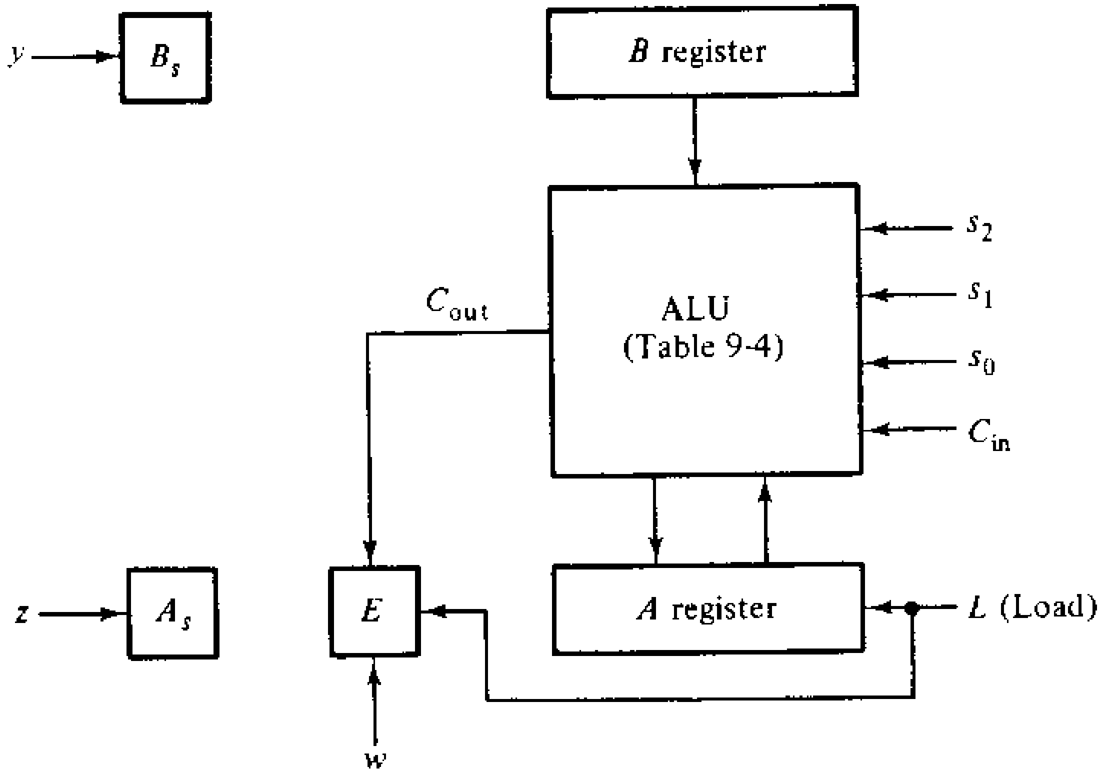


The flowchart of Fig. 10-7 shows how we can implement sign-magnitude addition and subtraction with the equipment of Fig. 10-6. An operation is initiated by either input q_s or input q_a. Input q_s initiates a subtraction operation, so the sign of B is complemented. Input q_a initiates an add operation, 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 $\neq$. 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, $A \leftarrow \overline{A} + 1$. 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.

4. Data Processor Register

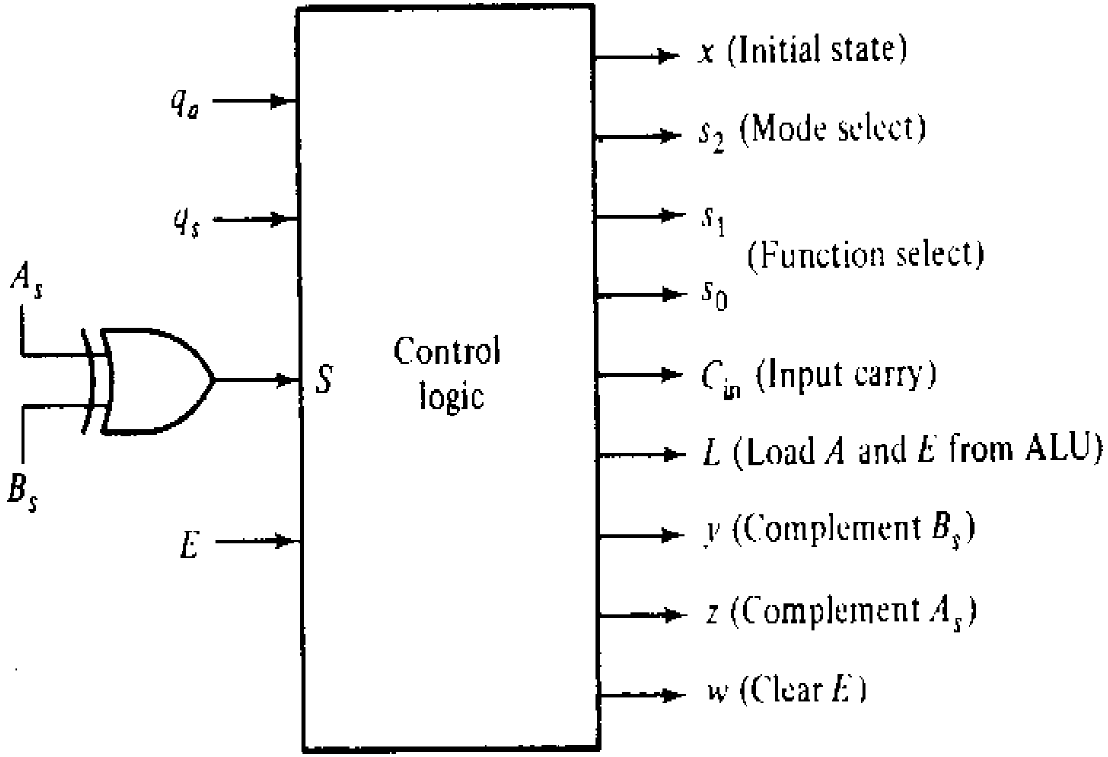
- The operations between A and B can be done with the ALU.
- The operations with As, Bs and E must be initiated with separate control variables.



5. Control Block Diagram

- The control receives five inputs: two form the external environment and three from the data-processor.
- To simplify the design we define new variable S:

S = As xor Bs



Control State Diagram

T_0 : Initial state $x = 1$

$T_1 : B_s \leftarrow \bar{B}_s$

T_2 : nothing

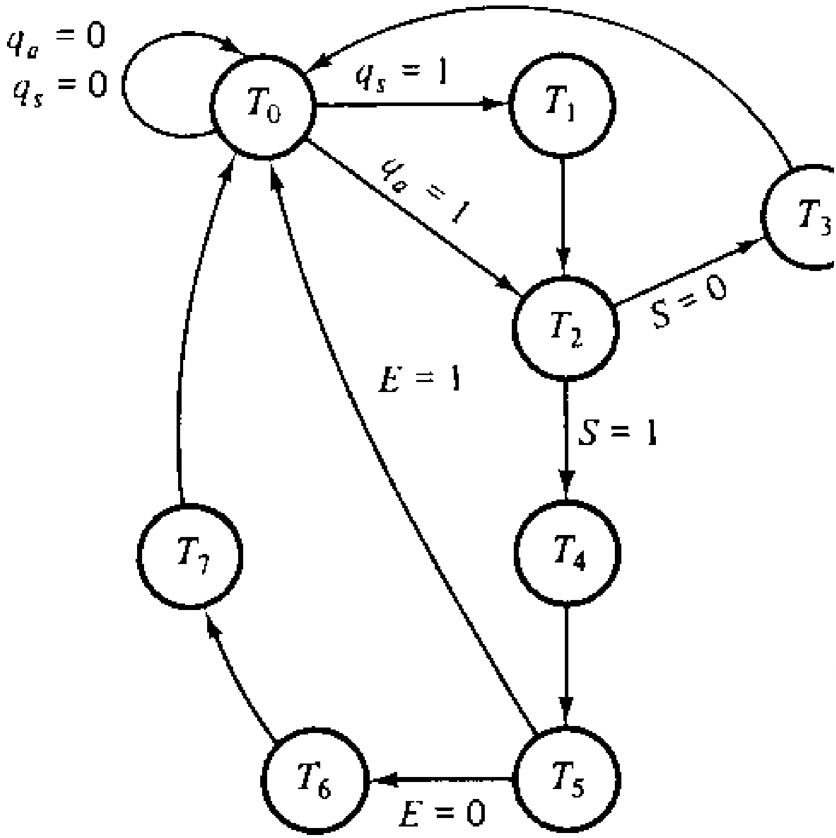
$T_3 : A \leftarrow A + B,\ E \leftarrow C_{out}$

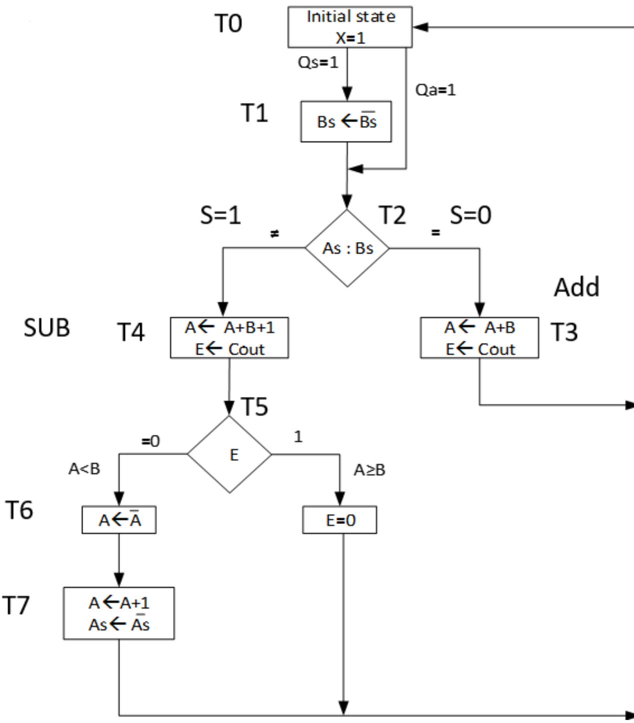
$T_4 : A \leftarrow A + \bar{B} + 1,\ E \leftarrow C_{out}$

