MADHYA PRADESH BHOJ(OPEN) UNIVERSITY BHOPAL

Computer Organization and Architecture

MASTER OF SCIENCE (CYBER SECURITY)

BLOCK INTRODUCTION

In this block, we will explore the foundational concepts of computer organization. We will begin by discussing the basic organization of a computer and the functional units involved in program execution. We will then delve into digital components such as flip-flops, counters, registers, and decoders, which form the building blocks of digital circuits. We will also cover data representation and computer arithmetic, including the design of ALUs and control units. Finally, we will learn about instruction sets, instruction formats, and addressing modes, which are essential to understanding how a computer executes instructions.

The structure of Block 1 is as follows:

Unit 1: Basic Organization of the Computer
Unit 2: Digital Components
Unit 3: Data Representation
Unit 4: Computer Arithmetic
Unit 5: Instruction Sets
Unit 6: Addressing Modes
Unit 7: Input-Output Organization

While going through a unit, you will notice some along-side boxes, which have been included to help you know some of the difficult, unseen terms. Again, we have included some relevant concepts in ‘‘LET US KNOW’’ along with the text. And, at the end of each section, you will get “CHECK YOUR PROGRESS” questions. These have been designed to self-check your progress of study. It will be better if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with “ANSWERS TO CHECK YOUR PPROGRESS” given at the end of each unit.

UNIT 1: BASIC ORGANIZATION OF THE COMPUTER

UNIT STRUCTURE

1.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe the basic organization of computer
- learn about the functional characteristics of computer
- describe the concepts of different cycles of program execution such as fetching, decoding and execution

1.2 INTRODUCTION

This is the first unit of this course. In this unit, we will discuss the basics of the organization of computer. An effort is made to discuss the concept of different cycles of program execution such as fetching, decoding and execution. The organization of CPU, and Control Unit have also been discussed in this unit. In the next unit, we will explore the different combinational circuits and sequential circuits. Concepts related to counters and registers are discussed in the next unit.

1.3 BASIC ORGANIZATION OF THE COMPUTER

The basic organization of a typical computer is shown below in figure 1.1.

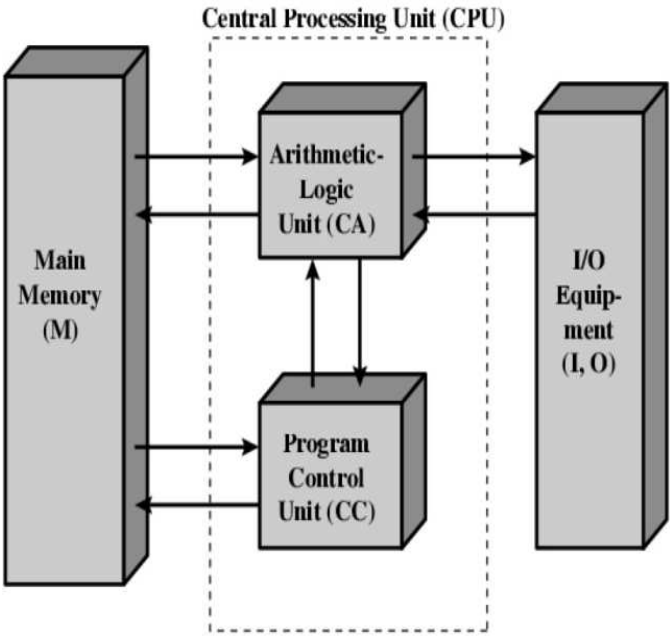
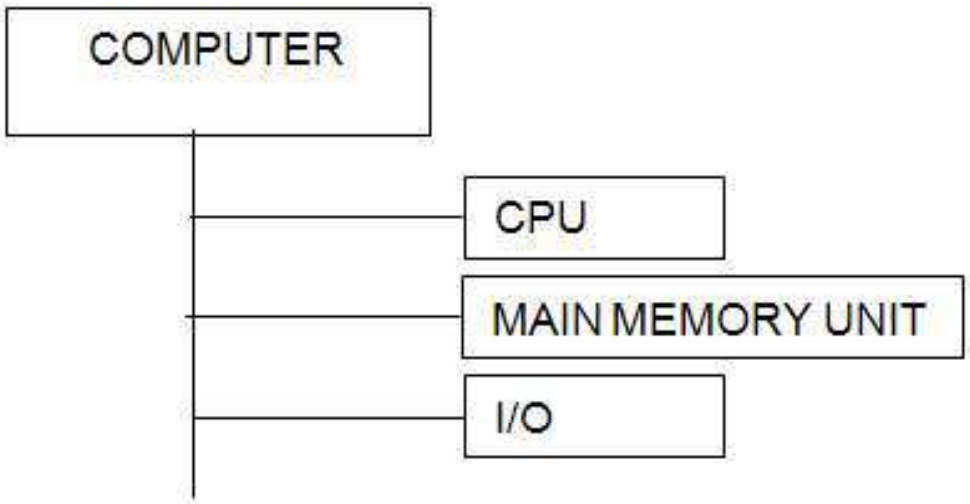


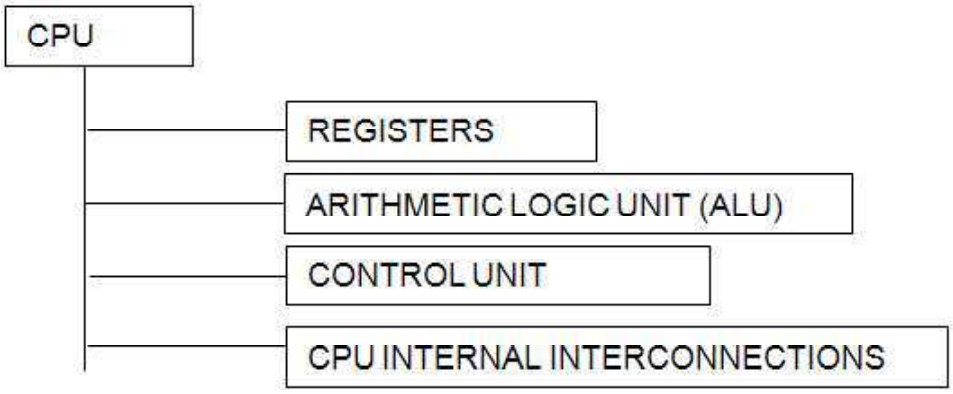
Figure 1.1: Basic Organization of a typical computer

From the above figure, which represents the basic block diagram of a computer, we find that the computer consists of basically three parts, namely, the Central Processing Unit i.e. CPU, Main Memory Unit, and the Input and Output devices. The basic organization of a typical computer is illustrated below in a simple form.



Organization of CPU:

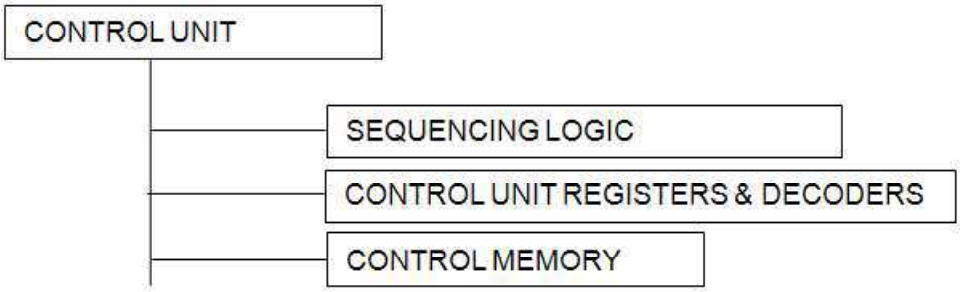
The organization of a typical CPU is as follows:



There are different types of **Registers** involving here, namely, Accumulator, Temporary Register, Instruction Register, Flag Register, General Purpose Register (GPR), Program Counter etc. Arithmetic Logic Unit (ALU), performs arithmetic and logic operations and needs some components such as:

1. Registers to store arguments or operands and results
2. Buses to carry data from registers to the ALU and results back to the register unit
3. Two registers for accessing memory with associated buses
4. An instruction unit to get instructions from computer memory as needed. This includes a program counter, which always points to the address of the next instruction to be accessed and a register called instruction register (IR).
5. A control unit that instructs the ALU on what to do.

The organization of a typical **Control Unit** is as follows:



CHECK YOUR PROGRESS

Q1. Computer consists of basically …….. parts.

Q2. The names of basic parts of a typical computer are ……

Q3. The names of the parts involved for the organization of a typical CPU are ……

Q4. GPR stands for ……..

Q5. Is there any need of a control unit in ALU? Answer briefly.

1.4 BLOCK LEVEL DESCRIPTION OF THE FUNCTIONAL UNITS

From the block level description, there are five functional units of the computer from the program execution point of view. They are:

1. Input unit
2. Output unit
3. Memory unit
4. Arithmetic and Logic unit
5. Control unit

The diagram representing the basic functional units of a computer is shown below:

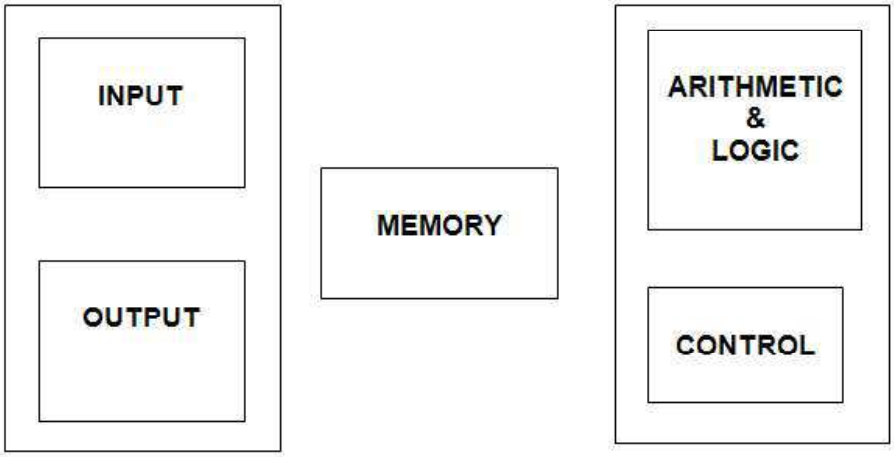


Figure 1.2: Block Representation of Basic Functional Units of a Computer

The users or programmers supply information through some electromechanical devices such as key-boards, mouse etc. and all the information is to be accepted by the Input unit.

The received information is either to be stored in memory or to be used by the ALU directly to perform the desired arithmetic or logical operation. If received information is to be stored in memory first, then the desired operation will happen in the latter (whenever necessary).Output unit will display the result. Control unit is responsible to control the entire task to happen in each and every block of the computing system.

FETCH, DECODE AND EXECUTE CYCLE

Programs i.e. set of instructions along with the data, on which instructions are to be executed are stored in memory. They are to be brought from memory into the CPU. To fetch an instruction and necessary data from memory, and to execute it, there are some necessary steps that a CPU has to carry out. These necessary steps constitute **Instruction cycle**. There are two cycles in an instruction cycle, one is Fetch cycle and other is Execute cycle.

In **Fetch cycle**, the Opcode (which specifies the operation to be performed, for eg. ADD, SUB etc.) is to be fetched from memory by the CPU. To fetch an instruction from memory, again there are some steps, And these steps constitute a Fetch cycle. Execute cycle is constituted by the steps need to carry out data from memory and to perform the operation specified in the opcode of an instruction.

Therefore we can say that,

Instruction cycle = Fetch Cycle + Execute Cycle

An instruction is to be fetched from memory by the processor at the beginning of each instruction cycle. Program Counter (PC) holds the address of the instruction to be fetched next. The fetched instruction is to be loaded into a register known as Instruction Register (IR). The processor has to interpret the instruction and to perform the required action. From fetching to execution of instruction can be described as follows:

- **Instruction Fetch :** Read instruction from memory location
- **Instruction Operation Decoding:** Analyze instruction to determine the type of operation to be performed and the **operand(s)** (data on which operation is to be performed)
- **Operand address calculation :** The memory address of the data is to be calculated if data are not supplied directly from input
- **Operand fetch:** Fetch the operand from memory or read it from I/O
- **Data Operation:** Perform the operation according to the instruction
- **Operand Store:** Write the result into memory or out to I/O

CHECK YOUR PROGRESS

Q6. There are …… numbers of functional units in a computer

Q7. The names of functional units in a computer are ……..

Q8. There are …….. cycles in an instruction cycle, one is ……. cycle and other one is ……..cycle GPR.

1.5 LET US SUM UP

- The computer consists of basically four parts. The names of basic parts involving the organization of a typical computer are: the Central Processing Unit i.e. CPU, Main Memory Unit, Input and Output devices and the System Interconnection.
- There are different types of Registers involved, namely, Accumulator, Temporary Register, Instruction Register, Flag Register, General Purpose Register (GPR), Program Counter etc
- Buses carry data from registers to the ALU and the results back to the register unit.
- An instruction fetch unit is needed to get instructions from computer memory. This includes a program counter, which

always points to the address of the next instruction to be accessed.

- A control unit instructs the ALU on what to do. To fetch an instruction and the necessary data from memory, and to execute it, there are some necessary steps that a CPU has to carry out. These necessary steps constitute instruction cycle.
- There are two cycles in an Instruction cycle: one is Fetch cycle and other is Execute cycle.

1.6 FURTHER READINGS

1) Mano, M. M. (2006). Computer systems architecture.

2) Hamacher, V. C., Vranesic, Z. G., Zaky, S. G., Vransic, Z., & Zakay, S. (1984). Computer organization (Vol. 3). New York et al.: McGraw-Hill.

1.7 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: Four

Ans to Q No 2: CPU, Main Memory Unit, Input, Output, System Interconnections

Ans to Q No 3: Register, ALU, Control Unit, CPU, Internal Interconnections

Ans to Q No 4: General Purpose Register

Ans to Q No 5: Yes

Ans to Q No 6: 5 (five)

Ans to Q No 7: Input, Output, Memory, ALU, Control Unit

Ans to Q No 8: Two, Fetch, Execute

1.8 MODEL QUESTIONS

Q1. Draw the basic block diagram of a typical computer.

Q2. Explain the basic organization of a computer.

Q3. Write the names of the basic parts of a typical computer.

Q4. Write the names of the parts involved for the organization of a typical CPU.

Q5. Explain the basic concept of the organization of an ALU.

Q6. Write the names of the parts involved for the organization of a typical control unit.

Q7. How many numbers functional units are there in a computer? Give their names.

Q8. Define Instruction cycle, Fetch cycle and Execution cycle.

Q9. Describe the steps involved from fetching to execution of instruction.

UNIT 2: DIGITAL COMPONENTS

UNIT STRUCTURE

2.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- define combinational circuit and sequential circuit
- describe the working principle of Half-adder, Full-adder
- describe the working principle of Half-subtractor, Full-subtractor
- describe the working principle of Multiplexer, Demultiplexer
- describe the working principle of Encoder, Decoder
- describe the working principle of Flip-Flop, Register, Counter

2.2 INTRODUCTION

In the previous unit, we have discussed about the basic organization of the computer. In this unit, we will discuss the various combinational and sequential circuits to build your understanding on different digital components that are used in digital applications. A sound knowledge on these components will make you feel confident to understand the working principle of many digital devices in general and the computer system in particular. Combinational circuits, sequential circuits along with counters and registers are covered in detail in this unit. In this next unit, we will explore different data representation systems.

2.3 COMBINATIONAL CIRCUITS

To obtain a desired output according to a Boolean equation, various logic gates are often interconnected. These circuits are called combinational circuits. These are the combinations of different fundamental logic gates and hence the name Combinational Circuit. Its main characteristic is that the output is solely determined by the present inputs.

Inputs to a combinational circuit are given as either 0 or 1 and outputs are also available as either 0 or 1. In the process of designing a combinational circuit, first, a truth table is formed for the Boolean expression, that describes the combinational circuit. Next, the function is simplified and then the simplified function is implemented with gates to obtain the logic diagram of the combinational circuit. We may also obtain the Boolean expression from this logic diagram after determining the truth table.

2.3.1 Half-Adder

Half-adder is a circuit that can add two binary bits. Its outputs are SUM and CARRY. The following truth table shows the various combinations of inputs and the corresponding outputs of a half-adder. X & Y denote inputs and C & S denotes the two outputs CARRY & SUM.

Table 2.1: Truth Table for a Half-Adder

X	Y	CARRY (C)	SUM (S)
0	0	0	0
0	1	0	1 ($\overline{X}Y$)
1	0	0	1($X\overline{Y}$)
1	1	1 (XY)	0

The minterms for Sum and CARRY are shown in the bracket. The Sum-Of-Product equation for SUM is :

$$S = \overline{X}Y + X\overline{Y} \quad \ldots\ldots\ldots\ldots (1)$$
$$= X \oplus Y$$

Similarly, the SOP equation for the CARRY is:

$$C = XY \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Combining the logic circuits for equation (1) & (2) we get the circuit for Half-Adder as :

Figure 2.1: Half-Adder Circuit and Symbol

2.3.2 Full-Adder

Full-Adder is a logic circuit to add three binary bits. Its outputs are SUM and CARRY. In the following truth table X,Y,Z are inputs and Ć and Ś are CARRY & SUM.

Table 2.2: Truth Table for Full- Adder

X	Y	Z	CARRY (Ć)	SUM (Ś)
0	0	0	0	0
0	0	1	0	1 ($\overline{X}\,\overline{Y}$Z)
0	1	0	0	1 ($\overline{X}$Y$\overline{Z}$)
0	1	1	1 ($\overline{X}$YZ)	0
1	0	0	0	1 (X$\overline{Y}\,\overline{Z}$)
1	0	1	1 (X$\overline{Y}$Z)	0
1	1	0	1 (XY$\overline{Z}$)	0
1	1	1	1 (XYZ)	1 (XYZ)

The minterms are written in the brackets for each 1 output in the truth table. From these, the SOP equation for full summation can be written as:

$$\acute{S} = \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ$$
$$= \overline{X}\,(\overline{Y}Z + Y\overline{Z}) + X(\overline{Y}\,\overline{Z} + YZ)$$
$$= \overline{X}S + X\overline{S} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Here, S is SUM of Half-Adder.

Again, SOP equation for Full–Adder CARRY is :

$$\acute{C} = \overline{X}YZ + X\overline{Y}Z + XY\overline{Z} + XYZ$$
$$= \overline{X}YZ + XYZ + X\overline{Y}Z + XY\overline{Z}$$
$$= (\overline{X} + X)YZ + X(\overline{Y}Z + Y\overline{Z})$$
$$= YZ + XS$$
$$= C + XS \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

Here also, C means CARRY of half-adder and S means SUM of half-adder.

Now, using two half-adder circuits and one OR gate we can implement equation (3) and (4) to obtain a full-adder circuit as follows.

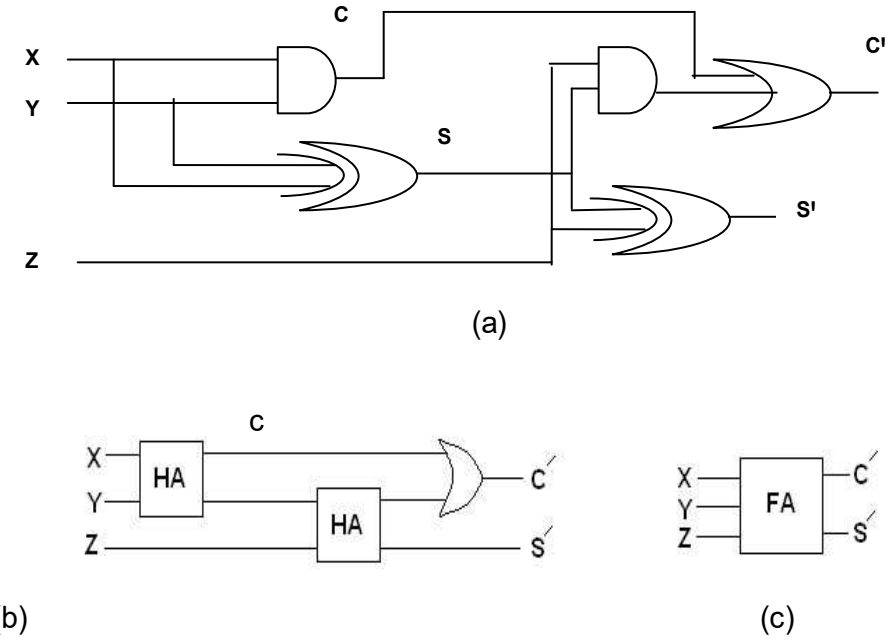


Figure 2.2: Full-Adder Circuit

(a) logic diagram, (b) block diagram (c) Symbol

2.3.3 Half - Subtractor

A half-subtractor subtracts one bit from another bit. It has two outputs, DIFFERENCE (D) and BORROW (B).

Table 2.3: Truth Table for Half-Subtractor

X	Y	BORROW(B)	DIFFERENCE (D)
0	0	0	0
0	1	1 ($\overline{X}$ Y)	1 ($\overline{X}$ Y)
1	0	0	1 (X $\overline{Y}$)
1	1	0	0

The mean terms are written within parenthesis for output 1 in each column. The SOP equations are :

$$D = \overline{X}Y + X\overline{Y} = X \oplus Y$$

$$= S \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

$$B = \overline{X}Y \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6)$$

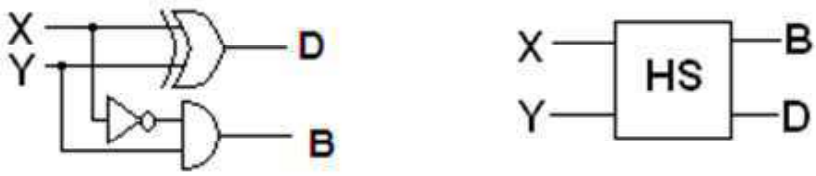


Figure 2.3: The Half-Subtractor circuit and symbol

2.3.4 Full-Subtractor

A full-subtractor circuit finds the Difference and Borrow on the subtraction operation involving three binary bits.

Table 2.4: Truth Table of Full-Subtractor

X	Y	Z	BORROW (B´)	DIFFERENCE(D´)
0	0	0	0	0
0	0	1	1 ($\overline{X}\,\overline{Y}Z$)	1 ($\overline{Y}\,\overline{X}Z$)
0	1	0	1 ($\overline{X}Y\overline{Z}$)	1 ($\overline{X}Y\overline{Z}$)
0	1	1	1 ($\overline{X}YZ$)	0
1	0	0	0	1 ($X\overline{Y}\,\overline{Z}$)
1	0	1	0	0
1	1	0	0	0
1	1	1	1 (XYZ)	1 (XYZ)

The SOP equation for the DIFFERENCE is :

$$D' = \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ$$
$$= \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ + \overline{X}\,\overline{Y}Z$$
$$= (\overline{X}Y + X\overline{Y})\,\overline{Z} + (XY + \overline{X}\,\overline{Y})Z$$
$$= D\,\overline{Z} + \overline{D}Z \quad \ldots\ldots\ldots\ldots\ldots\ldots (7)$$

And SOP equation for BORROW is :

$$B' = \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + \overline{X}YZ + XYZ$$
$$= \overline{X}\,\overline{Y}Z + XYZ + \overline{X}Y\overline{Z} + \overline{X}YZ$$
$$= (\overline{X}\,\overline{Y} + XY)Z + \overline{X}Y(\overline{Z} + Z)$$
$$= \overline{D}Z + \overline{X}Y \quad \ldots\ldots\ldots\ldots\ldots\ldots (8)$$

In equation (7) and (8) , D stands for DIFFERENCE of half-subtractor. Now, from the equations (7) and (8) we can construct a full-subtractor using two half-subtractor and an OR gate.

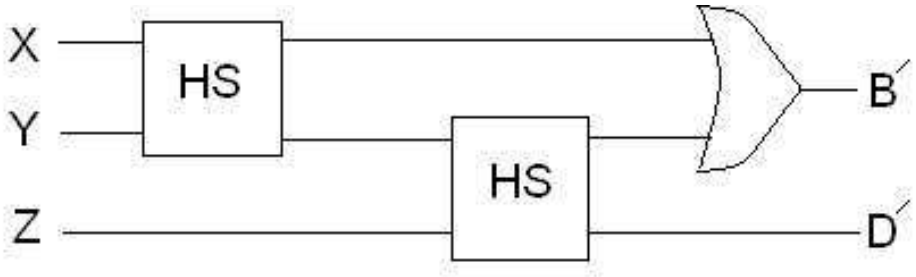
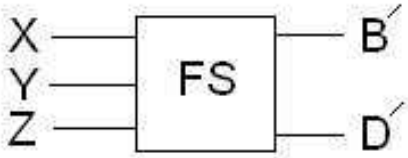


Figure 2.4: Full-Subtractor Circuit

The symbol of full-subtractor is :



CHECK YOUR PROGRESS

Q1. A half-adder can add

(a) Two binary number of 4 bit each (b) Two binary bit

(c) Add half of a binary number (d) None of these

Q2. A full-adder is a logic circuit that has two outputs namely:

(a) product & sum (b) sum & borrow

(c) sum & carry (d) carry & borrow

Q3. A half-subtractor can perform

(a) subtraction of two binary bits (b) product of two binary bits

(c) complement of half binary bits (d) none of these

Q4. A full-subtractor has the ability to do

(a) subtraction of two binary numbers (b) subtraction of three binary bits

(c) product of three binary bits (d) division three binary bits

2.3.5 Multiplexer

Multiplexer is a circuit which has many inputs and only one output. Multiplexer can select any one of its many inputs by applying a control signal and steer the selected input to the output. Generalized block diagram of a multiplexer is shown in figure 2.5.

