Unit-2

DataPath Design

Introduction

- An instruction set processor consist of two important units:
  - Data Processing Unit (DataPath)
  - Program Control Unit

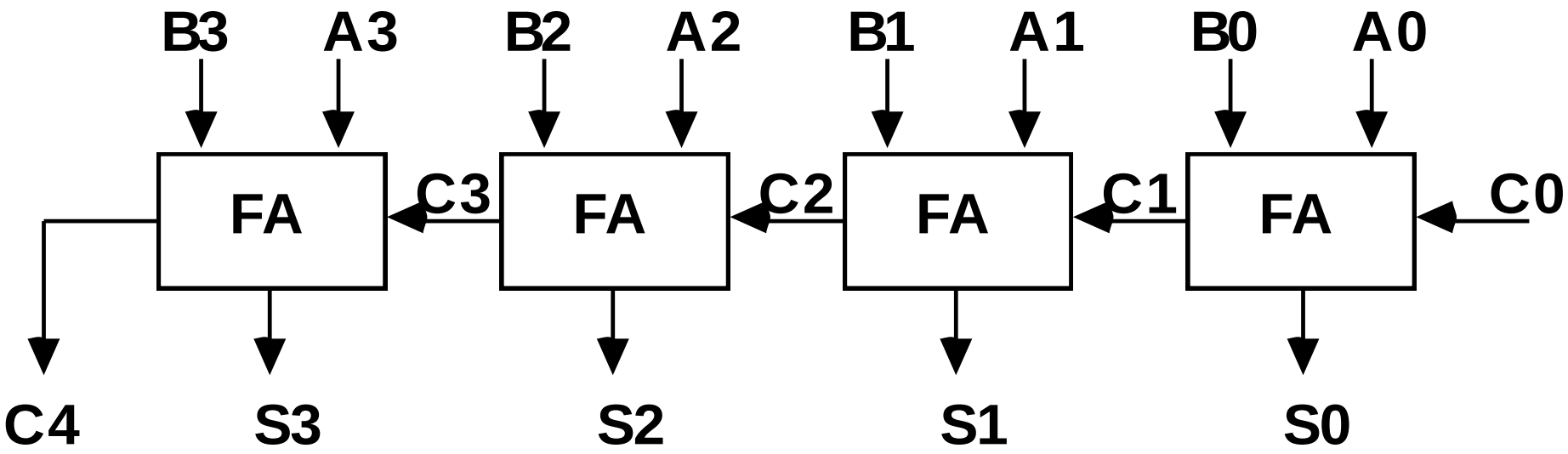
Addition

Fixed Point Arithmetic

- Add & subtract instructions for fixed binary numbers are found in the instruction set of every computer.
- To implement the add ,microoperation with hardware we need registers that hold the data and the digital component that performs the arithmetic addition .
- The digital circuit that forms the arithmetic sum of 2 bits and a previous carry is called a full adder.

- The digital circuit that generates the arithmetic sum of two binary numbers of any length is called a binary adder.
- The binary adder is constructed with full adder circuits connected in cascade.
- For Eg: The below diag shows the interconnections of 4 full adders to provide a 4 bit binary adder.

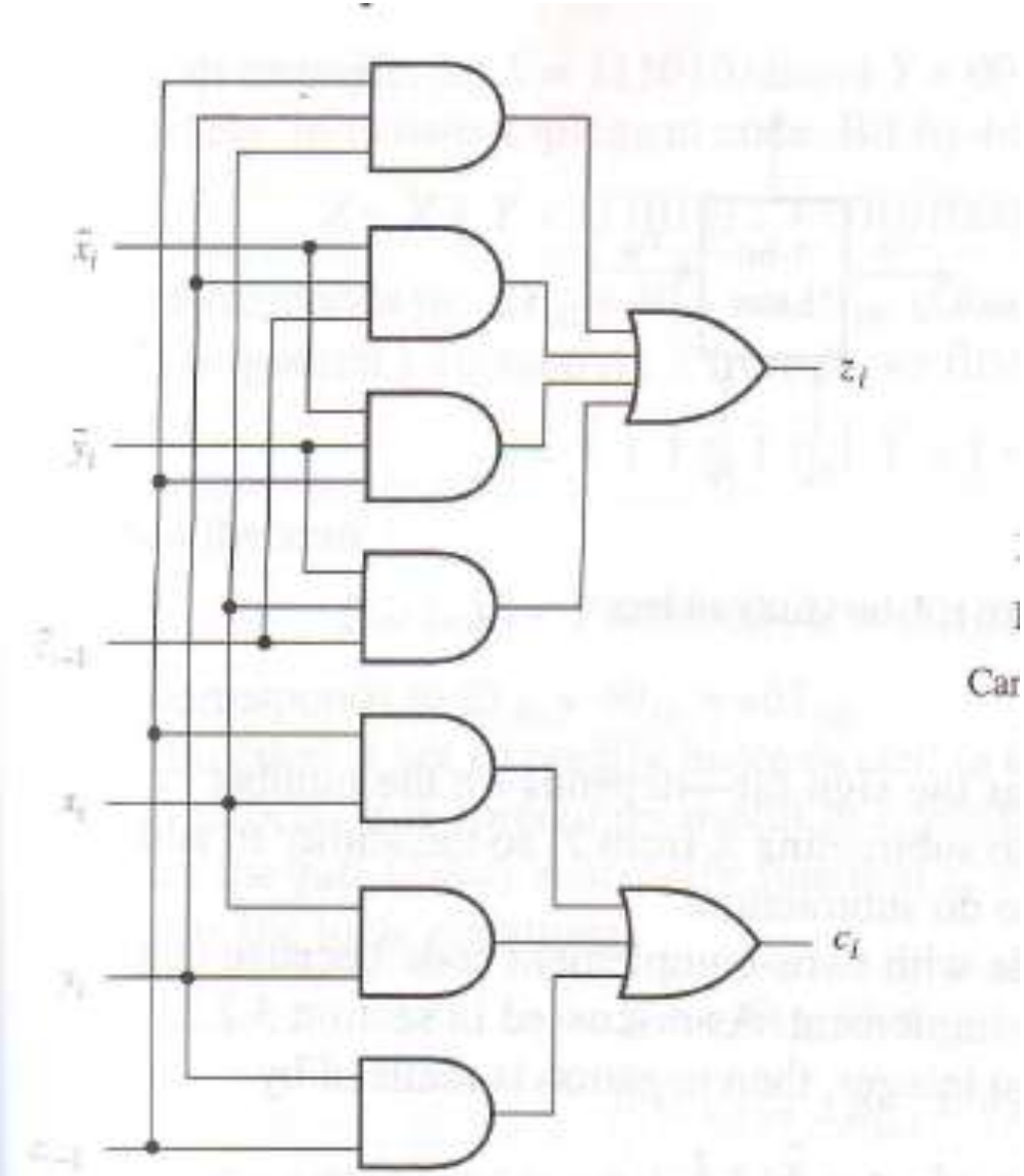
Binary Adder



How We Can Make 1 Bit Full Adder

- By using only AND & OR gates
- By using XOR, AND & OR gates

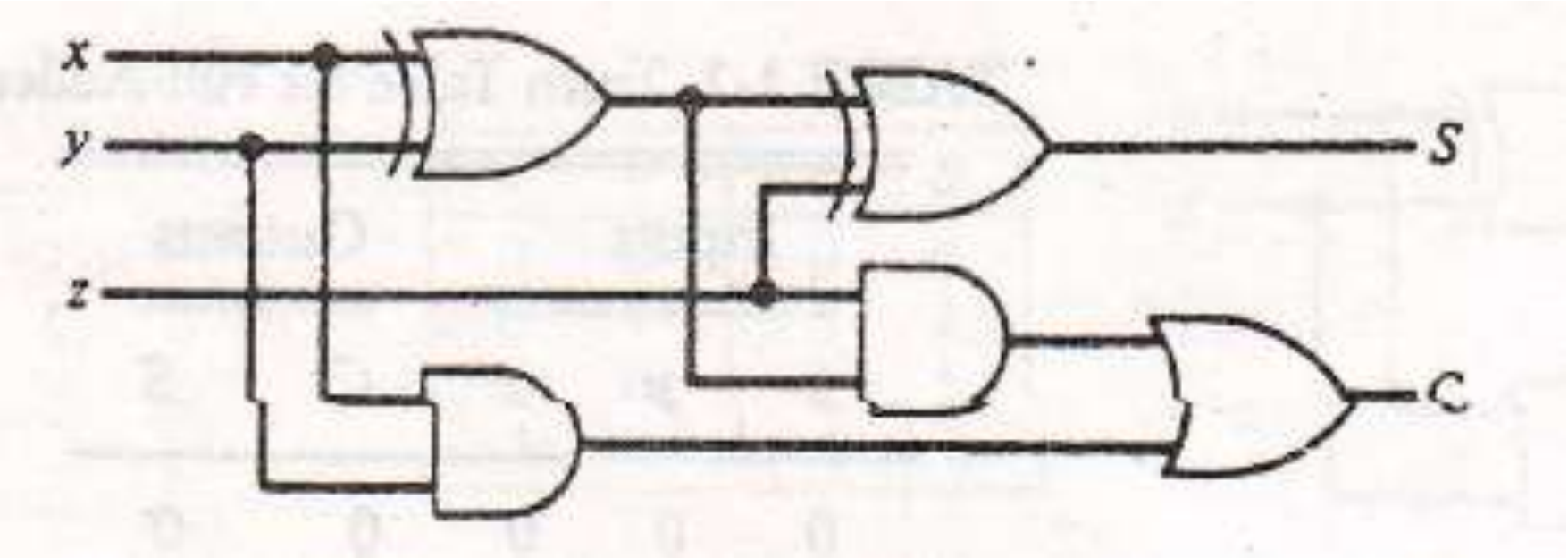
By Using AND & OR Gates



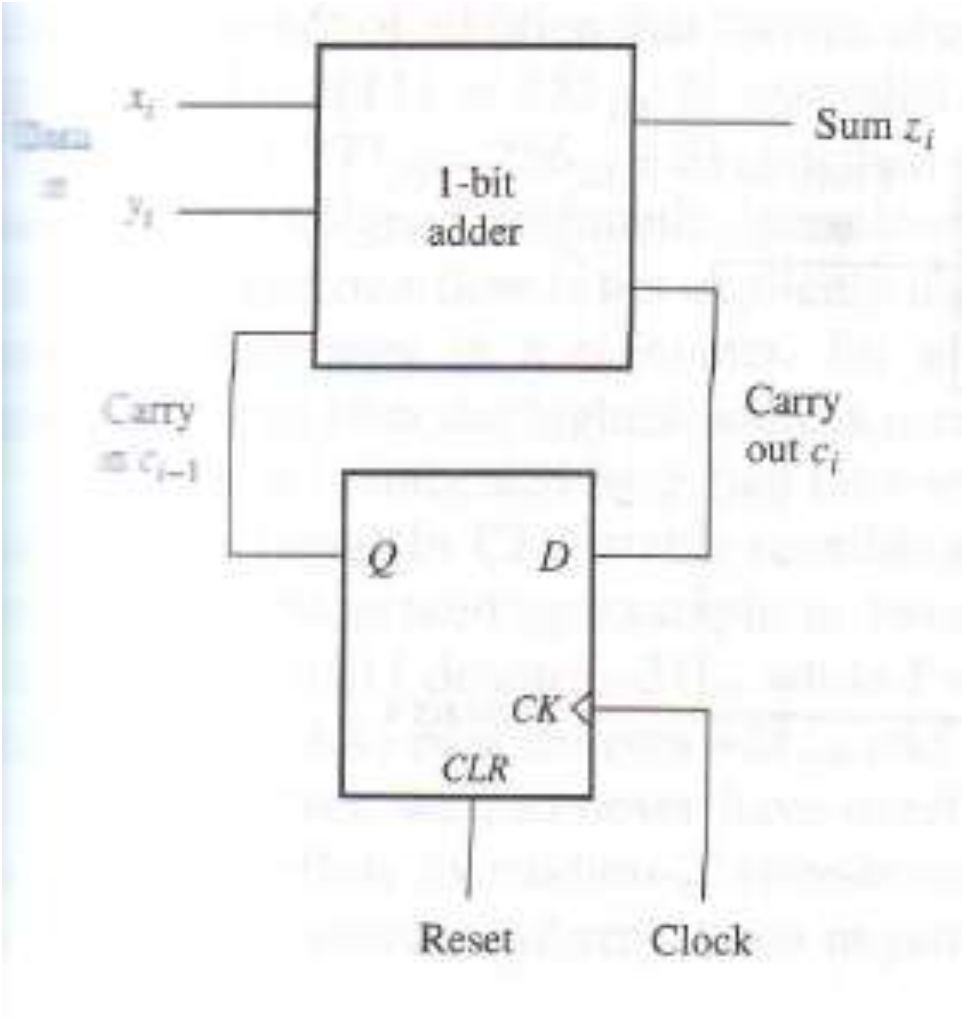
Output From Each Gate

- First AND= Ci-1, Yi,Xi
- Second AND = Xi',Ci'-1,Yi
- Third AND =Yi', Xi',Ci-1
- Fourth AND = Ci'-1, Xi, Yi'
- Fifth AND = Xi, Ci-1
- Sixth AND = Yi, Xi
- Seventh AND = Yi, Ci-1

By Using XOR, AND & OR Gate



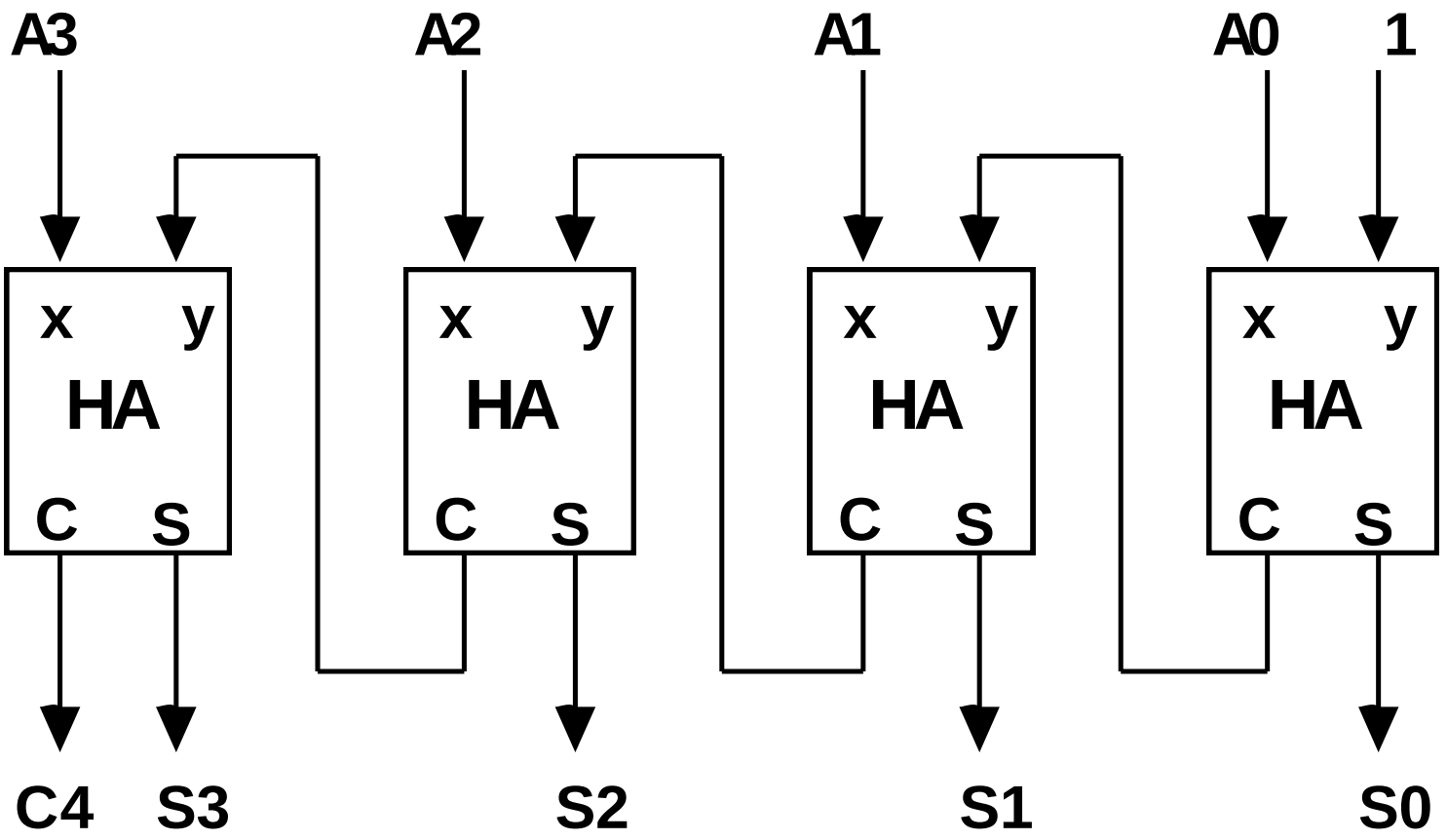
A Binary Adder Can Also Be Shown As:



Binary Incrementer

- The increment micro operation adds 1 to a number in a register.
- This micro operation is easily implemented with a binary counter & this can be accomplished by means of Half Adders.
- The output carry of one half adder is connected to one of the inputs of the next higher order half adder.

Binary Incrementer

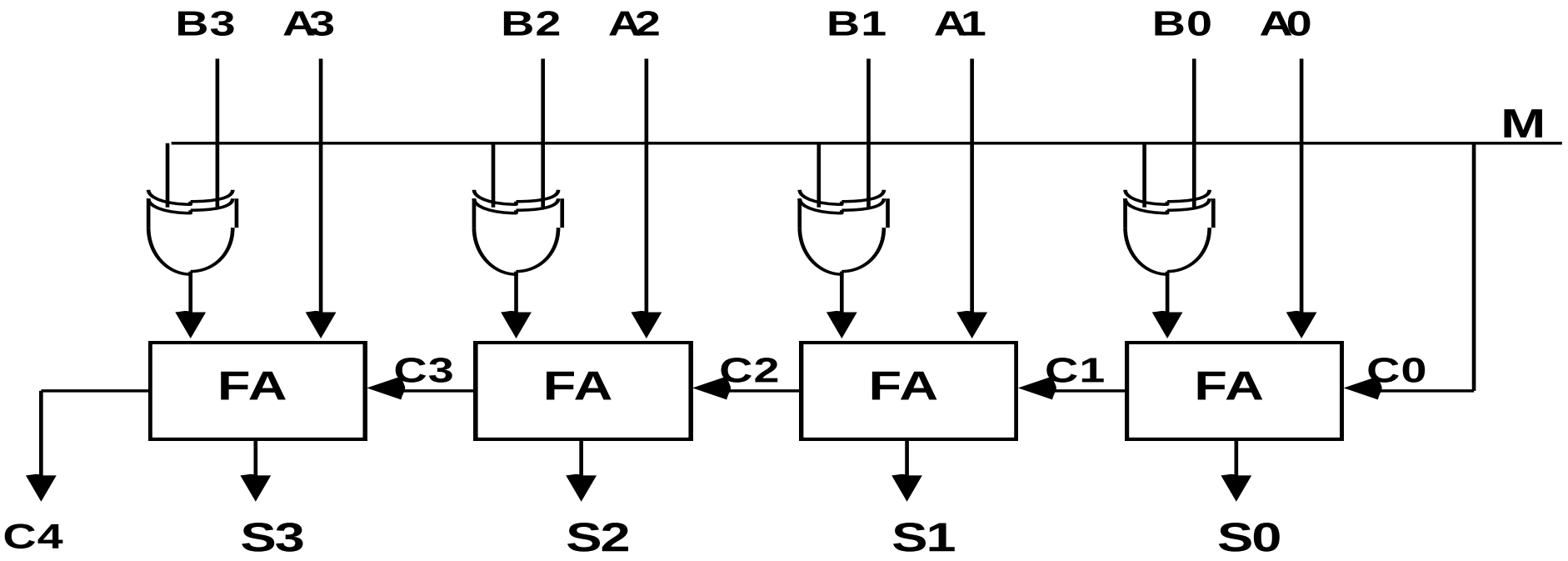


Binary Adder Subtractor

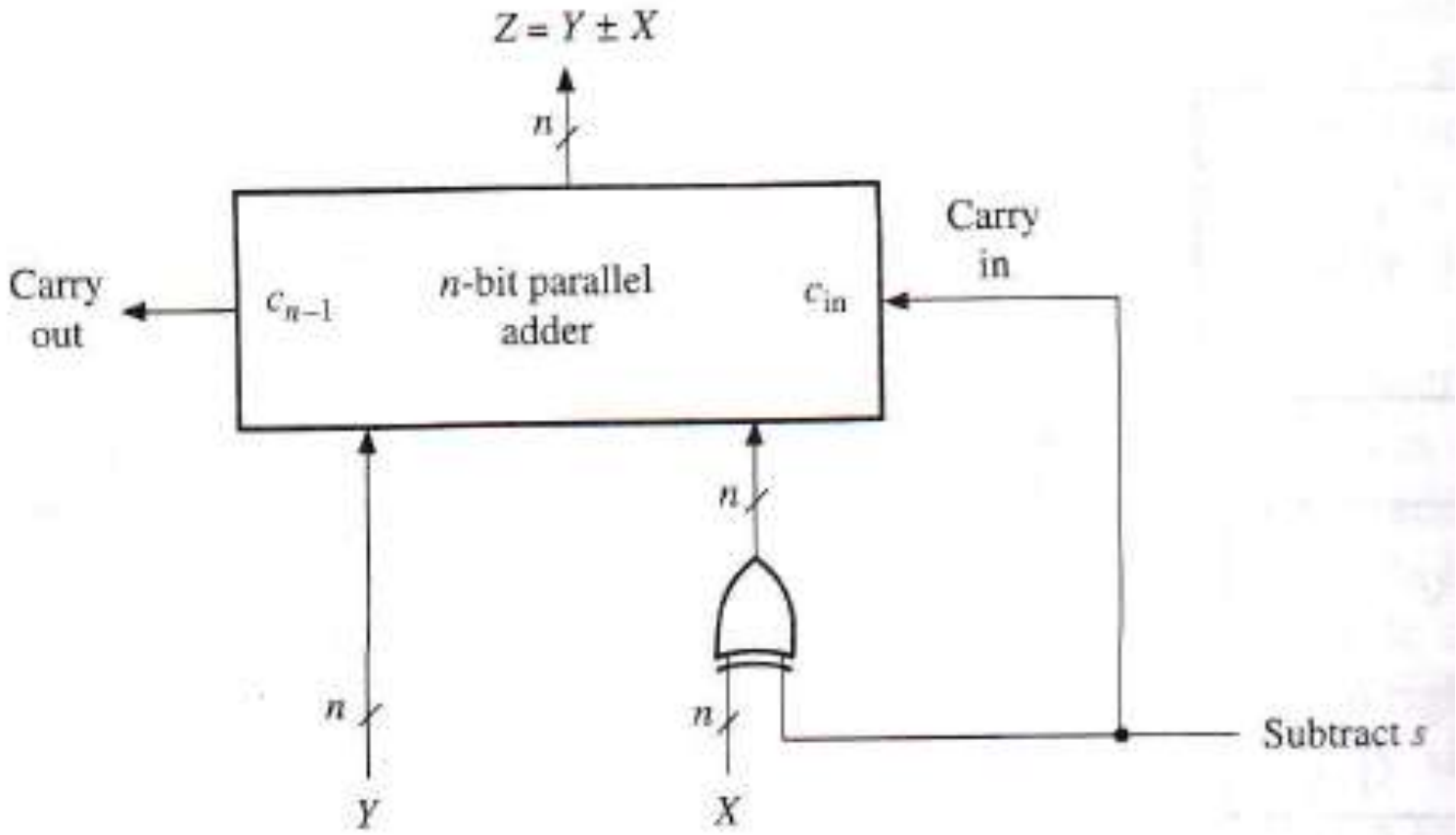
- The addition & subtraction operations can be combined into one common circuits by including an XOR gate with each full adder. This combinational circuit is known as Binary Adder Subtractor.