$T_5 : E \leftarrow 0$

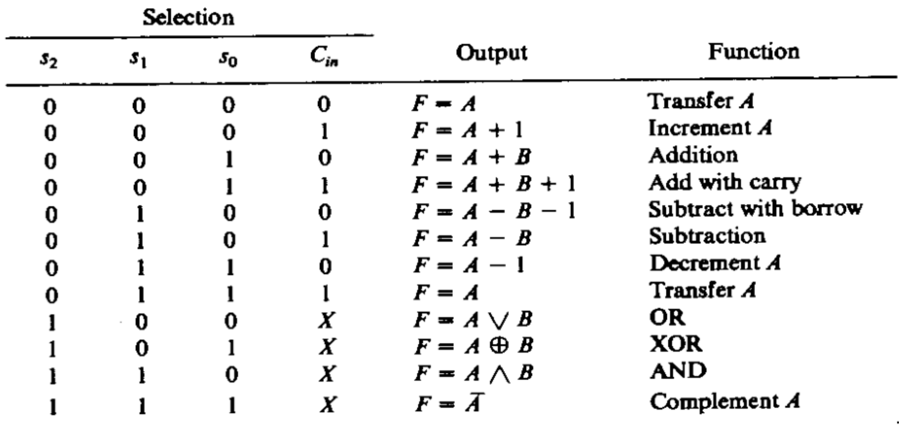
$T_6 : A \leftarrow \bar{A}$

$T_7 : A \leftarrow A + 1,\ A_s \leftarrow \bar{A}_s$



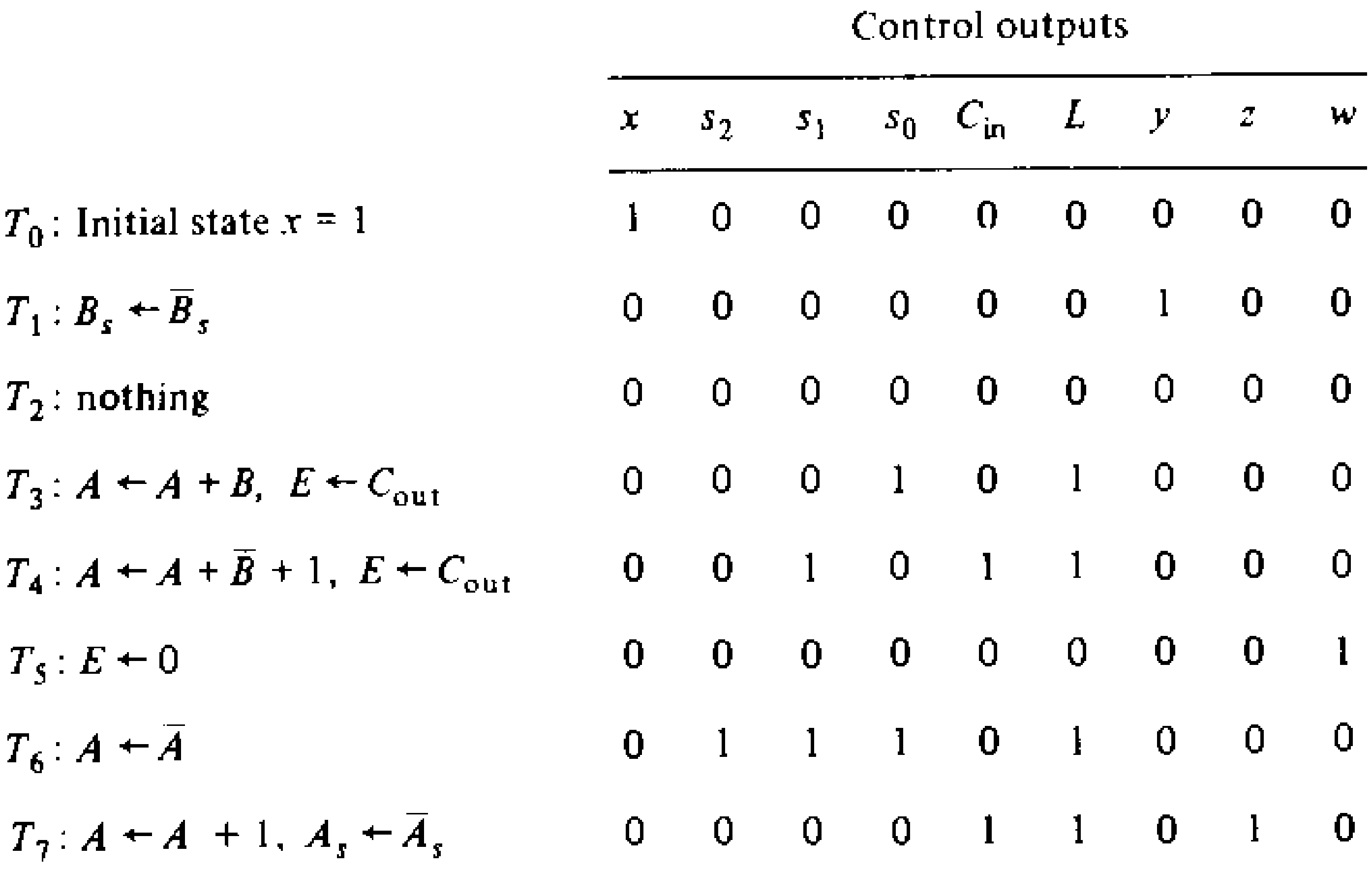


We start by assigning an initial state, T_0, to the sequential controller. We then determine the transition to other states T_1, T_2, T_3, and so on. For each state, we determine the microoperations that must be initiated by the control circuit. This procedure produces the state diagram for the controller,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

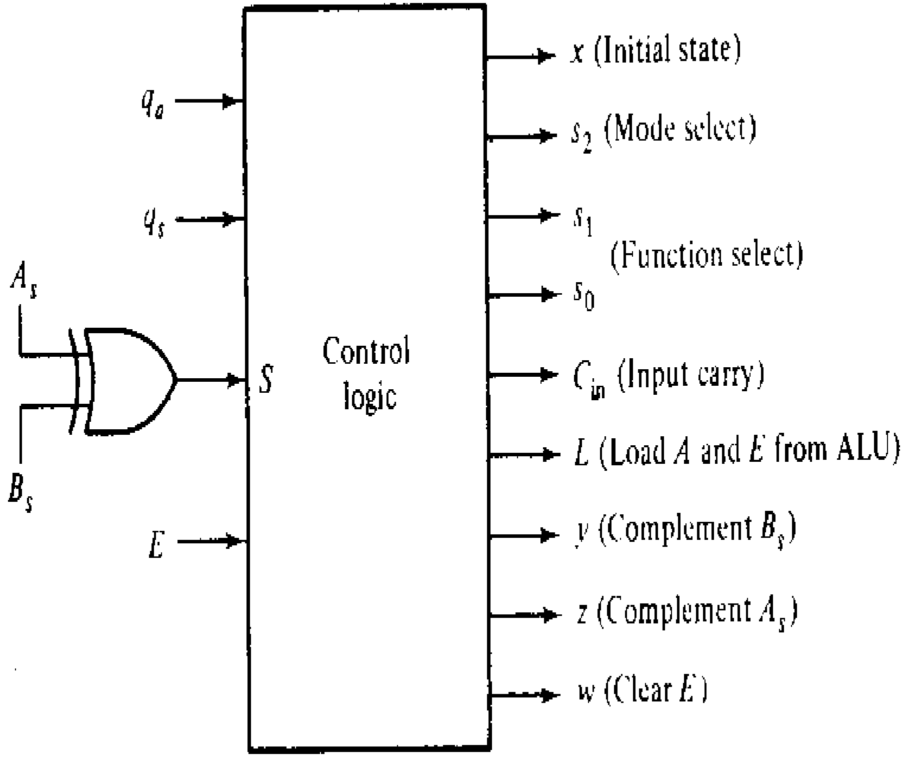


Selection				Output	Function
s_2	s_1	s_0	C_{in}		
0	0	0	0	$F = A$	Transfer A
0	0	0	1	$F = A + 1$	Increment A
0	0	1	0	$F = A + B$	Addition
0	0	1	1	$F = A + B + 1$	Add with carry
0	1	0	0	$F = A - B - 1$	Subtract with borrow
0	1	0	1	$F = A - B$	Subtraction
0	1	1	0	$F = A - 1$	Decrement A
0	1	1	1	$F = A$	Transfer A
1	0	0	X	$F = A \vee B$	OR
1	0	1	X	$F = A \oplus B$	XOR
1	1	0	X	$F = A \wedge B$	AND
1	1	1	X	$F = \bar{A}$	Complement A

