Binary Adder

- To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the arithmetic addition.

- The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.

- Figure 4-6 shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder. An n-bit binary adder requires n full-adders.

Binary Adder

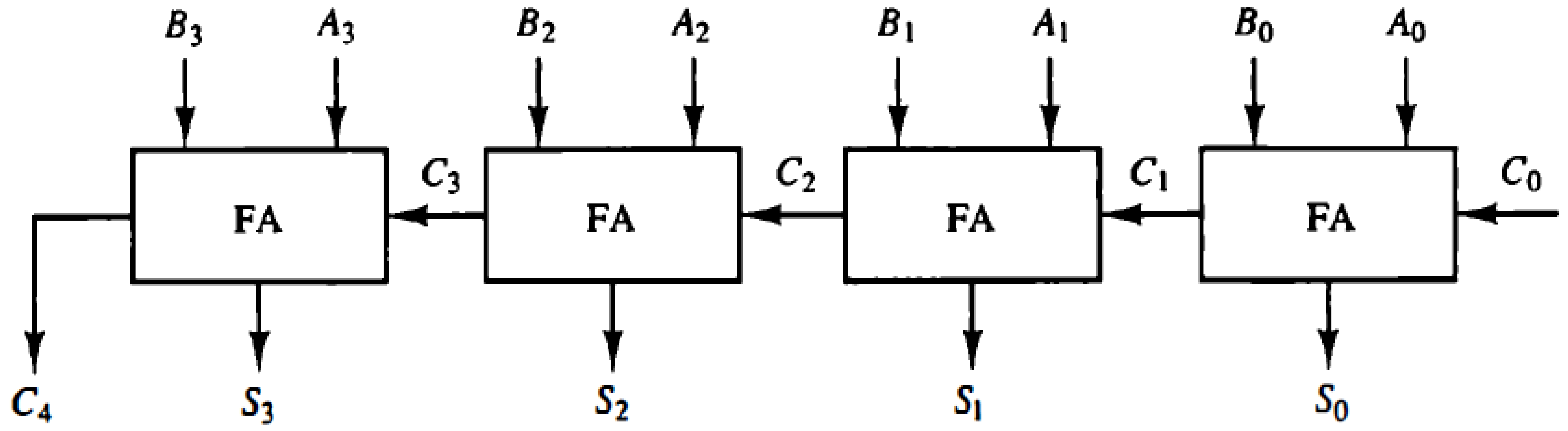
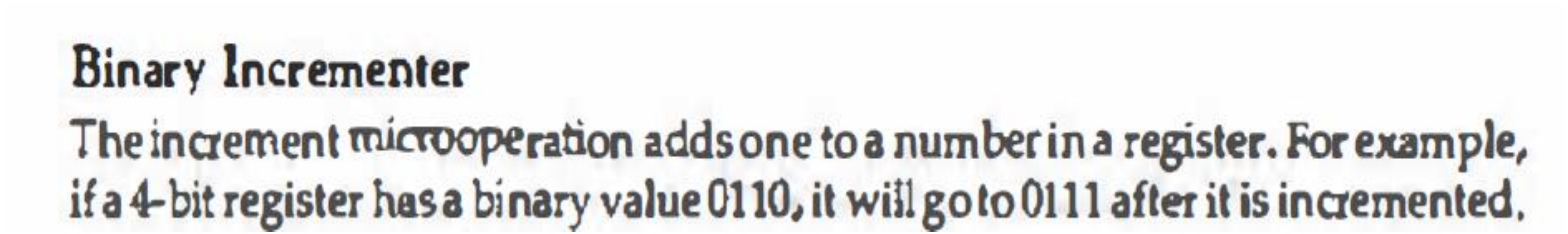


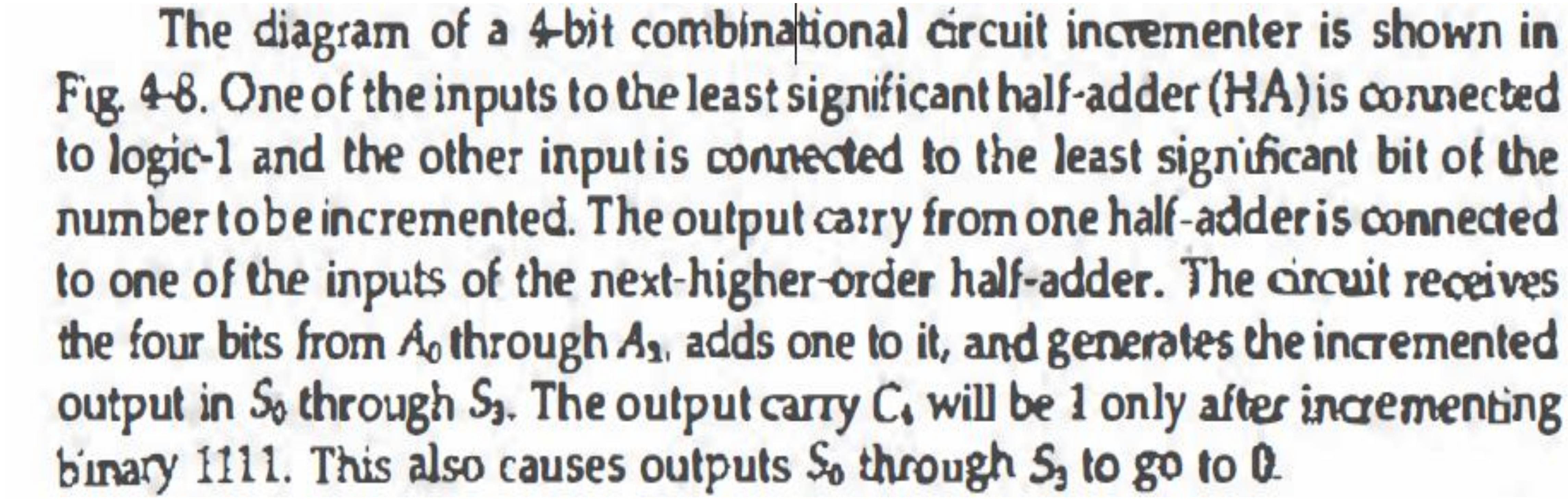
Figure 4-6 4-bit binary adder.

Binary lncrementer


Binary Incrementer

The increment microoperation adds one to a number in a register. For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented.

Binary Incrementer



The diagram of a 4-bit combinational circuit incrementer is shown in Fig. 4-8. One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented. The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder. The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3. The output carry C_4 will be 1 only after incrementing binary 1111. This also causes outputs S_0 through S_3 to go to 0.

Binary Incrementer

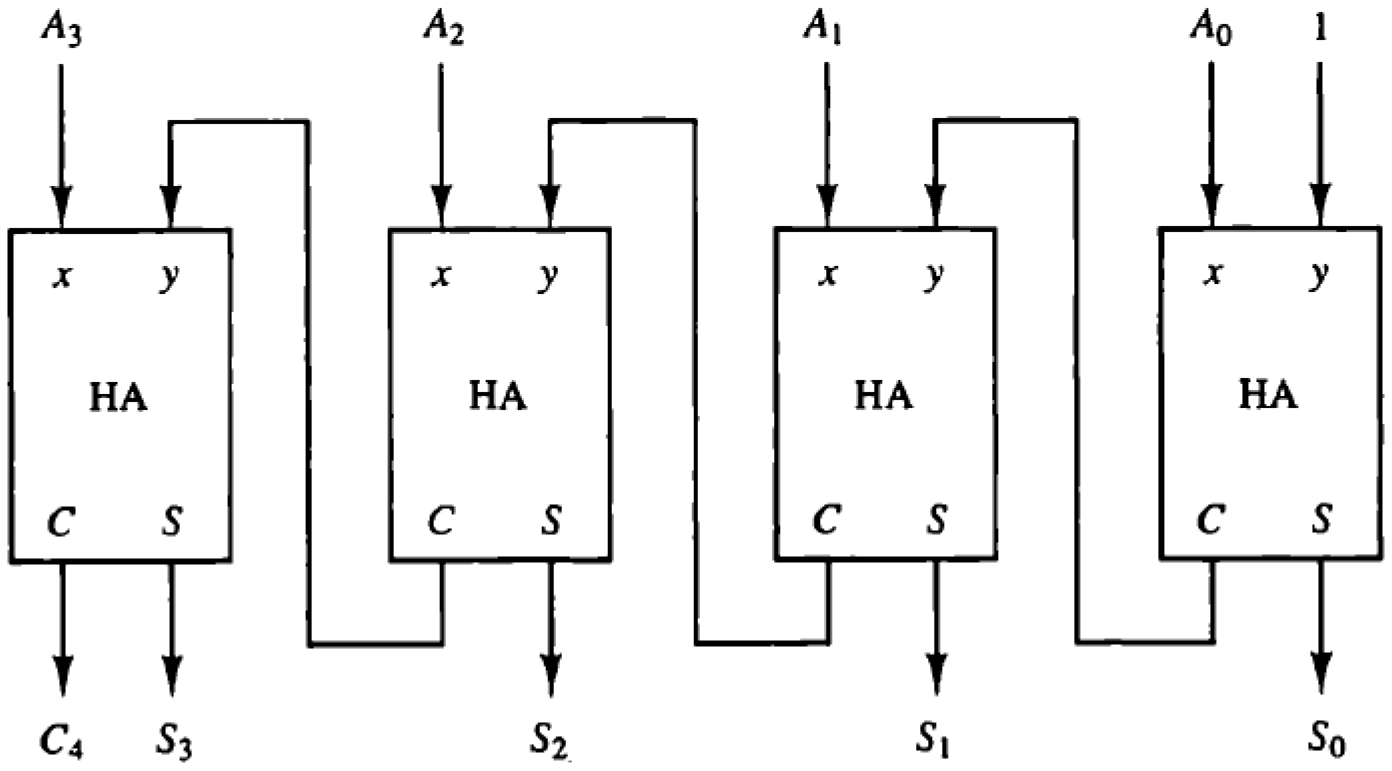
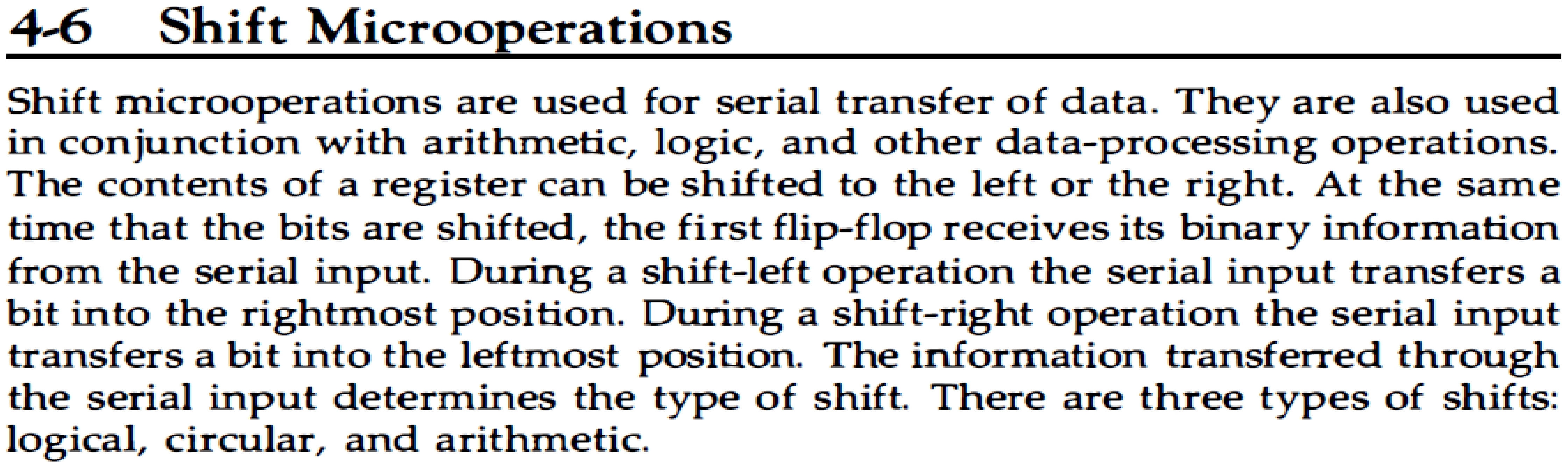