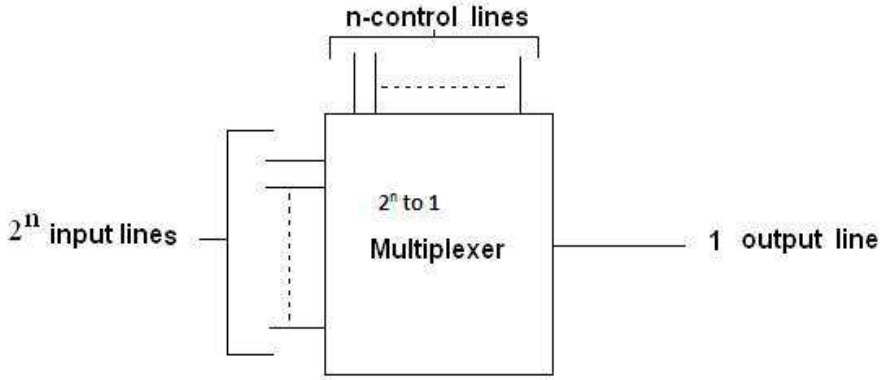


Figure 2.5: Block Diagram of Multiplexer.

It has 2^n inputs, n – numbers of control or selection lines and only one output. A multiplexer is also called a many – to – one data selector.

8-to-1 MULTIPLEXER:

Figure 2.6 shows a 8-to-1 multiplexer, where there are 8 inputs, 3 control or selection lines and 1 output. The eight inputs are labeled as $I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7$ and the selection lines are labeled as A, B, C, where input is steered to the output and depends on the value of ABC. For example, if ABC = 000, then the upper AND gate is enabled and all other AND gates are disabled. As a result, the input I_0 alone is steered to the output. Similarly if ABC = 110 then the AND gate connected to the data line I_6 is enabled while all the other AND gates are disabled. Therefore, the input I_6 appears at the output. Hence when ABC = 110 , the output is Y = I_6 .

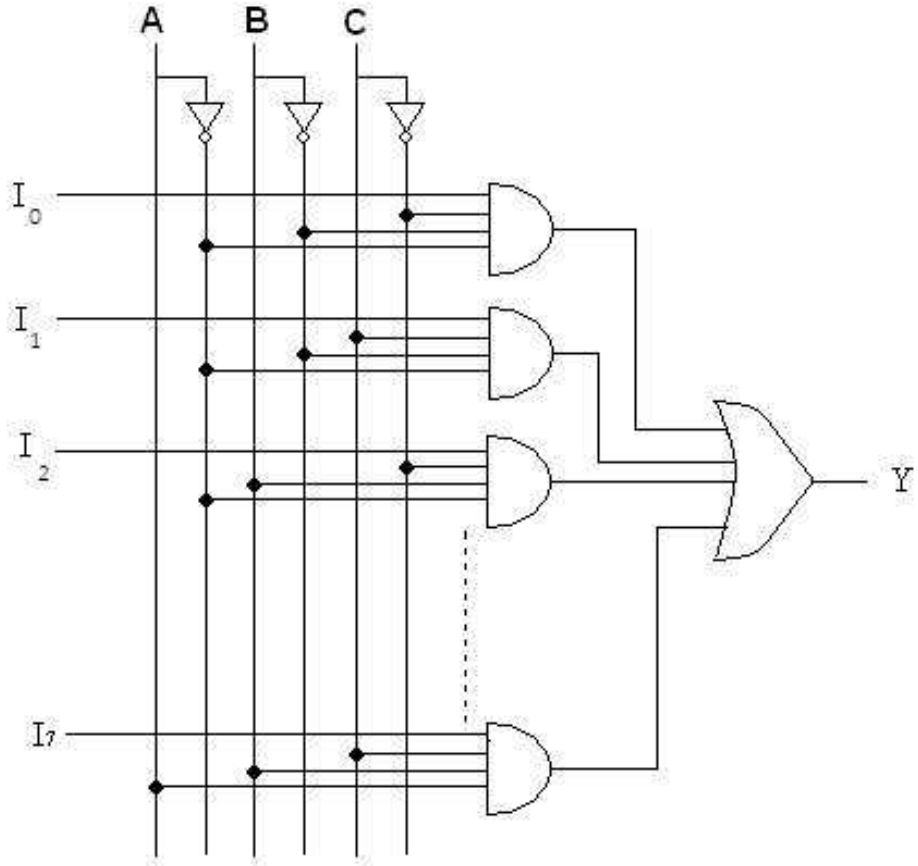


Figure 2.6: 8-to-1 Multiplexer

16-to-1 MULTIPLEXER

Figure 2.7 shows a 16 – to – 1 multiplexer. In this circuit, there are 16 data input lines, 4 control lines and 1 output. The input lines are denoted by $I_0, I_1, I_2, I_3, \ldots\ldots\ldots\ldots I_{15}$ and the control lines are donated by ABCD. The output is denoted by Y.

Out of 16 input lines, only one is transmitted to the output depending upon the value of ABCD. If ABCD = 0000 then I_0 is steered to the output since the upper AND gate is enabled alone and all others are disabled. Similarly if ABCD = 0010 then I_2 appears at the output. If ABCD = 1111 then the last AND gate is only enabled and therefore I_{15} appears at the output.

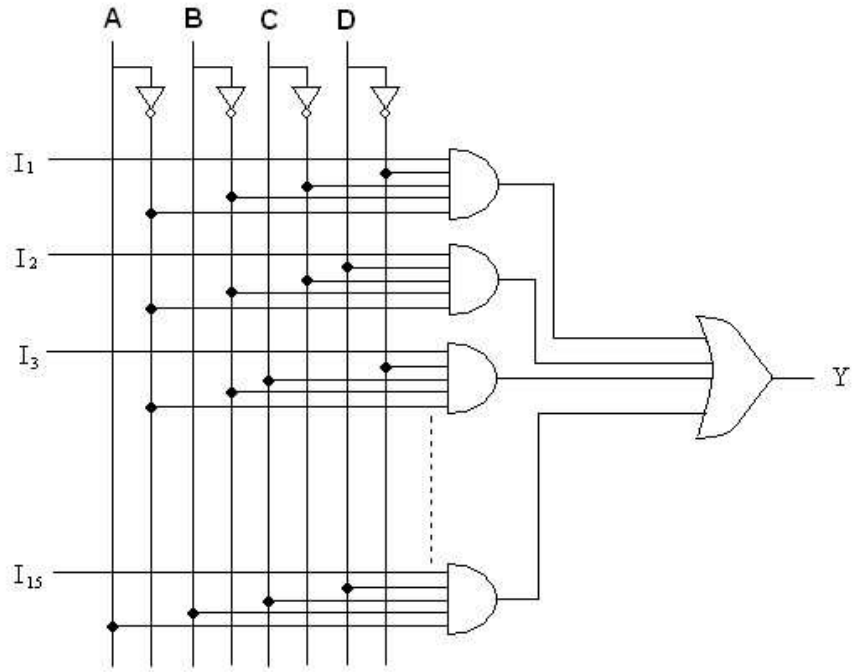


Figure 2.7: 16-to-1Multiplexer

2.3.6 De-Multiplexer

De-multiplexer means One-to-Many. It is opposite to the multiplexer. It has 1 input and many outputs. With the application of appropriate control signal, the common input data can be steered to one of the output lines. Figure 2.8 shows a generalized block diagram of a de-multiplexer.

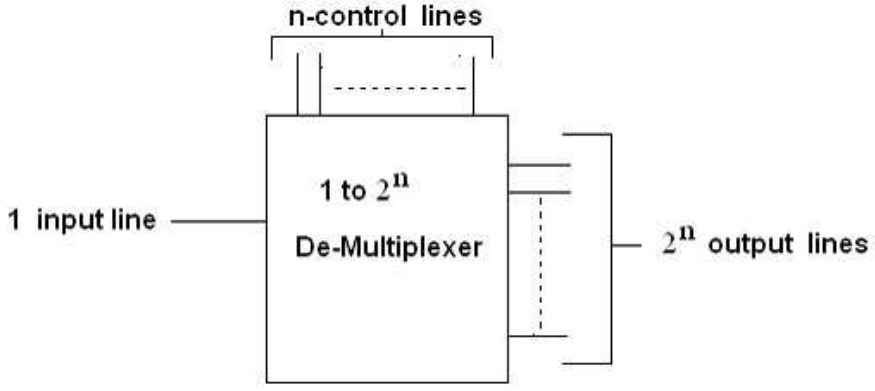


Figure 2.8: Block diagram of a De-Multiplexer

To have 2^n output lines, there must be n control lines in a de-multiplexer.

1-to-16 DE-MULTIPLEXER

In figure 2.9 we have shown a 1-to- 16 de-multiplexer.

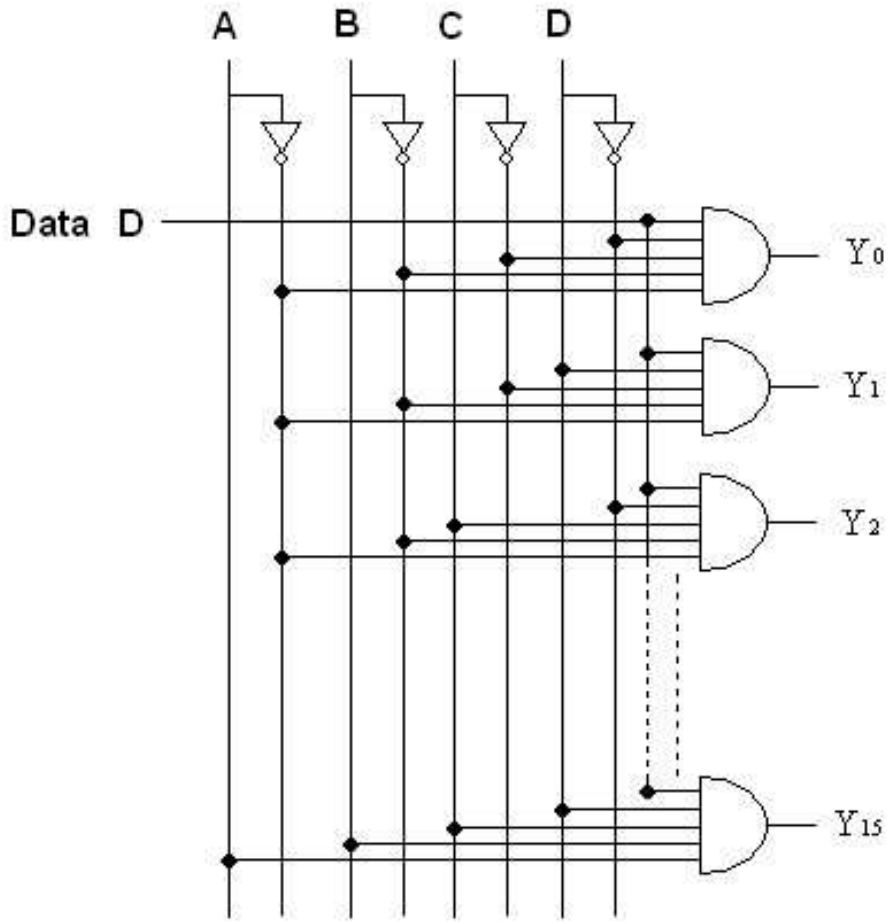


Figure 2.9: 1-to-16 De-Multiplexer

Here the data input line is denoted by D. This input line is connected to all the AND gates through which output appears. Depending upon the control signal, only one AND gate becomes enabled and the data input D appears through that AND gate. So, when ABCD = 0000, the upper AND gate is enabled and data input D appears at Y_0 as output.

When ABCD = 1111, the bottom AND gate becomes enabled and D appears at Y_{15} as output. For other combinations, D' appears at other output terminal.

CHECK YOUR PROGRESS

Q5. Multiplexer means _____

(a) multiple to many (b) one to many

(c) many to one (d) one to one

Q6. A 16-to-1 multiplexer has _____

(a) 1 control lines (b) 2 control lines

(c) 3 control lines (d) 4 control lines

Q7. De-multiplexer means _____

(a) deduct multiple bits (b) one-to-many

(c) multiple-to-multiple (d) one-to-one

2.3.7 Encoder

An encoder converts a digital signal into a coded signal. A generalized view of an encoder is shown in figure 2.10

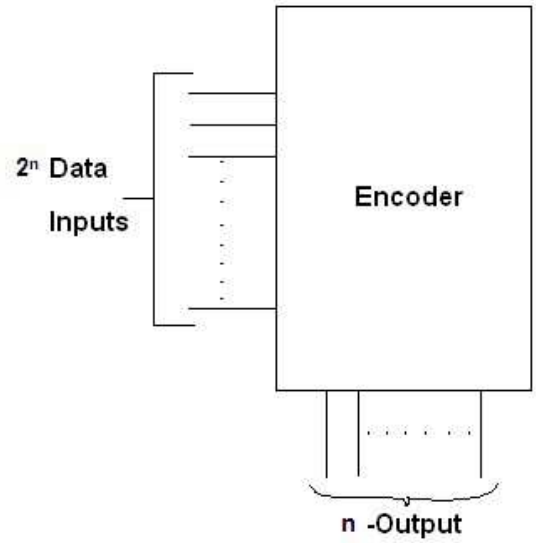


Figure 2.10: Block Diagram of an Encoder

In figure 2.10, we can see that there are 2^n input lines and n – numbers of output lines. Out of 2nd inputs, only one input line is active at a time. Encoder generates a coded output which is unique for each of the active input.

DECIMAL- TO -BCD ENCODER

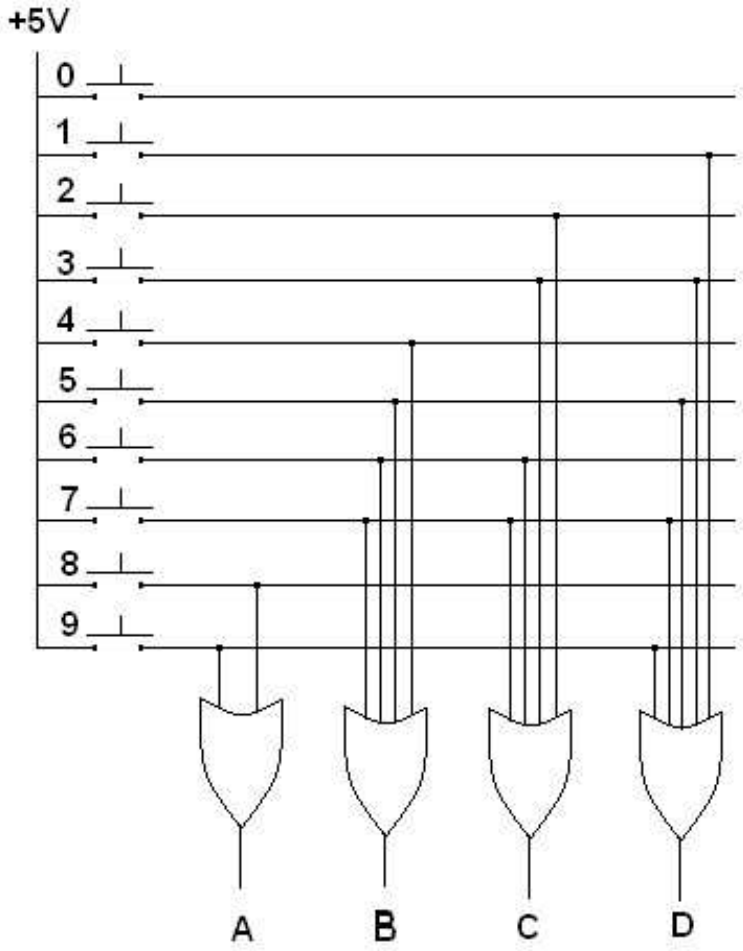


Figure 2.11: Decimal-to-BCD Encoder

A decimal to BCD encoder is shown in Fig 2.11 This circuit generates BCD output when any one of the push button switches is pressed. For example, if button 6 is pressed, the B and C OR gates have high inputs and the corresponding output becomes

ABCD = 0110

If button 8 is pressed, the OR gate A receives a high input and therefore the output becomes

ABCD = 1000

2.3.8 Decoder

A decoder is a digital circuit which has n-input lines and 2^n output lines. A decoder and a de-multiplexer has similarity. In a de-multiplexer, there is a single input line connected to every output 'AND' gate whereas in a decoder the input line is absent.

A 3–to–8 decoder as shown in Figure 2.12. Its truth table is shown below:

Table 2.5: Truth table for a 3-to-8 decoder

Inputs			Outputs							
A	**B**	**C**	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

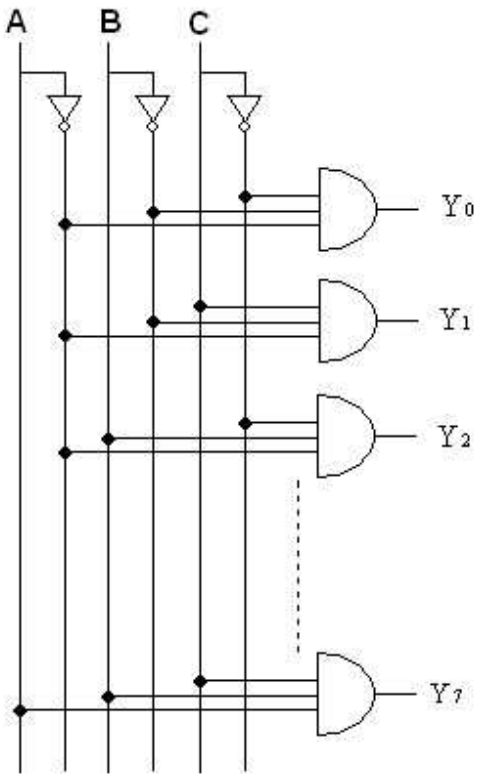


Figure 2.12: 3-to-8 decoder

2.3.9 Magnitude Comparator

A magnitude comparator circuit can compare two binary numbers to determine which one is greater than the other or their equality. Such a magnitude comparator has three output lines for A > B, A = B, A < B where A & B are two n-bits binary numbers. Every bit of one number is compared with the corresponding bit of the other number by ExOR gate.

A 4-bit magnitude comparator, SN 7485 is available in chip form. The block diagram of SN 7485 is shown in figure 2.13. It compares two 4-bit binary numbers $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$. Three output terminals are available for A < B, A = B and A > B.

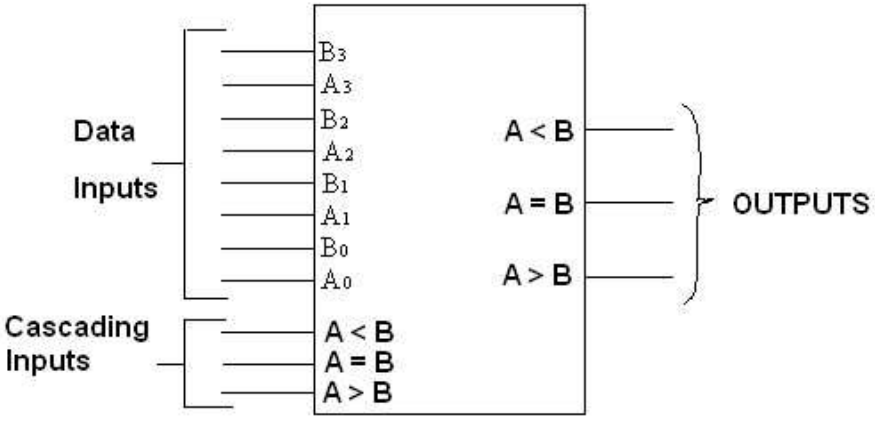


Figure 2.13: Two 4-bit words magnitude comparator SN 7485

Table 2.6: Function Table SN 7485

Comparing Inputs				Cascading Inputs			Outputs		
A_3B_3	A_2B_2	A_1B_1	A_0B_0	**A > B**	**A < B**	**A = B**	**A > B**	**A < B**	**A = B**
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$		$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$		$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$		$A_1 = B_1$	$A_0 = B_0$	L	L	H	L	L	H
	$A_2 = B_2$								
	$A_2 = B_2$								
	$A_2 = B_2$								

To compare any number having more than 4-bits, two or more such chip can be cascaded. The A > B, A < B and A = B outputs of a stage that handles less significant bits are connected to the corresponding cascading inputs of the next stage that handles the more significant bits.

CHECK YOUR PROGRESS

Q8. An encoder–

(a) converts a digital input to another form of digital output.

(b) converts analog input to digital output

(c) selects one out of many inputs.

(d) none of these.

Q9. Decoder has n inputs line and–

(a) n output lines.

(b) 2^n output lines.

(c) n^2 output lines.

(d) no output lines.

Q10. Magnitude comparator–

(a) compares two multi bit binary number.

(b) magnify any digital signal.

(c) compress binary numbers.

(d) check error in a binary number.

2.4 SEQUENTIAL CIRCUITS

If the output of a circuit depends on its present inputs and the immediate past output, then the circuit is called **sequential circuit**. To build a sequential circuit, we need memory circuits and combinational circuits. Flip-Flop is used as memory circuit, the application of which we would see in counter, register etc.

2.5 FLIP-FLOPS

A digital circuit that can produce two states of output, either high or low is called a multivibrator. There are three types of multivibrators. They are monostable, bi-stable and a stable.

A Flip-Flop is a bi-stable multivibrator and therefore it has two stable states of output-either high or low. Depending on its inputs and previous output, its new output is either high (or 1) or low (or 0). Once the output is fixed, the inputs can be removed and then also the already fixed output will be retained by the flip-flop. Hence, a flip-flop can be used as basic circuit of memory for storing for one bit of data. To store multiple bits we can use multiple numbers of Flip-Flop. These are also used to build counter, register etc.

There are many types of Flip-Flops. Some of them are:

- RS Flip-flop
- D Flip-Flop
- JK Flip-Flop
- MS Flip-Flop

2.5.1 RS Flip-Flop

RS Flip-Flop is also called Set-Reset Flip-Flop. A RS flip-flop can be built using many different circuits. Here, we have shown two RS flip-flop circuits using NOR and NAND gate. To study its working principle, we can use any one of them. In the following section, let us consider the RS flip-flop using NOR gate.

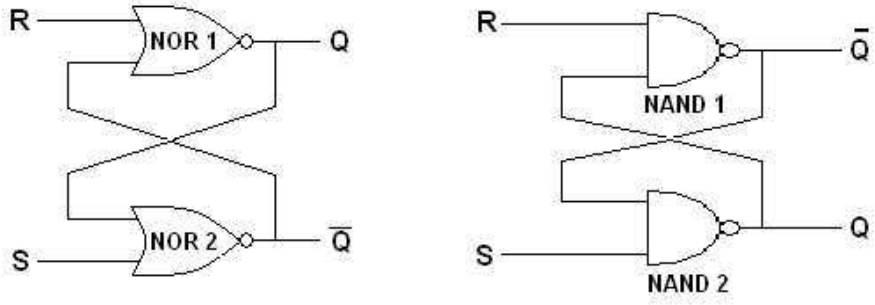


Figure 2.14: Basic circuit of RS Flip-Flop using NOR and NAND gate

RS flip-flop has two inputs —Set(s) and Reset(R). It has two outputs, Q and $\overline{Q}$. It should be noted that $\overline{Q}$ is always the complement of Q. Various combinations of inputs and their corresponding outputs are listed in the truth table below:

Table 2.7: Truth Table of RS Flip-Flop with NOR gates

R	S	Q	Action
0	0	Last value	No change
0	1	1	Set
1	0	0	Reset
1	1	?	Forbidden

The first input condition in the table is R=0, S=0. Since a 0 input has no effect on its output, the Flip-Flop retains its previous state. Hence Q remains unchanged.

The second input condition R=0, S=1 forces the output of NOR gate2 low. This low output will reach NOR gate1 and when both inputs of NOR gates1 are low, its inputs Q will be high. Thus a 1 at the S input will SET the flip-flop and Q will be equal to 1.

The third input condition R=1, S=0 will force the output of NOR gate1 to low. This low will reach NOR gate2 and force its outputs to high. Hence, when R=1, S=0, then Q=0, $\overline{Q}$=1. Thus the flip-flop is RESET.