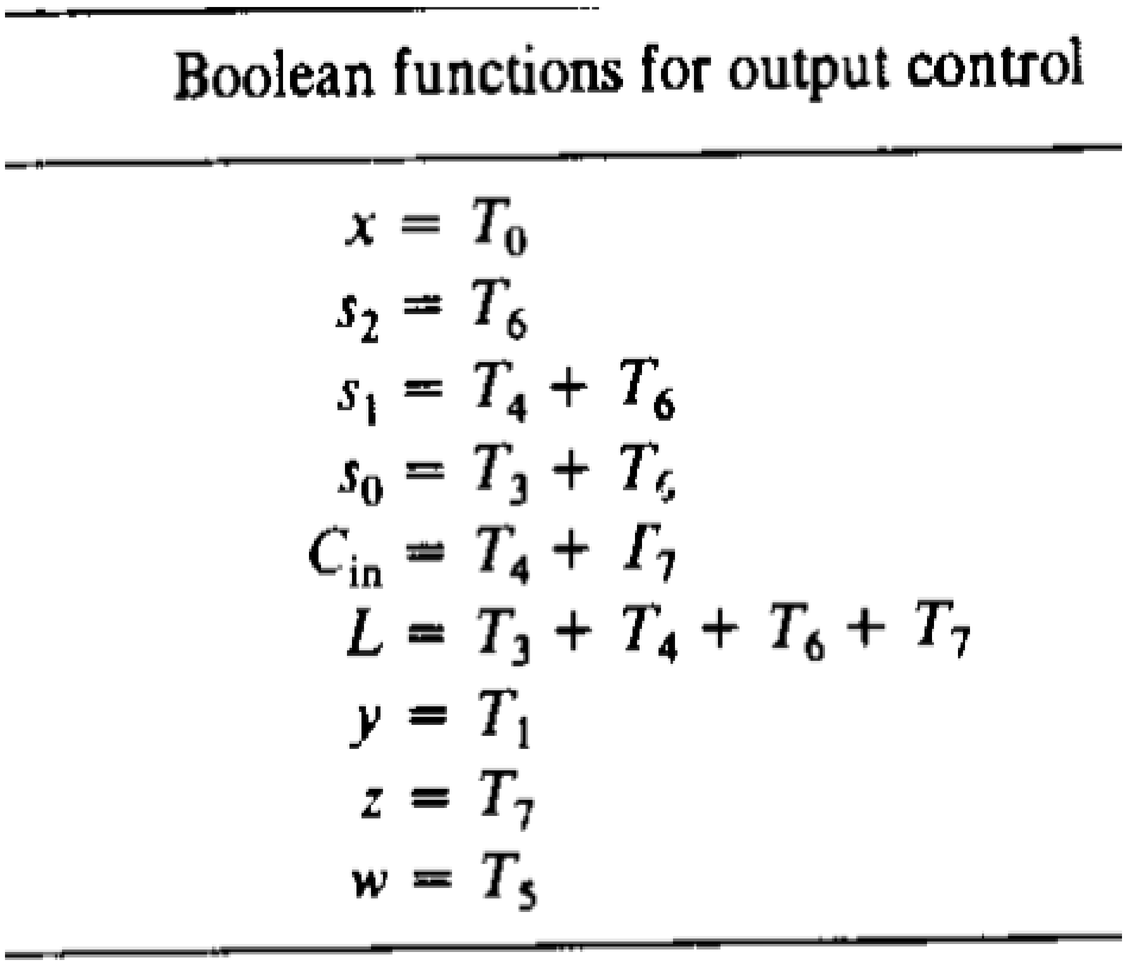
Sequence of register transfers



	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: $B_s \leftarrow \bar{B}_s$	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: $A \leftarrow A + \bar{B} + 1$, $E \leftarrow C_{out}$	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: $A \leftarrow \bar{A}$	0	1	1	1	0	1	0	0	0
T_7: $A \leftarrow A + 1$, $A_s \leftarrow \bar{A}_s$	0	0	0	0	1	1	0	1	0



Logic Equation for Hardwired Control Unit

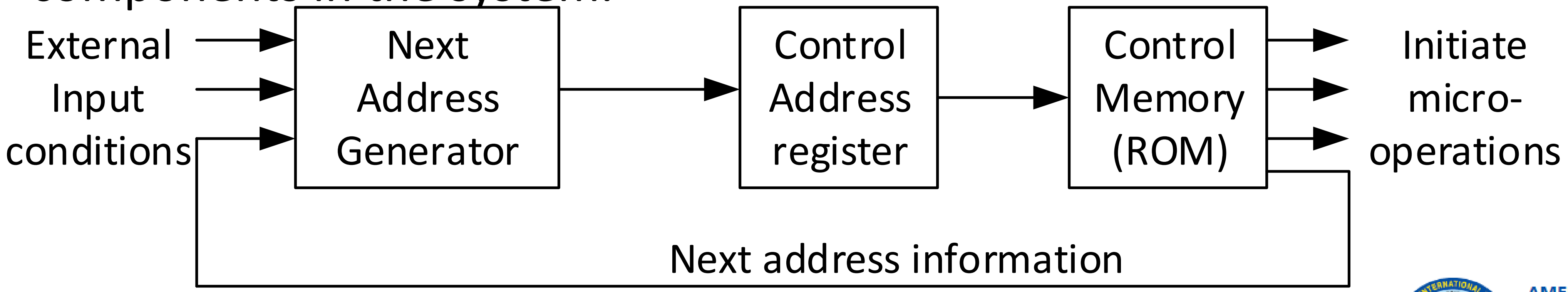


Boolean functions for output control

$$
\begin{aligned}
x &= T_0 \\
s_2 &= T_6 \\
s_1 &= T_4 + T_6 \\
s_0 &= T_3 + T_6 \\
C_{\text{in}} &= T_4 + T_7 \\
L &= T_3 + T_4 + T_6 + T_7 \\
y &= T_1 \\
z &= T_7 \\
w &= T_5
\end{aligned}
$$

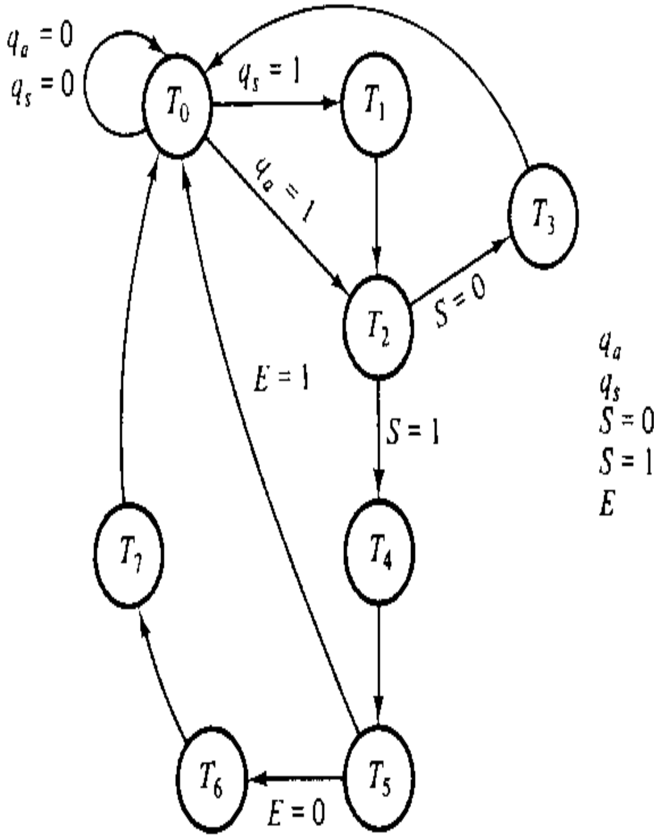
Microprogram Control

- In microprogram control, the control variables that initiate microoperations are stored in memory.
- The control memory is usually a ROM.
- The control variables stored in memory are read once at a time to initiate the sequence of microoperations for the system.
- The words stored in a control memory are microinstructions, and each microinstruction specifies one or more microoperations for the components in the system.



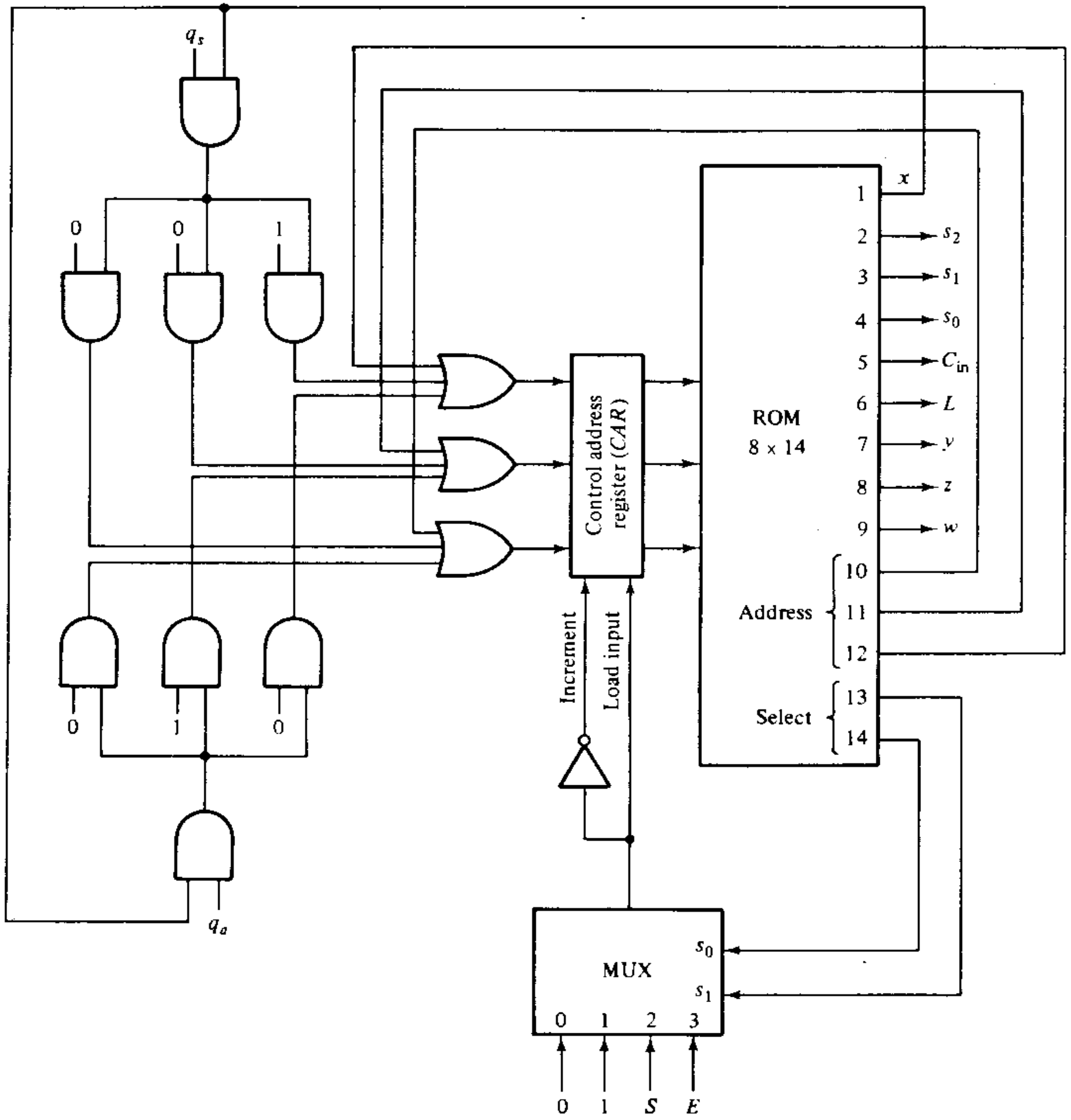
Microprogram Control Contd..

