AMERICAN INTERNAIONAL UNIVERSITY – BANGLADESH

Where leaders are created

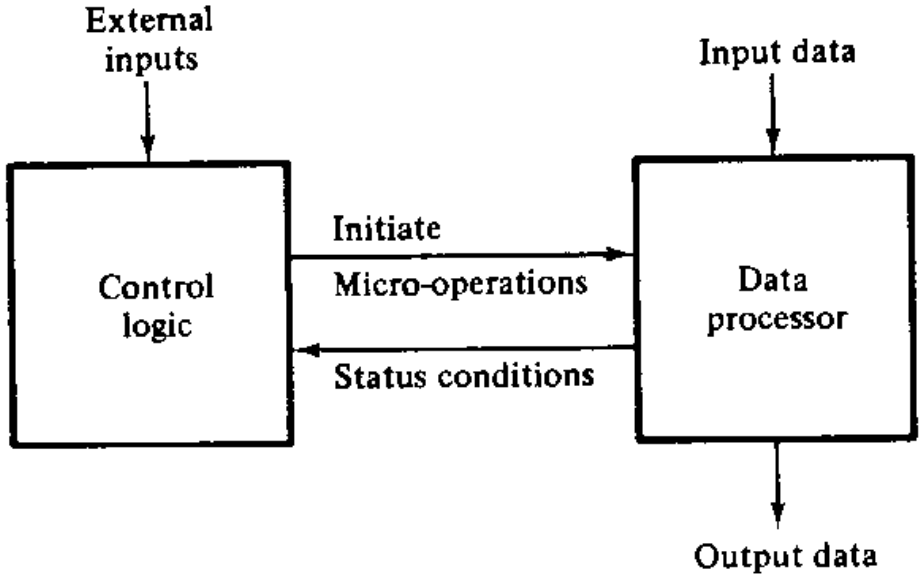
Control Logic Design

[Ch # 10]

Department of EEE

Introduction

- The process of logic design is a complex undertaking.
- The relationship between the control and the data processor in a digital system is shown in below figure.
- The data processor part may be general purpose processor unit or it may consist of individual registers.
- The control initiates all microoperations in the data processor.



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

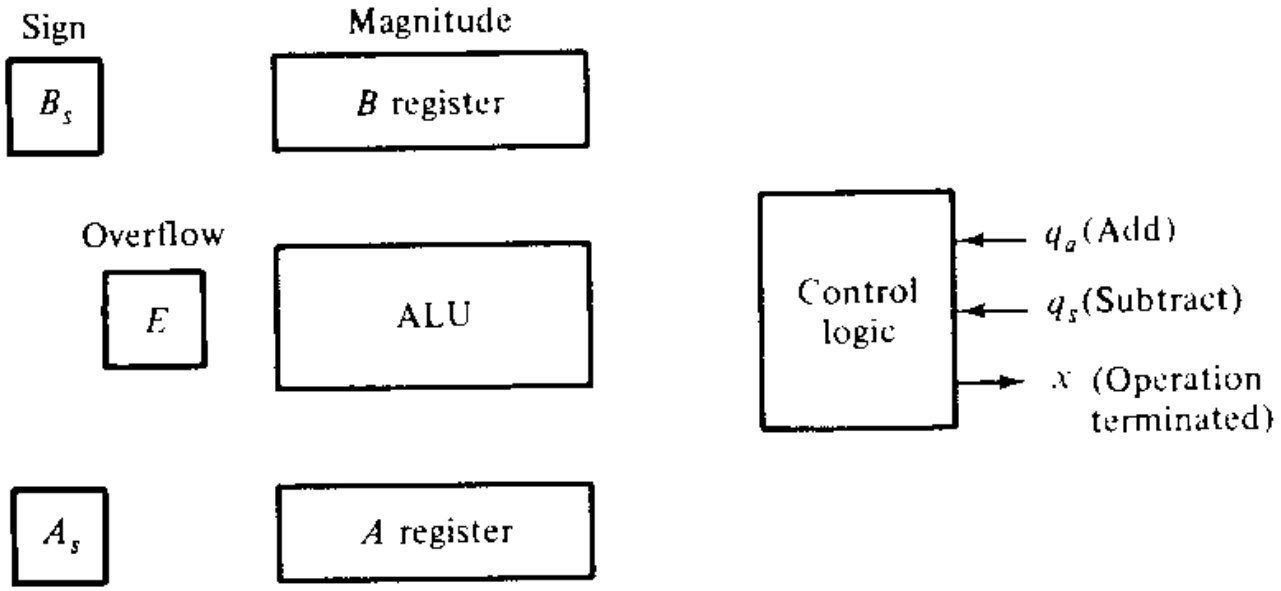
1. Statement of the problem

- 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	

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 Flip Flops 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

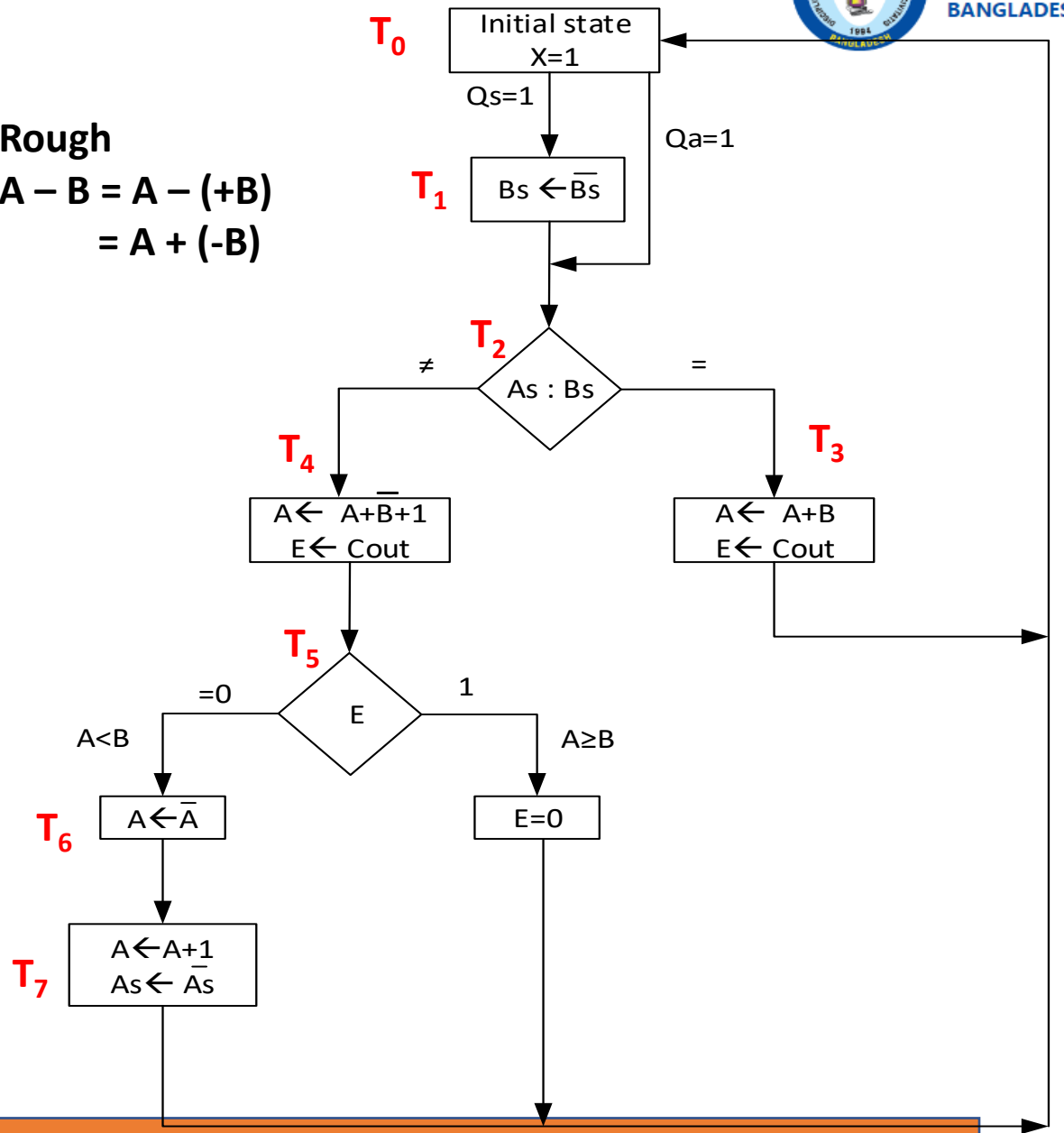
An operation is initiated by either input q_s or input q_a. Input q_s initiates a subtraction operation, so the sign of B is complemented. Input q_a initiates an add operation and the sign of B is unchanged. The next step is to compare the two signs. The decision block marked with As:Bs symbolizes this decision. If the signs are equal, we take the path marked by the symbol =. Otherwise , we take the path marked by ≠