- Here, a mode bit is used to find addition and subtraction is done by the circuit. If the value of mode bit is 0 it means addition is performed and if it's value is 1 it means subtraction is performed.

Binary Adder Subtractor



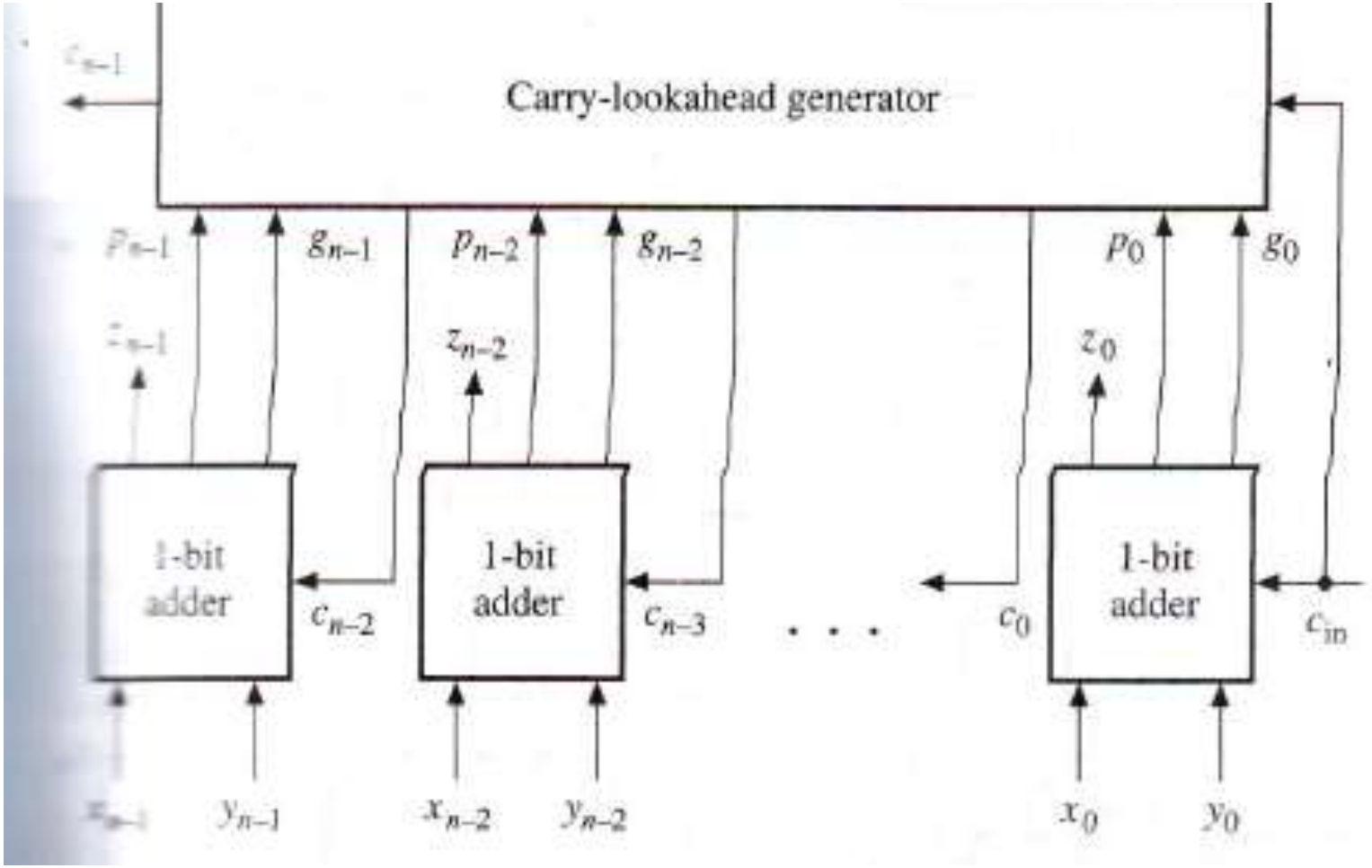
$Z = Y \pm X$ can be diagramatically represented as:



Overflow

- Overflow is indicated by a flag bit v in operations involving signed numbers, this flag is found in CPU status register.
- There V is indicated as:

High Speed adder



- gi=xiyi
- pi=xi+yi

ci=xiyi+xici−1+yici−1

ci=gi+pi(ci−1)

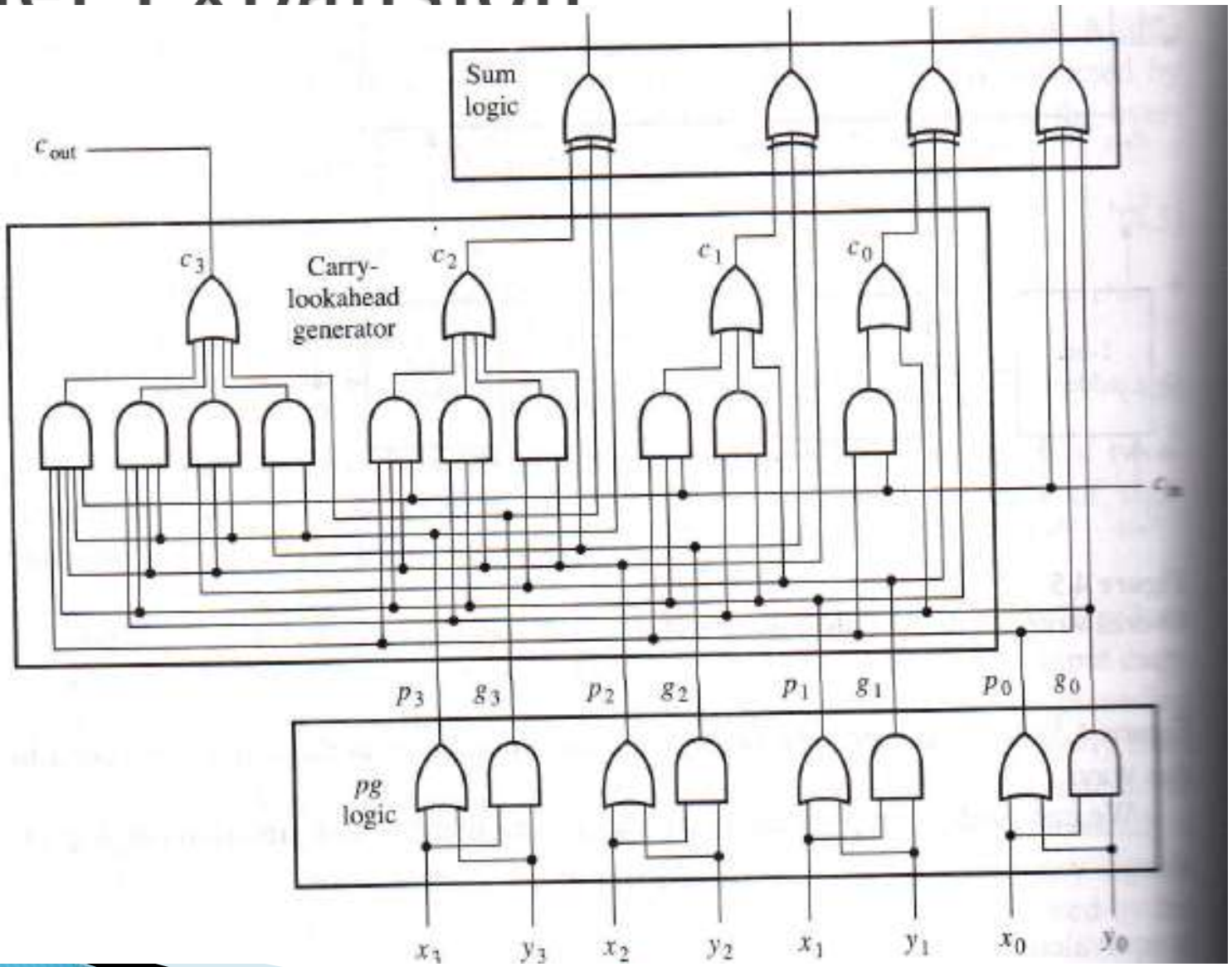
co = g0+p0cin

c1= g1+p1g0+p1p0cin

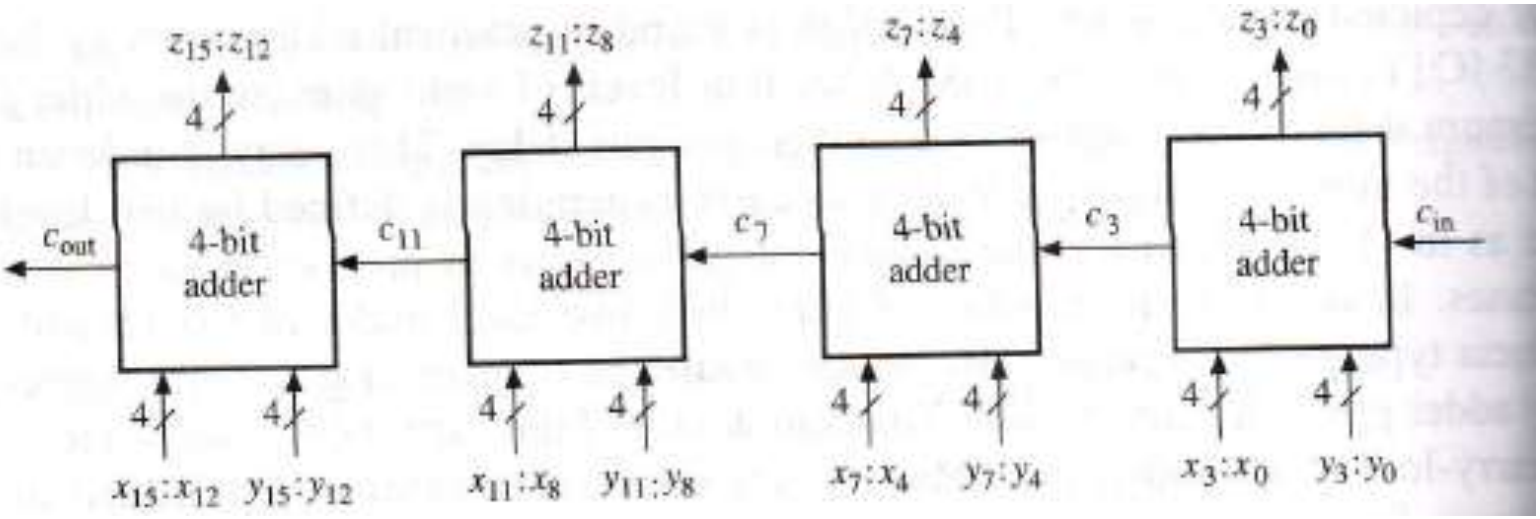
c2= g2+p2g1+p2p1go+p2p1p0cin

c3=g3+p3g2+p3p2g1+p3p2p1g0+p3p2p1p0cin

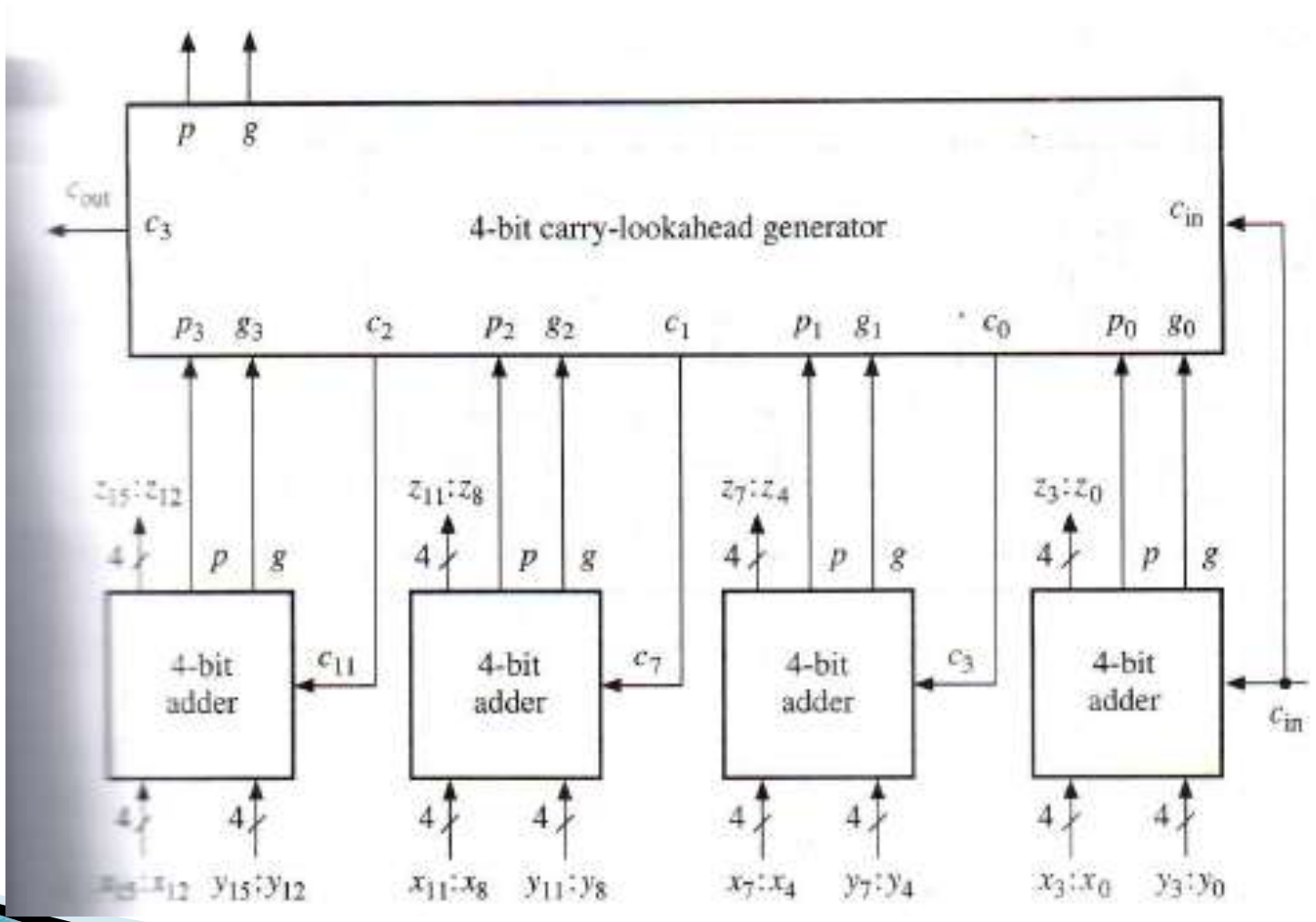
Adder Expansion



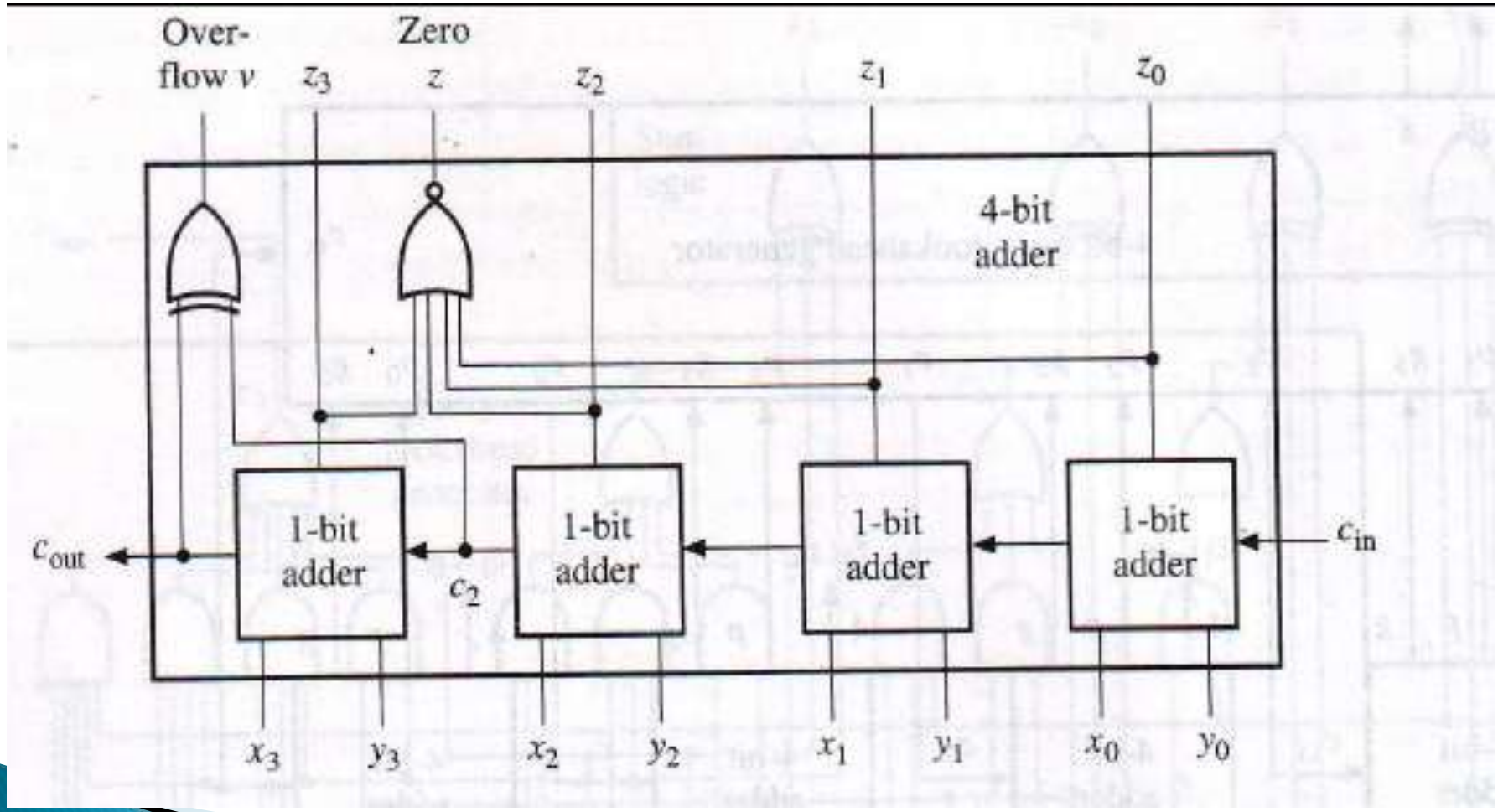
16 Bit Adder That consist of 4, 4 Bit Binary Adder