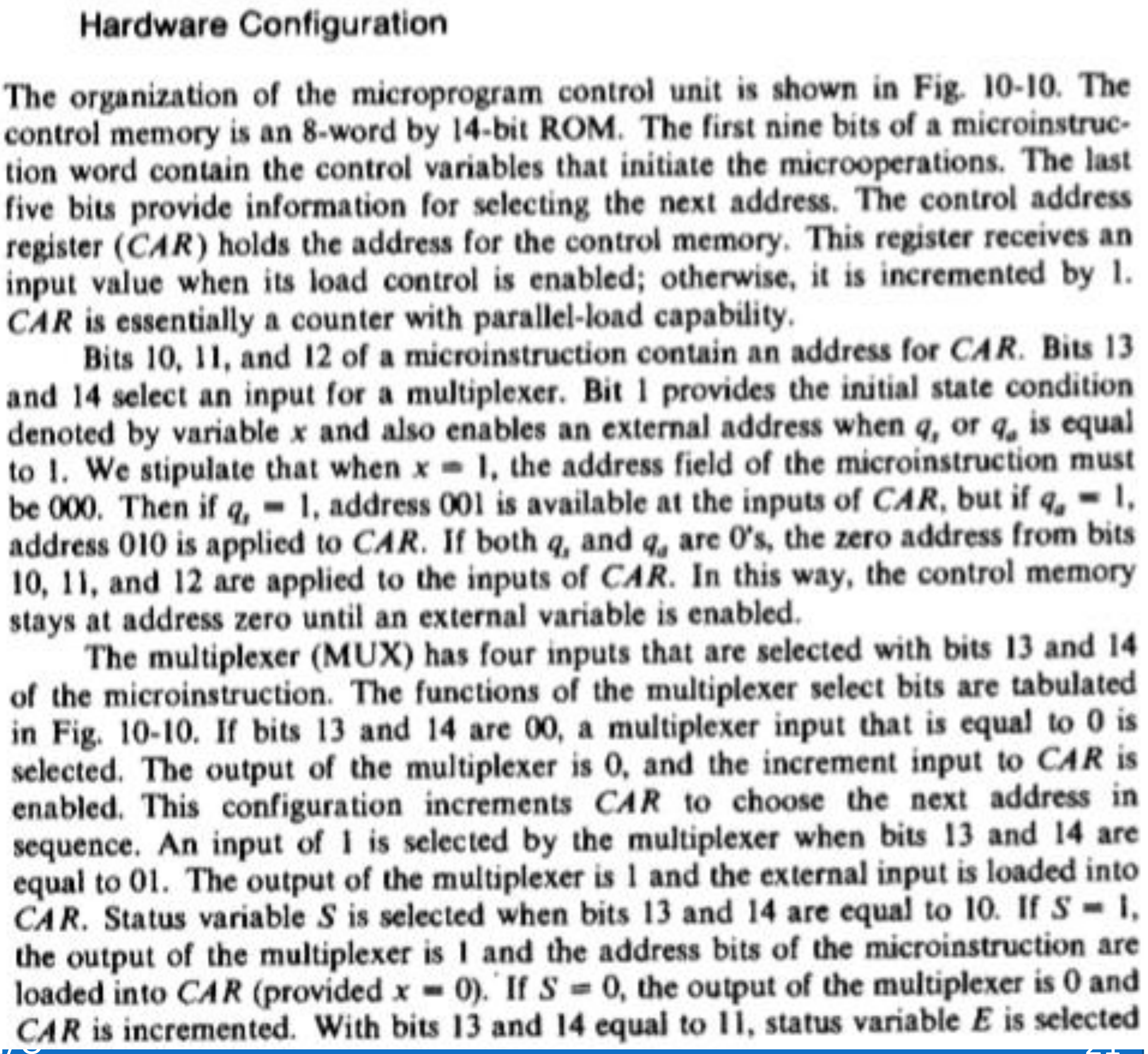
- Inspection of state diagram reveals that the address sequencing in the microprogram control must have the following capabilities.
    - Provision for loading an external address as a result of the concurrence of external signals q_a and q_s.
    - Provision for sequencing consecutive addresses.
    - Provision for choosing between two addresses as a function of present value of the status variables S and E.
- Each microinstruction must contain a number of bits to specify the way that the next address is to be selected.



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



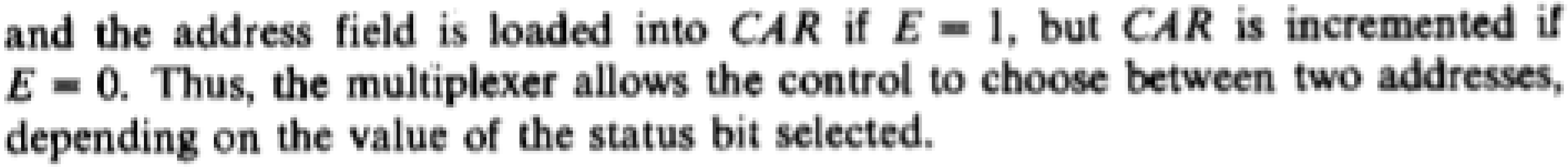


Hardware Configuration

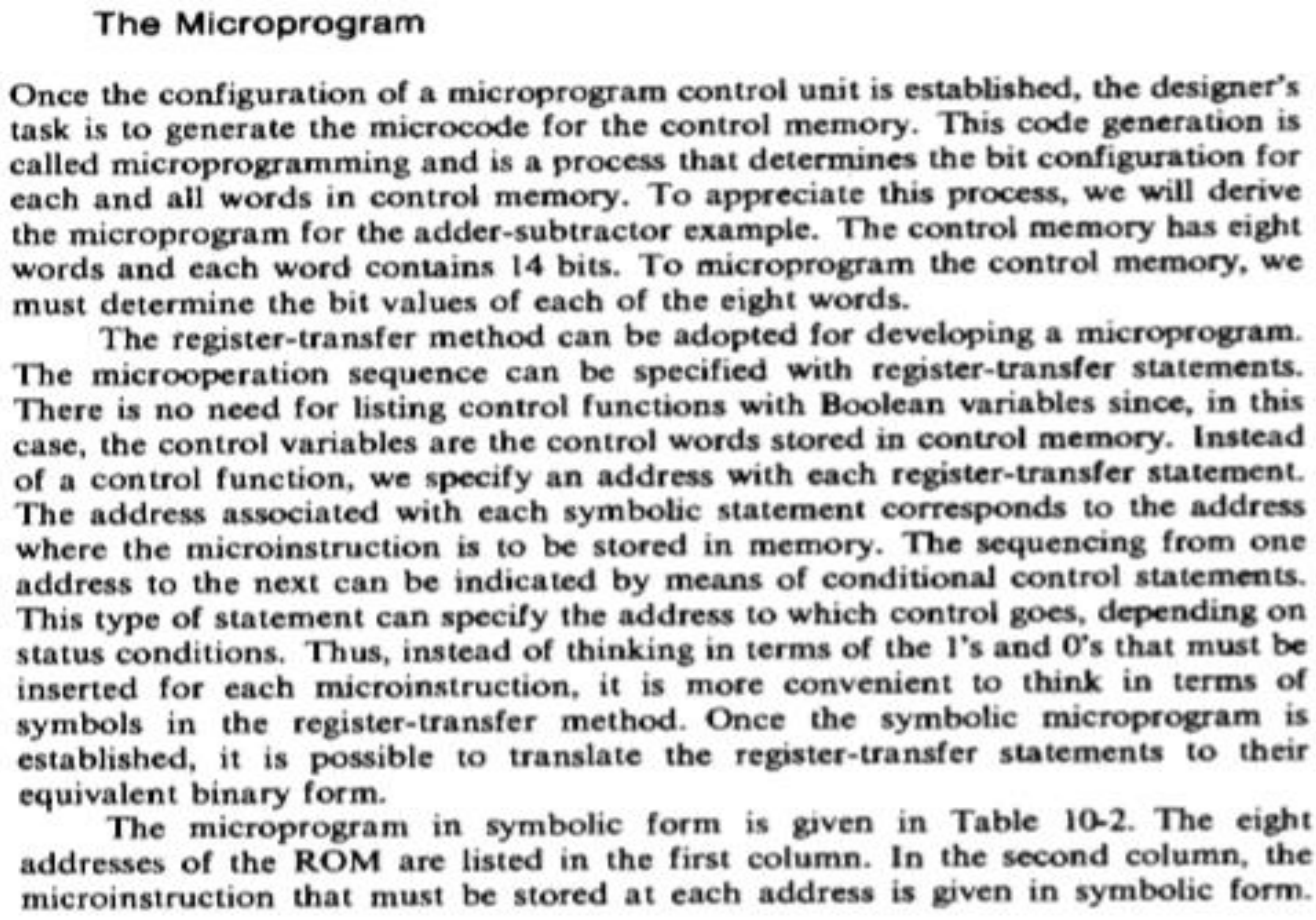
The organization of the microprogram control unit is shown in Fig. 10-10. The control memory is an 8-word by 14-bit ROM. The first nine bits of a microinstruction word contain the control variables that initiate the microoperations. The last five bits provide information for selecting the next address. The control address register (*CAR*) holds the address for the control memory. This register receives an input value when its load control is enabled; otherwise, it is incremented by 1. *CAR* is essentially a counter with parallel-load capability.

Bits 10, 11, and 12 of a microinstruction contain an address for *CAR*. Bits 13 and 14 select an input for a multiplexer. Bit 1 provides the initial state condition denoted by variable x and also enables an external address when q_s or q_a is equal to 1. We stipulate that when $x = 1$, the address field of the microinstruction must be 000. Then if $q_s = 1$, address 001 is available at the inputs of *CAR*, but if $q_a = 1$, address 010 is applied to *CAR*. If both q_s and q_a are 0's, the zero address from bits 10, 11, and 12 are applied to the inputs of *CAR*. In this way, the control memory stays at address zero until an external variable is enabled.