The last input condition in the table R=1, S=1 is forbidden since it forces both the NOR gates to the low state. It means both Q=0, and =0 at the same time, which violates the basic definition of flip-flop that requires Q to be the complement of Q Hence this input condition is forbidden and its output is unpredictable.

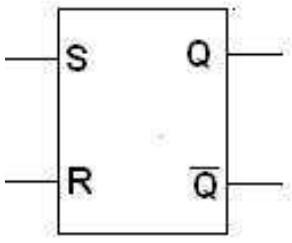


Figure 2.15: Symbol of RS Flip-Flop

CLOCK INPUT

For synchronization of operation of multiple flip-flop, an additional signal is added to all types of flip-flop. It is called clock signal, generally abbreviated as CLK. Addition of CLK signal ensures that whatever may be the input to the flip-flop, it affects the output only when CLK signal is given. Figure 2.16 shows a clocked RS flip-flop.

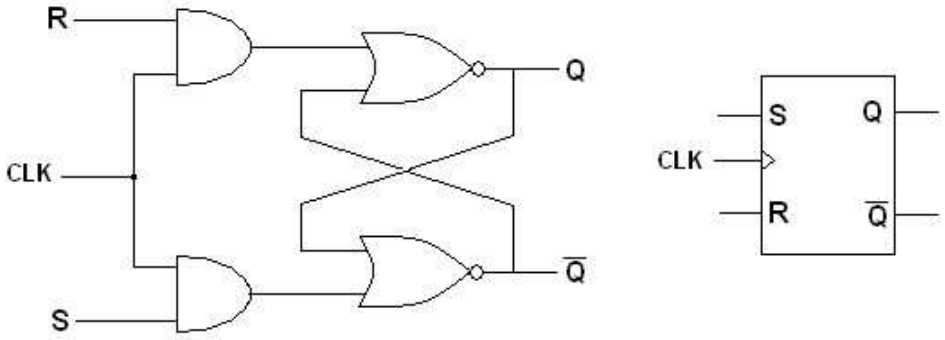


Figure 2.16: Basic circuit and symbol of Clocked RS Flip-Flop

2.5.2 D Flip-Flop

D flip-flop is a modification of RS flip-flop. In RS flip-flop, when both the inputs are high, R=1, S=1 and the output becomes unpredictable and this input combination is termed as forbidden. To avoid this situation, the RS flip-flop is modified so that both the inputs cannot be same at the same time. The modified flip-flop is called D flip-flop. Figure 2.17 shows a clocked D flip-flop.

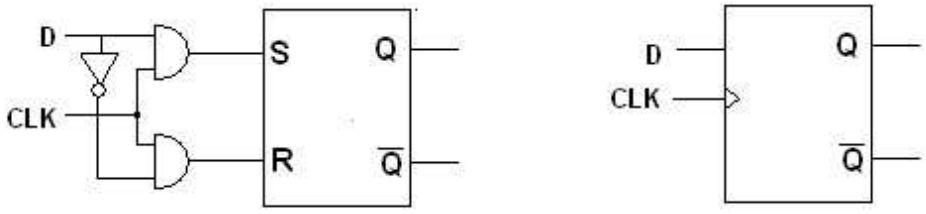


Figure 2.17: (a) Clocked D Flip-Flop (b) Symbol of D Flip-Flop

In D flip-flop both inputs of RS flip-flip are combined together to make it one by a NOT gate so that inputs can not be same at a time. Hence, in D flip-flop there is only one input. The truth table is given in table 2.8.

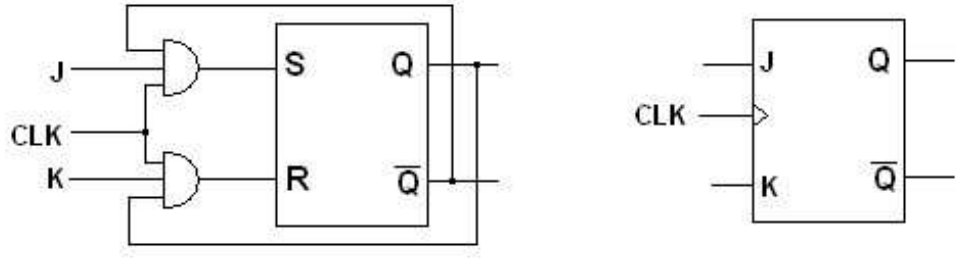
Table 2.8: Truth Table of D Flip-Flop

CLK	D	Q
0	X	Last state
1	0	0
1	1	1

In a clocked D flip-flop the value of D cannot reach the output Q when the clock pulse is low. During a low clock, both AND gates are disabled, therefore, D can change value without affecting the value of Q and this fact is expressed by putting an “X” as D input in the truth table 2.8. On the other hand, when the clock is high, both AND gates are enabled. In this situation, Q is forced to be equal to the value of D. In another way, we can say that in the D flip-flop above, Q follows the value of D while the clock is high. This kind of D flip-flop is often called a D latch.

2.5.3 JK Flip-Flop

In RS flip-flop, the input R= S=1 is called forbidden as it causes an unpredictable output. In JK flip-flop this condition is used by changing the RS flip-flop in some way. In JK flip-flop both input can be high simultaneously and the corresponding toggle output makes the JK flip-flop a good choice to build counter- a circuit that counts the number of +ve or –ve clock edges. Figure 2.18 shows one way to build a JK flip-flop.



(a) (b)

Figure 2.18: (a) JK Flip-Flop (b) Symbol of JK Flip-Flop

Table 2.9: Truth table for JK Flip-Flop

CLK	J	K	Q
X	0	0	Last state
1	0	1	0
1	1	0	1
1	1	1	Toggle

The inputs J and K are called control inputs because their combinations decide what will be the output of JK flip-flop when a +ve clock pulse arrives. When J and K are both low, both the AND gates are disabled. Therefore the CLK pulse has no effect. The first input combination of the truth table shows this and under this case the output Q retains its last state i.e. no change of state.

When J is low, K is high, the upper AND gate is disabled while the lower AND gate is enabled. Hence the flip-flop cannot be set; instead it is reset, i.e. Q=0. This is shown by the second entry in the truth table.

When J is high, K is low the upper AND gate is enabled while the lower one is disabled. So the flip-flop is set there by making Q=1.

When J and K are both high, then the flip-flop is set or reset depending on the previous value of Q. If Q is high previously, the lower AND gate sends a RESET trigger to the flip-flop on the next clock pulse. Then Q becomes equal to 0. On the other hand if Q is low previously, the upper AND gate sends a SET trigger on the flip-flop making Q=1.

So, when J = K = 1, Q changes its value from 0 to 1 or 1 to 0 on the positive clock pulse. This changing of Q to $\overline{Q}$ or to Q is called toggle. Toggle means to change to the opposite state.

Any flip-flop may be driven by +ve as well as –ve clock. As such JK flip-flop can also be driven by positive clock as well as negative clock. Figure 2.19 shows symbol of positive clocked and negative clocked JK flip-flop.

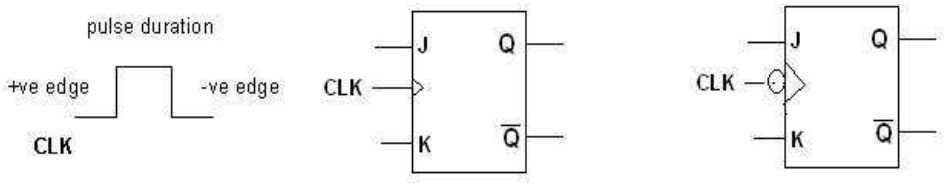


Figure 2.19: Positive and Negative clocked JK Flip-Flop

If a JK flip flop is driven by clock pulse duration, due to the feedback connection as shown in fig 2.18(a), the state of the flip flop i.e. value of Q, may be repeatedly change many times in a single clock pulse. So, it is highly desirable to make the flip flop sensitive to driven by pulse transition not pulse duration.

RC Differentiator Circuit:

The clock pulse applied to a flip-flop is a square wave signal. Clock pulse is used to achieve synchronization of flip-flop operation. To make the synchronization more precise, the square wave pulse is further modified to make it a narrow spike by using a RC differentiator circuit as shown in figure 2.20.

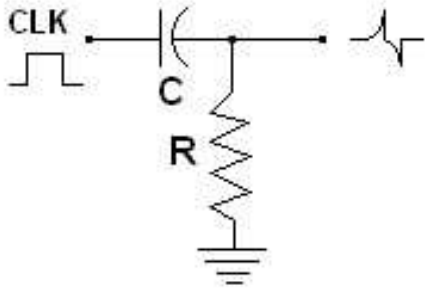


Figure 2.20 : RC Differentiator Circuit

The upper tip of the differentiated pulse is called positive edge and the lower tip is called negative edge. When a flip-flop is triggered by this type of narrow spike, it is called edge triggered flip-flop. If the flip-flop is driven by +ve edge, it is called +ve edge triggered flip-flop. If it is a driven by negative edge, it is called negative edge triggered flip-flop.

Racing

In a flip-flop if the output toggles more than once during a clock edge, then it is called racing. All flip-flop has a propagation delay, means the output changes its state after a certain time period from applying the input and the clock pulse. So, when a flip-flop is edge triggered, then due to propagation delay the output cannot affect the input again because, by that time, the edge of clock pulse has already passed away. If the propagation delay of a flip-flop is 20 ns and the width of the spike is less than 20 ns, then the returning Q and $\overline{Q}$ arrive too late to cause false triggering.

2.5.4 MS Flip-Flop

MS flip-flop is another way to avoid racing. Figure 2.21 shows one way to build MS flip-flop using two JK flip-flop, one of which is positive edge triggered and the other is negative edge triggered. The first JK flip-flop is master and the later is called slave. The master responds to its J and K inputs at the positive edge. J=1, K=0, the master sets on the positive clock edge. The high Q output of the master drive the j input of the slave. So, at the negative clock edge, the slave also sets, copying the action of the master.

When j=0, K=1, master resets at +ve clock edge and the slave resets of the –ve clock edge.

When j=K=1 master toggles at +ve clock edge, and the slave toggles at the –ve clock edge.

Hence whatever master does, the slave copies it.

MS flip-flop is a very popular flip-flop in industry due to its inherent resistance to racing. Hence it is extensively used to build counters.

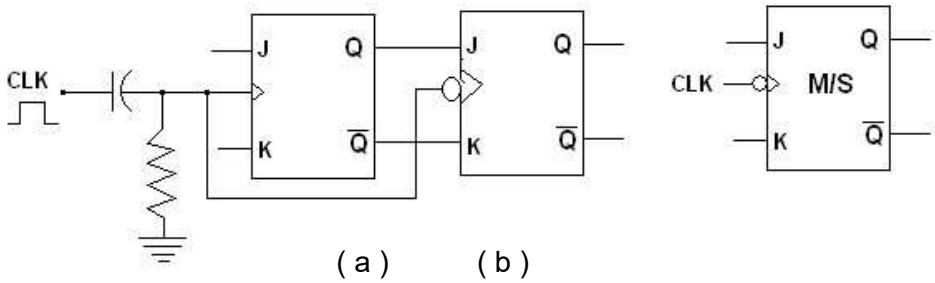


Figure 2.21: (a) Edge triggered JK MS Flip-Flop
(b) Symbol of JK MS Flip-Flop

CHECK YOUR PROGRESS

Q11. A flip-flop is basically a _____

(a) mono-stable multi-vibrator (b) a stable multi-vibrator

(c) bi-stable multi-vibrator (d) none of these

Q12. JK flip-flop has the specialty in _____

(a) fast response time (b) toggle property

(c) spike shaped clock input (d) preset input

Q13. In MS flip-flop the master changes state _____

(a) after the slave (b) with the slave at the same time

(c) before the slave (d) never

2.6 COUNTER

A counter is one of the most useful sequential circuits in a digital system. A counter driven by a clock can be used to count the number of clock cycles. Since the clock pulse has a definite time period, the counter can be used to measure time, the time period or frequency.

There are basically two types of counter— *Synchronous counter* and *asynchronous counter*.

Counters are constructed by using flip-flops and other logic gates. If the flip-flops are connected serially then the output of one flip-flop is applied as input to the next flip-flop. Therefore, this type of counter has a cumulative setting time due to propagation delay. Counter of this type are called serial or asynchronous counter. These counters have speed limitation.

Speed can be increased by using parallel or synchronous counter. Here, flip-flops are triggered by a clock at a time and thus the setting time is equal to the propagation delay of a single flip-flop. But this type of synchronous counters require more hardware and hence it is costly.

Combination of serial or parallel counter is also done to get an optimize solution of speed and hardware/cost. If each clock pulse advances the contents of the counter by one, it is called up counter. If the content of the counter goes down at each clock pulse, it is called down counter.

Before operation, some time it is required to reset all the flip-flops to zero. It is called “Clear”. Some time, it is required to set the flip-flops. It is called preset. To do these, two extra inputs are there in every flip-flop called CLR and PR.

2.6.1 Asynchronous Counter

When the output of a flip-flop is used as the clock input for the next flip-flops it is called asynchronous counter.

Asynchronous counters are also called ripple counter because flip-flop transitions ripple through from one flip-flop to the next in sequence until all flip-flop reach a new state.

A binary ripple counter can be constructed by using clocked JK flip-flop. Figure 2.22(a) shows three MS JK flip-flops connected in series. The clock drives flip-flop A. The output of A drives B and the output of B drives C. J and K inputs of all the flip-flops are connected to positive to make them equal to 1. Under this condition each flip-flop will change state (toggle) with a negative transition at its clock point.

In the counter shown in Figure 2.22 (a) the flip-flop A changes its state at the negative edges of the clock pulses. Its output is applied to the B flip-flop as its clock input.

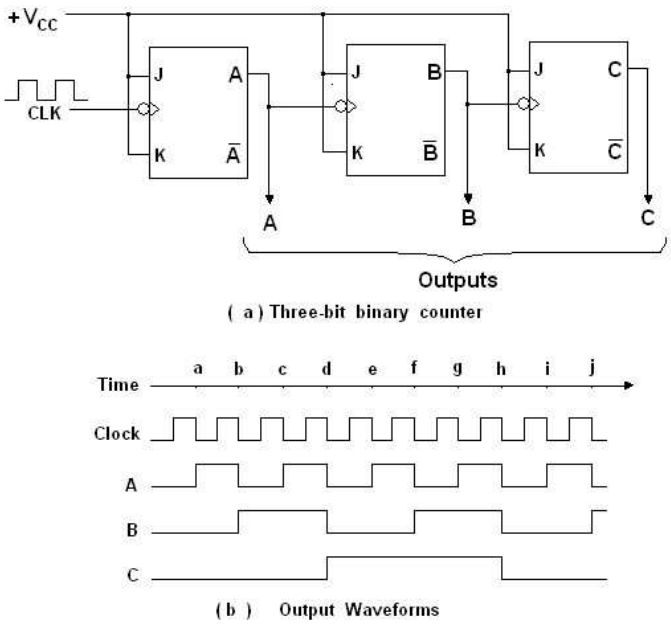


Figure 2.22: Three Bit Binary Counter

The output of B flip-flop toggles at the negative edges of the output of A flip-flop. Similarly output of B flip-flop is used as clock input of the C flip-flop and therefore C toggles at the negative edges of the output of B flip-flop. We can see that triggering pulses move through the flip-flops like a ripple in water.

The wave form of the ripple counter is shown in Figure 2.22(b). It shows the action of the counter as the clock runs. To understand the wave form, let us assume that the counter is cleared before the operation. The A output is assumed as the list significant bit(LSB) and C is the most-significant-bit (MSB). Hence, at very beginning the contents of the counter is CBA=000.

Flip-flop A changes its state to 1 after the negative pulse transition. Thus, at point a on the time line, A goes high. At point b, to it goes low, at c it goes back to high and so on.

Now, output of A acts as clock input of B. So, each time the output of A goes low, flip-flop B will toggle. Thus, at point b on the time line, B goes high, at point d it goes low, and toggles back high again at point f and so on.

Since B acts as the clock input for C, each time the output of B goes low, the C flip-flop toggles. Thus, C goes high at point d on the time line, it goes back to low again at point h.

We can see that the output wave form of A has half the frequency of the clock input wave. B has half the frequency than that of A and C has half the frequency than that of B.

We can further see that since the counter has 3 flip-flops cascaded together, it progresses through 000—— 001——010—-011—100—101—110—111 as its CBA output. After CBA= 111, it starts the cycle again from CBA=000. One cycle from 000—111 takes 8 clock pulses, as it is evident from the wave form as well as from the truth table.

2.6.2 Synchronous Counter

An asynchronous counter or ripple counter has limitation in its operating frequency. Each flip-flop has a delay time which is additive in asynchronous counter. In synchronous counter the delay of asynchronous counter is overcome by the use of simultaneous applications of clock pulse to all the flip-flops. Hence, in synchronous counter, the common clock pulse triggers all the flip-flops simultaneously and, therefore, the individual delay of flip-flop does not add together. This feature increases the speed of

synchronous counter. The clock pulse applied can be counted by the output of the counter.

To build a synchronous counter, flip-flops and some additional logic gates are required. Figure 2.23 shows a three stage synchronous or parallel binary counter along with its output wave forms and truth table. Here the J and K inputs of each flip-flop is kept high and, therefore, the flip-flops toggle at the negative clock transition at its clock input. From figure 2.23 we can see that the output of A is ANDed with CLK to drive the 2nd flip-flop and the outputs of A, B are ANDed with CLK to drive the third flip-flop. This logic configuration is often referred to as “steering logic” since the clock pulses are steered to each individual flip-flop.

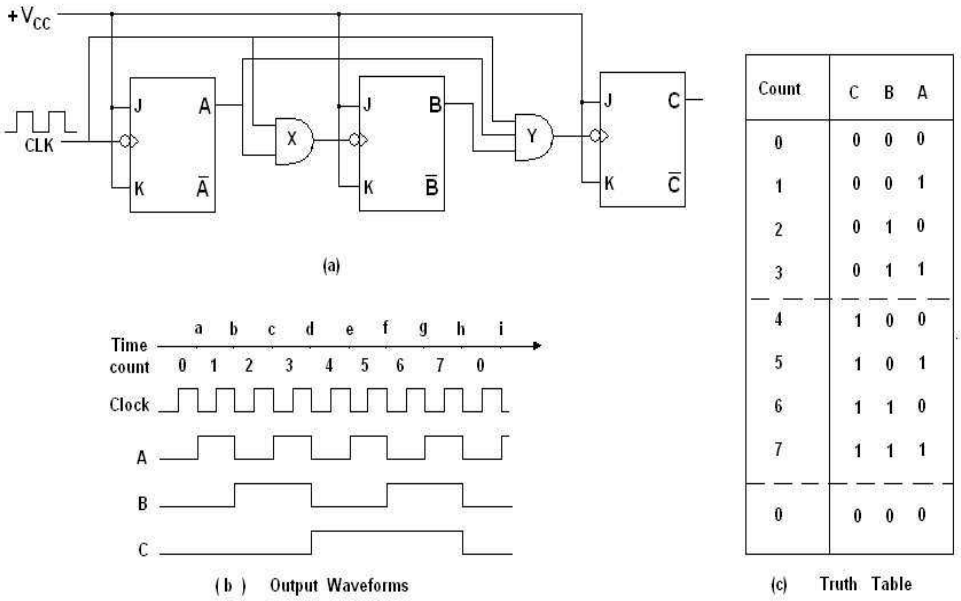


Figure 2.23: Parallel Binary Counter

The AND gate Y is enabled only when both A and B are high and it transmits the clock pulses to the clock input of the 3rd flip-flop. The 3rd flip-flop toggles state with every fourth negative clock transition at d and h on the time line.

The wave form and the truth table show that the synchronous counter progresses upward in a natural binary sequence from 000 to 111. The total count from 000 to 111 is 8 and hence this counter can also be called MOD-8 counter, in count up mode.

CHECK YOUR PROGRESS

Q14. State True or False:

a) Counters are non sequential digital circuits.

b) Asynchronous counters are faster in operation than synchronous counters.

c) The natural progression of a counter is called MODE.

d) Counters can be used to build digital clock.

2.7 REGISTER

A number of flip-flop connected to store binary number is called a Register. The number to be stored is entered or shifted into the register and also taken out or shifted out as per necessity. Hence, registers are also known as shift register.

Registers are used to store data temporarily. Registers can be used to perform some important arithmetic operations like complementation, multiplication, division etc. It can be connected to form counters, to convert serial data to parallel and parallel to serial data.

Types of registers: According to shifting of binary number different types of registers are:

- Serial In—Serial Out(SISO)
- Serial In –Parallel Out (SIPO)
- Parallel In –Serial Out(PISO)
- Parallel In –Parallel Out(PIPO)

2.7.1 Serial In – Serial Out Register (SISO)

Figure 2.24 shows a typical 4 bit SISO register using flip-flops. Here the content of the register is named as QRST. Let us consider that all flip-flops are initially reset. Hence at the beginning QRST= 0000. Let us consider a binary number 1011 which we want to store in the SISO register.

At time A: A 1 is applied at the D input at the first flip-flop. At the negative edge of the CLK pulse, this 1 is shifted into Q. The O of Q is shifted into R, O of R is shifted into S and O of S is shifted into T. The output of flip-flop just after time A is QRST=1000.

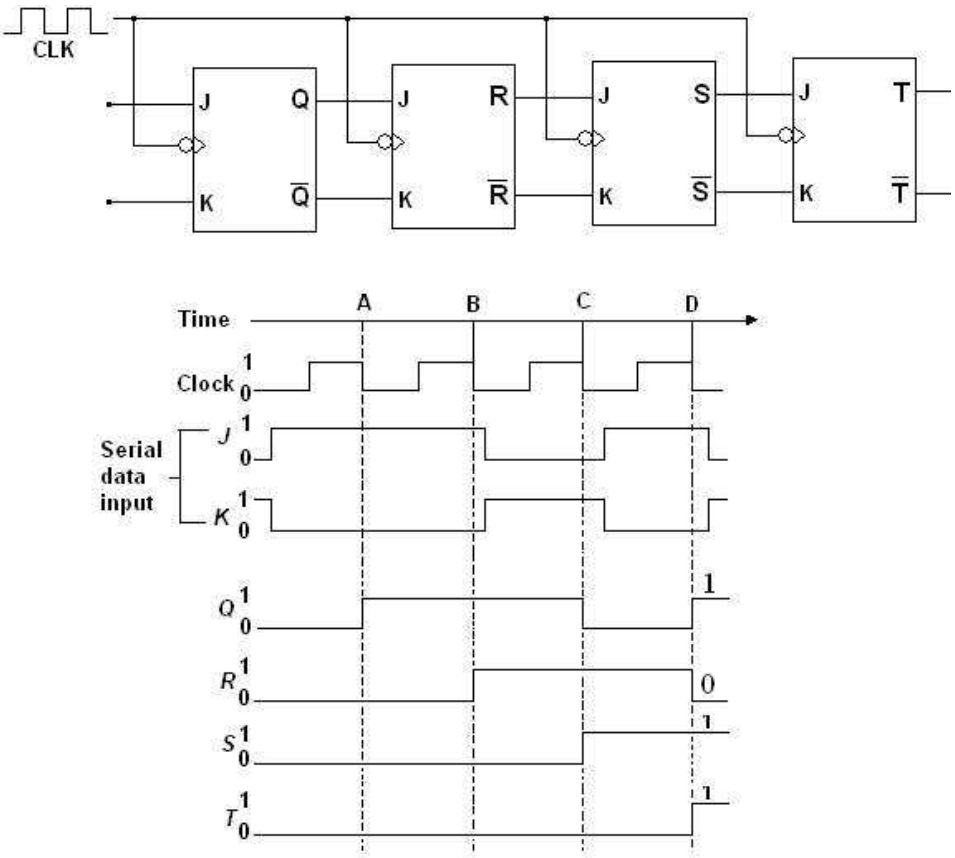


Figure 2.24: 4 Bit Serial In – Serial Out Shift Register

At time B: Another 1 is applied in the data input of the first flip-flop. So at the negative CLK edge, this 1 is shifted to Q. The 1 of Q is shifted in R.,O of R shifted in S, O of S is shifted into T. so, at the end of time B the output of all the flip-flops is QRST=1100.

At time C: A O(zero) is applied in the D input of the 1st first flip-flop. At the negative CLK edge, this O shifts to Q. The 1 of Q shifts into R.,1 of R shifts into S, 1 of R shifts into S, O of S shifts into T. Hence, the output becomes QRST=0110.

At time D: 1 is applied to D input of the first flip-flop. So this 1 shifts into Q at the negative transmission of CLK. The previous O of Q shifts into R, the 1 of R shifts into S, the 1 of S shifts into T. Hence at the end of time D, the registers contains QRST=1011.

In the above steps, using 4 CLK pulses, we have shifted a 4 bit binary number 1011 in the register in a serial fashion.