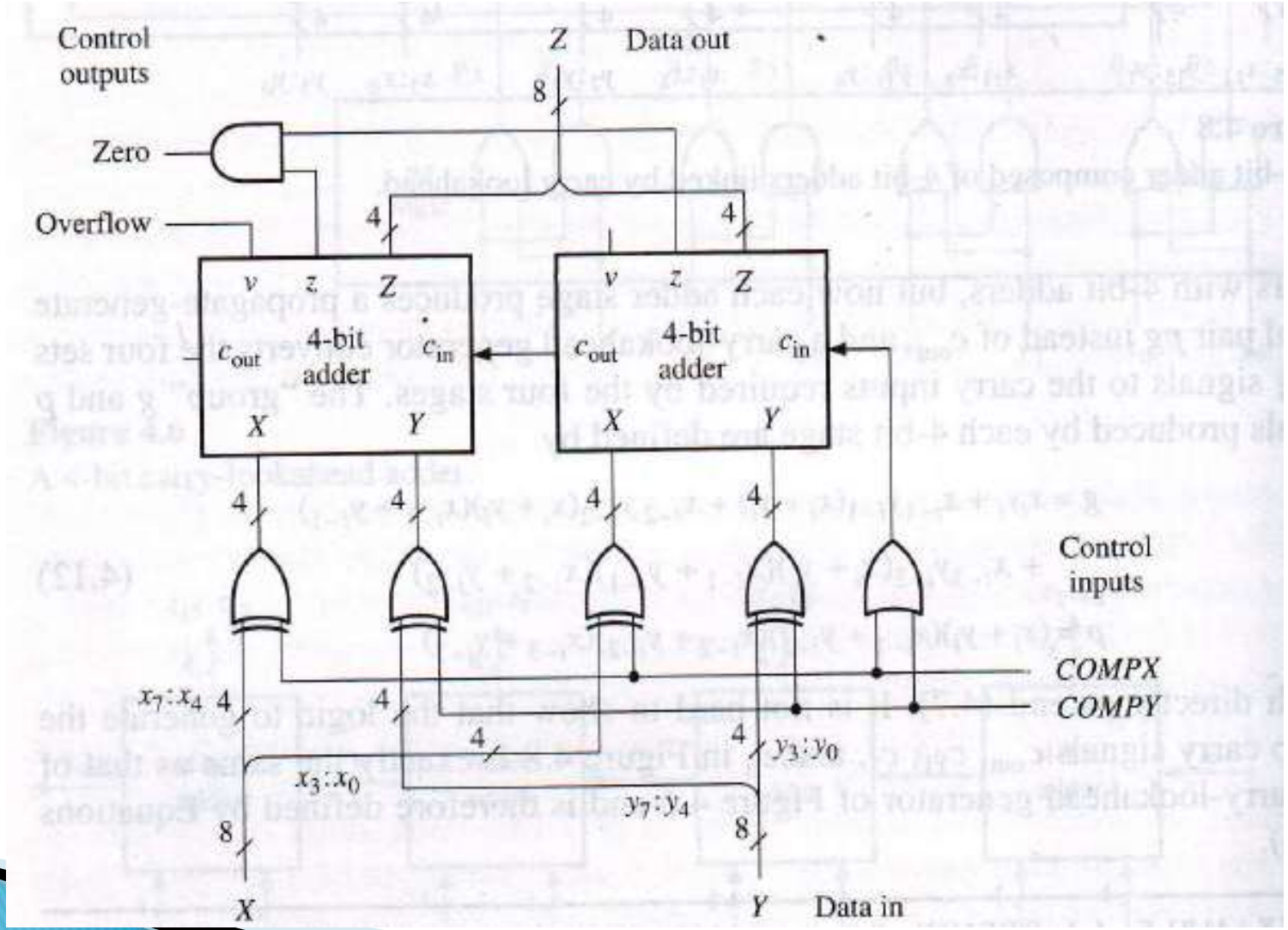
Above Diag With Carry-Look Ahead Generator



4 Bit Adder Module with Overflow



An 8 bit Adder Subtractor



Multiplication

Introduction

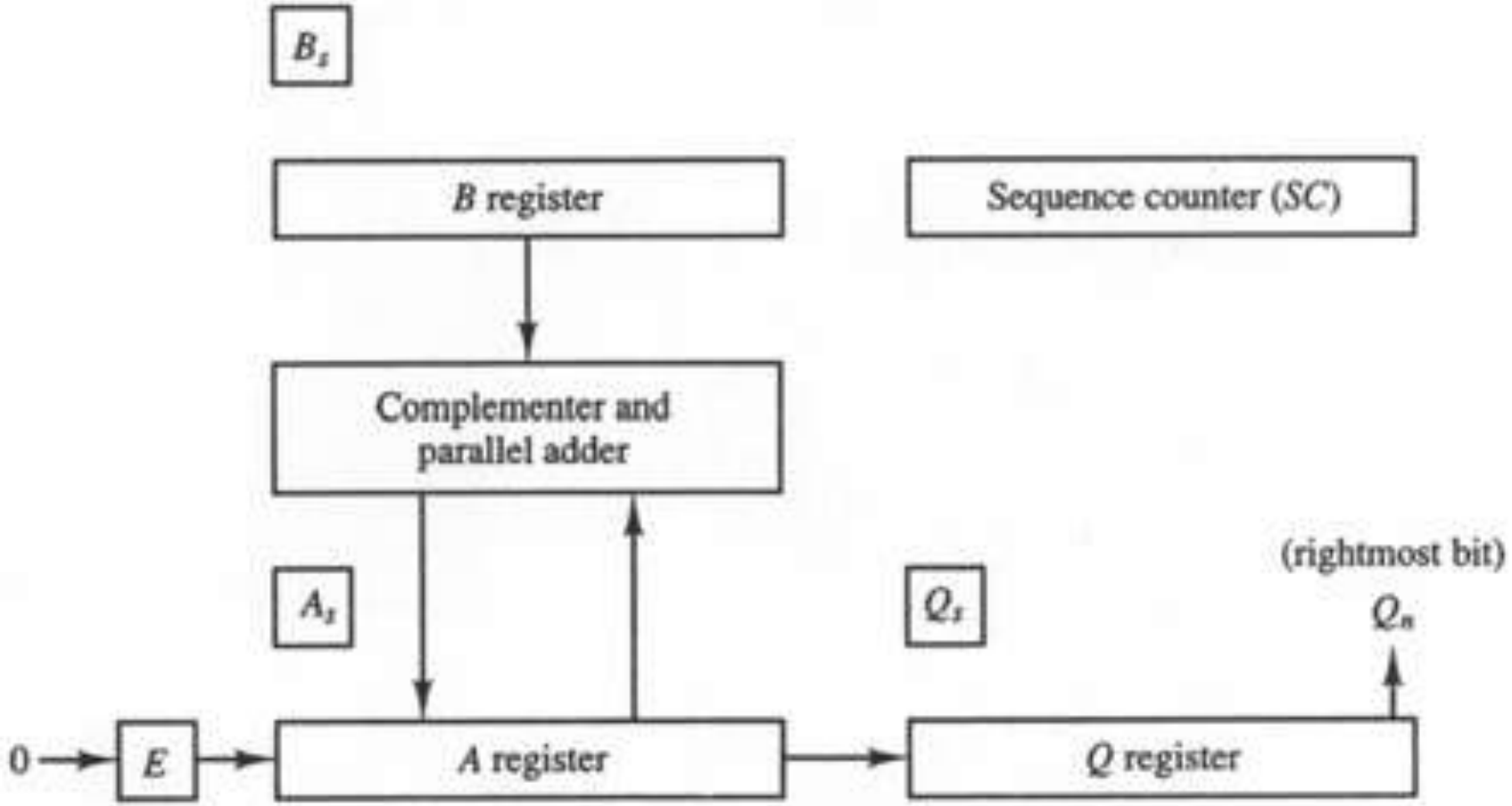
- Fixed point multiplication requires substantially more hardware than the fixed point addition.

- While first bit is multiplied then we have to shift 0 bit to the left, when second bit is multiplied then there is a shift of 1 bit. Similarly for third, fourth bit and so on.

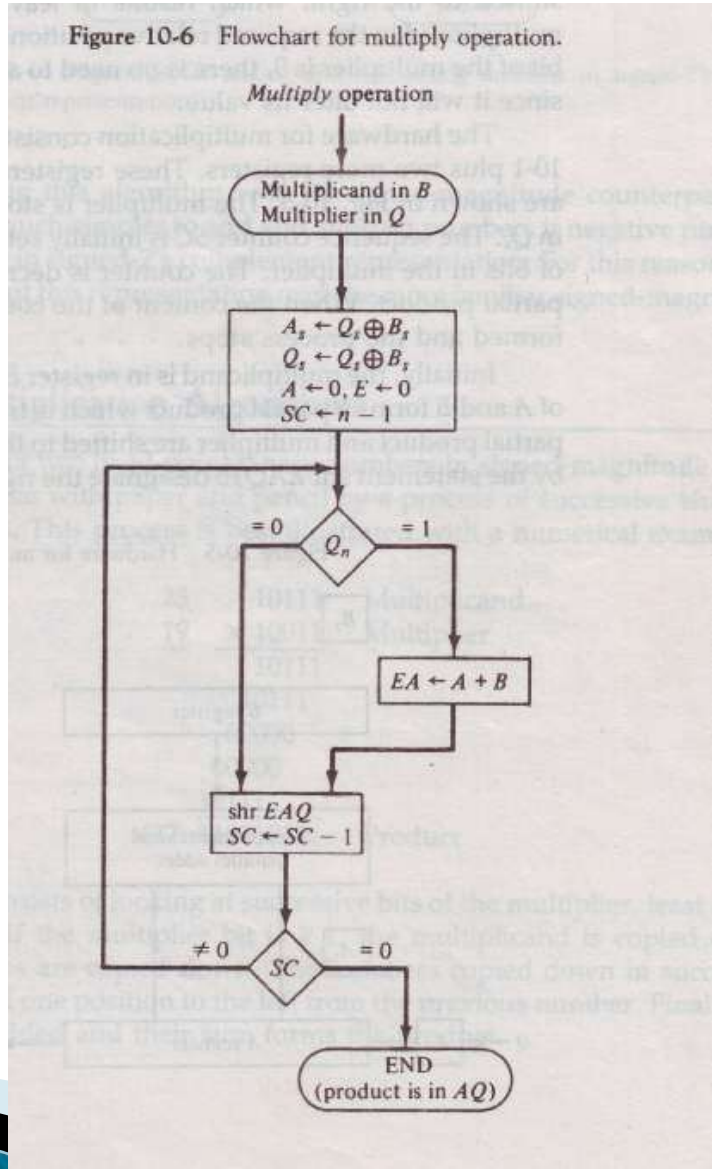
- The value of Partial Product is $P_{i+1} = P_i + x_j 2^i Y$

- So final product is $P = \sum_{j=0}^{3} x_j 2^i Y$

Hardware For Multiply Operation



Flow Chart For Multiply Operation



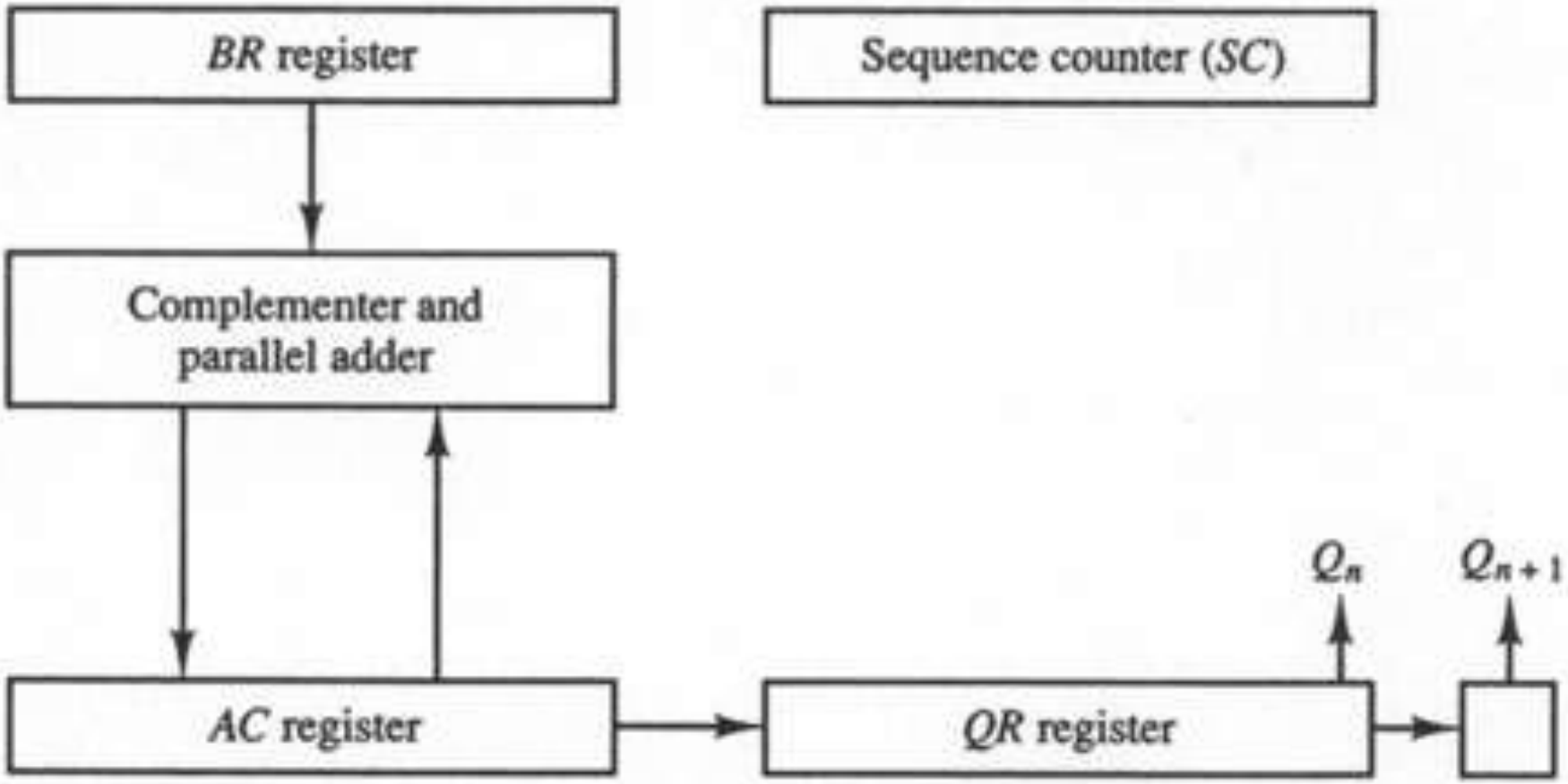
For Eg: 23 X 19 = 437

Multiplicand $B = 10111$	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
$Q_n = 1$; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
$Q_n = 1$; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
$Q_n = 0$; shift right EAQ	0	01000	10110	010
$Q_n = 0$; shift right EAQ	0	00100	01011	001
$Q_n = 1$; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in $AQ = 0110110101$				

Multiplication of Twos Complement Multiplier Can be Done by:

- James E Robertson Algorithm
- Booth's Algorithm

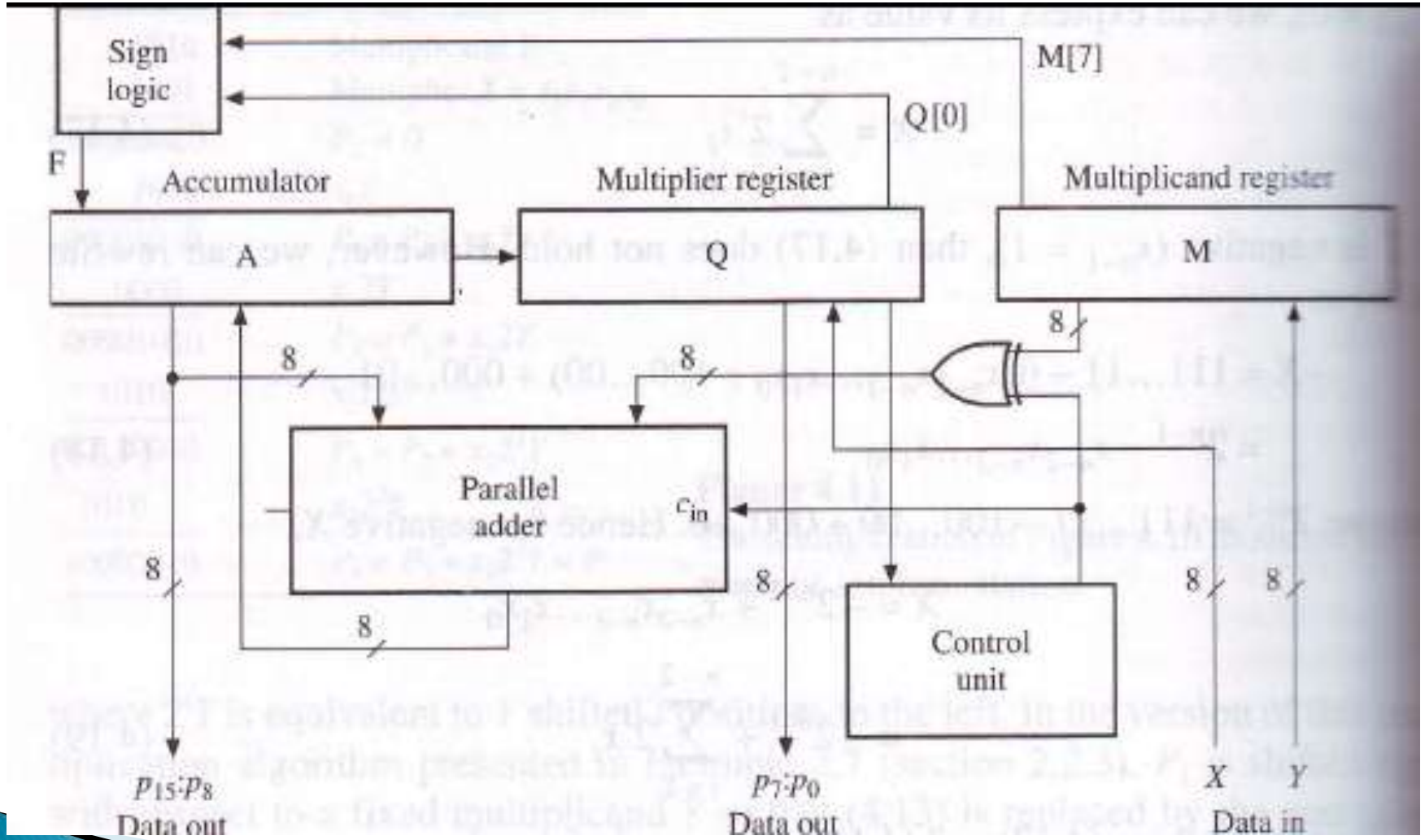
Booth's Algorithm



−9 X −13 = 117

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

Diagram of Multiplication

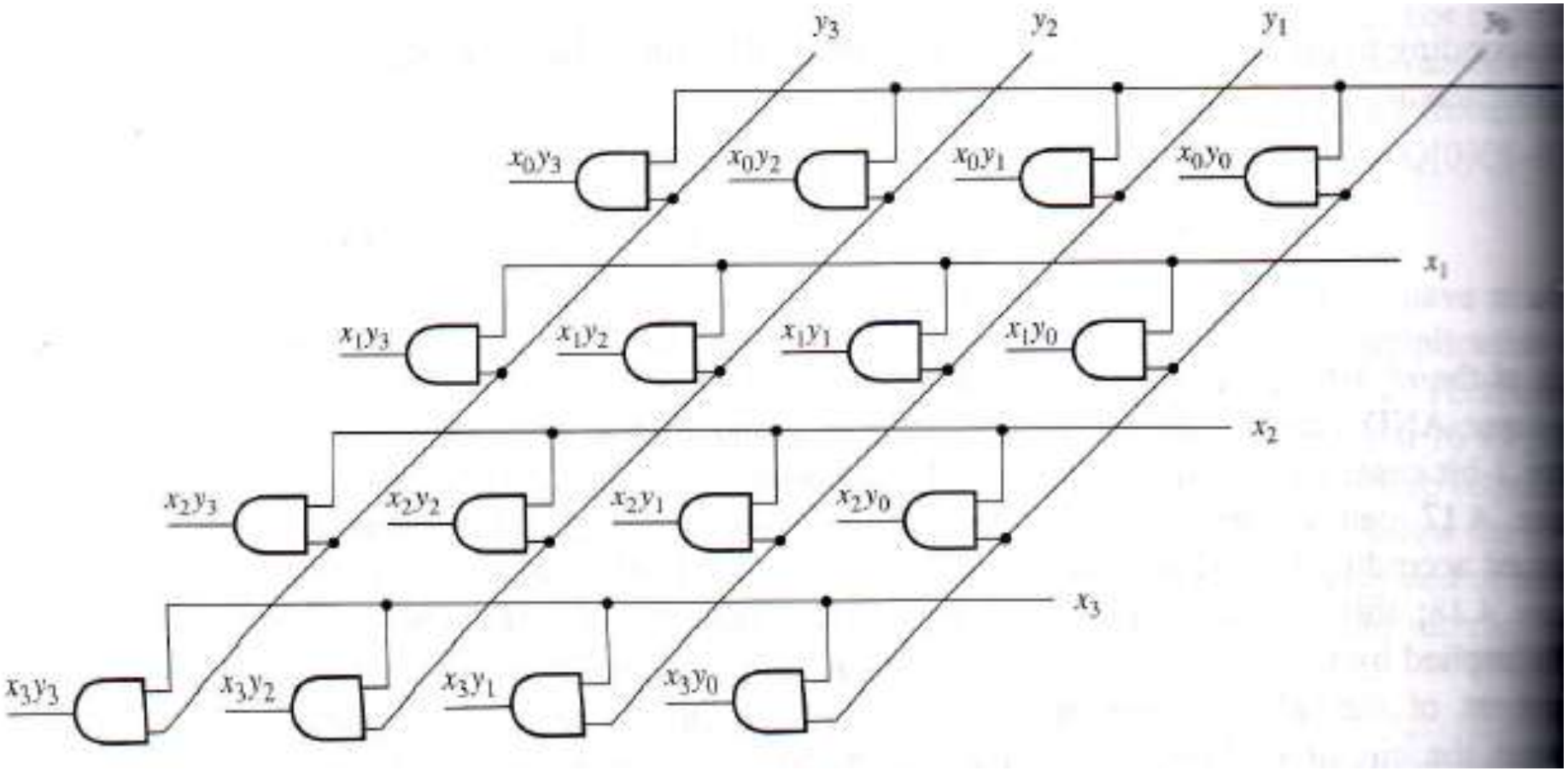


Array Multiplier

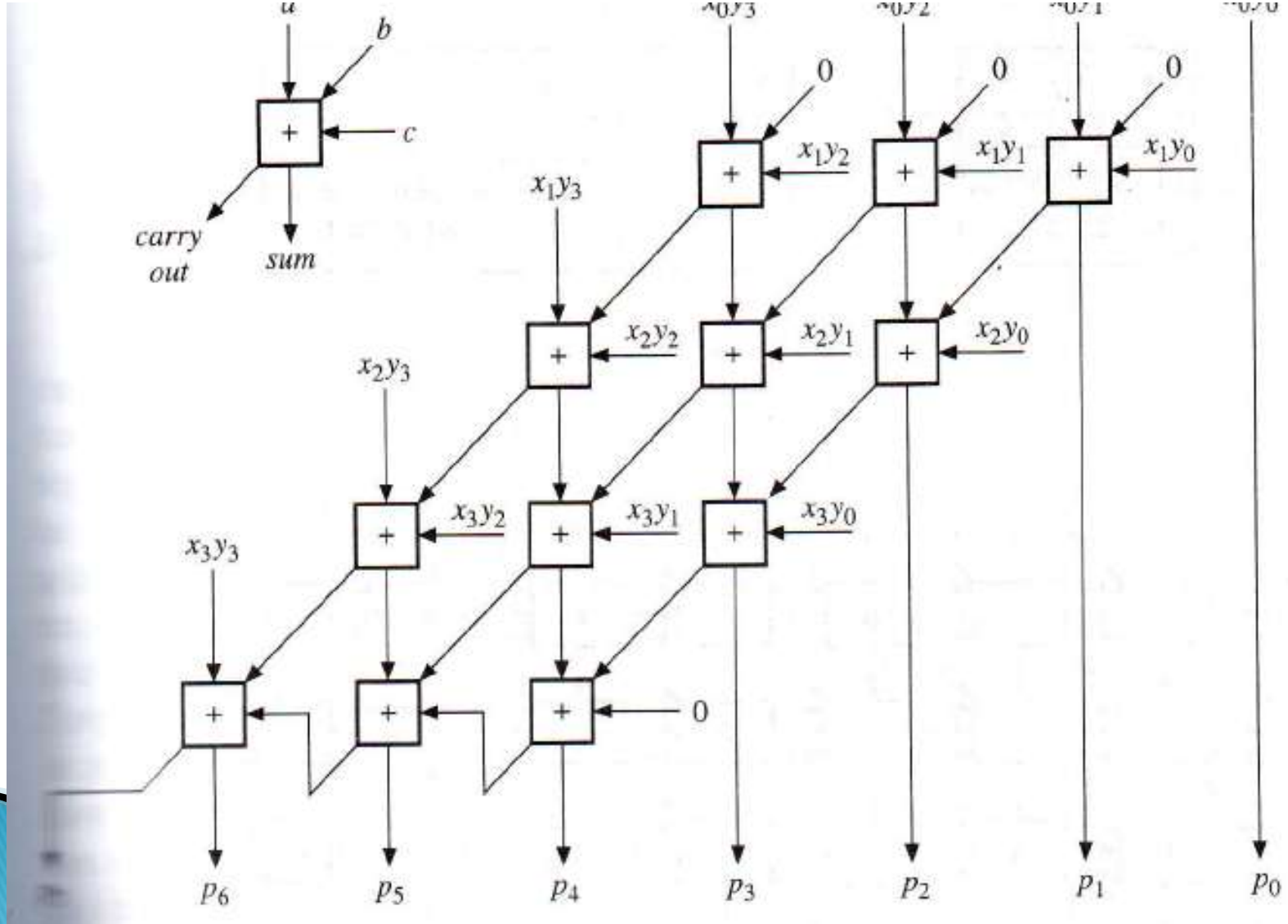
- Checking the bits of the multiplier one at a time and forming partial products is a sequential operation that requires a sequence of add and shift micro operations.

- The multiplication of two binary numbers can be done with one micro operation by means of a combinational circuit that forms the product bits all at once . This is done by making use of an array multiplier.

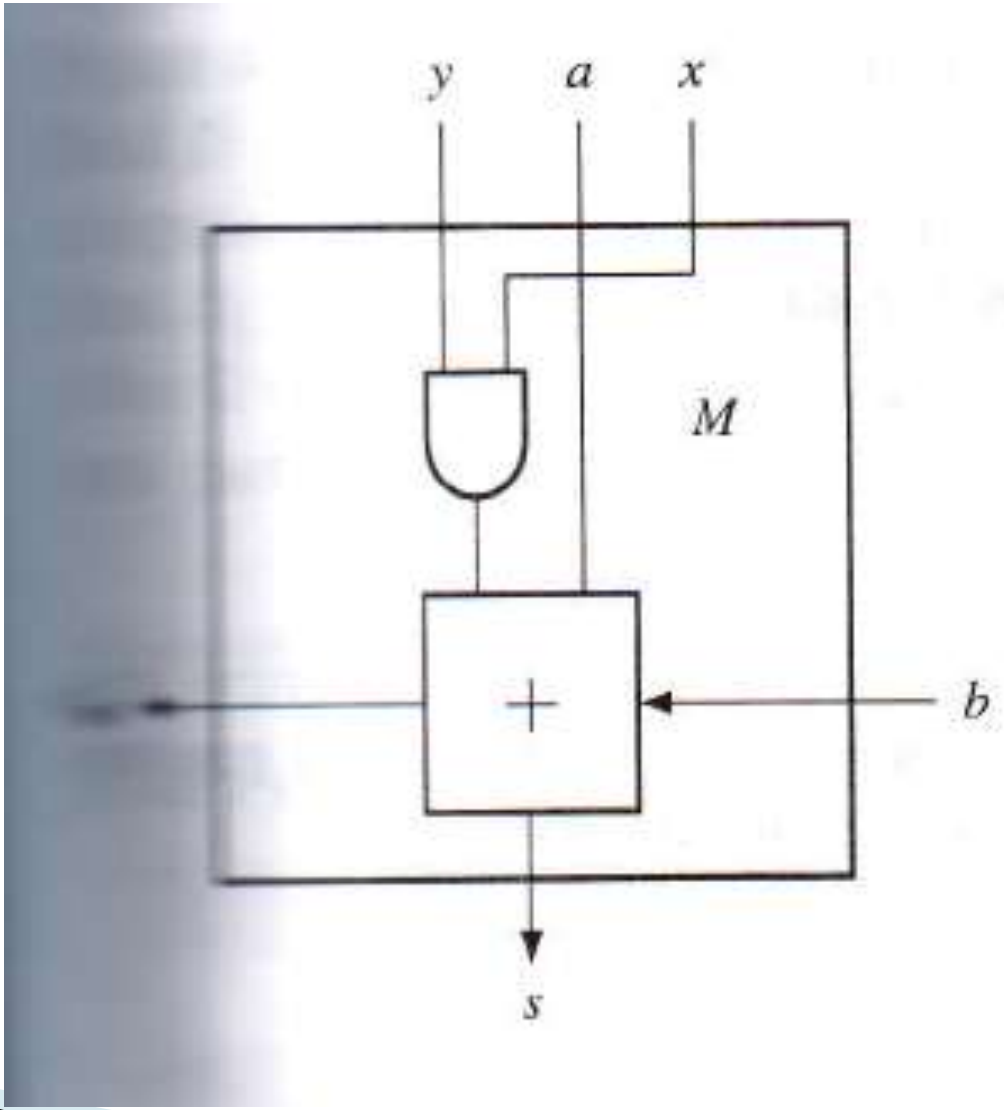
AND Array for 4×4 bit Multiplication:



Full Adder Array For 4×4 Bit



One Cell Consist of



Division

Introduction

- Division is done by 2 methods
  - Restoring Method
  - Non-Restoring Method

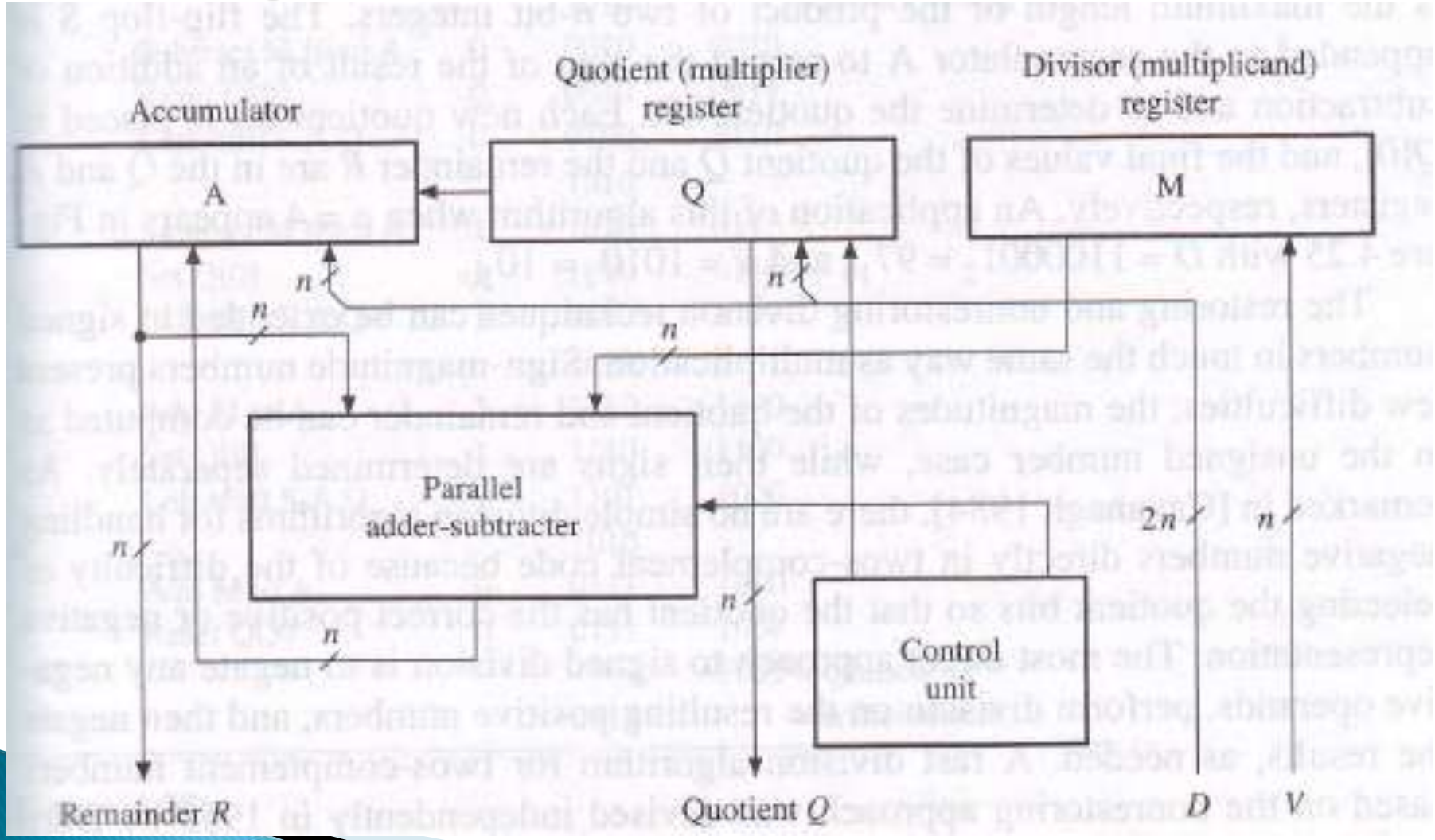
Restoring Method (Algorithm)

- Step-1 Initialize the register with Dividend
  - ASHL
  - ADD B'+1
- Step-2 Check If E=1 or E=0
- E=1
  - Reset Qn SC=SC-1 goto step-4
  - ASHL EAQ
  - Add B'+1
- E=0
  - Leave Q_n =0 & Add B
  - Restore the Remainder SC=SC-1 goto step-4
  - ASHL
  - ADD B'+1
- Step- 4 For Termination When SC=0 Neglect E, Remainder in A , Quotient in Q

Problems

- Dividend=01110 Divisor= 10001
  - Quotient=11010 Remainder=00110
- Dividend=30 Divisor=3 size=4
  - Quotient=1010 Remainder=0000
- Dividend =15 Divisor=2 size=4
  - Quotient=0111 Remainder=0001
- Dividend=38 Divisor=5 size=3 bit
  - Quotient=111 Remainder=011

Datapath of a Sequential n-bit binary Divider:



Non-Restoring Method

- Step-1 Initialize the register with Dividend
  - ASHL
  - ADD M'+1
- If S=0
  - Reset Q[0]
  - If Count=n-1 then goto step-4
  - Else
  - Count=count+1
  - SHL SAQ
  - ADD M'+1 then goto either step 2 or step-3
- If S=1
  - Set Q[0]
  - If Count=n-1 then goto step-4

Non-Restoring Method Contd..

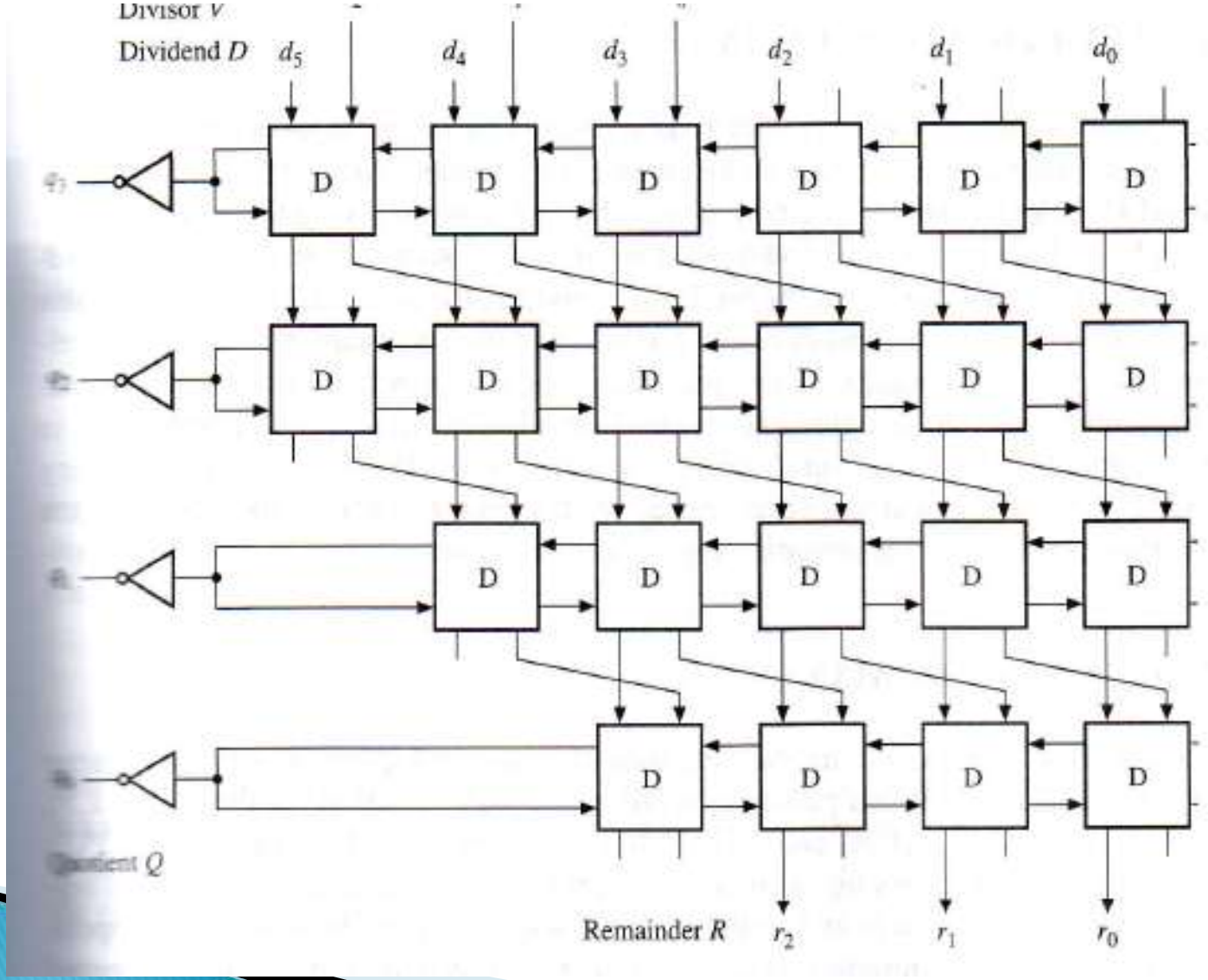
- Else
- Count=count+1
- SHL SAQ
- ADD M then goto either step 2 or step-3

- Step-4 if s=1 then Add M
  - Quotient in Q
  - Remainder in R

Problems Based On Non-Restoring Method:

- Dividend=1100001 Divisor=1010 Size=4 bits
- Dividend=30 Divisor=3 Size=4
- Dividend = 40 Divisor=2 Size=5
- Dividend = 25 Divisor=4 Size=4
- Dividend =15 Divisor=2 Size=4

Diag of Non-Restoring Method

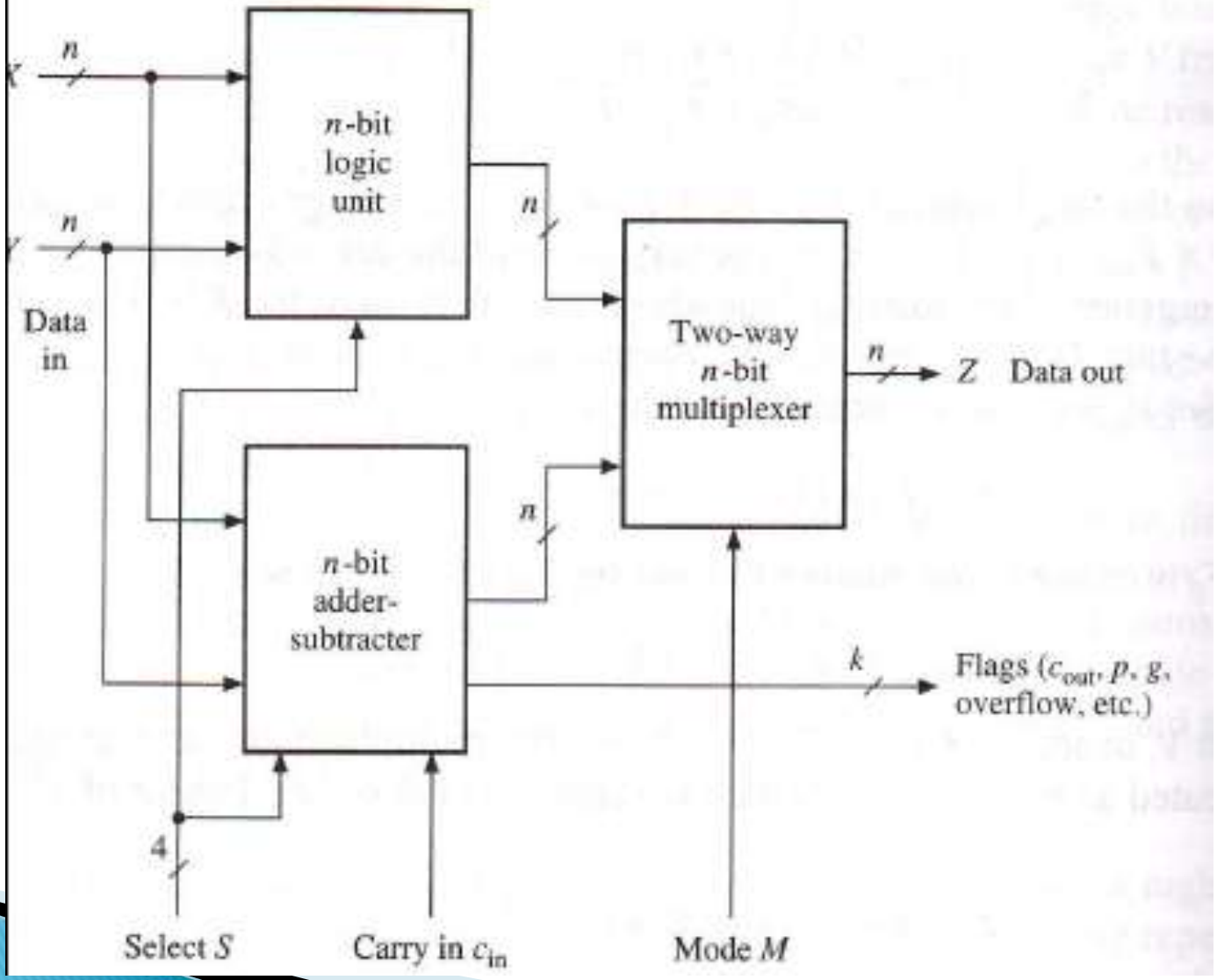


ALU

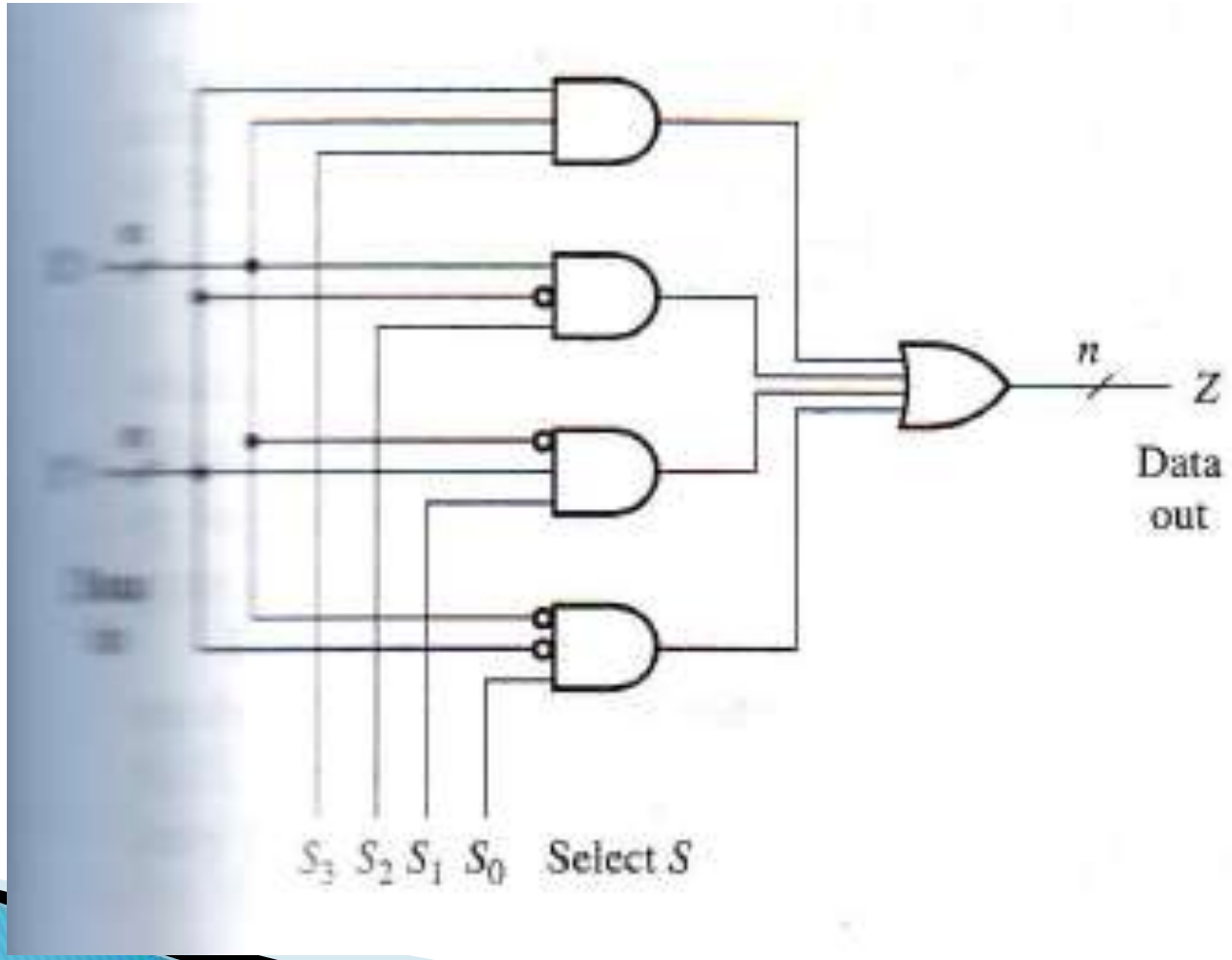
- The various circuit that are used to execute data processing instructions are usually combined in a single circuit called ALU.