Figure 4-8 4-bit binary incrementer.

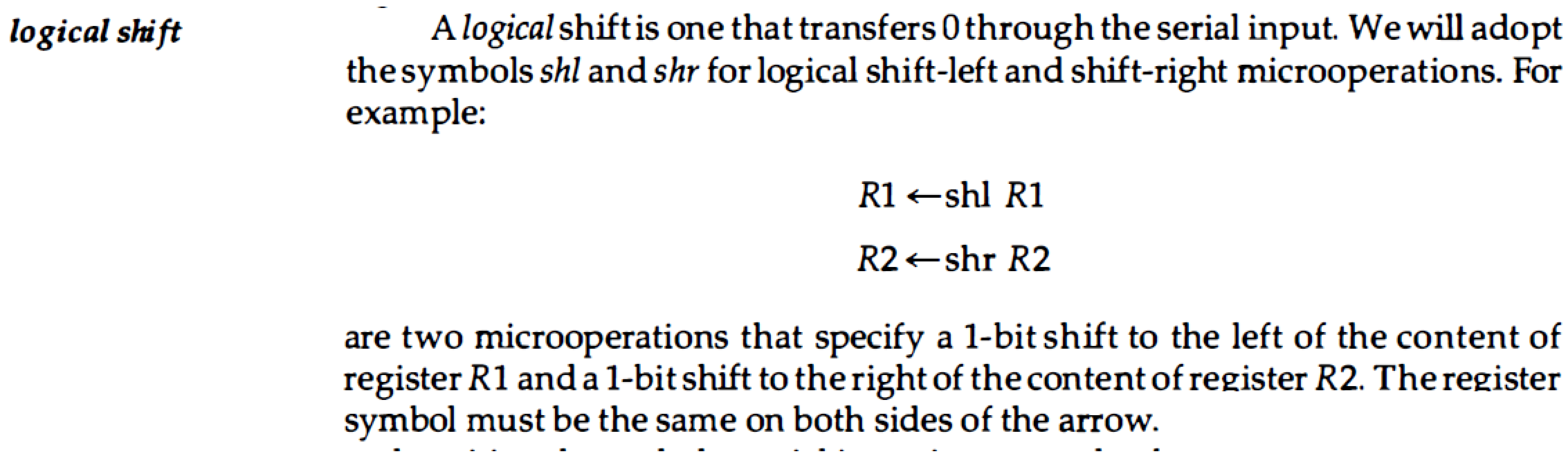
Shift Micro operations



4-6 Shift Microoperations

Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operations. The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input. During a shift-left operation the serial input transfers a bit into the rightmost position. During a shift-right operation the serial input transfers a bit into the leftmost position. The information transferred through the serial input determines the type of shift. There are three types of shifts: logical, circular, and arithmetic.

Shift Micro operations



logical shift

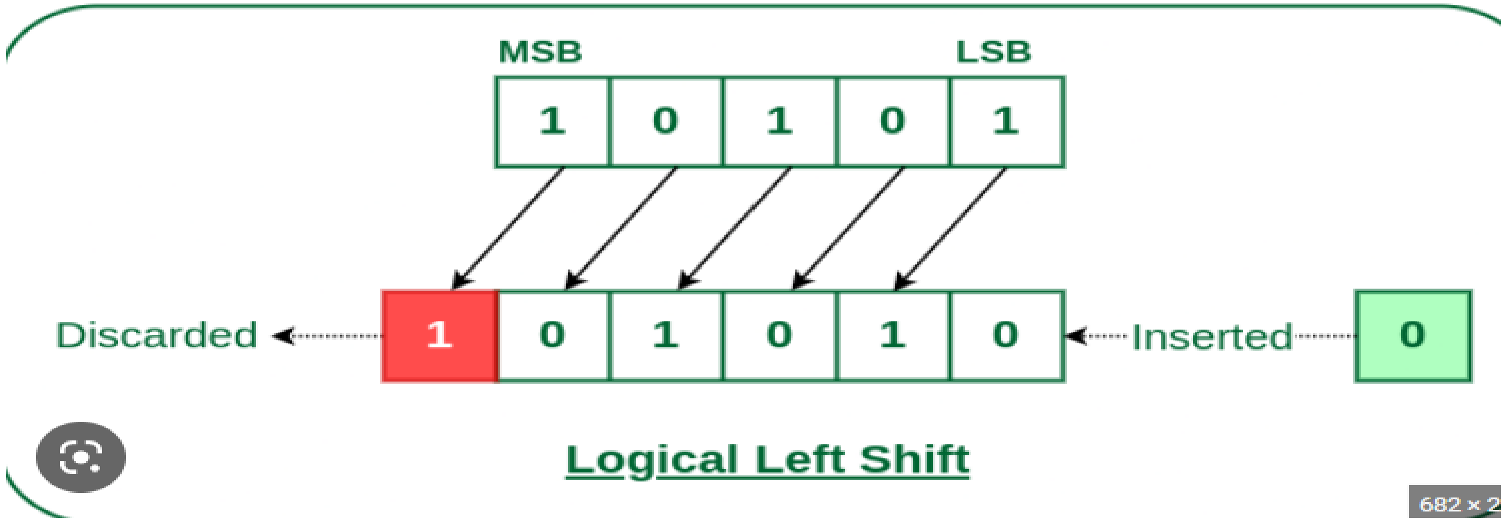
A *logical* shift is one that transfers 0 through the serial input. We will adopt the symbols *shl* and *shr* for logical shift-left and shift-right microoperations. For example:

$$R1 \leftarrow \text{shl } R1$$

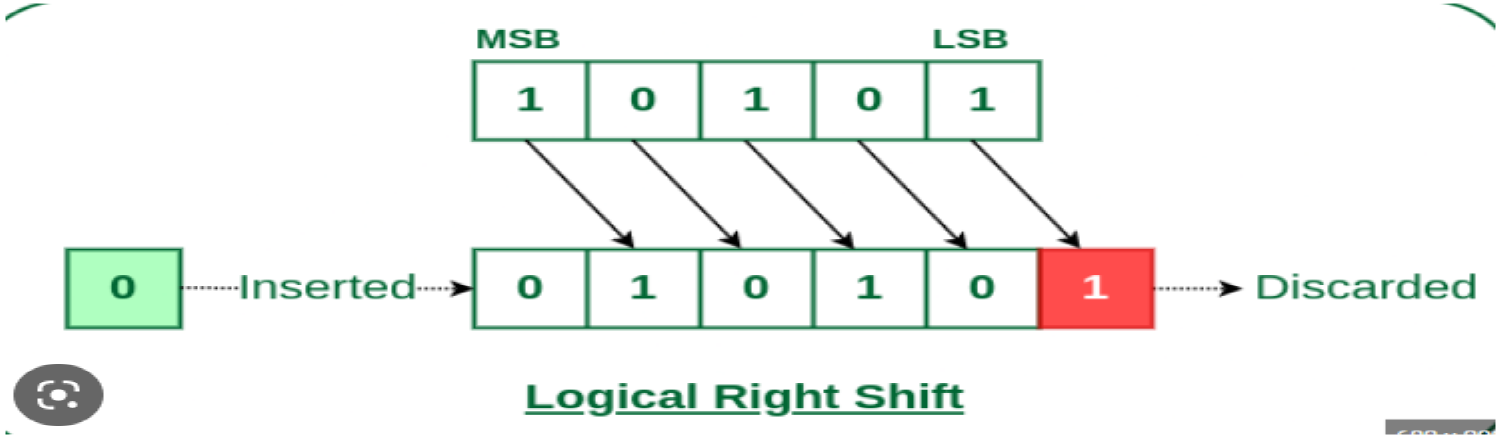
$$R2 \leftarrow \text{shr } R2$$

are two microoperations that specify a 1-bit shift to the left of the content of register $R1$ and a 1-bit shift to the right of the content of register $R2$. The register symbol must be the same on both sides of the arrow.