For equal sign, the contents of A is added to the contents 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 output carry C_{out}. The circuit then goes to its initial state and output 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.

Rough

A – B = A – (+B)

= A + (-B)



Flowchart for sign magnitude addition and subtraction contd..

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>B and the number in A is the correct result. The sign of the result again is equal to the original value of As. 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 As. The 2's complement of A can be done with one microoperation, A← A'+1. However, we want to use the previously designed ALU which 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.

Question: Draw a flow chart of sign magnitude addition and subtraction of A & B, where Q_A= 1 for addition and Q_S =1 for subtraction, the result stored in A and carry out is stored in E after ALU operation, and A_s and B_s are the sign of A and B respectively.

Twist:

***P – Q or P + Q**

***Ps and Qs are the sign of P and Q**

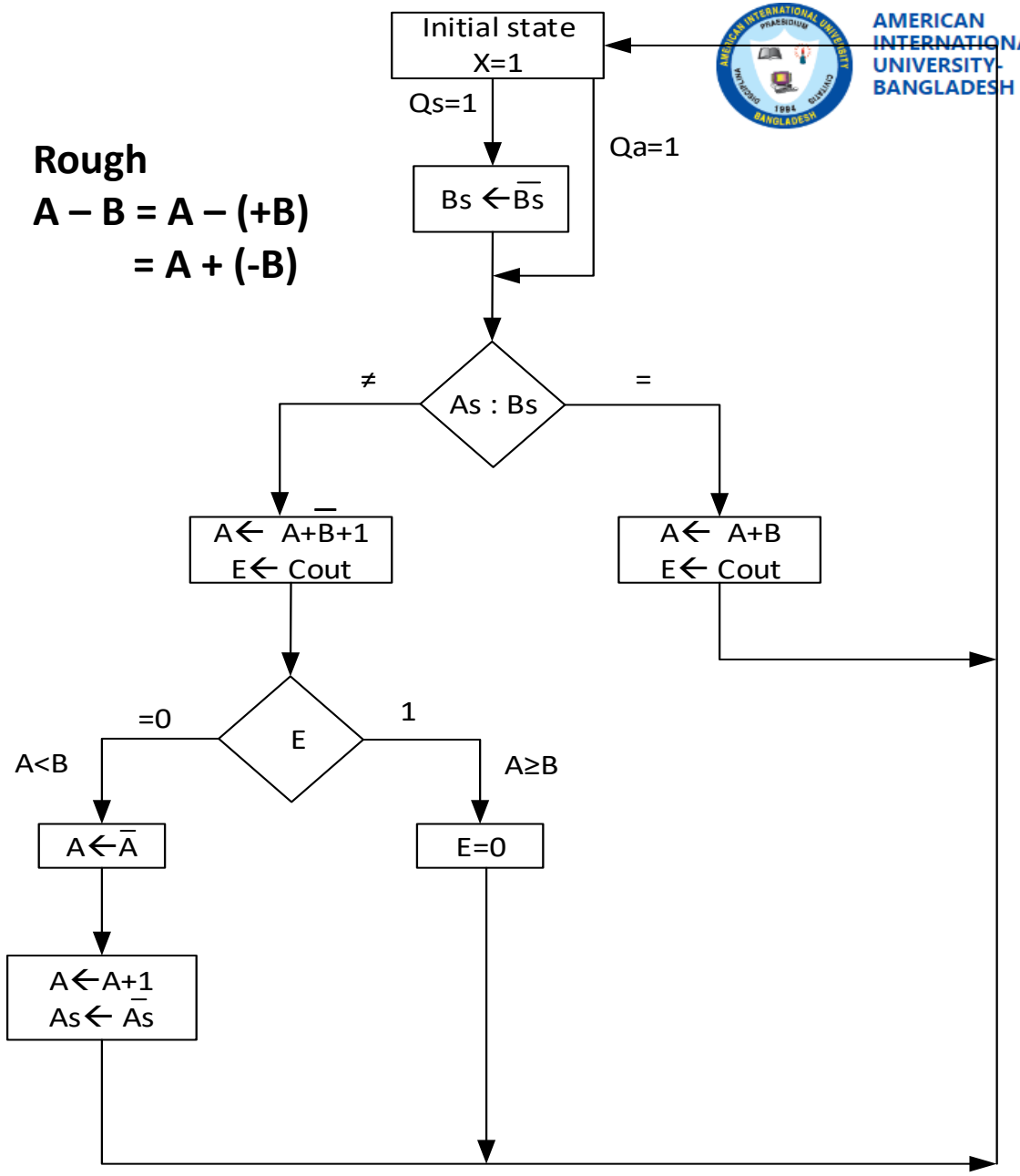
Example

Addition : +6 & +5
-6 & -5

Subtraction : +6 & +5
+5 & +6
-6 & -5
-5 & -6

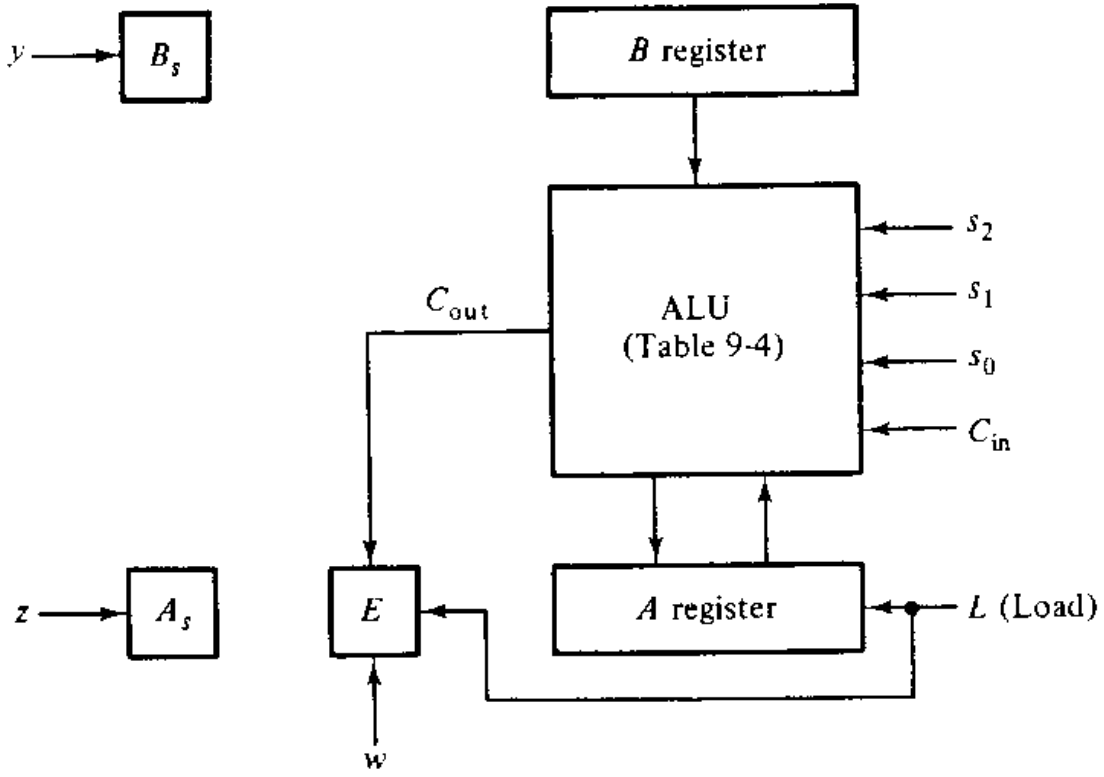
Rough

A – B = A – (+B)
= A + (-B)