To take out this binary number serially, we need another 4 CLK pulses and 4 O inputs into D pin of the first flip-flop. The binary number leave the register serially through the T pin of the last flip-flop.

2.7.2 Serial In—Parallel Out (SIPO)

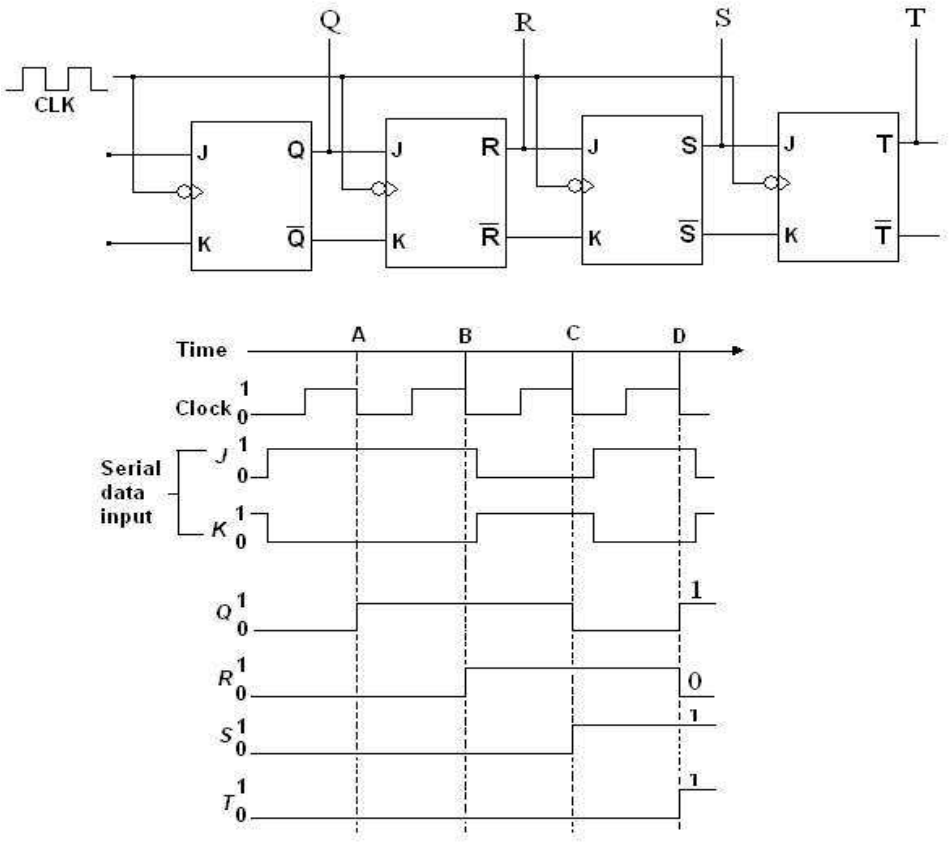


Figure 2.25: 4 Bit Serial In – Parallel Out Shift Register

In this type of shifts register, data are entered serially into the register and once data entry is completed it can be taken out parallelly. To take the data parallelly, it is simply required to have the output of each flip-flop to an output pin. All other constructional features are same as Serial In—Serial Out (SISO) register.

The shifting of data into SIPO is same as SISO registers. In the SIPO of Figure 2.25, a binary number, say, 1011 would be shifted just like the manner as described in the previous section. It would take 4 CLK pulses to complete the shifting. As soon as shifting is completed, the stored binary number becomes available in the output pins QRST. SIPO register is useful to convert serial data into parallel data.

2.7.3 Parallel In-Serial Out Register (PISO)

PISO register takes data parallel i.e. data are stored at a time and shifts data serially. Commercially available TTL IC for PISO is 54/74166. To understand the functional block diagram of 54/74166 we should first understand the following-

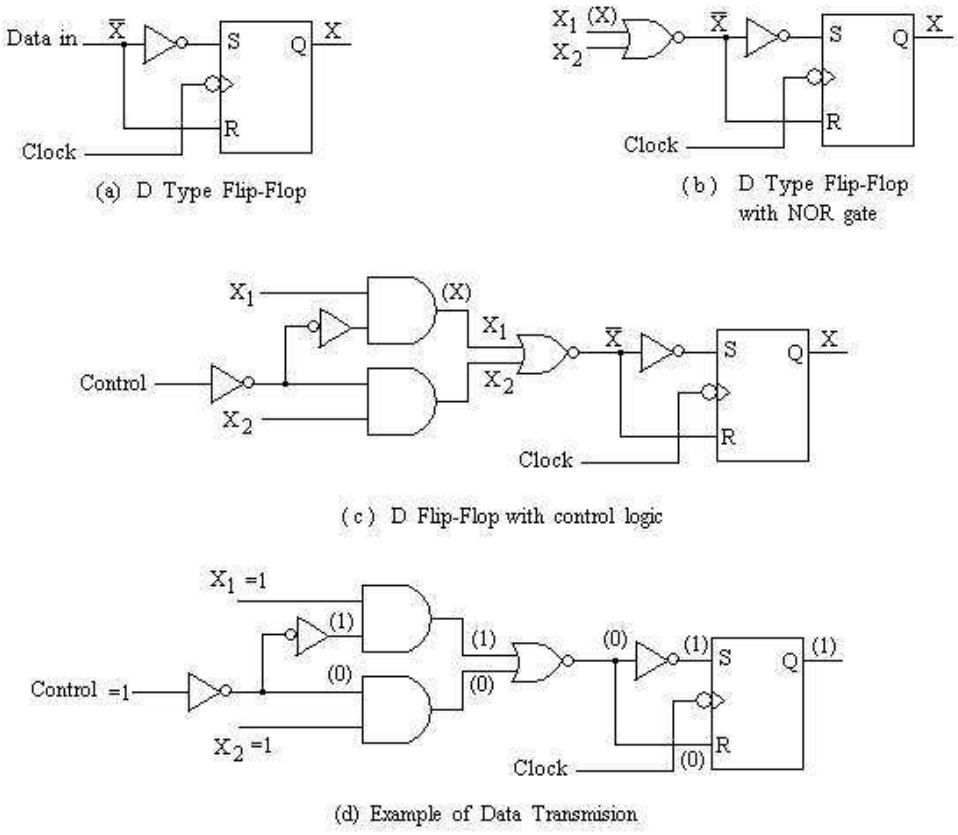


Figure 2.26: Building Block of Parallel In-Serial Out Register

Figure 2.26(a) is a clocked RS flip-flop, which is converted to D flip-flop by a NOT gate. The output of the flip-flop is 1 if Data IN ($\overline{X}$) is 0. Next add a NOR gate as in Figure 2.26(b). Here, if X_2 is at ground level, X_1 will be inverted by the NOR gate. As for example, if =1, then output of the NOR gate will be =0, thereby a 1 will be applied as S input into the flip-flop. This NOR gate allows entering data from two sources, either from or . To shift into the flip-flop, is kept at ground level and to shift into the flip-flop, is kept at ground level. Here, ground level connection implies a 0(zero) input.

Now in Figure 2.26(c) two AND gates and two NOT gates are added. These will allow the selection of data or data . If the control line is high, the upper AND gate is enabled and the lower AND gate is disabled. Thus, the data will enter at the upper leg of the NOR gate and at the same time the lower leg of the NOR gate is kept at ground. Opposite to this, if the control is low, the upper AND gate is disabled and the lower AND gate is enabled. So will appear at the lower leg of the NOR gate and during this time the upper leg of the NOR leg gate is kept at ground level.

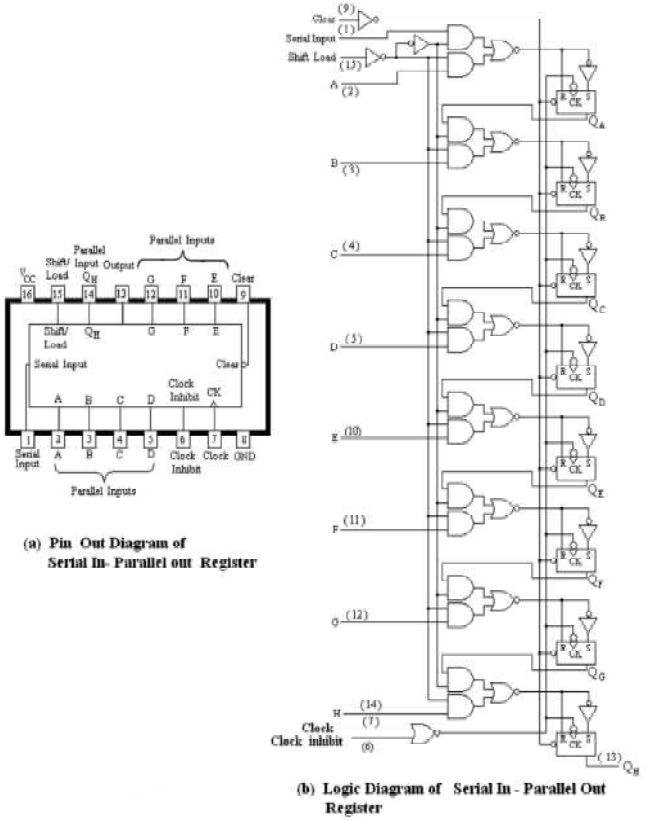


Figure 2.27: Circuit of 54/74166 (a) Pin Out Diagram of Serial In-Parallel Out Register (b) Logic Diagram of Serial In-Parallel Out Register

If we study the figure 2.27 of PISO we see that circuit of figure 2.26 (a) is repeated 8 times to form the 54/74/66 shift register. These 8 circuits are connected in such a style that it allows two operations: (1) The parallel data entry and (2) shifting of data serially through the flip-flop from Q_A toward Q_B

If figure 2.27 the X_2 input of figure 2.26(c) is taken out from each flip-flop to form 8 inputs named as ABCDEFGH to enter 8 bit data parallely to the register. The control is named here as SHIFT/LOAD which is kept

low to load 8 bit data into the flip-flops with a single clock pulse parallely. If the SHIFT/LOAD is kept high it will enable the upper AND gate for each flip-flop. If any input is given to this upper AND gate then a clock pulse will shift a data bit from one flip-flop to the next flip-flop. That means data will be shifted serially.

2.7.4 Parallel In –Parallel Out Register (PIPO)

The register of Figure 2.27 can be converted to PIPO register simply by adding an output line from each flip-flop.

The 54/74198 is an 8 bits such PIPO and 54/7459A is a 4 bit PIPO register. Here the basic circuit is same as Figure 2.26 ©. The parallel data outputs are simply taken out from the Q sides of each flip-flop. In Figure 2.28 the internal structure of 54/7459A is shown.

When the MODE CONTROL line is high, the data bits ABCD will be loaded into the register parallely at the negative clock pulse. At the same time the output is available $Q_A Q_B Q_C Q_D$. When the MODE CONTROL is low, then the left AND gate of the NOR gate is enabled. Under this situation, data can be entered to the register serially through SERIAL INPUT. In each negative transition, a data bit shifted serially from Q_A to Q_B, from Q_B to Q_C and so on. This operation is called right-shift operation.

With a little modification of the connection, the same circuit can be used for shift-left operation. To operate in shift-left mode, the input data is to be entered through D input pin. It is also necessary to connect Q_D to C, Q_C to B, Q_B to A as shown in Figure 2.28. MODE CONTROL line is high to enter data through the D input pin and each stored data bits of flip-flops will be shifted to left flip-flop on each negative clock transition. This is serial data and left shift operation.

To clock inputs—clock1 and clock2 is used here to perform shift right and shift left operation..

Hence 54/7495A can be used as Parallel In –Parallel Out shift register as well as shift right and shift left register.

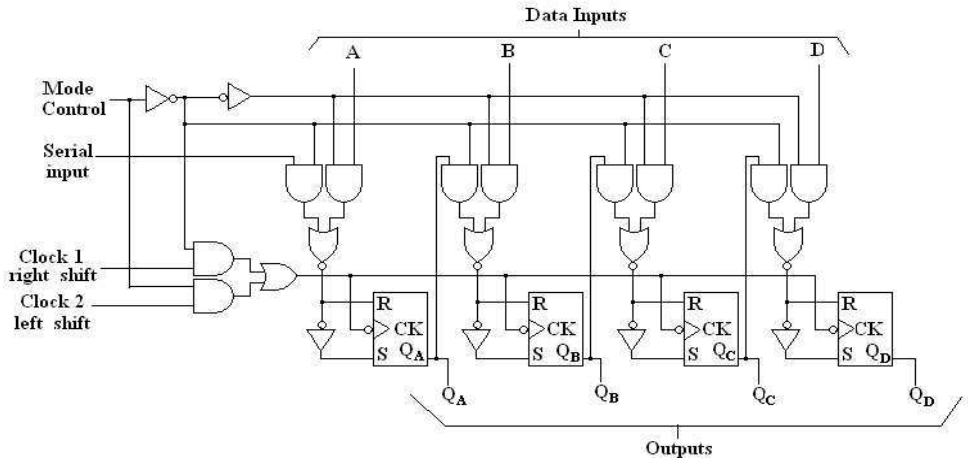


Figure 2.28: Parallel In- Parallel Out Shift Register

CHECK YOUR PROGRESS

Q15. Shift registers are_____.

a) basically a synchronous circuit

b) an asynchronous circuit

c) permanent memory

d) none of these

Q16. In SIPO _____.

a) data enters parallely and leaves serially

b) data enters serially and leaves serially

c) data enters serially and leaves parallely

d) data enters parallely and leaves parallely

2.8 LET US SUM UP

- Digital circuits are of two categories - combinational and sequential
- A combinational circuit is some combinations of logic gates as per specific relationship between inputs and outputs.

- Adder and subtractor circuits can perform binary addition and subtraction.
- A multiplexer is a combinational circuit which selects one of many inputs.
- Demultiplexer is opposite to a multiplexer.
- An encoder generates a binary code for 2^n input variables.
- A decoder decodes an information received from n input lines and transmits the decoded information to maximum outputs.
- A sequential circuit's output depends on past output and present inputs.
- A flip-flop is basically a single cell of memory which can store either 1 or 0.
- Sequential circuits use flip-flop as their building block.
- There are many types of flip-flop viz RS, D, JK, MS flip-flop.
- A counter is a sequential circuit that can count square waves give as clock input. There are two types of counters- asynchronous and synchronous counter.
- Shift registers are also sequential circuit which are used to store binary bits. They are of four different types - Serial In- Serial Out, Serial In- Parallel Out, Parallel In- Parallel Out and Parallel In-Serial Out register.

2.9 FURTHER READINGS

1) Mano, M. M. (2006). Computer systems architecture.
2) Mano, M. (1979). Digital logic. Computer Design. Englewood Cliffs Prentice-Hall.
3) Talukdar, P.H. Digital Techniques. N. L. Publications.
4) Lee, S. C. (1976). Digital circuits and logic design.
5) Leach, D. P., & Malvino, A. P. (1994). Digital Principles and Applications. Glencoe/McGraw-Hill.

2.10 ANSWER TO CHECK YOUR PROGRESS

Ans to Q No 1: (b)

Ans to Q No 2: (c)

Ans to Q No 3: (a)

Ans to Q No 4: (b)

Ans to Q No 5: (c)

Ans to Q No 6: (d)

Ans to Q No 7: (b)

Ans to Q No 8: (a)

Ans to Q No 9: (b)

Ans to Q No 10: (a)

Ans to Q No 11: (c)

Ans to Q No 12: (b)

Ans to Q No 13: (c)

Ans to Q No 14: (a) False (b) False (c) True (d) True

Ans to Q No 15: (a)

Ans to Q No 16: (c)

2.11 MODEL QUESTIONS

Q1. Distinguish between combinational circuit and sequential circuit.

Q2. With truth table and logic diagram explain the working of a full-adder circuit.

Q3. With truth table and logic diagram explain the working of a full-subtractor circuit.

Q4. What is a multiplexer? With diagram explain the working of a 8-to-1 multiplexer.

Q5. Explain the principle of an encoder. Draw a decimal-to-BCD encoder.

Q6. What are the differences between asynchronous and synchronous counter? Draw a MODE-8 counter and explain its working principle.

Q7. Draw logic diagram with output wave form of a 4-bit Serial In-Parallel Out shift register for an input of 1101. Explain its operation.

Q8. Why is square wave clock pulse converted to a narrow spike to be used for flip-flops? Draw a RC differentiator circuit to convert a square wave into a narrow spike.

Q9. What is racing? To get rid of racing what techniques are used?

Q10. What is a magnitude comparator? Draw a block diagram and the function table of the magnitude comparator SN 7485.

UNIT 3: DATA REPRESENTATION

UNIT STRUCTURE

3.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn all the data representation techniques
- describe the negative number representation techniques
- describe computer arithmetic and their implementation
- describe control and data path including data path components
- describe design of ALU and Data Path

3.2 INTRODUCTION

In the previous unit, combinational circuits, sequential circuits along with counters and registers were covered in detail. In this unit, we will discuss all the data representation systems such as Decimal Number System, Binary Number System, BCD, Octal Number System, Hexadecimal Number System, ASCII etc. We will also discuss the negative number representation methods used in binary system. The concepts of design of ALU, Data Path as well as Control Unit Design are also covered in this unit. In the next unit, we will explore the different types of instructions and addressing modes.

3.3 DATA REPRESENTATION

We use decimal number system to represent data. Decimal number system is one where the base is ten. The range of base of decimal number is from 0 to 9.

In digital computer there is a special number system to represent the data. This number system is called as Binary Number system.

There are some other number systems, namely:

- Octal number (Base is 8)
- Hexadecimal number (Base is 16)

3.3.1 Decimal Number System

The symbols of Decimal number system are 0,1,2,3,4,5,6,7,8,9.There are ten symbols called digits. The order of the least-significant digit (right-most digit) is 10^0 (units or ones), the second right-most digit is of the order of 10^1 (tens), the third right-most digit is of the order of 10^2 (hundreds), and so on. For example,

$468=4x10^2 + 6x10^1+8x10^0$

3.3.2 Binary Number System

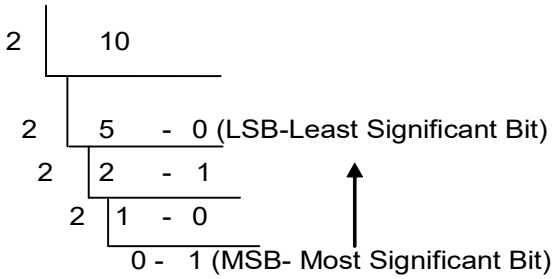
The base of binary number system is 2 as this number system follows two valued logic such as YES or NO, HIGH or LOW, ON or OFF, TRUE or FALSE. To represent two valued logic binary number system uses either 1 or i.e. the symbols of binary number system are 0,1. These 0 or 1 is called bit. Four numbers of bits are called nibble, and 8 numbers of bits together are called byte.

Conversion Technique:

From Decimal number to Binary Number:

Example 3.1: $(10)_{10}= (?)_2$

Answer:

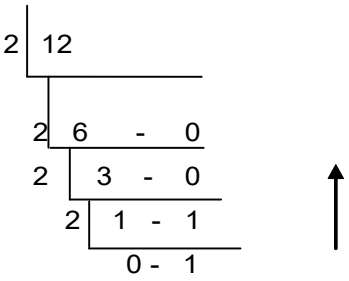


Therefore, $(10)_{10}= (1010)_2$

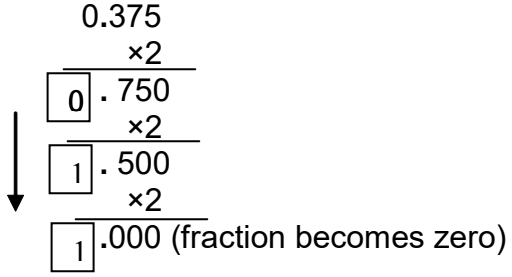
Example 3.2: $(12.375)_{10}= (?)_2$

Answer:

1st step:



2nd step:



Therefore $(0.375)_{10}= (0.011)_2$ and $(12.375)_{10}= (1100.011)_2$

From Decimal number to Binary Number:

The order of the least-significant digit (right-most digit) is 2^0 (units or ones), the second right-most digit is of the order of 2^1 (tens), the third right-most digit is of the order of 2^2 (hundreds), and so on. For example,

$110 = 1x2^2 + 1x2^1 + 0x2^0$

Example 3.3: $(1010)_2 = (?)_{10}$

Answer: $(1010)_2 = 1x2^3+0x2^2+1x2^1+0x2^0$

$= 1X8+0x4+1x2+0x1$

$= 8+0+2+0$

$= (10)_2$

Example 3.4: $(1100.011)_2 = (?)_{10}$

Answer: 1100.011 $=1x2^3+1x2^2+0x2^1+0x2^0+0x2^{-1}+1x2^{-2+}1x2^{-3}$

$= 8+4+0+0+0+.25+.125$

$=12.375$

Therefore, $(1100.011)_2 = (12.375)_{10}$

3.3.3 Octal Number System

The symbols of Octal number system are 0,1,2,3,4,5,6,7.There are eight symbols. The order of the least-significant digit (right-most digit) is 8^0 (units or ones), the second right-most digit is of the order of 8^1 (eight), the third right-most digit is of the order of 8^2 (sixty four), and so on. For example,

$564=5x8^2 + 6x8^1+4x8^0$

Table 3.1 shows the equivalent octal numbers corresponding to their decimal numbers:

Table 3.1 : Decimal and equivalent Octal Number System

DECIMAL NUMBER	OCTAL NUMBER	DECIMAL NUMBER	OCTAL NUMBER	DECIMAL NUMBER	OCTAL NUMBER
0	0	1	1	2	2
3	3	4	4	5	5
6	6	7	7	8	10
9	11	10	12	11	13
12	14	13	15	14	16
15	17	16	20	17	21
18	22	19	23	20	24
21	25	22	26	23	27
24	30	25	31	26	32
27	33	28	34	29	35
30	36	31	37	32	40

…… AND SO ON.

3.3.4 Hexadecimal Number System

The symbols of Hexadecimal number system are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. There are sixteen symbols. The order of the least-significant digit (right-most digit) is 16^0 (units or ones), the second right-most digit is of the order of 16^1 (sixteen), the third right-most digit is of the order of 16^2 (two hundred fifty six), and so on. For example,

$$564=5x16^2 + 6x16^1+4x16^0$$

Table 3.2 shows the equivalent hexadecimal numbers corresponding to their decimal numbers:

Table 3.2 : Decimal and equivalent hexadecimal number system

DECIMAL NUMBER	HEXADE XIMAL NUMBER	DECIMAL NUMBER	HEXADE XIMAL NUMBER	DECIMAL NUMBER	HEXADE XIMAL NUMBER
0	0	1	1	2	2
3	3	4	4	5	5
6	6	7	7	8	8
9	9	10	A	11	B
12	C	13	D	14	E
15	F	16	10	17	11
18	12	19	13	20	1A
21	1B	22	1C	23	1D
24	1E	25	1F	26	14
27	15	28	16	29	17
30	18	31	19	32	2A
33	2B	34	2C	35	2D
36	2E	37	2F	38	20
39	21	40	22	41	23

…AND SO ON.

3.3.5 Binary Coded Decimal (BCD)

Binary Coded Decimal (BCD) is a system for coding a decimal number in which each digit of a decimal number is represented individually by its binary equivalent. Since there are 10 digits in decimal number system, so we need 4 bits to represent each of these (0 to 9) decimal numbers in binary. BCD consists of four number of binary digits. For example, equivalent binary number of decimal number 2 is 10 and equivalent BCD is 0010. Some more examples of BCD are as follows:

Decimal number	Binary Number	BCD
5	101	0101
7	111	0111
8	1000	1000
9	1001	1001

From decimal number 0 to 9 there is nothing to observe for finding equivalent BCD other than four binary bits. From decimal number 10, there is a vital point to observe regarding the equivalent BCD. The binary equivalent of 10 is 1010 and though the binary equivalent of 10 consists of four numbers of binary bits, it does not represent the equivalent BCD. Because at decimal 10, there are two decimal digits (1 and 0) where each decimal digit may be converted to their equivalent BCD as follows:

$(10)_{10}$ = 1 0

= 0001 0000

It will violate the basic principle of BCD number system if the equivalent BCD of $(10)_{10}$ contains eight numbers of binary bits.

3.3.6 American Standard Code for Information Interchange (ASCII)

ASCII (/'æski/ ASS-kee), abbreviated from American Standard Code for Information Interchange, is a character encoding standard. **ASCII** codes represent text in computers, telecommunications equipment, and other devices.

3.3.7 Negative Number Representation in Binary System

There are three techniques to represent negative number in binary number system, namely:

1. Sign Magnitude Method
2. Signed 1s' Complement Method
3. Signed 2s' Complement Method

3.3.7.1 Sign Magnitude Method

In this method 0 is used as the sign of positive number and 1 is used as the sign of negative number. The procedure to find the equivalent binary number for a negative decimal number is as follows:

i. Find the equivalent binary number for the decimal number

ii. Put 0 as the sign bit (the MSB, bit number 8) if the number is positive, otherwise 1

iii. Separate the sign bit from the other bits by putting a comma (,)

For example,

$(10)_{10}$=0, 0001010

$(-10)_{10}$=1, 0001010

● 3.3.7.2 Signed 1s' Complement Method

The procedure to find the equivalent binary number for a negative decimal number is as follows:

i. Find the equivalent binary number for the decimal number

ii. Replace each 0 of the equivalent binary number for the said decimal number by 1 and each 1 by 0

iii. Separate the sign bit from the other bits by putting a comma (,)

For example,

$(10)_{10}$=0, 0001010

$(-10)_{10}$=1, 1110101

● 3.3.7.3 Signed 2s' Complement Method

The procedure to find the equivalent binary number for a negative decimal number is as follows:

i. Find the equivalent binary number for the decimal number by applying 1s' complement method.

ii. Add 1 with the result getting from 1s' complement method to get 2's complement.

iii. Separate the sign bit from the other bits by putting a comma (,)

For example,

$(10)_{10}$=0, 0001010

$(-10)_{10}$=1, 1110101

1

1, 1110110

CHECK YOUR PROGRESS

Q1. What dose BCD stand for ?

Q2. $(10)_{10} = (?)_{16}$

Q3. Base of Octal Number is.....

Q4. Can we find out the equivalent binary number for a negative decimal number?

3.4 COMPUTER ARITHMETIC AND THEIR IMPLEMENTATION

A. Addition of Binary Number

1. Add $(111)_2$ with $(101)_2$

```
  111
  101        1+1 =0, Carry 1
 1100        1+0+Carry 1=0, Carry 1
             1+1+ Carry 1=1, Carry 1
```

2. Add $(1011)_2$ with $(110)_2$

```
 1011
  110       1+0=1
10001       1+1=0, Carry 1
            0+1+Carry 1=0, Carry 1
            1+Carry 1 =0, Carry 1
```

B. Subtraction of Binary Number

1. Subtract $(101)_2$ from $(111)_2$

```
  111
 -101       1-1 =0
  010       1-0 =1
            1-1 = 0
```

2. Subtract $(110)_2$ from $(1011)_2$

1011
-110 1-0=1
101 1-1=0
0-1+Borrow 1=1 [after borrowing 1, it becomes 10, i.e. 2, so (2-1)=1.]

3. Subtract $(1011)_2$ from $(1101)_2$

1101
- 1011 1-1=0
0010 0-1+Borrow 1=1
0-0=0
1-1=0

C. Addition of Octal Number

1. Add 473 with 645

473
645 3+5 =8
1340 =10
=0+ Carry 1
7+4+ Carry 1=12
=14
=4+Carry 1
4+6+ Carry 1=11
=13
=3+Carry 1

2. Add 123 with 567

123
567 3+7=10=12=2+Carry 1
712 2+6+Carry 1=9=11=1+Carry 1
1+5+Carry 1=7

D. Subtraction of Octal number

1. Subtract 473 from 645

```
  645
  473        5-3=2
  ---
  152        4-7+ Borrow 1=12-7=5
             [In Octal Number system Borrow 1 value is 8]
             5-4=1
```

2. Subtract 123 from 511

```
  511
  123        1-3+Borrow 1=9-3=6
  366        0-2+Borrow 1=8-2=6
  ---
             4-1=3
```

E. Addition of Hexadecimal number

1. Add 473 with ADC

```
  4 7 3
  ADC        3+C              =3+12=15
  -----                       =5+ Carry 1
1 5 B 5      7+D+ Carry 1     =7+13+Carry 1
                               =21
                               =1B
                               = B + Carry 1
             4+A+ Carry 1     =4+10+Carry1
                               =15
                               =5+Carry 1
```

2. Add 123 with 567

123	
567	3+7=10=A
68A	2+6=8
	1+5=6

F. Subtraction of Hexadecimal number

1. Subtract 473 from 645

645	
473	5-3=2
1D2	4-7+ Borrow 1=20-7=13=D
	[In Hexadecimal Number system Borrow 1 value is 16]
	5-4=1

2. Subtract 123 with 511

511	
123	1-3+Borrow 1=17-3=14=E
3EE	0-2+Borrow 1=16-2=14=E
	4-1=3

3.5 CONTROL AND DATA PATH

To perform the functional characteristics in a CPU, there are basically two sections, one is *control section* and another one is *data section*. Control unit, which is the only element in the control section, is responsible for providing different types of control signals to the data section where data section is nothing but the data path. A CPU basically consists of different types of registers (such as Accumulator, Temporary Register, Instruction Register, Flag Register, General Purpose Register, Program Counter,

Stack Pointer, Interrupt Block etc.), ALU, Memory unit, Control unit. There are some buses for interconnecting the different components of CPU. The registers, the ALU, and the interconnecting buses are collectively referred to as the **data path**. Each bit in data path is functionally identical. The data path is capable of performing certain operations on data items.

3.6 DATA PATH COMPONENTS

We already know that there are some buses for interconnecting the different components of CPU. Now the question is what is Bus? Collection of wires or distinct lines is called Bus. There are three types of bus, namely, Address Bus, Data Bus and Control Bus.

Data are stored in main memory against a memory location. Address bus is responsible for carrying the address of the main memory location from where the data can be accessed.

Data buses are used for transmission of data. Control Bus is used for providing different types of control signals, to indicate the direction of data transfer and to coordinate the timing of events during the transfer.

The other important components of data paths are as follows:

Accumulator: It is a very special register. Irrespective of the location, the immediate input data will be available at Accumulator. And after an arithmetic operation, the result will be available at the Accumulator.

PC (Program Counter): It is responsible for holding the address of the instruction to be executed next.

IR (Instruction Register): It stores or holds the instruction which is current or executing.

Instruction Cache: 'Fast' memory where the next instruction comes from Reg [index].

Arithmetic Logic Unit (ALU): It performs all the arithmetic and logical operations.

Data Cache: Data read from or written to 'fast' memory.

Multiplexer: Multiplexer has many Inputs and single out put line. So, we can say that multiplexer is responsible for selecting one output from multiple inputs based upon control signal(s).

Single-Cycle Data Path: Each instruction executes in one clock cycle.

Multi-Cycle Data Path: Each instruction takes multiple clock cycles.

Types of elements in the Data path are:

1. State element:
   - i. A memory element, i.e., it contains a state e.g., program counter, instruction memory
2. Combinational element:
   - i. Elements that operate on values e.g., adder, ALU
3. Now, we will look at data path elements required by the different classes of instructions such as
   - i. Arithmetic and logical instructions
   - ii. Data transfer instructions
   - iii. Branch instructions

CHECK YOUR PROGRESS

Q5. To perform the functional characteristics of a CPU, basically how many types of section are there?

Q6. How many types of buses are there in a computer?

Q7. What is Program Counter?

Q8. By whom Data read's from or written to 'fast' memory?

Q9. Multiplexer has ..... inputs and ..... output line

Q10. In Hexadecimal Number system Borrow 1 value is .....

Q11. The registers, the ALU, and the interconnecting buses are collectively referred to as the.....

3.7 DESIGN OF ALU AND DATA PATH

The ALU (Arithmetic-Logic Unit, or Patterson and Hennessy's data path) performs all the arithmetic and logical operations. There is a communication path provided by data buses between ALU and storage elements. The ALU processor is normally composed of single element (one-bit) processors (primarily adders). These single element processors are assembled together to form a desired bit number processor. Some other processing elements such as a shifter are also included to it. An example of Bit Slicing is shown in the figure below for MIPS computer:

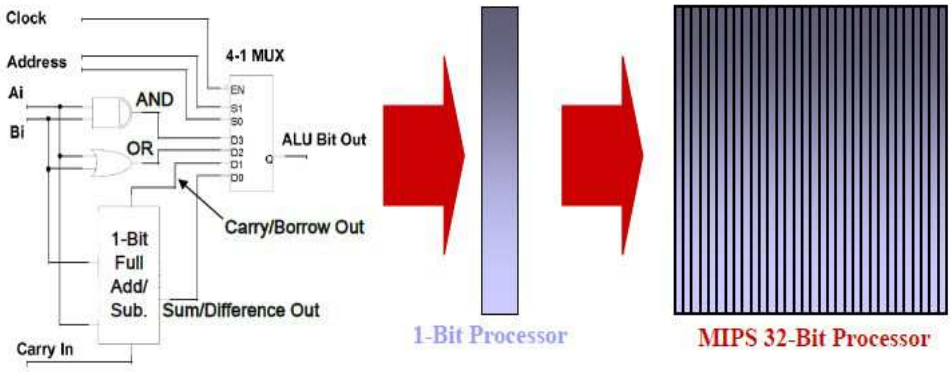


Figure 3.1: Bit Slicing for MIPS computer

ALU needs some components such as:

1. Registers to store arguments and results
2. Buses to carry data from registers to the ALU and results back to the register unit
3. Two memory access units, with associated buses
4. An instruction fetch unit to get instructions from computer memory as needed. This includes a program counter, which always points to the address of the next instruction to be accessed.
5. A second path to memory to obtain data to be used in the program and to store data back into memory as required.
6. A control unit that tells the ALU what to do.

The typical ALU is associated with the other block of a typical CPU is shown below in figure 3.2.

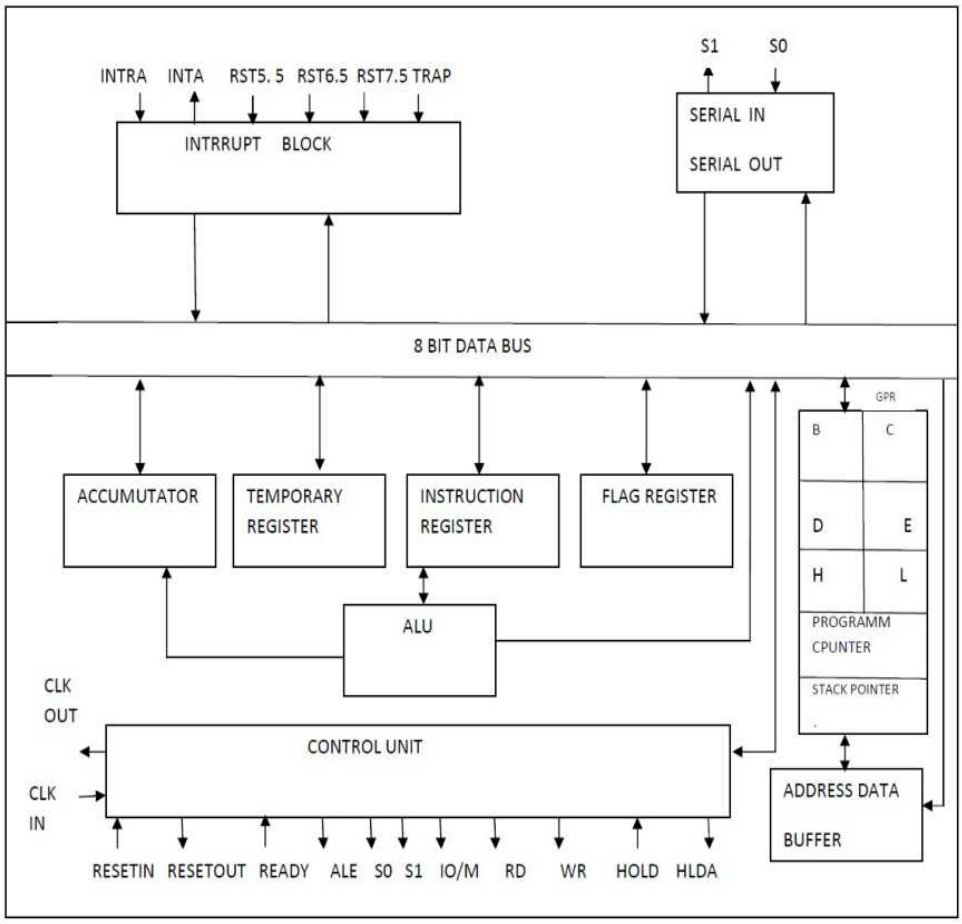


Figure 3.2: A typical ALU is associated with the other block of a typical CPU

The design of data path with ALU is distinctly realized from the above figure. ALU is connected with:

1. Accumulator. Data can transmit from ALU to Accumulator.
2. Instruction Register
3. Flag Register
4. And with the internal data bus.

Design of ALU is based on the points given below:

1. Instruction for a particular operation will come from instruction register directly.
2. Data on which operation is to be performed i.e. the operand will come to ALU from General Purpose Register (GPR) (or from memory location).

3. Either arithmetic or logical operation, which one is required, will be done by the different types of circuit (eg. Adder, Subtractor, Multiplier, Level Shifter, All logic gates etc.) of which ALU consists.
4. After an operation result the result will be available at Accumulator.
5. After an arithmetic operation done by ALU, different types of information about the result such as whether the result is positive number or negative number, whether the result has some value other than zero, whether there is some carry that has occurred during the operation or not etc. will be given by the Flag Register.

3.8 CONTROL UNIT DESIGN

Control unit is responsible for providing different types of control signals to the data section. The figure 3.2 shows that the control unit is directly connected to the Internal Data Bus. The design of a control unit demands a connection with crystal oscillator to produce suitable clock frequency for the processor. The other signals to be supplied by the control unit are as follows:

RESET IN: RESET signal is one by the application of which a digital circuit gets its proper output according to the status of its Input. The CPU is held at reset condition as long as RESET is supplied.

RESET OUT: To indicate the reset status of the CPU, it is used.

READY: To indicate whether the peripheral device is ready to give response, READY is used by the processor.

ALE: ALE stands for Address Latch Enable. It is used to demultiplex the address and data buses which are in multiplexed mode.

S0, S1: These are used to indicate different types of operation as shown below:

S0	S1	operation
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

IO/M: IO stands for Input/output and M stands for memory. IO is a high enable signal whereas M is a low enable signal. If IO/M=1 then Input/output will be selected for data transmission, otherwise memory device.

RD: RD means READ operation.

WR: WR means WRITE operation.

HOLD: Say, another device is requesting for the address and data buses.

To indicate this condition HOLD signal is used.

HLDA: This signal is for acknowledging the HOLD signal.

CHECK YOUR PROGRESS

Q12. To demultiplex the address and data buses which are in multiplexed mode, which control signal is used?

Q13. To indicate the reset status of the CPU, which control signal, is used?

3.9 LET US SUM UP

- In digital computer there is a special number system to represent the data. This number system is called the Binary Number system
- There are some other number systems, namely:
    - Decimal (Base is 10)
    - Octal number (Base is 8)
    - Hexadecimal number (Base is 16)
- There are three techniques to represent negative number in binary number system, namely:
    - Sign Magnitude Method
    - Signed 1s' Complement Method
    - Signed 2s' Complement Method
- The registers, the ALU, and the Interconnecting Buses are collectively referred to as the data Path.
- There are three types of bus, namely: Address bus, Data bus and control bus.

- Address Bus: Data are stored in the main memory against a memory location. Address bus is responsible for carrying the address of the main memory location from where the data can be accessed.
- Data Bus: Data buses are used for transmission of data.
- Control Bus: Control Bus is used for providing different types of control signals, to indicate the direction of data transfer and to coordinate the timing of events during the transfer.
- The ALU (Arithmetic-Logic Unit, or Patterson and Hennessy's data path) performs all the arithmetic and logical operations.
- ALU is connected with :
    - Accumulator. Data can transmit from ALU to Accumulator.
    - Instruction Register
    - Flag Register
    - And with the internal data bus.
- Control unit is responsible for providing different types of control signals to the data section such as RESET IN, RESET OUT, HOLD, HLDA, READY, ALE, IO/M, R/W etc.

3.10 FURTHER READINGS

1) Mano, M. M. (2006). Computer systems architecture.
2) Hamacher, V. C., Vranesic, Z. G., Zaky, S. G., Vransic, Z., & Zakay, S. (1984). Computer organization (Vol. 3). New York et al.: McGraw-Hill.

3.11 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: Binary Coded Decimal

Ans to Q No 2: A

Ans to Q No 3: 8

Ans to Q No 4: Yes

Ans to Q No 5: Two

Ans to Q No 6: Three

Ans to Q No 7: Program Counter holds the address of the instruction to be executed next.

Ans to Q No 8: Data Cache

Ans to Q No 9: Many, Single

Ans to Q No 10:16

Ans to Q No 11: Data Path

Ans to Q No 12: ALE

Ans to Q No 13: RESET OUT

3.12 MODEL QUESTIONS

Q1. Find the following:

i. $(134.56)_{10} = (?)_2$

ii. $(101101.11)_2 = (?)_{10}$

iii. $(543)_{10} = (?)_{16}$

Q2. Find the following:

i. Add $(101)_2$ with $(11)_2$

ii. Add $(467)_8$ with $(1231)_8$

iii. Add $(BCF)_{16}$ with $(101)_{16}$

Q3. Find the following:

i. Subtract $(11)_2$ from $(101)_2$

ii. Subtract $(467)_8$ from $(1231)_8$

iii. Subtract $(101)_{16}$ from $(BCF)_{16}$

Q4. Find out the binary equivalent number of (-11) by following all three negative number representation technique of binary number

Q5. Define Data Path. Write all the important components of Data Path.

Q6. What is Bus? Explain in detail.

Q7. Explain the design of an ALU with its data path

Q8. Write about the entire control signal associated with the control unit in detail.

UNIT 4: COMPUTER ARITHMETIC

UNIT STRUCTURE

4.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about integer both signed and unsigned representation
- learn how to represent both fixed point and floating point
- describe signed and magnitude form of integers.
- learn the concept of complements
- learn how to find 1's complement of binary numbers
- calculate 1's complement arithmetic
- learn how to find 2's complement and its arithmetic.

4.2 INTRODUCTION

Computer arithmetic is a branch of computer engineering that deals with methods of representing integers and real values in digital systems and

efficient algorithms for manipulating such numbers by means of hardware circuits or software routines. On the hardware side, various types of adders, subtractors, multipliers, dividers, square-rooters, and circuit techniques for function evaluation are considered. Both abstract structures and technology-specific designs are dealt with. Software aspects of computer arithmetic include complexity, error characteristics, stability, and certifiability of computational algorithms.

Computer arithmetic is a subfield of digital computer organization that deals with the hardware realization of arithmetic functions which has a major thrust to design of hardware algorithms and circuits to enhance the speed of numeric operations. It encompasses the study of number representation, algorithms for operations on numbers, implementations of arithmetic units in hardware, and their use. The basic application of computer arithmetic is in the general purpose systems for fast primitive operations for processor data paths. On the other hand, it has also applications on special purpose systems which are given below:

i) Signal and image processing
ii) Real-time 3D graphics
iii) HDTV, image compression
iv) Network processors (data compression, encryption/decryption)

In this unit we will learn about basic numeric operations. Here we will see how two integer numbers are represented in different format and do arithmetic operations on both unsigned and signed integers that a digital computer system understand.

4.3 INTEGER REPRESENTATION

There are basically two types of integer representation in digital computer system. They are:

i) Unsigned integer representation.
ii) Signed integer representation.

We have already learn how to convert an unsigned integer from one base to another. For example, 25_{10} decimal equivalent to 11001_2 binary numbers. So, for unsigned integer, it is very simple and easy to convert in

digital computer system. But for signed integer its needs some additional calculations.

Signed numbers require additional issues to be addressed. When an integer variable is declared in a program, many programming languages automatically allocate a storage area that includes a sign as the first bit of the storage location. By convention, a “1” in the high-order bit indicates a negative number. The storage location can be as small as an 8-bits (byte) or as large as several words, depending on the programming language and the computer system. The remaining bits (after the sign bit) are used to represent the number itself. How this number is represented depends on the method used.

In a computer system that uses signed-magnitude representation and 8 bits to store integers, 7 bits can be used for the actual representation of the magnitude of the number. This means that the largest integer an 8-bit word can represent is $2^7 – 1$ or 127 (a zero in the high-order bit, followed by 7 ones). The smallest integer is 8 ones, or –127. Therefore, N bits can represent $–2^{(N–1)} – 1$ to $+2^{(N–1)} – 1$.

We know that for n-bit number, the range for natural number is from 0 to $2^n –1$. There are three different schemes or methods to represent signed (negative) and unsigned integer. These are

i) Signed and Magnitude form.
ii) 1’s complement form.
iii) 2’s complement form

4.3.1 Signed Integer Representation

We know that for n-bit number, the range for natural number is from 0 to $2^n–1$.

For n-bit, we have all together different combination, and we use these different combination to represent numbers, which ranges from 0 to $2^n–1$.

If we want to include the negative number, naturally, the range will decrease. Half of the combinations are used for positive number and other half is used for negative number.

For n-bit representations, the range is from:

$$-2^{n-1} - 1 \text{ to } +2^{n-1} - 1$$

For example, if we consider 8-bit number, then range for natural number is from 0 to 255.

But for signed integer the range is from –127 to +127.

4.3.2 Fixed-Point Representation

Let us take an example to show and make clearly how binary numbers is represented in fixed-point.

We know that binary representation of 41.6875 is 101001.1011.

To store this number, we have to store two part of information, the part before decimal point and the part after decimal point. This is known as *fixed-point representation* where the position of decimal point is fixed and number of bits before and after decimal point are also predefined.

If we use 16 bits before decimal point and 8 bits after decimal point, in signed magnitude form, the range is from $-2^{16} - 1$ to $+2^{16} - 1$ and precision is $2^{(-8)}$. One bit is required for sign information, so the total size of the number is 25 bits.

[1(sign) + 16 (before decimal point) + 8 (after decimal point)]

1 bit (Sign bit)	16 bits (before decimal point)	8 bits (after decimal point)

Figure 4.1: Fixed-Point Representation

4.3.3 Floating-Point Representation

In digital computers, floating-point numbers consist of three parts: (i) a Sign bit, (ii) an Exponent part (representing the exponent on a power of 2), and (iii) a fractional part called a Significant (which is a fancy word for a Mantissa). For example a floating point representation is given below which a 14 bits model consists of 1 bit as sign bit, 5 bits as Exponent part and 8 bits as Mantissa part.

1 bit (Sign bit)	5 bits (Exponent)	8 bits (Mantissa)

Figure 4.2: Floating-Point Representation

In this representation, numbers are represented by a Mantissa comprising the significant digits and an Exponent part of Radix R.

The format is: *Mantissa* * $R^{Exponent}$

Numbers are often normalized, such that the decimal point is placed to the right of the first non zero digit.

For example, the decimal number, 5236 is equivalent to $5.236*10^3$

To store this number in floating point representation, we store 5236 in Mantissa part and 3 in Exponent part.

4.4 SIGNED MAGNITUDE FORM

In signed magnitude form, one particular bit is used to indicate the sign of the number, whether it is a positive number or a negative number. Other bits are used to represent the magnitude of the number.

For an n-bit number, one bit is used to indicate the signed information and remaining (n–1) bits are used to represent the magnitude. Therefore, the range is from $–2^{n–1}–1$ to $+ 2^{n–1}–1$.

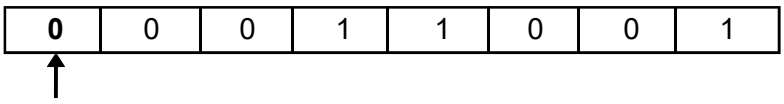
Generally, Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 in signed bit indicates positive number and 1 in signed bit indicates negative number.

For example, 01011001 represents +169

but 11011001 represents –169

For example, if in a computer, the word size is 1 byte (8 bits), then, +25 will be represented as follows:

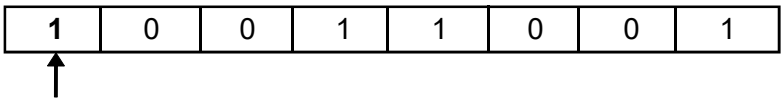
Number in binary notation as binary equivalent of 25_{10} is 11001



MSB (0 for + sign)

–25 will be represented as follows:

Number in binary notation as binary equivalent of 25_{10} is 11001



MSB (1 for - sign)

4.5 THE CONCEPT OF COMPLEMENT

Complement Systems Number theorists have known for hundreds of years that one decimal number can be subtracted from another by adding the difference of the subtrahend from all nines and adding back a carry. This is called taking the nine's complement of the subtrahend, or more formally, finding the diminished radix complement of the subtrahend. The advantage that complement systems give us over signed magnitude is that there is no need to process sign bits separately, but we can still easily check the sign of a number by looking at its high-order bit.

The concept of complements is used to represent signed number.

Let us consider a number system of base-r or radix-r. There are two types of complements.

i) The radix complement or the r's complement.

ii) The diminished radix complement or the (r–1)'s complement.

4.5.1 Diminished Radix Complement

Given a number N in base r having n digits, the (r–1)'s complement of N is defined as $(r^n – 1) – N$.

For decimal numbers, r = 10 and r – 1 = 9, so the 9's complement of N is $(10^n – 1) – N$

For example, 9's complement of 5642 is: $(10^4 – 1) – 5642 = 9999 – 5642 = 4357$

4.6 ONE'S (1'S) COMPLEMENT

It is important to note at this point that although we can find the nine's complement of any decimal number or the one's complement of any binary number, we are most interested in using complement notation to represent negative numbers. We know that performing a subtraction, such as 10–7, can be also be thought of as "adding the opposite'', as in 10 + (–7). Complement notation allows us to simplify subtraction by turning it into addition, but it also gives us a method to represent negative numbers. Because we do not wish to use a special bit to represent the sign (as we did in signed-

magnitude representation), we need to remember that if a number is negative, we should convert it to its complement. The result should have a 1 in the leftmost bit position to indicate the number is negative. If the number is positive, we do not have to convert it to its complement. All positive numbers should have a zero in the leftmost bit position.

4.6.1 One's (1's) Complement Arithmetic

The one's complement form of any number is obtained by simply changing each 0 in the number to 1 and each 1 in the number to 0.

For example, 1's complement of binary number 100010 will be 011101 and for 011011 will be 100100.

Example 3.1: Express 23_{10} and -9_{10} in 8-bit binary one's complement form.

$23_{10} = +(00010111)_2 = 00010111_2$

$-9_{10} = -(00001001)_2 = 11110110_2$

a) If Subtranhend is smaller than the Minuend

In this case we follow the following three simple steps:

Step 1: Find one's complement of the subtrahend.

Step 2: Proceed as in addition

Step 3: Ignored the carry and add 1 to the total end-around-carry.

Example 3.2: Add 23_{10} to -9_{10} using one's complement arithmetic.

Or, Subtract 9 from 23.

Solution:	0 0 0 1 0 1 1 1	(+23)
	+ 1 1 1 1 0 1 1 0	+(–9)
The last carry is added:	0 0 0 0 1 1 0 1	
	+1	
Sum is:	0 0 0 0 1 1 1 0	$+(14)_{10}$

b) If Subtranhend is larger than the minuend

In this case we follow the following three simple steps:

Step 1: Find one's complement of the subtrahend.

Step 2: Proceed as in addition.

Step 3: Complement the result and place a negative sign in front of the result.

Example 3.3: Add 9_{10} to -23_{10} using one's complement arithmetic.

Or, subtract 23 from 9

Solution:

	0 0 0 0 1 0 0 1	(+9)
	+1 1 1 0 1 0 0 0	+(–23)
No carry	1 1 1 1 0 0 0 1	

Now, complementing the result and putting a –ve sign we get,

$0\ 0\ 0\ 0\ 1\ 1\ 1\ 0 = -14_{10}$

The primary disadvantage of one's complement is that we still have two representations for zero: 00000000 and 11111111 i.e., +ve 0 and –ve 0. For this and other reasons, computer engineers long ago stopped using one's complement in favor of the more efficient two's complement representation for binary numbers.

Note that total numbers that can be represented using 1's complement are 2^N-1, where N is the word size (number of bits in a word).

4.7 TWO'S (2'S) COMPLEMENT ARITHMETIC

We know that all negative numbers can be represented in 2's complement form. Subtraction between two numbers can be achieved by adding the 2's complement of the number. Two's complement method represents positive numbers in their true form that means their binary equivalents and negative numbers in 2's complement form.

Now we will discuss how to represent of signed numbers in 2's complement form.

Consider the eight bit number 01011100, 2's complements of this number is 10100100. If we perform the addition between these numbers we have:

	0	1	0	1	1	1	0	0
	1	0	1	0	0	0	1	1
1	0	0	0	0	0	0	0	0

Since we are considering an eight bit number, so the 9^{th} bit (MSB) of the result can not be stored. Therefore, the final result is 00000000.

Since the addition of two number is 0, so one can be treated as the negative of the other number. So, 2's complement can be used to represent negative number.

The 2's complement form of any number is obtained by simply adding 1 to LSB of the 1's complement of that number.

For example 2's complement of 101110 is calculated as follows:

First find the 1's complement of 101110, which is 010001.

Now add 1 with LSB of 010001 is 010010, which is the required 2's complement of 101110.

For example,

i) Using 2's complement representation subtract 20 from 25

Solution: We have $25_{10} = 1\ 1\ 0\ 0\ 1$

Ignore the carry $-20_{10} = 0\ 1\ 1\ 0\ 0$ (2's Complement of 20_{10})

$1\ 0\ 0\ 1\ 0\ 1$

So, answer is +5

ii) Add (–20) and (–25) using Using 2's complement representation

Solution: We have $-25_{10} = 1\ 0\ 0\ 1\ 1\ 1$ (2's Complement of 25_{10})

$-20_{10} = 1\ 0\ 1\ 1\ 0\ 0$ (2's Complement of 20_{10})

Ignore carry $1\ 0\ 1\ 0\ 0\ 1\ 1$

and the result is –(2's complement of 010011)

So, answer is -101101 = - 45

iii) Using 2's complement representation subtract 25 from 20.

Solution: We have $-25_{10} = 0\ 0\ 1\ 1\ 1$ (2's Complement of 25_{10})

$+\ 20_{10} = 1\ 0\ 1\ 0\ 0$

$1\ 1\ 0\ 1\ 1$

As no carry, so the answer is: –(2's complement of 11011)

$= -0\ 0\ 1\ 0\ 1$

$= -5$

Note that total numbers which a word of N bits can present, using 2's complement representation, are 2^N.

CHECK YOUR PROGRESS

Q.1: What are three different schemes or methods to represent signed (negative) and unsigned integer?

Q.2: What is the range of signed integer representation of n bits?

Q.3: What is the format of the floating point representation?

Q.4: What is the signed and magnitude representation of (–20) in 8 bit word length?

Q.5: Find 1's and 2's complement of followings:

i) 100010 ii) 1010110

Q.6: State whether the following statements are true (T) or false (F):

a) For n-bit number, the range for natural number is from 0 to 2^n–1.

b) Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 is used for negative number.

c) The format of floating point representation is The format is: $Mantissa * R^{Exponent}$.

d) The two's complement form of any number is obtained by simply changing each 0 in the number to 1 and each 1 in the number to 0.

e) Given a number N in base r having n digits, the (r–1)'s complement of N is defined as (r^n–1) – N.

4.8 LET US SUM UP

- Computer arithmetic is a branch of computer engineering that deals with methods of representing integers and real values in digital systems and efficient algorithms for manipulating such numbers by means of hardware circuits or software routines.
- There are three different schemes or methods to represent signed (negative) and unsigned integer. These are:

i) Signed and Magnitude form.
ii) 1's complement form.
iii) 2's complement form

- We know that for n-bit number, the range for natural number is from 0 to 2^n–1.
- In digital computers, floating-point numbers consist of three parts: (i) a Sign bit, (ii) an Exponent part (representing the exponent on a power of 2), and (iii) a fractional part called a Significant (which is a fancy word for a Mantissa).
- Generally, Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 in signed bit indicates positive number and 1 in signed bit indicates negative number.
- Consider a number system of base-r or radix-r. There are two types of complements.
  i) The radix complement or the r's complement.
  ii) The diminished radix complement or the (r–1)'s complement
- Given a number N in base r having n digits, the (r–1)'s complement of N is defined as $(r^n–1) – N$ **.**
- The one's complement form of any number is obtained by simply changing each 0 in the number to 1 and each 1 in the number to 0.
- The total numbers that can be represented using 1's complement are $2^N–1$, where N is the word size (number of bits in a word).
- The 2's complement form of any number is obtained by simply adding 1 to LSB of the 1's complement of that number.
- The total numbers which a word of N bits can present, using 2's complement representation, are 2^N.

4.9 FURTHER READING

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI Learning Pvt. Ltd.
2) Hammacher, Carl. (Fifth Edition) (International Edition, 2002). *Computer Organization*. McGrawHill.

3) Rajaraman, V. (Fouth Edition, 2008). *Fundamentals of Computers*.

4) Stallings, William. (2004). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.

4.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: There are three different schemes or methods to represent signed (negative) and unsigned integer. These are

i) Signed and Magnitude form

ii) 1's complement form

iii) 2's complement form

Ans. to Q. No. 2: For n-bit representations of signed integer, the range is from $-2^{n-1}-1$ to $+2^{n-1}-1$

Ans. to Q. No. 3: In the representation of floating point, numbers are represented by a Mantissa comprising the significant digits and an Exponent part of Radix R.

The format is: $Mantissa * R^{Exponent}$

Ans. to Q. No. 4: –20 will be represented as follows:

Number in binary notation as binary equivalent of 20_{10} is 10100

1	0	0	1	0	1	0	0

MSB (1 for - sign)

Ans. to Q. No. 5: The 1's and 2's complement of followings are

i) 1's complement of 100010 is 011101

2's complement of 100010 is 011110

ii) 1's complement of 1010110 is 0101001

2's complement of 1010110 is 0101010

Ans. to Q. No. 6: a) True, b) False, c) True, d) False, e) True.

4.11 MODEL QUESTIONS

Q.1: What are different schemes to represent signed and unsigned integer in computer arithmetic?

Q.2: How is signed integer represented in signed and magnitude form?

Q.3: What do you mean by floating-point representation of any number in digital computer? What are its different parts?

Q.4: What is 1's complement ? Explain with example.

Q.5: What do you mean by 2's complement? Explain how can a negative integer represented using 2's complement concept.

*** ***** ***

UNIT 5: INSTRUCTION SETS

UNIT STRUCTURE

5.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe the elements of machine instructions
- learn about instruction format, representation and types of instructions
- explain different types of operands and operations
- learn about assembly language notations.

5.2 INTRODUCTION

In the previous unit we have learnt the numbers that are represented in the binary number system and the 2's complement system. We have already discussed the fundamental issues related to the arithmetic operations

used to support computations in computer. In this unit, we will focus on the way the programs are executed in a computer from the machine instruction viewpoint.

The program instructions and the data operands are stored in the memory. The instructions are brought from the memory to the processor and then executed to perform a given task. Operands and results must also be moved between the memory and the processor. The tasks carried out by the computer consist of some small steps. The computer must have instructions capable of performing the different types of operations. In this unit, we will discuss about the machine instructions and the different types.

5.3 ELEMENTS OF A MACHINE INSTRUCTION

The operation of a computer is determined by the instruction executed by the CPU. Such instructions are referred to as machine instructions or computer instructions. A collection of such instructions that the computer can execute is called an instruction set.

There are different elements of a machine instruction. These are as follows:

1) Operation Code.

   The operation code specifies the operation to be performed. For example, ADD, MOV, MULT etc. The operation is specified by a binary code.

2) Source operand reference.

   Source operand references are the operands that are the inputs for the operation. i.e., operands on which operation is to be performed.

3) Result operand reference.

   The operation may produce a result.

4) Next instruction reference.

After the execution of an instruction is complete, the Next instruction reference tells where from the next instruction is to be fetched.

Generally, the next instruction immediately follows the current instruction and there is no explicit reference to the next instruction. However, in some cases, the next instruction may be in the main memory or in the

secondary memory. In such cases, there is an explicit reference to the next instruction, and then the main memory or virtual memory address must be supplied.

5.4 INSTRUCTION REPRESENTATION

Each of the machine instruction is represented by a sequence of bits in the computer. The instructions are divided into fields corresponding to the constituent elements of the instruction. The machine instructions are highly machine specific.

An example of an instruction is shown in the figure below. Let us assume that the CPU is a 16-bit CPU. 4-bits are used for the operation code. Thus there can be $2^4 = 16$ different sets of instructions. The instruction has two operands each of 6-bits. Thus, there can be $2^6 = 64$ different operands for each operand reference.

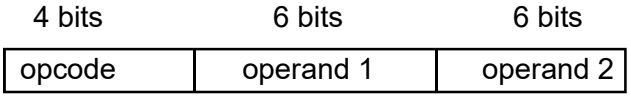


Figure. 5.1: A Simple Instruction Format

To deal with binary representation is not an easy task and so a symbolic representation of the machine instructions is used. The opcodes are represented by abbreviations called *mnemonics*. Common examples are:

ADD	Add
SUB	Subtract
MULT	Multiply
DIV	Division
LOAD	Load data from memory to CPU
STORE	Store data to memory from CPU

Let us take the following machine instruction as an example,

ADD R, A

This instruction means add the value contained in memory location A to the contents of the register R.

5.5 INSTRUCTION TYPES

A computer should have a set of instructions that allows the users to formulate any data processing works. Instructions can be classified according to their operations and address reference.

According to the operations, the instructions can be classified as

a) Data Processing
b) Data Storage
c) Data Movement
d) Control

Data Processing: The data processing instructions include both logic and arithmetic instructions. The logic instructions perform the logical operations on the bits of the word. The arithmetic instructions provide computational capabilities for processing numeric data.

Data Storage: The logical and arithmetic operations are performed on the data in the processor registers. So there must be memory instructions for transferring the data or moving the data from the memory to the processor registers before operation and transfer data from CPU registers to memory after operations.

Data movement: The I/O instructions constitute the data movement instructions. These instructions are needed to transfer the data and the programs between the memory registers and input-output devices.

Control: The test and branch instructions are the control instructions. The test instructions test the value of a word or the status of a computation. The branch instructions are used to branch to a different set of instructions.

5.6 NUMBER OF ADDRESSES

The processor architecture can also be described in terms of the number of addresses contained in each instruction. According to the address reference, the instructions can be classified as three address instructions, two address instructions, one address instructions and zero address instructions.

Three address instruction: Symbolically, the three address instruction can be represented as follows:

ADD A, B, C

Where A, B and C are the variables. A and B are called Source operands and variable C is called destination operand. ADD is the operation to be performed on the operands.

Two address instruction: Symbolically, it can be represented as follows:

ADD A, B

This instruction adds up the contents in A and B and stores the resultant in B overwriting the previous contents of B. Here, A is the source operand and B is both source as well as destination operand.

One address instruction: Symbolically, it can be represented as follows:

ADD A

This instruction adds the contents of A to the contents of the processor register called accumulator and then the result is stored back to the accumulator destroying the previous contents. Another example of a one address instruction is:

LOAD A

It copies the contents of the memory location A into the accumulator.

Zero address instruction: These types of instructions are used in machines that stores operands in a pushdown stack. A stack stores the locations in last-in-first-out order.

CHECK YOUR PROGRESS

Q.1: Fill in the blanks:

i) The ..... are directly executed by the CPU of a computer.

ii) The ..... specifies the operation to be performed.

iii) Machine instruction is represented by a sequence of ..... in the computer.

iv) Instructions can be classified according to their operations and .....

v) The arithmetic instructions provide ..... capabilities for processing numeric data.

vi) According to the address reference, the machine instructions can be classified as ....., ..., ... and ..... instructions.

vii) The ..... and ..... instructions are the control instructions.

5.7 TYPES OF OPERANDS

Every instruction has two parts– *operands* and *operation code* (opcode). Operand is another name for data. There may be different types of operands. There may be different types of operands:

a) Addresses
b) Numbers
c) Characters
d) Logical data

Addresses: sometimes, calculations must be performed on the operand reference in an instruction to determine physical address.

Numbers: The common numeric data types are integer, point and decimal.

Characters: The common form of data for documentation purpose is character strings. Most of the computers use ASCII (American Standard Code for Information Interchange) code for characters represented by 7-bit pattern and 8-bit patterns. The 8^{th} bit may be set to zero or used as a parity bit for error detection. The EBCDIC (Extended Binary Coded Decimal Interchange Code) code is also used to encode characters.

Logical data: The units of data are bit, byte, word or double word. Bit means either a 0 or 1. When data is viewed as a 1-bit data, it is logical data. The logical data are used to store Boolean or binary data items. Using logical data, the bits of data can be manipulated.

5.8 TYPES OF OPERATIONS

A set of instructions are given to the computer to carry out different data processing tasks. The following are the basic types of operations that the machine instructions should be able to carry out.

- Data Transfer
- Data Processing
- Program Sequencing and Control

5.8.1 Data Transfer Operations

The data transfer operations are the most fundamental type of operations. Such transfer operations that are included here are as given below.

a) Data transfer between memory and CPU registers
b) Data transfers between CPU registers
c) Data transfer between processor and input-output devices

These instructions must specify the location of the source and the destination operands. Also the length of the data to be transferred and the mode of addressing must also be specified.

a) Data transfer between memory and CPU registers: The arithmetic and logical operations are performed on the data stored in the processor registers. Thus, before any operation, the data needs to be transferred from the memory to the CPU registers and after the operation, the resultant data has to be transferred from those registers to the memory. The two basic operations that are involved in these memory access are **Load** (Read) and **Store** (Write).

LOAD operation fetches word from memory to the processor. **STORE** operation transfers the data from the processor to the memory. The data or instructions which are transferred between processor and memory may be a byte or a word. The processor has a number of registers.

Example, R2 ⟵ [LOC]

This instruction transfers the contents of memory location to the register R2.

b) **Data transfers between CPU registers:** Data transfers also occur between the CPU registers. If the source and destination are registers, then the processor simply causes data to be transferred from one register to another. This operation is internal to the processor. As for example, R3 ⟵ [R2]

This instruction transfers data from CPU register R2 to the CPU register R3.

c) **Data transfer between processor and input-output devices:** The transfer of data between the input devices and processor and the output devices is called I/O data transfer. For example, R1 ⟵ [DATA IN]

This instruction states that the contents of I/O register, DATA IN are transferred to the processor register R1. DATA IN and DATA OUT are the registers by which the processor reads the contents from the keyboard and sends the data for display.

The rate of output data transfer is much higher than the rate of the input data transfer. However, the speed of a processor is much higher than these and can execute millions of instructions per second. So, to overcome the speed difference between these devices, synchronization mechanism should be used for proper transfer of data between them.

As shown in the figure below, the $\mathbf{S_{IN}}$ and $\mathbf{S_{OUT}}$ status bits are used to synchronize data transfer between display and the processor respectively.

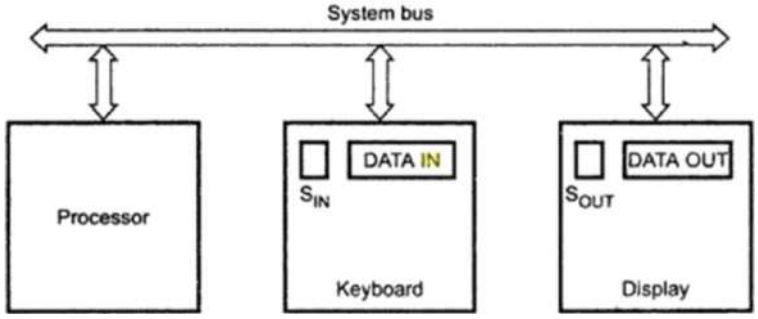


Figure 5.2: Bus Connection for Processor, Keyboard and Display

When a key is pressed, the corresponding character code is stored in DATA IN and the S_{IN} bit is set to 1. When the processor finds the S_{IN} = 1, it reads the contents of DATA IN register. After completing the read operation, the S_{IN} is automatically set to 0. If another key is pressed, the corresponding character is entered in the DATA IN register and the S_{IN} is set to 1 and the process is repeated again.

Similarly, when a character is to be transferred from processor to the display, the DATA OUT register and the S_{OUT} status bits are used. When the processor wants to transfer data to the display unit, the S_{OUT} is set to 1 and after the transfer is completed the S_{OUT} bit is cleared to 0.

5.8.2 Data Processing Operations

Data processing operations include the following types of operations.

- Arithmetic operations,
- Logical operations,
- Shift and rotate operations

The arithmetic operations basically include the following:

Add,
Subtract,
Multiply,
Divide,
Increment,
Decrement,
Negate.

For example, to add the contents of the registers R1 and R2, and to store the results in another register R3, the following instruction is used.

ADD R1, R2, R3

The logical operations include the following:

AND,
OR,

NOT,

XOR,

Compare,

Shift,

Rotate

For example, the expression:

AND R1, R2, R3

States that the contents of the processor registers R1 and R2 are logically AND operated and the result is stored in R3.

The **Compare** operation makes logical or arithmetic comparisons of two or more operands. With a logical shift, the bits of a word are shifted left or right. There are two logical shift instructions: logical shift left and logical shift right.

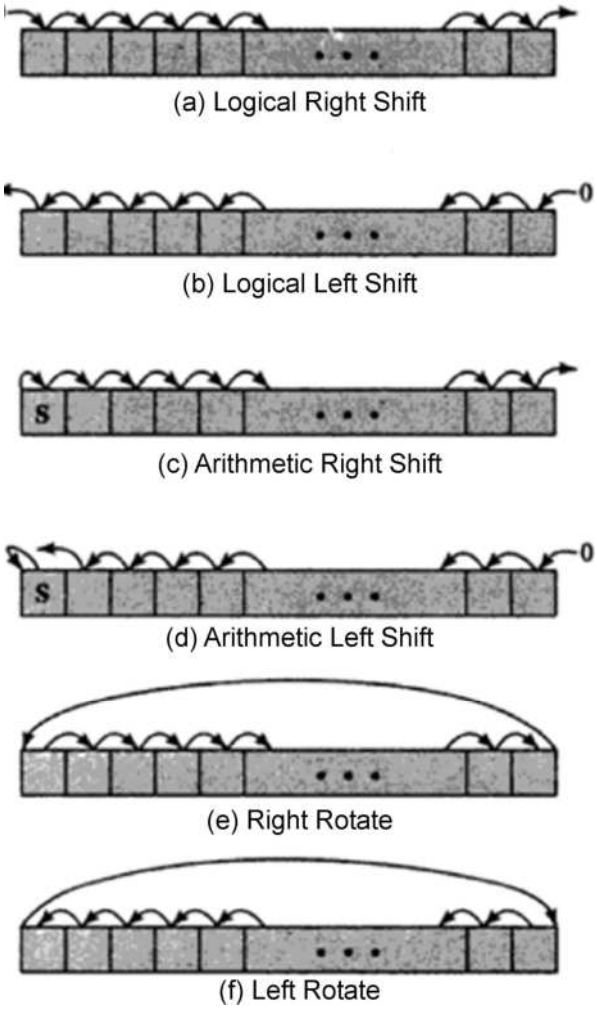


Figure 5.3: Shift and Rotate Operation

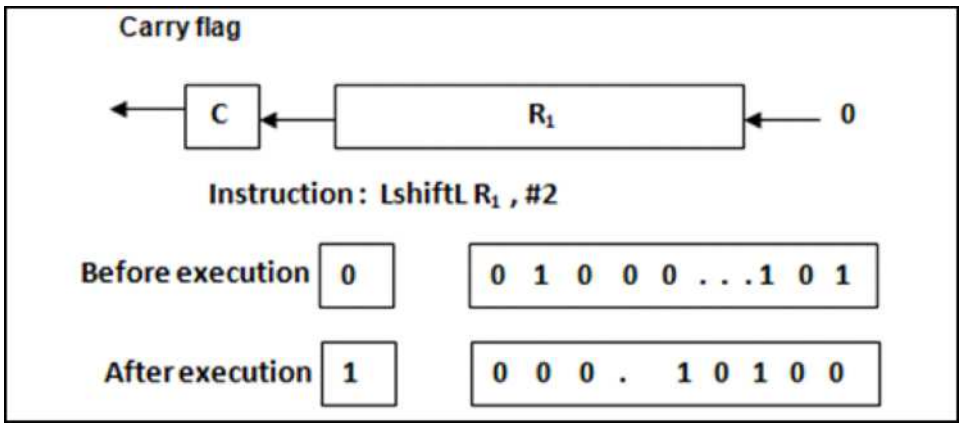
On one end, the bit shifted out is lost. On the other end, a 0 is shifted in. The logical shifts are used for isolating fields within a word. The 0s that are shifted into the word displace the unwanted information which is shifted off from the other end.

The general syntax for logical shift operations are:

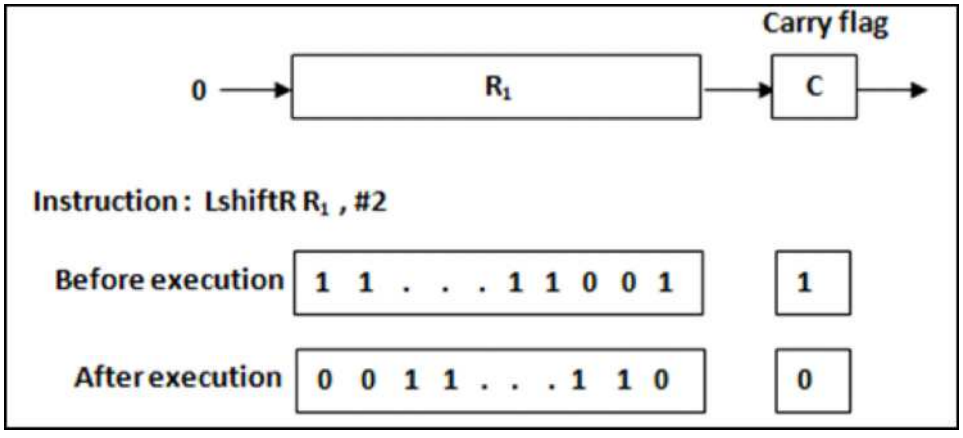
LshiftL dst, count

LshiftR dst, count

The count may be a number or the contents of a processor register. The figure (Figure. 5.4) shows the operation of LshiftL and LshiftR operations.



(a) Example of LShiftL instruction



(b) Example of LShiftR instruction

Figure: 5.4 : LshiftL and LshiftR operations

Rotate is also called cyclic shift operation. In shift operations, the bits shifted out of the operand are lost except for the last bit which is retained in the Carry flag (C). On the other hand, the rotate operation

preserves all the bits being operated on. One of the uses of the rotate operation is to bring each bit successively into the leftmost bit, where it can be identified by testing the sign of the data. There are two basic types of rotate operations– rotate left and rotate right.

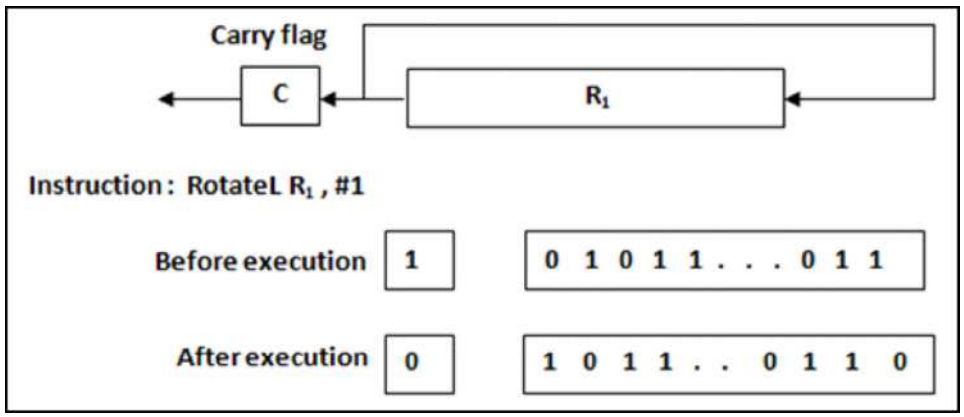


Figure: 5.5: Example of RotateL Instruction

5.8.3 Program Sequencing and Control

The program sequencing and control mainly includes the test and branch operations. Test operations are used to test the value of data and the branch operations are used to branch to a different set of operations depending on the decision made. Programs sometimes need decision making. The computer is asked to perform one thing if one condition is met and another thing if another condition holds.

A branch instruction is also called a jump instruction. One of the operand of a branch instruction is the address of the next instruction to be executed. Most often, the instruction is a conditional branch instruction. That is, a jump is made only if a condition is met otherwise the next instruction in the sequence is executed. If in a branch instruction, the branch is always made, then it is an unconditional branch.

A branch can be either a forward (an instruction with a higher address) or a backward (an instruction with a lower address). The figure below shows how an unconditional and a conditional branch

can be used to create a repeating loop of instructions. The instructions in locations 202 through 210 will be executed repeatedly until the result of subtracting Y from X is 0.

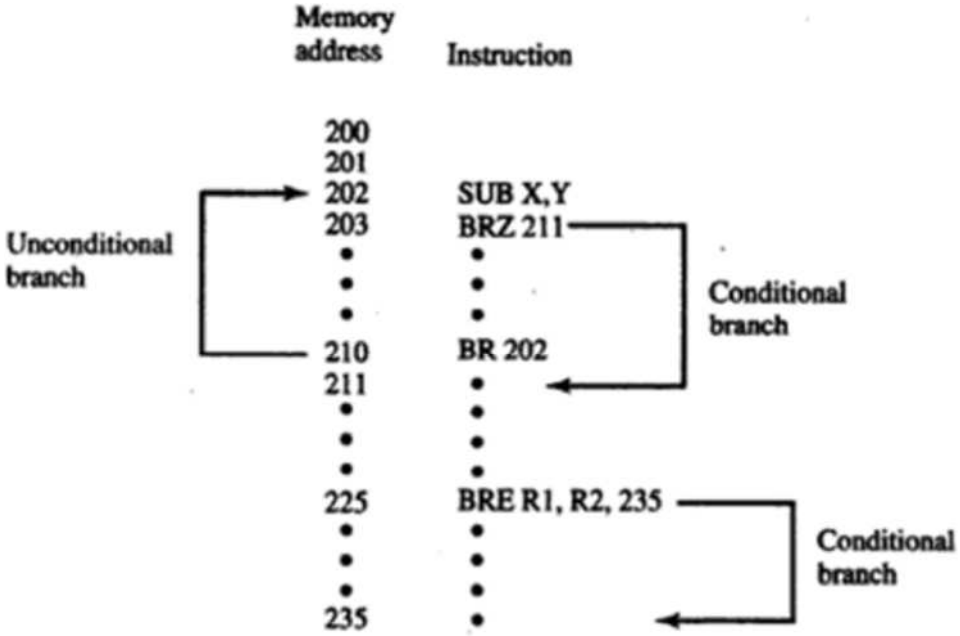


Figure.5.6: Branch Instructions

Skip instructions are also transfer of control instructions. The skip implies that an instruction be skipped and thus the implied address of the skip instruction is equal to the address of the next instruction plus one instruction length.

5.9 INSTRUCTION FORMATS

The CPU fetches the instruction from the memory, decodes it and executes the instruction according to its type. The structure of an instruction is called instruction format. It defines the layout of the bits of an instruction in terms of its constituent fields. The instruction format consists of an operation code or opcode, and zero or more operands. The operands may be implicit or explicit. The explicit operands are referenced using one of the addressing modes discussed next.

An instruction is of various lengths depending upon the number of addresses it contains. The design of an instruction format is complex. The key design issues are discussed below.

Instruction length: The instruction length is affected by the memory size, organization, bus structure, processor speed and processor complexity. The programmers want more opcodes, more operands, more addressing modes and greater address ranges. Shorter programs can be written to do a particular task with more opcodes and more operands. More addressing modes also gives the programmers flexibility in implementing certain functions. These requirements of more number of things requires more bits and so results in longer instructions. However, sometimes longer instructions may be not that useful and be a waste of space.

There are other considerations also beyond the length. Either the length of the instruction should be equal to the memory transfer length or one should be a multiple of the other. A related consideration is the memory transfer rate. The processor speed is very fast and can execute more number of instructions than the number that is fetched from the memory. So a solution to this can be to use a cache memory, another solution can be to use shorter instructions. Thus 16-bit instructions can be fetched at twice the rate of 32-bit instructions but probably can be executed less than twice as fast.

Allocation of bits: Another difficult issue with the design of the instruction format is how to allocate the bits in that format. The trade-offs are complex. There are some factors that can be useful to determine the use of addressing bits.

- Number of addressing modes. Addressing modes may be implicit or explicit.
- Number of operands
- Register versus memory. With registers only a few bits are needed to specify the register. The more registers can be used for operand references, the fewer bits are needed.
- Address range. The range of addresses that can reference memory is related to the number of address bits. As this has a severe limitation, direct addressing is rarely used. With displacement addressing, the range is opened up to the length of the address register.
- Address granularity. Granularity of addresses is another important factor for addresses that references memory rather than registers.

5.10 ASSEMBLY LANGUAGE NOTATION

Assembly language notations are used to represent machine instructions and programs. These notations use assembly language formats. Example of some assembly notations are as follows.

MOVE R1, R2

This assembly language instruction involves the transfer of contents between the registers. The contents of register R1 are transferred to R2. The contents of R1 are unchanged but those of R2 are overwritten.

ADD R1, R2, R3

This expression states that the contents of the registers R1 and R2 are added and the result is stored in the register R3.

The assembly language notations have three fields– operation, source and destination.

CHECK YOUR PROGRESS

Q.2: Fill in the blanks:

i) ..... and ..... are the two parts of an instruction.

ii) When data is viewed as a ..... data, it is logical data.

iii) Logical data are used to store ..... items.

iv) The ..... and ..... operations are performed on the data stored in the processor registers.

v) The two basic operations that are involved in memory access are ..... and .....

vi) ..... Operation transfers the data from the processor to the memory.

vii) The transfer of data between the input devices and processor and the output devices is called ..... transfer.

viii) There are two logical shift instructions called ..... and .....

ix) Rotate is also called ..... operation.

x) A branch instruction is also called a ..... instruction.

5.11 LET US SUM UP

- The operation of a computer is determined by the instruction executed by the CPU. A collection of such instructions that the computer can execute is called an *instruction set*.
- Each of the machine instruction is represented by a sequence of bits in the computer.
- The opcodes are represented by abbreviations called *mnemonics*.
- The machine instructions are represented by a sequence of bits in the computer.
- Instructions can be classified according to their operations and address reference.
- According to the operations, the instructions can be classified as Data Processing, Data Storage, Data Movement, and Control.
- According to the address reference, the instructions can be classified as three address instructions, two address instructions, one address instructions and zero address instructions.
- Every instruction has two parts– operands and operation code (opcode).
- The basic types of operations that the machine instructions should be able to carry out are Data transfer, Data processing and Program sequencing and control.
- Data transfer operations may be:
  - Data transfer between memory and CPU registers,
  - Data transfers between CPU registers,
  - Data transfer between processor and input-output devices.
- The two basic operations that are involved in memory access are Load (Read) and Store (Write).

- Data processing operations includes,
  - Arithmetic operations,
  - Logical operations,
  - Shift and rotate operations.
- The *Compare* operation makes logical or arithmetic comparisons of two or more operands.
- With a logical shift, the bits of a word are shifted left or right.
- Rotate is also called cyclic shift operation.
- In shift operations, the bits shifted out of the operand are lost except for the last bit which is retained in the Carry flag (C).
- Rotate operation preserves all the bits being operated on.
- Test operations are used to test the value of data and the branch operations are used to branch to a different set of operations depending on the decision made.
- A branch instruction is also called a jump instruction. One of the operand of a branch instruction is the address of the next instruction to be executed.
- A branch can be either a forward (an instruction with a higher address) or a backward (an instruction with a lower address).
- Skip instructions are also transfer of control instructions.
- The structure of an instruction is called instruction format. It defines the layout of the bits of an instruction in terms of its constituent fields.
- Assembly language notations are used to represent machine instructions and programs.
- The assembly language notations have three fields namely operation, source and destination.

5.12 FURTHER READING

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI Learning Pvt. Ltd.
2) Gill, D. N. S. & Dixit, J. B. (2008). *Digital Design and Computer Organisation*. Firewall Media.

3) Null, L. & Lobur, J. (2014). *The Essentials of Computer Organization and Architecture*. Jones & Bartlett Publishers.

4) Stallings, William. (2004). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.

5.13 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) Machine instructions, ii) operation code, iii) bits, iv) address reference, v) computational, vi) three address, two address, one address, zero address, vii) test, branch

Ans. to Q. No. 2: i) Operands, operation code; ii) 1-bit; iii) Boolean data; iv) Arithmetic, logical; v) Read, Write; vi) Store; vii) I/O data; viii) logical shift left, logical shift right; ix) cyclic shift; x) jump

5.14 MODEL QUESTIONS

Q.1: What are the elements of a machine instruction? Explain.

Q.2: How can a machine instruction be represented?

Q.3: Explain the various types of operands and operations.

Q.4: What are Shift and Rotate logical operations?

Q.5: Describe the Assembly language notation.

*** ***** ***

UNIT 6: ADDRESSING MODES

UNIT STRUCTURE

6.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about addressing
- describe how addressing is done
- describe different addressing modes for accessing register and memory operands.

6.2 INTRODUCTION

In the previous unit we have learnt what the machine instructions does and their formats and types. We have examined the type of operands and operations that may be specified by machine instructions. Also we have gone through the assembly language notation. In this unit, we will learn about how the address of an operand is specified, and how are the bits of an instruction organized to define the operand addresses and operation of that instruction.

An instruction is a group of bits that instruct the computer to perform a specific operation. A sequence of instructions is called a computer program. An instruction specifies the operation in the operation code field on the data specified by the operands in the operand field. There are different types of addressing capabilities in an instruction format. Addressing is an instruction design issue. There are different addressing modes that allow us to specify where the operands are located. An addressing mode can specify a location in memory, or a register or a constant. There are certain modes that allow shorter addresses, and some allow us to determine the location of the actual operand, also called the effective address of the operand dynamically. The typical addressing modes that are employed in the processor of a computer are discussed in this unit.

6.3 ADDRESSING

The techniques of specifying the address of the data are known as addressing modes. Thus the term addressing refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting the address field of an instruction before the instruction is actually executed. Computers use addressing mode techniques for the following purposes:

a) To reduce the number of bits in the field of instruction,
b) Giving programming versatility to the user by providing facilities such as pointers to memory
c) Specifying rules for modifying or interpreting address field of the instruction.

There are a variety of addressing techniques that are been applied. The most common of them are:

- Immediate
- Direct
- Indirect
- Register
- Register Indirect

The computer architectures provide more than one of these addressing modes. So, there is a question that how the processor can determine which addressing mode is being used in a particular instruction. There are several approaches for this to be done. Often, the different symbols/assembler syntax are used for different addressing modes. Also, one or more bits in the instruction format can be used as a *mode* field.

To explain the addressing modes, we use the following notation:

A = contents of an address field in the instruction that refers to a memory

R = contents of an address field in the instruction that refers to a register

EA = actual (effective) address of the location containing the referenced operand

(X) = contents of location X

The different addressing modes are explained in details in the sections below. Let us take an example to understand the various addressing modes. Suppose the two word instruction “LOAD AC”. The instruction is at

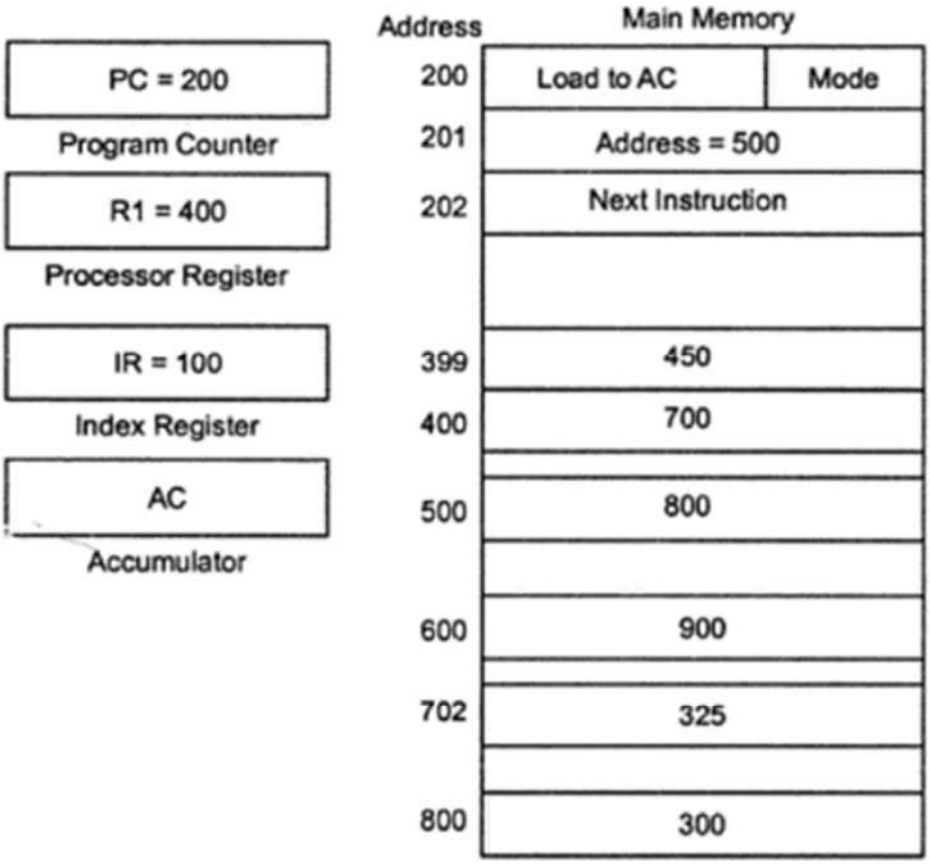


Figure 6.1. Example for Addressing Mode

address 200 and 201. The address field is 500. The first word of the instruction specifies the operation code and mode. The second word specifies the address part. The PC (Program Counter) which fetches the next instruction has the value 200. The content of the processor register R1 is suppose 400 and that of the Index register is 100. The Accumulator, AC receives the operand after the instruction is executed.

6.4 IMMEDIATE ADDRESSING

This is the simplest form of addressing. In the immediate mode, the operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than the address field. The operand can be a byte, word or double word of data. Immediate addressing is so-named because the value to be referenced immediately follows the operation code in the instruction. Thus, the data to be operated on is part of the instruction.

OPERAND = A

The instruction format for immediate addressing mode is shown in Figure 6.2 below:

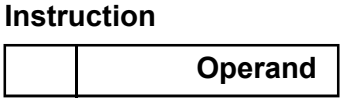


Figure 6.2: Instruction Format for Immediate Addressing Mode

In this mode, the source operand is a constant. The immediate data must be preceded by the "#" sign. It is very fast because the value to be loaded is included in the instruction. However, as the value is to be loaded is fixed at compile time, it is not very flexible.

In the figure 6.1, discussed in section 6.3, in the immediate mode of addressing the second word of the instruction is taken as the operand. i.e., the operand is 500.

This mode of addressing can be used to load information into any of the registers.

6.5 DIRECT ADDRESSING

When the operands are specified in the memory addressing mode, direct access to main memory data segment is required. However it results in slower processing of data. In direct addressing, the address field contains the effective address of the operand. The *Effective Address* is the location or the address where operand is present. It is defined to be the memory address obtained from the computation by the given addressing mode.

Direct addressing is so-named because the value to be referenced is obtained by specifying its memory address directly in the instruction.

EA = A

It requires only one memory reference and no special calculation. In the figure 6.1, discussed in section 6.3, the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.

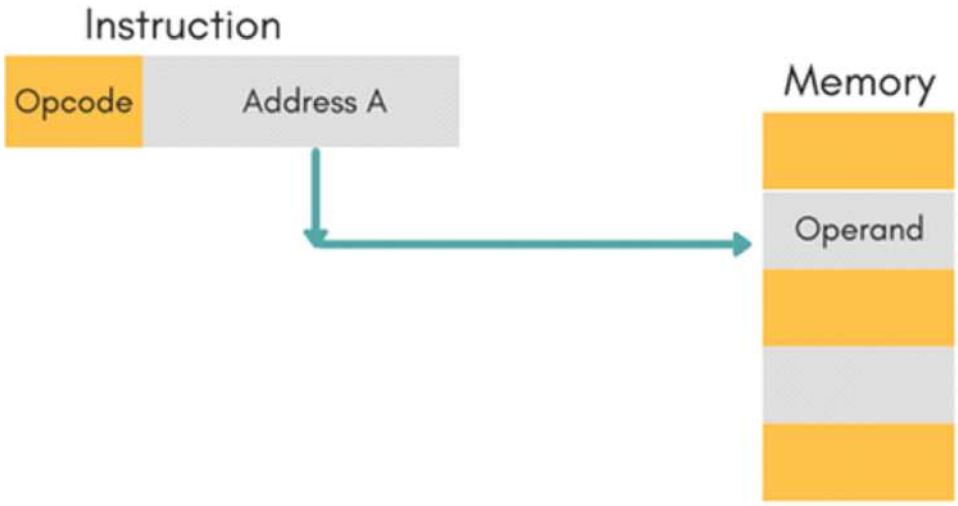


Figure 6.3: Direct Addressing

In direct memory addressing, one of the operands refers to a memory location and the other operand references a register.

Direct addressing is quite fast because although the value to be loaded is not included in the instruction, it is quickly accessible. It is also flexible than immediate addressing because the value to be loaded is given in the address.

CHECK YOUR PROGRESS

Q.1: Fill in the blanks:

i) The way in which the operand of an instruction is specified is referred to as .....

ii) In the ....., the operand is specified in the instruction itself.

iii) Immediate addressing is so-named because the value to be referenced immediately follows the ..... in the instruction.

iv) The addressing mode specifies a rule for interpreting the ..... field of an instruction before the ..... is actually executed.

v) The immediate data must be preceded by the ..... sign.

vi) The ..... is the location or the address where operand is present.

vii) ..... requires only one memory reference and no special calculation.

viii) In direct memory addressing, one of the operands refers to a ..... and the other operand references a .....

6.6 INDIRECT ADDRESSING

Indirect addressing is a powerful addressing mode that provides an exceptional level of flexibility. The limitation of direct addressing is that it provides only a limited address space as the length of the address field is usually less than the word length. To overcome this limitation, one solution can be to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is known as indirect addressing.

EA = (A)

The parentheses means "*Contents of*".

If the address bits of the instruction code are used as an address of the actual operand, it is termed as indirect addressing. Two memory references are needed to access data. First one is to access the address of the operand and the next one to access the data from that address.

The address field of instruction gives the address where the effective address is stored in memory. This slows down the execution, as this includes multiple memory lookups to find the operand.

In the figure 6.1 discussed in Section 6.3, the effective address is stored in memory at address 500. Therefore the effective address is 800 and the operand is 300.

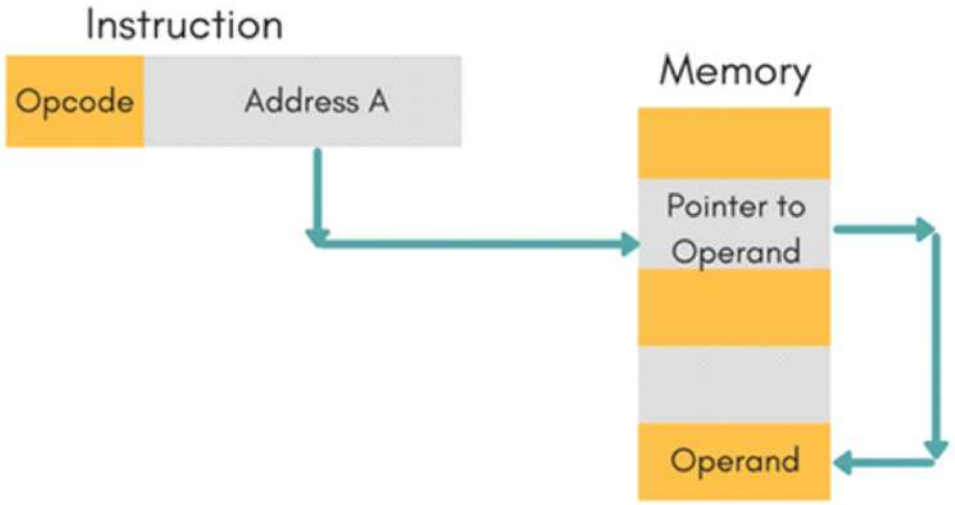


Figure 6.4: Indirect Addressing

The advantage of this addressing mode is that for a word length of N, an address space of 2^N is now available. However, if *k* is the length of the address field, the number of the effective addresses that may be referenced at any time is limited to 2^k. The bits in the address field specify a memory address that is to be used as a pointer to the operand.

The disadvantage is that instruction execution requires two memory references to fetch the operand. One is to get its address and the second is to get its value.

6.7 REGISTER

Register addressing and direct addressing are almost similar with the only difference that the address field refers to a register in Register addressing rather than a main memory address.

EA = R

In register addressing mode, instead of memory a register is used to specify the operand. In the register mode, in the figure 6.1, discussed in section 6.3, the operand is in R1and 400 is loaded into AC. There is no effective address in this case.

The advantages of register addressing are that only a small address field is needed in the instruction and no time-consuming memory reference is required. The disadvantage of register addressing is that the address space is very limited. The access time for the registers is much lesser as compared to the main memory access. As there are only a limited number of registers as that of the main memory locations, so the use of the registers should be efficient. If the operand in the register that has been brought from the main memory is used for multiple operations, then the register addressing is used efficiently. However, if the operand in the register is used for only once in the processor and then after that it is returned back to the main memory, then it is a wastage step.

6.8 REGISTER INDIRECT

Register indirect addressing is also called indexed addressing or base addressing. To load a value from memory into a register using register-indirect addressing, a second register, known as the base register is used. This base register holds the actual memory address that the program is interested in. In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.