- There are 2 types of ALU are:
  - Combinational ALU
  - Sequential ALU

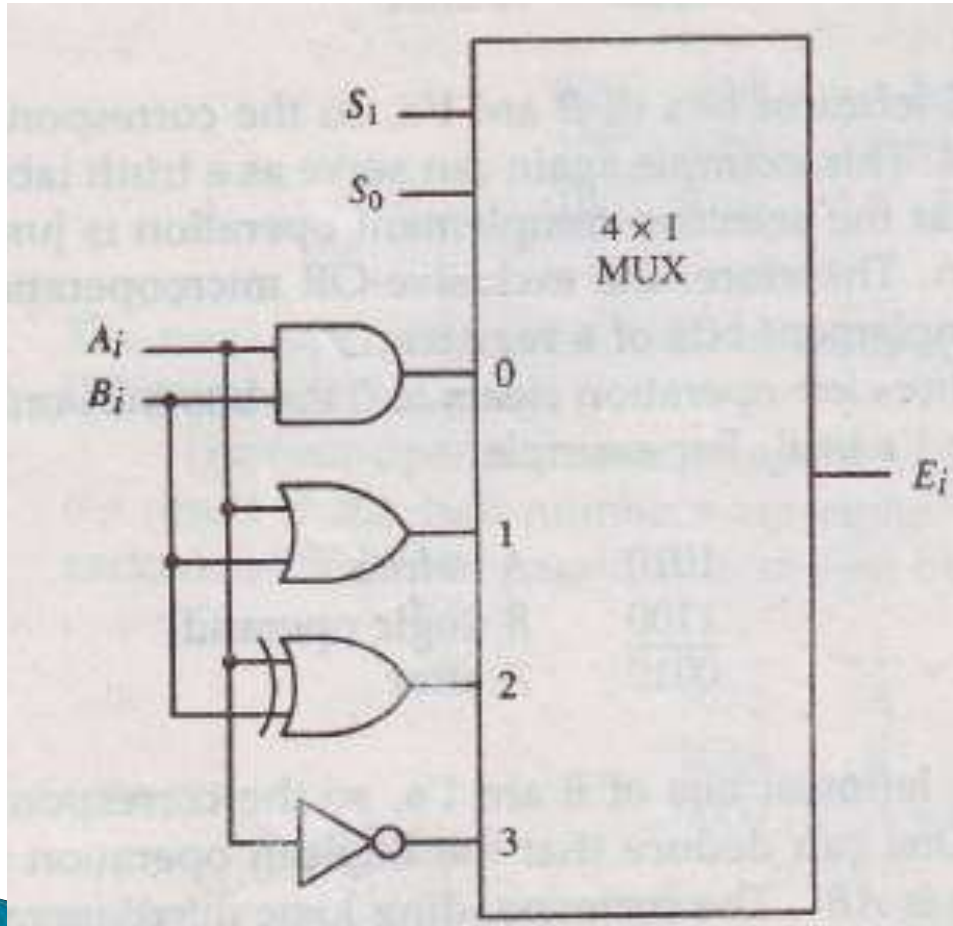
Combinational ALU (A Basic n bit ALU)



A n-bit Logic Unit



One Stage of The Logic Circuit



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

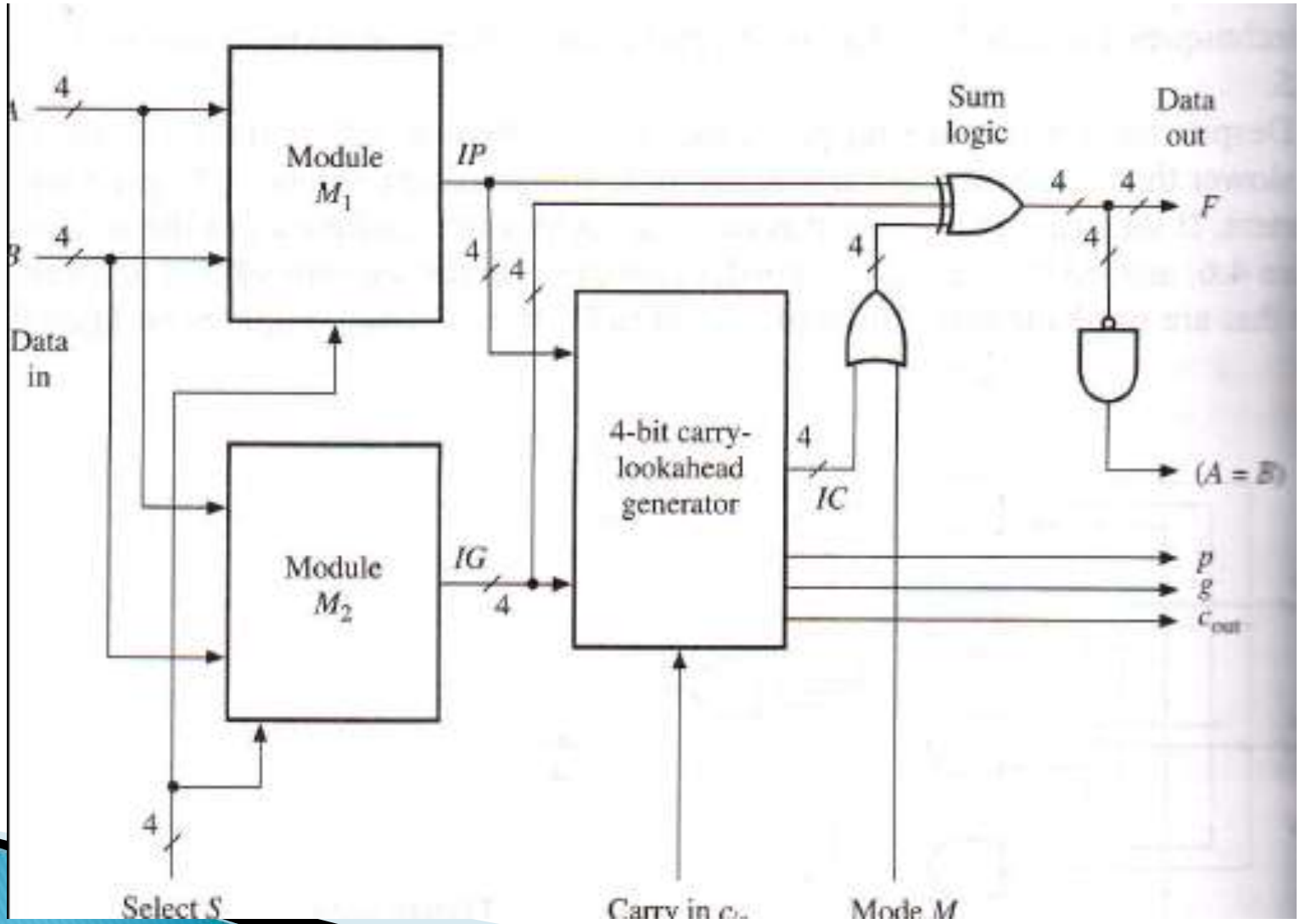
(b) Function table

A Register Level View of The 74181 4-bit ALU

- Here the output of the whole circuit comes form:

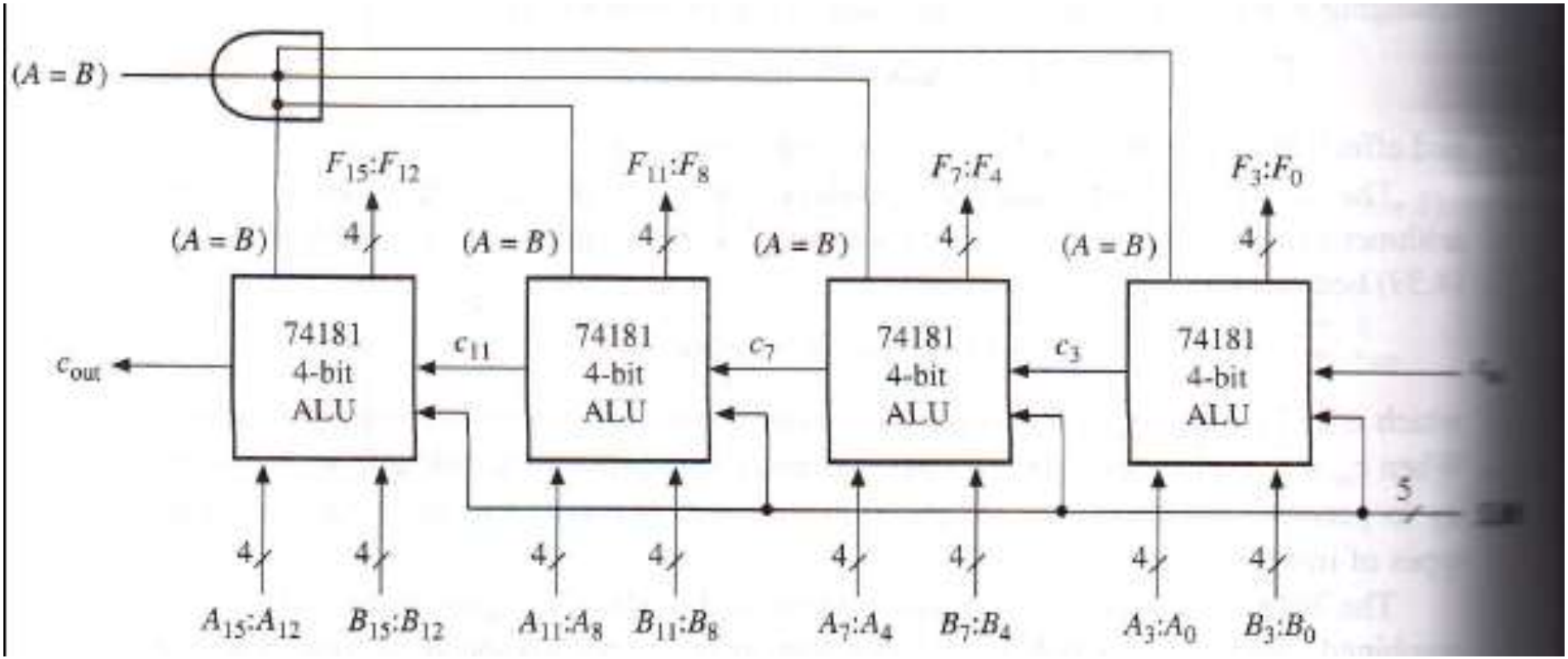
$$F_i = IP_i \oplus IG_i \oplus (IC_{i-1} + M)$$

A Register Level View of The 74181 4-bit ALU Contd…



- In the above circuit
  - When M=0 then arithmetic operation performs.
  - When M=1 then logic operation is performed.

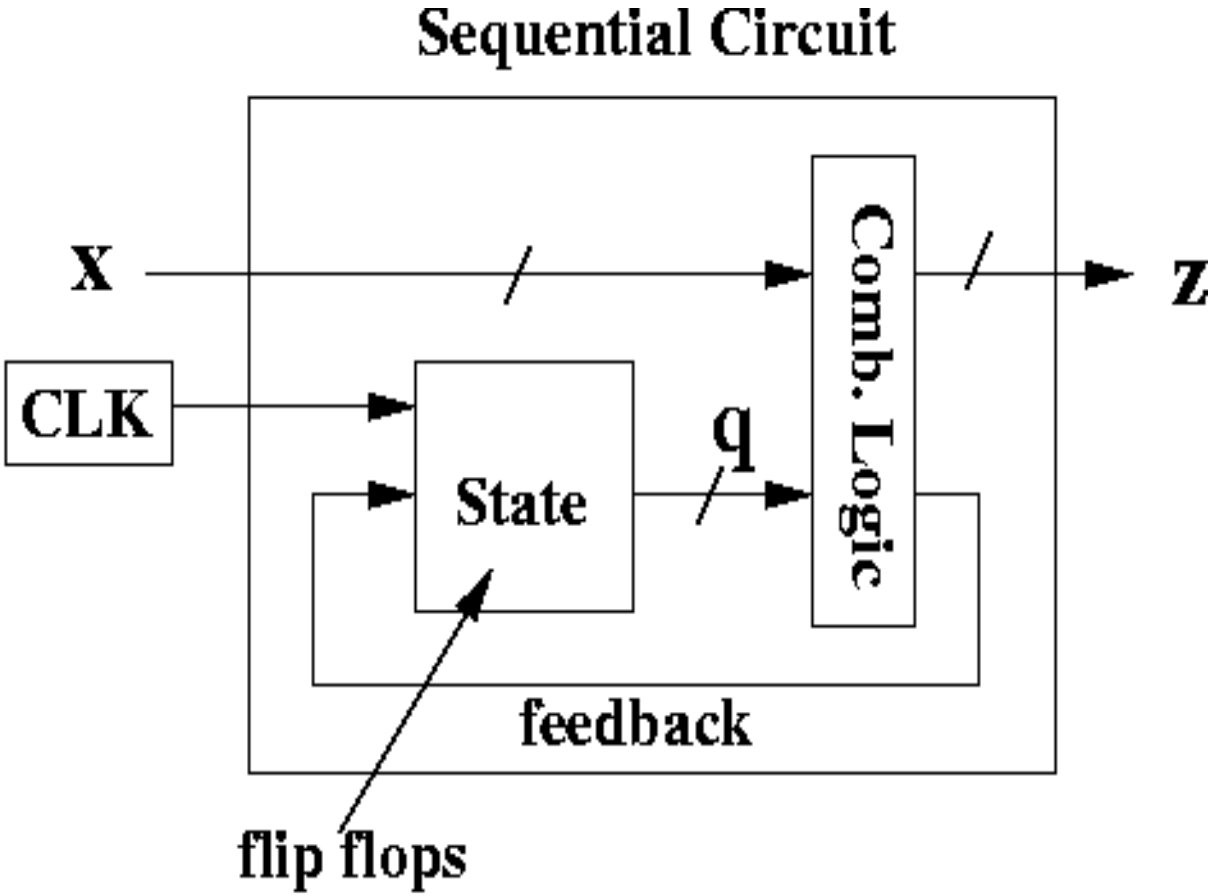
16 Bit Combinational ALU



Sequential ALU

- A sequential ALU uses flip-flops. Sequential circuits compute their output based on the input and the state and the updation of state is based on clock.

- Whereas combinational logic circuits implement Boolean functions it means their functions is only based on their input and they are not based on clocks.


Sequential Circuit
x
z
CLK
State
q
Comb. Logic
feedback
flip flops

Some Points Related To Sequential Circuits Are:

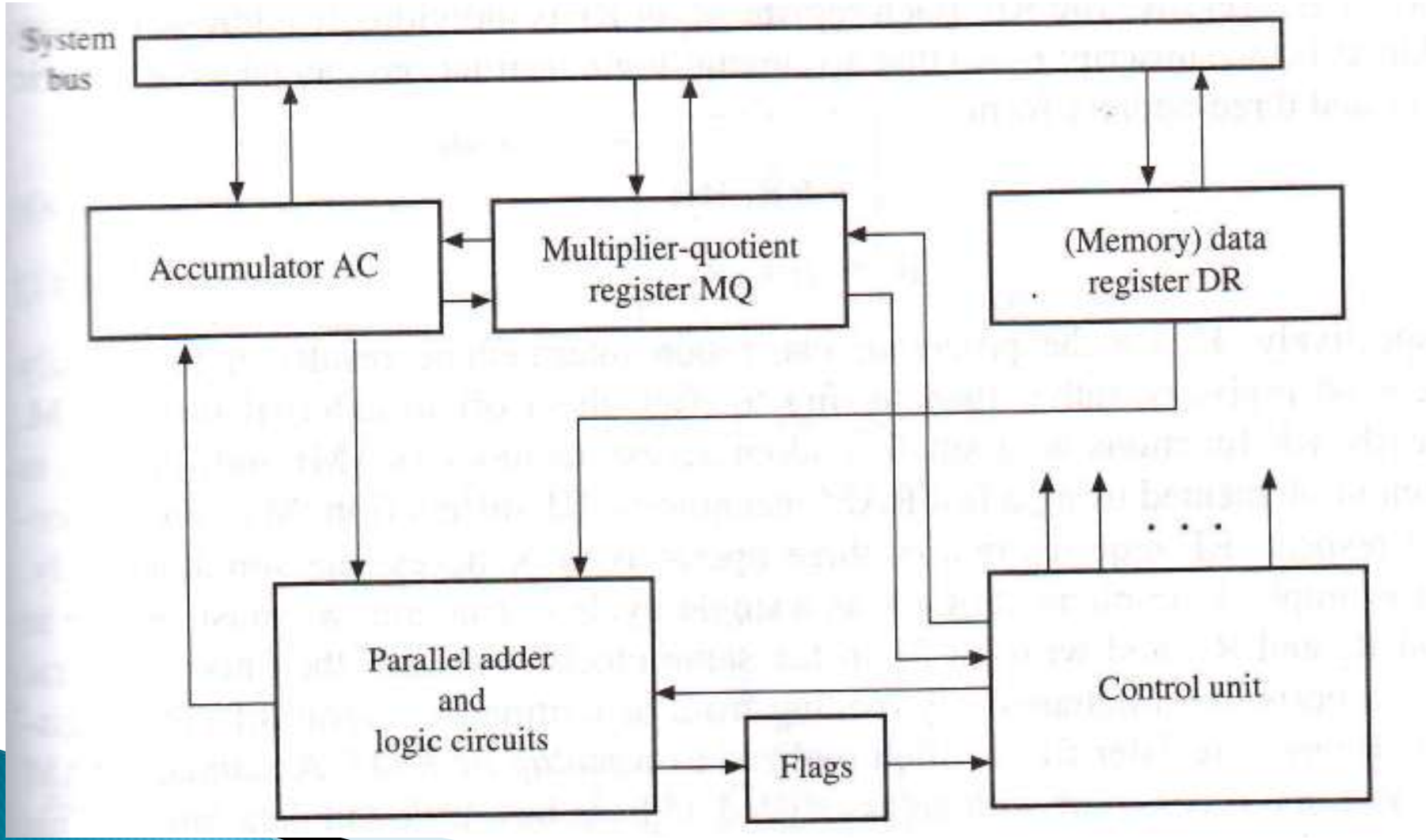
- A sequential circuit has inputs and outputs.
- A sequential logic circuit uses a clock.
- There is a box inside the circuit called state and this box contains flip-flops and this flip-flop basically store a k bit number for representing the current state.
- Here, the output is computed based on the input and state.