4. Data Processor Register

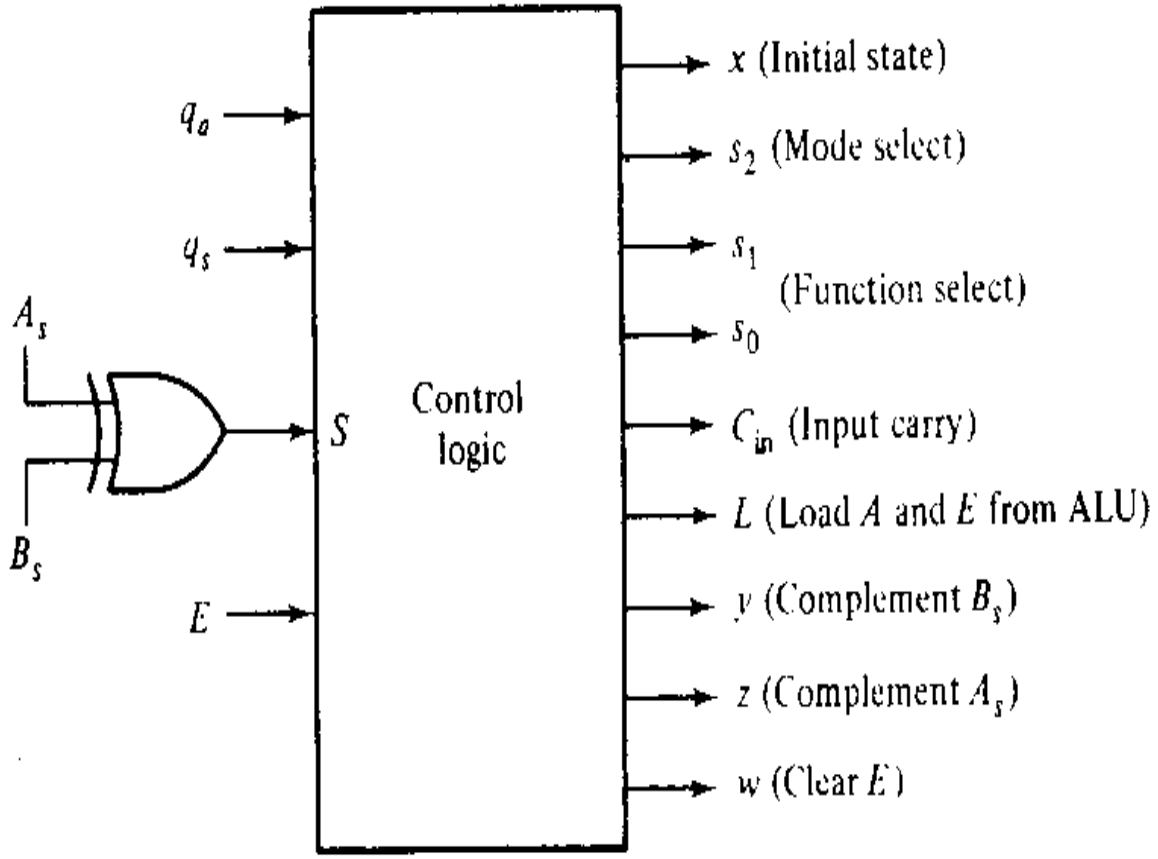
- The operations between A and B can be done with the ALU.
- The operations with As, Bs and E must be initiated with separate control variables.