The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

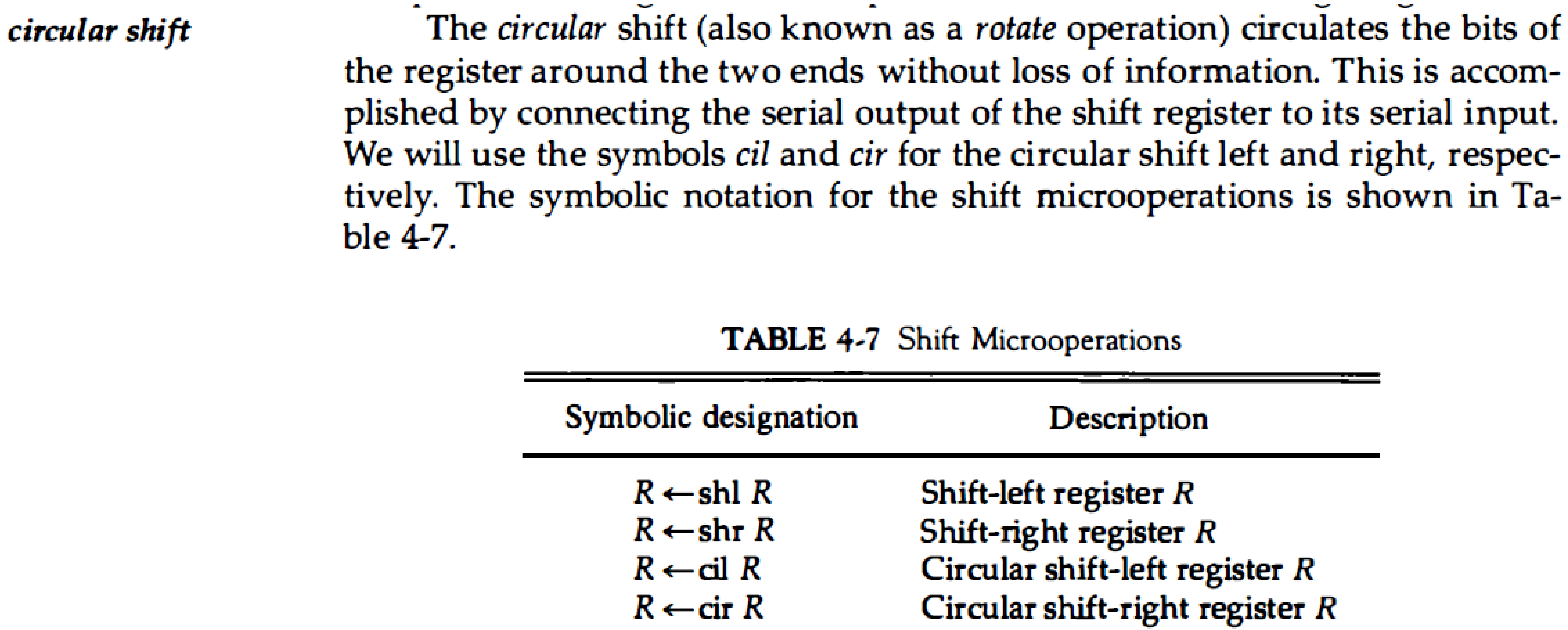


Logical Left Shift



Logical Right Shift

Shift Micro operations



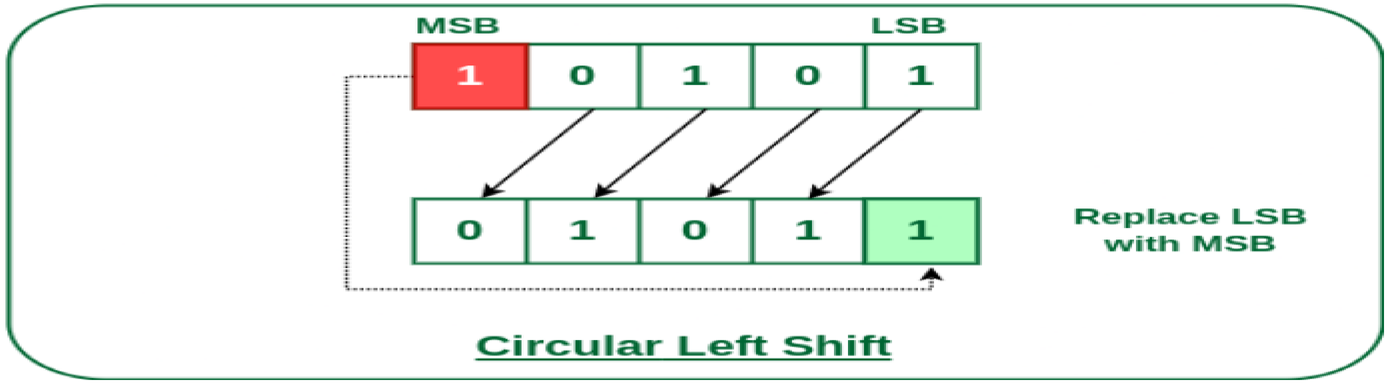
circular shift

The *circular* shift (also known as a *rotate* operation) circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input. We will use the symbols *cil* and *cir* for the circular shift left and right, respectively. The symbolic notation for the shift microoperations is shown in Table 4-7.

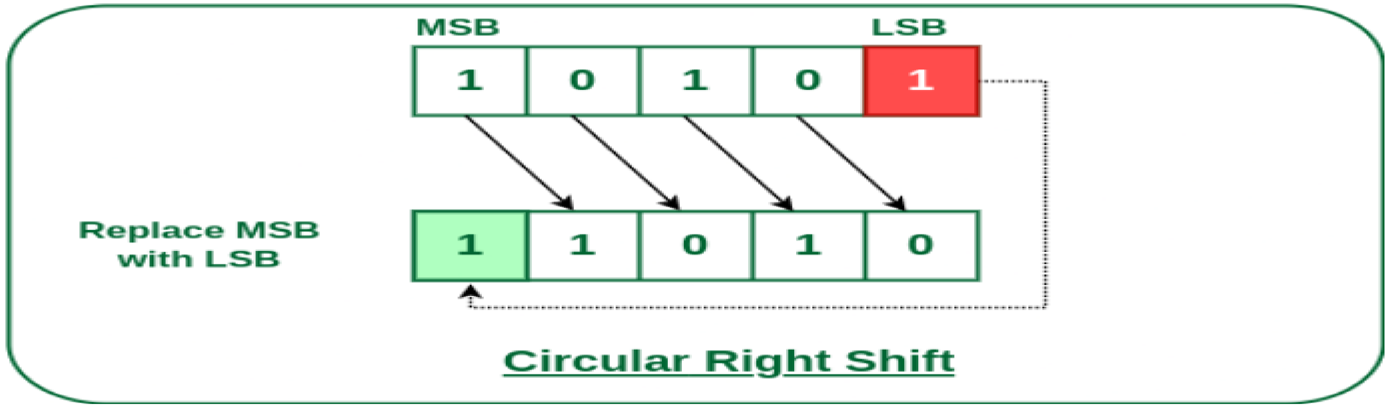
TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R

Shift Micro operations



Circular Left Shift



Circular Right Shift

Arithmetic shift operations

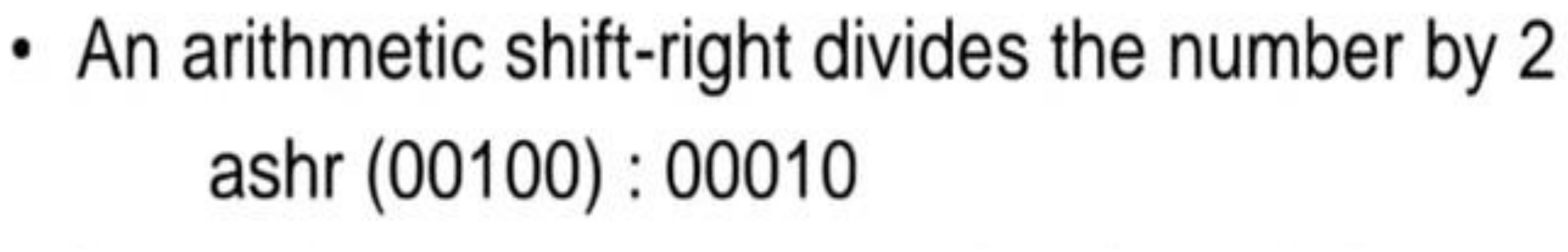
An arithmetic shift is a micro operation that shifts a signed binary number to the left or right.

An arithmetic shift-left multiplies signed binary number by 2.

An arithmetic shift-right divides the binary number by 2.

The sign bit is 0 for positive and 1 for negative.

Arithmetic Right shift operations

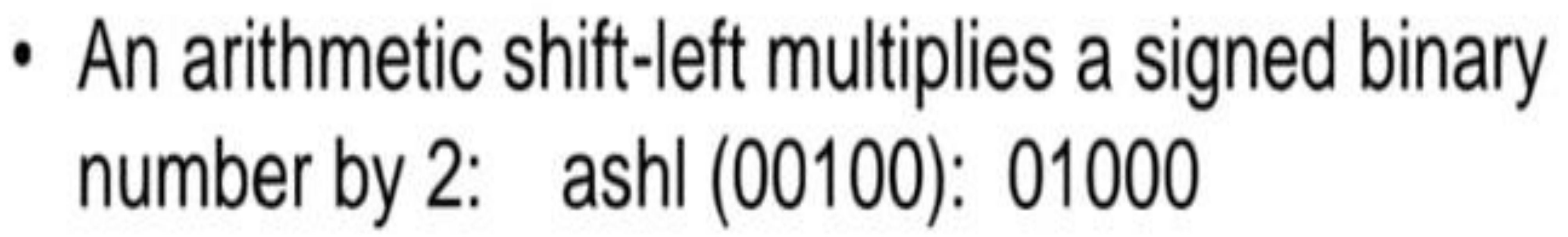


- An arithmetic shift-right divides the number by 2

  ashr (00100) : 00010

•A ***Right Arithmetic Shift*** of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with the value of the previous (now shifted one position to the right) MSB.