Some Points Related To Sequential Circuits Are Contd….:

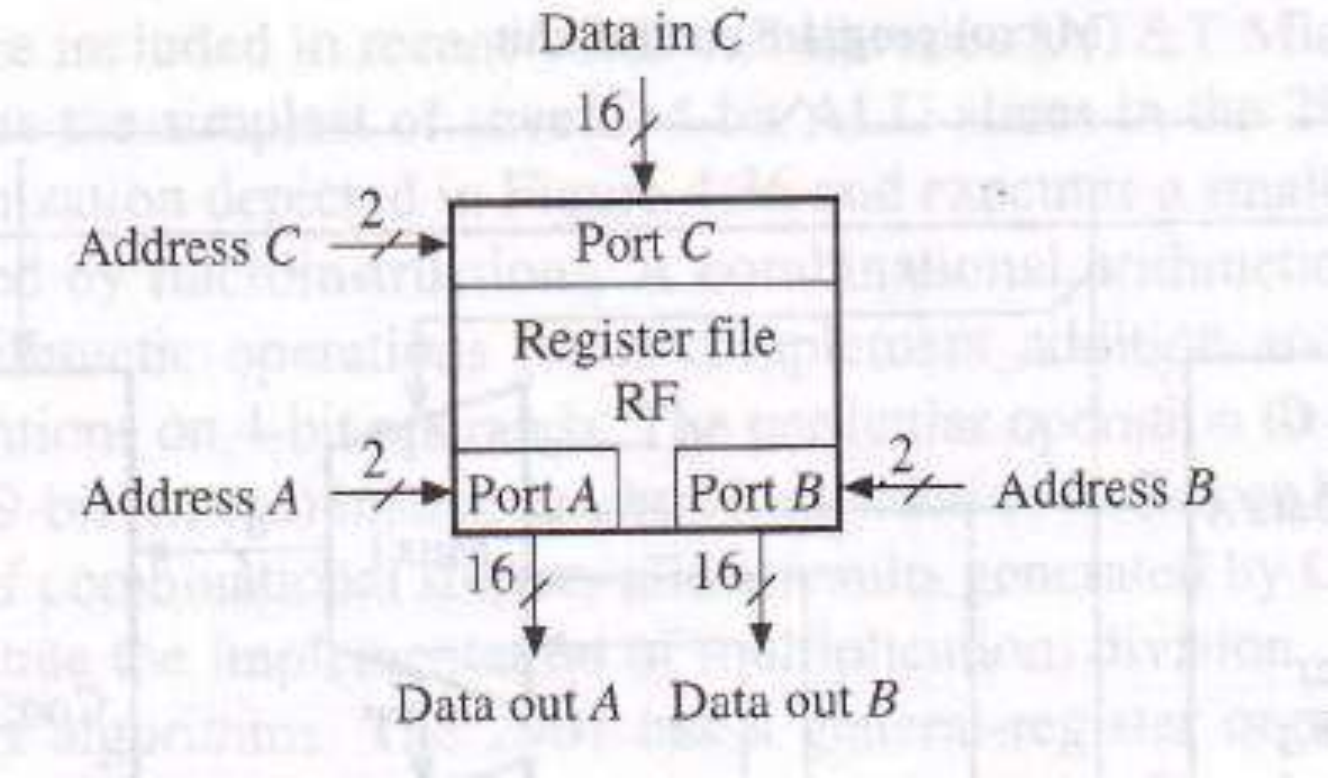
- The state may be updated at each positive clock edge. When there is not a positive clock edge, the state remains unchanged.

- The information needed to update to the state comes from the current state and the input which is fed through combinational logic and fed back into the state box, telling the state box how to update itself.

Structure of a Basic Sequential ALU:

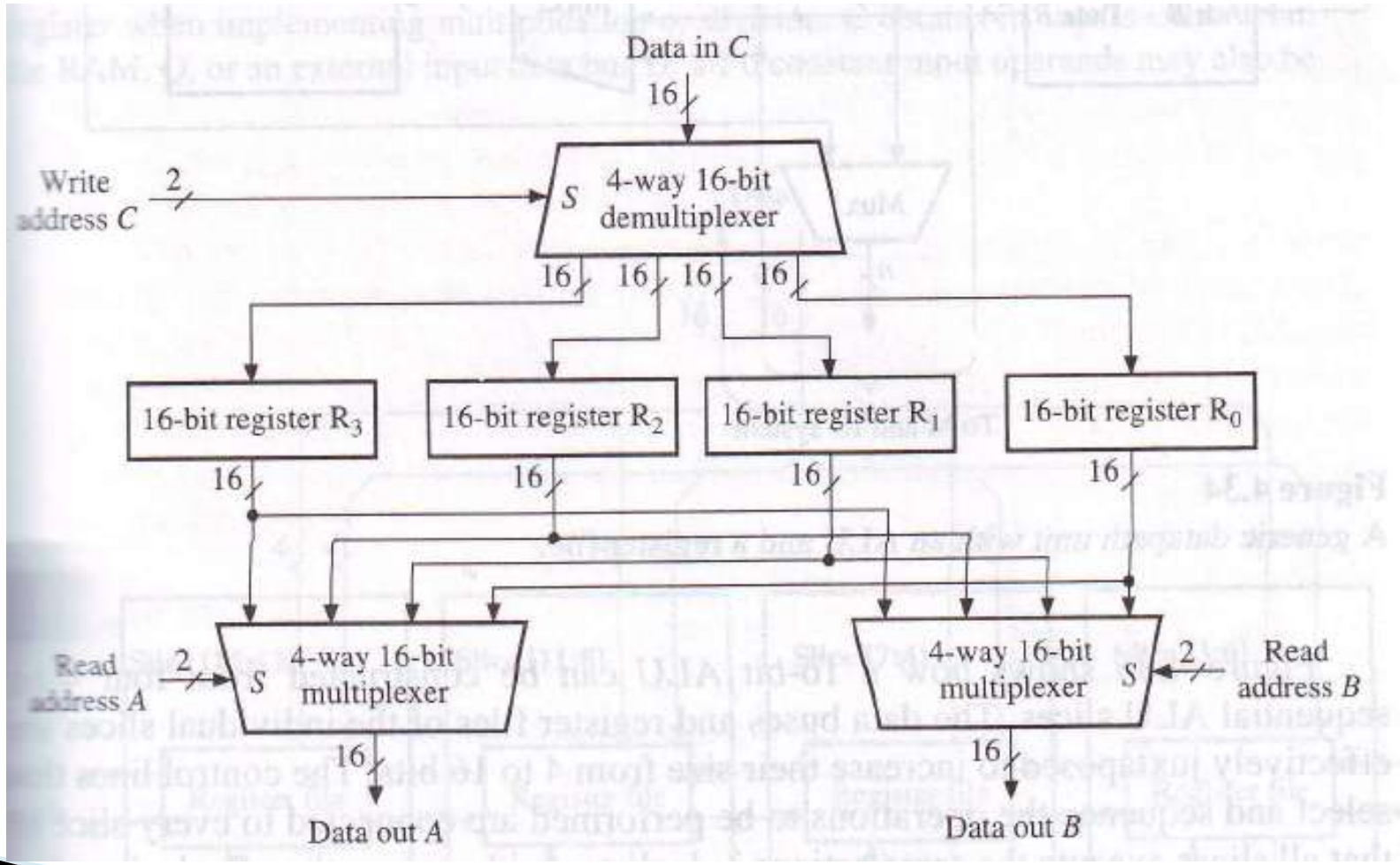


A Register File With 3 Access Ports



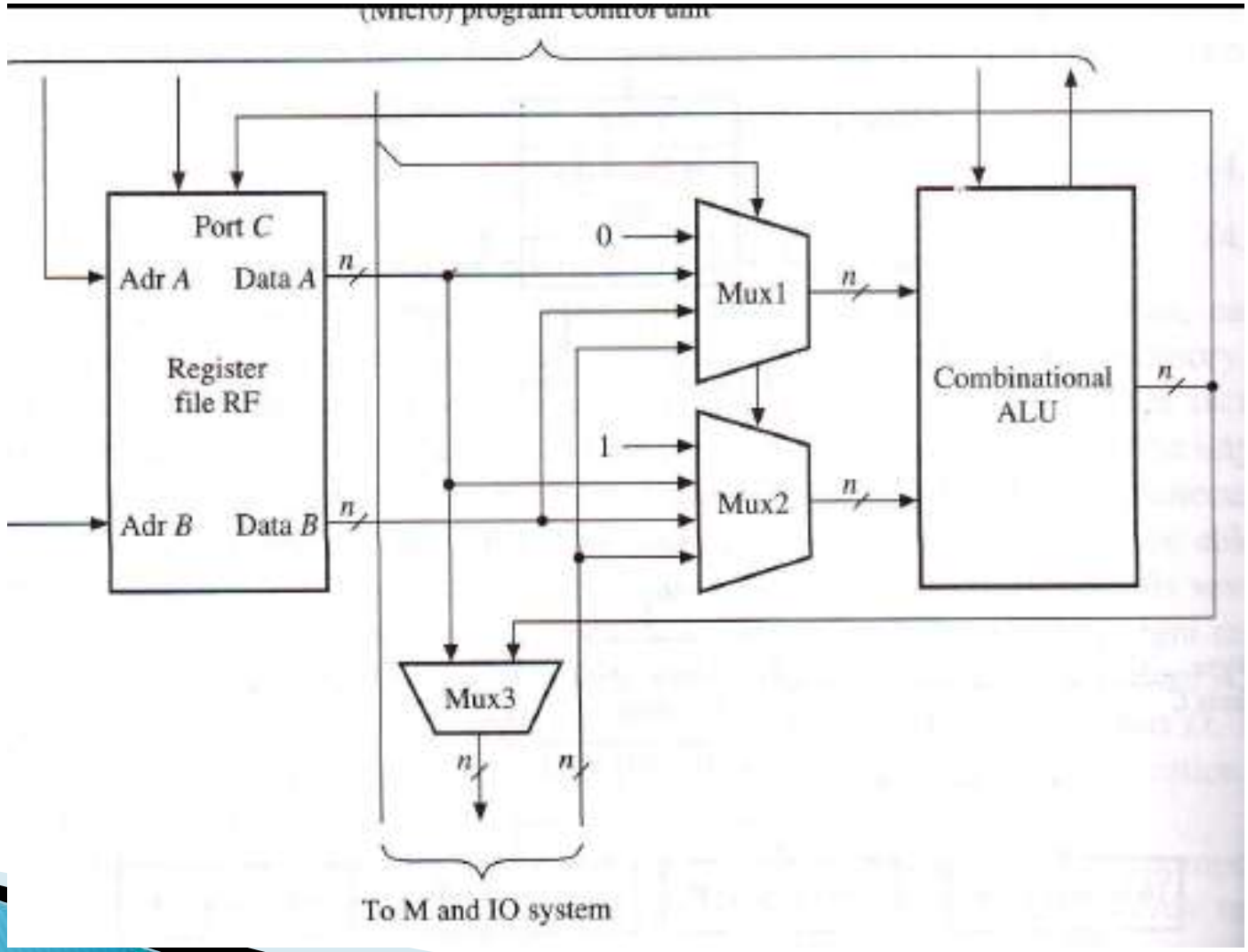
A Symbol

A Register File With 3 Access Ports

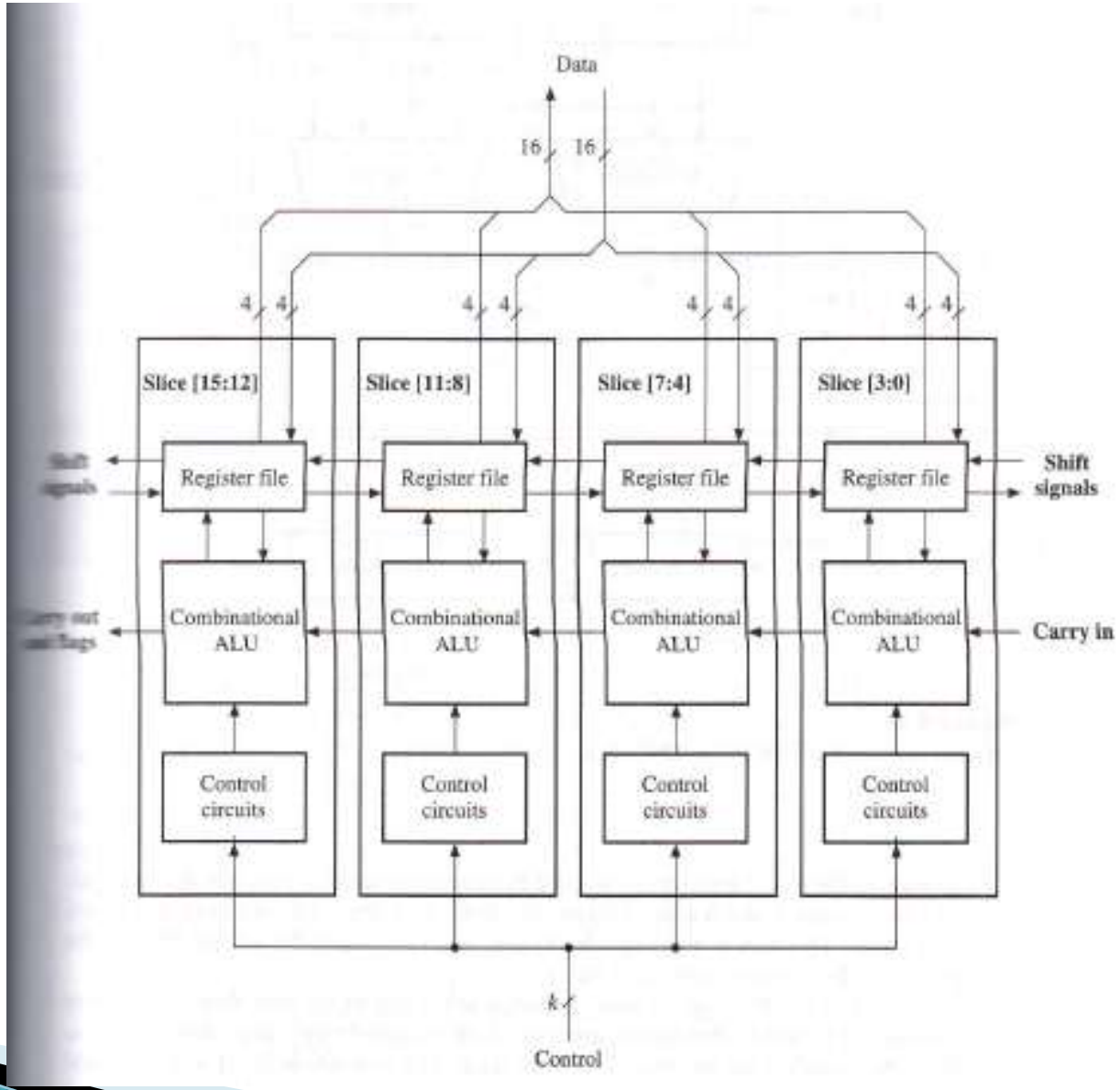


A Logic Diagram

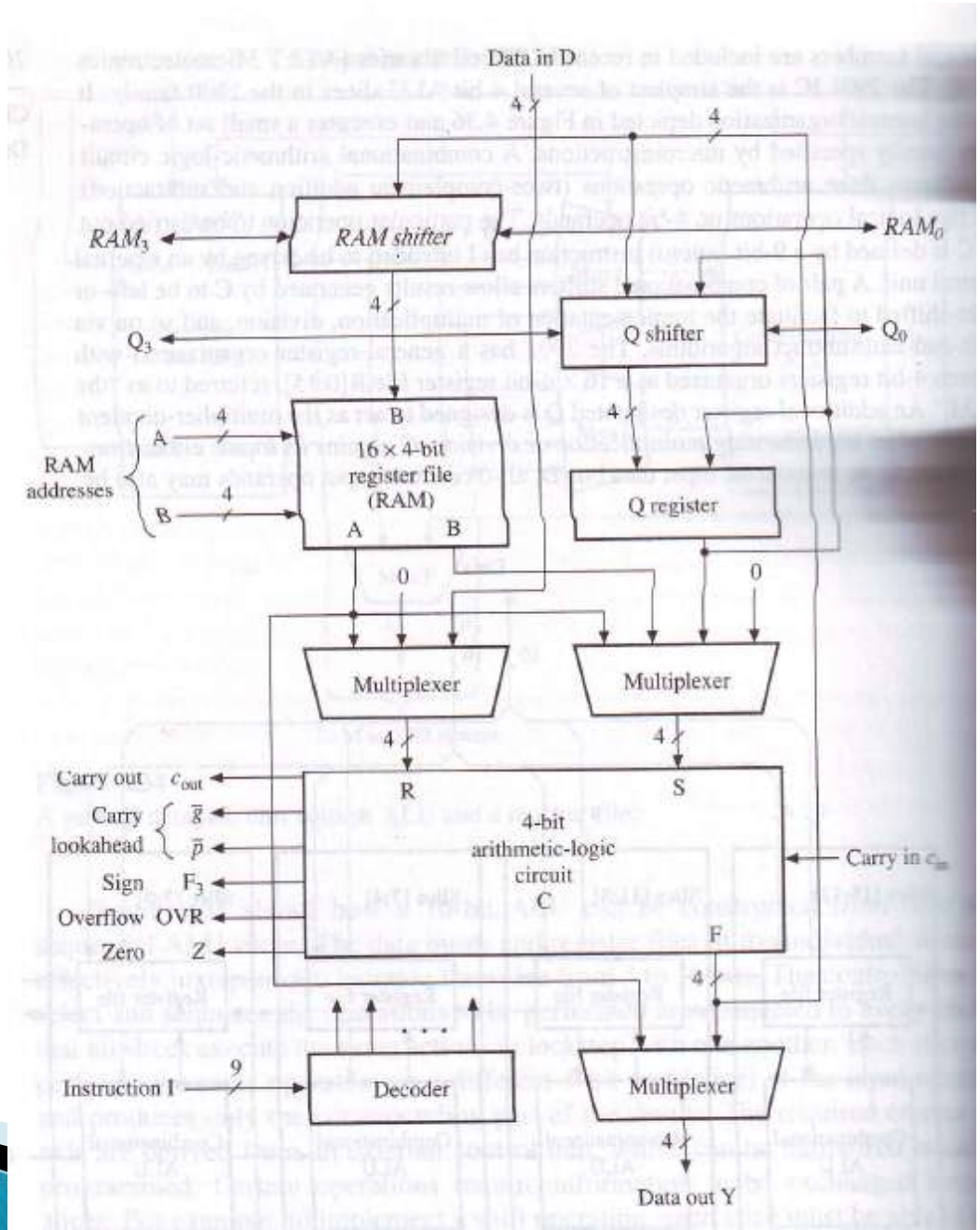
A Generic Datapath Unit With an ALU & Register File:



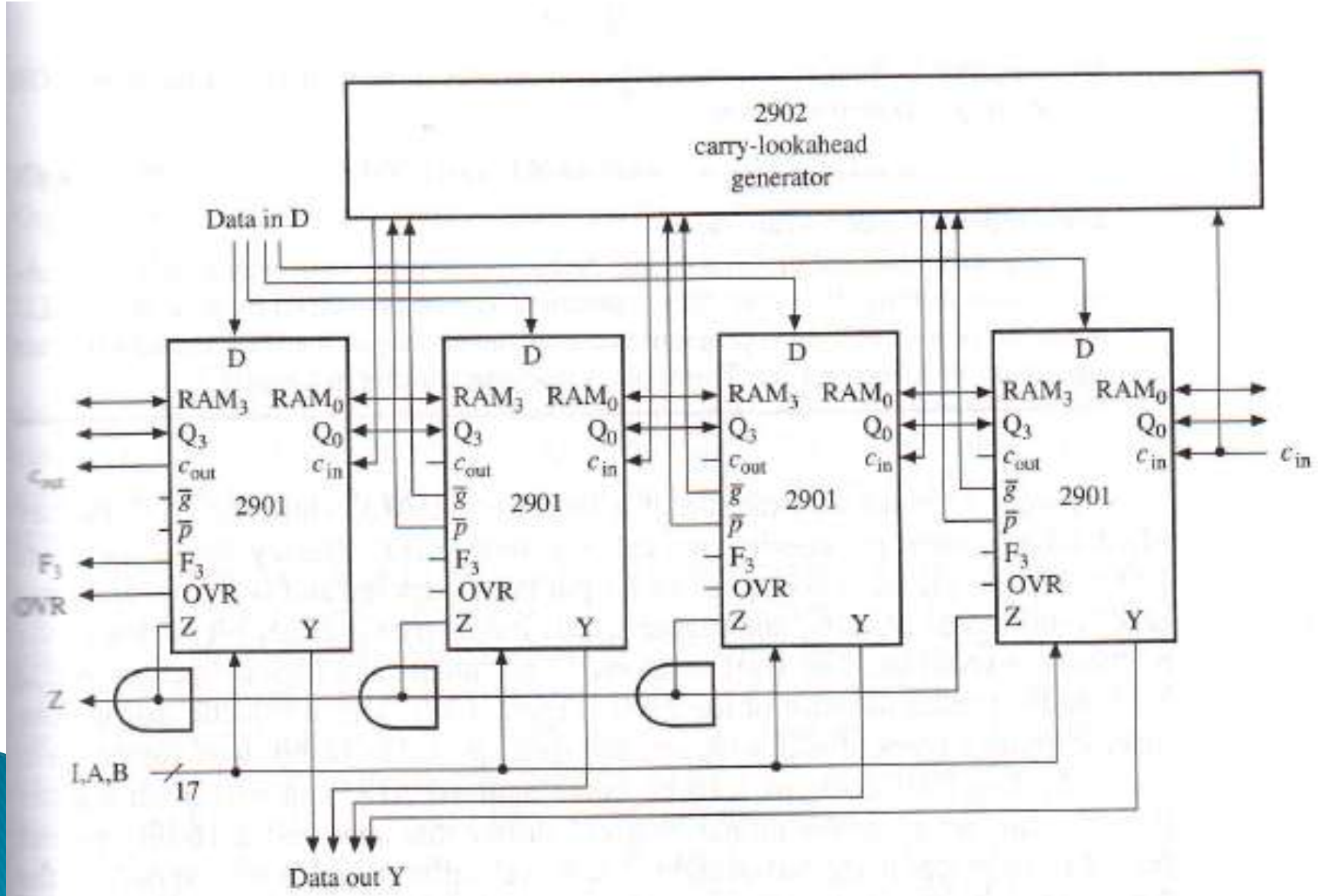
16-Bit ALU composed of four 4-bit Slices:



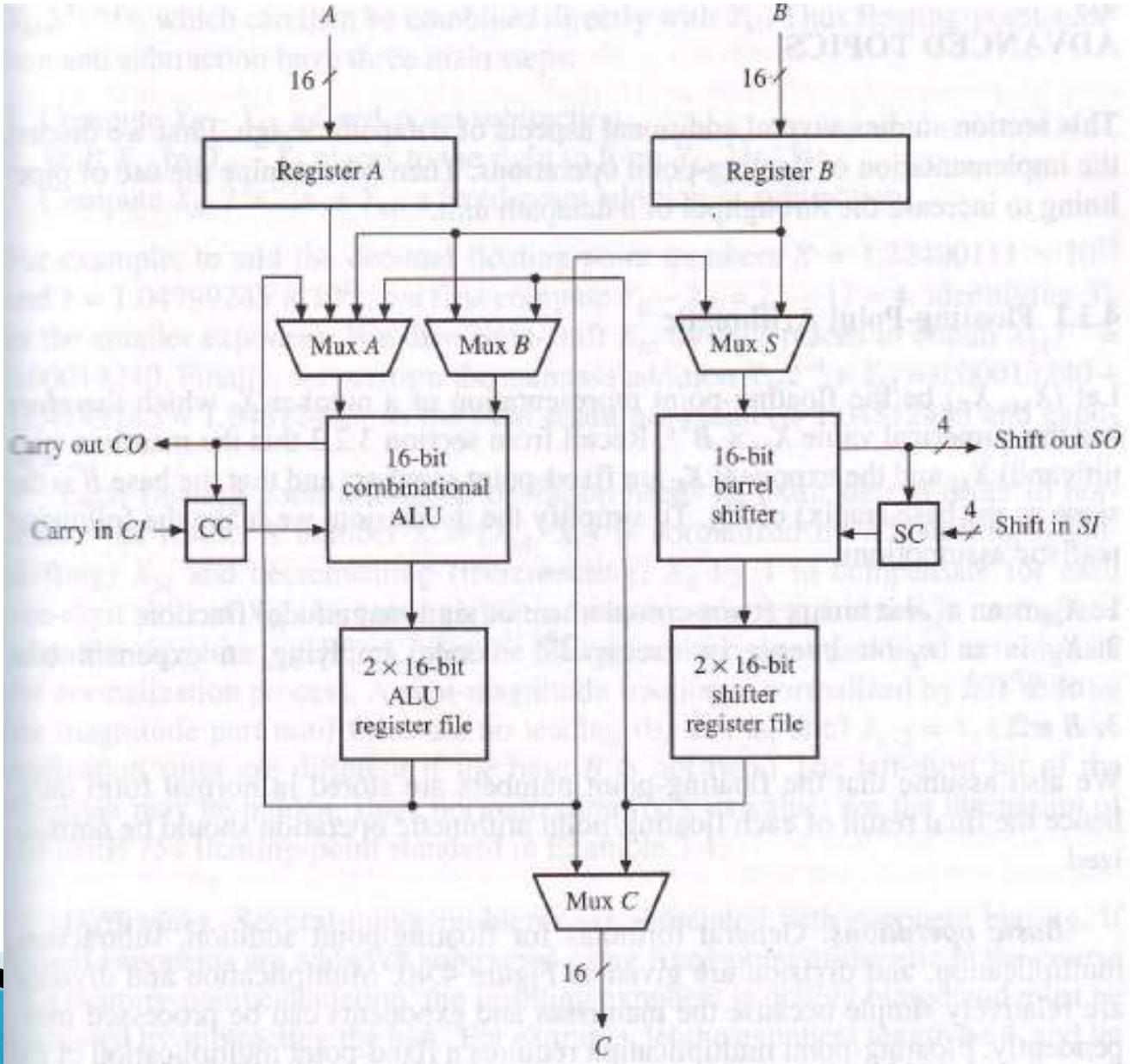
Organization of The 2901 4 bit ALU Slice:



A 16-Bit 4 Slice Array of 2901 Employing CarryLook-Ahead Generator:



Organization of GEC Plessey 1601 ALU & Barrel Shifter:



Floating Point Arithmetic

- Let (X_m, X_E) be the floating point representation of a number X which has numerical value $X_M \times B^{XE}$.
- Where X_m is the mantissa and the exponent X_E are fixed point numbers.

- The four basic arithmetic operations for floating point numbers are:
  - $X+Y = (X_m \ 2^{X_E - Y_E} + Y_M) \times 2^{Y_E}$
  - $X-Y = (X_m \ 2^{X_E - Y_E} - Y_M) \times 2^{Y_E}$
  - $X \times Y = (X_m \times Y_M) \times 2^{X_E + Y_E}$
  - $X / Y = (X_m / Y_M) \times 2^{X_E - Y_E}$

- The floating point operation has 4 main steps:
  - Compare The Exponent
  - Align The Mantissas
  - Add The Mantissas
  - Normalize The Result
- The Floating point addition and subtraction has 3 main steps:
  - Compute $Y_E - X_E$
  - Shift X_M by $Y_E - X_E$ places to the right to obtain X_M $2X_E - Y_E$.
  - Compute $(X_m \ 2^{X_E - Y_E} + / - Y_M) \times 2^{Y_E}$

Algorithm For Addition

- The first step of the algorithm is equalization of the exponent which is done by subtracting them and aligning the mantissa by shifting one of them until the difference between the exponent has been reduced to 0.

- The aligned mantissas are added.

- Finally the result is normalized, if it is necessary by again shifting the mantissa and making a compensating change in the exponent.

Algorithm For Addition Contd…

- Note: The left most bit position of E indicates the sign of the exponent if it is 0 then the exponent is negative if it is 1 then it is positive.
- Load E1, E2, AC & DR
- Compare E=E1-E2
- If E<0 Then Ac:=right-shift(Ac), E:=E+1
- If E>0 Then DR:= right-shift(DR), E:=E-1
- If E=0 Then AC:=AC+DR, E:= max(E1, E2)

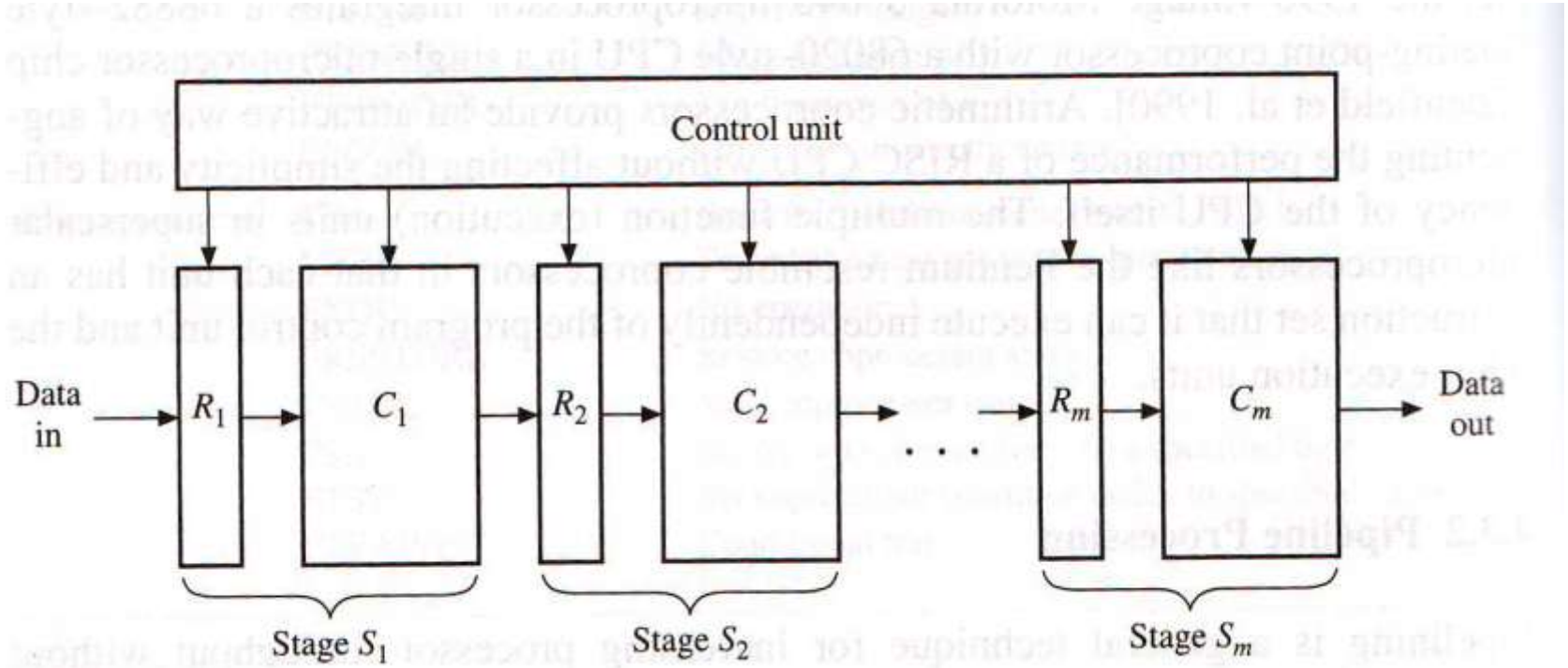
For Eg:

- X = 0 01111111 10000000000.....
- Y = 0 10000111 00101011101000.....

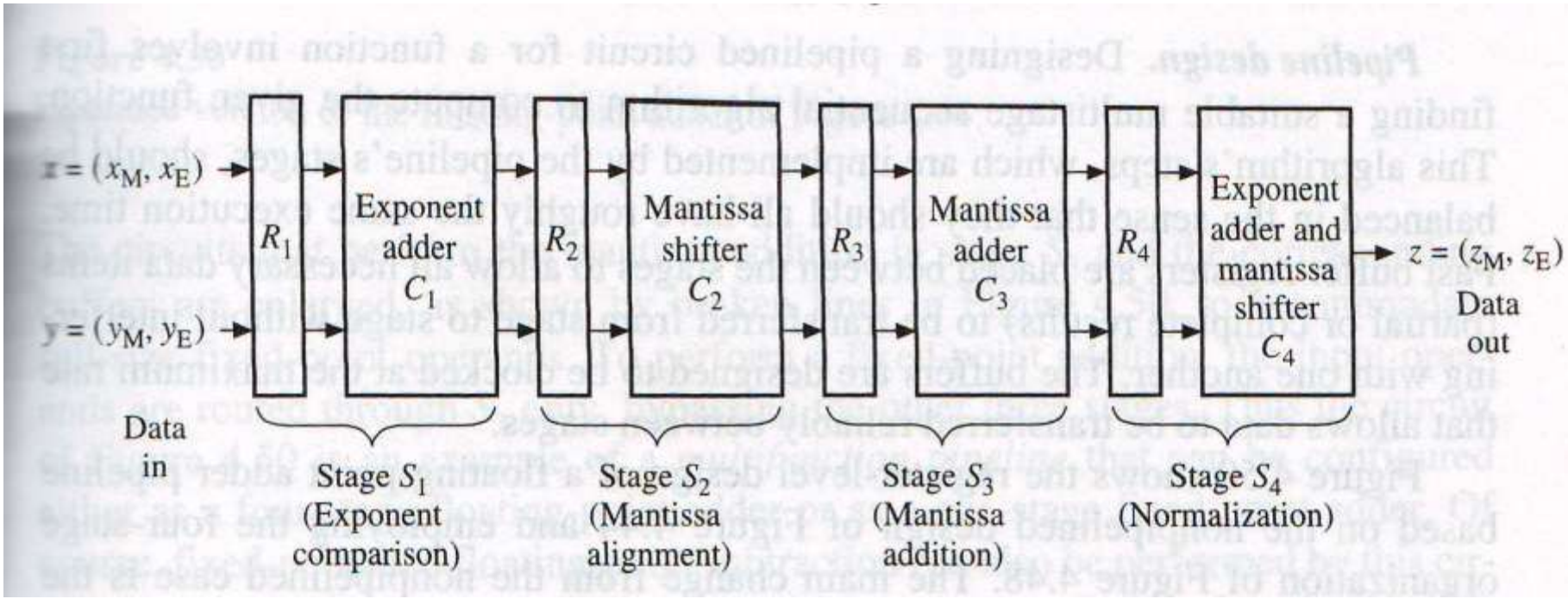
- In the above example the number has 32 bit floating point format of IEEE standard 754. In this format each number N has 23 bit fractional part with a hidden bit and an 8 bit exponent E in excess-127 code.

	Exponent registers			Mantissa registers	
	E1	E2	E	AC	DR
	01111111	10000111	0000000	110000000000 . . . 00	100101011010 . . . 00
	$= X_E$	$= Y_E$		$= 1.X_M$	$= 1.Y_M$
			01110111		
			$= X_E - Y_E$		
			01111000	011000000000 . . . 00	
			01111001	001100000000 . . . 00	
			01111010	000110000000 . . . 00	
			01111011	000011000000 . . . 00	
			01111100	000001100000 . . . 00	
			01111101	000000110000 . . . 00	
			01111110	000000011000 . . . 00	
			01111111	000000001100 . . . 00	
			10000000		
			10000111	100101100111000 . . . 00	
			$= Y_E$	= AC + DR	
			10000111	100101100111000 . . . 00	
			$= (X + Y)_E$	$= 1.(X + Y)_M$	

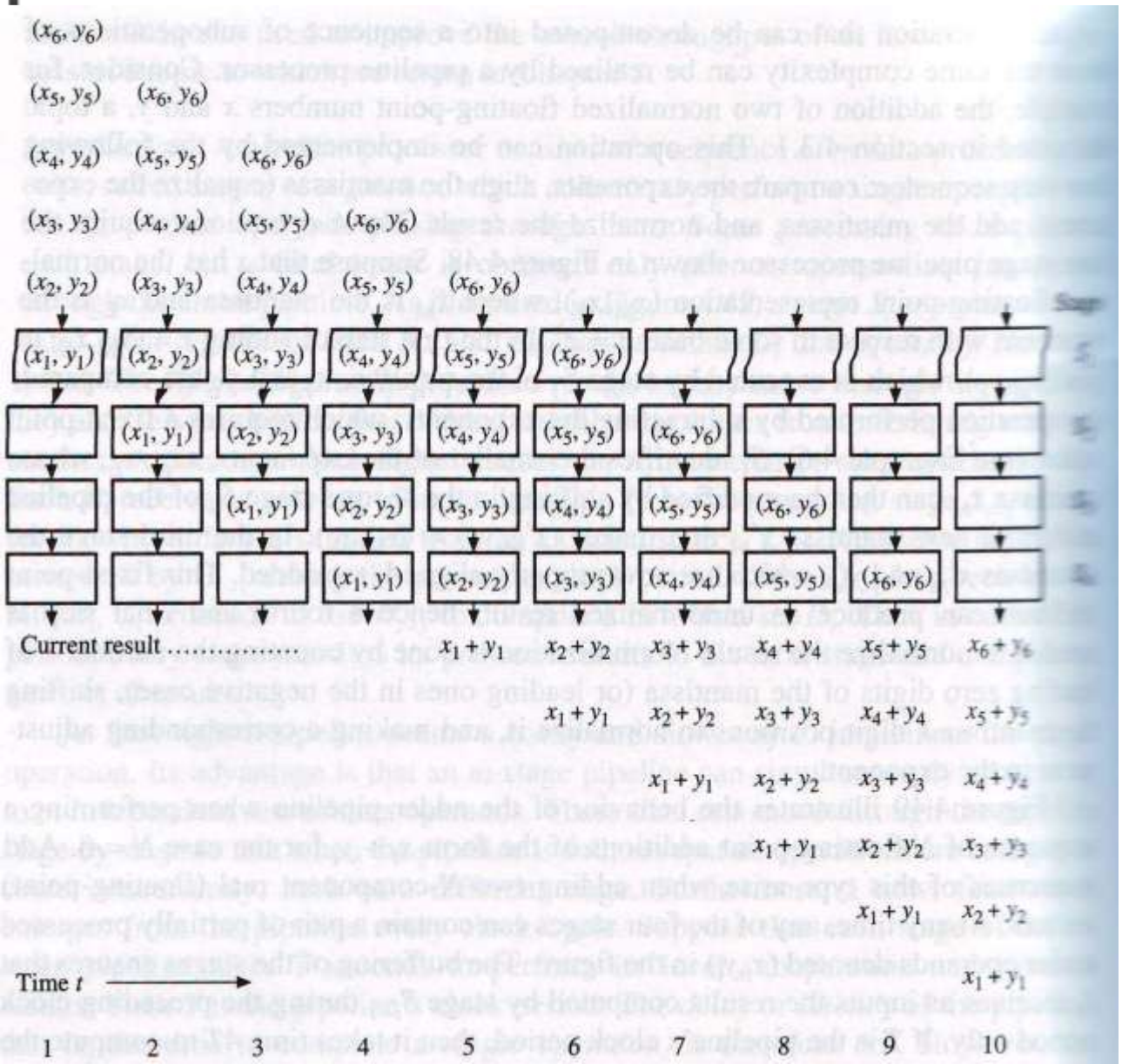
Pipeline Processing

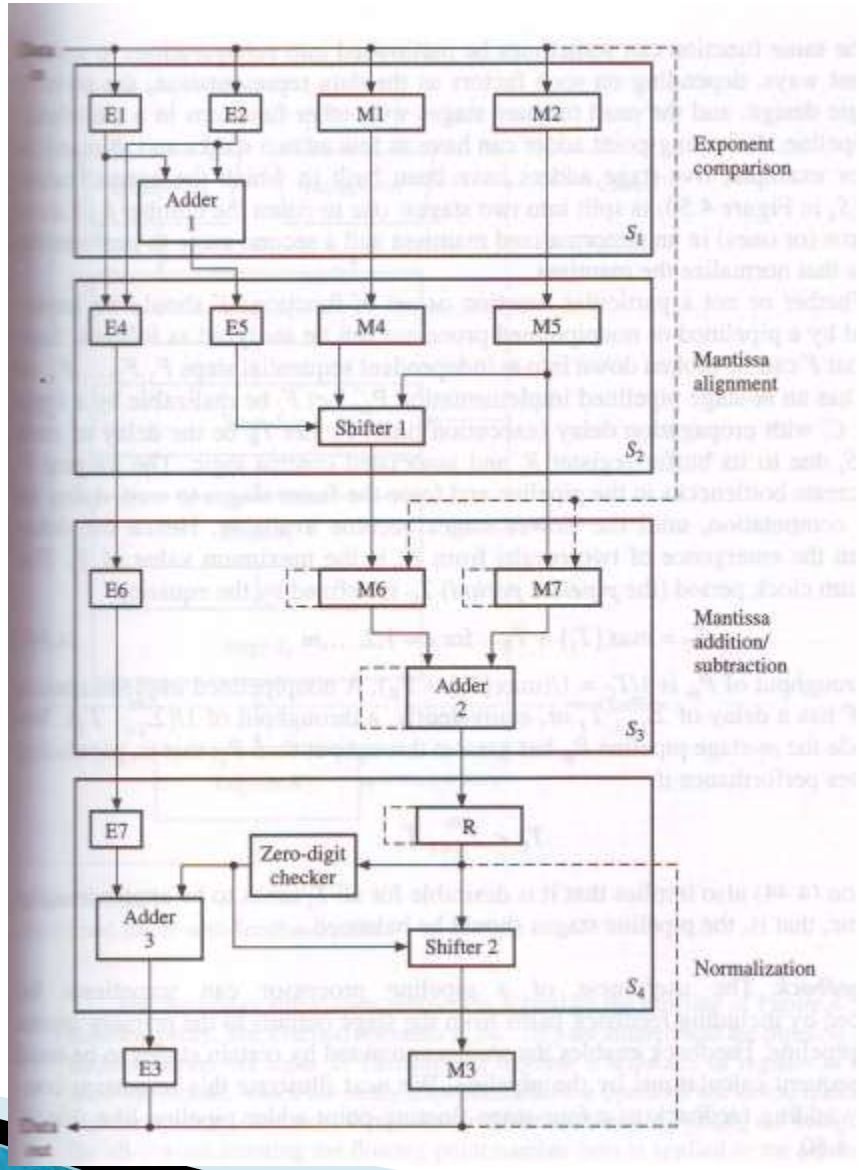


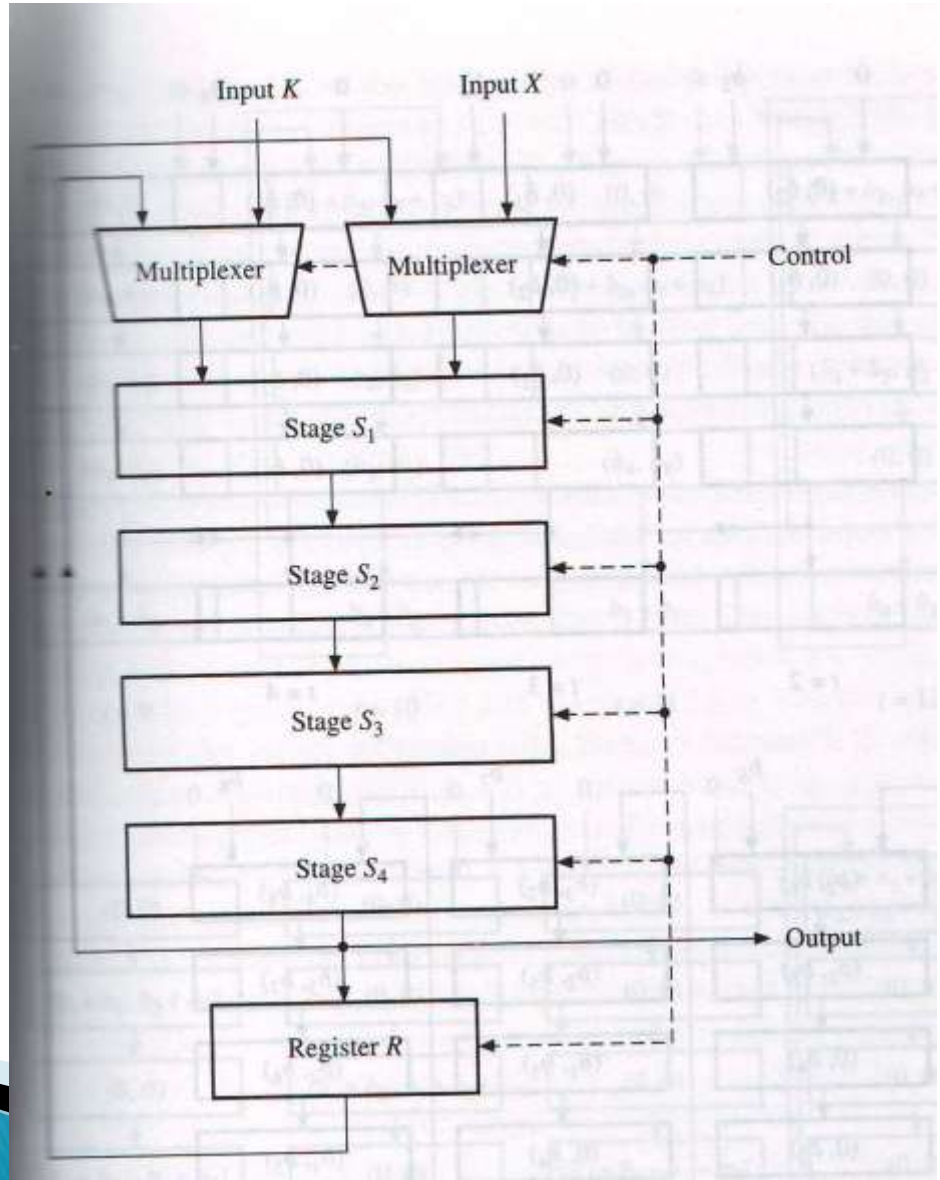
Four Stage Floating Point Adder Pipeline:



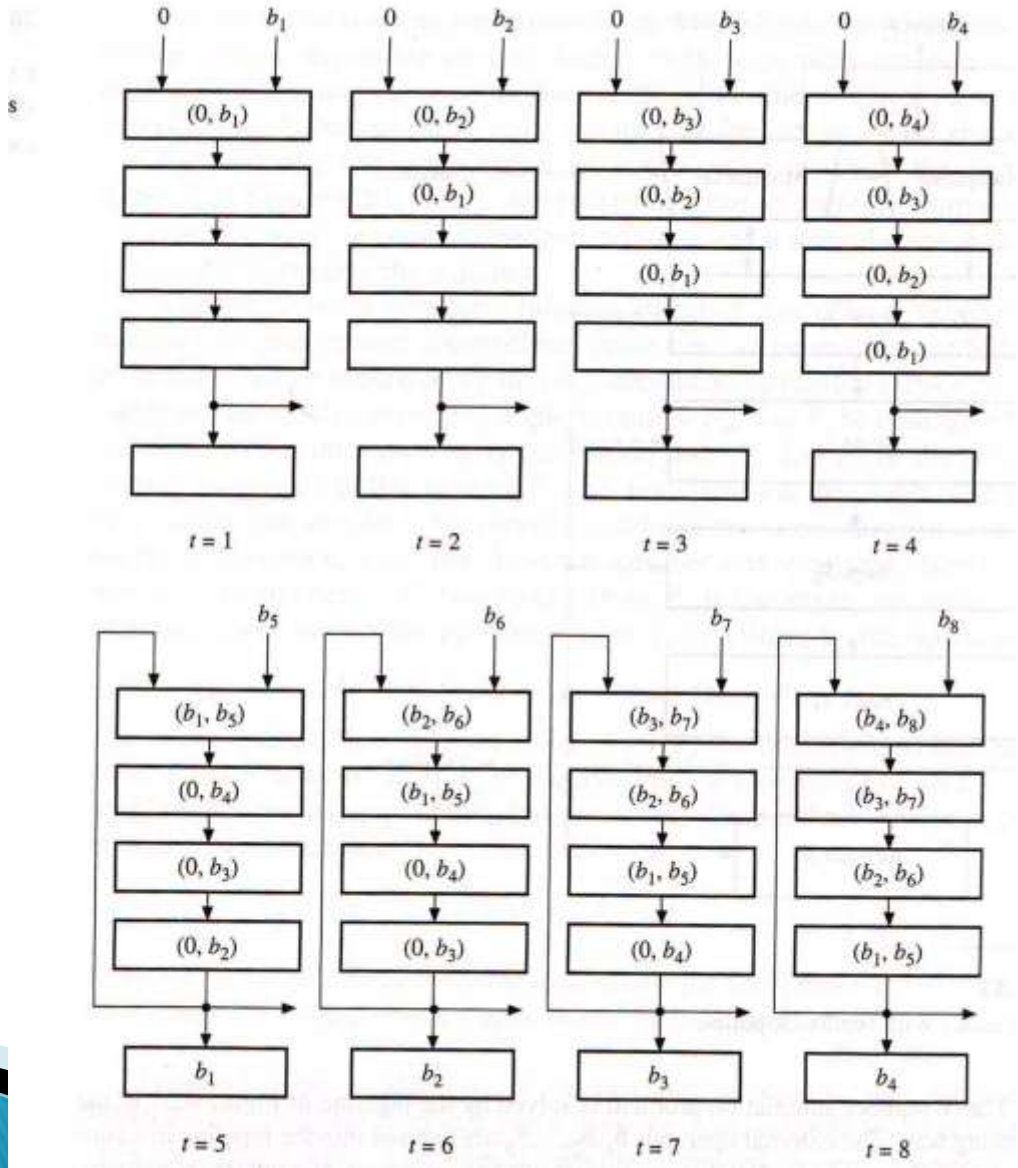
Operation of the 4 stage floating-point adder pipeline:

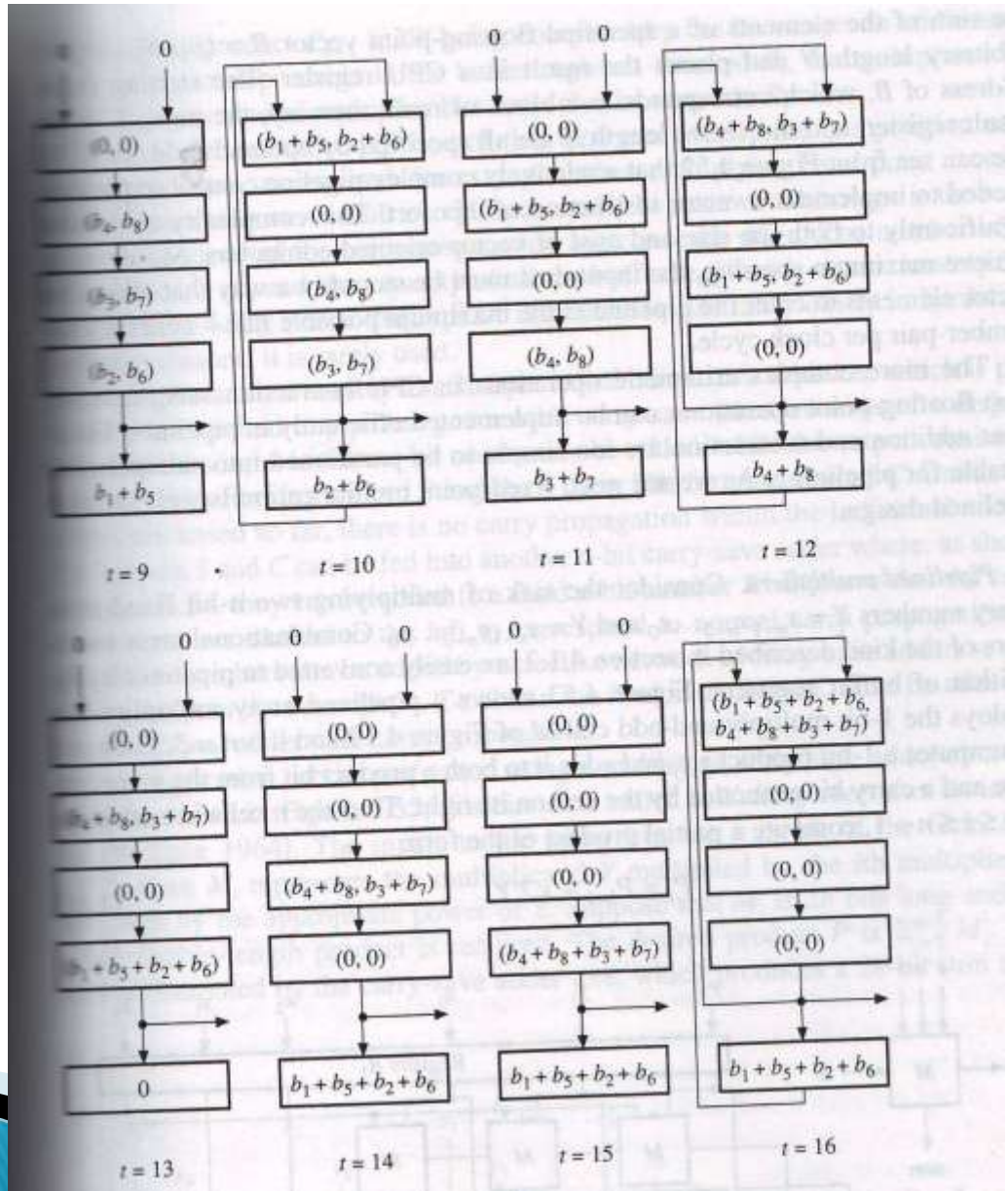



E1
E2
M1
M2
Adder 1
Exponent comparison
S_1
E4
E5
M4
M5
Shifter 1
Mantissa alignment
S_2
E6
M6
M7
Adder 2
Mantissa addition/ subtraction
S_3
E7
R
Zero-digit checker
Adder 3
Shifter 2
Normalization
S_4
E3
M3


Input K
Input X
Multiplexer
Multiplexer
Control
Stage S_1
Stage S_2
Stage S_3
Stage S_4
Output
Register R

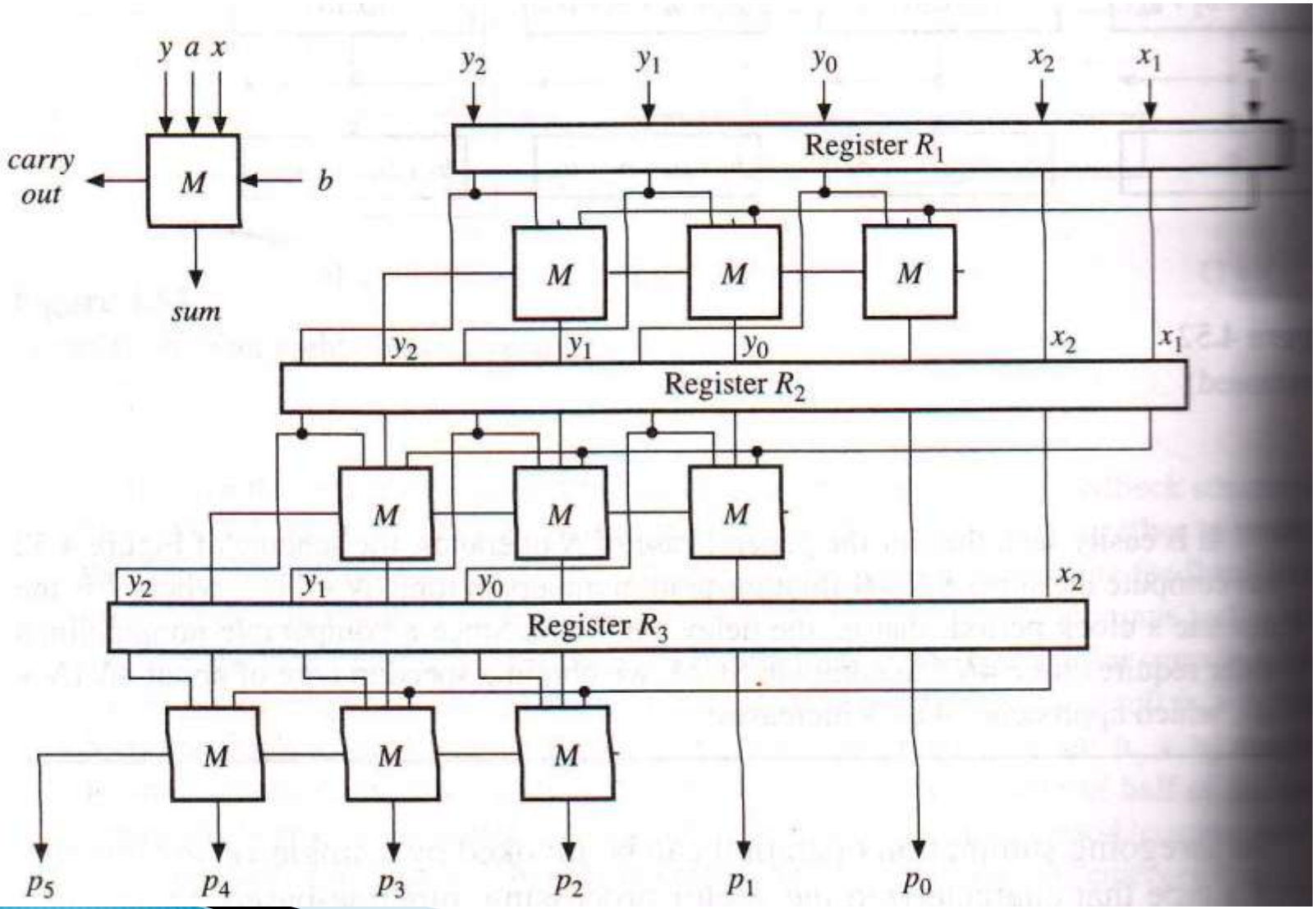
Summation of An Eight Element



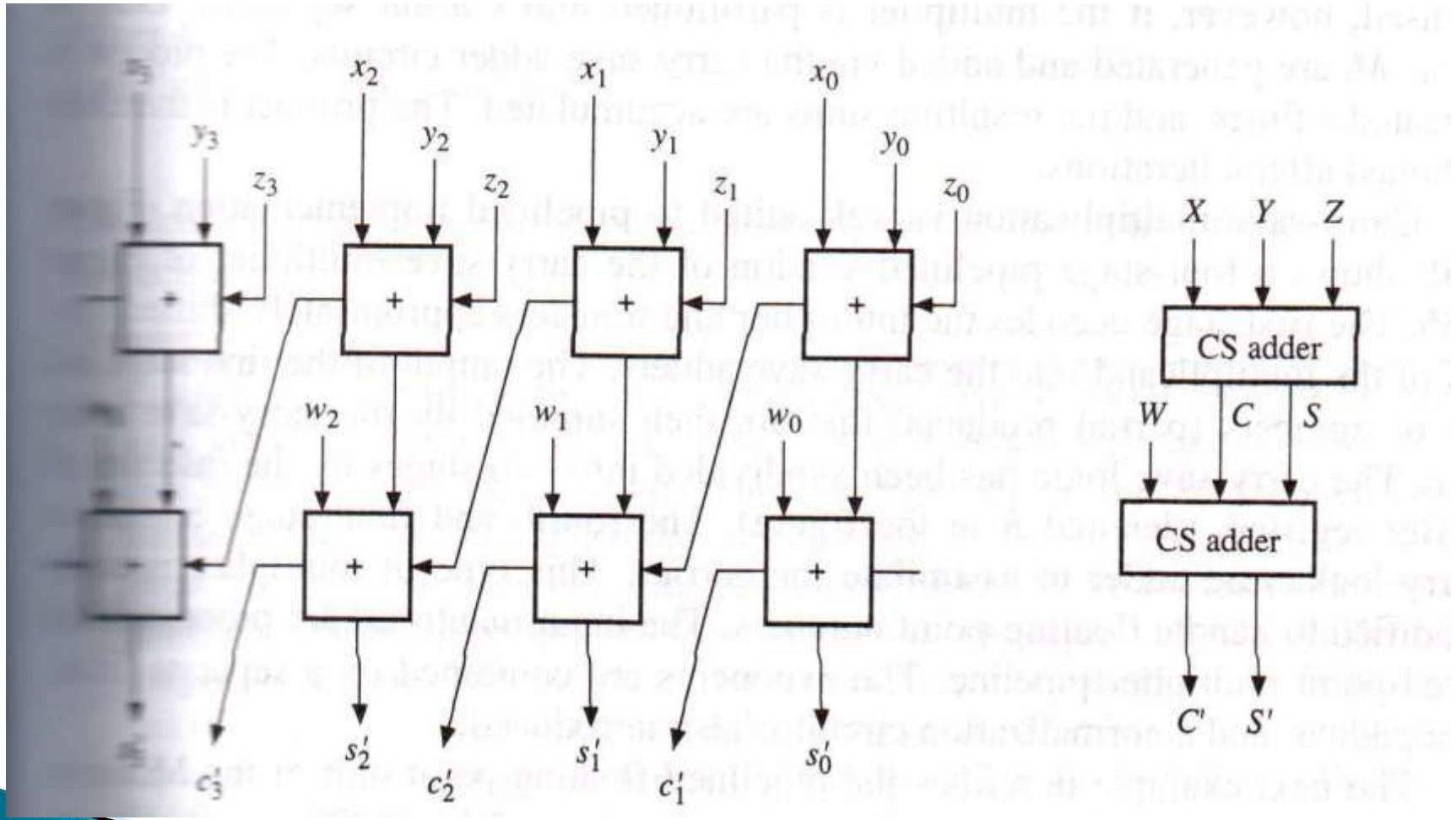

0
(0, 0)
(b_4, b_8)
(b_3, b_7)
(b_2, b_6)
$b_1 + b_5$
$t = 9$
$(b_1 + b_5, b_2 + b_6)$
(0, 0)
(b_4, b_8)
(b_3, b_7)
$b_2 + b_6$
$t = 10$
(0, 0)
$(b_1 + b_5, b_2 + b_6)$
(0, 0)
(b_4, b_8)
$b_3 + b_7$
$t = 11$
$(b_4 + b_8, b_3 + b_7)$
(0, 0)
$(b_1 + b_5, b_2 + b_6)$
(0, 0)
$b_4 + b_8$
$t = 12$
(0, 0)
$(b_4 + b_8, b_3 + b_7)$
(0, 0)
$(b_1 + b_5 + b_2 + b_6)$
0
$t = 13$
(0, 0)
(0, 0)
$(b_4 + b_8, b_3 + b_7)$
(0, 0)
$b_1 + b_5 + b_2 + b_6$
$t = 14$
(0, 0)
(0, 0)
(0, 0)
$(b_4 + b_8 + b_3 + b_7)$
$b_1 + b_5 + b_2 + b_6$
$t = 15$
$(b_1 + b_5 + b_2 + b_6, b_4 + b_8 + b_3 + b_7)$
(0, 0)
(0, 0)
(0, 0)
$b_1 + b_5 + b_2 + b_6$
$t = 16$

Pipelined Multiplier

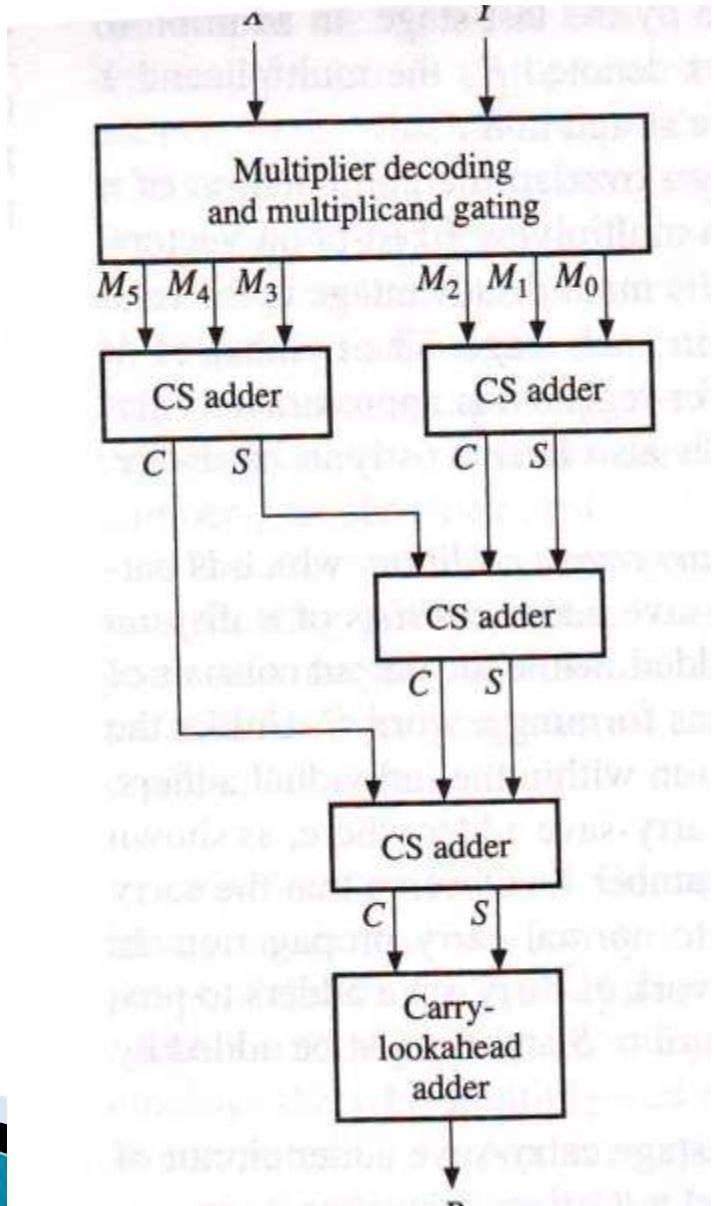
Multiplier Pipline



A Two Stage Carry-Save Adder



A Carry-Save Multiplier (Wallac-Tree)



A Pipelined Carry-Save Multiplier: