Computer Data Representation, Register Transfer and Micro-operations

Marwadi University

Diploma Studies
Computer Engineering

Unit – 1

Title - Computer Data Representation & Register Transfer and Micro-operations

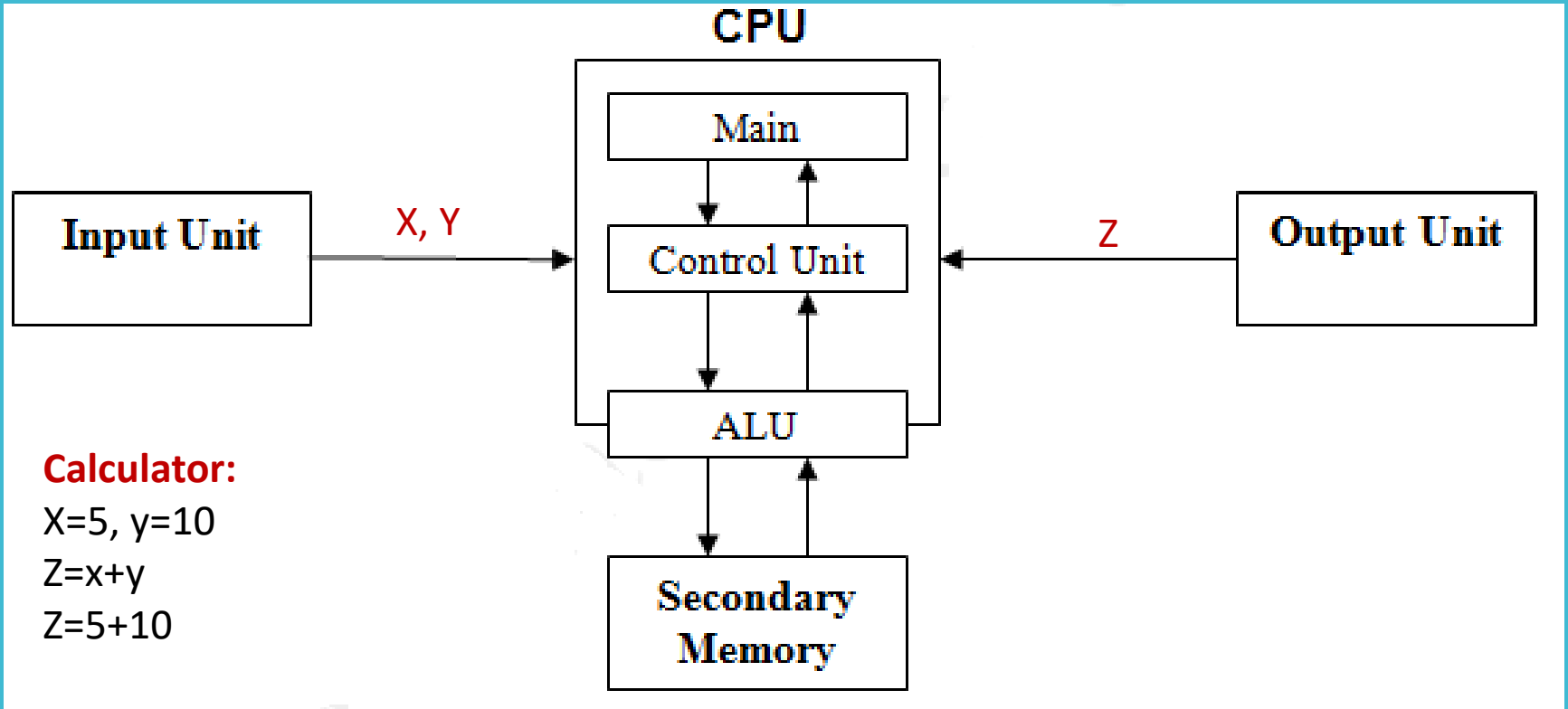
Computer Organization
(09CE2401)

Prof. Smit Thacker

❖ Computer Architecture

- Computer architecture is concerned with the structure and behavior of the various functional modules of the computer and how they interact to provide the processing needs of the user.

❖ Computer Architecture



Calculator:

X=5, y=10

Z=x+y

Z=5+10

Computer Organization

- Computer organization is concerned with the way the hardware components are connected together to form a computer system.

Data Representation

(1) Numeric data

--Numbers (real, Integer)

(2) Non-numeric data

--letters, symbols

(3) Data structures

--list, tree, stack etc...

Program and many more...

Numeric Data Representation

Decimal	Binary	Octal	Hexadecimal

Decimal

Decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) to represent numbers, and refer to the system as the decimal system.

The decimal system is said to have a base, or radix, of 10.

This means that each digit in the number is multiplied by 10 raised to a power corresponding to that digits position

Decimal

$$83 = (8 \times 10^1) + (3 \times 10^0)$$
$$4728 = (4 \times 10^3) + (7 \times 10^2) + (2 \times 10^1) + (8 \times 10^0)$$

The same principle holds for decimal fractions, but negative powers of 10 are used.

$$0.256 = (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

Decimal

A number with both an integer and fractional part has digits raised to both positive and negative powers of 10:

$$442.256 = (4 \times 10^2) + (4 + 10^1) + (2 \times 10^0) + (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

MSB LSB

4	7	2	2	5	6
100s	10s	1s	tenths	hundredths	thousandths
10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}
position 2	position 1	position 0	position –1	position –2	position –3

Binary

In the binary system, we have only two digits, 1 and 0. Thus, numbers in the binary system are represented to base 2.

$$10_2 = (1 \times 2^1) + (0 \times 2^0) = 2_{10}$$

$$11_2 = (1 \times 2^1) + (1 \times 2^0) = 3_{10}$$

$$100_2 = (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4_{10}$$

Octal

In the octal system (radix 8). The eight symbols are

0, 1, 2, 3, 4, 5, 6, and 7.

$$(736.4)_8 = 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1}$$

$$= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 = (478.5)_{10}$$

Hexadecimal

Hexadecimal (radix 16) , The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, 0, E, and F.

The last six symbols are, unfortunately, identical to the letters of the alphabet and can cause confusion at times.

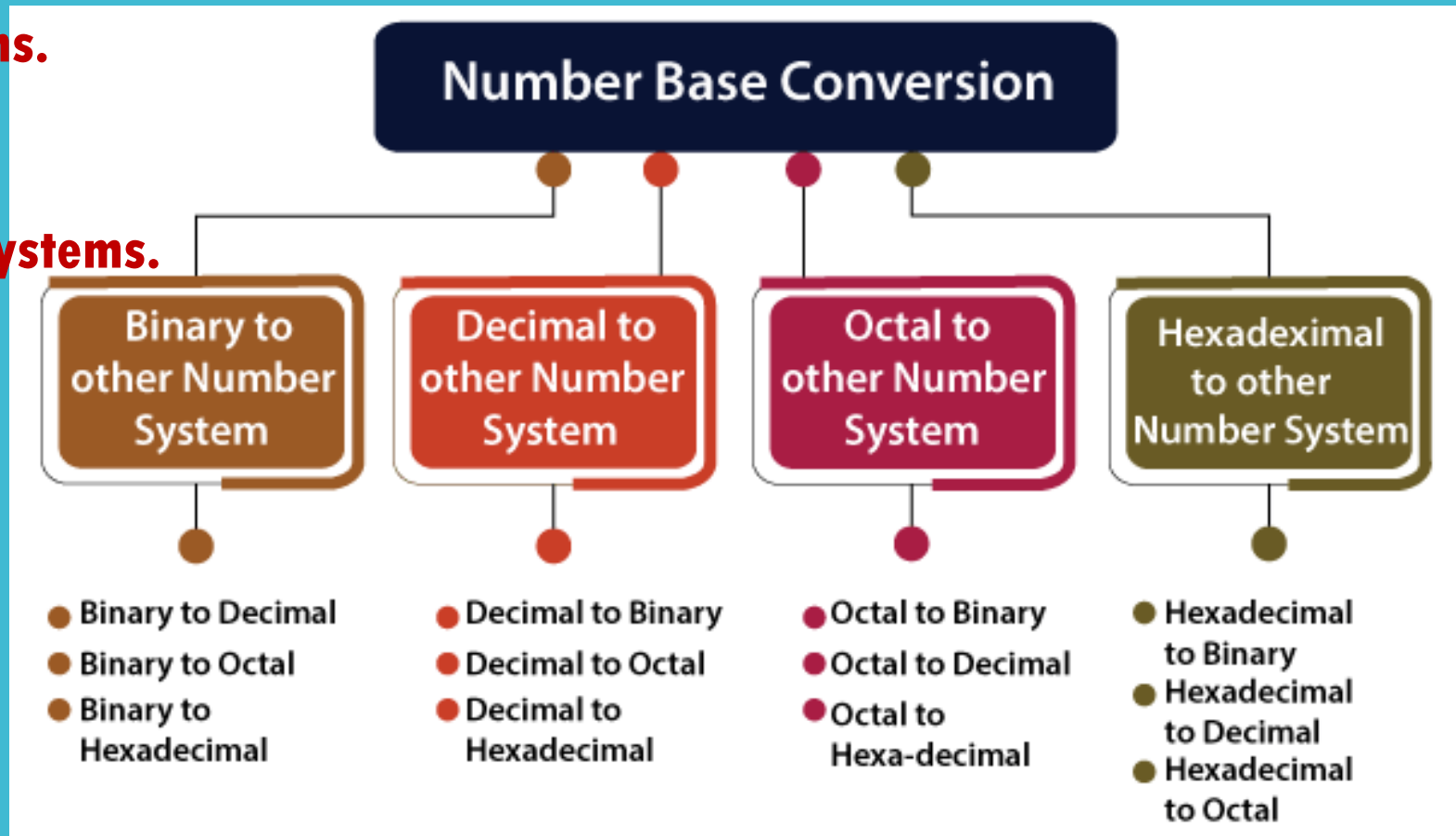
However, this is the convention that has been adopted.

$$(F3)_{h6} = F \times 16 + 3 = 15 \times 16 + 3 = (243)_{10}$$

Number Base Conversion

As, we have four types of number systems so each one can be converted into the remaining three systems. There are the following conversions possible in Number System

1) Binary to other Number Systems.
2) Decimal to other Number Systems.
3) Octal to other Number Systems.
4) Hexadecimal to other Number Systems.



Binary to other Number Systems

There are three conversions possible for binary number, i.e., binary to decimal, binary to octal, and binary to hexadecimal.

The conversion process of a binary number to decimal differs from the remaining others.

Let's take a detailed discussion on Binary Number System conversion.

Binary to Decimal Conversion

The process of converting binary to decimal is quite simple.

The process starts from multiplying the bits of binary number with its corresponding positional weights.

And lastly, we add all those products.

Example 1: (10110.001)2

We multiplied each bit of (10110.001)2 with its respective positional weight, and last we add the products of all the bits with its weight.

$(10110.001)2=(1\times2^4)+(0\times2^3)+(1\times2^2)+(1\times2^1)+(0\times2^0)+(0\times2^{-1})+(0\times2^{-2})+(1\times2^{-3})$

$(10110.001)2=(1\times16)+(0\times8)+(1\times4)+(1\times2)+(0\times1)+(0\times1/2)+(0\times1/4)+(1\times1/8)$

$(10110.001)2=16+0+4+2+0+0+0+0.125$

$(10110.001)2=(22.125)10$

Binary to Octal Conversion

- The base numbers of binary and octal are 2 and 8, respectively.
- In a binary number, the pair of three bits is equal to one octal digit.
- There are only two steps to convert a binary number into an octal number which are as follows:

1. In the first step, we have to make the pairs of three bits on both sides of the binary point. If there will be one or two bits left in a pair of three bits pair, we add the required number of zeros on extreme sides.
2. In the second step, we write the octal digits corresponding to each pair.

Binary to Octal Conversion - Example

- **Example 1: (1111101011011.0011)2**
- 1. Firstly, we make pairs of three bits on both sides of the binary point.

  111 110 101 011.001 1

- On the right side of the binary point, the last pair has only one bit.
- To make it a complete pair of three bits, we added two zeros on the extreme side.

  111 110 101 011.001 100

- 2. Then, we wrote the octal digits, which correspond to each pair.

  (1111101011011.0011)2=(7653.14)8

Binary to Hexadecimal Conversion

- The base numbers of binary and hexadecimal are 2 and 16, respectively.
- In a binary number, the pair of four bits is equal to one hexadecimal digit.
- There are also only two steps to convert a binary number into a hexadecimal number which are as follows:

1. In the first step, we have to make the pairs of four bits on both sides of the binary point.

   If there will be one, two, or three bits left in a pair of four bits pair, we add the required number of zeros on extreme sides.

2. In the second step, we write the hexadecimal digits corresponding to each pair.

Binary to Hexadecimal Conversion - Example

- **Example 1: (101101011011.0011)2**

- 1. Firstly, we make pairs of four bits on both sides of the binary point.

  111 1010 1011.0011

On the left side of the binary point, the first pair has three bits. To make it a complete pair of four bits, add one zero on the extreme side.

  0111 1010 1011.0011

2. Then, we write the hexadecimal digits, which correspond to each pair.

  (011110101011.0011)2=(7AB.3)16

Decimal to other Number System

Decimal to other Number System

- The decimal number can be an integer or floating-point integer.
- When the decimal number is a floating-point integer, then we convert both part (integer and fractional) of the decimal number in the isolated form(individually).
- There are the following steps that are used to convert the decimal number into a similar number of any base 'r'.

1. In the first step, we perform the division operation on integer and successive part with base 'r'. We will list down all the remainders till the quotient is zero. Then we find out the remainders in reverse order for getting the integer part of the equivalent number of base 'r'. In this, the least and most significant digits are denoted by the first and the last remainders.

2. In the next step, the multiplication operation is done with base 'r' of the fractional and successive fraction. The carries are noted until the result is zero or when the required number of the equivalent digit is obtained. For getting the fractional part of the equivalent number of base 'r', the normal sequence of carrying is considered.

Decimal to Binary Conversion

- For converting decimal to binary, there are two steps required to perform, which are as follows:

1. In the first step, we perform the division operation on the integer and the successive quotient with the base of binary(2).

2. Next, we perform the multiplication on the integer and the successive quotient with the base of binary(2).

Decimal to Binary Conversion - Example

Example 1: (152.25)10

Step 1:

- **Divide the number 152 and its successive quotients with base 2.**

Operation	Quotient	Remainder
152/2	76	0 (LSB)
76/2	38	0
38/2	19	0
19/2	9	1
9/2	4	1
4/2	2	0
2/2	1	0
1/2	0	1(MSB)

- **(152)10=(10011000)2**

Example 1: (152.25)10

Step 2:

- **Now, perform the multiplication of 0.27 and successive fraction with base 2.**

Operation	Result	carry
0.25×2	0.50	0
0.50×2	0	1

- **(0.25)10=(.01)2**

Decimal to Octal Conversion

For converting decimal to octal, there are two steps required to perform, which are as follows:

1. In the first step, we perform the division operation on the integer and the successive quotient with the base of octal(8).

2. Next, we perform the multiplication on the integer and the successive quotient with the base of octal(8).

Octal to other Number System

Octal to other Number System

- Like binary and decimal, the octal number can also be converted into other number systems.
- The process of converting octal to decimal differs from the remaining one.
- Let's start understanding how conversion is done.

Octal to Decimal Conversion

- The process of converting octal to decimal is the same as binary to decimal.
- The process starts from multiplying the digits of octal numbers with its corresponding positional weights. And lastly, we add all those products.
- Let's take an example to understand how the conversion is done from octal to decimal.

Example 1: (152.25)8

Step 1: We multiply each digit of 152.25 with its respective positional weight, and last we add the products of all the bits with its weight.

$(152.25)8=(1\times8^{2})+(5\times8^{1})+(2\times8^{0})+(2\times8^{-1})+(5\times8^{-2})$

$(152.25)8=64+40+2+(2\times1/8)+(5\times1/64)$

$(152.25)8=64+40+2+0.25+0.078125$

$(152.25)8=106.328125$

So, the decimal number of the octal number 152.25 is **106.328125**

Octal to Binary Conversion

- The process of converting octal to binary is the reverse process of binary to octal. We write the three bits binary code of each octal number digit.
- Example 1: (152.25)8
- We write the three-bit binary digit for 1, 5, 2, and 5.
- (152.25)8=(001101010.010101)2

- So, the binary number of the octal number 152.25 is (001101010.010101)2

Octal to hexadecimal conversion

- For converting octal to hexadecimal, there are two steps required to perform, which are as follows:

1. In the first step, we will find the binary equivalent of number.
2. Next, we have to make the pairs of four bits on both sides of the binary point.

If there will be one, two, or three bits left in a pair of four bits pair, we add the required number of zeros on extreme sides and write the hexadecimal digits corresponding to each pair.

Octal to hexadecimal conversion - Example

Example 1: (152.25)8

Step 1: We write the three-bit binary digit for 1, 5, 2, and 5.

(152.25)8=(001101010.010101)2

- So, the binary number of the octal number 152.25 is (001101010.010101)2

Step 2:

1. Now, we make pairs of four bits on both sides of the binary point.

0 0110 1010.0101 01

Octal to hexadecimal conversion - Example

- On the left side of the binary point, the first pair has only one digit, and on the right side, the last pair has only two-digit.
- To make them complete pairs of four bits, add zeros on extreme sides.

0000 0110 1010.0101 0100

2. Now, we write the hexadecimal digits, which correspond to each pair.

(0000 0110 1010.0101 0100)2=(6A.54)16

Hexa-decimal to other Number System

Hexa-decimal to Decimal Conversion

- The process of converting hexadecimal to decimal is the same as binary to decimal.
- The process starts from multiplying the digits of hexadecimal numbers with its corresponding positional weights. And lastly, we add all those products.

Hexa-decimal to Decimal Conversion - Example

- **Example 1: (152A.25)16**
- **Step 1:We multiply each digit of 152A.25 with its respective positional weight, and last we add the products of all the bits with its weight.**

- $(152A.25)16=(1\times16^{3})+(5\times16^{2})+(2\times16^{1})+(A\times16^{0})+(2\times16^{-1})+(5\times16^{-2})$
- (152A.25)16=(1×4096)+(5×256)+(2×16)+(10×1)+(2×16-1)+(5×16-2)
- (152A.25)16=4096+1280+32+10+(2×1⁄16)+(5×1⁄256)
- (152A.25)16=5418+0.125+0.125
- (152A.25)16=5418.14453125

- **So, the decimal number of the hexadecimal number 152A.25 is 5418.14453125**

Hexadecimal to Binary Conversion

- The process of converting hexadecimal to binary is the reverse process of binary to hexadecimal.
- We write the four bits binary code of each hexadecimal number digit.

Example 1: (152A.25)16

- We write the four-bit binary digit for 1, 5, A, 2, and 5.
- (152A.25)16=(0001 0101 0010 1010.0010 0101)2

Hexadecimal to Octal Conversion

- For converting hexadecimal to octal, there are two steps required to perform, which are as follows:

1. In the first step, we will find the binary equivalent of the hexadecimal number.

2. Next, we have to make the pairs of three bits on both sides of the binary point. If there will be one or two bits left in a pair of three bits pair, we add the required number of zeros on extreme sides and write the octal digits corresponding to each pair.

Hexadecimal to Octal Conversion - Example

Example 1: (152A.25)16

Step 1: We write the four-bit binary digit for 1, 5, 2, A, and 5.

- (152A.25)16=(0001 0101 0010 1010.0010 0101)2
- So, the binary number of hexadecimal number 152A.25 is (0011010101010.010101)2

Step 2: Then, we make pairs of three bits on both sides of the binary point.

- 001 010 100 101 010.001 001 010

Then, we write the octal digit, which corresponds to each pair.

- (0010101001010.0010010)2=(12452.112)8

1's Complements

- **Note:- 1's and 2's complement can easily determined on Binary Number Only.**

(1) $(45)_{10} = (101101)_2$

1's Complement:-

To find 1's Complement, Just reverse the bits of Binary Value (0 -> 1, 1-> 0).

Binary	**1 0 1 1 0 1**
1's Complement	**0 1 0 0 1 0**

2's Complements

- To Find 2's complement, Just add 1 in the 1's Complement.

(1) $(45)_{10} = (101101)_{2}$

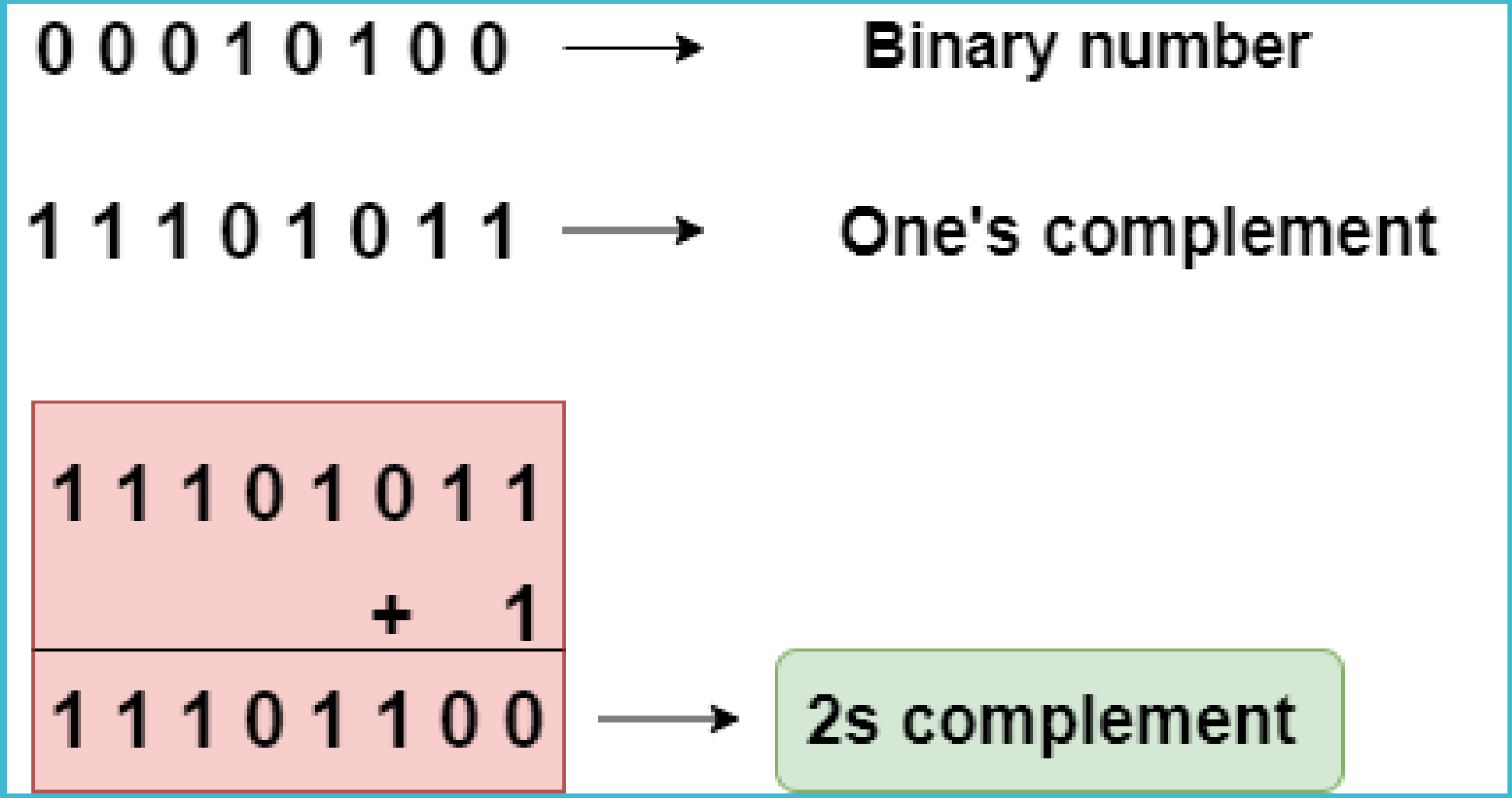
1's Complement:-

To find 1's Complement, Just reverse the bits of Binary Value (0 -> 1, 1-> 0).

Binary	1 0 1 1 0 1
1's Complement	0 1 0 0 1 0
	1

2's Complement	0 1 0 0 1 1

2’s complement



2's Complements

- Directly Find 2's Complement form Binary Number
- From the Right to Left of Binary Number, find the 1st 1, and write the same Binary Value.
- After the First 1, Reverse the remaining Bits of Binary Value.
- Examples:
- (1) (101010)
- (2) (10100)
- (3) (100101)

Complement No. System

N's Complement
(Radix/Base)

(nk-1)

(n-1)'s Complement
(Diminished Radix/Base)

(nk-1)-x

N = Radix / Base
X =Given Number
K – Total Digits in x

- **To Find (n-1)'s Complement of any given base, Find the highest Number of that base .**
- **For Ex. (1) $(\_\ \_\ \_)_6$ -> (5 5 5)**
- **(2) $(\_\ \_\ \_)_{10}$ -> (9 9 9)**

Complement No. System

- Example 1:- $(123)_8$ <- Given Number,
- Base/Radix is 8
- Max Number is 7
- So, 7 7 7 <- Radix Number
- - 1 2 3 <- Given Number

---

6 5 4 <- (n-1)'s Complement [7's Complement]

\+ 1

---

6 5 5 <- (n)'s Complement [8's Complement]

Complement No. System

- Example 2:- $(123)_{16}$ <- Given Number,
- Base/Radix is 16
- Max Number is 15
- So, 15 15 15 -> (F F F) <- Radix Number
- - 1 2 3 <- Given Number

E D C <- (n-1)'s Complement [15's Complement]

\+ 1

E D D <- (n)'s Complement [16's Complement]

Register Transfer Language

- The digital system modules are constructed from such digital components as **register**, decoders, arithmetic elements, and control logic.
- Register **holds some data** and **operations are performed** on them (micro-operations like clear, load, shift etc..)
- The **symbolic notation** used to describe the **micro-operation transfers** among registers is called a register transfer language.

Register Transfer Language

The term "register transfer" implies the availability of hardware logic circuits that can perform a stated micro operation, The word "language" is borrowed from programmers

Register Transfer

Register is a very fast computer memory, used to store data/instruction in execution.

A **Register** is a group of **flip-flops** with each flip-flop capable of storing **one bit** of information.

An **n-bit register** has a group of **n flip-flops** and is capable of storing binary information of **n-bits.**

Register Transfer

The registers used by the CPU are often termed as **Processor Registers.**

A **Processor Register** may hold an instruction, a storage address, or any data.

The computer needs processor registers for **manipulating data** and a register **holding a memory address.**

The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed

Some commonly used Register

(1) Accumulator: This is the most common register, used to store data taken out from the memory.

(2) General Purpose Registers: This is used to store data intermediate results during program execution. It can be accessed via assembly programming.

Some commonly used Register

(3) Special Purpose Registers:

Users do no access these registers. These registers are for computer system.

I. MAR: Memory Address Register are those registers that holds the address for memory unit.
II. MBR: Memory Buffer Register stores instruction and data received from the memory and sent from the memory.
III. PC: Program Counter points to the next instruction to be executed.
IV. IR: Instruction Register holds the instruction to be executed.

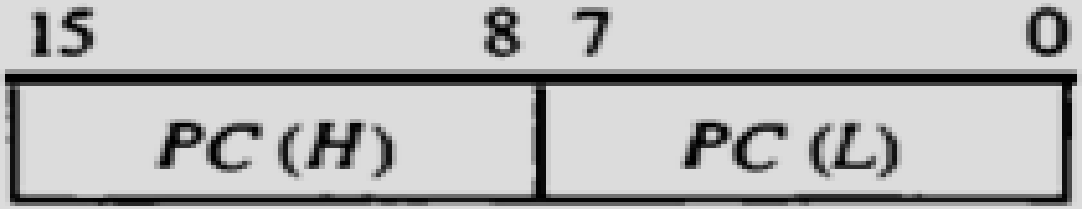
Register Transfer

The term Register Transfer means the availability of Hardware Logic Circuits that can perform a stated micro-operation and transfer the result of the operation to the same or another register.

(a) Register *R*

(b) Showing individual bits

(c) Numbering of bits



(d) Divided into two parts

L = Low Bit H = High Bit

Register Transfer

Register and Memory configuration for a basic computer

Register	Symbol	No. of Bits	Function
Data Register	DR	16	Holds memory operand
Accumulator	AC	16	Processor Register
Program Counter	PC	12	Holds Address of the Instruction
Address Register	AR	12	Holds Address for the memory
Instruction Register	IR	16	Holds instruction code
Temporary Register	TR	16	Holds temporary data
Input Register	INPR	8	Carries Input character
Output Register	OUTR	8	Carries output character

Register Transfer

Basic Symbols used for Register Transfer

Symbol	Description	Example
Letters and Numbers	Denotes a Register	MAR, R1, R2
( )	Denotes a part of register	R1(8-bit) R1(0-7)
<-	Denotes a transfer of information	R2 <- R1
,	Separate two micro-operations	R1<-R2, R2<-R1
:	Denotes conditional operations	P : R2 <- R1 if P=1
Naming Operator (:=)	Denotes another name for an already existing register/alias	Ra := R1

Register Transfer

Destination Source

$$R2 \leftarrow R1$$

a transfer of the content of register R1 into R2.

And a replacement of the content of R2 by the content of R1.

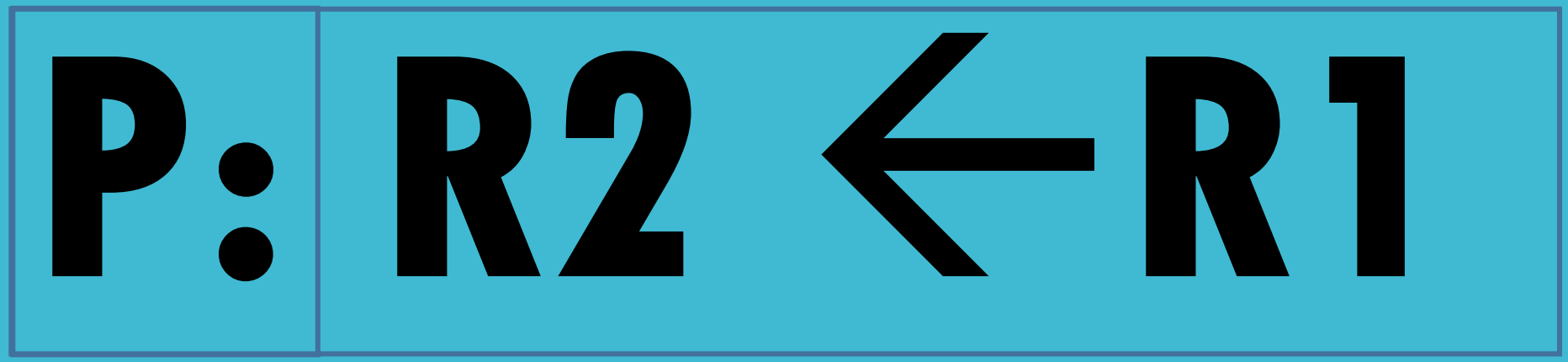
The content of the source register R1 does not change after the transfer

Register Transfer (predetermined condition)

$$\text{If } (P = 1) \text{ then } (R2 \leftarrow R1)$$

transfer to occur only under a predetermined condition.

Register Transfer (Control function)



- A control function (terminated with colon) is a Boolean variable that is equal to 1 or 0.
- It symbolizes the requirement that the transfer operation be executed by the hardware only if P = 1

Register Transfer (Control function)

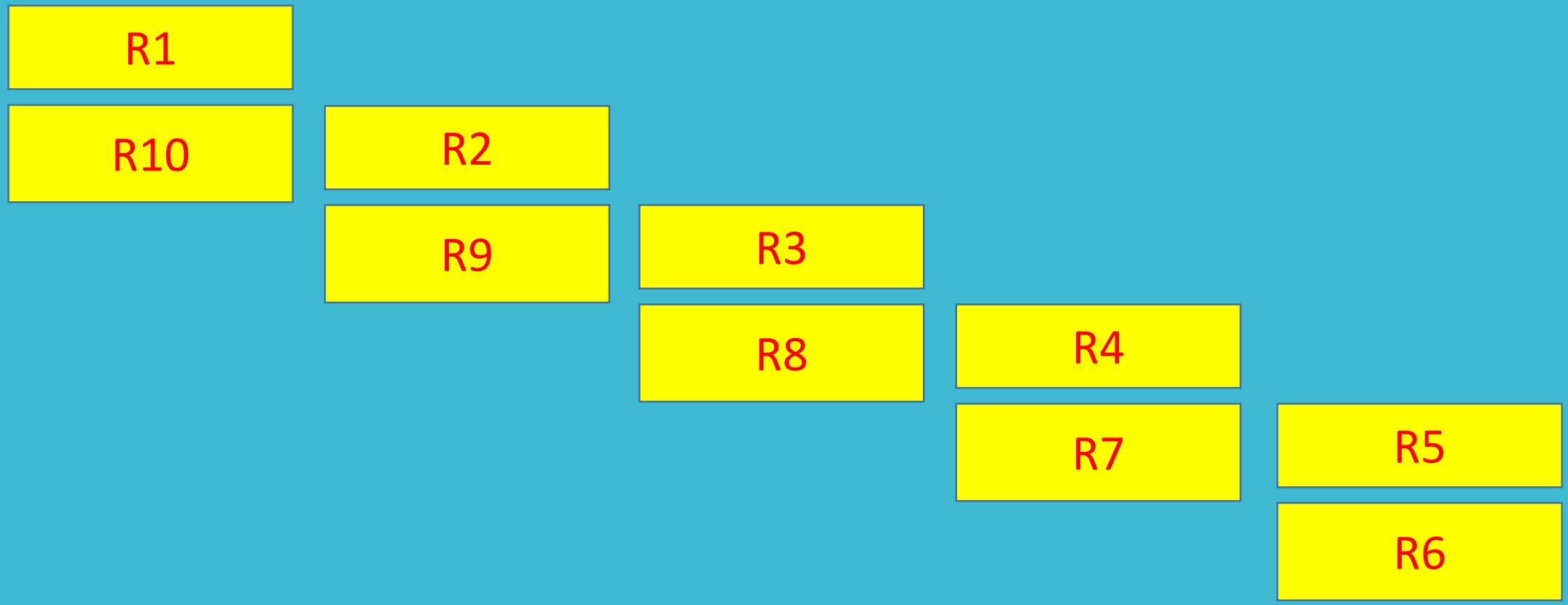
$$T: R2 \leftarrow R1, R1 \leftarrow R2$$

The arrow denotes a transfer of information and the direction of transfer.

A comma is used to separate two or more operations that are executed at the same time

Bus and Memory Transfers

Digital computers has many registers.



Bus and Memory Transfers

- Digital computers has many registers.
- Must have path for information transfer, one to another
- Number of wires will be excessive if separate wires used.
- **Solution** –

  Using common bus system

Bus and Memory Transfers

- **Common Bus System** - for transferring information between registers in a multiple-register configuration is a common bus system.
- It consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.

Figure 4-3 Bus system for four registers.

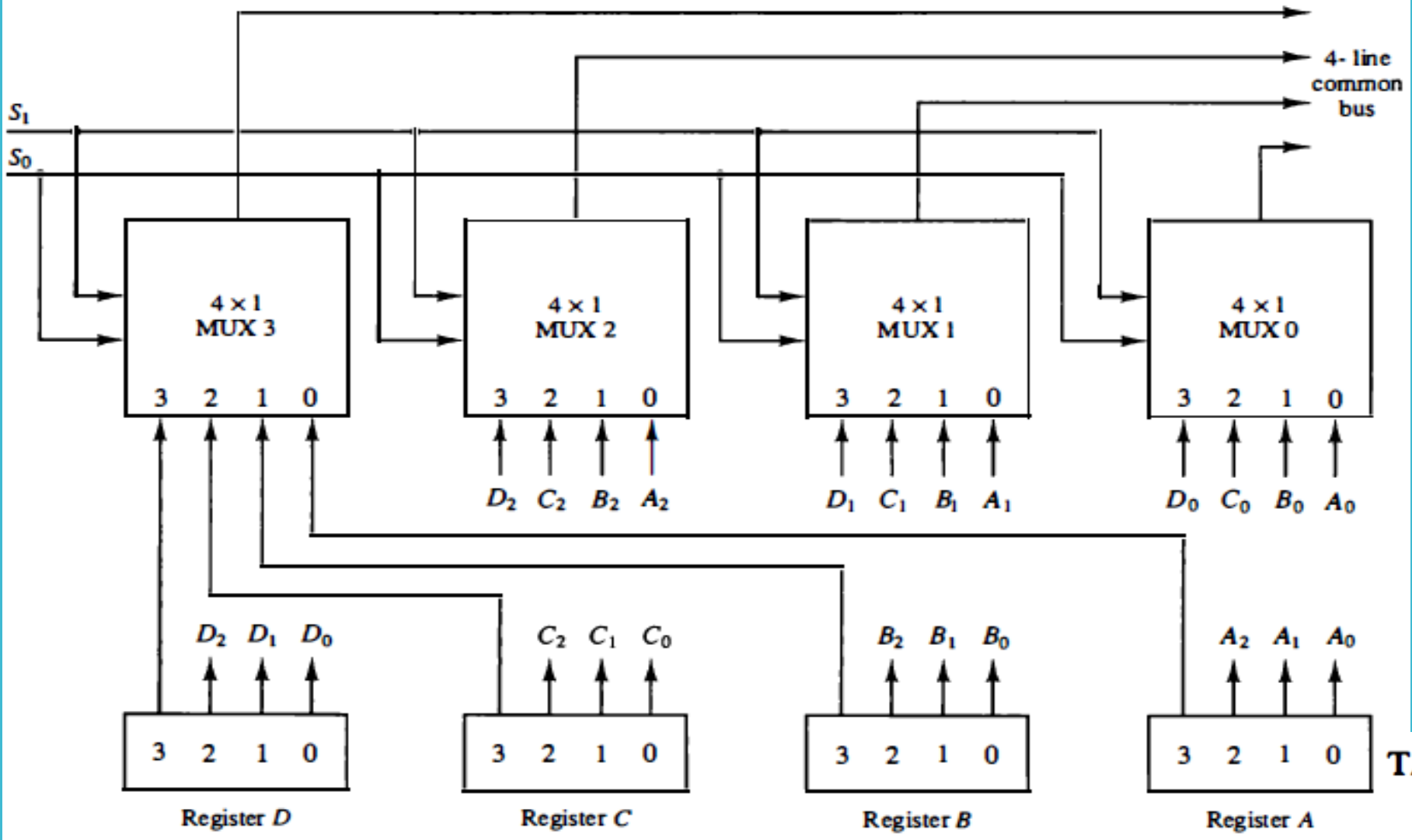


TABLE 4-2 Function Table for Bus of Fig. 4

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Common Bus System

Bus and Memory Transfers

- **Common Bus System** - For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus.
- Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

Tri-State Bus Buffers (3 State Buffer)

- A bus system constructed with three-state gates instead of multiplexers, a digital circuit that exhibits three states.
- Two of the states equivalent to logic 1 and 0 as in a conventional gate. The third state is a high-impedance state.
- The high-impedance state behaves like an open circuit, means that the output is disconnected and does not have a logic significance. commonly used in the design of a bus system is the buffer gate.
- Buffer Gate is used when we wants to add some delay in our output

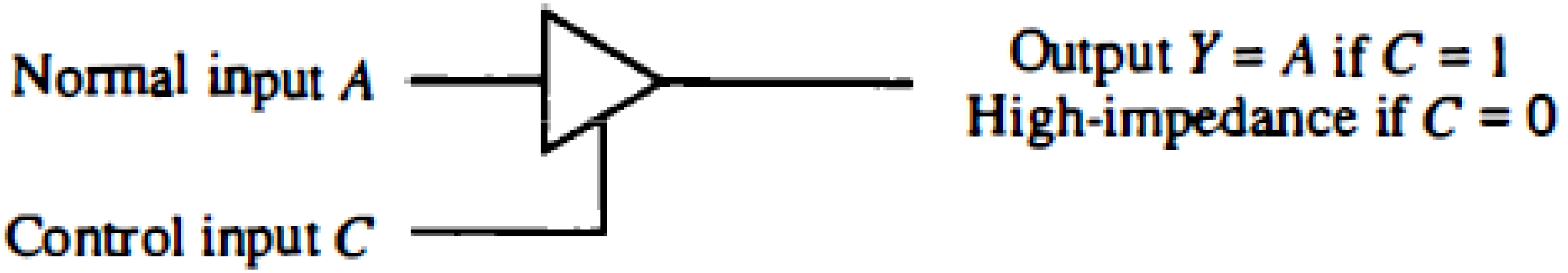
Tri-State Bus Buffers (3 State Buffer)

Definition:

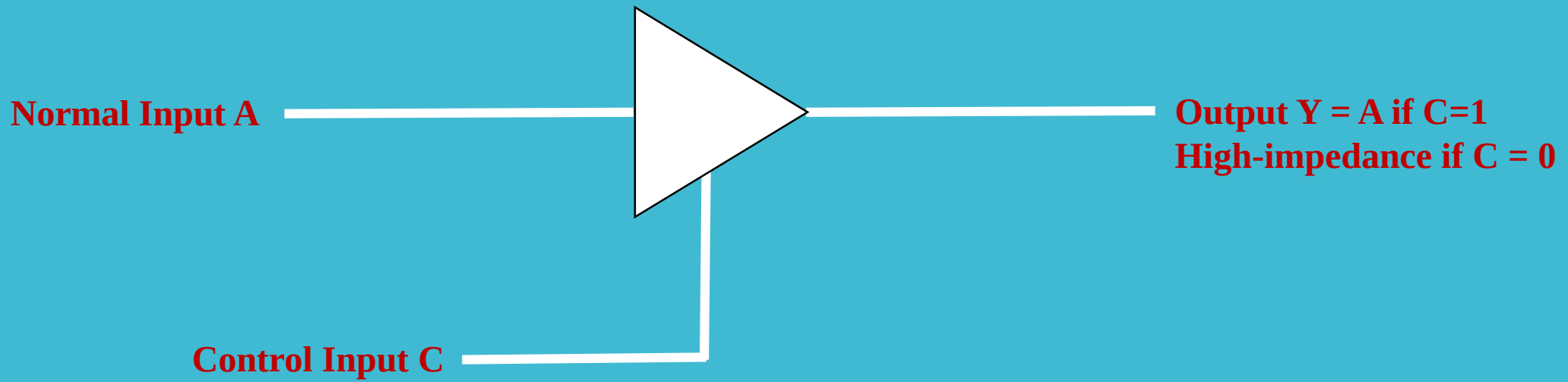
- A three-state bus buffer is an integrated circuit that connects multiple data sources to a single bus. The open drivers can be selected to be either a logical high, a logical low, or high impedance which allows other buffers to drive the bus.
- Now, let's see the more detailed analysis of a 3-state bus buffer in points:
  1) As in a conventional gate, 1 and 0 are two states.
  2) The third state is a high impedance state.
  3) The third state behaves like an open circuit.
  4) If the output is not connected, then there is no logical significance.
  5) It may perform any type of conventional logic operations such as AND, OR, NAND, etc.

Tri-State Bus Buffers (3 State Buffer)

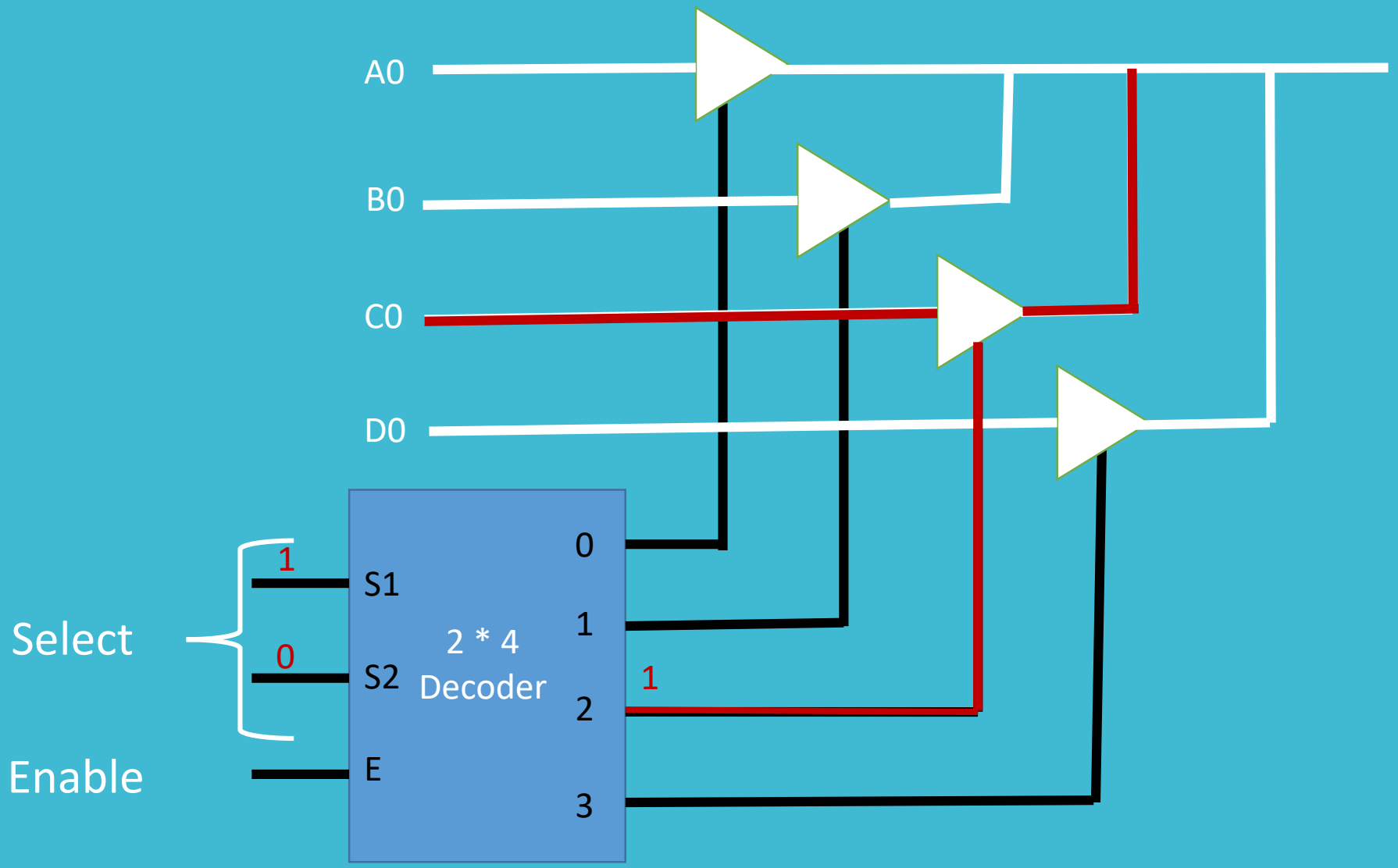
Figure 4-4 Graphic symbols for three-state buffer.



Tri-State Bus Buffers (3 State Buffer)



Common bus System using decoder and tri-state Buffers



Tri-State Bus Buffers (3 State Buffer)

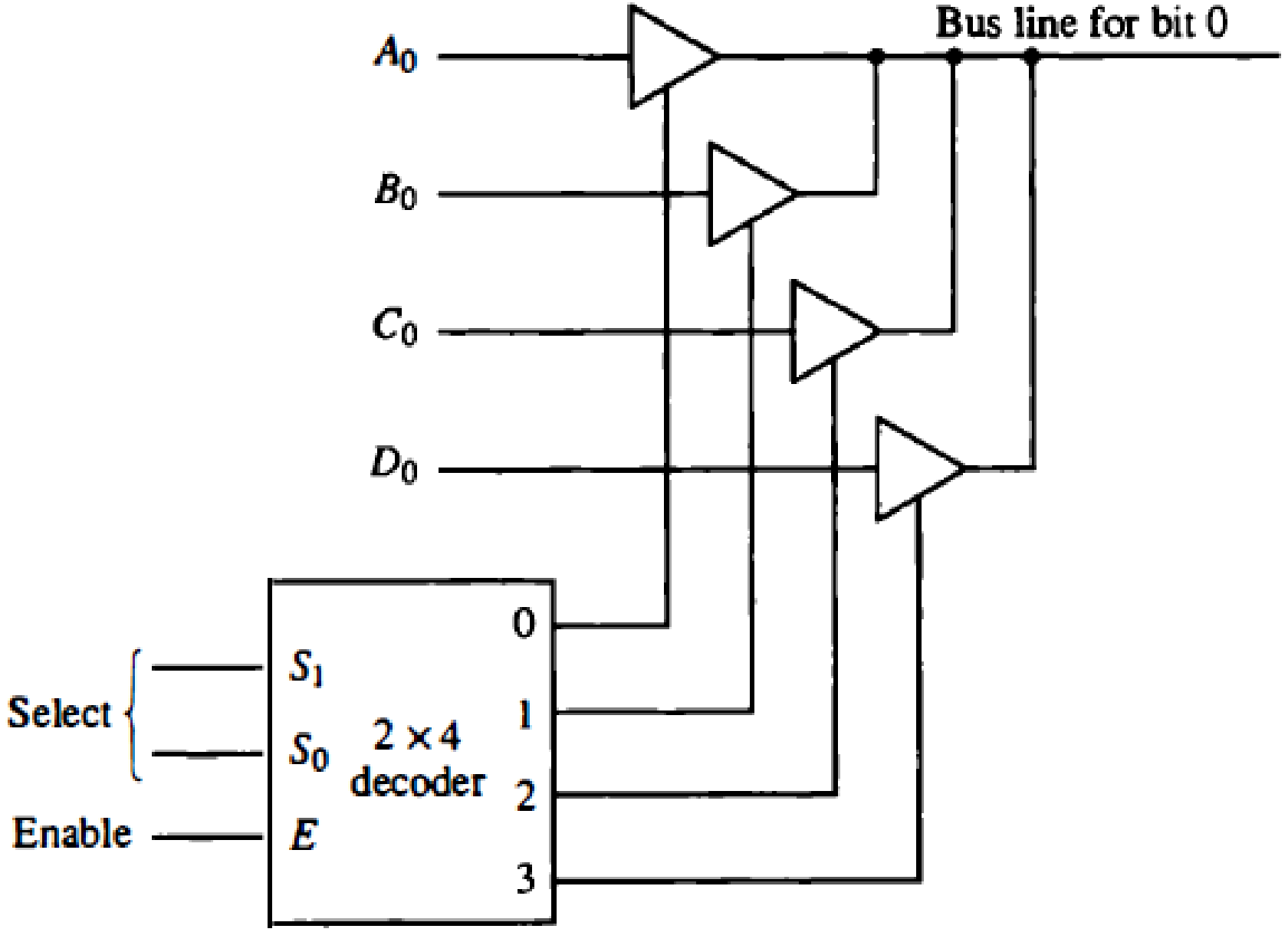


Figure 4-5 Bus line with three state-buffers.

Logic Gates

- Logic gates are the basic building blocks of any digital system.
- It is an electronic circuit having one or more than one input and only one output.
- The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

Different Logic Gates are as listed below:

(1) AND Gate

(2) OR Gate

(3) NOT Gate

(4) NAND Gate

(5) NOR Gate

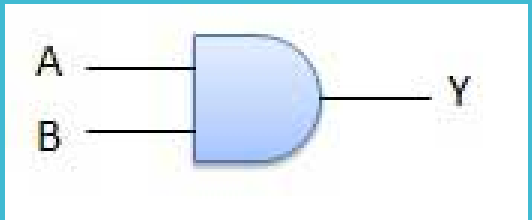
(6) XOR Gate

(7) XNOR Gate

Logic Gates – AND Gate

- A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

Logic diagram: A*B

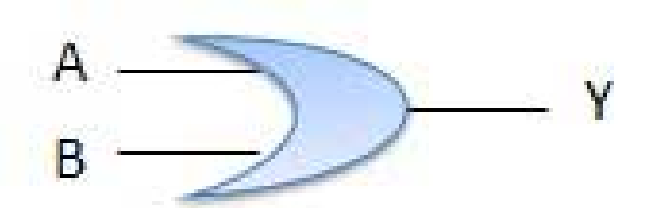


Truth Table

Inputs		Output
A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

Logic Gates – OR Gate

- A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.
- **Logic diagram: A*B**



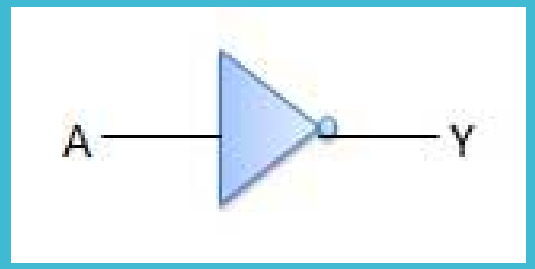
Truth Table

Inputs		Output
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Logic Gates – NOT Gate

- NOT gate is also known as Inverter. It has one input A and one output Y.
- Logic diagram: A*B

Y	=	NOT A
Y	=	$\overline{A}$



Truth Table

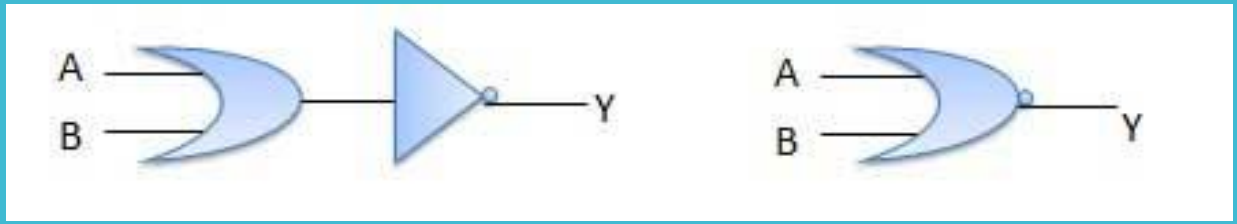
Input	Output
0	1
1	0

Logic Gates – NAND Gate

- A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

Y	=	A NOT AND B NOT AND C ..... N
Y	=	A NAND B NAND C ..... N

- Logic diagram: A*B



Truth Table

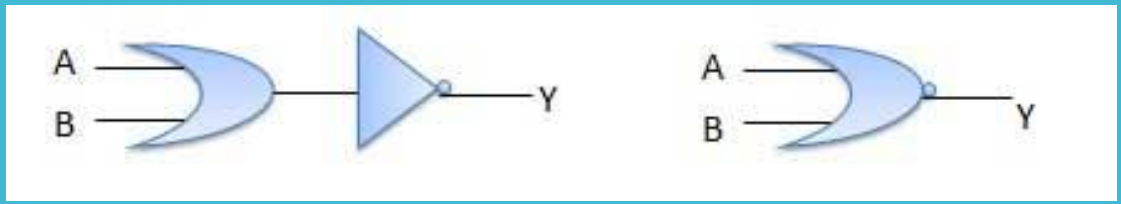
Inputs		Output
A	B	$\overline{A\ B}$
0	0	1
0	1	1
1	0	1
1	1	0

Logic Gates – NOR Gate

- A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

Y = A NOT OR B NOT OR C ..... N
Y = A NOR B NOR C ..... N

- **Logic diagram:**



Truth Table

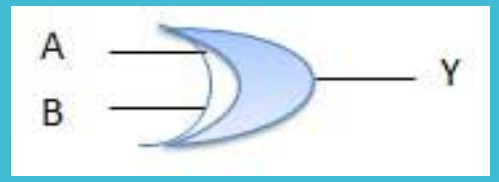
Inputs		Output
A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

Logic Gates – XOR Gate

- XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor.
- The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$
$$Y = A \oplus B \oplus C \ldots\ldots N$$
$$Y = \overline{A}B + A\overline{B}$$

- **Logic diagram:**



Truth Table

Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Logic Gates – XNOR Gate

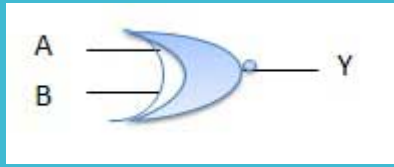
- XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor.
- The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$

$$Y = A \odot B \odot C \ldots\ldots N$$

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

- **Logic diagram:**



Truth Table

Inputs		Output
A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Arithmetic Micro-operations

- In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers.
- The basic Arithmetic Micro-operations are classified in the following categories:

1) Addition
2) Subtraction
3) Increment
4) Decrement
5) Shift

Arithmetic Micro-operations

- The following table shows the symbolic representation of various Arithmetic Micro-operations.

Symbolic Representation	Description
R3 ← R1 + R2	The contents of R1 plus R2 are transferred to R3.
R3 ← R1 - R2	The contents of R1 minus R2 are transferred to R3.
R2 ← R2'	Complement the contents of R2 (1's complement)
R2 ← R2' + 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2' + 1	R1 plus the 2's complement of R2 (subtraction)
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 - 1	Decrement the contents of R1 by one

Logic Micro-operations

- Operations -> AND, OR, NOT, NOR, NAND, XOR/EXOR, XNOR.
- Bits of the registers are consider separately as binary variable.
- Ex.- R1 <- R1 ^ R2, [^ = AND operation]

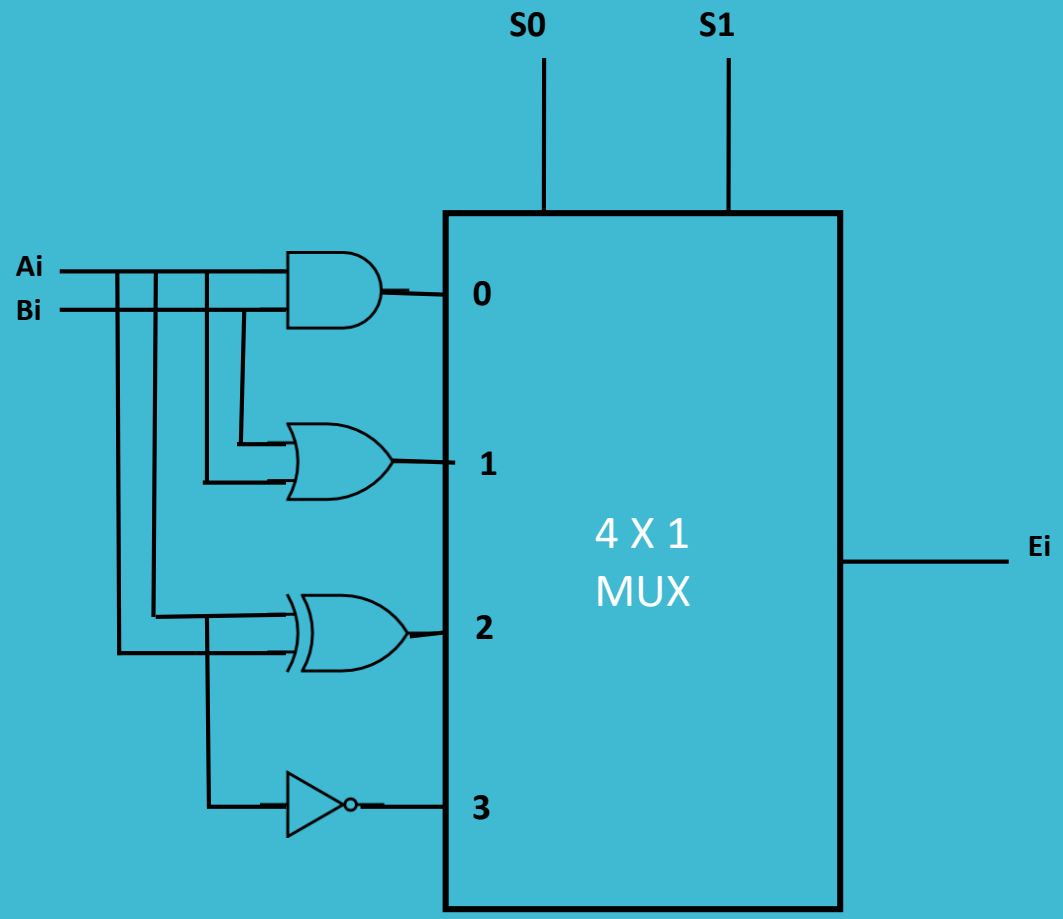
R1 = 1 1 0 0

R2 = 1 0 0 1

AND 1 0 0 0 [New value of R1 = 1 0 0 0]

- Logic Micro-operations is used for bit manipulations and decision making.
- There are 4 commonly used Micro-operations i.e., AND, OR, XOR, Complement.

For Hardware implementation of Logic Operation



- Select lines will select one of the 4 input and enale that as output.
- i = 0,1,……..n (n bit reg)

S1	S0	Ei	Operation
0	0	Ai ∧ Bi	AND
0	1	Ai ∨ Bi	OR
1	0	Ai ⊕ Bi	XOR
1	1	$\overline{Ai}$	Complement

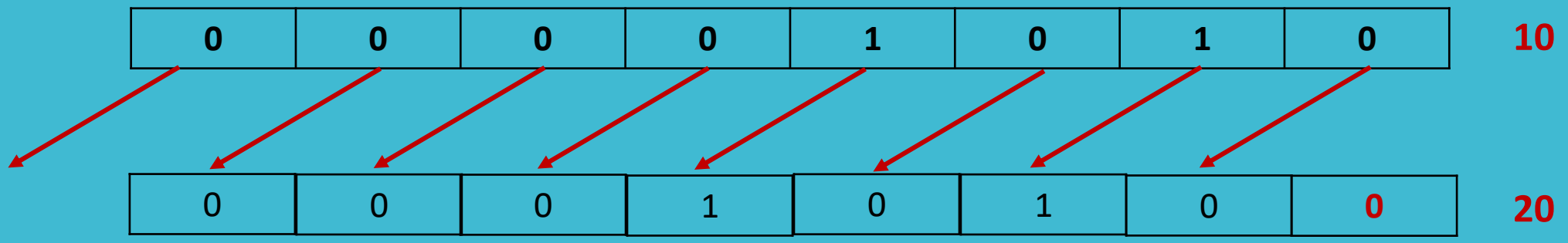
Shift Micro-operations

Shift Micro-operations- Logical Left Shift:

- In this shift, one position moves each bit to the left one by one.
- The Empty least significant bit (LSB) is filled with zero (i.e, the serial input), and the most significant bit (MSB) is rejected.
- The left shift operator is denoted by the double left arrow key (<<). The general syntax for the left shift is shift-expression << k.

Shift Micro-operations- Logical Left Shift:

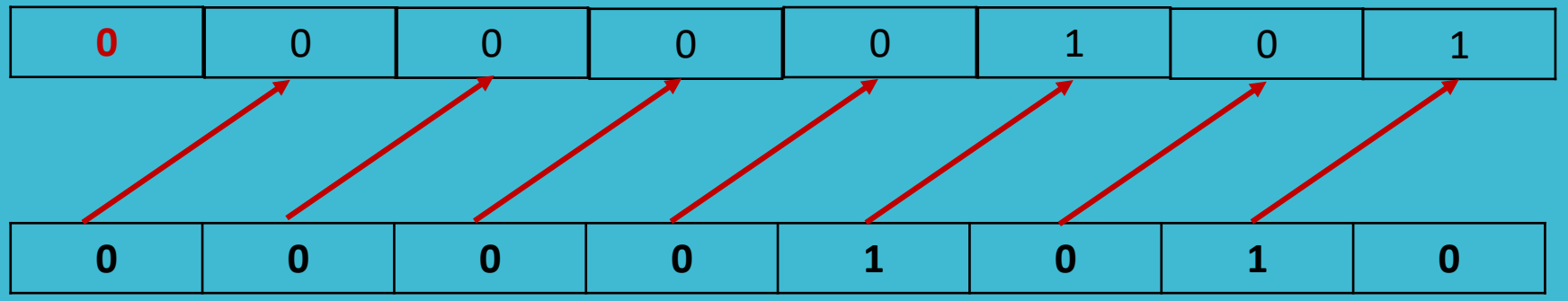
Example:-



- **Left Shift denotes multiplication of 2**

Shift Micro-operations- Logical Right Shift:

Example:-

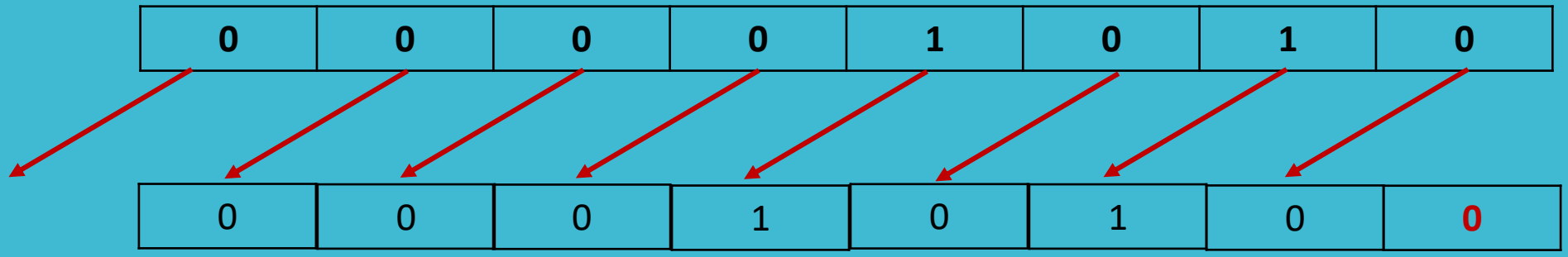


- **Left Shift denotes Division of 2**

Arithmetic Micro-operations- Arithemetic Left Shift:

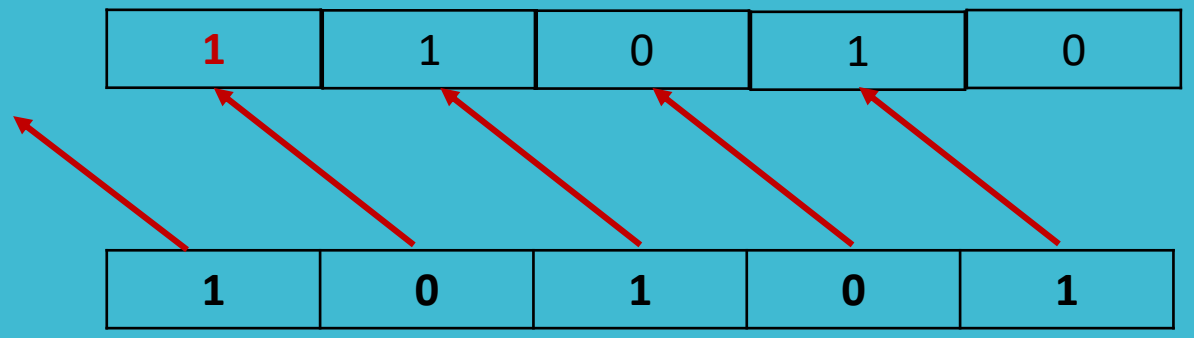
Example:-

In this shift, each bit is moved to the left one by one. The empty least significant bit (LSB) is filled with zero and the most significant bit (MSB) is rejected. Same as the Left Logical Shift.



Arithmetic Micro-operations- Arithmetic Left Shift:

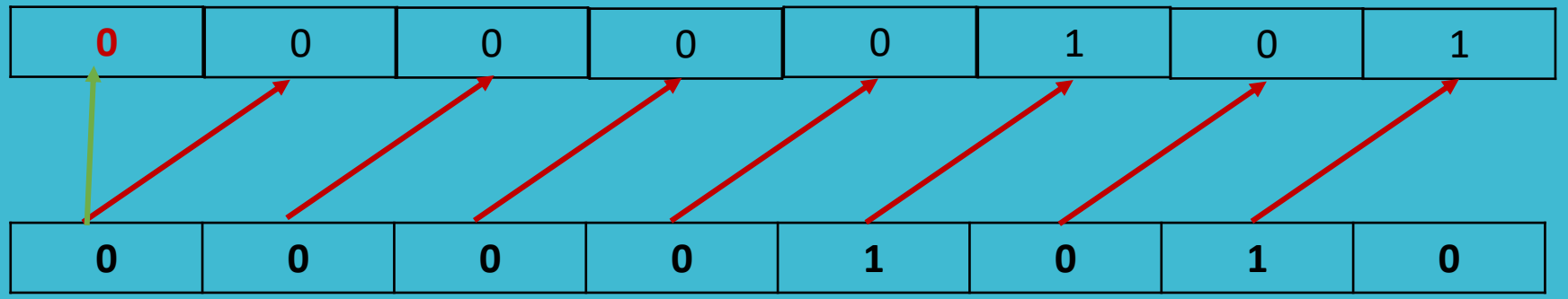
Example 2:-



Arithmetic Micro-operations- Arithmetic Right Shift:

Example 1:-

In this shift, each bit is moved to the right one by one and the least significant(LSB) bit is rejected and the empty most significant bit(MSB) is filled with the value of the previous MSB.



Arithmetic Micro-operations- Arithmetic Right Shift:

Example 2:-