The multiplexer (MUX) has four inputs that are selected with bits 13 and 14 of the microinstruction. The functions of the multiplexer select bits are tabulated in Fig. 10-10. If bits 13 and 14 are 00, a multiplexer input that is equal to 0 is selected. The output of the multiplexer is 0, and the increment input to *CAR* is enabled. This configuration increments *CAR* to choose the next address in sequence. An input of 1 is selected by the multiplexer when bits 13 and 14 are equal to 01. The output of the multiplexer is 1 and the external input is loaded into *CAR*. Status variable S is selected when bits 13 and 14 are equal to 10. If $S = 1$, the output of the multiplexer is 1 and the address bits of the microinstruction are loaded into *CAR* (provided $x = 0$). If $S = 0$, the output of the multiplexer is 0 and *CAR* is incremented. With bits 13 and 14 equal to 11, status variable E is selected



and the address field is loaded into CAR if $E = 1$, but CAR is incremented if $E = 0$. Thus, the multiplexer allows the control to choose between two addresses, depending on the value of the status bit selected.

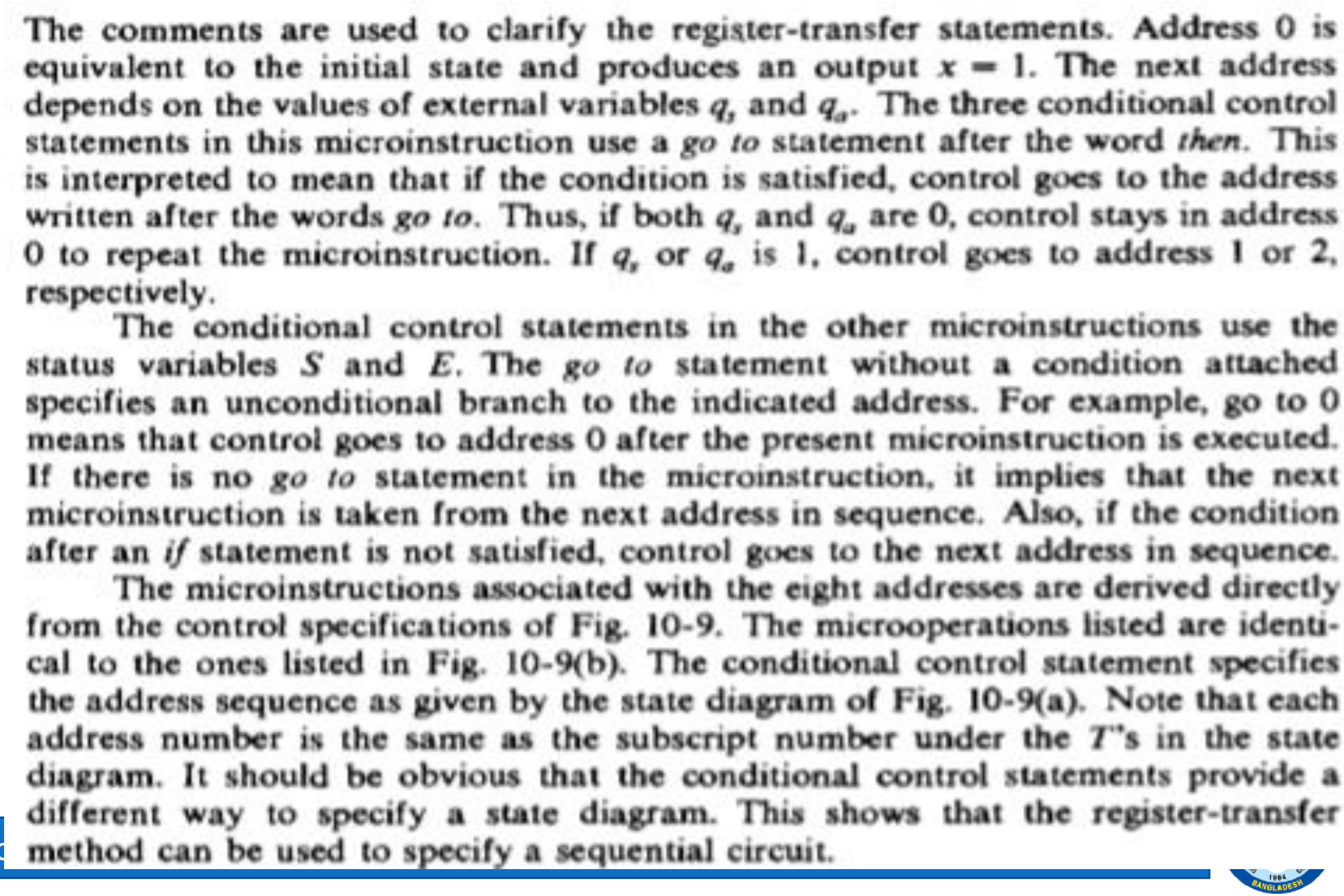


The Microprogram

Once the configuration of a microprogram control unit is established, the designer's task is to generate the microcode for the control memory. This code generation is called microprogramming and is a process that determines the bit configuration for each and all words in control memory. To appreciate this process, we will derive the microprogram for the adder-subtractor example. The control memory has eight words and each word contains 14 bits. To microprogram the control memory, we must determine the bit values of each of the eight words.

The register-transfer method can be adopted for developing a microprogram. The microoperation sequence can be specified with register-transfer statements. There is no need for listing control functions with Boolean variables since, in this case, the control variables are the control words stored in control memory. Instead of a control function, we specify an address with each register-transfer statement. The address associated with each symbolic statement corresponds to the address where the microinstruction is to be stored in memory. The sequencing from one address to the next can be indicated by means of conditional control statements. This type of statement can specify the address to which control goes, depending on status conditions. Thus, instead of thinking in terms of the 1's and 0's that must be inserted for each microinstruction, it is more convenient to think in terms of symbols in the register-transfer method. Once the symbolic microprogram is established, it is possible to translate the register-transfer statements to their equivalent binary form.

The microprogram in symbolic form is given in Table 10-2. The eight addresses of the ROM are listed in the first column. In the second column, the microinstruction that must be stored at each address is given in symbolic form.

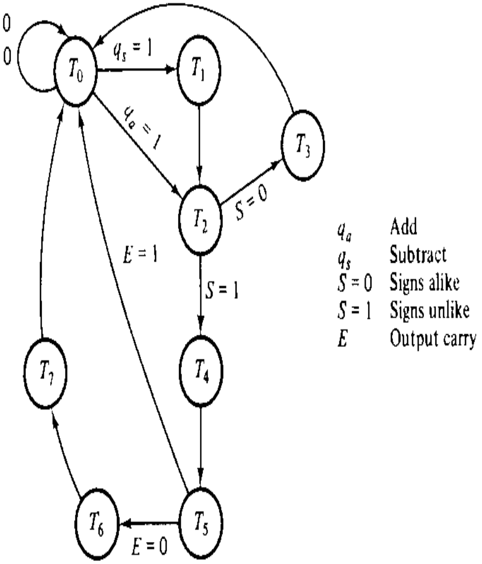


The comments are used to clarify the register-transfer statements. Address 0 is equivalent to the initial state and produces an output $x = 1$. The next address depends on the values of external variables q_s and q_a. The three conditional control statements in this microinstruction use a *go to* statement after the word *then*. This is interpreted to mean that if the condition is satisfied, control goes to the address written after the words *go to*. Thus, if both q_s and q_a are 0, control stays in address 0 to repeat the microinstruction. If q_s or q_a is 1, control goes to address 1 or 2, respectively.

The conditional control statements in the other microinstructions use the status variables S and E. The *go to* statement without a condition attached specifies an unconditional branch to the indicated address. For example, go to 0 means that control goes to address 0 after the present microinstruction is executed. If there is no *go to* statement in the microinstruction, it implies that the next microinstruction is taken from the next address in sequence. Also, if the condition after an *if* statement is not satisfied, control goes to the next address in sequence.

The microinstructions associated with the eight addresses are derived directly from the control specifications of Fig. 10-9. The microoperations listed are identical to the ones listed in Fig. 10-9(b). The conditional control statement specifies the address sequence as given by the state diagram of Fig. 10-9(a). Note that each address number is the same as the subscript number under the T's in the state diagram. It should be obvious that the conditional control statements provide a different way to specify a state diagram. This shows that the register-transfer method can be used to specify a sequential circuit.