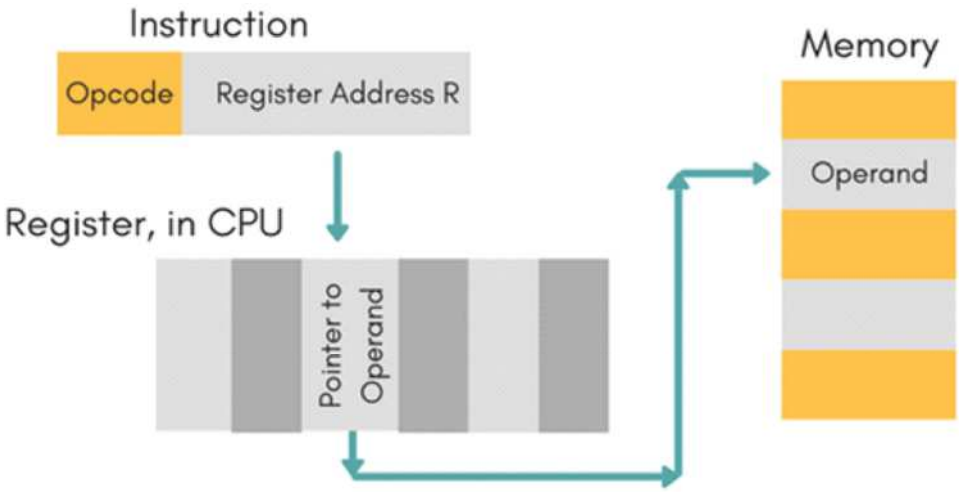


Figure 6.5: Register Addressing

CHECK YOUR PROGRESS

Q.2: Fill in the blanks:

i) The techniques of specifying the address of the data are known as .....

ii) An instruction specifies the operation in the ..... field on the data specified by the operands in the operand field.

iii) ..... is a powerful addressing mode that provides an exceptional level of flexibility.

iv) The address field of instruction, in indirect mode of addressing, gives the address where the ..... is stored in memory.

v) The bits in the address field specify a memory address that is to be used as a ..... to the operand.

vi) The address field in Register addressing refers to a ..... rather than a main memory address.

vii) The memory access time for the registers is much ..... as compared to the main memory addresses.

viii) Register indirect addressing is also called ..... addressing.

ix) The ..... holds the actual memory address that the program is interested in.

x) The use of the registers in addressing should be ..... as there are a limited number of registers.

6.10 LET US SUM UP

- An instruction is a group of bits that instruct the computer to perform a specific operation.
- An instruction specifies the operation in the operation code field on the data specified by the operands in the operand field.
- An addressing mode can specify a location in memory, or a register or a constant.
- The techniques of specifying the address of the data are known as addressing modes.
- There are a variety of addressing techniques that are been applied. The most common of them are immediate, direct, indirect, register and register indirect.
- In the immediate mode, the operand is specified in the instruction itself.
- Immediate addressing is so-named because the value to be referenced immediately follows the operation code in the instruction.
- In direct addressing, the address field contains the effective address of the operand.
- *Effective Address* is defined to be the memory address obtained from the computation by the given addressing mode.
- In direct memory addressing, one of the operands refers to a memory location and the other operand references a register.
- If the address bits of the instruction code are used as an actual operand, it is termed as indirect addressing.
- Register addressing and direct addressing are almost similar with the only difference that the address field refers to a register in Register addressing rather than a main memory address.

- As there are only a limited number of registers as that of the main memory locations, so the use of the registers should be efficient.
- Register indirect addressing is also called indexed addressing or base addressing.
- In register indirect addressing the base register holds the actual memory address that the program is interested in.

6.11 FURTHER READING

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI Learning Pvt. Ltd.
2) Gill, D. N. S. & Dixit, J. B. (2008). *Digital Design and Computer Organisation*. Firewall Media.
3) Null, L. & Lobur, J. (2014). *The Essentials of Computer Organization and Architecture*. Jones & Bartlett Publishers.
4) Stallings, William. (2004). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.

6.12 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) Addressing; ii) immediate mode; iii) operation code; iv) address, operand; v) #; vi) Effective Address; vii) Direct addressing; viii) memory location, register

Ans. to Q. No. 2: i) addressing modes; ii) operation code; iii) Indirect addressing; iv) effective address; v) pointer; vi) register; vii) lesser; viii) indexed; ix) base register; x) efficient

6.13 MODEL QUESTIONS

Q.1: What is the purpose of using address mode techniques by a computer? Summarize the various addressing modes.

Q.2: What are the differences between direct and indirect addressing modes?

*** ***** ***

UNIT 7: INPUT - OUTPUT ORGANIZATION

UNIT STRUCTURE

7.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe the different input-output techniques
- describe interrupt driven and programmed I/O techniques
- define direct memory access (DMA)
- describe daisy chaining technique

7.2 INTRODUCTION

In the previous units, we have discussed the different types of sequential and combinational circuits and data representation in binary number system, decimal number system, BCD, ASCII systems etc. We have also learnt about the different instruction formats and types in the previous unit along with different types of addressing modes and assembly language notation. In this unit, we will explore the different concepts related to input output organization. Different I/O techniques like programmed I/O, interrupt driven I/O and DMA are covered in detail in this unit. Priority and

daisy chaining technique are also covered in this unit. In the next unit, we discuss about cache and virtual memory.

7.3 INPUT OUTPUT ORGANIZATION

The transfer of information between the main memory and the outside world is through the input-output devices. An I/O system is composed of I/O devices (peripherals), I/O control units, and software to carry out the I/O transaction(s) through a sequence of I/O operations. An I/O transaction example is reading a block of data from disk to memory. It is simply a sequence of I/O operations (instructions) to transfer data between the peripheral devices and main memory, and to enable the central processing unit (CPU) to control the peripheral devices connected to it. Thus, I/O operations are of two classes: control operations and data transfer operations. There are different techniques for communication between the memory and the I/O devices which will be discussed in the following sections.

7.4 DIFFERENT I/O TECHNIQUES

The different I/O techniques are categorized into three types based on how information is transferred between the main memory and the input/output devices, that is, whether CPU interaction is used continuously or the CPU is interrupted by the device only at the time of transfer. They are:

1. Programmed Input/Output
2. Interrupt driven Input/Output
3. Direct memory Access

The **I/O module,** also called an I/O interface works as the mediator between the I/O devices and the CPU. The information from the I/O devices to the CPU and vice-versa is conveyed by the I/O module. An I/O module is connected to the system bus at one end and is connected to the number of I/O devices on the other end.

The I/O module is used for many reasons:

a. Due to the speed mismatch between the CPU and the peripherals, it is improper to use a slow peripheral on a high speed system bus.

b. Many peripherals might be required to be connected to the same system bus but it may be difficult to do so.
c. Data format of the peripherals might be different from that used by the CPU.

The I/O module functions are as given below:

i. Provide control and timing signals
ii. Communicate with CPU
iii. Communicate with I/O devices.

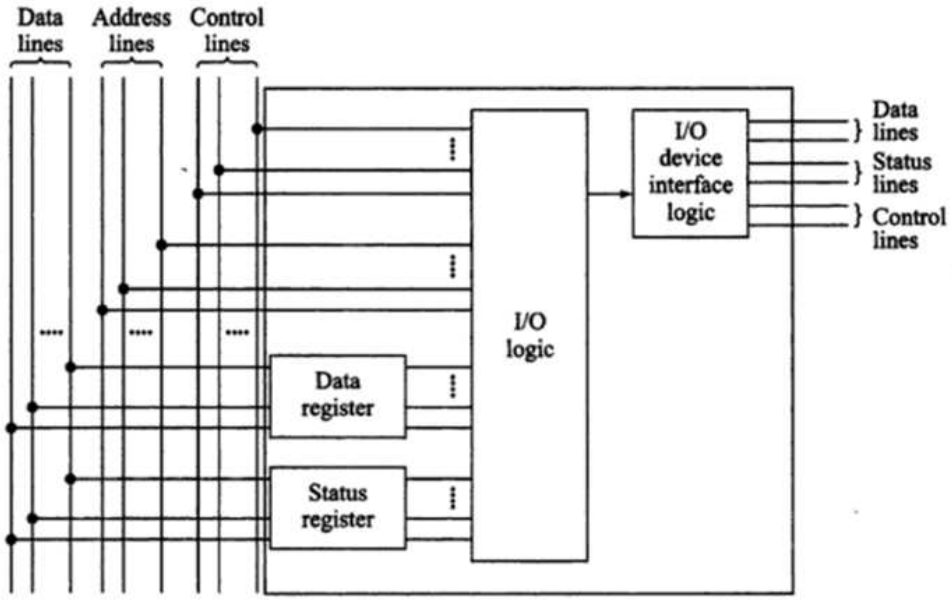


Figure 7.1: General structure of the I/O module

7.4.1 Programmed I/O

The programmed I/O is the simplest type of I/O technique for the exchange of data or any type of communication between the processor and the external devices. With programmed I/O, data are exchanged between the processor and the I/O module i.e. data may be transferred from I/O device to the CPU as input or from CPU to the I/O device as output.

At the time of executing an I/O instruction the CPU issues a command, then waits for I/O operations to be complete. As the CPU is faster than the I/O module, the CPU has to wait a long time for the concerned I/O module to be ready for either reception or transmission of data. The

CPU, while waiting, must repeatedly check the status of the different I/O modules one after another serially and this process is known as **polling**. As a result, the level of the performance of the entire system is severely degraded.

Programmed I/O basically works in these ways:

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later

Programmed I/O Mode Input Data Transfer:

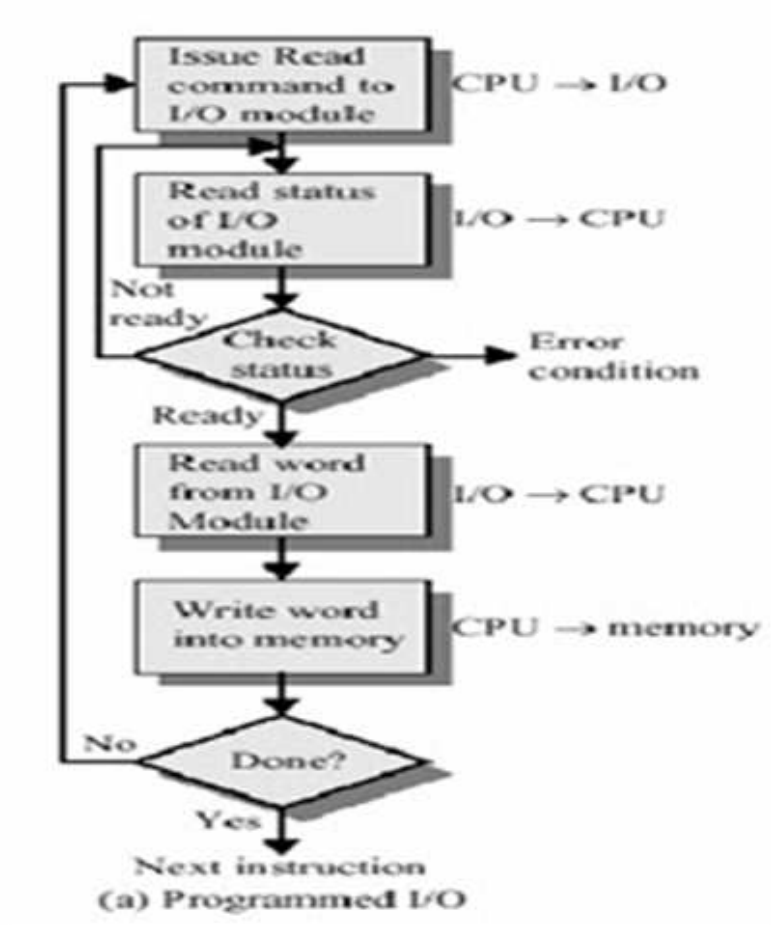


Figure 7.2: Prgorammed I/o mode Input Data Transper

Advantages & Disadvantages of Programmed I/O

Advantages	- simple to implement, suitable for very small amount of data transfer
	- very little hardware support
Disadvantages	- busy waiting
	- ties up CPU for long period with no useful work

7.4.2 Interrupt-Driven I/O

Interrupt driven I/O is another technique which is dealing with I/O operations. It is a way of controlling input/output activity in which the peripheral that needs to send or receive data sends a signal.

The problem with Programmed I/O is that the processor has to wait for a long time for the concerned I/O module to be ready for either reception or transmission of data. The processor must continuously check the status of the I/O module and as such the performance of the system is affected.

An alternative to this is for the CPU to issue a command to the I/O module and to proceed to do some other useful task until the device requesting the service sends an interrupt to the CPU when it is ready to exchange data. The processor then executes the data transfer and on completion resumes its former processing.

The interrupt technique requires more complex hardware and software, but makes far more efficient use of the computer's time and capacities.

The basic operations of Interrupt are as follows:

1. CPU issues read command.
2. I/O module gets data from peripheral while the CPU does other work.
3. I/O module interrupts CPU.
4. CPU requests data.
5. I/O module transfers data.

Advantages & Disadvantages of Interrupt Drive I/O

Advantages	-	fast
	-	efficient, no wastage of CPU time
Disadvantages	-	can be tricky to write if using a low level language
	-	can be tough to get various pieces to work well Together
	-	usually done by the hardware manufacturer / OS maker, e.g. Microsoft

In interrupt driven I/O implementation, there are two design issues. First, there may be multiple I/O modules and the issue is how the processor will determine which device issued the interrupt. Secondly, if multiple interrupts have occurred how the processor can decide which interrupt to process? To solve this, there are four categories of techniques that can be used.

- **Multiple Interrupt Lines**

The straightforward approach is to provide multiple interrupt lines between the processor and the I/O modules. This allows multiple modules to be handled at the same time. However, it is not practical to assign many bus lines and processor pins to interrupt lines. One of the reasons is that there might be more than one I/O module attached to a single line. This defeats the purpose of this technique.

- **Software Poll**

Whenever an interrupt is detected by the processor, it branches to an interrupt service routine which will poll each and every I/O module to determine the exact interrupting module. The processor raises a poll which could be in the form of a command line. The address of the respective I/O module which is interacted by the poll will be then placed on the address line. The module will respond positively if it is responsible for setting the interrupt. Alternatively, every I/O module has an addressable status register which can be read by the processor to determine the interrupting module. The drawback of this technique is that it is time consuming, as the processor

has to poll all the devices connected to it one after another or in some specified order.

- **Daisy Chain (Hardware Poll)**

Daisy chain is a hardware poll. Whenever there is an interrupt, the processor sends out an interrupt acknowledge which will propagate throughout the series of I/O modules. This process will continue until it reaches a requesting module which will then respond by placing a word on the data lines. The processor subsequently directs the module to its specific device-service routine. This method is also known as vectored interrupt.

- **Bus Arbitration**

This method involves the I/O module gaining control over the bus before requesting for the interrupt. It is limited to only one module at a time. The processor sends an acknowledge signal whenever it detects an interrupt. The requesting module then places its data on the data lines.

CHECK YOUR PROGRESS

Fill in the blanks:

Q1. The _____ works as the mediator between the I/O devices and the CPU.

Q2. It is improper to use a slow peripheral on a _____ system bus due to the speed mismatch between the CPU and the peripherals.

Q3. In programmed I/O, the CPU issues a _____ and then waits for I/O operations to be complete.

Q4. The processor must continuously check the status of the I/O module in _____.

Q5. The interrupt I/O technique requires more ____ hardware and software.

Q6. _____ allows multiple modules to be handled at the same time.

Q7. Every I/O module has an addressable _____ which can be read by the processor to determine the interrupting module.

Q8. Daisy chain is a _____ poll.

Q9. _____ is limited to only one module at a time.

Q10. The CPU, in programmed I/O, must repeatedly check the status of the I/O module, and this process is known as _____.

7.4.3 Direct Memory Access (DMA)

We have till now discussed two different methods of data transfer which requires active intervention of the CPU to transfer data between the memory and the I/O module. However, both these two approaches suffer from two drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can serve a device.
- The CPU is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

Thus, to transfer large blocks of data at a high speed, a special control unit may be provided to allow the transfer to be direct between the main memory and the I/O device. This technique, where the data is directly transferred from the I/O device to the memory or vice versa without continuous involvement of the processor, is called Direct Memory Access or DMA. The CPU has just to initiate the transfer by initializing some parameter in DMA transfers. It is a sophisticated I/O technique in which a ***DMA controller*** replaces the CPU and takes care of both the I/O and the memory. By using DMA, the data transfer rates became faster. The parameters that are to be initialized by the CPU in DMA transfer are:

- Starting address of data from or to which the data are to be transferred;
- number of memory words (count) to be transferred whether it is a read or write operation.

The figure 5.3 below shows how the transfer takes place using a DMA Controller.

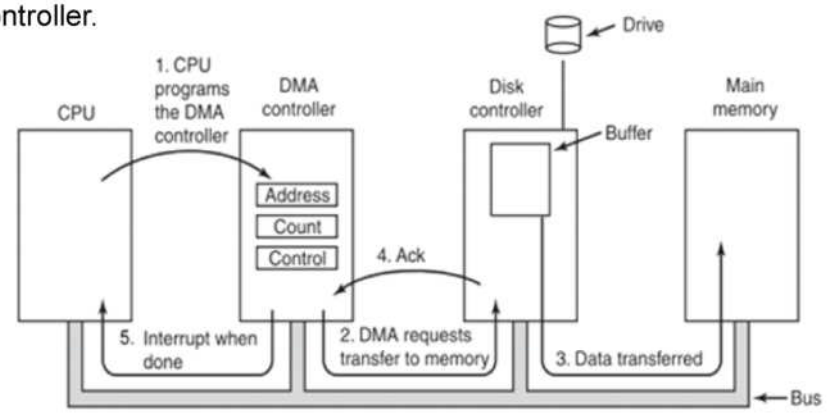


Figure 5.3: Operation of a DMA transfer

Step 1: The CPU first programs the DMA controller by setting its registers so that the DMA controller knows what to transfer. It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum.

Step 2: The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller. This read request looks like any other read request, and the disk controller does not know whether it came from the CPU or from a DMA controller.

Step 3: The memory address where the data is to be written is on the bus' address lines; so, when the disk controller fetches the next word, it knows where to write the word.

Step 4: When the write is complete, the disk controller sends an acknowledgement signal to the DMA controller. The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0.

On completion of the transfer, the DMA controller interrupts the CPU to let it know that the transfer is complete. System performance improves by separate processing of the transfers to and from the peripherals. For example, between camera memory and USB port.

7.5 PRIORITY AND DAISY CHAINING TECHNIQUE

We have already come to know that at **Interrupt driven I/O,** priority of the Interrupt signal will play a vital role. Now, the question is: what is priority of signal? Priority means the weightage or importance. If the priority of the interrupt signal is more than the current execution, then the processor is to stop the current execution and to show response to the recently coming interrupt request. The processor will continue its current execution if the priority of the current execution is more than that of interrupt signal. For a particular processor, different interrupt signals have different types of priority. For example, Intel 8085 processor has five number of interrupt signals, namely, INTR, RST5.5, RST6.5, RST7.5 AND TRAP. There is an interrupt acknowledgement whose name is INTA. Among these five interrupt signals, INTR has the lowest priority and trap has the highest priority.

Say, the processor is receiving INTR and TRAP interrupt request at a time from device 1 and from device 2 respectively as shown below:

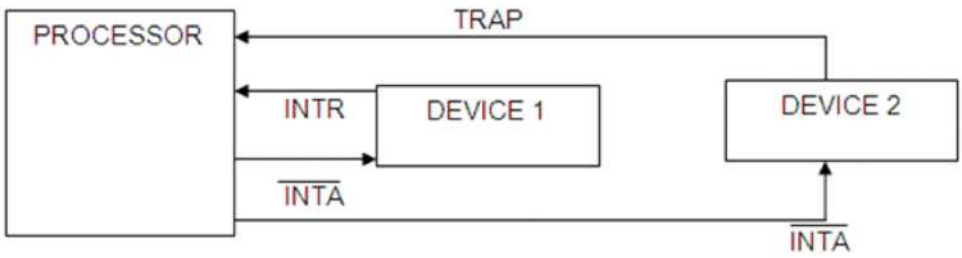


Figure 7.4: Schematic Representation of Interrupt signal INTR and TRAP

Then the processor will compare the priorities of INTR and TRAP with the priority of the current execution. If both INTR and TRAP have more priority than the current execution, then it will again check the priority between the two interrupt requests. And the processor will give response to the device 2 as TRAP has more priority than INTR sent by device 2.

But what will happen if both the devices send the interrupt requests having the same priority, say INTR! Then the concept of Daisy Chaining will come. In Daisy Chaining technique, arrangement will be as shown below:

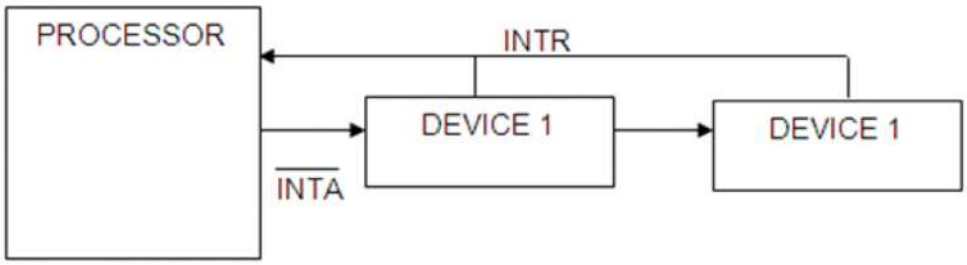


Figure 7.5: Schematic Representation of Interrupt signal INTR only

In this type of situation, interrupt acknowledgement will receive device 1 at first. That means, in daisy chaining technique the device which is electrically closest to the processor will acquire the highest priority. If device 1 has pending request then it will not pass the INTA signal to the device 2 until the completion of the pending task. If device 1 has no pending request then it will pass the INTA to the device 2.

CHECK YOUR PROGRESS

Fill up the blanks:

Q11. To transfer large blocks of data at a high speed, the technique used is called _____.

Q12. The _____ initiates the transfer by issuing a read request over the bus to the disk controller.

Q13. When the write is complete, the _____ sends an acknowledgement signal to the DMA controller.

Q14. If the priority of the interrupt signal is more than the current execution, what the processor will do?

Q15. Write the name of Interrupt signals of a typical processor, say Intel 8085. Arrange them in ascending order according to their priority.

7.6 LET US SUM UP

- The different I/O techniques are categorized into three types -
  - Programmed Input/Output
  - Interrupt driven Input/Output
  - Direct memory Access.
- The I/O module works as the mediator between the I/O devices and the CPU.
- The CPU issues a command and then waits for I/O operations to be complete.
- The CPU, while waiting, must repeatedly check the status of the different I/O modules in some order and this process is known as Polling.
- Interrupt driven I/O is another technique which deals with I/O operations.

- The problem with Programmed I/O is that the processor has to wait for a long time for the concerned I/O module to be ready for either reception or transmission of data.
- The CPU has to issue a command to the I/O module and to proceed to do some other useful task until the device requesting the service sends an interrupt to the CPU when it is ready to exchange data
- To transfer large blocks of data at a high speed, a special control unit may be provided to allow the transfer to be direct between the main memory and the I/O device.
- During priority technique, the processor will give response to the device which sends interrupt signal having more priority.
- In daisy chaining technique the device which is electrically closest to the processor will acquire the highest priority.

7.7 FURTHER READINGS

1) Stallings, W. (2000). Computer organization and architecture: designing for performance. Pearson Education India.
2) Chaudhuri, P. P. (2008). Computer organization and design. PHI Learning Pvt. Ltd..
3) Godse, A. P., & Godse, D. A. (2008). Computer Organization and Architecture. Technical Publications.

7.8 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: I/O module
Ans to Q No 2: high speed
Ans to Q No 3: command
Ans to Q No 4: programmed I/O
Ans to Q No 5: Complex
Ans to Q No 6: Multiple Interrupt Lines
Ans to Q No 7: status register
Ans to Q No 8: Hardware
Ans to Q No 9: Bus Arbitration

Ans to Q No 10: Polling

Ans to Q No 11: Direct Memory Access

Ans to Q No 12: DMA controller

Ans to Q No 13: disk controller

Ans to Q No 14: Processor is to stop the current execution, and to show response to the recent coming interrupt request.

Ans to Q No 15: INTR, RST5.5, RST6.5, RST7.5 AND TRAP. There is an interrupt acknowledgement which name is INTA. The ascending order according to their priority is as follows:

INTR< RST5.5< RST6.5< RST7.5< TRAP

7.9 MODEL QUESTIONS

Q1. Explain the different I/O techniques used in the computer.

Q2. What is Programmed I/O? What are its advantages and disadvantages?

Q3. Explain the differences between the Programmed I/O and Interrupt driven I/O.

Q4. What are the two design issues in interrupt driven I/O implementation? How can these be solved?

Q5. Explain the direct memory access.

Q6. What is the main disadvantage of polling technique? Explain the functional characteristics of polling technique.

Q7. What is Interrupt signal? Write about the concept.

Q8. Write about the functional characteristics of Interrupt driven I/O.

Q9. Differentiate between priority and daisy chaining technique. Explain in detail.

****