Arithmetic left shift operations

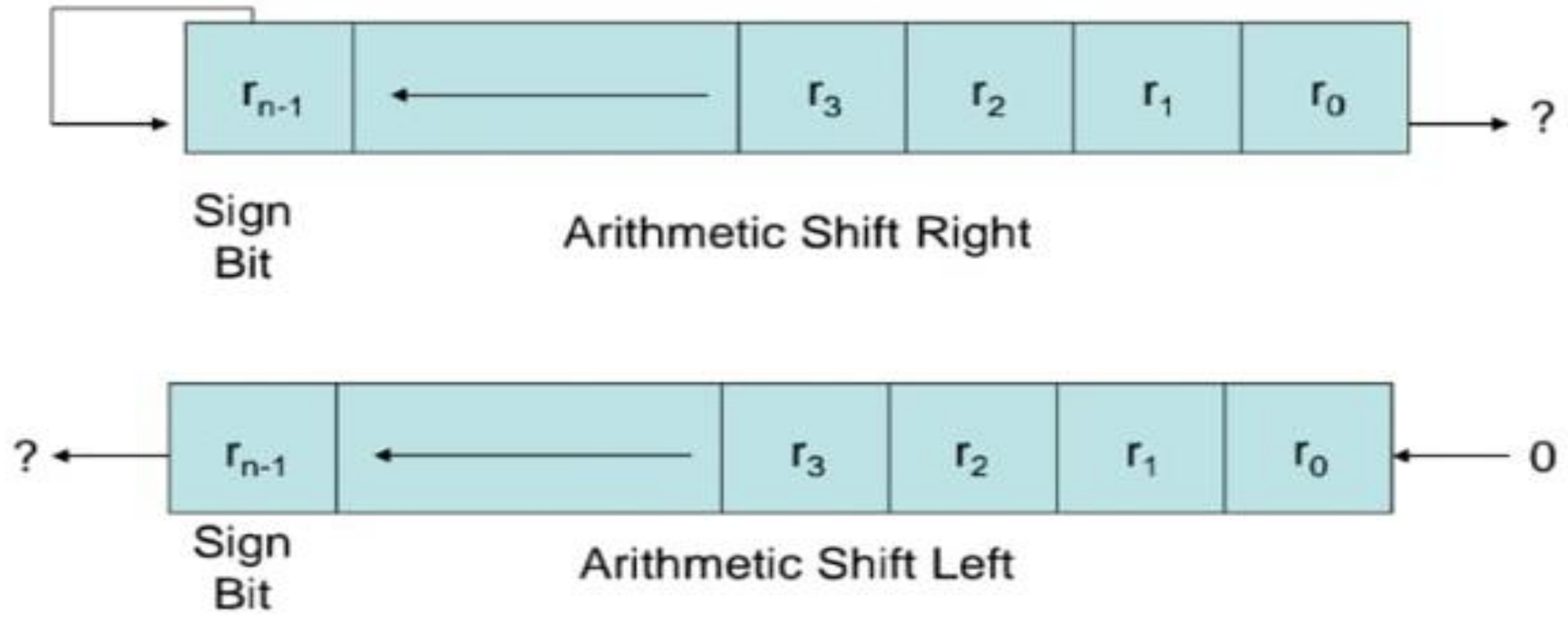


- An arithmetic shift-left multiplies a signed binary number by 2: ashl (00100): 01000

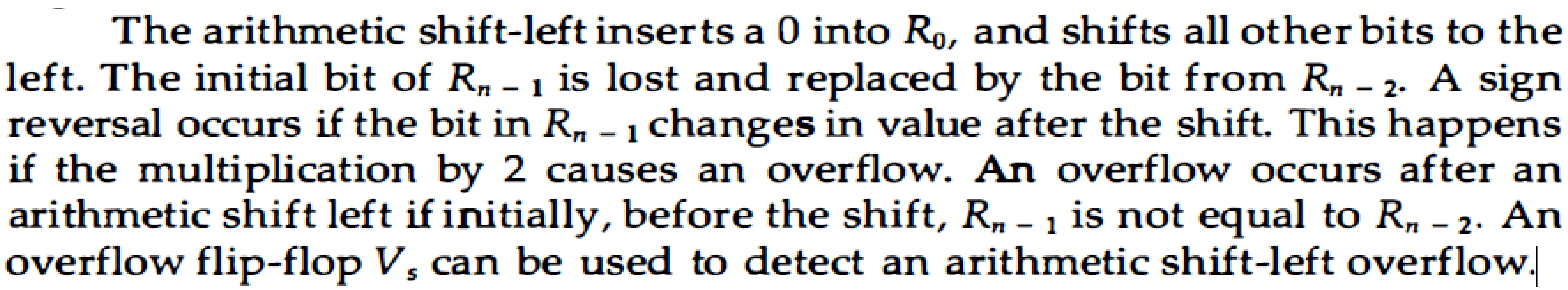
A ***Left Arithmetic Shift*** of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded.

Arithmetic left shift operations

4-6 Shift Microoperations
Arithmetic Shifts cont.



Arithmetic left shift operations



The arithmetic shift-left inserts a 0 into R_0, and shifts all other bits to the left. The initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2}. A sign reversal occurs if the bit in R_{n-1} changes in value after the shift. This happens if the multiplication by 2 causes an overflow. An overflow occurs after an arithmetic shift left if initially, before the shift, R_{n-1} is not equal to R_{n-2}. An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow.

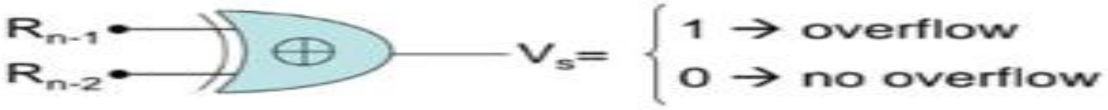
Arithmetic left shift operations

$$V_s = R_{n-1} \oplus R_{n-2}$$

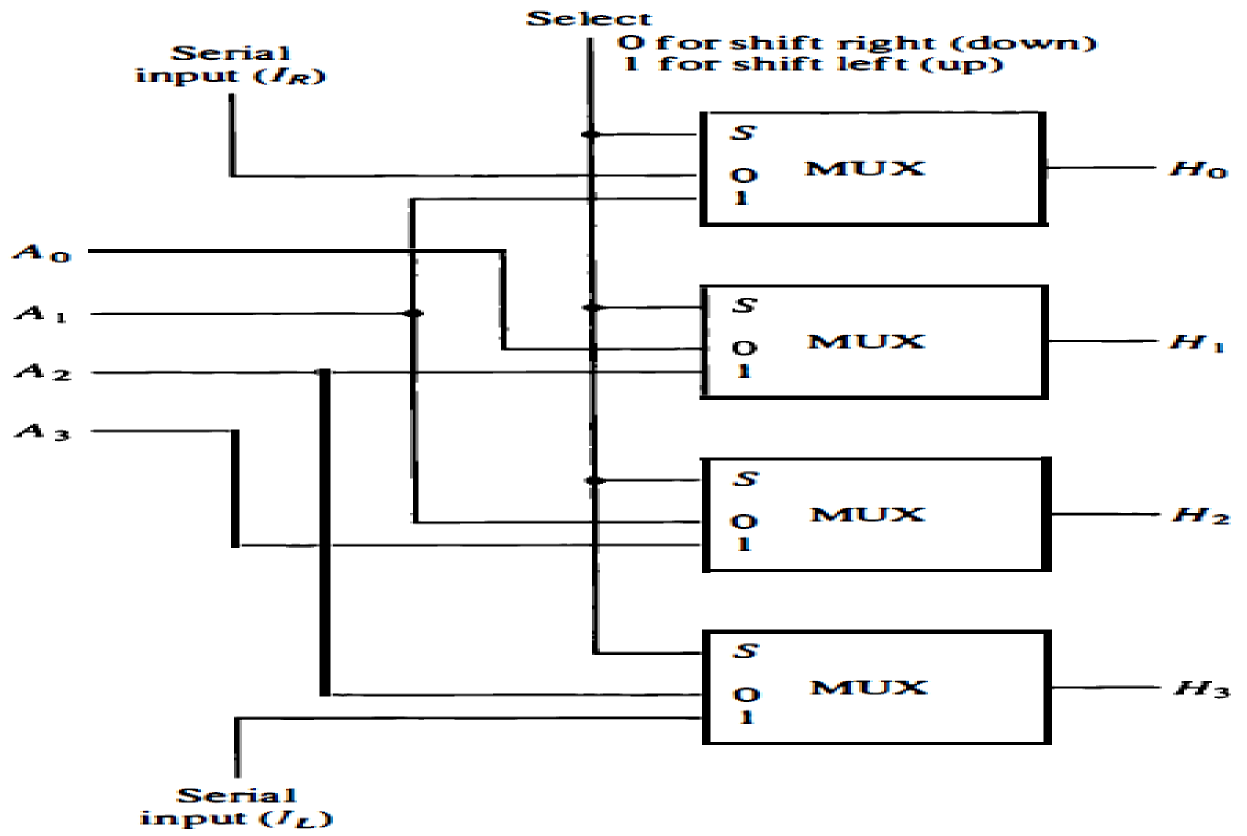
If $V_s = 0$, there is no overflow, but if $V_s = 1$, there is an overflow and a sign reversal after the shift. V_s must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

- An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow

$$V_s = R_{n-1} \oplus R_{n-2}$$



Hardware Implementation



Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Figure 4-12 4-bit combinational circuit shifter.

Hardware Implementation

A combinational circuit shifter can be constructed with multiplexers as shown in Fig. 4-12. The 4-bit shifter has four data inputs, A_0 through A_3, and four data outputs, H_0 through H_3. There are two serial inputs, one for shift left

(I_L) and the other for shift right (I_L). When the selection input $S = 0$, the input data are shifted right (down in the diagram). When $S = 1$, the input data are shifted left (up in the diagram). The function table in Fig. 4-12 shows which input goes to each output after the shift. A shifter with n data inputs and outputs requires n multiplexers. The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.