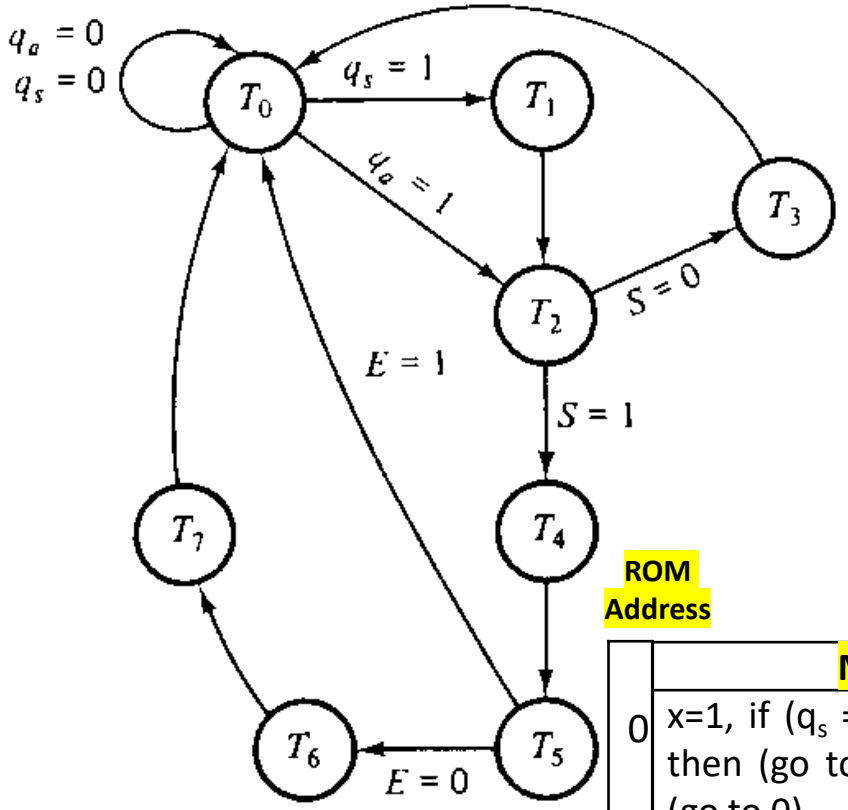
5. Control Block Diagram

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

S = As xor Bs



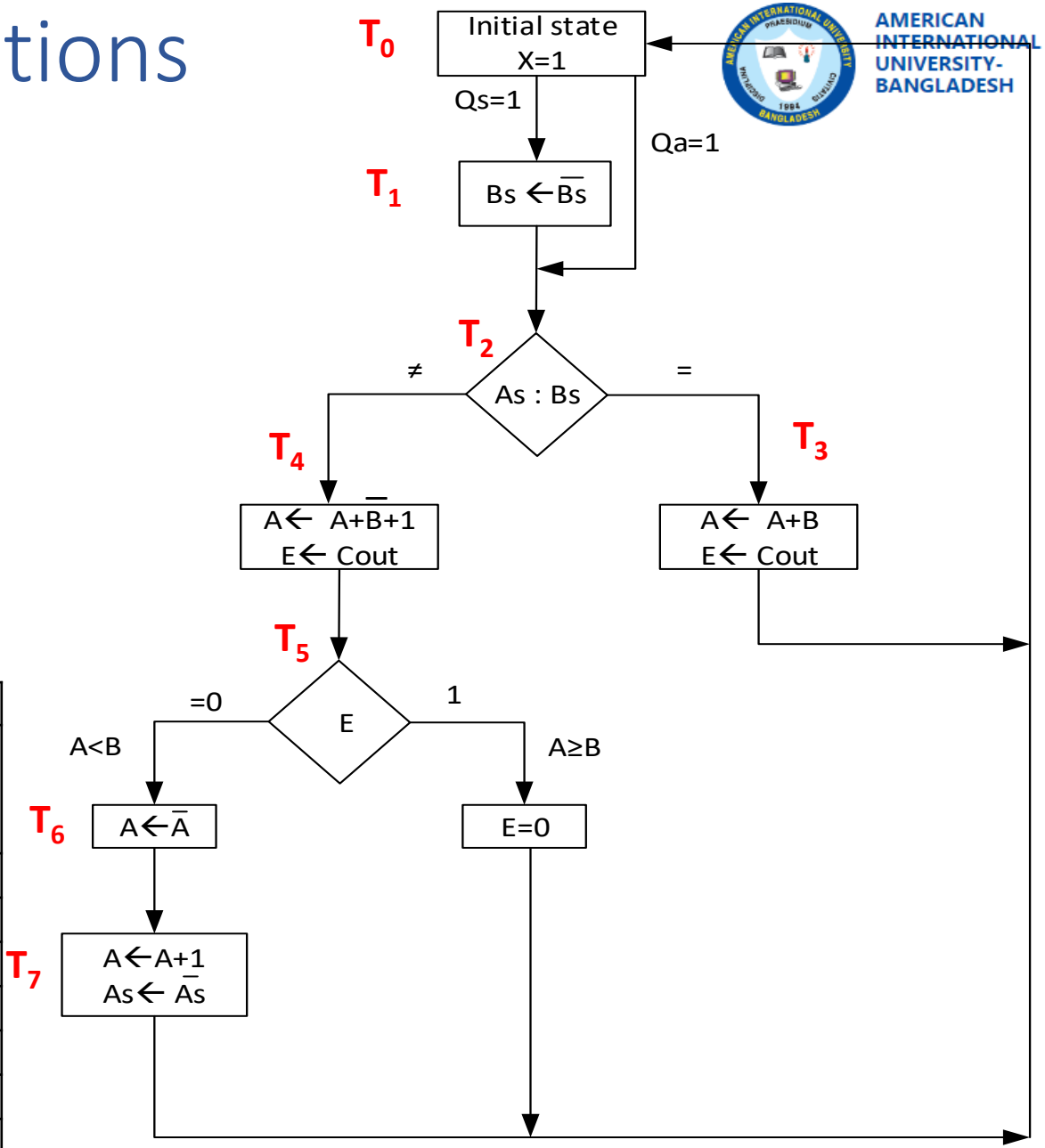
State Diagram and Microinstructions



State Diagram

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

ROM Address	Microinstructions
0	x=1, if (q_s =1) then (go to 1), If (q_a=1) then (go to 2), if (q_a and q_s = 0) then (go to 0)
1	Bs ← Bs'
2	If (S = 1), then *go to 4*
3	A← A+B, E←Cout, *go to 0*
4	A←A+B'+1, E←Cout
5	If(E = 1) then *go to 0*, E←0
6	A←A'
7	A←A+1, As←As', *go to 0*



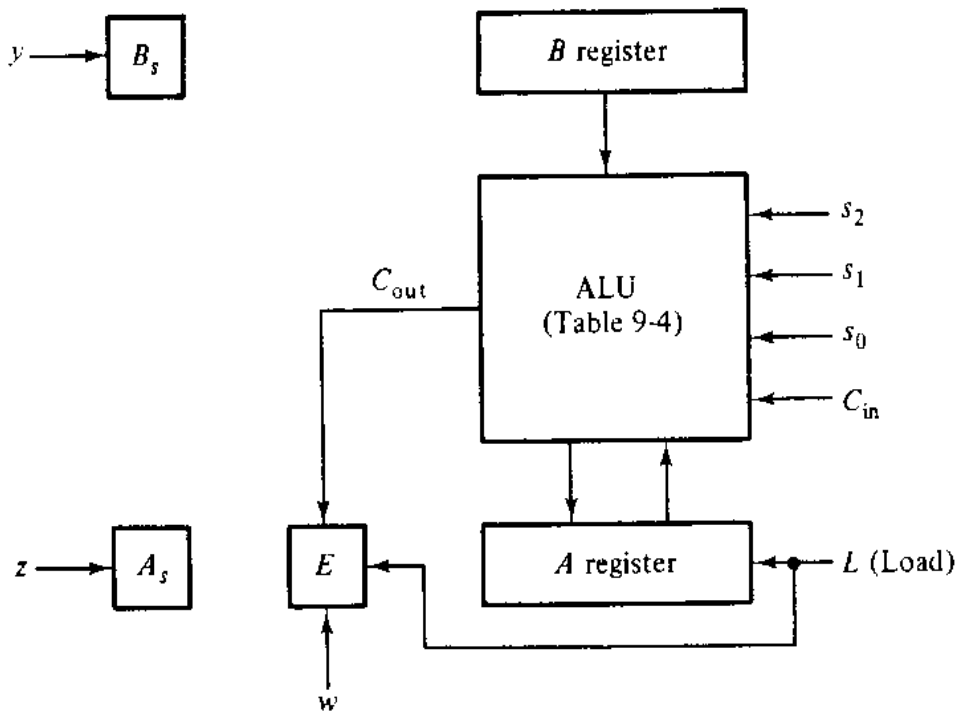
Sequence of register transfers

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

Boolean Function

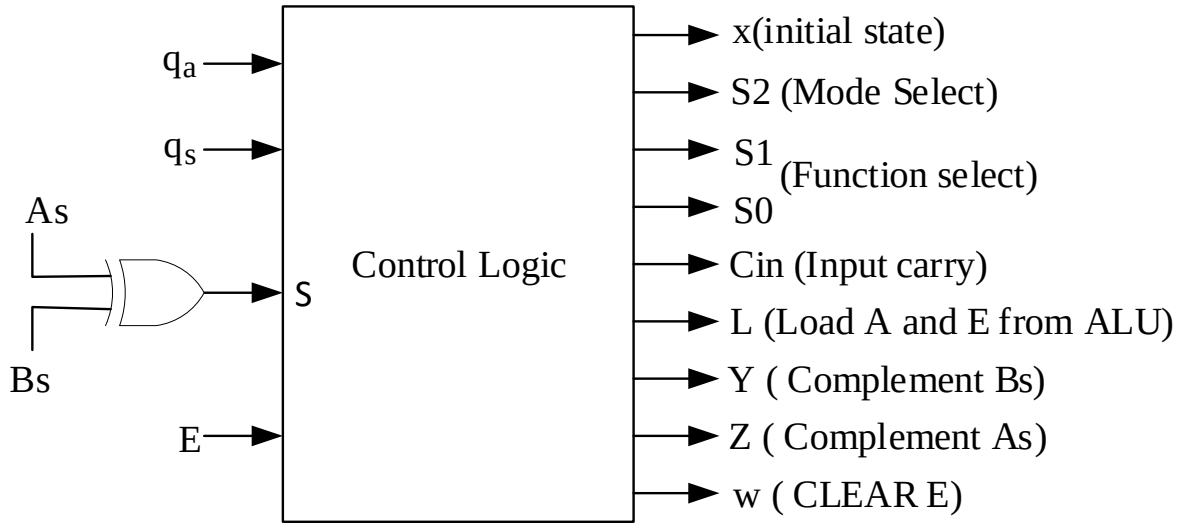
$x = T_0$
$s_2 = T_6$
$s_1 = T_4 + T_6$
$s_0 = T_3 + T_6$
$C_{in} = T_4 + T_7$
$L = T_3 + T_4 + T_6 + T_7$
$y = T_1$
$z = T_7$
$w = T_5$

Binary code	F with $C_{in} = 0$	F with $C_{in} = 1$
0 0 0	$A, C \leftarrow 0$	$A + 1$
0 0 1	$A + B$	$A + B + 1$
0 1 0	$A - B - 1$	$A - B$
0 1 1	$A - 1$	$A, C \leftarrow 1$
1 0 0	$A \vee B$	—
1 0 1	$A \oplus B$	—
1 1 0	$A \wedge B$	—
1 1 1	$\overline{A}$	—



To clear 'E', w pin needs to be activated

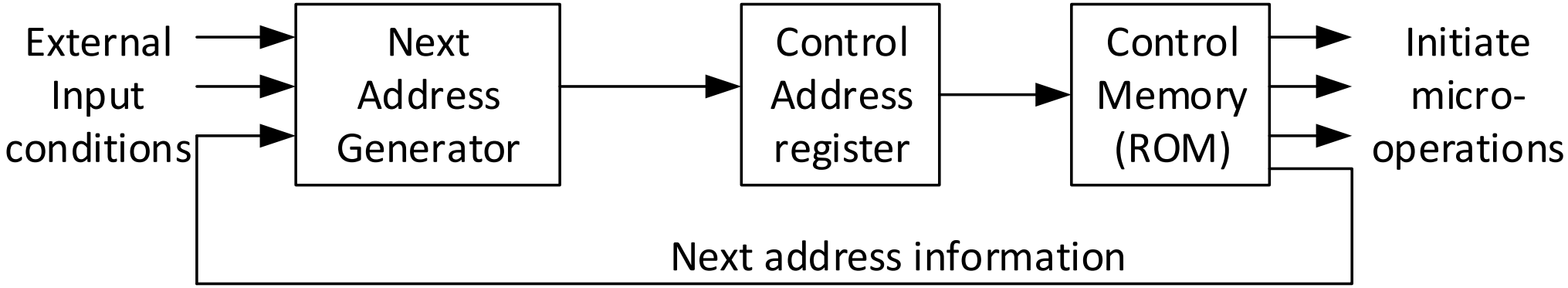
Q1. Design control state diagram for the previous flowchart. Develop control outputs for the signals given in following control logic block diagram using the Table and control states you defined.



Binary code	F with $C_{in} = 0$	F with $C_{in} = 1$
0 0 0	$A, C \leftarrow 0$	$A + 1$
0 0 1	$A + B$	$A + B + 1$
0 1 0	$A - B - 1$	$A - B$
0 1 1	$A - 1$	$A, C \leftarrow 1$
1 0 0	$A \vee B$	—
1 0 1	$A \oplus B$	—
1 1 0	$A \wedge B$	—
1 1 1	$\bar{A}$	—

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.**



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 *(sequential addressing)*.
  - Provision for choosing between two addresses *(conditional addressing)* as a function of the 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

Table for the functions of the multiplexer select bits

ROM bits 13	14	MUX Select Function
0	0	Increment CAR
0	1	Load input to CAR
1	0	Load inputs to CAR if S = 1, increment CAR if S = 0
1	1	Load inputs to CAR if E = 1,increment CAR if E = 0

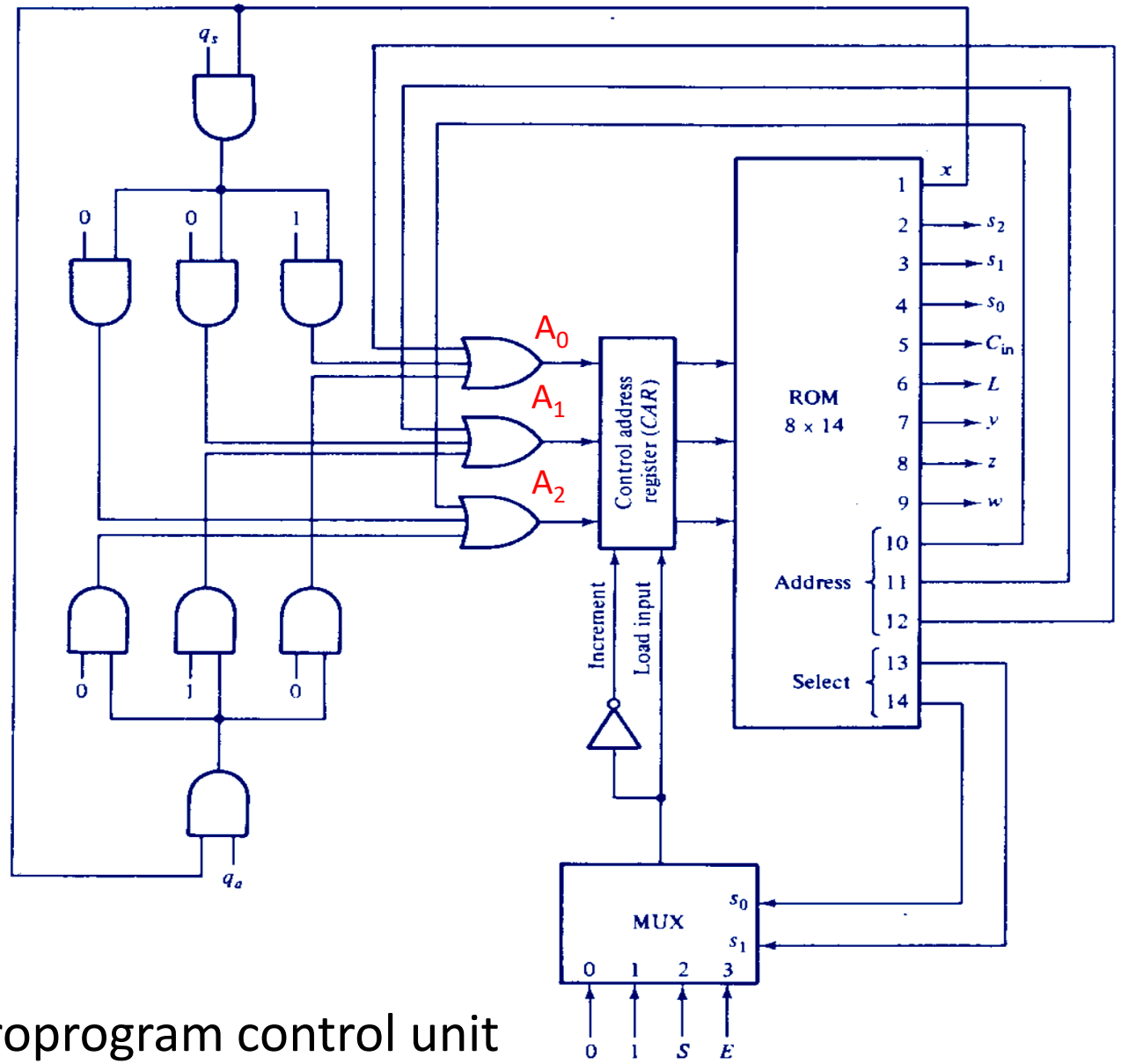


Fig. 10-10 Organization of the microprogram control unit

Hardware Configuration

The control memory is an 8-word (usually, 1 word = 2 contiguous 8-bit bytes, i.e. 1 word = 16 bits) by **14-bit ROM**.

The first 9 bits of a micro-instruction word contain the control variables that initiate the micro-operations. The last 5 bits provide information to select 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. That is, CAR is a **counter** having parallel-load capability.

Hardware Configuration

Bits 10, 11, and 12 of a micro-instruction contain an address for the CAR.

Bits 13 and 14 select an input for a multiplexer.

Bit 1 provides the initial state condition denoted by a variable *x* and also enables an external address when q_s and q_a are equal to 1.

We stipulate that when $x = 1$, the address field of the micro-instruction 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 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.

Hardware Configuration

The multiplexer (MUX) has four inputs that are selected with bits 13 and 14 of the micro-instructions. The functions of the multiplexer select bits are tabulated in the Table of 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 01. The output of the multiplexer is 1, and the external input is loaded into CAR.

The 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 micro-instruction are loaded into the CAR (only if $x = 0$). If $S = 0$, the output of the multiplexer is 0 and the CAR is incremented.

Hardware Configuration

Table for the functions of the multiplexer select bits

ROM bits 13	ROM bits 14	MUX Select Function
0	0	Increment CAR
0	1	Load input to CAR
1	0	Load inputs to CAR if S = 1, increment CAR if S = 0
1	1	Load inputs to CAR if E = 1,increment CAR if E = 0

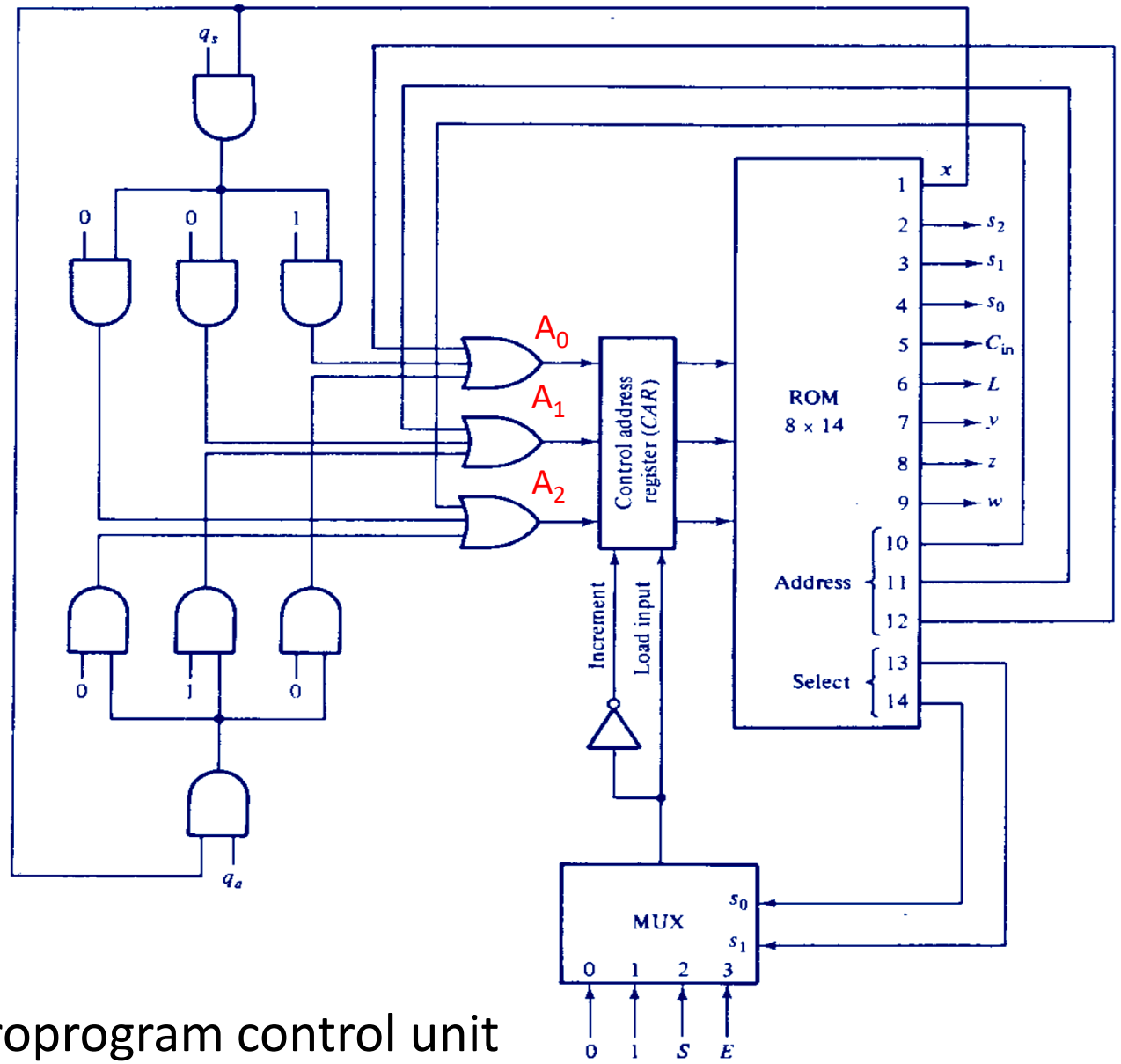


Fig. 10-10 Organization of the microprogram control unit

Hardware Configuration

AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

With bits 13 and 14 equal to 11, the status variable *E* is selected and the address bits of the micro-instruction are loaded into the CAR if *E* = 1, but the CAR is incremented if *E* = 0. Thus, the multiplexer allows the control to choose between two addresses, depending on the value of the selected status bit.

Once the configuration of a micro-program control unit is established, the designer's task is to generate the micro-code for the control memory. This code generation is called micro-programming and is a process that determines the bit configuration for each all words in control memory.

To appreciate this process, we will derive the micro-program for the adder-subtractor example. The control memory has 8 words and each word contains 14 bits. To micro-program the control memory, we must determine the bit values of each of the eight words.

Hardware Configuration

The register-transfer method can be adopted for developing a micro-program. The micro-program 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 micro-instruction 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.

Hardware Configuration

Thus, instead of thinking in terms of the 1's and 0's that must be inserted for each micro-instruction, it is more convenient to think in terms of symbols in the register-transfer method. Once the symbolic micro-program is established, it is possible to translate the register-transfer statements to their equivalent binary form.

The micro-program 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 micro-instruction that must be stored at each address is given in symbolic form. The comments are used to clarify the register-transfer statements.

Microprogram for Control Memory

Table 10-2 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

Control State Diagram

T_0: Initial state $x = 1$

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

T_2: nothing

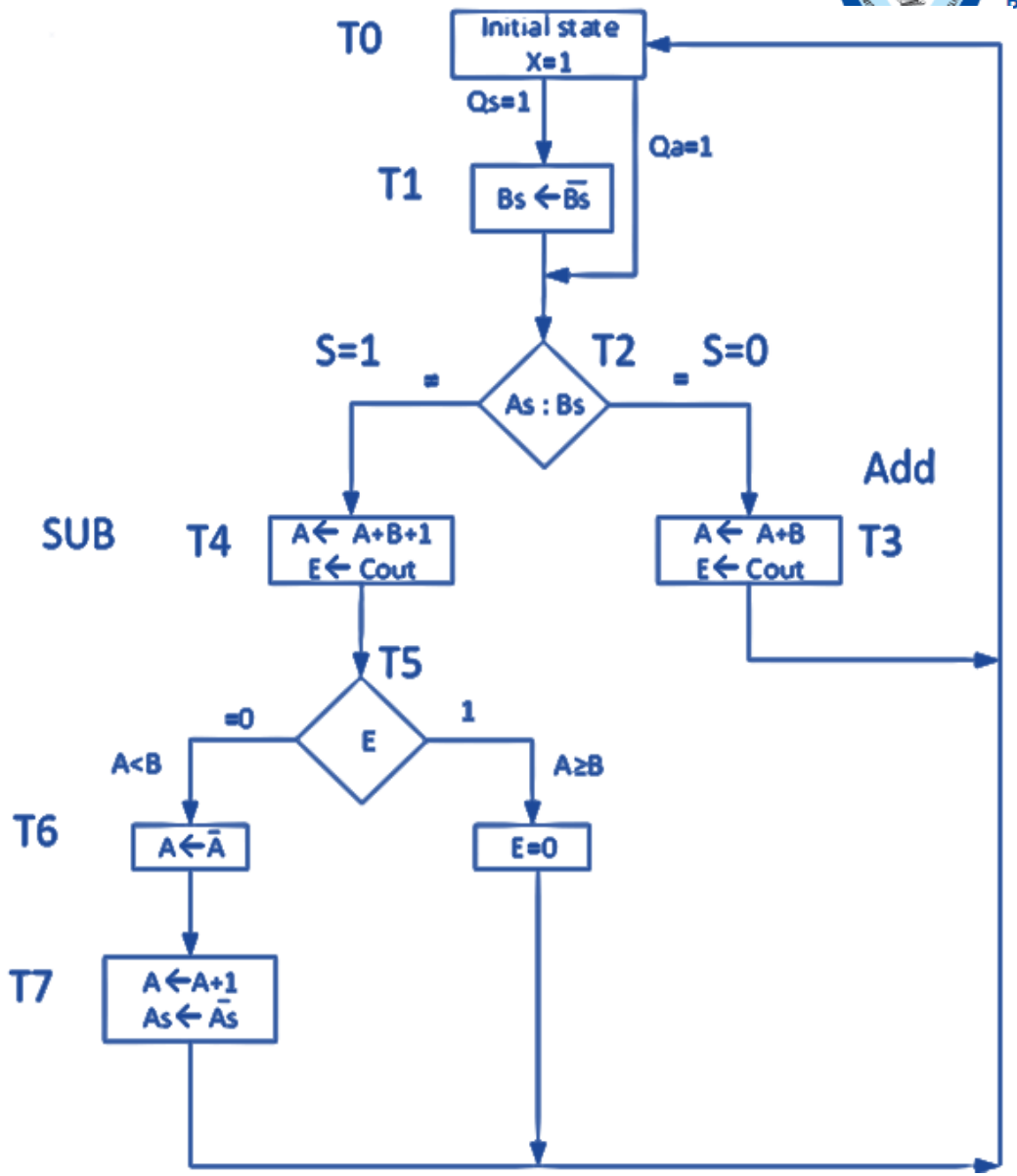
$T_3: A \leftarrow A + B,\ E \leftarrow C_{out}$

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

$T_5: E \leftarrow 0$

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

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



Hardware Configuration

The equivalent binary form of micro-program is given in Table 10-3. The eight binary format addresses of the ROM are listed in the first column.

In the second column, the micro-instruction that must be stored at each address is given in the binary or machine language format.

In the third column, the contents of each word of ROM is given in binary format. This is for programming the ROM.

The first 9 bits in each ROM word give the control word that initiates the specified micro-operations. These are taken from Fig. **10-9 (b).** The last 5 bits in each ROM word are derived from the conditional control statements in the symbolic program.

Microprogram for Control Memory

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

Hardware Configuration

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 micro-instruction use a *go to* statement, 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 micro-instruction.

If q_s or q_a is 1, control goes to address 1 or 2, respectively.

Hardware Configuration

Table for the functions of the multiplexer select bits

ROM bits 13	ROM bits 14	MUX Select Function
0	0	Increment CAR
0	1	Load input to CAR
1	0	Load inputs to CAR if S = 1, increment CAR if S = 0
1	1	Load inputs to CAR if E = 1,increment CAR if E = 0

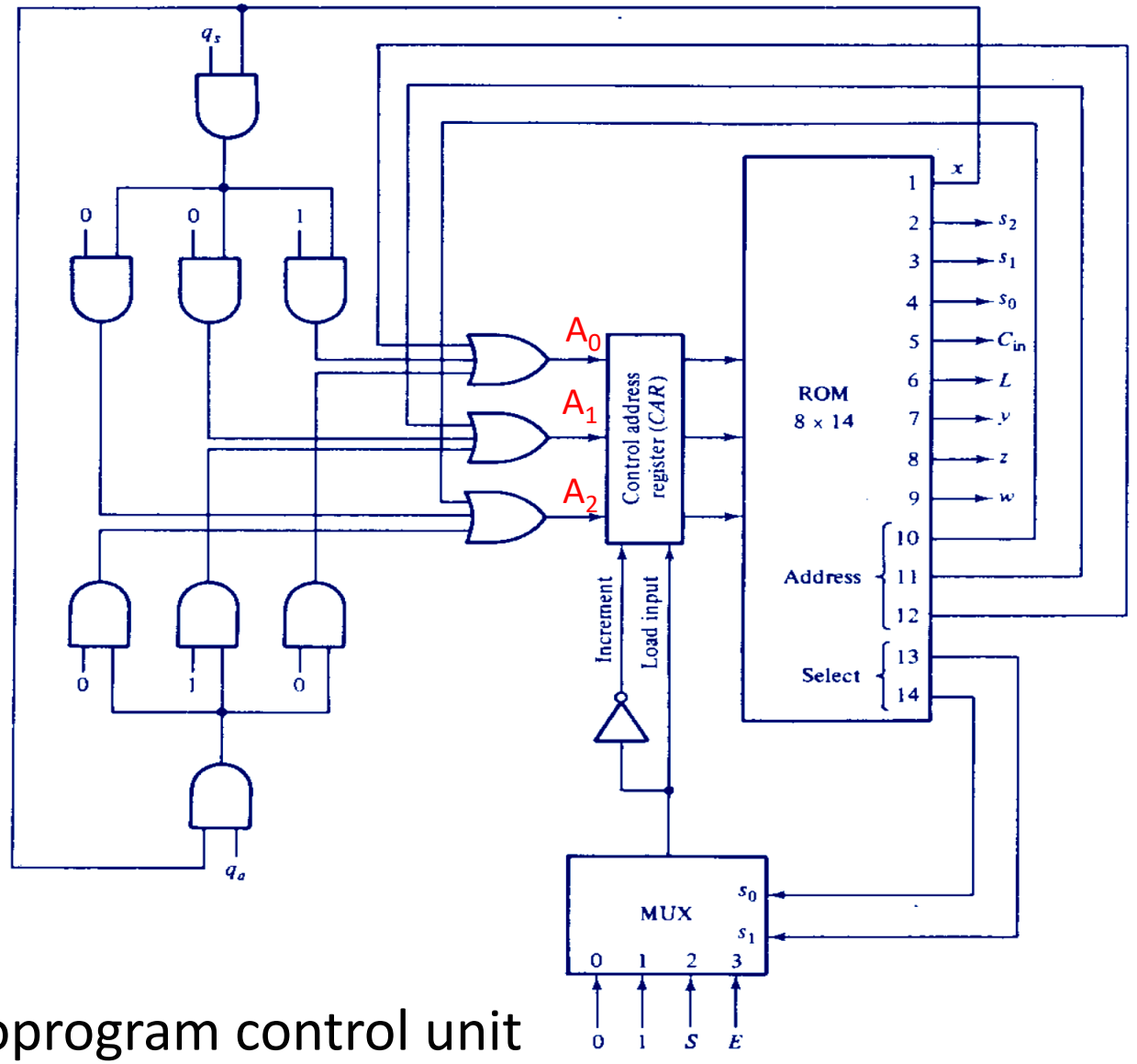


Fig. 10-10 Organization of the microprogram control unit

Example: Flow chart for counting the number of 1's in register R1 and storing in R2.

AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

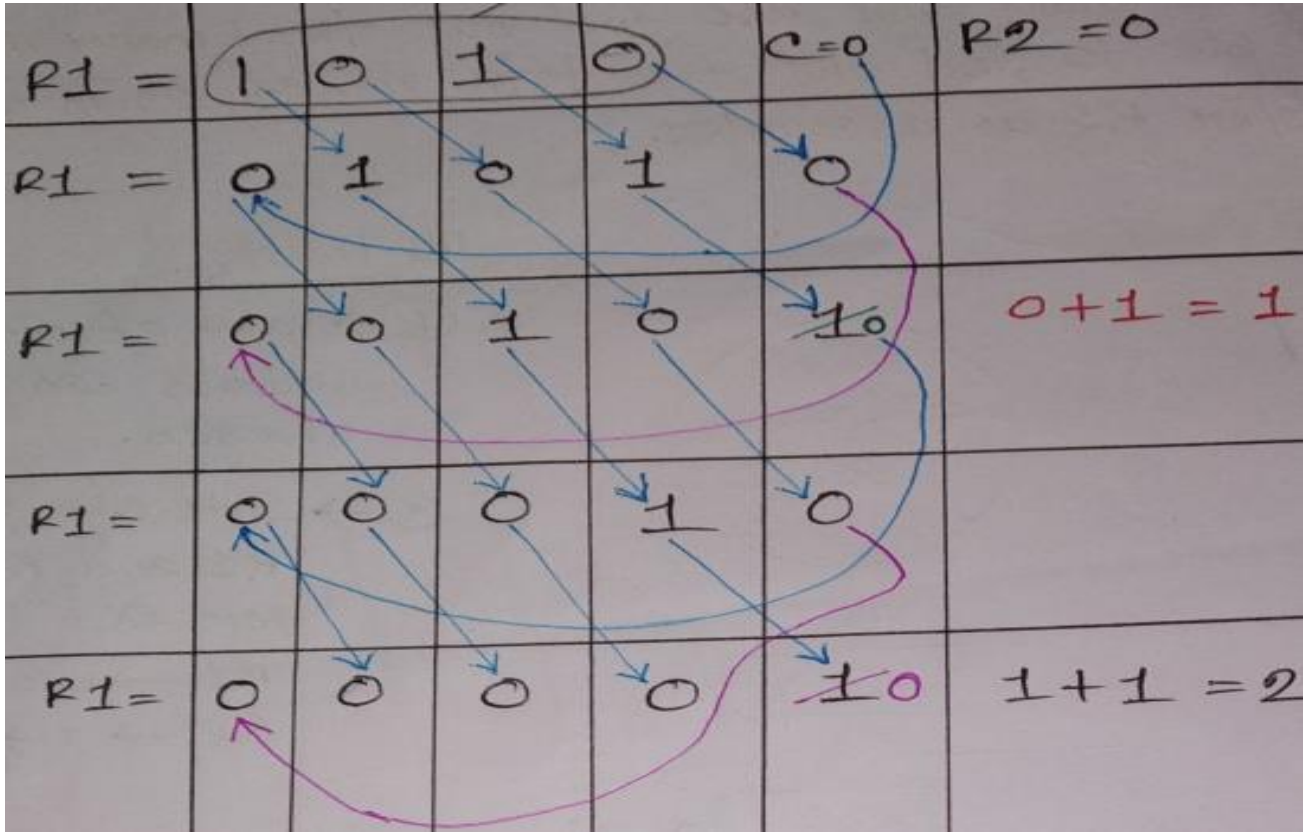
Example

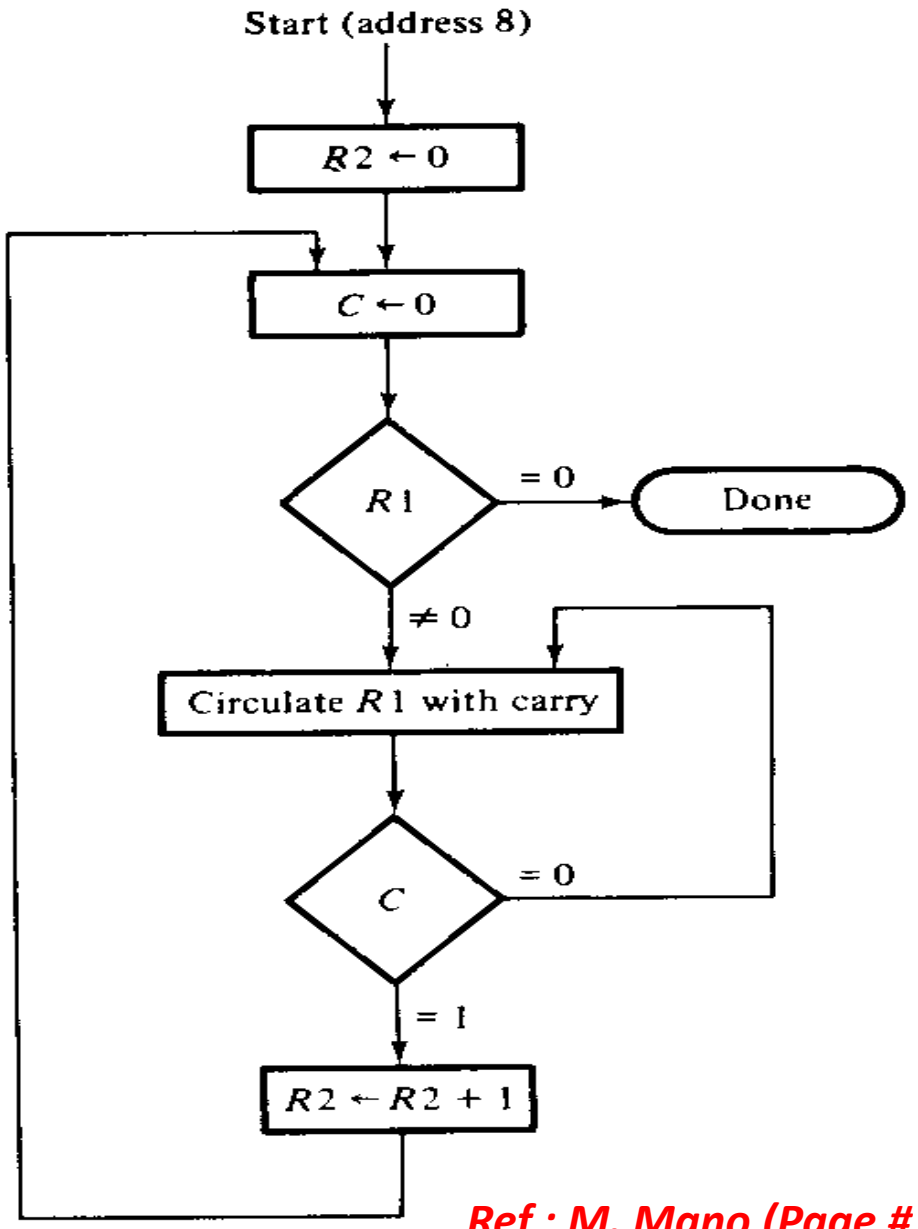
R1 : 1010 ------ two 1's

R2 : binary of two ---- 010

R1 → Register

R2 → Counter





Ref : M. Mano (Page # 432)

Homework

Twist:
***Find total numbers of zeros**
***Register, counter interchanged**

AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Q1. Draw the flow chart to find out the total number of 1's in register, R1. Use register R2 as counter. [See previous example]

Q2. Draw the flow chart to find out the total number of 1's in register, R2. Use register R1 as counter.

Q3. Draw the flow chart to find out the total number of 0's in register, R1. Use register R2 as counter.

Determine the outputs of the R5 (in binary) and R2 (in decimal) registers as well as of the carry flag after each clock cycle or timing state. Determine the number of states that are required to complete the operation.

	R1								C	R2
Timing States	1	0	1	0	1	0	1	0	1	0
T1										
T2										
T3										
T4										
T5										
T6										
T7										
T8										