The Microprogram for Control memory



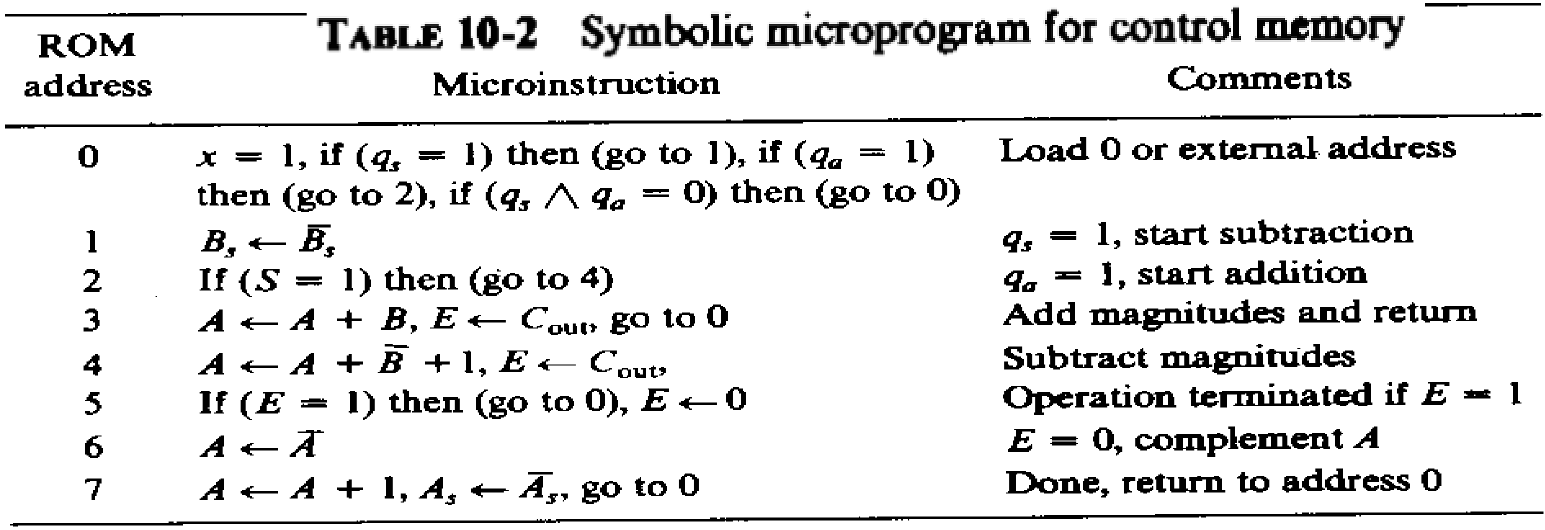
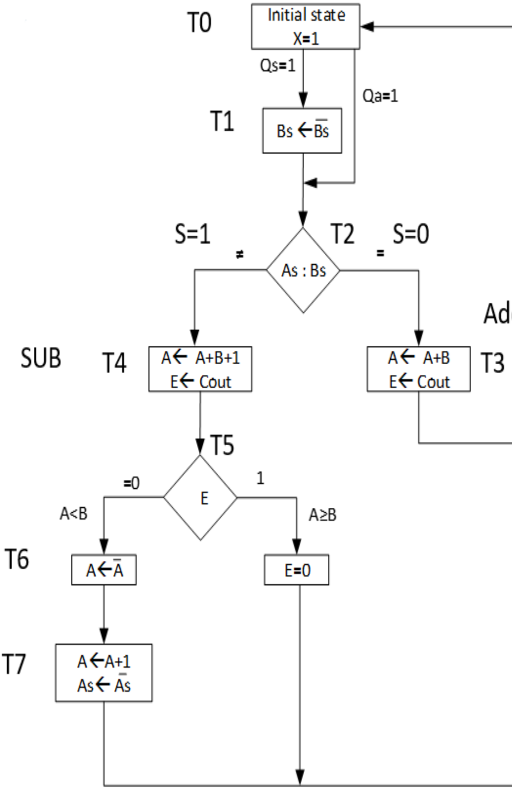
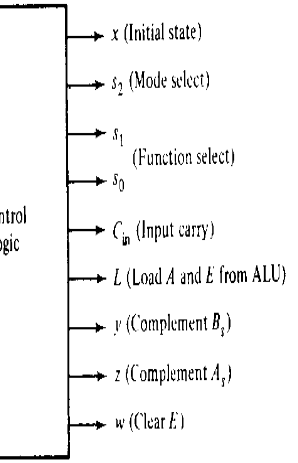
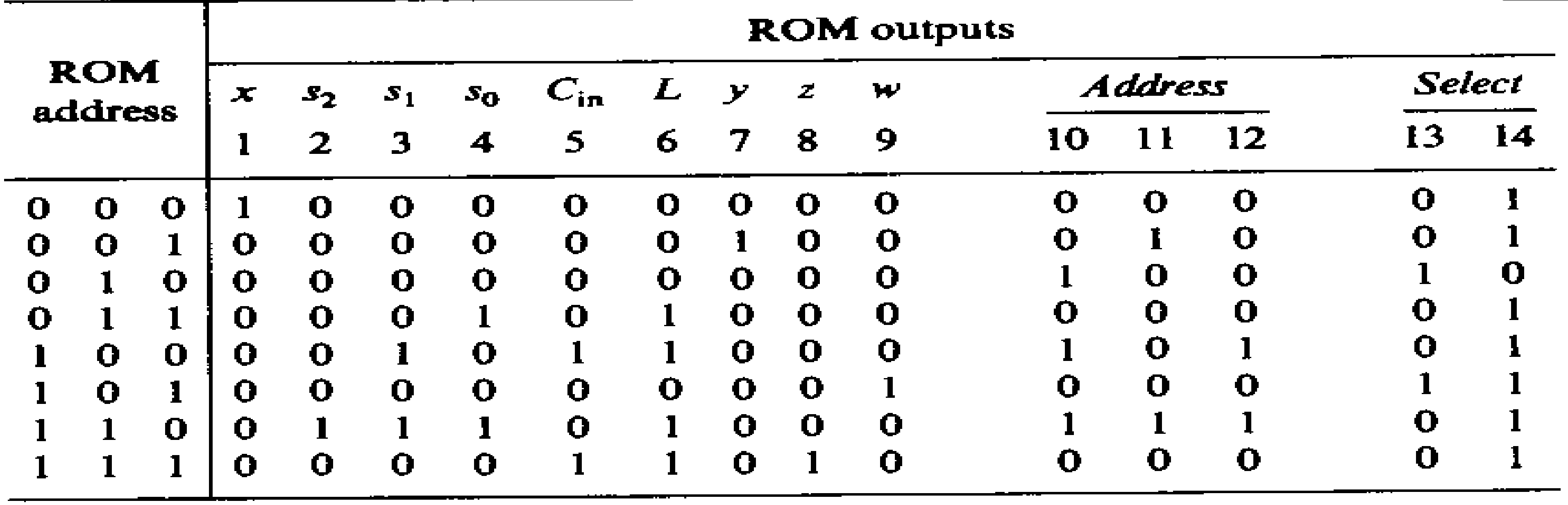


TABLE 10-2 Symbolic microprogram for control memory

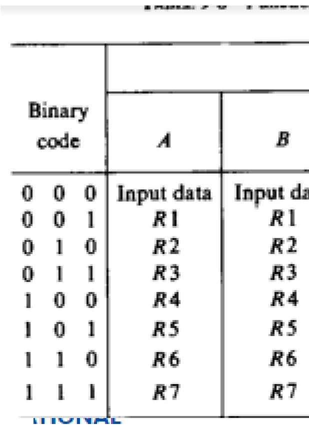
ROM address	Microinstruction	Comments
0	$x = 1$, if ($q_s = 1$) then (go to 1), if ($q_a = 1$) then (go to 2), if ($q_s \wedge q_a = 0$) then (go to 0)	Load 0 or external address
1	$B_s \leftarrow \overline{B}_s$	$q_s = 1$, start subtraction
2	If ($S = 1$) then (go to 4)	$q_a = 1$, start addition
3	$A \leftarrow A + B$, $E \leftarrow C_{out}$, 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







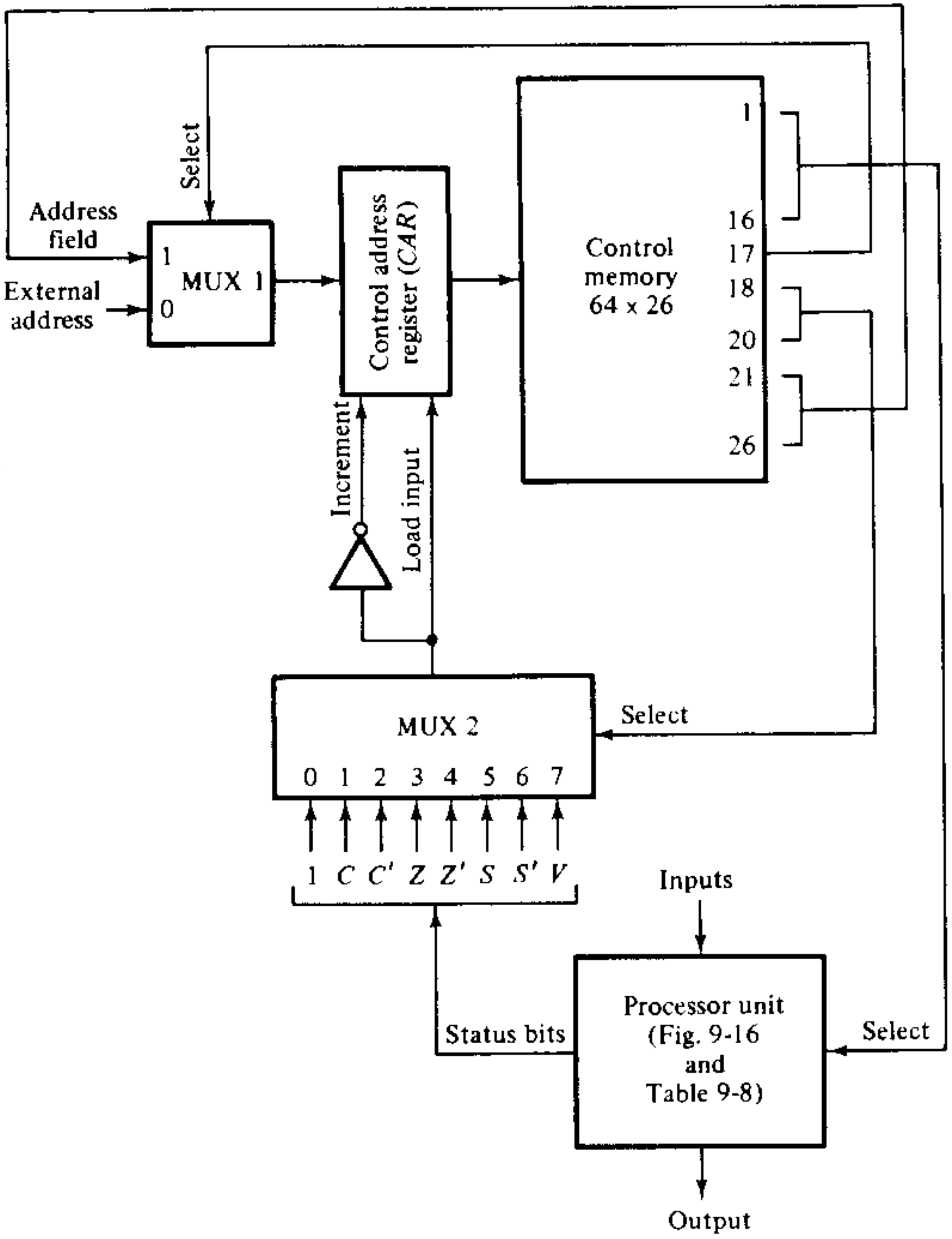
ROM address			ROM outputs									Address			Select	
			x	s_2	s_1	s_0	C_{in}	L	y	z	w					
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	1
1	0	1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



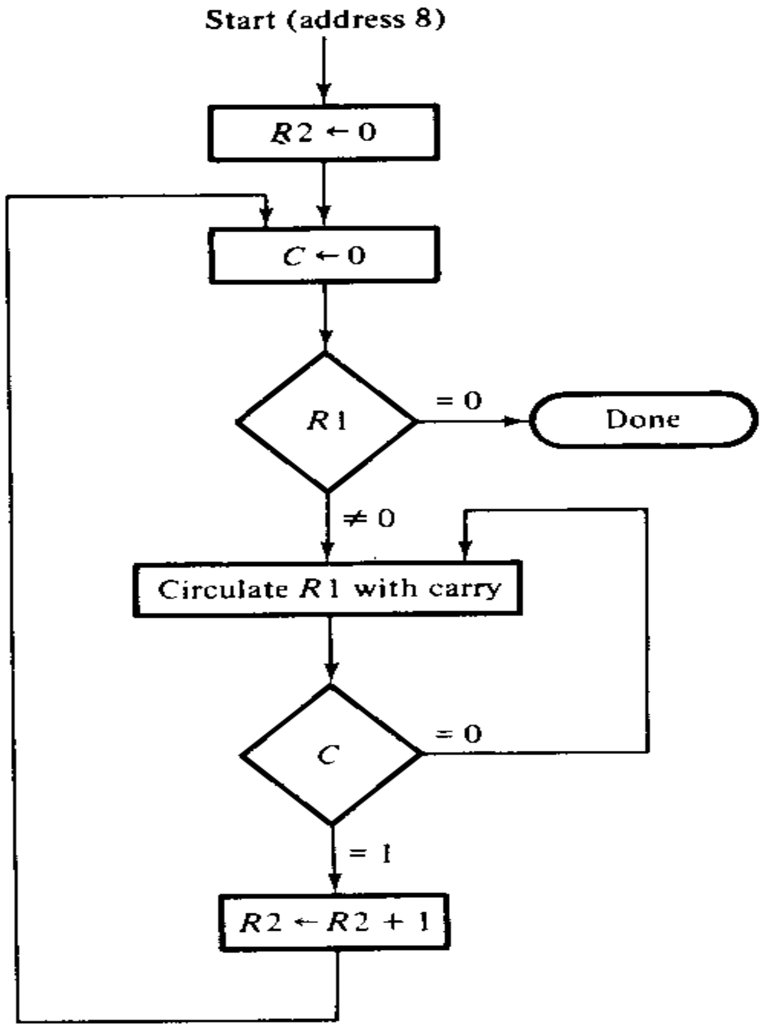
Binary code			A	B
0	0	0	Input data	Input da
0	0	1	$R1$	$R1$
0	1	0	$R2$	$R2$
0	1	1	$R3$	$R3$
1	0	0	$R4$	$R4$
1	0	1	$R5$	$R5$
1	1	0	$R6$	$R6$
1	1	1	$R7$	$R7$

Control of Processor Unit

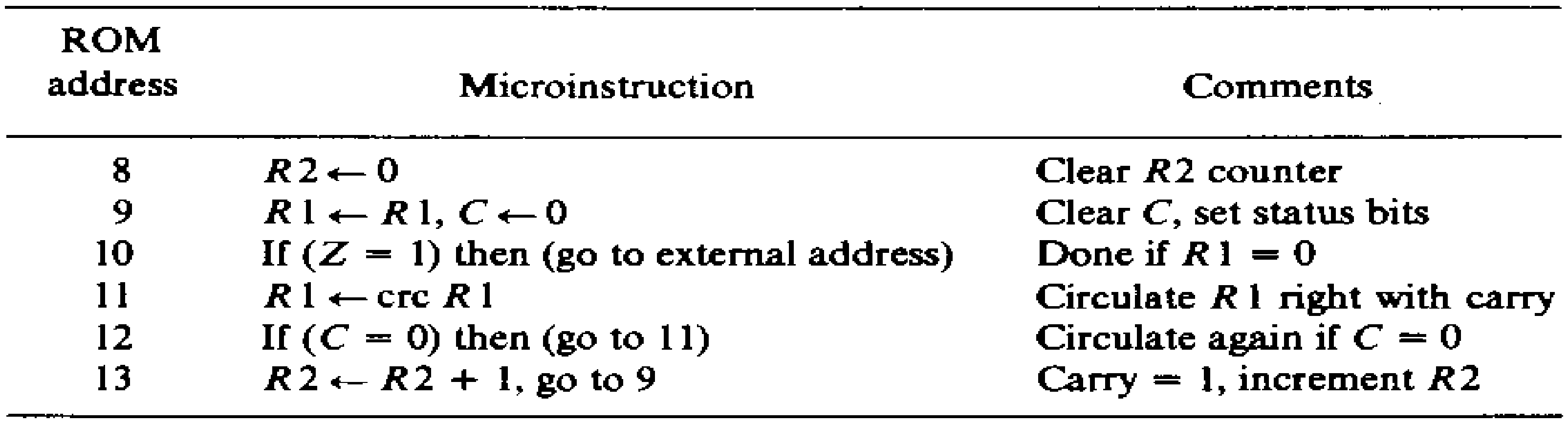
- To construct correct microprograms, it is necessary to specify exactly how the status bits are affected by each microoperation in the processor.
- Th S (sign) and Z (Zero) bits are affected by all operations.
- The C(Carry) and V (Overflow) bits do not change after the following ALU operations:
  - The four logic operations OR,AND,XOR, and complement.
  - The increment and decrement operations.



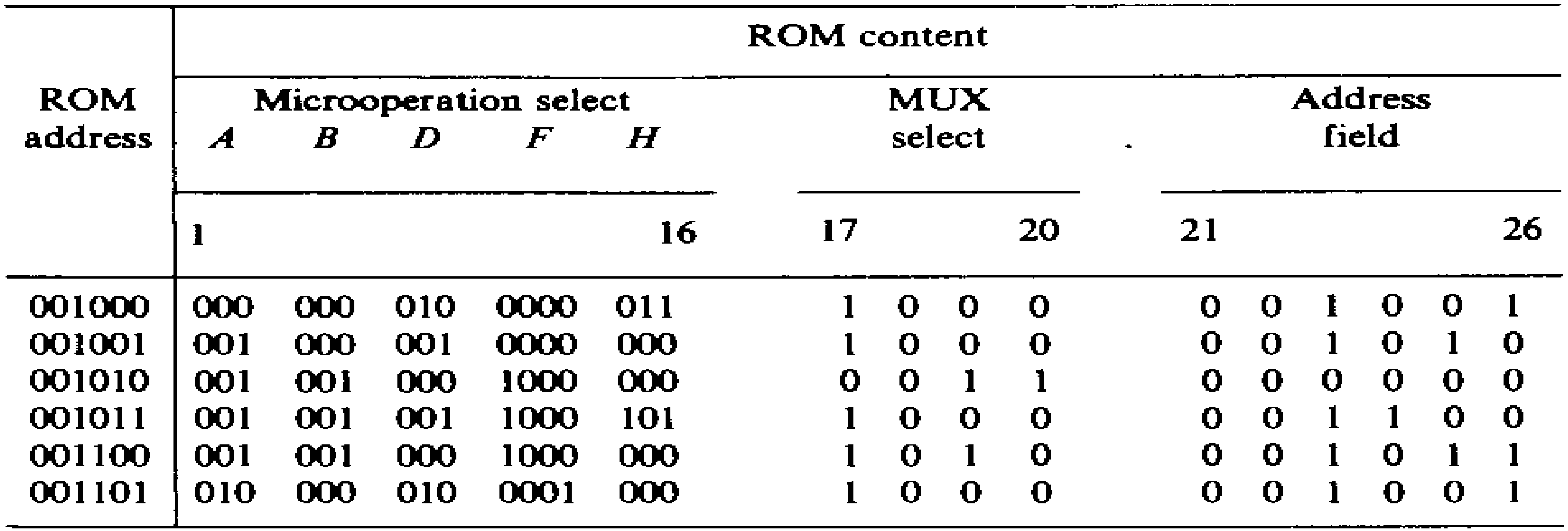
Flow chart for counting the number of 1’s in register R1



Binary microprogram to count the number of 1’s in R1



ROM address	Microinstruction	Comments
8	$R2 \leftarrow 0$	Clear $R2$ 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 $R1$ 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 $R2$



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

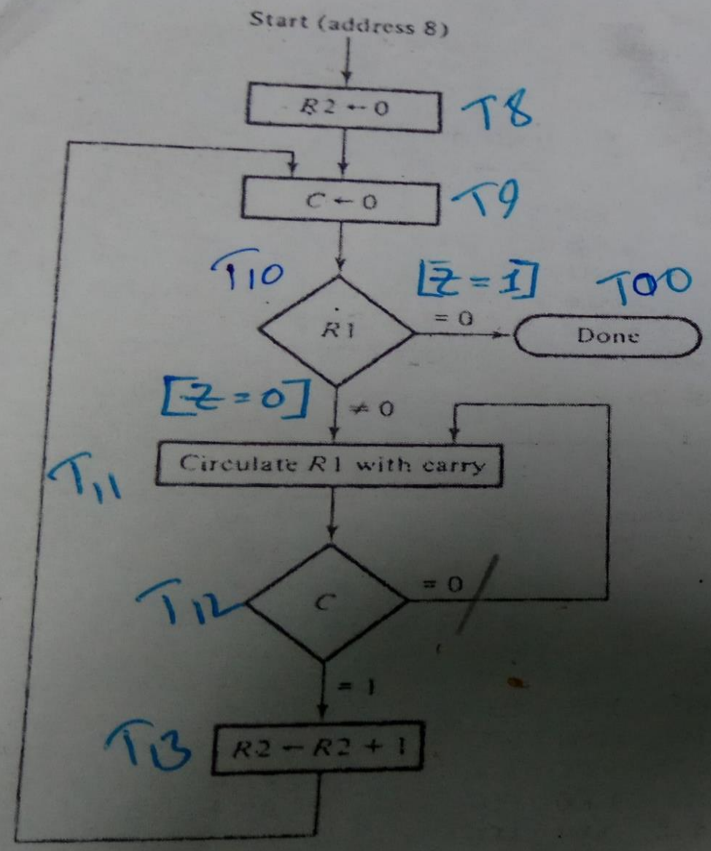


Figure 10-12 Flowchart for counting the number of 1's in register R1

Table 10-4 Symbolic microprogram to count the number of 1's in R1

ROM address	Microinstruction	Comments
8	$R2 \leftarrow 0$	Clear R2 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 R1 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 R2

Table 10-5 Binary microprogram to count the number of 1's in R1

ROM address	ROM content						
	Microoperation select					MUX select	Address field
	A	B	D	F	H		
	1				16	17 20	21 26
T_8 001000	000	000	010	0000	011	1 0 0 0	0 0 1 0 0 1 T_9
T_9 001001	001	000	001	0000	000	1 0 0 0	0 0 1 0 1 0 T_{10}
T_{10} 001010	001	001	000	1000	000	0 0 1 1 → Z	0 0 0 0 0 0 T_0
T_{11} 001011	001	001	001	1000	101	1 0 0 0	0 0 1 1 0 0
T_{12} 001100	001	001	000	1000	000	1 0 1 0	0 0 1 0 1 1
T_{13} 001101	010	000	010	0001	000	1 0 0 0	0 0 1 0 0 1

load op, $0 = \overline{C}$

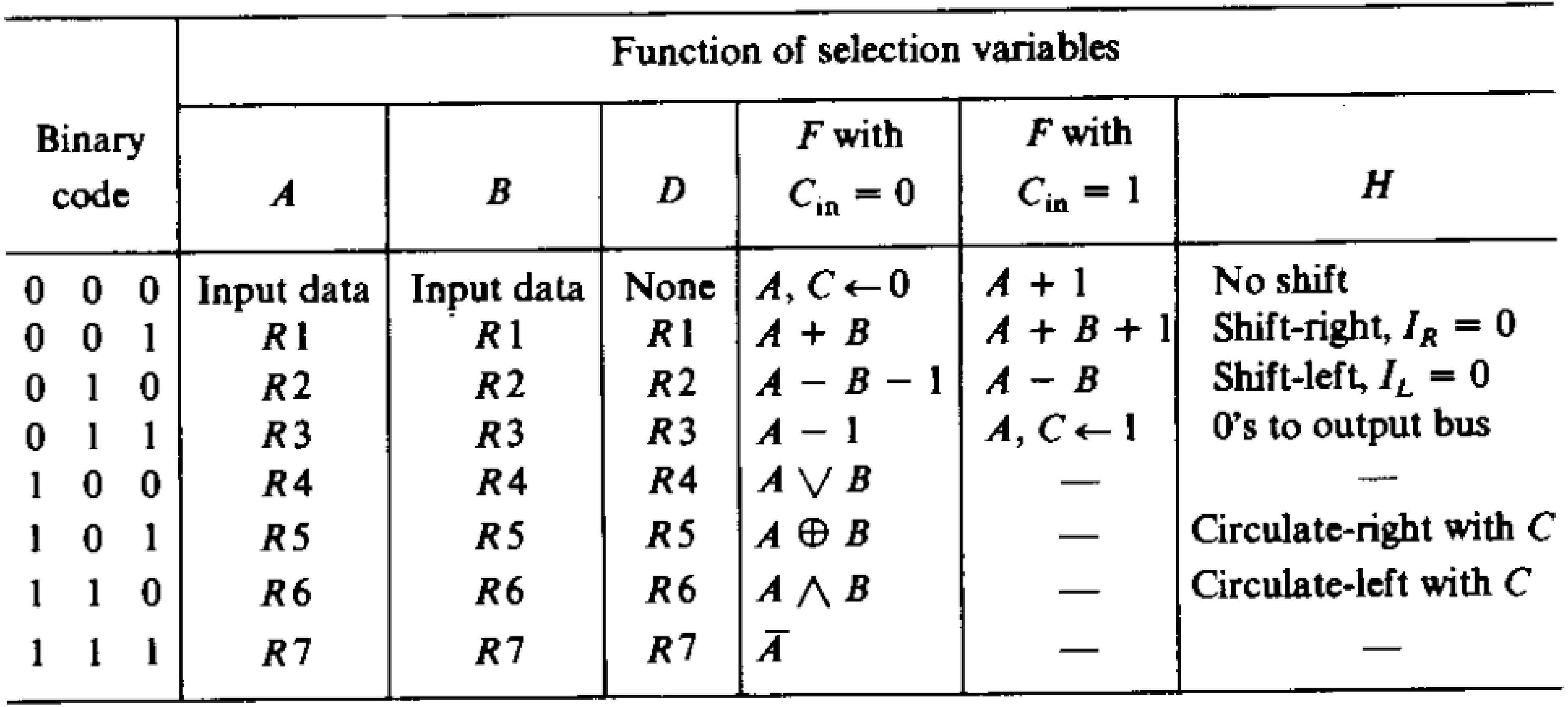
next Address $\underline{1} = C$

→ Z = 1, load o/p

→ Z = 0, next add
(T_{11})

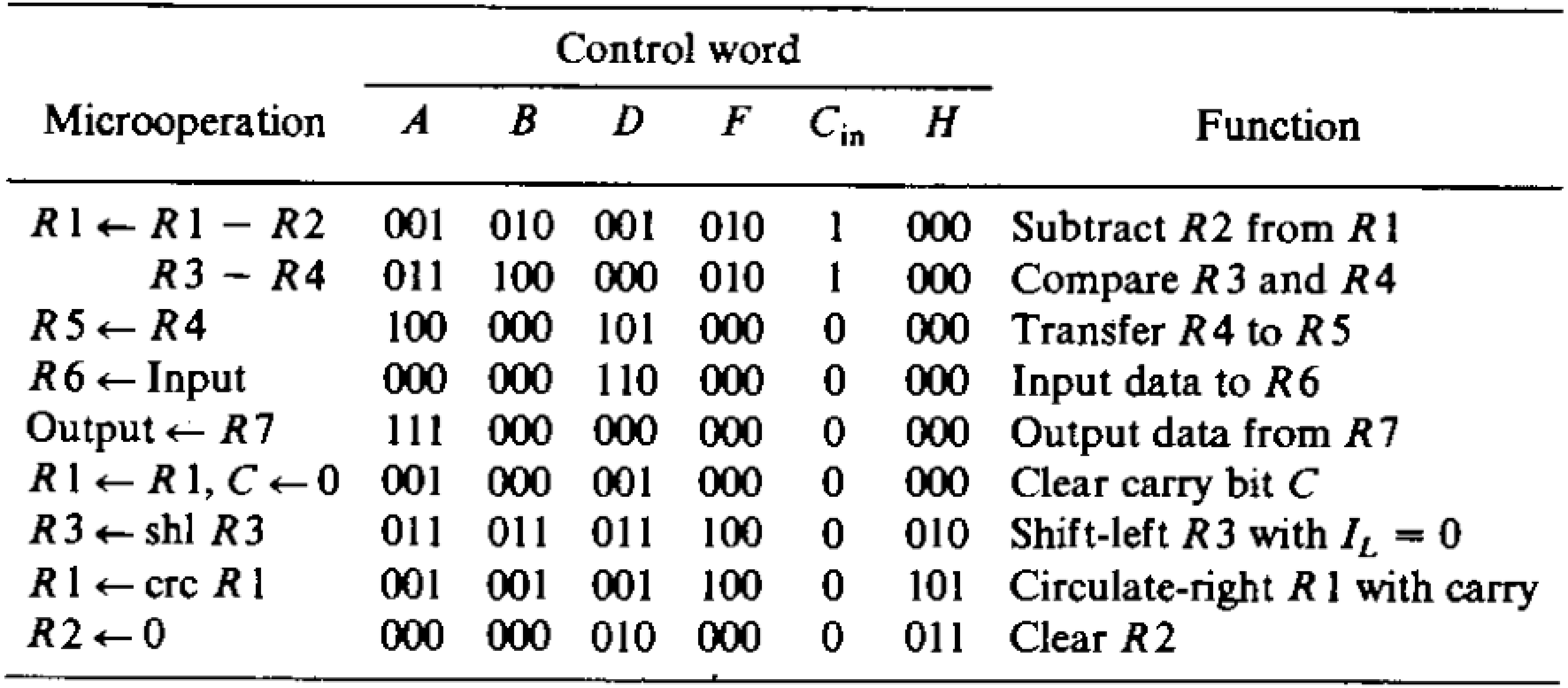
→ External (0)
Address or
next Address

TABLE 9-8 Functions of control variables for the processor of Fig. 9-16



Binary code	Function of selection variables					
	A	B	D	F with $C_{in} = 0$	F with $C_{in} = 1$	H
0 0 0	Input data	Input data	None	$A, C \leftarrow 0$	$A + 1$	No shift
0 0 1	$R1$	$R1$	$R1$	$A + B$	$A + B + 1$	Shift-right, $I_R = 0$
0 1 0	$R2$	$R2$	$R2$	$A - B - 1$	$A - B$	Shift-left, $I_L = 0$
0 1 1	$R3$	$R3$	$R3$	$A - 1$	$A, C \leftarrow 1$	0's to output bus
1 0 0	$R4$	$R4$	$R4$	$A \vee B$	—	—
1 0 1	$R5$	$R5$	$R5$	$A \oplus B$	—	Circulate-right with C
1 1 0	$R6$	$R6$	$R6$	$A \wedge B$	—	Circulate-left with C
1 1 1	$R7$	$R7$	$R7$	$\bar{A}$	—	—

TABLE 9-9 Examples of microoperations for processor



	Control word						
Microoperation	A	B	D	F	C_{in}	H	Function
$R1 \leftarrow R1 - R2$	001	010	001	010	1	000	Subtract $R2$ from $R1$
$R3 - R4$	011	100	000	010	1	000	Compare $R3$ and $R4$
$R5 \leftarrow R4$	100	000	101	000	0	000	Transfer $R4$ to $R5$
$R6 \leftarrow$ Input	000	000	110	000	0	000	Input data to $R6$
Output $\leftarrow R7$	111	000	000	000	0	000	Output data from $R7$
$R1 \leftarrow R1, C \leftarrow 0$	001	000	001	000	0	000	Clear carry bit C
$R3 \leftarrow$ shl $R3$	011	011	011	100	0	010	Shift-left $R3$ with $I_L = 0$
$R1 \leftarrow$ crc $R1$	001	001	001	100	0	101	Circulate-right $R1$ with carry
$R2 \leftarrow 0$	000	000	010	000	0	011	Clear $R2$