Arithmetic logic shift unit

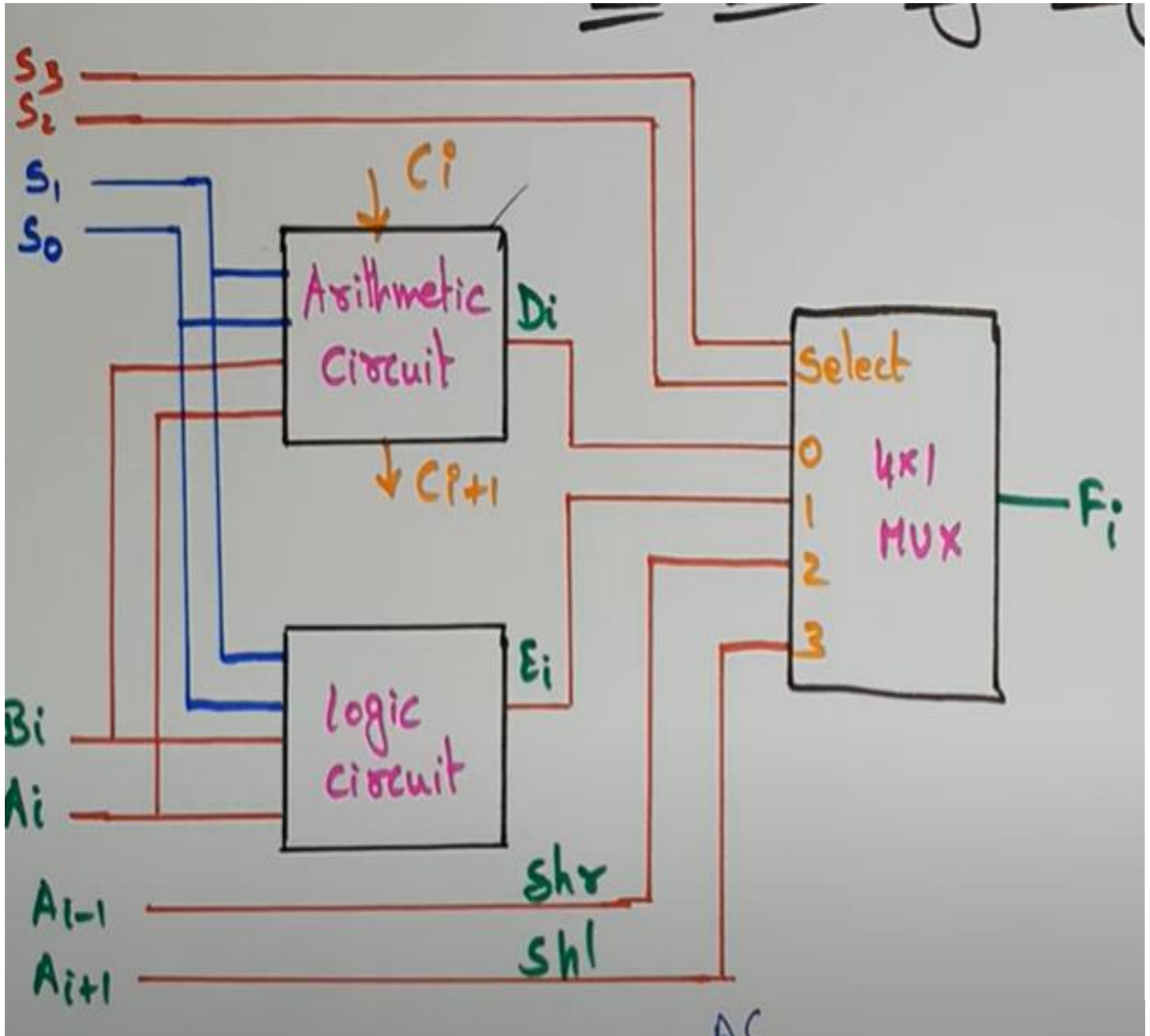
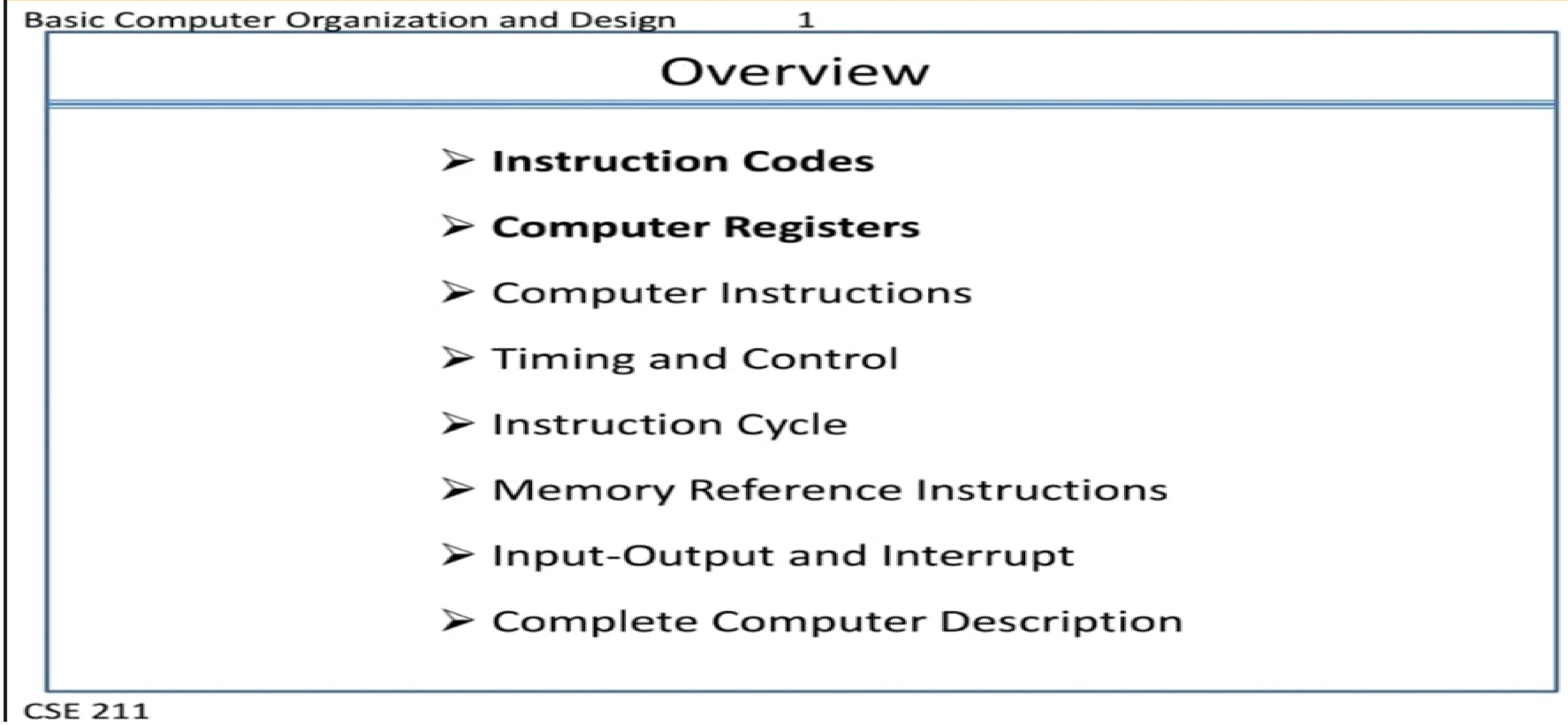


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F

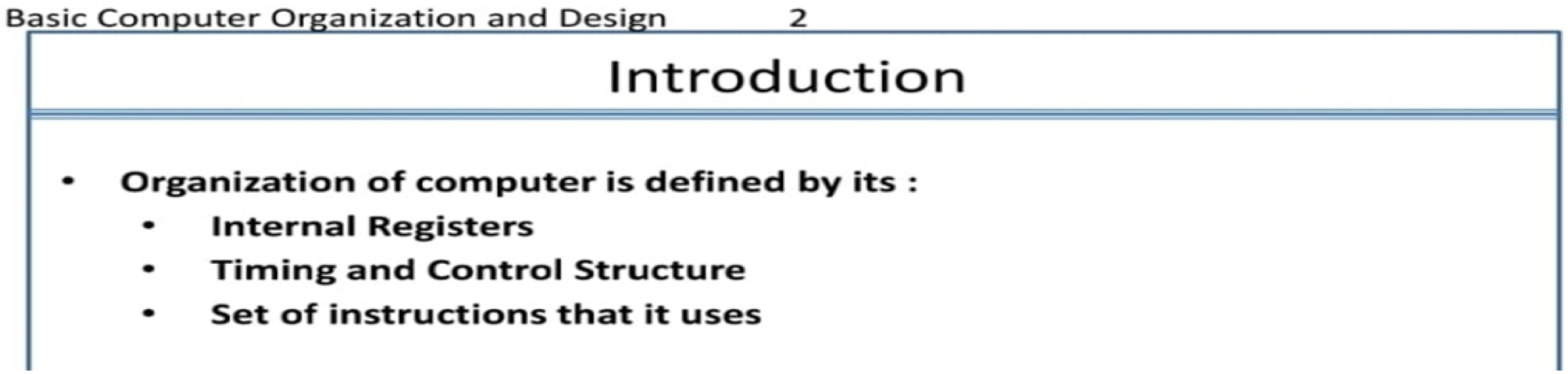
Computer Organization



Overview

- Instruction Codes
- Computer Registers
- Computer Instructions
- Timing and Control
- Instruction Cycle
- Memory Reference Instructions
- Input-Output and Interrupt
- Complete Computer Description

Computer Organization



Basic Computer Organization and Design 2

Introduction

- Organization of computer is defined by its :
  - Internal Registers
  - Timing and Control Structure
  - Set of instructions that it uses

The internal organization of a digital system is defined by the sequence of microoperations it performs on data stored in its registers

Computer Organization

The general purpose digital computer is capable of executing various micro-operations and, in addition., can be instructed as to what specific sequence of operations it must perform.

The user of a computer can control the process by means of a program.

A program is a set of instructions that specify the operations operands, and the sequence by which processing has to occur.

Computer Organization

The data processing task may be altered by specifying a new program with different instructions or specifying the same instructions with different data.

Computer Organization

Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register.

The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.

Every computer has its own unique instruction set. The ability to store and execute instructions, the stored program concept, is the most important property of a general-purpose computer.

Computer Organization

An instruction code is a group of bits that instruct the computer to perform a specific operation.

It is usually divided into parts, each having its own particular interpretation.

The most basic part of an instruction code is its operation part.

The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.

Computer Organization

At this point we must recognize the relationship between a computer operation and a microoperation.

An operation is part of an instruction stored in computer memory.

It is a binary code that tells the computer to perform a specific operation.

Computer Organization

The control unit receives the instruction from memory and interprets the operation code bits.

It then issues a sequence of control signals to initiate microoperations in internal computer registers.

For every operation code, the control issues a sequence of microoperations needed for the hardware implementation of the specified operation.

For this reason, an operation code is sometimes called a macro-operation because it specifies a set of microoperations.

Computer Organization

The operation part of an instruction code specifies the operation to be performed.

This operation must be performed on some data stored in processor registers or in memory.

An instruction code must therefore specify not only the operation but also the registers or the memory words where the operands are to be found, as well as the register or memory word where the result is to be stored.

Computer Organization

Memory words can be specified in instruction codes by their address.

Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2^k registers.

Instruction code formats are conceived by computer designers who specify the architecture of the computer.

Stored Program Organization

The simplest way to organize a computer is to have one processor register and an instruction code format with two parts.

The first part specifies the operation to be performed and the second specifies an address.

The memory address tells the control where to find an operand in memory.

This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.

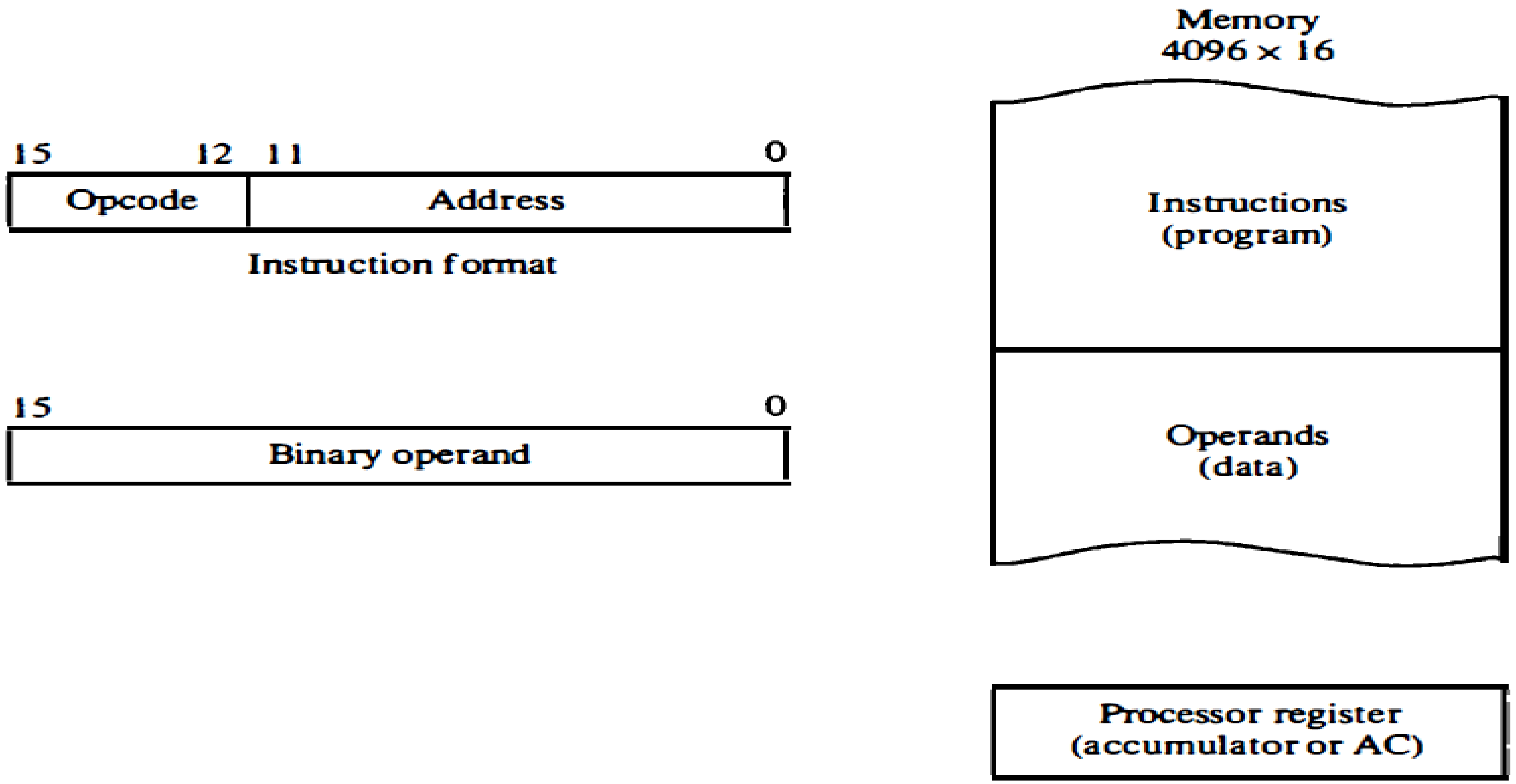
Stored Program Organization

Figure 5-1 depicts this type of organization. Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$.

If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.

Stored Program Organization

Figure 5-1 Stored program organization.



Stored Program Organization

Computers that have a single-processor register usually assign to it the name accumulator and label it AC.

The operation is performed with the memory operand and the content of AC.

If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes.

For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register. They do not need an operand from memory.

Tutorial

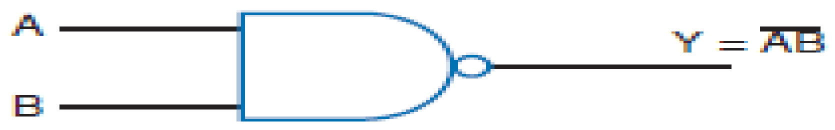
- What are universal gates ? Explain with truth table ?

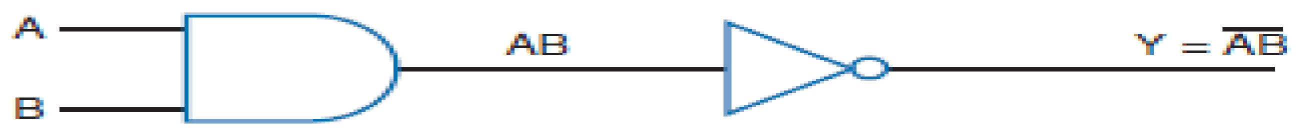
NAND GATE

1. A NAND gate is a combination of an inverter and an AND gate.

FIGURE 33–7

Logic symbol for an NAND gate.

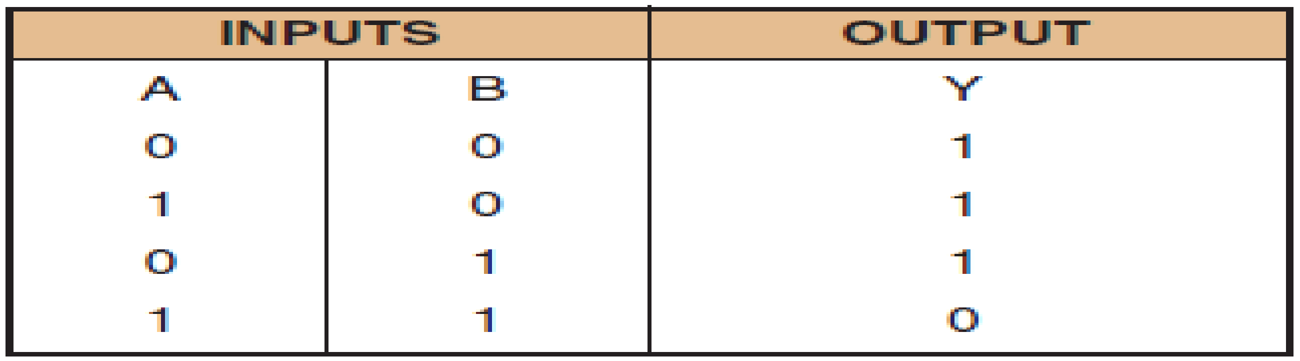




- The algebraic formula for NAND-gate output i$Y = \overline{AB}$,

FIGURE 33–8

Truth table for a two-input NAND gate.

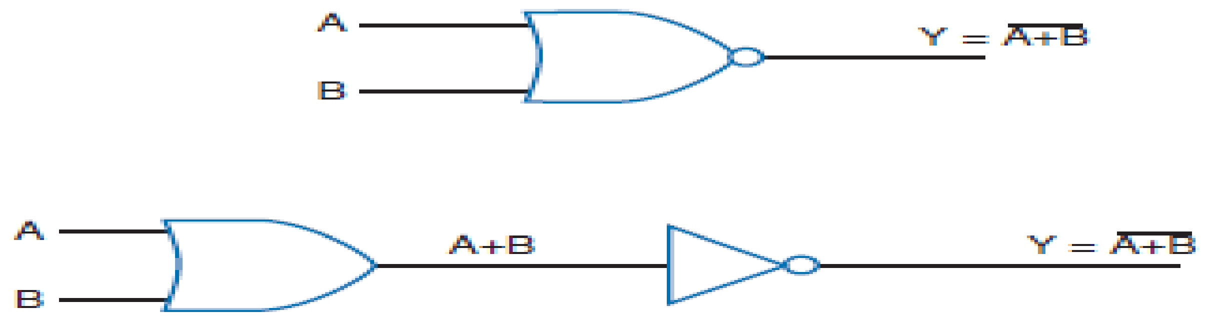


INPUTS		OUTPUT
A	B	Y
0	0	1
1	0	1
0	1	1
1	1	0

NOR GATE

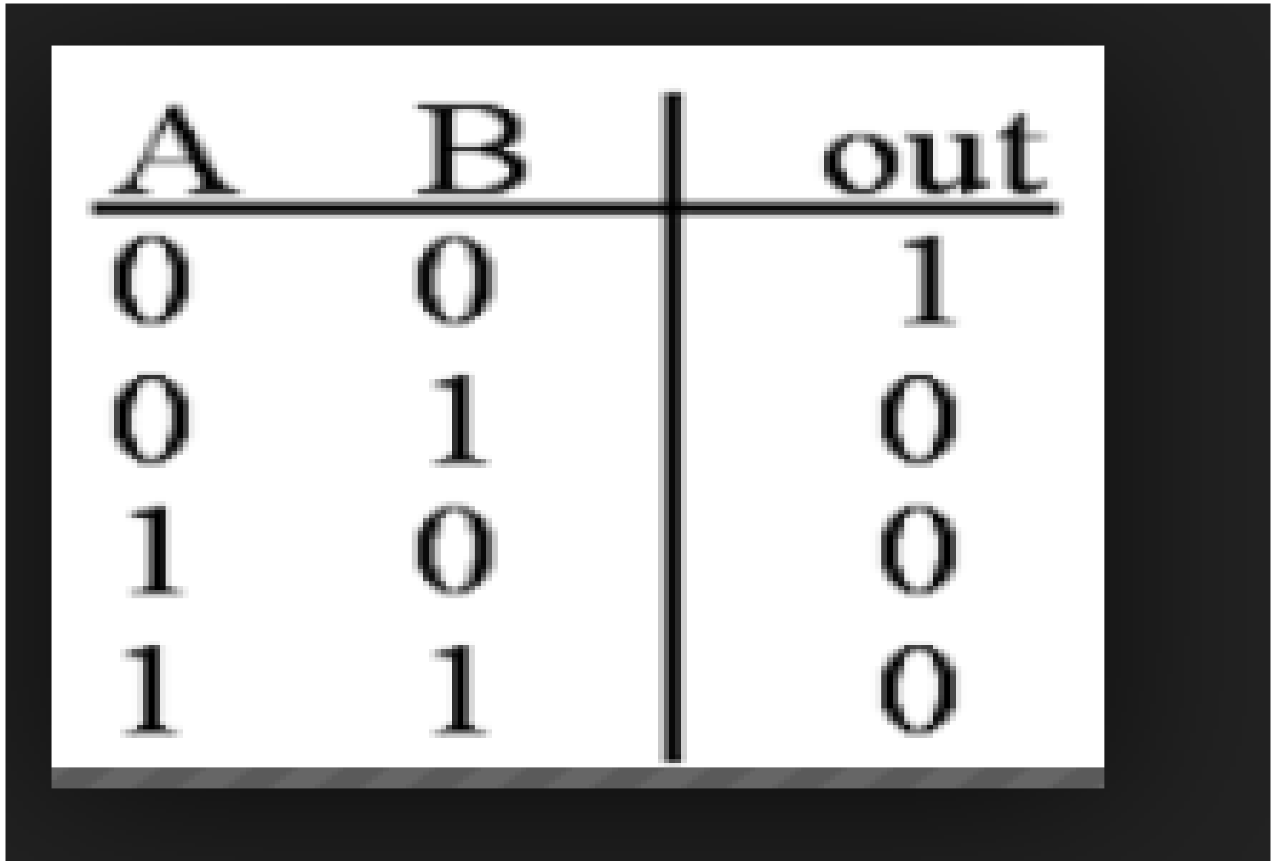
1. A **NOR gate is a combination of an inverter and an** OR gate. Its name derives from its NOT-OR function.
2. Also shown is its equivalency to an OR gate and an inverter.

FIGURE 33–10
Logic symbol for a NOR gate.



- The algebraic expression for NOR-gate output is

$$Y = \overline{A + B},$$

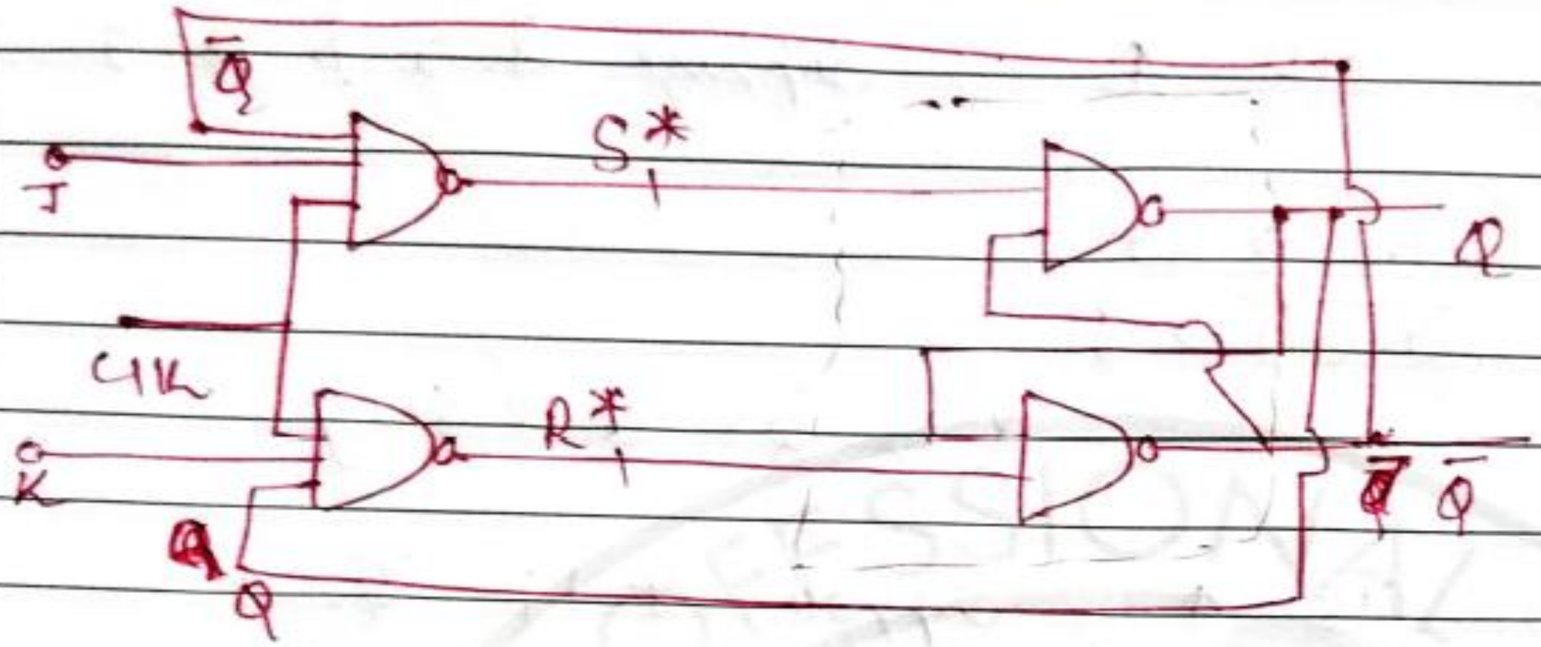


A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

Tutorial

- What is JK Flip flop ? Explain with truth table ?

J-K flip flop



SR NAND Latch

S^*	R^*	Q	$\bar{Q}$
0	0	Invalid	
0	1	1	0
1	0	0	1
1	1	NC/memory	

Case 1:- Clk $\rightarrow \uparrow$; J = 0 ; K = 0

$R^* = 1$ & $S^* = 1$

Case 2:- $Clk \rightarrow \uparrow$; $J=0$; $K=1$; $Q=0$; $\overline{Q}=1$ (Reset)

a) If suppose $Q=0$ & $\overline{Q}=1$

$$\left.\begin{array}{l} S^* = \overline{\overline{Q}.J.clk} = 1 \\ R^* = \overline{Q\cdot K.clk} \Rightarrow 1 \end{array}\right\} \text{NC state}$$

so $Q=0$ & $\overline{Q}=1$ (Reset)

(b) If suppose $Q=1$ & $\overline{Q}=0$

$S^* = \overline{0.0.1} \Rightarrow 1$

$R^* = \overline{1.1.1} \Rightarrow 0 \Rightarrow$ $Q=0$ & $\overline{Q}=1$ (Reset)

Case III :- $S^* = \overline{\bar{Q} \cdot J \cdot Clk}$

$R^* = \overline{Q \cdot K \cdot Clk}$

previous state

A) $J = 1$; $K = 0$; $Clk = \uparrow$; Suppose $Q = 0$ & $\bar{Q} = 1$

$S^* = \overline{1 \cdot 1 \cdot 1} \Rightarrow 0$

$R^* = \overline{0 \cdot 0 \cdot 1} \Rightarrow 1$

$Q = 1$; $\bar{Q} = 0$ { Set }

(B) $J = 1$; $K = 0$, $Clk = \uparrow$; Suppose $Q = 1$; $\bar{Q} = 0$

$S^* = \overline{0 \cdot 1 \cdot 1} \Rightarrow 1$ } NC (No change)

$R^* = \overline{1 \cdot 0 \cdot 1} \Rightarrow 1$ } means

$Q = 1$; $\bar{Q} = 0$ (Remain Same)

Case IV ; CLK = ↑ ; J=1, K=1

$S^* = \overline{\bar{Q} \cdot J \cdot CLK}$
$R^* = \overline{Q \cdot K \cdot CLK}$.

Previous state assume
$Q=1$; $\bar{Q}=0$

$S^* = \overline{0 \cdot 1 \cdot 1} \Rightarrow 1$ } $Q=0$ & $\bar{Q}=1$
$R^* = \overline{1 \cdot 1 \cdot 1} \Rightarrow 0$ } Reset

$Q_{n+1} = 0$ & $\bar{Q}_{n+1} = 1$

$Q_{n+1} = \bar{Q}_n$

$Q=0$ & $\bar{Q}=1$

$S^* = \bar{1} = 0$ } SET
$R^* = 1$ }

$Q=1$; $\bar{Q}=0$

$Q_{n+1} = 1$ ($\bar{Q}_n$ }
(Toggling)

Page N

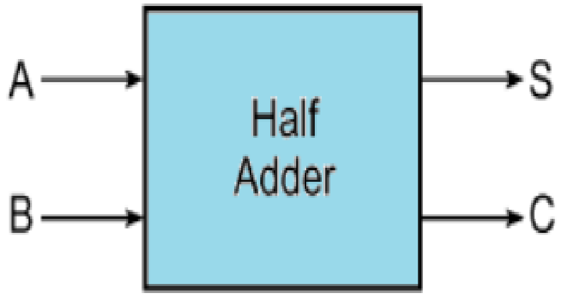
Clock	J	K	Q_{n+1}, O/P	$\bar{Q}_{n+1}$,	State
↑	0	0	NC	NC	[illegible]
↑	0	1	0	1	Reset
↑	1	0	1	0	Set
↑	1	1	$\bar{Q}_n$	Q_n	Toggle

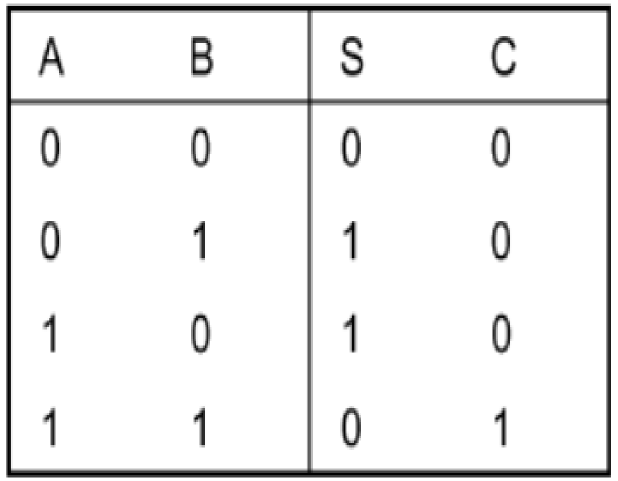
Thus as long as J=K=1; the o/p will keep toggling indefinitely. This multiple toggling in JK is called Race around condition.

Tutorial

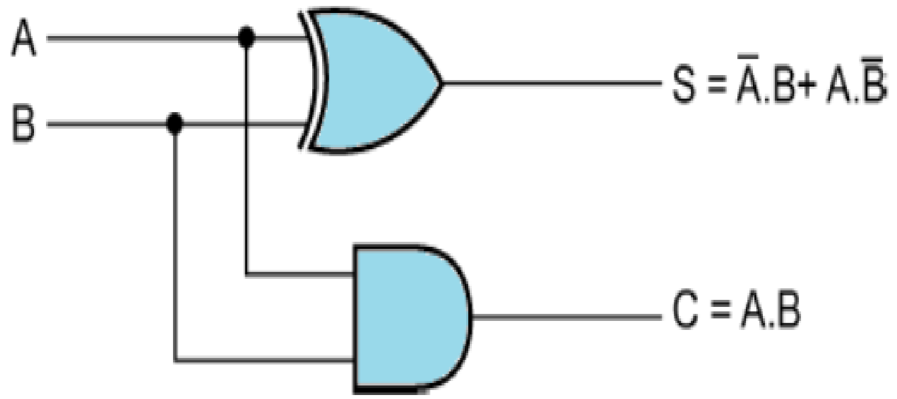
- What is the difference between half adder and Full adder ?

Adder





A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



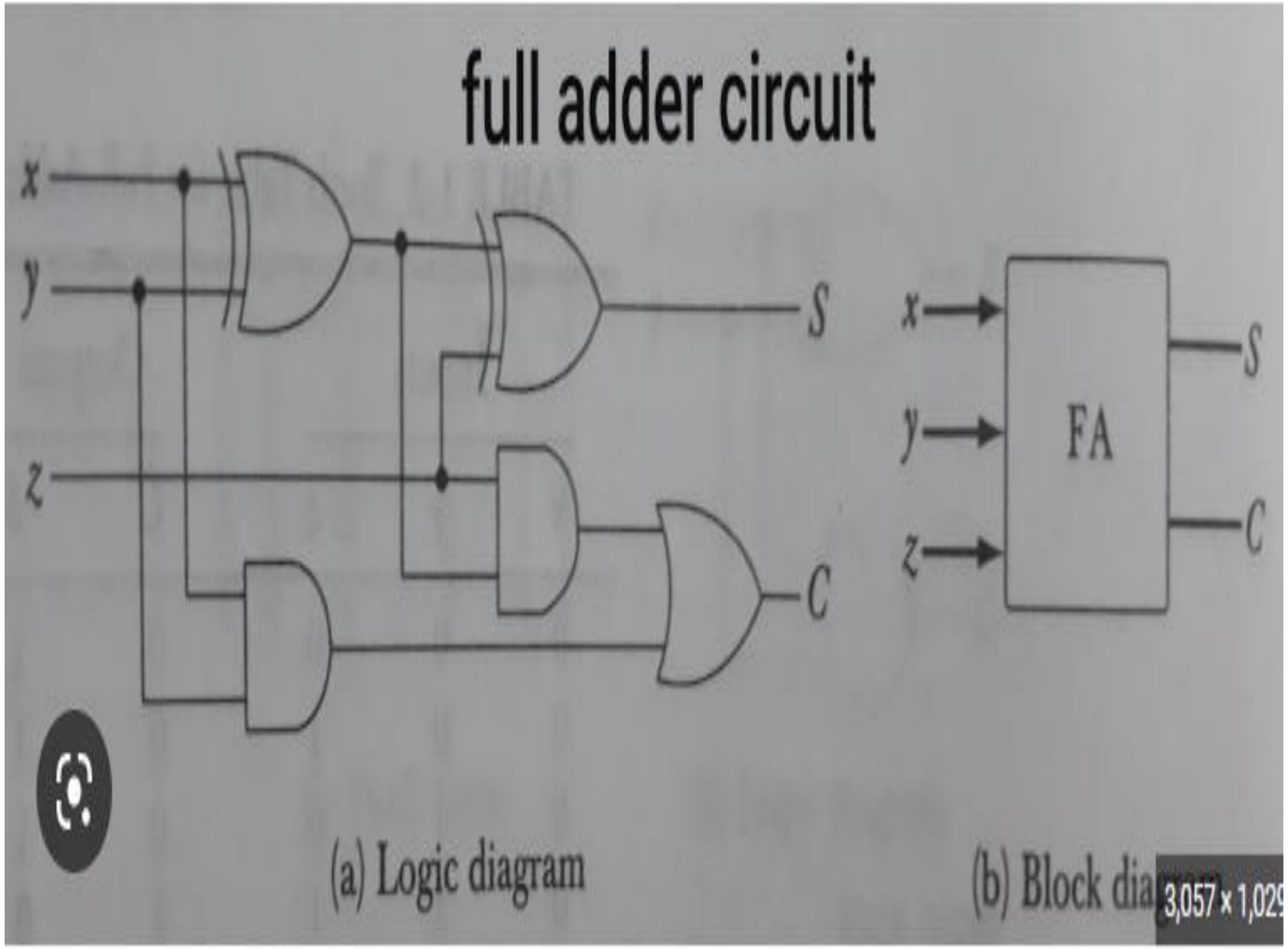
A half adder is a type of adder, an electronic circuit that performs the addition of numbers.

The half adder is able to add two single binary digits and provide the output plus a carry value.

It has two inputs, called A and B, and two outputs S (sum) and C (carry).

The common representation uses a XOR logic gate and an AND logic gate.

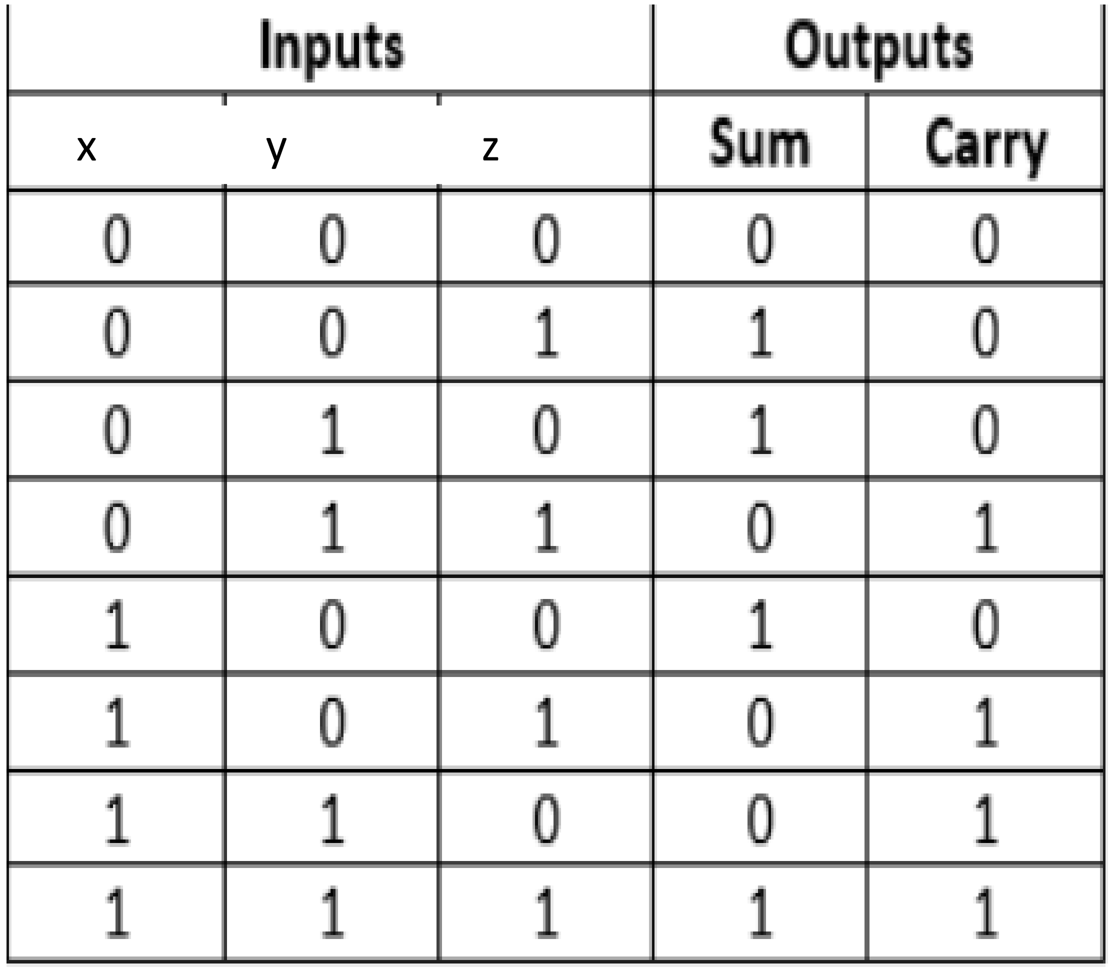
Adder



Sum:- x(xor)y(xor)z

Carry = xy+xz+yz

Adder



Inputs			Outputs	
x	y	z	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1