



MSCPH522

M. Sc. III SEMESTER

**MEMORY DEVICE AND
MICROPROCESSOR**



**DEPARTMENT OF PHYSICS
SCHOOL OF SCIENCES
UTTARAKHAND OPEN UNIVERSITY**

Board of Studies

Prof. P. D. Pant

Director School of Sciences
Uttarakhand Open University, Haldwani

Prof. P. S. Bisht,

SSJ Campus, Kumaun University, Almora.

Dr. Kamal Devlal

Department of Physics
School of Sciences, Uttarakhand Open University

Prof. S.R. Jha,

School of Sciences, I.G.N.O.U.,
Maidan Garhi, New Delhi

Prof. R. C. Shrivastva,

Professor and Head, Department of
Physics, CBSH, G.B.P.U.A.&T.
Pantnagar, India

Pogramme Coordinator : Dr. Kamal Devlal

Department of Physics (School of Sciences)

Dr. Kamal Devlal, Assistant Professor & Pogramme coordinator

Dr. Vishal Sharma, Assistant Professor

Dr. Gauri Negi, Assistant Professor

Dr. Meenakshi Rana, Assistant Professor (AC)

Dr. Rajesh Mathpal, Assistant Professor(AC)

Unit writing and Editing

Editing**Dr. Rajesh Mathpal**

Department of Physics
School of Sciences, Uttarakhand Open
University

Writing**1. Dr. Rajesh Mathpal**

Department of Physics
School of Sciences, Uttarakhand Open University

2. Dr. Abhishek Tomar

Department of Electronics and Communication
G B Pant University of Ag. And Tech. Pantnagar
US Nagar, Uttarakhand

3. Dr. Hrishitosh Bisht

Department of Electronics and Communication
Engineering, College of Technology
G B Pant University of Ag. And Tech. Pantnagar
US Nagar Uttarakhand

Course Title and Code : Memory Device and Microprocessor (MSCPH522)

ISBN :

Copyright : Uttarakhand Open University

Edition : 2022

Published By : Uttarakhand Open University, Haldwani, Nainital- 263139

Printed By :

Memory Device and Microprocessor



**DEPARTMENT OF PHYSICS
SCHOOL OF SCIENCES
UTTARAKHAND OPEN UNIVERSITY**

Phone No. 05946-261122, 261123

Toll free No. 18001804025

Fax No. 05946-264232, E. mail info@uou.ac.in

<http://uou.ac.in>

Contents

Course 13: Digital Electronics and Communication System

Course code: MSCPH522

Credit: 3

Unit number	Block and Unit title	Page Number
BLOCK – I Logic families and memories		
1	Logic families	5-35
2	Memory organization and expansion	36-71
BLOCK – II Microprocessor hardware and Interface		
3	Microprocessor architecture and Microcomputer system	72-96
4	8085 Microprocessor and memory interfaces	97-126
5	Interfacing I/O device	127-151
BLOCK – III 8085 Microprocessor programming		
6	8085 Microprocessors Programming	152-172
7	Assembly language Programming	173-188
8	Counters time delay, Stack and code conversion	189-211
9	Advanced microprocessor	212-245

UNIT- 1**LOGIC FAMILIES**

STRUCTURE

- 1.1.** Introduction to Logic Families
- 1.2. Objectives
- 1.3. Saturated and unsaturated logic circuits
- 1.4. Performance Characteristics
- 1.5. Resistance- Transistor Logic circuits
- 1.6. Transistor- Transistor Logic circuits
- 1.7. Integrated Injection Logic
- 1.8. Emitter Coupled Logic
- 1.9. MOSFET, CMOS, and Tri-state Logic
- 1.10. Logic families and their performance characteristics

Summary

Glossary

References

Questions

Answers

1.1 INTRODUCTION TO LOGIC FAMILIES

The semiconductor devices are of two types: bipolar and unipolar. Based on these two, digital integrated circuits are available using these bipolar and unipolar technologies in the fabrication process. A group of ICs with the same logic levels and supply voltages for performing various logic functions have been fabricated using a specific circuit configuration known as *logic family*. Logic circuits can be interconnected without additional circuitry. The output of one logic circuit can be used as input to another logic circuit. This presents the electronic circuits in each IC digital logic family and helps to analyze its electrical operation. The digital logic families are as follows:

- RTL Resistor-transistor logic
- DTL Diode-transistor logic
- TTL Transistor-transistor logic
- ECL Emitter-coupled logic
- MOS Metal-oxide semiconductor
- CMOS Complementary metal-oxide semiconductor

The RTL and DTL logic families have historical significance and are rarely used nowadays for designing digital circuits. RTL was one among the logic family used widely. RTL is admitted as a part of your syllabus because it forms a good starting point to explain the basic operations of digital gates. TTL has now replaced DTL, as the operation of TTL is easy to understand. The TTL logic family is derived from the DTL logic family. TTL, ECL, and CMOS have a large number of small-scale integrated (SSI) circuits, as well as medium-scale integrated (MSI), large-scale integrated (LSI), and very large-scale integrated (VLSI) components. MOS is extensively used for LSI and VLSI components.

The primary circuit used in each IC digital logic family is either a NAND gate or a NOR gate. These are also called universal gates because they can be used to build any other complicated digital circuits. NAND and NOR gates are usually defined by the Boolean functions that they implement in terms of binary variables. When analyzing them as electronic circuits, it is necessary to investigate their input-output relationships in terms of two voltage levels: a high level designated by H and a low level designated by L. Table 1.1 and Table 1.2 represent the truth tables of a positive logic NAND and NOR gates, respectively.

Inputs		Output
X	Y	Z
L	L	H
L	H	H
H	L	H
H	H	L

Table 1.1: Positive logic NAND gate

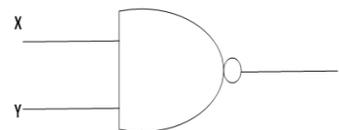


Fig.1.1: NAND gate

Inputs		Output
X	Y	Z
L	L	H
L	H	L
H	L	L
H	H	L

Table1.2: Positive logic NOR gate

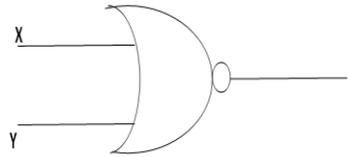


Fig.1.2: NOR gate

Figures 1.1 and 1.2 are the symbols of NAND and NOR gates. In a positive logic system, we assign a binary 1 to H and 0 to L, whereas in a negative logic system, we assign a binary 1 to L and 0 to H.

The logic family can be categorized into two heads: The bipolar logic family and the unipolar logic family. A bipolar junction transistor (BJT) can be of *nnp* or *pnp*, i.e., a semiconductor of one type (either n or p) is sandwiched between semiconductors of the other type (p or n respectively), thus resulting in two *pn* junctions. In BJT, the total current flowing is due to both the charge carriers, i.e., electrons and holes. Thus, we say that the BJT is a bipolar device. In contrast, a Field-effect transistor (FET) is a unipolar device. In a unipolar device, the resulting current is due to only the flow of one type of charge carrier, which can be either electron (n-channel) or holes (p-channel). RTL, DTL, TTL, and ECL are based on bipolar transistors, whereas MOS and CMOS are unipolar devices. CMOS devices are constructed using metal-oxide-semiconductor field-effect transistors (MOSFET or MOS).

In this chapter, you will first study the difference between saturated and unsaturated logic families and the various performance characteristics to compare these logic families. Then we will learn about the bipolar and unipolar logic families and analyze how the basic gates can be constructed using them. We will also study the performance characteristics of these logic families.

1.2. OBJECTIVES

After studying this unit, you should be able to-

- Learn about saturated and unsaturated logic circuits
- Familiarize with unipolar and bipolar devices of logic families
- Understand the various performance characteristic parameters
- Understand the working of different logic families such as RTL, TTL, I²L, ECL, MOSFETs, and CMOS
- Compare all logic families with their performance characteristics
- Solve problems based on the above logic families.

1.3. SATURATED AND UNSATURATED LOGIC FAMILIES

We can classify *bipolar logic families* into two categories:

- I. Saturated logic family, and
- II. Unsaturated logic families.

Those logic circuits in which transistors are driven into saturation are called saturation logic. In saturated logic, the transistor switches between the off and saturation regions. Those circuits that avoid saturation of their transistors are designated as non-saturated logic; in non-saturated logic, the transistor is switched between the off and active regions. Saturated logic families are slower because of the additional delay in bringing the transistor out of saturation. The switching device is always an npn transistor, with pnp transistors used as loads or current sources. The saturated logic families are RTL, DCTL, DTL, HTL, TTL, I²L. The non-saturated logic families are ECL and Schottky TTL.

UNIPOLAR LOGIC FAMILIES:

Unipolar devices include metal-oxide-semiconductor (MOS) devices, and the MOS logic function is being implemented in MOSFETs only. MOSFET logic family can be categorized as the following:

- I. PMOS
- II. NMOS
- III. CMOS

It is to be noted that in PMOS, only p-channel MOSFETs can be used, whereas, in NMOS, only n-channel MOSFETs can be used. In CMOS, both p- and n-channel MOSFETs can be used.

Example 1: Which type of unipolar logic family exhibits its usability for the applications requiring low power consumption?

- a) PMOS
- b) NMOS
- c) CMOS
- d) All of the above

Answer: CMOS

Example 2: Which among the bipolar logic families is adopted explicitly for high-speed application?

- a) DTL
- b) TTL
- c) ECL
- d) I²L

Answer: ECL

1.4. PERFORMANCE CHARACTERISTICS

The wide use of IC in the digital world and the development of various IC fabrication technologies have made it important to know the characteristics of IC logic families. Classification of digital ICs is based on the number of components fabricated on the chip.

IC Classification	Basic gates	Number of Components
Small scale integration (SSI)	Less than 12	Up to 99
Medium scale integration (MSI)	12-99	100-999
Large-scale integration (LSI)	100-999	1,000-9,999
Very-large-scale integration (VLSI)	Above 1,000	Above 10,000

Table 1.3: Classification of Digital ICs

The performance characteristics of IC digital logic families are generally compared by analyzing the basic gate circuit in each family. The various characteristics of digital ICs used to compare their performances are:

- 1) Speed of operation,
- 2) Power dissipation,
- 3) Figure of merit,
- 4) Fanout,
- 5) Current and Voltage parameters,
- 6) Noise immunity,
- 7) Operating temperature range,
- 8) Power supply requirements, and
- 9) Flexibilities available.

First, you will understand the basic definitions of the above terms and their properties. This will help you compare different performance characteristics of the IC logic families, which will, in turn, be beneficial in selecting a logic family for a particular application.

1) Speed of Operation

Speed of operation is also referred to as propagation delay time. The propagation delay is defined as the signal propagating from input to output with an average transition delay time when the signal alters its binary value. A definite amount of time is needed to propagate the signal from input to output. This definite amount of time is said to be a propagation delay of the gate. It is specified in nanoseconds (ns), ($1 \text{ ns} = 10^{-9}$ of a second). The signal traveling from the input to the output passes through a series of gates that are connected

together. The total delay of the circuit can be calculated by adding all the propagation delays through gates. When each gate has a short propagation delay, the speed of the operation becomes important. Therefore, there should be a certain minimum number of gates between the input and output terminals of the circuit. When the output signal goes from high level to low level, then the propagation delay time is represented as t_{PHL} . Similarly, whenever the output alters its state again from the low to the high level, the transition delay is represented as t_{PLH} . The average propagation delay time can therefore be calculated by taking the average of the two delays.

The average of both the delays, i.e., t_{PHL} and t_{PLH} is called average propagation delay time.

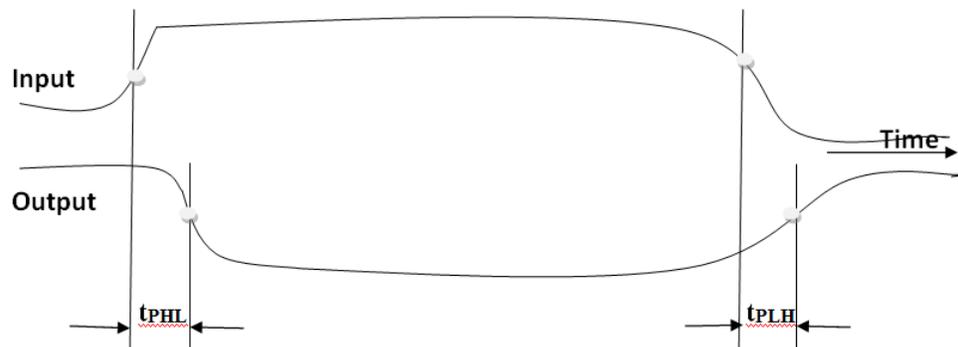


Fig.1.3: Measurement of propagation delay

For example, let the gate delays be $t_{PHL} = 9$ ns and $t_{PLH} = 9$ ns with a load resistance of 200Ω and a load capacitance $= 30$ pF. Then the average propagation delay of the gate will be equal to $(9+9)/2 = 9$ ns.

Sometimes it is necessary to know the maximum delay time of a gate instead of the average value under certain conditions. Also, it is necessary to consider the maximum delay to ensure proper operation if the speed of operation is critical.

2) Power Dissipation

A certain amount of power is required to operate any electronic circuit. The power dissipation can be specified in milliwatts (mW) and tells the power requirement by the gate. Power dissipation represents the power delivered to the gate from the power supply and does not include the power supplied by any other gate. Taking an IC consisting of four gates will require power from its supply which is four times the power dissipated at each gate. The calculation for the power dissipation can be done as follows:

The amount of power dissipated at the gate can be calculated from the supply voltage V_{CC} and the current I_{CC} drawn by the circuit. The power dissipation is the product of the supply voltage, i.e., V_{CC} , and the current drawn by the circuit, i.e., I_{CC} . So, the average current is:

$$I_{cc(avg)} = \frac{I_{CCH} + I_{CCL}}{2}$$

Average power dissipation, therefore, can be calculated using the formula given below:

$$P_{D(avg)} = I_{cc(avg)} * V_{CC}$$

3) Figure of Merit (FoM)

It is defined as the product of speed and power. It is specified in pico Joules (ns×mW = pJ).

$$\text{Figure of Merit} = \text{Propagation delay time (ns)} \times \text{power (mW)}$$

The low value of FoM is desirable. If high speed is demanded, i.e., low propagation delay, then there is an increase in power dissipation.

4) Fanout

Fan-out, defined by a number, is the maximum number of inputs that can be connected to a gate's output without degrading the normal operation of the gate. We define a standard load as the total amount of current required by the input of another gate in the same logic family. Many times, the term fanout is used rather than loading. The term loading refers to the fact that the output of a gate can supply a certain amount of limited current. Above this current value, the gate cannot operate properly and is said to be *overloaded*. We specify loading rules for a digital circuit family. Using these loading rules, we can calculate the maximum loading amount for each circuit's output in the logic family.

To calculate the Fanout, we use the net current available in the output of the gate and the amount of current needed in each gate input. As shown in Figures 1.4 (a) and (b), several gates are connected to the output of one gate. The output of the gate in Figure 1.4 (a) represents a high voltage level and behaves like the current source I_{OH} to all the other gate inputs connected to it. Each gate input requires a current I_{IH} for proper operation. In the same way, the output of the gate in Figure 1.4 (b) represents a low voltage level. It behaves like a current sink I_{OL} for all the gate inputs connected to it. Each gate input supplies a current I_{IL} . The ratio I_{OH}/I_{IH} or I_{OL}/I_{IL} is used to calculate the Fanout of a gate. We take the smaller value of the ratio.

Example 3: The standard gate has the following values for the currents:

$$I_{OH} = 400 \mu\text{A}$$

$$I_{IH} = 40 \mu\text{A}$$

$$I_{OL} = 16 \text{ mA}$$

$$I_{IL} = 1.6 \text{ mA}$$

The two ratios give the same number in this case:

$$\frac{400 \mu\text{A}}{40 \mu\text{A}} = \frac{16 \text{ mA}}{1.6 \text{ mA}} = 10$$

Therefore, the Fanout of the standard gate given in the example is 10. It implies that the output of this standard gate cannot be connected to the inputs of more than 10 other gates in the same logic family. If we connect more than ten gates in this case, then this standard gate may not be able to drive or sink the amount of current needed from the connected inputs.

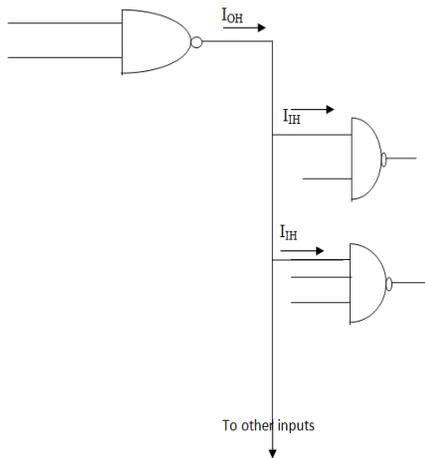


Fig1.4 (a): High-level output

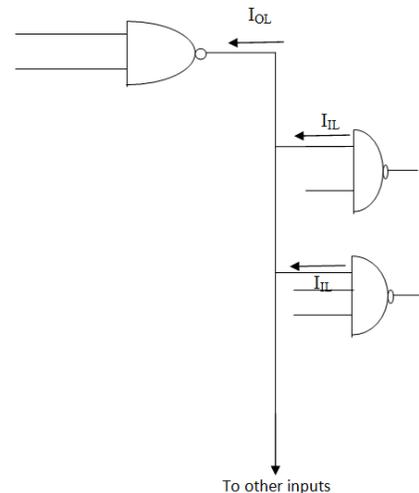


Fig1.4 (b): Low-level output

Example 4: Which of the following has the maximum Fanout?

- a) CMOS, b) PMOS, c) ECL, and d) I^2L

Answer: CMOS

Example 5: If a logic circuit has a fan out of 4, then the circuit

- a) Has 4 inputs
 b) Has 4 outputs
 c) Can drive a maximum of 4 inputs
 d) Gives output 4 times the input

Answer: Gives output 4 times the input.

Example 6: Calculate the Fanout of a TTL circuit with the following specifications:

$I_{OL}(\text{max})=32\text{mA}$, $I_{IL}(\text{max})=1.6\text{mA}$, $I_{OH}(\text{max})=400\mu\text{A}$, $I_{IH}(\text{max})=10\mu\text{A}$

Answer: Fan-out (high)= $400/10=40$

Fan-out (low)= $32/1.6=20$

As mentioned in theory, a smaller fanout value is desirable. So **Fanout for this case is 20.**

Self Assessment Question (SAQ) 1: For an open-collector, the specifications are:

$$\begin{aligned} V_{OH} &= 2.4\text{V} \\ V_{IH} &= 0.4\text{V} \\ I_{OH} &= 250\mu\text{A} \\ I_{IL} &= 16\text{mA} \\ I_{IH} &= 40\mu\text{A} \\ I_{IL} &= -1.6\text{mA} \end{aligned}$$

Calculate R_c for open collector gate. Assume $V_{CC} = 5\text{V}$, fan-out=8.

5) Current and Voltage Parameters

The following current and voltages are specified, which are very useful in the design of digital circuits.

The high-level input voltage, V_{IH} : This is the minimum input voltage recognized by the gate as logic 1.

The low-level input voltage, V_{IL} : This is the maximum input voltage recognized by the gate as logic 0.

The high-level output voltage, V_{OH} : This is the minimum output voltage recognized by the gate as logic 1.

The low-level output voltage, V_{OL} : This is the maximum output voltage recognized by the gate as logic 0.

High-level input current, I_{IH} : This is the minimum current that must be supplied by a driving source corresponding to 1-level voltage.

Low-level input current, I_{IL} : This is the minimum current that must be supplied by a driving source corresponding to 0-level voltage.

High-level output current, I_{OH} : This is the maximum current that the gate can sink in 1 level.

Low-level output current, I_{OL} : This is the maximum current that the gate can sink in 0 level.

High-level supply current, $I_{CC}(1)$: This is the supply current when the output of the logic gate is at logic 1.

Low-level supply current, $I_{CC}(0)$: This is the supply current when the output of the logic gate is at logic 0.

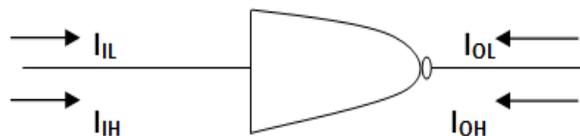


Fig.1.5: A Gate with current directions marked

6) Noise Margin

Various electrical signals from industrial and other similar sources can give unwanted voltage to the connecting wires between logic circuits. These undesirable unwanted signal obtained is termed *noise*. Here, two types of noise are considered, DC noise and AC noise. DC noise is generally because of the drift in the voltage levels of a signal. In contrast, AC noise is a random pulse that some other switching signals may create. So, noise is termed as an unwanted signal which is superimposed upon the normal operating signal.

Noise Margin is defined as the maximum noise voltage added to an input signal of a digital circuit that does not cause an undesirable change in the circuit output. It is desired in most applications to operate the circuits reliably in a noisy environment. Therefore, the Noise margin (NM) represents the maximum noise signal tolerated by the gate and is usually expressed in volts.

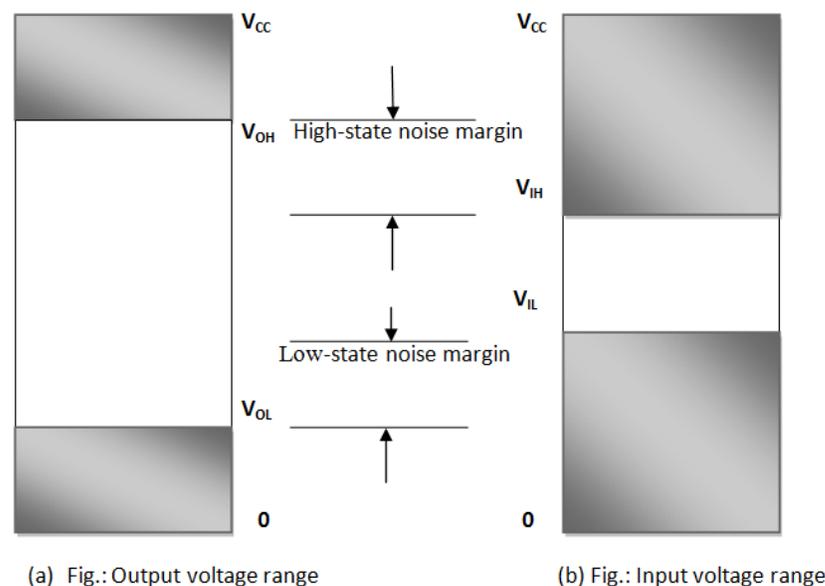


Fig. 1.6: Noise Margin

The noise margin calculation requires information on the output voltage signal and the input voltage signal of the gate. The different signals that are required in the calculation of the noise margin are given in Figure 1.6. Output gate voltages between V_{CC} and V_{OH} are considered the high-level state, and the output gate voltage between 0 and V_{OL} is considered the low-level state. Voltages between V_{OL} and V_{OH} are called indeterminate voltages. They usually do not appear under normal operating conditions except during transition between the two levels. To compensate for any noise signal, the circuit must be designed so that V_{IL} is greater than V_{OL} and V_{IH} is less than V_{OH} . The noise margin is the difference (whichever is smaller) between $V_{OH} - V_{IH}$ or $V_{IL} - V_{OL}$.

7) Operating Temperature

The operating temperature range must be very well known for the proper functioning of an IC. The following temperature range is accepted:

- For consumer and industrial applications: 0 to +70°C and
- For military applications: -55°C to +125°C

8) Power Supply Requirements

Two important characteristics of choosing the proper power supply are (1) supply voltage(s) and (2) the amount of power required by the IC.

9) Flexibilities Available

Several IC logic families have certain flexibilities. Some of these flexibilities available are described in the following points:

- The breadth of the series: types of logic functions available in the series
- Popularity of the series: If a series becomes more popular, then it can be manufactured in bulk, reducing the cost of the manufacturing and thus making it readily available because of multiple sources.
- Wired-logic capability: Additional logic can be created by connecting the outputs together. This will also not require any extra hardware.
- Availability of complement outputs: The requirement for additional inverters is eliminated.
- Type of output: Active pull-up, passive pull-up, open-collector/drain, and tristate.

1.5. RESISTANCE –TRANSISTOR LOGIC CIRCUIT

The resistor-transistor logic (RTL) was one of the widely and most popular used logic circuits before the introduction of ICs. It merely consisted of resistors and transistors and was the first logic family to be integrated. Although it is not being used nowadays, studying it will introduce some important concepts that may be used in analyzing all types of gates.

The basic circuit of an RTL digital logic family is the NOR gate, as shown in Figure 1.7. Each input is associated with one resistor and one transistor. The collector terminals of the transistors are tied together at the output. The voltage levels for the given circuit are 0.2 V for the low level, and the voltage range from 1 to 3.6 V for the high level.

The analysis of the RTL gate is straightforward. **If any input of the RTL gate is high**, the corresponding transistor is driven into saturation which causes the output to go low, irrespective of the state of other transistors. All the transistors are cut off **when all inputs are at a low voltage** of 0.2 V. The reason is that at this low voltage, $V_{BE} < 0.6$ V, meaning that the transistor is driven into cutoff mode. Due to this, the output of the circuit is high, approaching the supply voltage V_{CC} . All this implies the conditions for the NOR gate. It should be noted that the noise margin for low signal input is $0.6 - 0.2 = 0.4$ V.

The Fanout of the RTL gate is limited by the output voltage value when it is high. As the output is loaded with a number of inputs, more current is consumed by the load. This current must flow through the 640Ω resistor. It can be shown that if the current gain of the transistor h_{FE} reduces to 20, then for a fanout of 5, the output voltage reduces to 1V. And hence, if any output voltage gets below 1V, then it may not drive the next transistor into saturation as required. The power dissipation (PD) of the RTL gate is nearly 12 mW, and the average propagation delay is 25 ns.

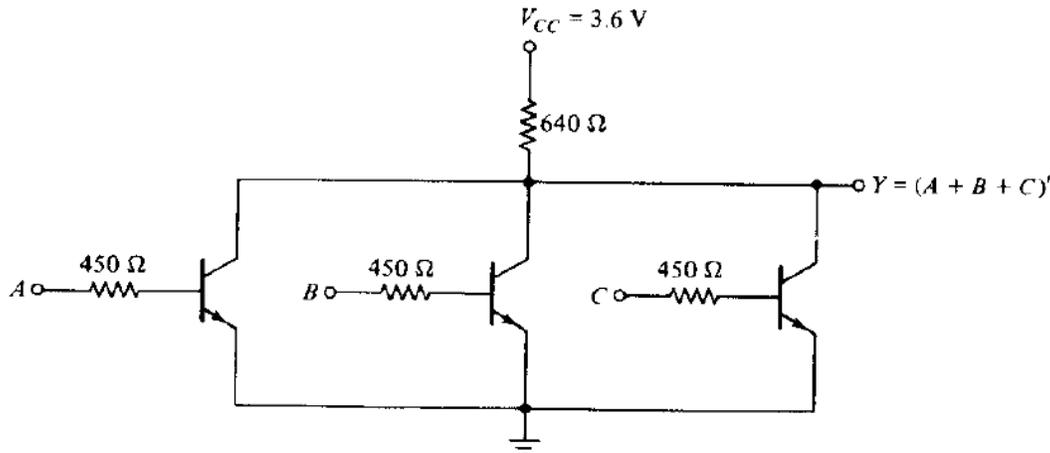


Fig.1.7: RTL basic NOR gate

Example 6: A RTL consists of which of the elements?

Answer: RTL consists of resistors and transistors.

1.6. TRANSISTOR –TRANSISTOR LOGIC CIRCUIT

The basic circuit in the TTL digital logic family is a slight improvement over the DTL gate. With time, TTL technology progressed and added improvements to the point that this logic family became the most widely used family in the digital design system. There are seven subfamilies or series of the TTL technology. The names and the characteristics of the seven TTL series appear in Table 1.4. Fanout, power dissipation, and propagation delay are also mentioned in the table. The speed-power product is an essential parameter for comparing various TTL series. This product of propagation delay and power dissipation is measured in picojoules (pJ). A low value of this product is desirable as it implies that a given propagation delay can be achieved without excessive power dissipation and vice versa. TTL ICs have a number designation that starts with 74 and follows with a suffix that identifies the series type. Examples are 7404, 74S86, and 74ALS161.

The first version of the TTL gate family was the standard TTL gate. Then it was redesigned with different resistor values to produce gates with lower power dissipation or higher speed. The propagation delay of a transistor circuit that goes into saturation depends mostly on storage time and RC time constants. Decreasing storage time decreases propagation delay. Decreasing resistor values in the circuit decreases RC time constants and reduce propagation delay. But, decreasing the

value of the resistor gives higher power dissipation, as a lower resistance draws more current from the power supply. The speed of the gate is inversely proportional to the propagation delay. In low-power TTL, the resistance value is kept more than what is used in standard TTL to reduce power dissipation. But this increases the propagation delay. In high-speed TTL, the resistance values are lowered to decrease the propagation delay at the expense of an increase in power dissipation. The next improvement in technology was the use of Schottky TTL, which used Schottky transistors to remove the storage time delay by preventing the transistors from going into saturation. This resulted in an increased circuit operation speed without an excessive increase in power dissipation. The low-power Schottky TTL reduces power dissipation by sacrificing some speed. Further work on the Schottky series led to the development of Advanced Schottky TTL with improved performance in terms of lower propagation delay and power dissipation. The Advanced Schottky TTL is the best in the TTL series providing the least power-speed product. The TTL series are available in small-scale integration (SSI) and in more complex forms such as MSI and LSI components. The main difference in the TTL family series is the internal construction of the basic NAND gate, not the digital logic they perform.

TTL Series Name	Prefix	Fanout	Power dissipation (mW)	Propagation Delay (ns)	Speed-Power product (pJ)
Standard	74	10	10	9	90
Low-power	74L	20	1	33	33
High-speed	74H	10	22	6	132
Schottky	74S	10	19	3	57
Low-power Schottky	74LS	20	2	9.5	19
Advanced Schottky	74AS	40	10	1.5	15
Advanced low-power Schottky	74ALS	20	1	4	4

Table 1.4: TTL Series and their characteristic

In any of the TTL series mentioned above, the output comes in three different types of configurations :

1. Open-collector output
2. Totem-pole output
3. Three-state (or tristate) output

1. Open-Collector Output Gate:

The basic DTL NAND gate and TTL NAND gate are shown in Figure 1.8. The TTL NAND gate is the modified circuit of the DTL NAND gate. In TTL, the input is connected to the multiple emitters in transistor Q_1 . These emitters behave as the diode in the DTL gate as they make a pn junction with a common base. The base-collector junction of Q_1 acts as another pn junction diode corresponding to D_1 in the DTL gate. Transistor Q_2 replaces the second diode, D_2 , in the DTL gate. The output of the TTL gate is taken from the open collector of Q_3 . A resistor R_L must be connected to V_{CC} external to the IC package for output to "pull up" to the high voltage level when Q_3 is off. If this resistor is not connected to the output terminal, then the output behaves like an open circuit. The basic circuit is a NAND gate.

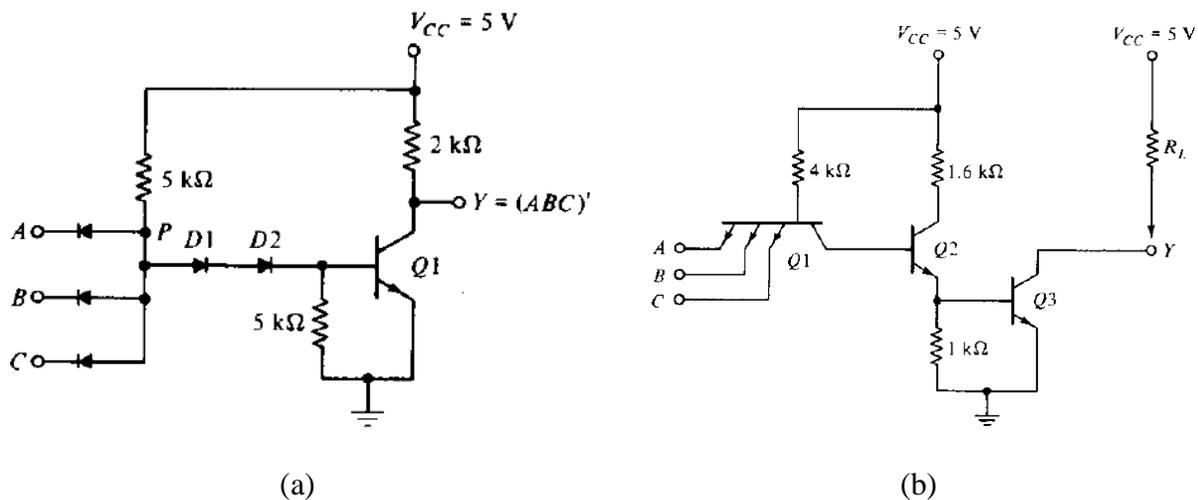


Fig.1.8: (a) Basic DTL NAND gate (b)Open collector TTL NAND gate [1]

In TTL, we consider the two voltage levels: 0.2V for Low level and voltages in the range 2.4V to 5V for high level. **If any input is low**, the corresponding base-emitter junction in Q_1 is forward-biased. V_{base} of $Q_1 = \text{input voltage} (= 0.2\text{V}) + V_{\text{be}} (= 0.7\text{V or } 0.9\text{V})$. For Q_3 to conduct, the path from Q_1 and Q_3 must overcome a potential of one diode drop in the base-collector pn junction of Q_1 and two V_{BE} drops in Q_2 and Q_3 . Therefore, the required voltage drop to make transistor Q_3 conduct is $3 \times 0.6\text{V} = 1.8\text{V}$. Since the base of Q_1 is maintained at 0.9V (0.2V+0.7V) by the input signal, the output transistor cannot conduct and is cut off. The output level will be high if an external resistor R_L is connected between the output and V_{CC} .

If all the inputs are high, transistors Q_2 and Q_3 conduct and saturate. V_{base} of $Q_1 = V_{\text{bc}} + V_{\text{be}}$ of $Q_2 + V_{\text{be}}$ of $Q_3 \approx 3 \times 0.7\text{V} = 2.1\text{V}$. Since all inputs are considered high (i.e., $> 2.4\text{V}$), the V_{BE} junctions of Q_1 are all reversed-biased. When the output transistor Q_3 saturates, the output voltage reduces to 0.2 V. That is a NAND operation.

When the TTL has to be connected to the input of other gates, in that case, the open-collector TTL gate operates without an external resistor. However, this is not recommended due to the low noise immunity. Open-collector gates can be used in three major applications

in day-to-day life: driving a lamp or relay, performing wired logic, and constructing a common bus system. Open collector output can drive a lamp placed in its output through a limiting resistor. When the output is low, then saturated transistor Q_3 forms a path for the current that turns the lamp on, whereas when the transistor is off, the lamp turns off as there is no path for current.

A wired-AND logic is performed if the outputs of various open-collector TTL gates are tied together with a single external resistor. The wired logic performed with open-collector TTL logic gates is shown in Figure 1.9. The AND operation performed by connecting the two outputs of the NAND gates using a wire is called a wired-AND connection. The Boolean function obtained from the circuit is the AND operation between the outputs of the two NAND gates.

$$Y = (AB)'. (CD)' = (AB+CD)'$$

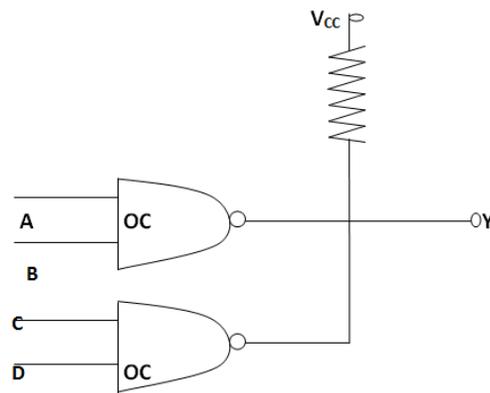


Fig.1.9: Wired-AND logic physical connection ($Y = (AB)'. (CD)' = (AB+CD)'$)

Self Assessment Question (SAQ) 2: Explain how open-collector gates can be used as a common bus.

2. Totem-Pole Output Gate:

The output impedance of a gate is usually a combination of resistive and capacitive loads. The capacitive load consists of the output transistor's capacitance, fanout gates' capacitance, and any stray wiring capacitance. When the output transitions from low to high state, the gate's output transistor also transitions from the saturation state to the cutoff state, with the total load capacitance, C , charging exponentially from the low to the high voltage level with a time constant equal to RC . For the open-collector gate, R is the external resistor marked R_L .

For a typical value of the capacitive load, $C = 15\text{pF}$ and $R_L = 4\text{K}\Omega$, the propagation delay of the open-collector TTL gate during the time it is in the turn-off position will be 35ns . If the passive pull-up resistor R_L is replaced with an active pull-up circuit, the propagation

delay decreases to 10ns. The above-described configuration is shown in Figure 1.10, and it is known as a *totem-pole* output since the transistor Q_4 "sits" upon transistor Q_3 . Totem-pole output along with the TTL gate is equivalent to the open-collector gate, except for output transistor Q_4 and diode D_1 . Transistors Q_2 and Q_3 go into the cutoff region when the output transitions to a high state because one of the inputs falls to a low state. However, the output will remain in the low state for a moment since the voltage across the load capacitance cannot change instantly (an inherent property of the capacitor).

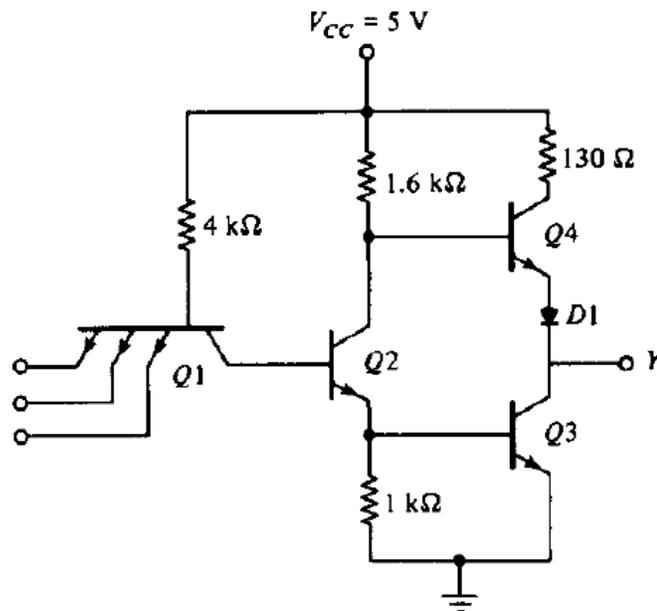


Fig.1.10: TTL gate with totem pole output [2]

When transistor Q_2 turns off, Q_4 starts to conduct because V_{CC} is connected to its base through a $1.6\text{ k}\Omega$ resistor. The current required for charging the load capacitance causes the transistor Q_4 to saturate instantly, and the output voltage increases with the value of time constant RC . But the value of R ($= 130\Omega$) + saturation resistance of Q_4 + diode resistance $\approx 150\Omega$ is much smaller than the passive pull-up resistance used in the open-collector circuit. As a result, the time taken to transition from a low to a high level is much faster.

With totem pole output circuits, the wired logic connection is prohibited. When we wire together the two totem poles, with the two gates having a high and a low output, an excessive amount of current is drawn that can produce sufficient heat, which can damage the transistors.

3. Three-state (or tristate) output:

As seen above, with totem-pole structures, the output of the two TTL gates cannot be connected just like in open-collector outputs. However, a special type of totem pole gate is available that allows the wired connection of the outputs to design a common-bus

system. When a totem pole output TTL gate has this property, it is then known as a *three-state (or tristate) gate*. The three state gates have three states of the outputs:

- I. a low-level state when the lower transistor in the totempole is on and the upper transistor is off,
- II. a high-level state when the upper transistor in the totempole is on and the lower transistor is off,
- III. a third state when both transistors of a totempole are off.

This third state results in an open circuit or high impedance, connecting many outputs directly to a common line. Thus, the three-state gates eliminate the requirement for open-collector gates in a bus configuration. Figure 1.11 (a) shows the symbol of a three-state buffer gate. If the control signal C is high, the gate is enabled and acts like a buffer with output the same as binary input; if the control signal is low, the gate is an open circuit with very high impedance. Figure 1.11 (b) shows the symbol of a three-state inverter gate. It produces a high impedance state when the control input is high. In the Figure, there are two small circles, one for the inverter output and the other for the control signal to indicate that the gate is enabled for a LOW value at C.

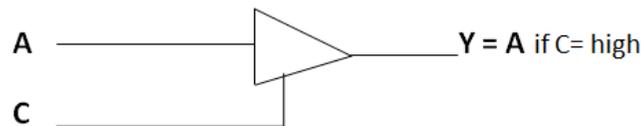


Fig.1.11 (a): Three-state buffer gate

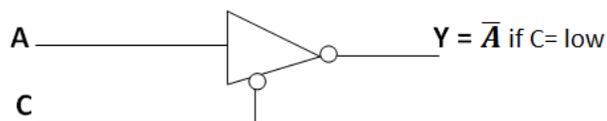


Fig.1.11 (b): Three-state inverter gate

1.7. INTEGRATED INJECTION LOGIC

The direct-coupled transistor logic (DCTL) circuit suffers from the difficulty of current hogging, making it very unsuitable. A new logic, integrated injection logic (I^2L), is developed that is based on the DCTL. I^2L uses a very small silicon chip area, consumes small power, and needs only four masks and two diffusions (compared to five masks and three diffusions for BJT). Therefore, overall we can say that I^2L is cost-effective and easy to fabricate. All the advantages mentioned above enable them to be widely used for MSI and LSI applications. It is not used in SSI and therefore is the only saturated bipolar logic used for LSI. The basic idea of the origin of I^2L

technology is to merge the components by integrating one region of the semiconductor such that it is a part of one or more devices. Since the main idea is based on merging the components, it is also known as merged-transistor logic (MTL). A significant amount of silicon chip area is saved in this process.

A. I²L Inverter

In Figure 1.12, the basic operation of I²L is illustrated using an inverter circuit. The transistor T_1 is off **when the input V_i is at a LOW level** (therefore, $I_{B1}=0$). We can thus say that the input source will act as a sink for the current I_1 . So, current I_2 will flow through the base of transistor T_2 and drives it to saturation. When transistors T_1 is OFF, and T_2 is ON, then $V_{be2} = V_{ce1} \approx 0.8$ V. On the other hand, **when the input V_i is at a HIGH level**, then the base current $I_{B1} = I_1$ + the current due to source V_i . This current is sufficient enough to drive the transistor T_1 into saturation. The value of $V_{ce1} = V_{ce,sat} \approx 0.2$ V driving the transistor T_2 into cutoff. Transistor T_1 behaves like a sink to the current source I_2 . Thus, the output voltage logic level is shown to complement the input voltage level, i.e., the transistor T_1 behaves as an inverter here. The logic swing is about $0.8\text{V} - 0.2\text{V} = 0.6\text{V}$.

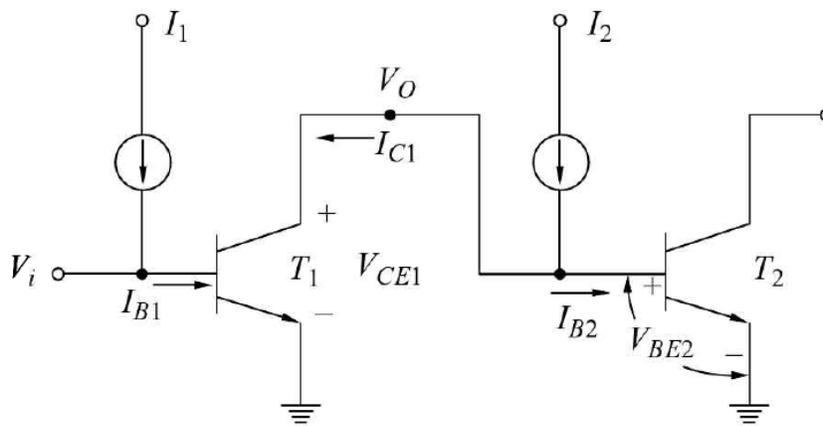


Fig.1.12: An I²L Inverter directly coupled to the following stage [1]

In Figure 1.13, we generate the output logic $\overline{A + B}$, $\overline{\overline{A + B}}$, $\overline{A + \overline{B}}$ and $\overline{\overline{A + \overline{B}}}$. Here, the base of the transistors T_1 , T_2 , and T_4 are tied together, and the emitters are also connected to the ground. Therefore, we can replace the transistors T_1 , T_2 , and T_4 in Figure 1.13 with a single transistor T_1' in Figure 1.14, having three collectors, one emitter, and one base. In the same way, other transistors with common bases can also be replaced with a single transistor (transistors T_2' , T_3' , and T_4' in Figure 1.14) having multiple collectors. Hence, we can redraw Figure 1.13 to Figure 1.14.

As seen in Figure 1.12, we require a method for supplying the base currents. To obtain these base currents for supply, the collector resistors of driving gates, shown by the dotted transistors in Figure 1.13, are treated as the base resistors of multiple-collector transistors T_2' , T_3' , shown in Figure 1.14. Likewise, the collector resistor of transistors T_1 and T_{10} behaves like the base resistors for T_2' and T_4' , respectively (as shown in Figure 1.14). V_{BB} indicates the supply voltages. The circuit drawn in the dashed box in figure 1.14 is either a part of other gates driven by the outputs or is omitted. It indicates that the I²L circuit has an open collector output, which either runs another I²L

circuit or will be connected so that the connection is through resistors to the supply voltage. Appropriate supply voltage and resistance values are used for getting appropriate output voltage levels for driving other gates, such as TTL.

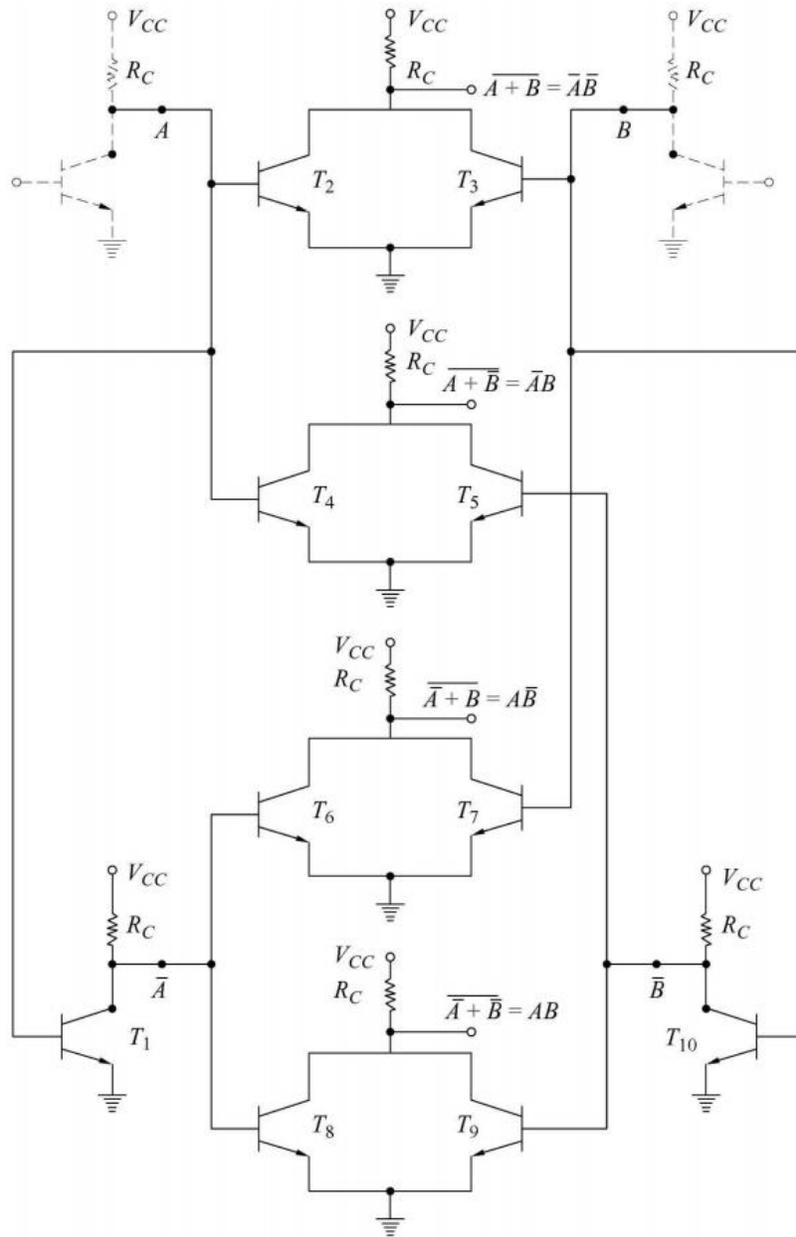


Fig.1.13:A DCTL Gate structure for generating functions of two logical variables [1]

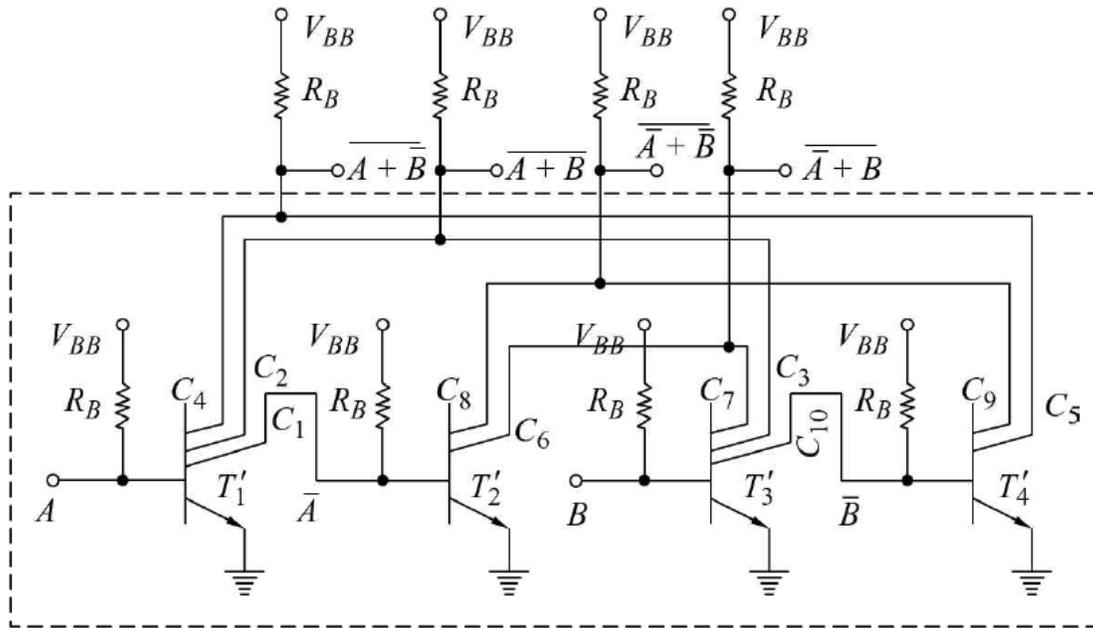


Fig.1.14: Fig.1.12 Redrawn with multiple-collector transistors [1]

1.8. EMITTER COUPLED LOGIC (ECL)

One of the fastest logic in all logic families is the Emitter-coupled logic (ECL). ECL comes under the non-saturated logic family. ECL is used in applications that demand very high-speed operation because, in ECL, we can achieve a propagation delay of 2 ns and even less than it. However, power dissipation and noise immunity are the worst in ECL among all the logic families. The main reason for very high speed is the difference amplifier configuration of the transistors, where the transistors never go into saturation, thereby eliminating the charge storage time. Since in ECL, a saturation of the transistors is eliminated, the transistors are switched between the active and the cutoff regions instead of switching between the ON and OFF states. The speed of operation of less than 1 ns for each gate is possible in ECL.

An ECL can be realized with the help of a difference amplifier with the emitters of the two transistors connected, and because of this, the logic is also called emitter-coupled logic. The basic circuit of ECL is shown in Figure 1.15. We can achieve both OR and NOR functions from the ECL output. Every input is connected to the base of a transistor. We use a HIGH voltage level of -0.8 V, and -1.8 V is considered a LOW voltage level. The ECL circuit shown in Figure 1.15 comprises a differential amplifier, a temperature and voltage compensated bias network, and an emitter follower output. The emitter outputs need a pull-down resistor for current to flow, and this is achieved by using an input resistor R_P connected to a negative voltage supply. The reference voltage is supplied by the internal temperature and the voltage-compensated bias circuit. The bias voltage is fixed at the mid-point of the signal logic swing. $V_{BB} = -1.3$ V. In the circuit, a constant value of

V_{BB} is maintained even when there is a change in supply voltage or temperatures, with the help of the transistor Q_6 and the two diodes of the voltage divider circuit. Any power supply can be used for the ground. However, to achieve the best noise immunity, we use $V_{CC} = \text{ground}$ and $V_{EE} = -5.2\text{V}$.

When any input is high, the respective transistor is on, and Q_5 is off. The transistor starts conducting when the input is -0.8V , and -1.6V appears on the emitters of all transistors. Since $V_{BB} = -1.3\text{V}$, V_{base} of $Q_5 = 0.3\text{V} + V_{\text{emitter}}$ of Q_5 . Thus, the transistor Q_5 will be at cutoff since the V_{BE} voltage of Q_5 is less than 0.6V . The current flows through the resistance R_{C2} into the base of Q_8 . This current across R_{C2} is so small that the voltage drop across R_{C2} can be ignored. Therefore, the OR output of the gate is $0 - V_{be} = -0.8\text{V}$, which is the high state. The current flowing through the resistance R_{C1} and the conducting transistor causes a voltage drop of about 1V below ground. Therefore, the NOR output is $-1\text{V} - 0.8\text{V} = -1.8\text{V}$, which is the low state.

In the same way, all the input transistors are in the off state, and only the transistor Q_5 conducts **if all inputs are maintained at a low level**. The collector-emitter junction voltage of the transistor is $V_{ce} = V_{BB} - V_{be} = -1.3\text{V} - 0.8\text{V} = -2.1\text{V}$. Since the base of each input is at a low voltage level (-1.8V), each base-emitter junction voltage $V_{be} = 0.3\text{V}$ with all input transistors in cutoff mode. R_{C2} draws current through Q_5 , resulting in a 1V drop and thereby making OR output voltage $= -1\text{V} - V_{be} = -1.8\text{V}$, or a low level. The current flowing through resistance R_{C1} is minimal and hence can be ignored. The NOR output voltage $= \text{ground voltage} - V_{be} = 0 - 0.8\text{V} = -0.8\text{V}$, which is a high state. This verifies the OR and the NOR operations of the circuit.

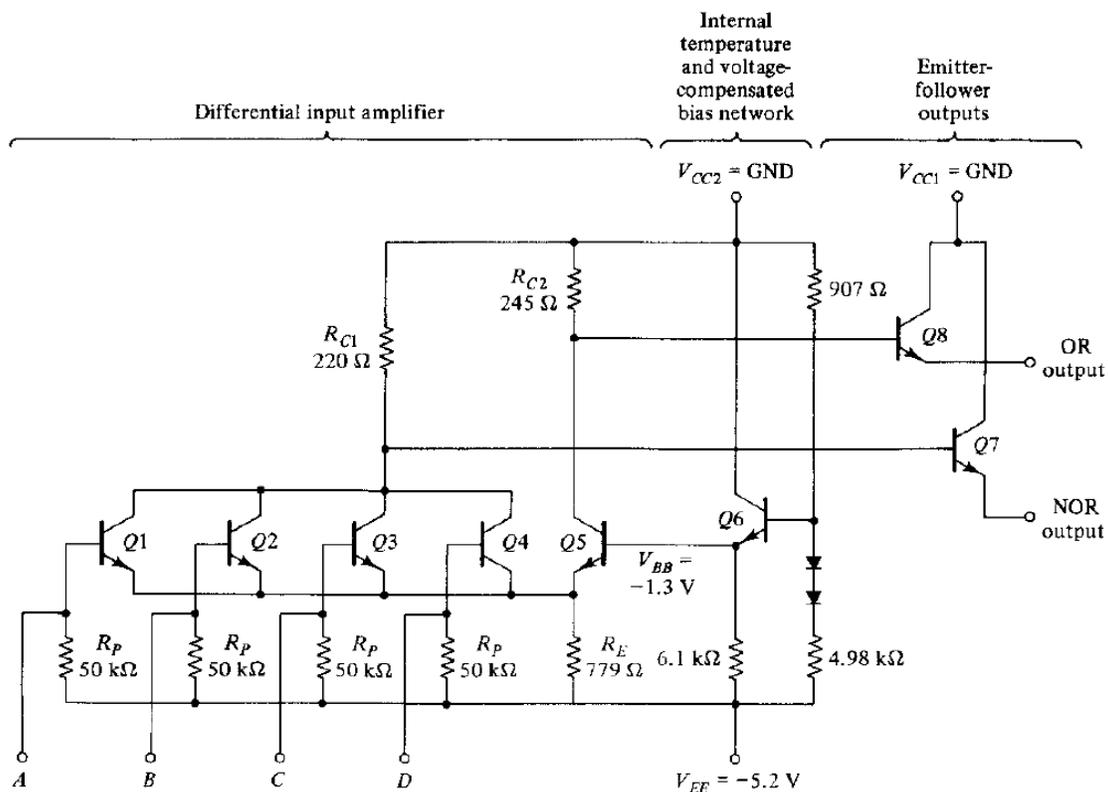


Fig.1.15: ECL basic gate [2]

1.9. MOSFET (PMOS, CMOS) AND TRI-STATE LOGIC

A field-effect transistor (FET) is a unipolar device, meaning that the current flowing in MOSFET is due to only one carrier. Junction FET (JFET) and metal-oxide-semiconductor FET (MOSFET) are the two types of FET. JFET is used in linear circuits, whereas MOSFET is used in digital circuits. MOS transistors can be fabricated in less area compared to the BJTs. Here in this chapter, you will study about MOSFET.

MOSFET's basic structure is shown in Figure 1.16. The *p-channel MOS* has a lightly doped substrate of n-type silicon material. Two regions are heavily doped by diffusion with a p-type impurity to make *source and drain*. The region between this source and drain is called a *channel*. *Gate* is separated from the channel by an insulated dielectric of SiO_2 . The negative voltage at the gate induces an electric field in the channel, which attracts holes from the substrate. As the magnitude of this negative voltage increases (keeping a sufficient voltage between the source and drain), more holes will start to accumulate below the gate, thereby increasing the conductivity and the amount of current flowing from the source to the drain.

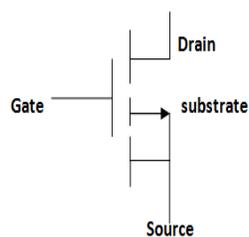


Fig.1.16 (a): p-channel

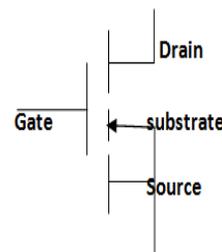
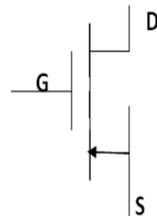


Fig.1.16 (b): n-channel

MOSFETs have high fabrication density and very low power dissipation. Due to these two features, the MOSFETs are becoming more popular in digital logic circuits. Circuits using only p-channel and n-channel devices are called PMOS and NMOS logic, respectively. If we fabricate both the p-channel and n-channel devices on the same chip, then it is called complementary-MOSFETs, and the logic based on these devices is known as CMOS logic. CMOS is being very popular because of its extremely low power dissipation. Fabrication of MOS digital ICs is usually done with the help of MOSFETs, and even the components like resistors can be formed using MOSFETs.

As you know, the fabrication of resistors needs a large area in the chip. Hence, if we use only MOSFETs in the ICs, then the size of the IC can be reduced, and, therefore, a very high fabrication packing density, or in other words, a higher number of components per unit of chip area, can be manufactured. Therefore, with the help of the MOS logic family, we can have a large-scale and

very large-scale integration. MOS is used to manufacture microprocessors, memories, and peripheral devices.

MOS Inverter: The basic MOS gate is an inverter, as shown in Figure 1.17. Here T_1 is the enhancement MOSFET acting as a driver, and T_2 can be an enhancement or depletion MOSFET acting as a load resistor. In the MOS circuits, as shown in Figure 1.17, the logic voltage levels are $V(0) \approx 0$ and $V(1) \approx V_{DD}$.

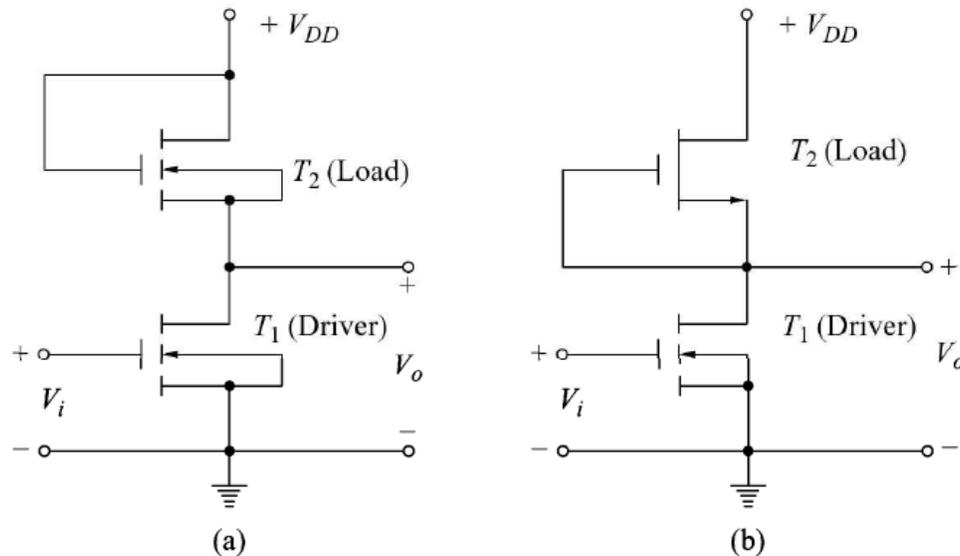


Fig.1.17: A MOS Inverter with (a) Enhancement load, (b) Depletion Load

The current hogging issue in the DCTL is not present in MOS logic circuits, despite both DCTL and MOS logic circuits having identical configurations.

PMOS and NMOS Logic: P-channel Metal-oxide-semiconductor (PMOS) is based on p-channel enhancement mode MOSFET. During the 1960s to 1970s, PMOS logic was widely used for LSI. Later it got replaced by NMOS and CMOS devices. PMOS transistors operate by creating an inverse layer in the n-type transistor body. This inversion layer is known as the p-channel. Just like MOSFETs, PMOS transistors have four modes: cutoff, triode, saturation, and velocity saturation. PMOS gates have the same arrangement as NMOS gates if the voltages are reversed.

CMOS Logic: A complementary MOSFET (CMOS) is the series connection of p-channel and n-channel MOSFETs with drains connected and output taken from the common drain. The two gates of p and -channel MOSFETs are connected, and the input is applied to this common gate. The fabrication of CMOS is a complicated process as both the p-channel and n-channel enhancement MOS devices are fabricated on the same chip. However, this reduces the packing density. But since the power dissipation in CMOS is negligible, it makes them more suitable for battery-

operated systems. Figure 1.18 shows the basic CMOS logic circuit operating as an inverter. The logic levels for this circuit are 0V (logic 0) and V_{DD} (logic 1).

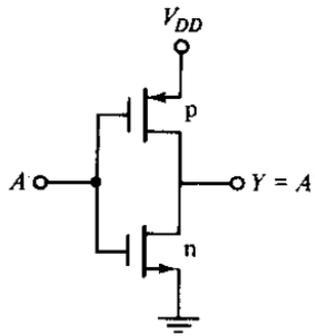


Fig.1.18: CMOS Inverter

V_{DD} is the source terminal of the p-channel, whereas the source terminal of the n-channel is at the ground. $V_{DD} = 3V$ to $18V$. To understand the basic operation of CMOS, you need to review the inverter operation:

- 1) N-channel MOS conducts when its gate-to-source voltage is positive.
- 2) P-channel MOS conducts when the gate-to-source voltage is negative.
- 3) Either device turns off when the gate to the source voltage is zero.

When the input is low, then the voltage at both gates is 0. The input voltage is at $-V_{DD}$ with respect to the source of the p-channel device and is at 0V with respect to the source of the n-channel. As a result, the p-channel device turns ON, and the n-channel device turns OFF. This forms a low impedance path between V_{DD} and the output and a high impedance path between the output and the ground. Hence under normal operating conditions, the output approaches high-level V_{DD} . **When the input is high**, both gates are at V_{DD} , and the situation gets reversed, i.e., the P-channel device is OFF, and the n-channel is ON. The output approaches a low-level state of 0V. This corresponds to the NOT logic operation.

The substrate capacitance limits the speed of CMOS. The latest technology, Silicon on sapphire (SOS), is being used to fabricate the microprocessor to decrease the effect of substrate capacitances. In MSI and LSI applications, CMOS is widely popular. CMOS is the only logic used for the VLSI devices fabrication.

Tri-state Logic: Generally, any logic circuit has two output states, LOW and HIGH. A HIGH state means the output is not in the low state. Similarly, a LOW state means the output is not in the high state. In microcomputers, microprocessors, and other digital systems, we require a common line called a bus to connect the outputs of several different gates to drive several gate inputs. This connection may lead to many problems, which are listed below:

- 1) To avoid huge current drain from power supply, the outputs of the totempole for TTL or active pull-up/down outputs for CMOS should not be connected together. Otherwise, ICs will get damaged.

- 2) Common-collector (or drain) and Open-collector (or drain) outputs may be connected together, and the resistor may be connected externally to the circuit. This raises loading problems. It also affects the speed of operation.

To solve these problems, an advanced circuit is used where a third state is developed at the output state, known as *the third state* or *high impedance state*, besides LOW and HIGH states. This type of advanced circuit is called TRI-STATE, tri-state logic (TSL), or *three-state logic*. TRI-STATE has been registered as the trademark of National Semiconductor Corporation of the USA. One functional difference between the TSL and wired-ORs: In wired-OR: $Y=Y_1+Y_2$, whereas in TSL, the output does not operate the Boolean function; instead, it has the ability to multiply many functions.

1.10. LOGIC FAMILIES AND THEIR PERFORMANCE CHARACTERISTICS

In the previous sections, you have studied different digital logic families. You are now familiar with the performance characteristics parameters and their significance in digital logic families. In this section, we will compare the different logic families with their performance characteristics such as Fanout, noise-margin, propagation delay time, power dissipation, current and voltage parameters, operating temperature range, flexibilities etc.

Logic Families	RTL	TTL	I ² L	ECL	MOS	CMOS
Parameters						
Basic gate	NOR	NAND	NOR	OR-NOR	NAND	NOR or NAND
Power dissipation (mW)	12	<10	6nW-70μW	40-55	20	50
Noise immunity	Nominal	Very good	Poor	Poor	Good	Very good
Propagation delay (ns)	12	<10	25-250	0.75	300	10-70
Figure of merit(pJ)	144	4-100	<1	40	60	0.24-0.7
Scale functions	High	Very high	LSI only	High	Low	High

Table 1.5: Comparison of different digital logic families with their performance characteristics

SUMMARY

Essential features of all the logic families have been explained, and the following important conclusions can be listed:

1. RTL and DTL circuits are rarely used nowadays because of low speed, more power dissipation, and low Fanout.
2. TTL has become one of the popular logic and is now available in various series with a wide range of operating speeds, power dissipation, and Fanout.
3. TTL has a large number of functions in SSI and MSI.
4. TTL ICs are available with totem-pole output (decreases the speed power product), open collector output (wired-AND operation and bus operation is possible using this), and tri-state outputs (TSL are ideal for bus connections).
5. TTL ICs with totem-pole outputs cannot be used for wired-OR or bus operation.
6. Unused inputs of TTL should not be left unconnected or floating.
7. ECL is the fastest among all logic families having a low noise margin and high power dissipation as its main disadvantage.
8. I²L is the only saturated bipolar logic used in LSI primary due to the following two properties: less silicon chip area required and low power consumption.
9. Since MOS devices occupy very small Silicon chip areas compared to their counterpart bipolar devices, they are the most popular logic for LSI. The major drawback of MOS devices is their low speed.
10. PMOS is not used so often.
11. CMOS is a widely used logic circuit because: (1) It has the lowest speed-power product, (2) it requires very little power to operate, (3) It has led to VLSI technology, and (4) Many large numbers of functions can be very quickly created using CMOS.
12. Active pull-up/down, open drain output and tristate (TSL) outputs are available in CMOS ICs.
13. Better noise immunity can be achieved in CMOS logic if Schmitt trigger inputs are used at the inputs of the CMOS ICs.
14. Active pull-up and pull-down devices cannot be connected together with the outputs of the CMOS for bus operations and wired-OR logic.

GLOSSARY

Active pull-up A circuit with active devices is used to pull up the output voltage of a logic circuit from LOW to HIGH.

Bipolar Logic circuits using bipolar junction semiconductor devices.

Bus A group of conductors carrying a related set of signals.

CMOS (Complementary metal-oxide-semiconductor) A MOS device that uses one p-channel and one n-channel device to make an inverter circuit.

ECL (Emitter coupled logic) A form of bipolar logic circuit that uses an emitter-coupled configuration.

Fanout Maximum number of similar logic gates which a logic gate can drive.

Figure of merit (FoM) Product of propagation delay time and power dissipation.

I²L (Integrated injection logic) Bipolar logic circuit uses only bipolar transistors.

LSI An IC chip having 100 to 1000 gates or 1000-10,000 transistors

MSI An IC chip having 13 to 99 gates or 100-1000 transistors.

Noise immunity A circuit's ability to withstand noise.

Noise Margin Measure of the noise that can be tolerated by a logic circuit.

Open collector output An output of an IC at the collector terminal of a BJT not connected to any other component inside the IC.

Pull-up resistor A resistor connected between output and supply voltage.

SSI An IC chip having up to 12 gates or 100 transistors.

Three state gate (tristate gate) Gate having a 1, 0, or high impedance outputs states.

Totem-pole output is the same as an active pull-up.

TSL (tristate logic) is the Same as tristate output.

Unipolar logic Logic circuits that have only MOS devices.

VLSI An IC chip having above 1000 gates or above 10,000 transistors.

REFERENCES

1. Modern Digital Electronics, R P Jain, Fourth Edition
2. Digital Logic and Computer Design, M Morris Mano
3. TTL Data Book. Dallas: Texas Instruments, 1988
4. CMOS Logic Data Book. Texas Instruments, 1984

QUESTIONS

(Should be divided into Short Answer Type, Long Answer Type and MCQs)

Short Answer type

1. A logic family using BJTs is known as alogic family.
2. A unipolar logic family uses only.....devices.
3. Figure of merit of a digital IC is given by.....
4. Outputs of TTL gates with active pull-up mustconnected together.
5. The input terminal of a CMOS circuit must.....
6. The states of a TSL are.....
7. is the fastest logic family.
8. TTL compatible CMOS logic families are.....
9. Logic family that combines the best features of CMOS and bipolar logic is.....
10. The state of a tristate output is

Long Answer type

- 1) Is it possible to use TTL to ECL translator for CMOS to ECL interfacing? Justify your answer.
- 2) Write a brief note on RTL, TTL, I²L, and ECL, and then compare all these.
- 3) Explain MOSFET. What are its types?
- 4) Write a short note on CMOS.
- 5) Write notes on:
 - i. Power dissipation
 - ii. Propagation Delay
 - iii. Fanout
 - iv. Current and voltage parameters
 - v. Noise margin and noise immunity
 - vi. Figure of merit
 - vii. Flexibility

MCQs

- 1) The full form of the abbreviations TTL and CMOS in reference to logic families are
 - a) Triple Transistor Logic and Chip metal oxide semiconductor
 - b) Tristate Transistor Logic and Chip metal oxide semiconductor
 - c) Transistor-Transistor Logic and Complementary metal oxide semiconductor
 - d) Transistor-Transistor Logic and Complementary metal oxide semiconductor
- 2) The output of 74 series of TTL gates is taken from a BJT in
 - a) Totem-pole and common collector configuration
 - b) Either totem-pole or common collector configuration
 - c) Common base configuration
 - d) Common collector configuration
- 3) Commercially available ECL gates use two ground lines and one negative supply in order to
 - a) Reduce power dissipation
 - b) Reduce loading effect
 - c) Increase Fanout
 - d) Eliminate the effect of power line glitches or the biasing circuit
- 4) Among the digital IC families, ECL, TTL, and CMOS
 - a) ECL has the least propagation delay
 - b) TTL has the largest Fanout
 - c) CMOS has the biggest noise margin
 - d) TTL has the lowest power consumption

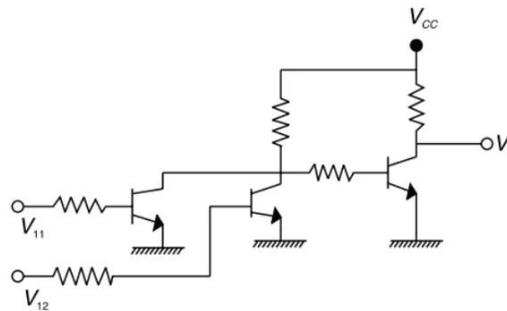
- 5) In standard TTL, the 'totem pole' stage refers to
- The multi-emitter input stage
 - The phase splitter
 - The output filter
 - Open collector output stage

- 6) Given that for a logic family,
- The high-level input voltage, V_{IH}
 The low-level input voltage, V_{IL}
 The high-level output voltage, V_{OH}
 The low-level output voltage, V_{OL}

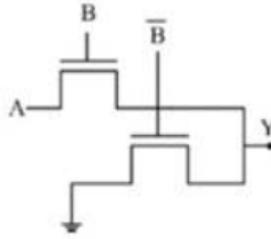
The correct relationship is:

- $V_{IH} > V_{OH} > V_{IL} > V_{OL}$
 - $V_{OH} > V_{IH} > V_{IL} > V_{OL}$
 - $V_{IH} > V_{OH} > V_{OL} > V_{IL}$
 - $V_{OH} > V_{IH} > V_{OL} > V_{IL}$
- 7) The ICs used in watches and calculators are of
- TTL
 - ECL
 - MOS
 - CMOS

- 8) The Figure shows the circuit of a gate in the RTL family. The circuit represents a



- NAND
 - NOR
 - AND
 - OR
- 9) The logic functionality realized by the circuit shown below is



- a) OR
- b) XOR
- c) NAND
- d) AND

Answers:

Answers to Self-Assessment Questions:

SAQ.1)

$$R_c(\max) = \frac{V_{CC} - V_{OH}}{I_{OH} + 8I_{IH}} = \frac{(5 - 2.4) \times 1000}{250 + 8 \times 40} K\Omega = 4.56 K\Omega$$

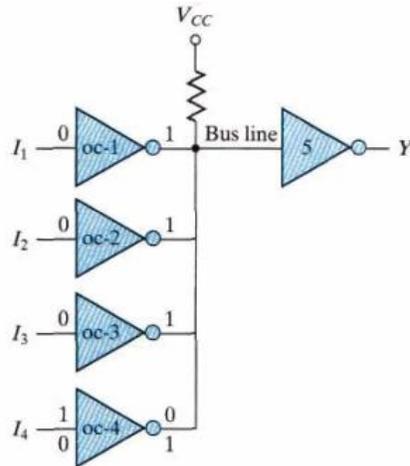
$$R_c(\min) = \frac{V_{CC} - V_{OL}}{I_{OL} + 8I_{IL}} = \frac{(5 - 0.4)}{16 - 8 \times 1.6} K\Omega = 1.44 K\Omega$$

Therefore, $1.44 K\Omega < R_c < 4.56 K\Omega$

SAQ. 2)

We can use the Open collector gates as a common bus by tying them together. At any time, we should maintain all the outputs of the gate connected to the bus, except one, as high. If we want to transmit 1 or 0 on the bus, the corresponding gate must be in a high or low state. We must use a control circuit to select the particular gate that drives the bus at any given time.

The Figure below shows the connection of the 4 sources connected to the common bus line. Each of the 4 inputs drives each open collector inverter. The inverters' outputs are tied together to form a common bus line. From the Figure, we can see that 3 inputs are set at 0, producing an output of 1 at the common bus line. The fourth input, I_4 , can thus now transmit the information through the common bus line to inverter gate 5. Using the wired-AND logic, the input to the inverter gate 5 will be the wired-AND functions of the outputs of all the inverter gates 1, 2, 3 and 4, to which the inputs I_1, I_2, I_3 , and I_4 are connected, i.e., input to inverter gate 5 = $1 * 1 * 1 * \bar{I}_4 = \bar{I}_4$. Therefore the output of the inverter gate 5 = $\overline{\bar{I}_4} = I_4$. Hence, we successfully transmitted the information at I_4 through the common bus line.



Short type answers:

1. Bipolar logic family
2. FETs (MOS)
3. Product of propagation delay time and power dissipation
4. Must not be connected
5. Should be either connected to the ground or supply voltage
6. Logic 1, logic 0, and high impedance state.
7. ECL
8. CMOS logic family
9. Bipolar CMOS logic
10. Logic 1, logic 0, and high impedance

MCQs Answers:

- 1) C
- 2) B
- 3) D
- 4) A
- 5) C
- 6) B
- 7) D
- 8) D
- 9) D

UNIT 2**MEMORY ORGANISATION AND
EXPANSION**

STRUCTURE

- 2.1 Introduction
- 2.2 Objectives
- 2.3 Basic terms and ideas
- 2.4 Classification and Characteristics of memories
 - 2.4.1 Principle of Operation
 - 2.4.2 Physical Characteristics
 - 2.4.3 Mode of Access
 - 2.4.4 Fabrication Technology
- 2.5 Memory organization and expansion
 - 2.5.1 Expanding memory size
 - 2.5.1.1 Expanding word size
 - 2.5.1.2 Expanding word capacity
- 2.6 Storage Devices
- 2.7 Memory Addressing
- 2.8 Sequential programmable logic devices
- 2.9 Content addressable memory

Summary

Glossary

References

Terminal Questions

Answers

2.1 INTRODUCTION

There is a need to store digital information in the digital processing system. This information comprises a binary form of the coded instructions, data to be processed, and intermediate and final results. The part of the digital system with storage capability is called *memory*. Semiconductor memory, also called MOS memory, is a semiconductor device storing data within the MOS memory cells on a Silicon integrated circuit memory chip. Earlier, magnetic tapes were commonly used as storage devices. As semiconductor technology has advanced, it is now become possible to make different types and sizes of semiconductor memories. Semiconductor memory is the main memory component of a microcomputer-based system to store programs, information, and other data. Semiconductor memory is immediately available to the microprocessor. Memories are becoming more popular because of their compact size, high speed, low cost, ease of expansion, and high reliability. Engineers designing digital processors should therefore know the working and limitations of the semiconductor memories thoroughly.

Semiconductor memories are usually constructed with the property of *Random access*, which means that the same amount of time is required to access information from any location in the memory. This property allows accessing the data efficiently in any random order. This is in sharp contrast to the data storage devices such as hard disks, CDs, pen drives etc., where we read and write data consecutively so that it can be accessed in the same order as it was written. There are many types of semiconductor memories depending on the semiconductor technology used. The two most common and widely used Random Access Memory (RAM) are (1) Static RAM (SRAM)- many MOS transistors per memory cell are used in this, and (2) Dynamic RAM (DRAM)- many MOS transistors and MOS capacitors per memory cell are used in this. In non-volatile memories such as EPROM, EEPROM, and flash memory, floating gate memory cells consisting of a single floating gate MOS transistor per cell are used.

Memory is a group or collection of cells that are used for the storage of data. A memory cell stores a word, and a word has binary information in groups of bits. A word memory is a collection of '1s' and '0s'. Following are the common terminologies used-

- Bit- Binary digit is logical 0 and 1
- Nibble- Combination of 4 bits
- Byte- Collection of 8 bits
- Word- set of bits treated as a unit.

A computer stores the data in words. The following are terminologies used for the higher order:

- Kilobyte or KB (1 KB) = 1024 bytes
- Megabytes or MB (1MB) = 1024 KB
- Gigabytes or GB (1GB) = 1024 MB
- Terabytes or TB (1TB) = 1024 GB

The memory capacity determines how much total number of bytes it can store. The capacity of semiconductor memory can be increased. The memory size is specified in terms of the number of words or bits in the word. Memory size can be increased by increasing the number of words or word size with the help of multiple chips.

In this unit, you will familiarize yourself with semiconductor memory types, characteristics of memory based on their operation, and fabrication technology. You will learn the memory organization and its expansion. Various storage devices will also be discussed briefly.

2.2 OBJECTIVES

After studying this unit, you should be able to-

- Learn about semiconductor memories and their classification based on their operation
- Be familiar with different other types of memories
- Utilize the memory organization and its expansion.
- Know the different storage devices with their applications
- Know about the memory address
- Solve problems based on the memory size, number of address lines, etc.

2.3 BASIC TERMS AND IDEAS

In any digital processing system, it is always required to *store* and *retrieve* digital data whenever it is required. In the early days, magnetic tapes/ discs were used early for this purpose. Nowadays, *semiconductor devices* are used to build these memory devices as they are easy to use, smaller in size, reliable, have a high speed of operation, and are easy to expand.

Memories can be constructed using *Flip-Flops*. In such a memory, the data can be written into memory and randomly read from memory. This is also known as *random access memory*. Figure 1 shows the *basic 1-bit memory element* with reading and writing operations. Memory integrated circuits are built using such basic 1-bit memory elements.

The main component of this 1-bit memory cell is *D-flip-flop*. The output 'Q' of the D flip-flop is the input 'D,' as long as the CK is at logic 1. When the signal at CK terminals transitions to logic 0, the output at terminals 'Q' does not change. It still outputs the value present on 'Q' before the CK terminal transitioned to logic 0. Figure 1, shown below, has 3 inputs, namely, D_i (*data input*), R/\bar{W} (*Read/Write control*), A_n (*address select*) and one output D_o (*data output*). $A_n = 1$ will select the memory cell for reading and writing operations. $R/\bar{W} = 1$, enables the reading operation from the cell, whereas $R/\bar{W} = 0$ enables the writing into the cell operation. As long as $A_n = 0$, the memory element is in hold mode, and all the read and write operations are blocked, thereby protecting its hold output till required.

The read operation is non-destructive, meaning the stored bit can be read as often as required without disturbing the information. Since the stored information is protected as long as the power is on, this type of memory is also called *volatile memory*. It is not required to clear the cell before

writing into it. The earlier bit/information will get automatically destroyed when something is written onto the cell.

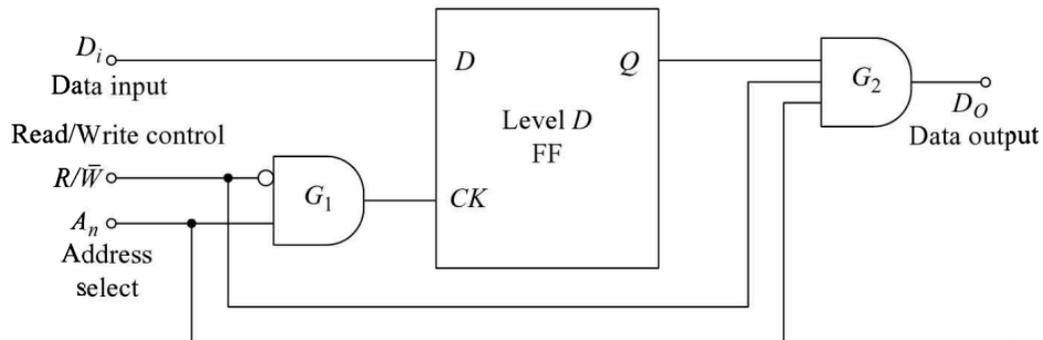


Fig.2.1: A basic 1-bit read/write memory cell

2.4 CLASSIFICATION AND CHARACTERISTICS OF MEMORIES

We can classify the different memory devices based on the principle of operation, physical characteristics, mode of access, and fabrication technology.

2.4.1 Principle of operation

Based on the principle of operation, semiconductor memories can be divided into two types of memories:

1. Random Access Memories (RAM)
2. Read Only Memories (ROM)

- **Random Access Memories (RAM)**

Random Access Memory is constructed with the property of Random access, which means that the same amount of time is required to access information from any location in the memory. This property allows accessing the data efficiently in any random order. The data can be stored and read to and from this memory many times. RAM is a volatile memory, i.e., the data is saved in RAM till the power supply is ON. When the supply is turned off, all the data stored in RAM will be lost. Therefore, this memory can be easily programmed, erased, and reprogrammed by the user. RAM is also called **read-and-write Memory, main Memory, or primary Memory**. RAM further can be divided into:

- i. Static RAM (SRAM)
- ii. Dynamic RAM (DRAM)

- i. **Static RAM (SRAM)**

SRAM consists of flip-flops as storage elements. Here, static means the memory will hold its content until the time supply is ON. Therefore, SRAM is also called volatile memory. The fabrication of SRAM can be done using either BJT or MOSFET; hence, SRAMs are comparatively faster. SRAM can be either synchronous or asynchronous. The operations in an SRAM are synchronized with the system clock, whereas in an asynchronous SRAM, the operations are not synchronized with the system clock. Usually, synchronous SRAMs are equipped with a *burst feature*, which allows a single address to read and write at up to four locations in the memory. The *Mod-4 counter*, which is a component of the burst logic circuit, produces a series of 4 internal addresses using the two lowest order address bits as 00, 01, 10, and 11 on succeeding clock pulses. The base address, or the external address applied, always comes first in the sequence.

ii. Dynamic RAM (DRAM)

DRAM stores the data in the form of charges in the capacitor and transistor pair available in the memory cell. Hence, it requires periodic refreshing and is slower than SRAMs. DRAMs can be fabricated using only MOSFETs and can store much more data than SRAM for a given size and cost. Different types of DRAM are:

- Fast Page Mode DRAM (FPM DRAM)
- Extended Data Out DRAM (EDO DRAM)
- Burst EDO DRAM (BEDO DRAM)
- Synchronous DRAM (SDRAM)

A comparison between SRAM and DRAM is given in Table 1.

Properties	SRAM	DRAM
Speed	faster	Less
Cost	More	less
Storage	Less	More
Fabrication	By bipolar devices	Only by unipolar devices
Power consumption	More	less
Applications	CPU's speed-sensitive cache	Larger system RAM space

Table1: Comparison between SRAM and DRAM

• Read Only Memories (ROM)

The form of memory where the data are once written cannot be changed even if the power supply goes off. The Read Only Memory (ROM), as its name suggests, is mostly used for reading memories. It does not necessarily follow that data cannot be written into it, either, since reading from memory is pointless unless data can also be written to and stored in the memory. Writing to

ROM requires a more involved process than writing to RAM, and it is often done outside of the system where ROM is used. ROM is used where permanent memory is needed. In other words, it is used to store information that is fixed, such as tables for various functions, fixed data, and instructions. Low cost, high speed, permanent memory features, and flexibility in the designing system have made ROM popular among other memories. ROM is utilized for the LSI manufacturing process.

Only at the time of manufacturing, as the final step of fabrication, is read-only memory programmed with the data provided by the user. After the fabrication process is completed, the stored data can not be erased. When the information is written in ROM, it is known as *programming* the ROM. ROM, further, can be divided into two types:

- i. Programmable ROM (PROM)
- ii. Mask ROM (MROM)

i. Programmable ROM (PROM)

Programmable Read-Only Memory is referred to as a PROM. This sort of ROM is programmed by the user using a special circuit known as a PROM programmer. When a PROM is programmed, its data is stored in this ROM permanently. This kind of ROM is appropriate for the storage of data that is of a permanent nature.

The data is stored as a charge on a capacitor in it. Each cell has a charge storage capacitor. PROM comprises an array of fusible links. Due to a leak in charge, the data tends to be lost. Again, PROM is divided into the following:

- Erasable PROM (EPROM)
- Electrically Erasable PROM (EEPROM)
- Flash memory
- PCM

Erasable PROM (EPROM)

Erasable Programmable Read Only Memory is abbreviated as EPROM. This type of ROM can be reprogrammed again and again. The process of erasing the data can be done using two methods: (1) By exposing the silicon chip to ultraviolet light and (2) By using electricity. If UV radiation is used for erasing purposes, then the ROM is called EPROM, whereas when electricity is used to erase the contents, then it's called EEPROM. There is a window in EPROM to enable the light to reach the silicon of the chip.

Electrically Erasable PROM (EEPROM)

Electrically Erasable Programmable Read Only Memory is abbreviated as EEPROM. Applying a higher-than-normal electrical voltage will allow you to delete and reprogram (write to) the device repeatedly. EEPROMs are *non-volatile ROMS* that allow for the erasure and reprogramming of single bytes of data. Because of this, EEPROM chips are sometimes known as *byte erasable chips*. In computing and other electronic devices, EEPROM is typically used to store small amounts of data. Memory cells of EEPROM are made up of floating-gate MOSFETs.

EEPROM was created by scientists at Hughes Aircraft and Intel in the late 1970s and early 1980s to replace EPROM and PROM Memory. The EPROM technology was widely utilized before EEPROM. If EPROM memory chips were exposed to UV light, they could be programmed and then erased. However, the chips could not be erased electrically. Moreover, the EPROM erasing procedure took upwards of an hour, which was adequate for the development conditions of the time but provided no flexibility for potentially faster environments in the future.

EEPROM technology was developed to meet these difficulties. EEPROM can be erased and programmed electrically based on the current EPROM structure. The typical lifespan of an EEPROM chip is 10,000–100,000 write cycles, which is significantly longer than the write cycles of an EPROM chip.

Flash memory

Flash memory is a type of *non-volatile memory* that can be erased and reprogrammed electrically. In contrast to EEPROM, which is erased and reprogrammed at the *byte level*, it is a type of EEPROM that is erased and programmed in the circuit in *large blocks*. The size of blocks can be in the range of hundreds to thousands of bits. These memories are based on floating-gate transistors, and floating-gate transistors are used to store bit information. It is easier to update flash memory because it can be written to memory in blocks rather than byte size. However, flash memory is less useful than RAM because RAM must be addressable at the byte level rather than the block level. It is also known as ‘non-volatile RAM.’ Flash memory got its name from the fact that a large block of memory could be erased at once, i.e., in a single action or ‘flash.’

Flash memory is used in consumer storage devices such as mobile phones, digital cameras, and computer memory sticks. Flash memory is also called *Flash Storage*. Flash memories can be of two types:

- i. NAND Flash memory- In a NAND flash cell, which is smaller and has fewer bit lines, floating gate transistors are connected together to increase storage density. NAND memory stores data in pages, which are larger than bytes but smaller than blocks.
- ii. NOR Flash memory- NOR flash connects different memory cells in parallel without sharing components, enabling random access to data. Data can be quickly read from the NOR flash. However, writing and erasing data is much slower than NAND flash memory. In NOR flash, information is saved at the byte level.

Some advantages of flash memories are listed below:

- a) Flash memory has high-speed transfer rates.
- b) Flash memories are compact in size.
- c) Due to their stability, flash memories are recommended for use on mobile devices.
- d) Flash storage does not require any physical components to function. As a result, it requires very little energy to operate, and no noise is produced.
- e) Flash memories are highly portable.

However, there are some limitations of Flash memory:

- a) It can not delete the data bit by bit, byte, or word. It can delete only block by block.
- b) Flash memory is always more expensive per gigabyte when compared to traditional hard disc drives.
- c) Although flash memory is more durable than hard disc drives, it does not have an infinite lifespan.
- d) Bit Flipping problems occur in NAND memory compared to NOR memory.
- e) Less reliability due to bad block. Bad blocks can't be used for storage systems.
- f) The usage of NOR and NAND memory is different.
- g) Many NAND flash memory devices use a process called program/erase to store data quickly, but this process eventually wears out and destroys flash drives. Therefore, heavy write loads cannot be supported by flash storage.

Phase Change Memory (PCM)

PCM stands for Phase Change Random Access Memory (P-RAM or just a Phase Change Memory). It is a non-volatile computer memory (NVRAM) type and is sometimes referred to as “*perfect RAM*” because of its exceptional performance features. Different industrial/academic groups also refer to them as phase change memory as PCM and PCRAM.

PCM is a cutting-edge form of memory technology with numerous promising storage application use cases. Fast RAM speeds are promised by PCM, but they can also be utilized to store data with less power consumption. Though its widespread commercial use in storage has been minimal, PCM is a technology that has been in various phases of study and development since the late 1960s. Nevertheless, many industries are investing in this technology, which has a promising future.

One of the main benefits of NVRAM is that information may be stored in PCM, unlike DRAM, even when the system is switched down. The non-volatile storage is made possible by the use of unique alloys, such as Germanium Antimony Tellurium (GST), in phase change memory. The alloy may be heated to change into one of two “phases” (crystalline or amorphous), which is how data is stored in the alloy. When compared to other memory storage technologies, such as NAND Flash Memory, PCM offers several advantages such as:

- **Faster Write Cycles:** PCM offers the possibility of much faster write cycles than NAND flash.
- **Faster Access Time:** Access times under 5s can be achieved with PCM, according to research.
- **Endurance:** Vendors and researchers estimate that PCM can handle more write cycles than NAND memory because it doesn't have to delete data first.
- **Reduced Energy Consumption:** PCM promises to use less power than its NAND competitors.
- **Executable PCM.** It can store data like NAND and execute code like the current DRAM.

ii. Mask ROM (MROM)

The contents of a mask ROM (MROM) are programmed by the IC manufacturer (rather than by the user). The term mask comes from the field of IC fabrication, where regions of a chip are masked off during the photolithography process. When working on a project, it's customary to employ rewritable non-volatile memory (such as EPROM or EEPROM) and switch to mask ROM once the code is complete. For instance, Atmel microcontrollers are available in EEPROM and mask ROM formats.

The affordability of mask ROM is its biggest advantage. Mask ROM is the most compact form of semiconductor memory per bit. Mask ROM is much less expensive than any other type of semiconductor memory because the cost of an IC heavily relies on its size. However, there is a considerable turnaround time from the design to the production phase and a considerable one-time masking cost. Design flaws are expensive; if a data or code fault is discovered, the mask ROM must **be replaced** to update the code or data. Some ICs simply include mask ROM. Other ICs include mask ROM in addition to many other components. Many microprocessors, in particular, have mask ROM to store their microcode. Some microcontrollers have mask ROM to store the bootloader or all of their firmware.

2.4.2 Physical Characteristics

Semiconductor memories, further, are categorized into their physical characteristics, such as:

1. Erasable or non-erasable, and
2. Volatile or non-volatile.

Erasable or non-erasable memories

Erasable memories are those in which the old data can be erased anytime, and new data can be written and stored. In contrast, if the digital information can not be erased, it is called *non-erasable memory*. For example, RAM is erasable memory, and ROM is non-erasable memory. Erase operation can be done electrically or with ultraviolet radiation.

Volatile or non-volatile memories

If the digitally stored information in memory is lost when the power supply goes off, this memory is known as *volatile memory*. In contrast, non-volatile memories are those in which the data remains intact until it is altered knowingly. For example, RAM is volatile memory, and all ROMs are non-volatile.

2.4.3 Mode of Access

The mode of access describes how data is read from or written to a memory location. Only reading is possible in ROMs. Mode of Access can be categorized as follows:

1. Sequential access, and
2. Random access

Sequentially accessed memories are also called sequential memories. However, RAM, and ROM, fall under the category of *random-access* memories. The access time for each memory location in

random-access memories is the same. However, in sequentially accessed memory, the access time varies depending on the location.

2.4.4 Fabrication Technology

Based on the fabrication technology, semiconductor memories can be divided into two types:

1. Bipolar, and
2. Unipolar.

Bipolar technology (TTL, ECL, etc.) or MOS technology is used in fabricating Static RAM, ROM, and PROM, whereas only unipolar technology (MOSFETs) is used in the fabrication of DRAM, EPROM, EEPROM, and flash memories. Figure 2 summarizes the classification of different semiconductors memory technologies studied above.

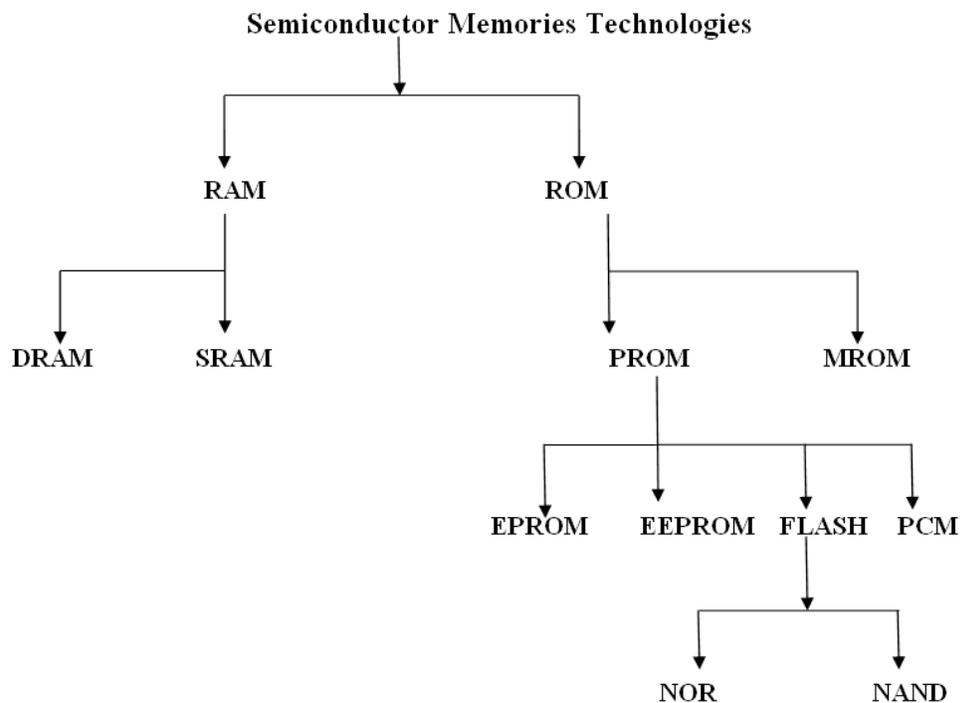


Fig.2.2: Classification of semiconductor memory technologies

2.5 MEMORY ORGANISATION AND EXPANSION

The semiconductor memory is a group of cells; each cell has its unique number known as *address*. To access the instruction, a memory request is made by the CPU. When the CPU has to read or write, a control signal is generated, such as 'read' or 'write.' The requirement for transferring instruction from memory to the CPU arises when the CPU executes the program.

Memory Request

Memory request has both an address and control signal. For example, when data is inserted into a stack, memory is consumed by each block. The memory chip capacity determines the number of memory cells.

Example1: Find the total number of cells in a 64k*8 memory chip.

Solution: Each cell size = 8

Number of bytes in 64k = $2^6 * 2^{10}$

So, the total number of cells = 2^{16}

Word Size

Word size tells the maximum number of bits a CPU can process at a time. Word size depends upon the type of processor. As shown in Figure 3, there can be different word sizes depending on the processor type. Word size is used in addresses, registers, floating-point numbers, and fixed-point numbers.

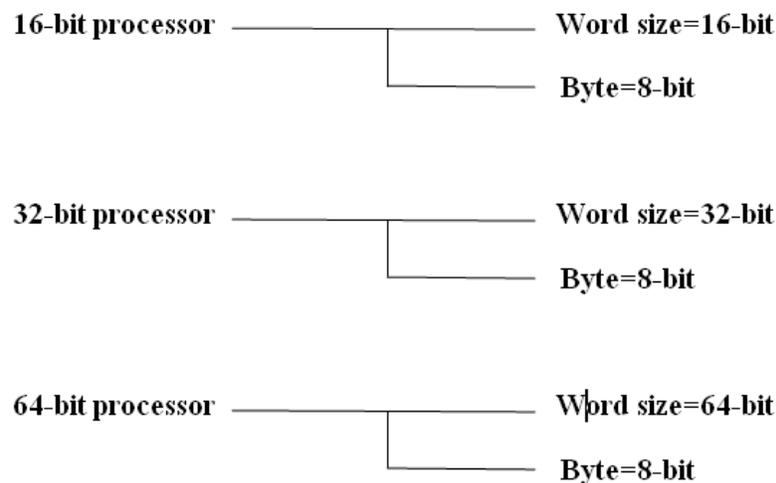


Fig.2.3: Different processors with their word and byte size

Figure 4 shows the block diagram of a memory device. The memory size is expressed with the help of two numbers, M and N, given by $M \times N$ bits. M expresses the number of locations, and the number of bits at each location is expressed by N. To put it another way, it suggests that the memory can accommodate M words, each of which can hold N bits. The most common number of words per chip are 64, 256, 512, 1024, 2048, 4096, etc., and the word sizes used are 1, 4, 8, 16, etc. Since each of the Memory's M locations is identified by a different address, P inputs are necessary to access any of the M locations, where $2^P = M$. These sets of lines are called the *address inputs* or *address buses*. In reality, the address input is applied to a $P \times M$ decoder circuit, which, depending on the address, activates one of its M outputs and therefore selects the desired

memory location. The address is defined in binary format. However, Octal and hexadecimal formats are frequently employed.

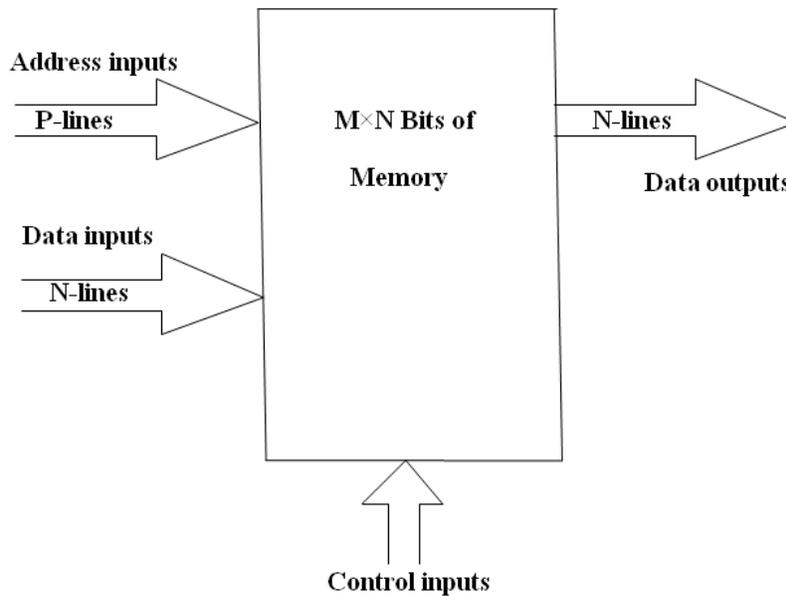


Fig.2.4: Block Diagram of a Memory Device

N inputs are necessary for any memory location to store or read data. *Data inputs and outputs* are two sets of N lines each, where inputs are needed to store data in the memory and outputs are needed to read data already stored there. A *bus* is a group of conductors carrying a common set of signals. As a result, the *input data bus* is the set of lines intended for data inputs, while the *output data bus* is intended for data outputs. Input and output data buses are *unidirectional*, meaning data can only flow in one direction. Most memory chips use the same lines for data input and output, referred to as a *bidirectional* bus. To achieve this, the time multiplexing of the data bus is done. As shown in Figure 5, depending on whether the control input is Read/Write, the data bus is used as an input bus for a specific time and an output bus for some other time.

For the device to receive instructions to carry out the specified action, several control inputs are required. For instance, a command signal is needed to inform the memory whether a read or write operation (R/\bar{W} in Figure 2.5) is intended. When the R/\bar{W} signal is **LOW**, the data bus acts as an input bus, allowing data to enter the memory, whereas when the R/\bar{W} signal is **HIGH**, the data bus acts as an output bus and reads the data from memory. Inputs for other commands include chip enable (CE), chip select (CS), etc. A minimum of two pins are needed for the power supply and ground in addition to the functional pins mentioned above. Figure 6 illustrates the internal layout of a 16x4 memory chip. Read and write operations are explained below.

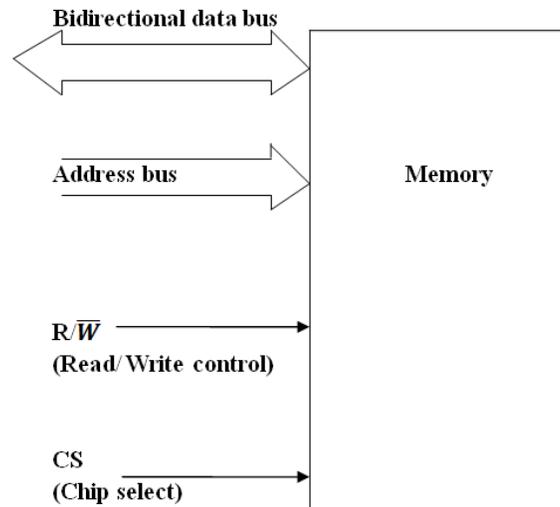


Fig.2.5: Block Diagram of Memory with Bidirectional Data Bus

Write Operation

It can be seen from Fig. 6 that to write a word into the designated memory location, a logic 1 voltage must be provided to the CS (chip select) and Write (WR) inputs, and a logic 0 voltage must be applied to the Read (RD) input. Because of this input combination, the outputs of AND gates A_1 and A_2 are 1 and 0, respectively. The value 1 at the output of A_1 activates the input buffers, causing the 4-bit word applied to the data inputs to be loaded (entered) into the designated (addressed) memory location. The outputs are set to Hi-impedance.

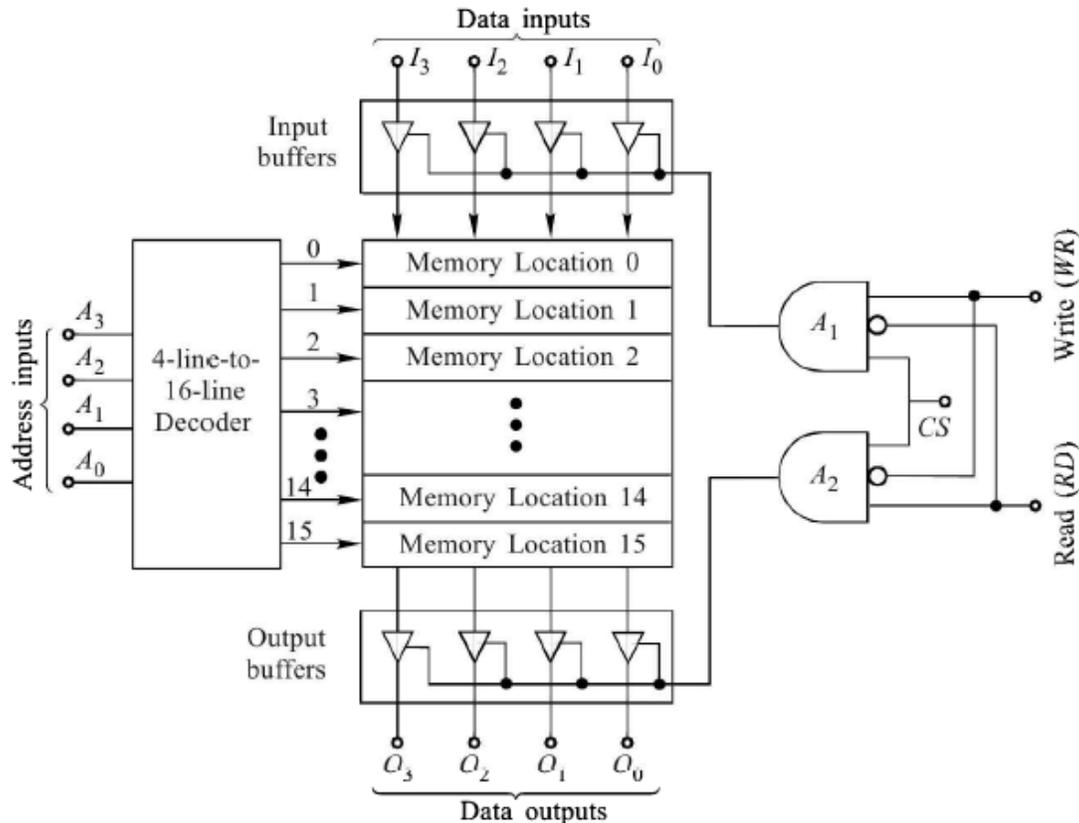


Fig.2.6: Internal Organization of a 16×4 Memory Chip [3]

Figure 7 depicts the numerous waveforms that occur during the writing procedure. The following are the essential properties of the write cycle:

- Write Cycle Time (t_{WC}): This is the minimum time required between successive write operations.
- Write Pulse Time (t_W): This is the minimum length of the write pulse.
- Write Release Time (t_{WR}): This is the minimum amount of time for which the address must be valid after the write pulse ends.
- Data Set Up Time (t_{DW}): This is the minimum amount of time for which the data must be valid before the write pulse ends.
- Data Hold Time (t_{DH}): This is the minimum amount of time for which the data must be valid after the write pulse ends.

Read Operation

Again from Fig. 6, it can be seen that to read the contents of a selected memory location, both Read (RD) and Chip Select (CS) inputs must be at logic 1, and Write (WR) must be at logic 0 levels. Because of this input combination, the outputs of AND gates A_1 and A_2 are 0 and 1, respectively. The value 1 at the output of A_2 activates the output buffers, thereby allowing the contents of the selected (addressed) memory location to appear at the data outputs. $RD=1$ tristate the input buffers so that data input does not affect the memory during a read operation.

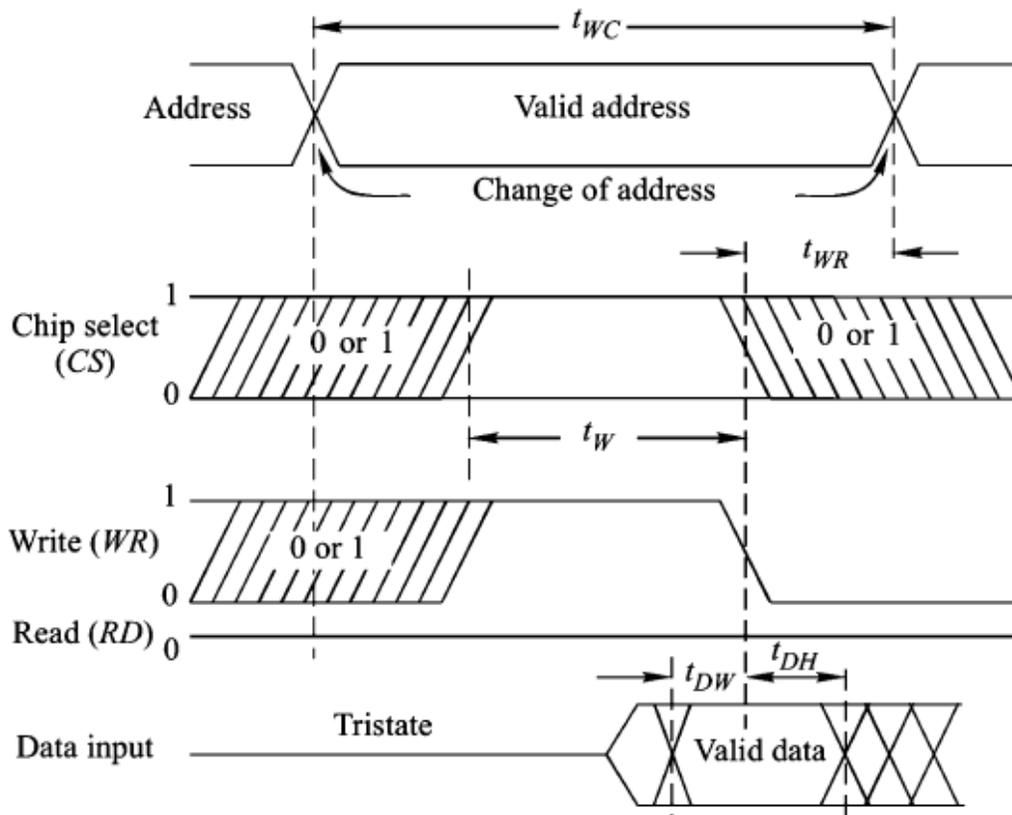


Fig.2.7: Write-Cycle Waveforms [3]

Fig. 8 shows several waveforms during the read operation. Some of the timing characteristics of the read cycle are:

- Read Cycle Time (t_{RC})- This is the minimum time required between successive read operations.
- Access Time (t_A)- Maximum time from the start of the valid address of read cycle to the time when the valid data is available at the data outputs.
- Read To Output Valid Time (t_{RD})- Maximum time delay between the start of read pulse and the availability of valid data at the data outputs.
- Read To Output Active Time (t_{RDX})- The minimum time delay between the beginning of read pulse and the output buffers coming to an active state.
- Chip-select To Output Valid Time (t_{CO})- Maximum time delay between the beginning of chip-select pulse and availability of valid data at the data outputs.
- Chip-select To Output Active Time (t_{CX})-Minimum time delay between the beginning of chip-select and output buffers coming to an active state.
- Output Tristate From Read (t_{OTD})- Maximum time delay between the end of read pulse and output buffers going to high impedance state.

- Data Hold Time (t_{OHA})- Minimum time for which the valid data is available at data outputs after the address ends.

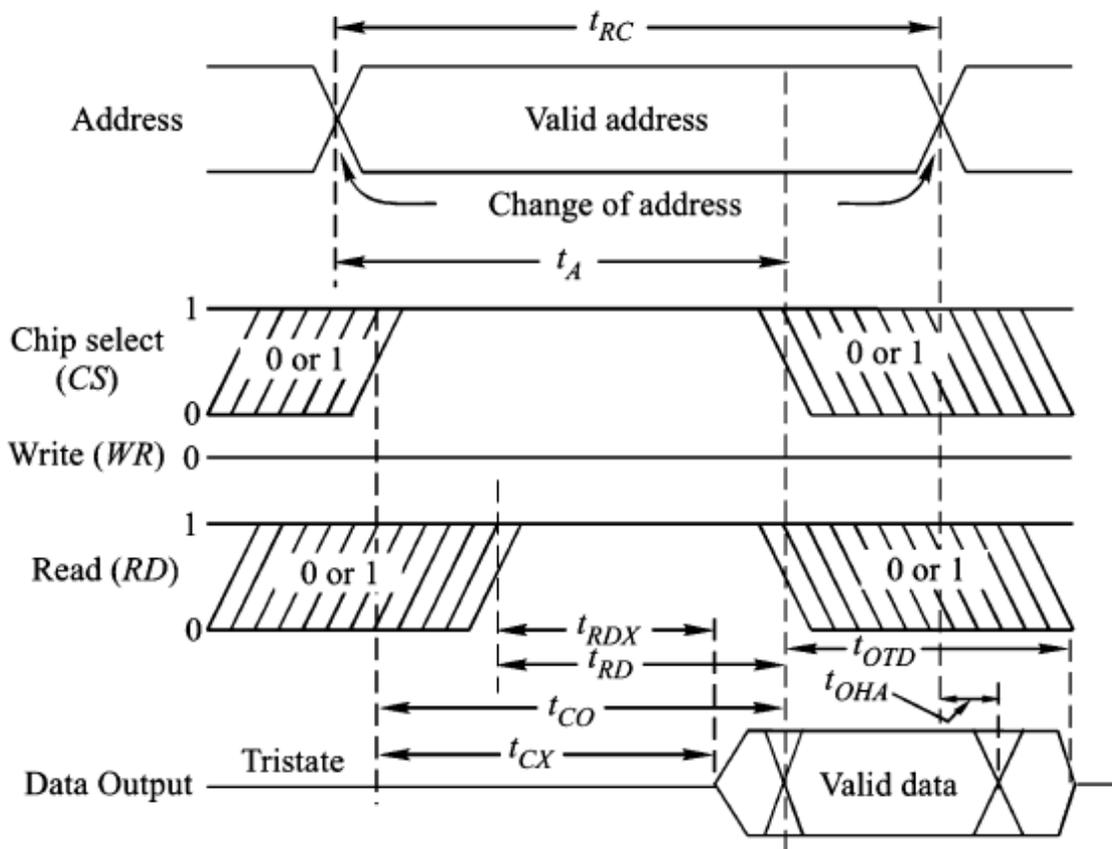


Fig.2.8: Read-Cycle Waveforms [3]

2.5.1 Expanding Memory size

A single memory IC chip cannot accommodate the number of words and/or word sizes needed for many memory applications. Therefore, several similar chips must be appropriately combined to produce this large number of words and/or word size.

2.5.1.1 Expanding Word Size

If the available memory has a word size of N , and we require memory to have a word size of n ($n > N$), then we will have to combine several similar memory ICs to obtain the desired word size. The number of IC chips needed must be an integer, i.e., the least integer function of n/N . These chips are to be connected in the following manner:

1. First, connect the corresponding address lines of each chip separately, i.e., connect the A_0 of each chip together, thereby making it the A_0 of overall Memory. Similarly, connect all other address lines together.

- Then connect the read terminal (RD) of all the ICs, making it the read terminal for the overall memory. Similarly, write (WR), and chip-select (CS) inputs connect.

The product of the number of chips and the word size of each chip is equal to the number of data input/output lines.

Example 2: Obtain a 16×8 memory using 16×4 Memory ICs.

Answer: Required word size $n = 8$

The word size of the available IC is $N=4$.

Therefore, the number of chips required to obtain the desired memory is $n/N = 2$.

Since each chip can store 16 4-bit words and it must store 16 8-bit words, each chip must store half of each word. The connections are depicted in Figure 9. Memory M_1 contains the higher order four bits $D_7, D_6, D_5,$ and D_4 of each 8-bit word, whereas memory M_0 contains the lower order four bits $D_3, D_2, D_1,$ and D_0 .

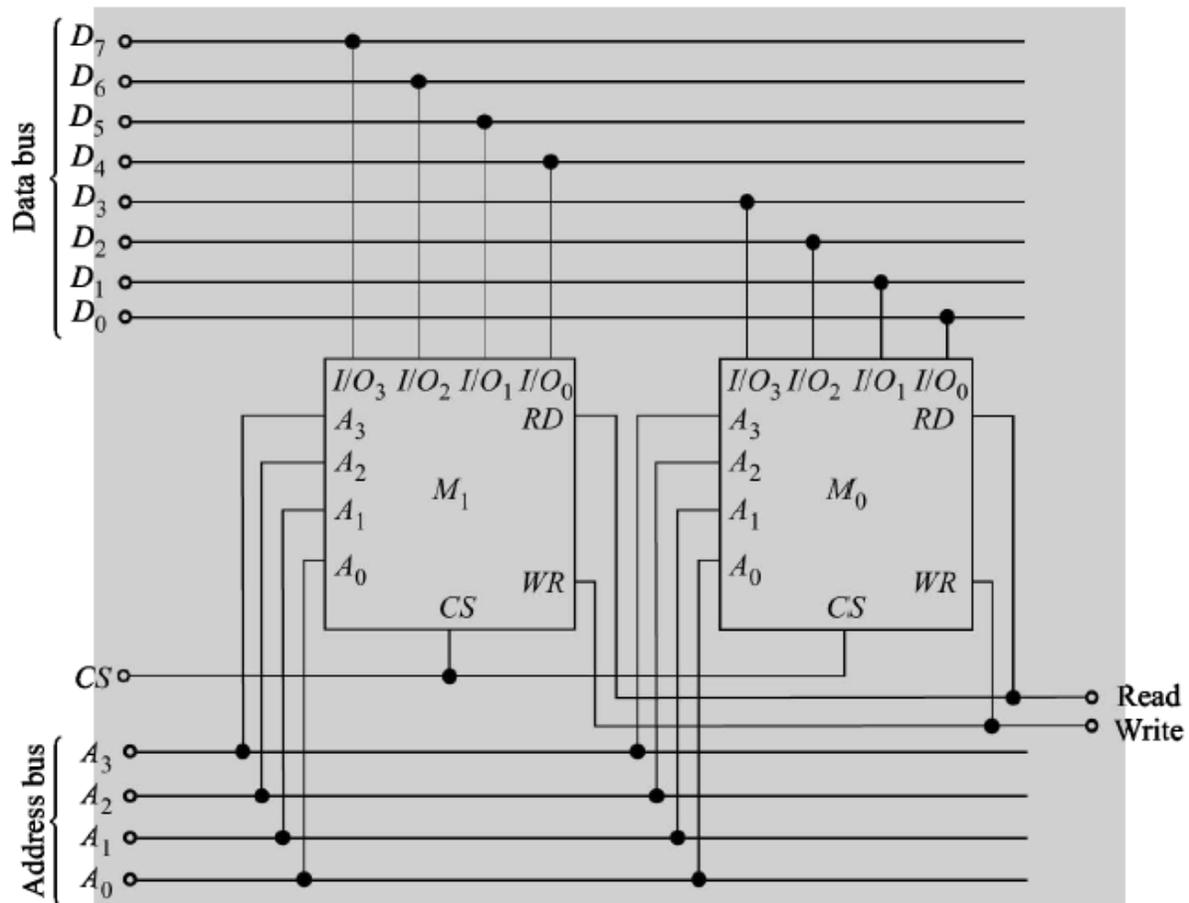


Fig. 2.9: 16×8 memory Obtained by Combining Two 16×4 Memory Chips.

2.5.2.2 Expanding Word Capacity

We can produce a memory with the desired number of locations by combining several memory chips. To obtain a memory of capacity m words, using the memory chips with M words each, the number of chips required is an integer value m/M , i.e., the least integer function of m/M . These chips should be connected in the following manner:

1. All the corresponding address line of each chip is connected separately.
2. RD terminal of each chip is connected. In the same way, the WR terminals of each chip are also connected.
3. A proper size decoder is used, and then its output is connected to one of the CS terminals of memory chips.

Example 3: Obtain a 2048×8 memory using 256×8 memory chips.

Answer: Number of chips needed = $2048/256=8$

Only one of the 2048 locations is to be accessed at any time, which will be in one of the eight chips, i.e., only one of the 8 chips should be selected at a time. The number of address lines to select any of the 2048 locations is 11 ($2^{11} = 2048$). The lower eight bits of address $A_7 - A_0$ shall be the same for each chip. The higher order three bits of address $A_{10} - A_8$ should select one out of eight chips. So, a 3-line-to-8-line decoder is needed. Figure 10 shows this memory connection. For read and write (R/\bar{W}), a common terminal is taken here. Logic 1 is for reading operation, and logic 0 is for writing operation. Chip select input is taken as active-low. Table 2 has the chip addresses.

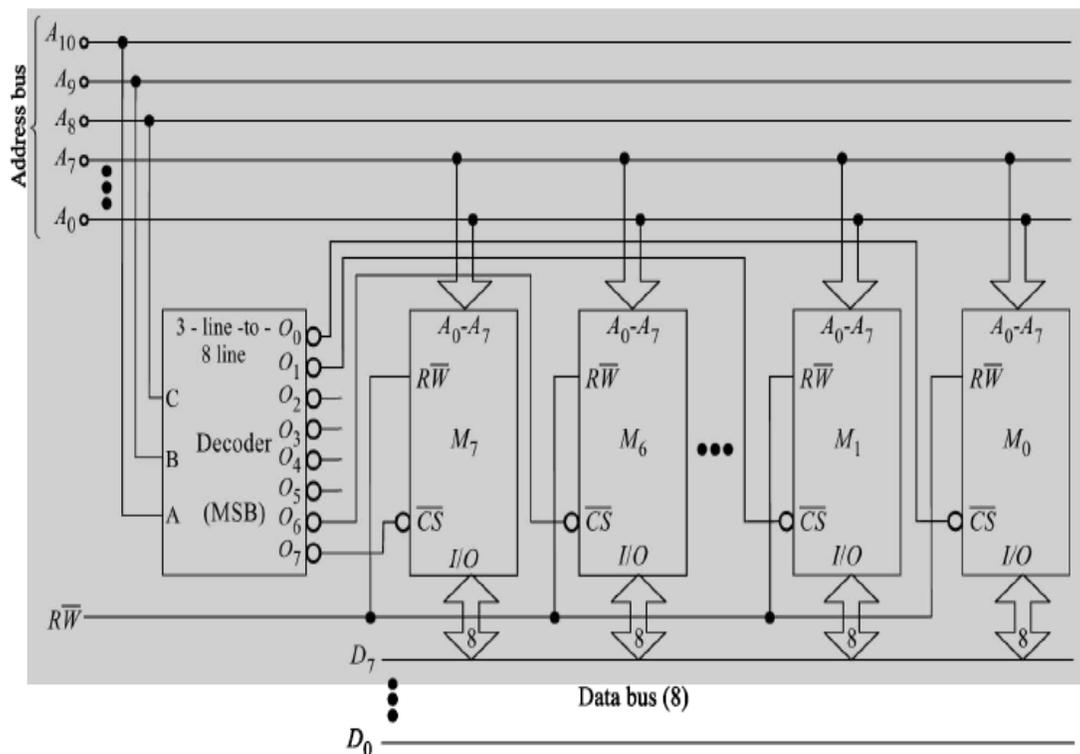


Fig. 2.10: A 2048×8 Memory Obtained by Combining Eight 256×8 Memory chips [3]

Memory Chip	Addresses (Hexadecimal)
M ₀	000-0FF
M ₁	100-1FF
M ₂	200-2FF
M ₃	300-3FF
M ₄	400-4FF
M ₅	500-5FF
M ₆	600-6FF
M ₇	700-7FF

Table2: Addresses of the memory Chips

Memory Hierarchy

All of the storage components used in a computer system—from the slow but high capacity auxiliary memory to the comparatively quicker main memory to the even smaller and faster cache memory accessible to the high-speed processor logic—are collectively referred to as the memory hierarchy system.

- Auxiliary Memory - It is at the bottom of the hierarchy and has an access time of 1000 times that of the main memory. Auxiliary Memory provides backup memory for storage, e.g., Magnetic disks and tapes.
- Main Memory – It has a central position in the hierarchy and is equipped with a CPU and auxiliary memory through an input/output processor (I/O).
- Cache memory – Program segments presently being run by the CPU are stored in the cache memory. The approximate access time ratio between cache and main memory is about 1 to 7 ~ 10. Whenever the CPU needs to access memory, it first checks the cache memory. If data is not found on cache memory, then the CPU moves to the main memory.

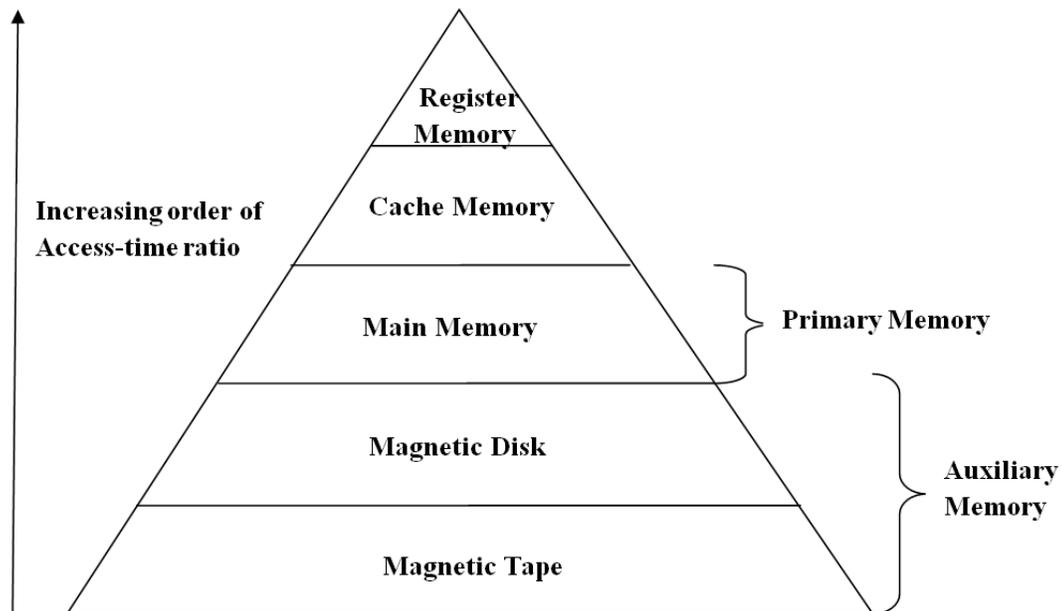


Fig.2.11: Memory Hierarchy

Hit Ratio

The *hit ratio* measures the performance of cache memory. When the CPU refers to memory and finds the word in the cache, it is said to produce a *hit*. If the word is not found in the cache, it is in the main memory. It is then called a *miss*. The ratio of the number of hits to the total CPU references to memory is called the *hit ratio*.

$$\text{Hit Ratio} = \frac{\text{Hit}}{(\text{Hit} + \text{Miss})}$$

Associate Memory

Associate Memory is also called content addressable Memory (CAM). Each bit position is comparable in this memory chip. Associate Memory has less storage capacity compared to other memory chips. Rather than allowing straightforward, direct access to the data based on the address, this particular memory is specialized for executing searches over data.

Associative memory is a type of ordinary semiconductor memory (often RAM) with additional comparison circuitry attached to it so that a search operation may be finished in a single clock cycle. It is a hardware search engine, a unique kind of computer memory utilized in high-searching applications.

2.6 STORAGE DEVICES

A storage device is essential computer hardware that stores information or data to process computational work. A computer could not operate or even start-up without a storage device, or we

may define a storage device as hardware that is used to store, transfer, or extract data files. Additionally, it has the capacity to permanently and temporarily store information and data. There are two categories of computer storage:

- **Primary Storage Devices:** It is sometimes referred to as main Memory and internal Memory. The program instructions, input data, and intermediate results are kept in this CPU section. Its size is often smaller. Examples of primary storage include RAM and ROM.
- **Secondary Storage Devices:** An externally located memory is called secondary storage. Programs and data are mainly stored there permanently and for a very long time. Secondary storage includes Hard Disk, CD, DVD, Pen/Flash drive, SSD, etc.

These primary and secondary storage devices have unique specifications and purposes. Several of the frequently used storage devices include:

1. **Primary Storage Devices:**

Common examples of primary storage devices are **RAM** (SRAM, synchronous SRAM, asynchronous SRAM, DRAM, Fast Page Mode DRAM (FPM DRAM), Extended Data Out DRAM (EDO DRAM), Burst EDO DRAM (BEDO DRAM), Synchronous DRAM (SDRAM)), **ROM** (Programmable ROM (PROM), Mask ROM (MROM), Erasable PROM (EPROM), Electrically Erasable PROM (EEPROM), Flash memory, PCM). We have already studied these memory devices in Section 2.4.1.

2. **Magnetic Storage Devices:**

Magnetic storage devices are very frequently used devices. In these devices, data storage is done by the magnetized medium. Various patterns of magnetization are used to store the data. Magnetic storage is divided into three types as follows:

- Disk Drive
- Diskette Drive
- Magnetic Tape

Disk Drive

Magnetic storage devices are mainly made up of disks. An example of a Disk drive is a hard disk drive (HDD). HDD has one or more disks and runs at a very high speed with a magnetizable coating medium. HDD has disks with a READ/WRITE head which reads and writes data from and to the disk.

Diskette Drives

Diskette drives are also known as floppy drives. Diskette drive comes under removable disk drives. A Floppy disk drive can be removed from the drive, known as *Floppy Disk Drive* or FDD, but a hard disk drive can not be removed from the drive. Floppy disks have a small storage capacity and are supposed to be utilized as portable storage, transferable from one machine to another. FDD can read and write data from and to the floppy disk. The Floppy disk has a plastic cover to remove dust.

Magnetic Tape

The reels of Magnetic tapes are coated with magnetizable elements for holding and server written on them. Tape drives have a huge storage capacity. It is used through servers, personal computers, etc. Tapes are used to achieve hundreds of terabytes (TB) of data. (1 terabyte = 1 trillion bytes = 2^{40} bytes).

3. Optical Storage Devices:

Optical storage uses recording data with the help of light. Optical storage devices can be used as removable disks. Optical storage is done with the help of drives. This type of storage has a system that is based on lasers that can be used to read or write to the disk. Examples of optical storage devices are compact disks (CD), Digital Versatile discs(DVD), and Blu-ray discs.

- i. Compact Disk: The CD stands for Compact Disc. On its surface, it has tracks and sectors for storing data. It has a round form and is composed of polycarbonate plastic. A CD may hold up to 700MB of data. It comes in two variations:
 - CD-R is an abbreviation for compact disc read-only. Once written, the data on this kind of CD cannot be removed. You can only read it.
 - Compact Disc read/write is abbreviated as CD-RW. This type of CD makes it simple to write to or delete data repeatedly.
- ii. Digital Versatile Disc: It is abbreviated as DVD. DVDs are flat, circular optical discs and can store data. It comes in two sizes, one being 4.7GB single-layer discs and the other being 8.5GB double-layer discs. Although DVDs resemble CDs, they have a larger storage capacity. There are two varieties:
 - Digital Versatile Disc read-only is known by the acronym DVD-R. The data on this kind of DVD cannot be removed after it has been written. It is purely read-only. Films and other media are typically written using it.
 - Digital Versatile Disc read/write is known by the acronym DVD-RW. Writing to and deleting data several times on this kind of DVD is simple.
- iii. Blu-ray Disc: Blu-ray discs are similar to CDs and DVDs but with a storage capacity of up to 25GB. A separate Blu-ray reader is required to play a Blu-ray disc. This Blu-ray technology reads a disc from a blue-violet laser, which allows for better density and more extended wavelength information storage.

4. Cloud and Virtual Storage:

Virtual or cloud storage systems have replaced secondary memory nowadays. The data is saved on the cloud for as long as we continue to pay for the cloud storage, and we may put our files and other items there. The major cloud service providers include Google, Amazon, Microsoft, and many others. We can afford the rent and yet receive all the advantages for the space we require. Despite the fact that it is physically being kept in a device in the service provider's data centers, the user is not involved in the device's upkeep.

5. Flash Memory Devices:

It is a more affordable, transportable storage option. Since it is more dependable and effective than other storage devices, it is the most widely utilized device for data storage. Some of the most popular flash memory gadgets include:

- **Pen Drive:** A USB flash drive with flash memory and an integrated USB interface is also referred to as a pen drive. We can read and write data into these devices considerably more quickly and effectively by connecting them directly to our PCs and laptops. These gadgets are quite portable. Typically, it runs from 1GB to 256GB.
- **SSD:** Solid State Drive, or SSD, is a mass storage device like HDDs. Since it does not have internal optical discs like hard drives, it is more durable. It is lighter than hard discs, consumes less power, and has read and write speeds 10 times faster than hard discs. However, these are also pricey. Although they perform the same purpose as hard discs, SSDs' internal parts are very different. Solid-state drives (SSDs) are hard drives without any moving parts, thus the name. SSDs use non-volatile storage to store data instead of magnetic discs. SSDs don't need to "spin up" since they don't have any moving components. It varies from 150GB to several TB and beyond.
- **SD Card:** A secure digital card is called an SD Card. It typically stores more data using electronic devices like phones, digital cameras, etc. It is portable, and the SD card is compact, making it simple to insert into electronic devices. It comes in various capacities, including 2GB, 4GB, 8GB, 16GB, 32GB, 64GB, 128GB, 256GB etc.
- **Multimedia Card:** Another name for it is MMC. It is an integrated circuit frequently found in digital cameras, automobile radios, and other devices. It is an external tool for data/information storage.

2.7 MEMORY ADDRESSING

A memory address is a unique identifier used by a device or CPU for data tracking. This binary address is defined by an ordered and finite sequence allowing the CPU to track the location of each memory byte. Memory addressing can also be understood as a specific assigned location in RAM to track the stored information. Nowadays, computers are addressed by bytes which are allocated to memory addresses. Before the CPUs do the processing, programs and data should be stored in unique memory address locations. More than one-byte data is sequentially segmented into multiple bytes with a range of corresponding addresses. Hardware devices and CPUs trace stored data by accessing memory addresses via data buses.

2.8 SEQUENTIAL PROGRAMMABLE LOGIC DEVICES

The most fundamental PLD devices are designed with programmable array logic (PAL) and programmable logic array (PLA). These devices are examples of logic devices known as sequential (simple) programmable logic devices (SPLDs). PLA device designs are built on the creation of two logic gate array structures. When combined, these arrays may be utilized to generate a *sum of products* that implement the proper Boolean logic equations. Input and output blocks, as well as a few customizable internal signal routing channels that can offer feedback on the output signal, are included in these devices.

In contrast to PLA devices, which allow programming of both the AND and OR planes, PAL devices have a fixed OR plane. In the two architectures, speed over logic flexibility is a trade-off. Nevertheless, both device topologies are relatively quick, with propagation delays (T_{pd}) in the nanosecond range.

The pin counts of PAL and PLA devices range from 16 to 28, while the range of logic cells is 8 to 24. These devices use EPROM and EEPROM technologies. The 22V10 is an example of PAL architecture. Figure 12 shows a typical SPLD connected in an IC. The sequential PLD is sometimes described as a simple PLD to differentiate it from the complex PLD. Flip-flops are a component of the SPLD IC chip in addition to the AND-OR array, as shown in Figure 13. A PAL or PLA is modified by incorporating a register made up of several flip-flops. The outputs of the circuit can come from the OR gates or the outputs of the flip-flops. The flip-flop outputs are included in the AND array's product terms via extra programmable connections. The flip-flops could be of the D or JK kind. The most common SPLD setup is the combinational PAL with D flip-flops. A PAL with flip-flops is referred to as a registered PAL since it has flip-flops in addition to the AND-OR array.

A microcell, shown in Figure 14, is a circuit that includes a *sum of products combinational logic functions* and an optional flip-flop in each segment of an SPLD. The combinational PAL and AND-OR arrays are identical. The output is driven by an edge-triggered D flip-flop connected to a common clock input, which changes state on a clock edge. As illustrated in Figure 14, a three-state buffer (or inverter) is connected to the flip-flop and is controlled by an output-enable OE. The flip-flop's output is routed back into one of the gates of programmable AND gates inputs to provide the current state condition for the sequential circuit. A typical SPLD has 8 to 10 macrocells within a single IC package. The common CLK input is shared by all three state buffers, as are the flip-flops.

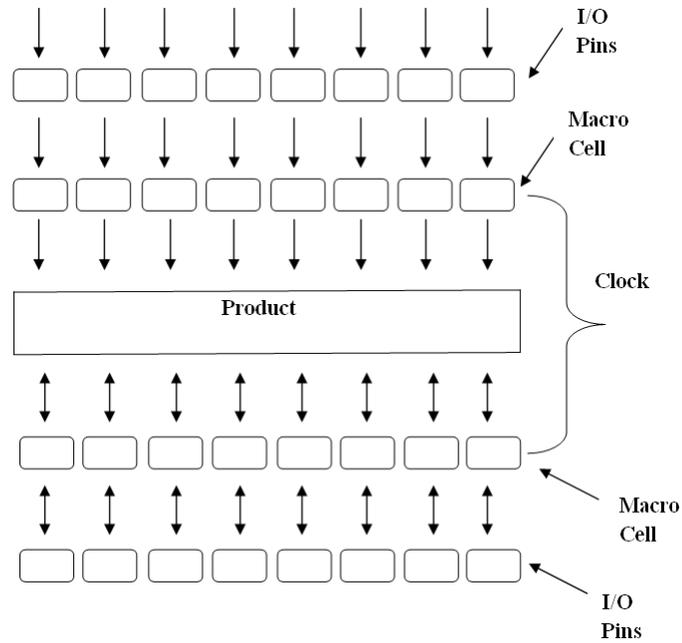


Fig.2.12: SPLDs

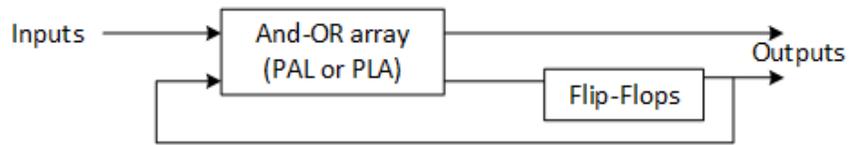


Fig.2.13: Sequential Programmable Logic Device

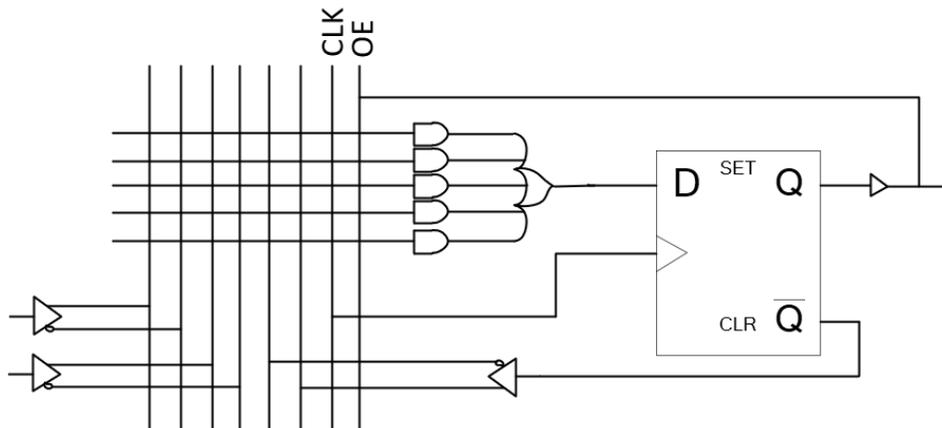


Fig.2.14: Basic Macrocell

Applications:

1. Machine control and automation,
2. Cybersecurity,
3. Medical Instrumentation,
4. Utility operation monitoring and control.

2.9 CONTENT ADDRESSABLE MEMORY

The Content Addressable Memory (CAM) comes under a special purpose random access memory device. CAM can be assessed by searching for data content. CAM is used in very high-speed searching applications. CAM is a storage device that stores memory in cells. It is sometimes known as associative memory or associative storage. CAM is being used widely in networking devices which speeds up forwarding information base and routing table operations. CAM is also used in cache memory. CAMs are made up with the help of MOS, CMOS, or bipolar technologies.

A CAM is quite different from the typical memory organization. The location address of memory organization is independent of memory content, whereas a CAM can search out or interrogate stored data based on its data contents. Therefore, CAM has become a powerful popular asset for many applications. For example, suppose a list containing several people, including their names, ages, professions, and nationality stored in a CAM. With the help of CAM, it is possible to find engineers from the list.

Operation of CAM

CAM performs read, write and associate operations, all three basic operations. The Block diagram of CAM is shown in Figure 14. The storage capacity of CAM is represented by $M \times N$ bits with M words of N bits each. It has N data input and N data output lines. Data input lines are I_0 to I_{N-1} . Data input is to be written into memory and for the keyword in case of associate operation. And data can be read out at D_0 through D_{N-1} .

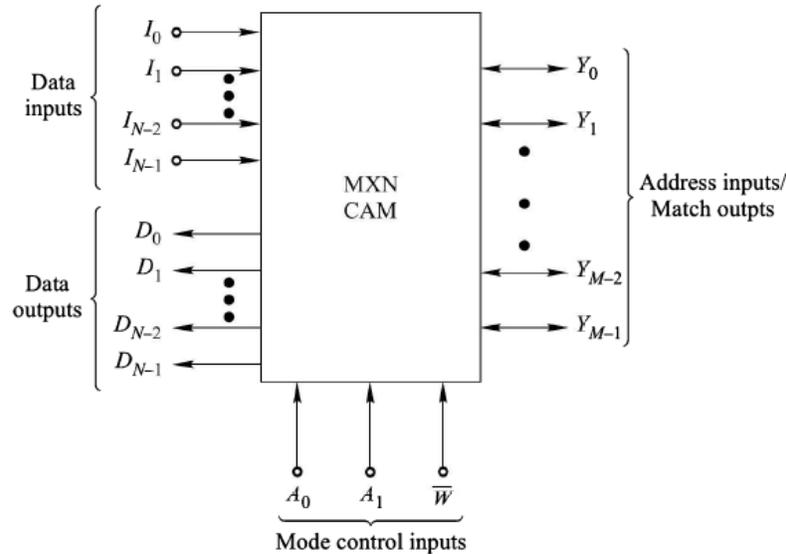


Fig.2.14: Block Diagram of CAM

Lines Y_0 to Y_{M-1} are output lines. Output Y lines are bidirectional. These lines can be utilized to select the storage function when there is a read or write operation. For each word, one address is there in CAM. (Y_0 is the address line for memory location 0, Y_1 is the address line for memory location 1, and so on). It is to be noted that in CAMs, linear selection addressing is utilized instead of coincident selection addressing. Output Y lines can be used to serve for match output, one for each memory location when there is a need to perform an association operation. (For example, a word stored in memory locations 5 and 8 has to match with the keyword, lines Y_5 and Y_8 will become HIGH to indicate the match condition). The mode control inputs select required operations. A CAM's read and write operations are the same as that of RAM. Input data will appear at the data outputs during the write operations. It is noted that the reading of data is not destructible.

Features of CAM:

- It is utilized in the database management system.
- It is more costly than RAM.
- It is suitable for parallel search.
- It returns the list of data word address that was located.

Advantages:

- Accurate,
- Very high speed,
- Input is associated with their memory contents.
- CAM can be cascaded whenever the lookup table size is to be increased.
- New entries can be added to the table.

Disadvantages:

- Costly,

- b) More power consumption,
- c) Frequently lookup requests,
- d) It Requires large footprints,

SUMMARY

Semiconductor memories are a necessary part of the digital system. Memory devices use memory cells to store digital data stored in binary form, i.e., logic 1 or logic 0. Various sizes are available for memory ICs. The memory size is specified in terms of the number of words and the number of bits in the word. Memory size can be increased by increasing the number of words or word size with the help of multiple chips. Broadly, memories can be categorized into RAM and ROM. RAM is a volatile memory, i.e., temporary memory, whereas ROM is a non-volatile memory, i.e., permanent memory. RAM, further, is divided into DRAM and SRAM. Dynamic RAM has less power consumption and more storage capacity than Static RAM. ROM can be divided into PROM and MROM. Again, PROM has two types EPROM and EEPROM. Flash memory is non-volatile memory, specifically EEPROM.

Glossary

Access time Time required for reading or writing a memory location.

Address The binary code of a memory location.

Address bus A parallel array of conductors is used for accessing a memory location.

Bidirectional bus A bus capable of transmitting data in both directions.

Cache memory A high-speed memory that stores the most recently used instructions or data from the slower main memory.

CAM (Content addressable memory) A special purpose RAM device that its contents can access.

Chip A piece of silicon or other semiconductor material on which an IC is fabricated.

Chip Enable An input control signal that, when activated, enables the chip.

Chip select An input control signal that allows the chip to be selected.

Cycle Time The minimum time between successive read or write cycles in memory.

Data bus A bus is used for carrying data.

DRAM Dynamic RAM

EEPROM Electrically Erasable Programmable Read Only Memory

EPROM Erasable Programmable Read Only Memory.

Erasable memory The memory which can be erased.

Flash Memory A specific type of EEPROM, non-volatile Memory, which allows in-circuit writing

Non-erasable memory The memory which can not be erased.

Non-volatile memory The memory is permanent even after the power supply goes off.

PROM Programmable Read Only Memory

RAMA Read-and-write semiconductor memory in which any memory location can be accessed for reading and writing at random.

ROM Read Only Memory A semiconductor memory in which data can be read.

Semiconductor Memory A memory fabricated using semiconductor material.

Static RAM A Random Access Memory in which read out and write into are not clocked.

Volatile memory A memory that loses its contents when power is turned off.

References

1. Stallings, William. *Computer organization and architecture: designing for performance*. Pearson Education India, 2003.
2. Mano, Morris M. "Computer systems architecture." (2006).
3. Jain, R. P. *Modern digital electronics*. Vol. 1, no. 10. Tata McGraw-Hill Education, New Delhi, 2003.

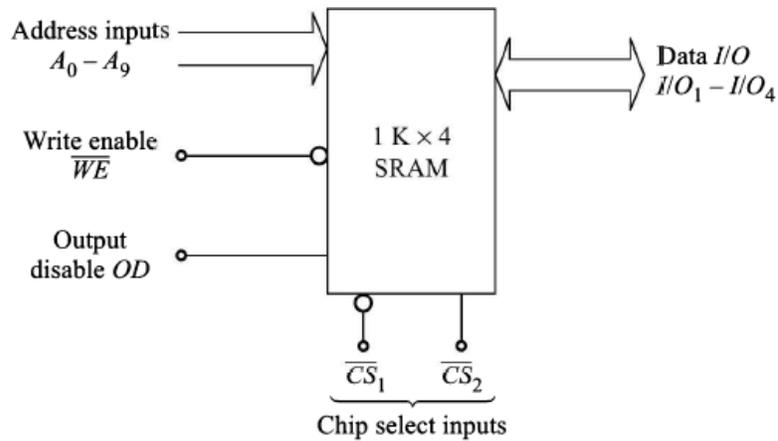
Questions

Short type questions

- 1) Information in a memory chip is stored in form.
- 2) The maximum number of bytes which can be stored in a memory of size 1024×8 is.....
- 3) The number of address lines required in memory of $128k \times 8$ is.....
- 4) An EPROM is aaccess memory.
- 5) An EPROM is erased by.....
- 6) CAM stands for.....
- 7) A dynamic RAM is fabricated using.....technology.
- 8) The number of IC chips of memory size 1024×4 required to have $16k \times 8$ memory will be.....
- 9) An SRAM with two ports is known as.....SRAM.
- 10) While specifying the memory size, the letter k stands for.....
- 11) How many address inputs are required to access 256 Bytes memory?
- 12) The storage capacity of a pendrive is 16GB. What is meant by 16GB?
- 13) Consider a memory size of 16 words. Find the binary address of each location.

Long type questions

- 1) For a memory with M words storage, find the number of pins required for addressing and the address range in binary format for each of the following cases:
 - a) $M=4$
 - b) $M=16$
 - c) $M=64$
 - d) $M=256$
- 2) Express the address range for each of the cases of question 1).
 - a) Hexadecimal format.
 - b) Octal format.
- 3) The Block diagram of a $1k \times 4$ bit static RAM is shown in the figure below. Find the number of RAM chips and other ICs required, if any, to obtain



Block Diagram of a 1k × 4 SRAM

- a) 4096 × 4bit RAM
 - b) 1024 × 8bit RAM
- 4) Explain the following:
- a) CAM
 - b) Flash Memory
 - c) Difference between SRAM and DRAM.
 - d) Memory Organisation.
- 5) The access time and cycle time for a set of memories are given in table. Determine the maximum rate which data can be accessed in each case.

Memory	Access time (ns)	Cycle time (ns)
A	1500	1500
B	300	580
C	450	450
D	200	200
E	60	60
F	800	800

- 6) Addressing of a 32K × 16 memory is realized using a single decoder. What is the minimum number of AND gates required for the decoder?
- 7) For the memory timing of the table given below, find the maximum rate (words/sec) at which
- a) Data can be stored, and
 - b) Data can be read.

Parameter	Time (ns)
t_{WC}	200

t_W	120
t_{WR}	0
t_{DW}	120
t_{DH}	0
t_{RC}	200
t_A	200
t_{RD}	70
t_{RDX}	20
t_{CO}	70
t_{CX}	20
t_{OTD}	60
t_{OHA}	50

Multiple type questions

- 1) A dynamic RAM consists of
 - a) 6 transistors
 - b) 2 transistors and 2 capacitors
 - c) 1 transistor and 1 capacitor
 - d) 2 capacitors only
- 2) The minimum number of MOS transistors required to make a dynamic RAM cell is
 - a) 1
 - b) 2
 - c) 3
 - d) 4
- 3) Each cell of a static RAM contains
 - a) 6 MOS transistors
 - b) MOS transistors and 2 capacitors
 - c) 2 MOS transistors and 4 capacitors
 - d) 1 MOS transistor and 1 capacitor
- 4) In a DRAM
 - a) Periodic refreshing is not required
 - b) Information is stored in a capacitor
 - c) Information is stored in a latch
 - d) Both read and write operations can be performed simultaneously
- 5) The access time of a bipolar RAM is of the order of
 - a) 20nsec
 - b) 20μsec
 - c) 20msec
 - d) 20sec
- 6) The access time of MOS RAM is of the order of

- a) 1nsec
 - b) 1μsec
 - c) 1msec
 - d) 1sec
- 7) RAM is also known as
- a) RWM
 - b) PROM
 - c) EAROM
 - d) EPROM
- 8) The density of DRAM is
- a) More than that of SRAM
 - b) Equal to that of SRAM
 - c) less than that of SRAM
 - d) Zero
- 9) PROM is available in
- a) Bipolar version only
 - b) MOS version only
 - c) Both bipolar and MOS version
 - d) None of the above
- 10) Which of the following memories can be programmed once by the user and then can not be erased and programmed?
- a) ROM
 - b) PROM
 - c) EPROM
 - d) EEPROM

Answers

Short type

- 1) Memory cell
- 2) 10
- 3) 17
- 4) Random access memory
- 5) UV light
- 6) Content Addressable Memory
- 7) MOS Technology
- 8) 32
- 9) Two port SRAM
- 10) Kilo
- 11) memory size = 256 bytes, Thus, $2^k = 256$ will give, $K = 8$

12) 16GB = 1024MB

13) Since $M=16$, therefore, $2^P=M$ gives $P=4$, i.e., for selecting one out of 16 words, a 4-bit address is required. The address is specified as $A_3 A_2 A_1 A_0$, where A_3 represents the MSB and A_0 represents LSB. The table below shows the address.

Word number	Binary address address			
	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table: Memory Addresses

Long Type

- 1) The number of pins P is given by $2^P = M$
 - a) $P=2$
Address range: $A_1A_0=00$ to 11
 - b) $P=4$
Address range: $A_3A_2A_1A_0=0000$ to 1111
 - c) $P=6$
Address range: $=A_5A_4A_3A_2 A_1A_0=000000$ to 111111
 - d) $P=8$
Address range: $=A_7A_6A_5A_4A_3A_2 A_1A_0=00000000$ to 11111111
- 2) a) 0 to 3; 0 to F; 00 to 3F; 00 to FF

- b) 0 to 3; 0 to 17; 00 to 77; 000 to 377
- 3) a) 4 chips of 2142 and one 1-out of 4 decoder IC will be required, b) 2 chips of 2142.
- 4) Read this unit to find answers
- 5) The maximum access rate = 1/ cycle time;
 Gives the maximum rate for each memory. It is given for each Memory.

Memory	Maximum rate
A	$\frac{1 \times 10^9}{1500} = 666666/s$
B	$\frac{1 \times 10^9}{580} = 1724137/s$
C	$\frac{1 \times 10^9}{450} = 2222222/s$
D	$\frac{1 \times 10^9}{200} = 5000000/s$
E	$\frac{1 \times 10^9}{60} = 16666666/s$
F	$\frac{1 \times 10^9}{800} = 1250000/s$

- 6) Memory size = $2^K \times m$ where K= address line, m= data line. Since $32K \times 16$ memory = $2^{15} \times 16$, therefore K = 15. Therefore, the number of AND gates required = 2^{15}
- 7) The maximum rate at which data can be stored is:

$$\frac{1}{t_{WC}} = \frac{1}{200 \times 10^{-9}} = 5 \times 10^6 \text{ words/s}$$

The maximum rate at which data can be read is:

$$\frac{1}{t_{RC}} = \frac{1}{200 \times 10^{-9}} = 5 \times 10^6 \text{ words/s}$$

Multiple types

- 1) C
- 2) A
- 3) A
- 4) B
- 5) A
- 6) B
- 7) A

- 8) A
- 9) C
- 10) B

UNIT 3 MICROPROCESSOR ARCHITECTURE AND MICROCOMPUTER SYSTEM

Structure

3.1. Introduction

3.2. Objective

3.3. Microprocessor

3.3.1. Basic Terms Used In Microprocessor

3.3.2. Features Of Microprocessor

3.4. Evolution of Micro Processors

3.5. Micro Processor Organization

3.5.1. Microprocessor Architecture and its Operations

3.5.2. Microprocessor-Initiated Operations and 8085 Bus Organization

3.5.3. The 8085 Bus Structure

3.5.4 Internal Data Operations and The 8085 Registers

3.5.5. The 8085 Programmable Registers

3.5.6. Peripheral or Externally Initiated Operations

3.6. Microprocessor Instruction Set and Computer Languages

3.7. Machine Language

3.7.1. Writing And Executing An Assembly Language Program

3.8. High-Level Languages

3.9. Operating System

3.9.1. Types of Operating Systems

3.10 Summary

3.11 Glossary

3.12 References

3.13 Suggested Readings

3.14 Terminal Questions

3.14.1 Short Answer Type

3.1. INTRODUCTION

Computer's Central Processing Unit (CPU) built on a single Integrated Circuit (IC) is called a microprocessor. A digital computer with one microprocessor which acts as a CPU is called microcomputer.

A microprocessor is a computer processor where the data processing logic and control is included on a single integrated circuit, or a small number of integrated circuits. The microprocessor contains the arithmetic, logic, and control circuitry required to perform the functions of a computer's central processing unit.

A microcomputer system consists of three components-the microprocessor, memory, and I/O (input/output)-as discussed in the previous chapter. The microprocessor manipulates data, controls the timing of various operations, and communicates with such peripherals (devices) as memory and I/O.

The data bus is a group of eight lines used for data flow .These lines are bidirectional-data flow in both directions between the MPU and memory and peripheral devices.

The data bus is a group of eight lines used for data flow .These lines are bidirectional-data flow in both directions between the MPU and memory and peripheral devices. The control bus is comprised of various single lines that carry synchronization signals.

The microprocessor uses this register to sequence the execution of instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched.

The number of bits in a word for a given machine is fixed, and words are formed through various combinations of these bits. For example a machine with a word length of eight bit scan has 756 a combination of eight bits-thus a language of 256 words.

Programming languages that are intended to be machine-independent are called high-level languages. The list includes such languages as C, FORTRAN, BASIC, PASCAL, and COBOL.

Thus, assembly language programs are compact and require less memory space; they are more efficient than the high-level language programs.

An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it.

3.2. OBJECTIVE

After studying this unit, you will learn about-

- Microprocessor
- Evolution of Micro Processors
- Micro Processor Organization
- 8085 Bus Structure
- 8085 Programmable Registers
- Machine Language
- High-Level Languages

3.3. MICROPROCESSOR

Computer's Central Processing Unit (CPU) built on a single Integrated Circuit (IC) is called a microprocessor. A digital computer with one microprocessor which acts as a CPU is called microcomputer. It is a programmable, clock driven, multipurpose, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as output. The microprocessor contains millions of tiny components like transistors, registers, and diodes that work together.

A microprocessor is a computer processor where the data processing logic and control is included on a single integrated circuit, or a small number of integrated circuits. The microprocessor contains the arithmetic, logic, and control circuitry required to perform the functions of a computer's central processing unit. The integrated circuit is capable of interpreting and executing program instructions and performing arithmetic operations. The microprocessor is a multipurpose, clock-driven, register-based, digital integrated circuit that accepts binary data as input, processes it according to instructions stored in its memory, and provides results (also in binary form) as output. Microprocessors contain both combinational logic and sequential digital logic, and operate on numbers and symbols represented in the binary number system.

The integration of a whole CPU onto a single or a few integrated circuits using Very Large Scale Integration (VLSI) greatly reduced the cost of processing power. Integrated circuit processors are produced in large numbers by highly automated metal oxide semiconductor (MOS) fabrication

processes, resulting in a relatively low unit price. Single chip processors increase reliability because there are much fewer electrical connections that could fail. As microprocessor designs improve, the cost of manufacturing a chip (with smaller components built on a semiconductor chip the same size) generally stays the same according to Rock's law.

Before microprocessors, small computers had been built using racks of circuit boards with many medium and small scale integrated circuits, typically of TTL type. Microprocessors combined this into one or a few large-scale ICs. The first commercially available microprocessor was the Intel 4004 introduced in 1971.

Continued increases in microprocessor capacity have since rendered other forms of computers almost completely obsolete, with one or more microprocessors used in everything from the smallest embedded systems and handheld devices to the largest mainframes and supercomputers.

A microcomputer system consists of three components-the microprocessor, memory, and I/O (input/output). The microprocessor manipulates data, controls the timing of various operations, and communicates with such peripherals (devices) as memory and I/O. The internal logic design of the microprocessor, called its **architecture**, determines how and when various operations are performed by the microprocessor. The system bus provides paths for the flow of binary information (data and instructions).

3.3.1. BASIC TERMS USED IN MICROPROCESSOR

Instruction Set - The group of commands that the microprocessor can understand is called Instruction set. It is an interface between hardware and software.

Bus - Set of conductors intended to transmit data, address or control information to different elements in a microprocessor. A microprocessor will have three types of buses, i.e., data bus, address bus, and control bus.

IPC (Instructions per Cycle) - It is a measure of how many instructions a CPU is capable of executing in a single clock.

Clock Speed - It is the number of operations per second the processor can perform. It can be expressed in megahertz (MHz) or gigahertz (GHz). It is also called the Clock Rate.

Bandwidth - The number of bits processed in a single instruction is called Bandwidth.

Word Length - The number of bits the processor can process at a time is called the word length of the processor. 8-bit Microprocessor may process 8-bit data at a time. The range of word length is from 4 bits to 64 bits depending upon the type of the microcomputer.

Data Types - The microprocessor supports multiple data type formats like binary, ASCII, signed and unsigned numbers.

3.3.2. FEATURES OF MICROPROCESSOR

Low Cost - Due to integrated circuit technology microprocessors are available at very low cost. It will reduce the cost of a computer system.

High Speed - Due to the technology involved in it, the microprocessor can work at very high speed. It can execute millions of instructions per second.

Small Size - A microprocessor is fabricated in a very less footprint due to very large scale and ultra large scale integration technology. Because of this, the size of the computer system is reduced.

Versatile - The same chip can be used for several applications, therefore, microprocessors are versatile.

Low Power Consumption - Microprocessors are using metal oxide semiconductor technology, which consumes less power.

Less Heat Generation - Microprocessors uses semiconductor technology which will not emit much heat as compared to vacuum tube devices.

Reliable - Since microprocessors use semiconductor technology, therefore, the failure rate is very less. Hence it is very reliable.

Portable - Due to the small size and low power consumption microprocessors are portable.

3.4. EVOLUTION OF MICRO PROCESSORS

The advent of low cost computers on integrated circuits has transformed modern society. General-purpose microprocessors in personal computers are used for computation, text editing, multimedia display, and communication over the Internet. Many more microprocessors are part of embedded systems, providing digital control over myriad objects from appliances to automobiles to cellular phones and industrial process control. Microprocessors perform binary operations based on Boolean logic, named after George Boole. The ability to operate computer systems using Boolean Logic was first proven in a 1938 thesis by master's student Claude Shannon, who later went on to become a professor. Shannon is considered "The Father of Information Theory".

Following the development of MOS integrated circuit chips in the early 1960s, MOS chips reached higher transistor density and lower manufacturing costs than bipolar integrated circuits by 1964. MOS chips further increased in complexity at a rate predicted by Moore's law, leading to large-scale integration (LSI) with hundreds of transistors on a single MOS chip by the late 1960s. The application of MOS LSI chips to computing was the basis for the first microprocessors, as engineers began recognizing that a complete computer processor could be contained on several MOS LSI chips. Designers in the late 1960s were striving to integrate the central processing unit (CPU) functions of a computer onto a handful of MOS LSI chips, called microprocessor unit (MPU) chipsets.

The first commercially produced microprocessor was the Intel 4004, released as a single MOS LSI chip in 1971. The single-chip microprocessor was made possible with the development of MOS silicon-gate technology (SGT). The earliest MOS transistors had aluminium metal gates, which Italian physicist Federico Faggin replaced with silicon self-aligned gates to develop the first silicon-gate MOS chip at Fairchild Semiconductor in 1968. Faggin later joined Intel and used his silicon-gate MOS technology to develop the 4004, along with Marcian Hoff, Stanley Mazor and Masatoshi Shima in 1971. The 4004 was designed for Busicom, which had earlier proposed a multi-chip design in 1969, before Faggin's team at Intel changed it into a new single-chip design. Intel introduced the first commercial microprocessor, the 4-bit Intel 4004, in 1971. It was soon followed by the 8-bit microprocessor Intel 8008 in 1972.

Other embedded uses of 4-bit and 8-bit microprocessors, such as terminals, printers, various kinds of automation etc., followed soon after. Affordable 8-bit microprocessors with 16-bit addressing also led to the first general-purpose microcomputers from the mid-1970s on.

The first use of the term "microprocessor" is attributed to Viatron Computer Systems describing the custom integrated circuit used in their System 21 small computer system announced in 1968.

Since the early 1970s, the increase in capacity of microprocessors has followed Moore's law; this originally suggested that the number of components that can be fitted onto a chip doubles every year. With present technology, it is actually every two years, and as a result Moore later changed the period to two years.

TABLE 3.1:

Microprocessor	Year of Invention	Word length	Memory addressing Capacity	Pins	Clock	Remark
4004	1971	4-bit	1 KB	16	750 KHz	First Microprocessor
8085	1976	8-bit	64 KB	40	3-6 MHz	Popular 8-bit Microprocessor
8086	1978	16-bit	1 MB	40	5-8 MHz	Widely used in PC/XT
80286	1982	16-bit	16MB real, 4 GB virtual	68	6-12.5 MHz	Widely used in PC/AT
80386	1985	32-bit	4GB real, 64TB virtual	132 14X14 PGA	20-33 MHz	Contains MMU on chip
80486	1989	32-bit	4GB real, 64TB virtual	168 17X17 PGA	25-100 MHz	Contains MMU, cache and FPU, 1.2

						million transistors
Pentium	1993	32-bit	4GB real,32-bit address,64-bit data bus	237 PGA	60-200 MHz	Contains 2 ALUs,2 Caches, FPU, 3.3 Million transistors, 3.3 V, 7.5 million transistors
Pentium Pro	1995	32-bit	64GB real, 36-bit address bus	387 PGA	150-200 MHz	It is a data flow processor. It contains second level cache also,3.3 V
Pentium II	1997	32-bit			233-400 MHz	All features Pentium pro plus MMX technology,3.3 V, 7.5 million transistors
Pentium III	1999	32-bit	64GB	370 PGA	600-1.3 MHz	Improved version of Pentium II; 70 new SIMD instructions
Pentium 4	2000	32-bit	64GB	423 PGA	600-1.3 GHz	Improved version of Pentium III

Itanium	2001	64-bit	64 address lines	423 PGA	733 MHz – 1.3 GHz	64-bit EPIC Processor
---------	------	--------	---------------------	------------	-------------------------	-----------------------------

3.5. MICRO PROCESSOR ORGANIZATION

3.5.1. MICROPROCESSOR ARCHITECTURE AND ITS OPERATIONS

The microprocessor is a programmable digital device, designed with registers, flip-flops, and timing elements. The microprocessor has set instructions, designed internally, to manipulate data and communicate with peripherals. This process of data manipulation and communication is determined by the logic design of the microprocessor, called the architecture.

The microprocessor can be programmed to perform functions on given data by selecting necessary instructions from its set. These instructions are given to the microprocessor by writing them into its memory. Writing (or entering) instructions and data is done through an input device such as a keyboard. The microprocessor reads or transfers one instruction at a time, matches it with its instruction set, and performs the data manipulation indicated by the instruction. The result can be stored in memory or sent to such output devices as LEDs or a CRT terminal. In addition, the microprocessor can respond to external signals. It can be interrupted, reset, or asked to wait to synchronize with slower peripherals. All the various functions performed by the microprocessor can be classified in three general categories:

- ✓ Microprocessor-initiated operations
- ✓ Internal operations
- ✓ Peripheral (or externally initiated) operations

To perform these functions, the microprocessor requires a group of logic circuits and a set of signals called control signals. However, early processors did not have the necessary circuitry on one chip; complete units were made up of more than one chip. Therefore, the term microprocessing unit (MPU) is defined here as a group of devices that can perform these functions with the

necessary set of control signals. This term is similar to the term central processing unit (CPU). However, later microprocessors include most of the necessary circuitry to perform these operations on a single chip. Therefore, the terms MPU and microprocessor often are used synonymously.

The microprocessor functions listed above are explained here in relation to the 8085 MPU but without the details of the MPUs. However, the general concepts discussed here are applicable to any microprocessor. The devices necessary to make up the 8085 MPUs will be discussed in the next chapter.

3.5.2. MICROPROCESSOR INITIATED OPERATIONS AND 8085 BUS ORGANIZATION

The MPU performs primarily four operations:

1. Memory Read: Reads data (or instructions) from memory.
2. Memory Write: Writes data (or instructions) into memory.
3. I/O Read: Accepts data from input devices.
4. I/O Write: Sends data to output devices.

All these operations are part of the communication process between the MPU and peripheral devices (including memory). To communicate with a peripheral (or a memory location), the MPU needs to perform the following steps:

Step 1: Identify the peripheral or the memory location (with its address).

Step 2: Transfer binary information (data and instructions).

Step 3: Provide timing or synchronization signals.

The 8085 MPU performs these functions using three sets of communication lines called buses: the address bus, the data bus, and the control bus.

ADDRESS BUS

The address bus is a group of 16 lines generally identified as A_0 to A_{15} . The address bus is unidirectional **bits** flow in one direction—from the MPU to peripheral devices. The MPU uses the address bus to perform the first function: identifying a peripheral or a memory location.

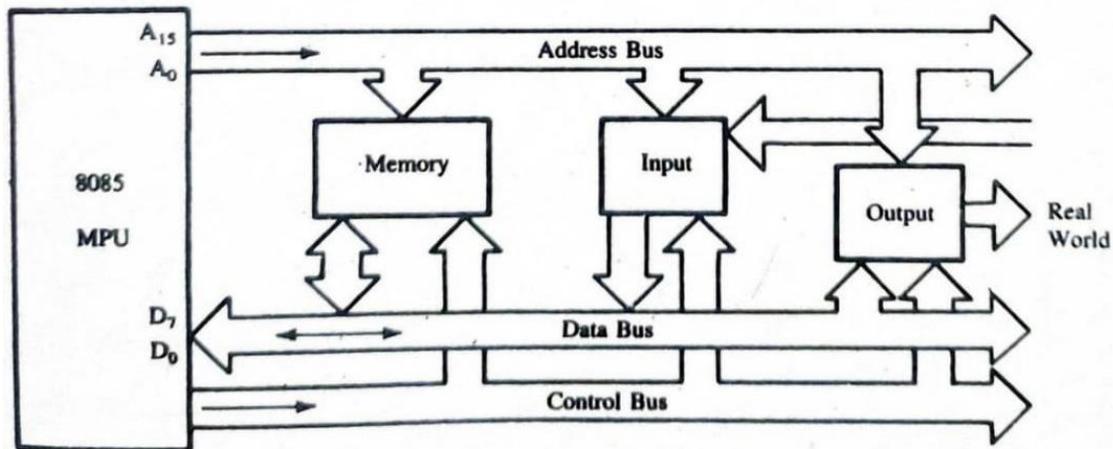


Fig. 3.1: Address Bus

3.5.3. THE 8085 BUS STRUCTURE

In a computer system, each peripheral or memory location is identified by a binary number, called an address, and the address bus is used to carry a 16-bit address. This is similar to the postal address of a house. A house can be identified by various number schemes. For example, the forty-fifth house in a lane can be identified by the two-digit number 45 or by the four-digit number 0045. The two-digit numbering scheme can identify only a hundred houses, from 00 to 99. On the other hand, the four-digit scheme can identify ten thousand houses, from 0000 to 9999. Similarly, the number of address lines of the MPU determines its capacity to identify different memory locations (or peripherals). The 8085 MPU with its 16 address lines is capable of addressing $2^{16} = 65,536$ (generally known as 64K) memory locations. As we know that, 1K memory is determined by

rounding off 1024 to the nearest thousand; similarly, 65,536 is rounded off to 64,000 as a multiple of 1K.

Most 8-bit microprocessors have 16 address lines. This may explain why microcomputer systems based on 8-bit microprocessors have 64K memory. However, not every microcomputer system has 64K memory. In fact, most single-board microcomputers have less than 4K of memory, even if MPU is capable of addressing 64K memory. The number of address lines is arbitrary; it is determined by the designer of a microprocessor based on such considerations as availability of pins and intended applications of the processor. For example, the Intel 8088 processor has 20 and the Pentium processor has 32 address lines.

DATA BUS

The data bus is a group of eight lines used for data flow. These lines are bidirectional-data flow in both directions between the MPU and memory and peripheral devices. The MPU uses the data bus to perform the second function: transferring binary information.

The eight data lines enable the MPU to manipulate 8-bit data ranging from 00 to FF ($2^8 = 256$ numbers). The largest number that can appear on the data bus is 11111111 (255_{10}). The 8085 is known as an 8-bit microprocessor. Microprocessors such as the Intel 8086, Zilog Z8000, and Motorola 68000 have 16 data lines; thus they are known as 16-bit microprocessors. The Intel 80386/486 have 32 data lines; thus they are classified as 32-bit microprocessors.

CONTROL BUS

The control bus is comprised of various single lines that carry synchronization signals. The MPU uses such lines to perform the third function: providing timing signals.

The term **bus**, in relation to the control signals, is somewhat confusing. These are not groups of lines like address or data buses, but individual lines that provide a pulse to indicate an MPU operation. The MPU generates specific control signals for every operation (such as Memory Read or I/O Write) it performs. These signals are used to identify a device type with which the MPU intends to communicate.

To communicate with a memory—for example, to read an instruction from a memory location—the MPU places the 16-bit address on the address bus. The address on the bus is decoded by an external logic circuit, which will be explained later, and the memory location is identified. The MPU sends a pulse called Memory Read as the control signal. The pulse activates the memory chip, and the contents of the memory location (8-bit data) are placed on the data bus and brought inside the microprocessor.

3.5.4 INTERNAL DATA OPERATIONS AND THE 8085 REGISTERS

The internal architecture of the 8085 microprocessor determines how and what operations can be performed with the data. These operations are:

1. Store 8-bit data.
2. Perform arithmetic and logical operations.
3. Test for conditions.
4. Sequence the execution of instructions.
5. Store data temporarily during execution in the defined R/W memory locations called the stack.

To perform these operations, the microprocessor requires registers, arithmetic/logic unit (ALU) and control logic, and internal buses (paths for information flow). Figure 3.2 is a simplified representation of the 8085 internal architecture; it shows only those registers that are programmable, meaning those registers that can be used for data manipulation by writing instructions.

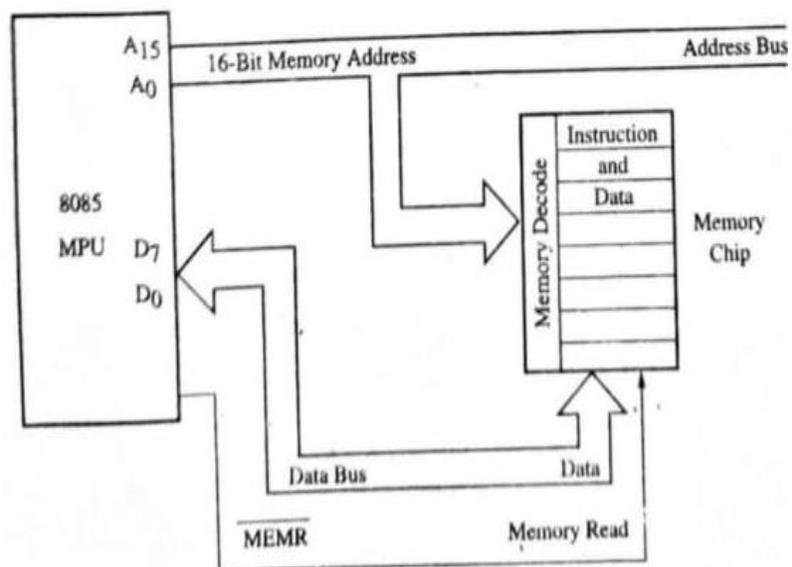
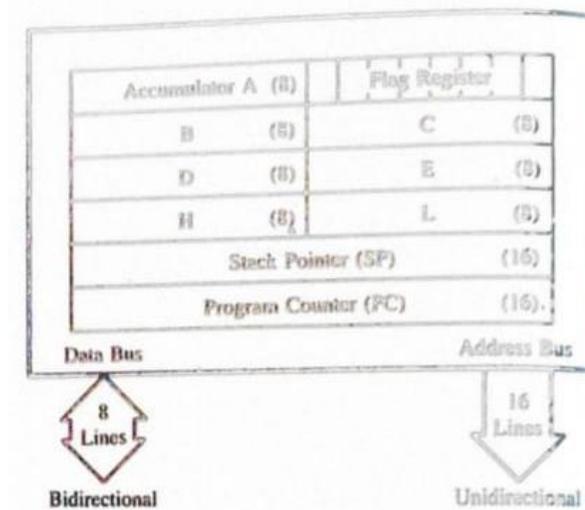


Fig. 3.2: Memory Read Operation**Fig. 3.3**

3.5.5. THE 8085 PROGRAMMABLE REGISTERS

These registers are described in reference to the five operations previously listed.

RESISTERS

The 8085 has six general-purpose registers to perform the first operation listed above; that is, to store 8-bit data during program execution. These registers are identified as B, C, D, E, H and L. They can be combined as register pairs-BC, DE and HL-to perform some 16-bit operations.

These registers are **programmable**, meaning that a programmer can use them to load or copy data from the registers by using instructions. For example, the instruction MOV B,C copies the data from register C to register B. Conceptually, the registers can be viewed as memory locations, except they are built inside the microprocessor and identified by specific letters for user convenience. Some microprocessors do not have these types of registers; instead, they use memory space as their registers.

ACCUMULATOR

The accumulator is an 8-bit register that is part of the arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

FLAGS

The ALU includes five flip-flops that are set or reset according to the result of an operation. The microprocessor uses them to perform the third operation; namely, testing for data conditions.

For example, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop that is used to indicate a carry, called the **Carry (CY) flag**, is set to one. When an arithmetic operation results in zero, the flip-flop called the **Zero (Z) flag** is set to one. The 8085 has five flags to indicate five different types of results or data conditions. They are Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. The most commonly used flags are Sign, Zero and Carry; the others will be explained as necessary. An 8-bit register shown in fig. called the **flag register**, adjacent to the accumulator. It is not used as an 8-bit register; five bit positions, out of eight, are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction. In the instruction set, the term *PSW* (Program Status Word) refers to the accumulator and the flag register.

These flags have critical importance in the decision-making process of the microprocessor. The conditions (set or reset) of the flags are tested through software instructions. For example, the instruction JC (Jump On Carry) is implemented to change the sequence of a program when the CY

flag is set. The importance of the flags cannot be emphasized enough; they will be discussed again in applications of conditional jump instructions.

PROGRAM COUNTER (PC)

This 16-bit register deals with the fourth operation, sequencing the execution of instructions. This register is a **memory pointer**. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.

The microprocessor uses this register to sequence the execution of instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

STACK POINTER (SP)

The stack pointer is also a 16-bit register used as a memory pointer; initially, it will be called the stack pointer register to emphasize that it is a register. It points to a memory location in R/W memory, called the **stack**. The beginning of the stack is defined by loading a 16-bit address in the stack pointer (register).

3.5.6. PERIPHERAL OR EXTERNALLY INITIATED OPERATIONS

External devices (or signals) can initiate the following operations, for which individual pins on the microprocessor chip are assigned: Reset, Interrupt, Ready, Hold.

Reset: When the reset pin is activated by an external key (also called a reset key), all internal operations are suspended and the program counter is cleared (it holds 0000H). Now the program execution can again begin at the zero memory address.

Interrupt: The microprocessor can be interrupted from the normal execution of instructions and asked to execute some other instructions called a **service routine** (for example, emergency procedures). The microprocessor resumes its operation after completing the service routine.

Ready: The 8085 has a pin called READY. If the signal at this READY pin is low, the microprocessor enters into a Wait state. This signal is used primarily to synchronize slower peripherals with the microprocessor.

Hold: When the HOLD pin is activated by an external signal, the microprocessor relinquishes control of buses and allows the external peripheral to use them. For example, the HOLD signal is used in Direct Memory Access (DMA) data transfer.

3.6. MICROPROCESSOR INSTRUCTION SET AND COMPUTER LANGUAGES

Microprocessors recognize and operate in binary numbers. However, each micro-processor has its own binary words, instructions, meanings, and language. The words are formed by combining a number of bits for a given machine. The word (or word length), as defined earlier, is the number of bits the microprocessor recognizes and processes at a time. The word length ranges from 4 bits for small, microprocessor-based computers, to 32 bits for such large computers as the IBM ES 9000 series. Another term commonly used to express word length is byte. The byte is defined as a group of eight bits. For example, a 16-bit microprocessor has a word length equal to two bytes. The term "nibble," which stands for a group of four bits, is also found in popular computer magazines and books. (A byte has two nibbles.)

The instruction is defined as a complete task (such as Add) the microprocessor can perform it can be made up of one or more words. Each machine has its own set of instructions based on the design of its CPU or its microprocessor. To be intelligible to the microprocessor, instructions must be written in binary lan-guage, also known as machine language. However, it is difficult for human beings to write programs in sets of Os and Is. Therefore, microprocessor manufacturers have devised English-like words to represent the binary instructions of a machine, and programmers can write programs using these words. These are called assem-bly language programs. Because an assembly language is specific to a given ma-chine programs written in assembly language are not transferable from one ma-chine to another. To circumvent this limitation, such general-purpose languages as BASIC, FORTRAN, PASCAL, and C have been devised so that a program written in these languages can be machine-independent. These languages are called high-level languages (HLL). This section deals with various aspects of these three types of languages: machine,

assembly, and high-level. The machine and assembly languages are discussed in the context of the 280 microprocessor.

3.7. MACHINE LANGUAGE

The number of bits in a word for a given machine is fixed, and words are formed through various combinations of these bits. For example a machine with a word length of eight bit scan has 756 a combination of eight bits-thus a language of 256 words. However, not all of these words need to be used in the machine. The microprocessor design engineer selects combinations of bit patterns and gives a specific meaning to each combination by using electronic logic circuits; this is called an instruction .The set of instructions designed into the machine makes up what is called the machine language, a binary language composed of Os and Is. Its words, its instructions, and their meanings are specific to each computer.

3.7.1. WRITING AND EXECUTING AN ASSEMBLY LANGUAGE PROGRAM

To write and execute an assembly lan-guage program manually on a single-board computer, with a Hex keyboard for input and LEOs (or seven-segment LEOs) for output, the following steps are necessary:

1. Write the instructions in mnemonics obtained from the instruction set supplied by the manufacturer.
2. Find the hexadecimal machine code for each instruction by searching through the set of instructions.
3. Enter (load) the program in the user memory in a sequential order by using the Hex keyboard as the input device.
4. Execute the program by pressing the Execute key. The answer will be displayed by the LEOs.

When the user program is entered by the keys, each entry is interpreted and converted into its binary equivalent by the monitor program and the machine code is stored as eight bits in each memory location in a sequence. When the Execute command is given, the microprocessor fetches

each instruction, decodes it and executes it in a sequence until the end of the program. The manual assembly procedure is commonly used in single-board micro-computers and is suited for small programs. However, the steps of looking up the machine codes and entering the program, which are tedious and object to errors, can be avoided by using an assembler on a microcomputer system.

The assembler is a program that translates the mnemonics entered by the ASCII keyboard into the corresponding binary machine codes of the microprocessor. Each microprocessor has its own assembler because the mnemonics and machine codes are specific to the microprocessor being used, and each assembler has certain rules that must be learned by the programmer. Assemblers are discussed in detail new topic.

3.8. HIGH-LEVEL LANGUAGES

Programming languages that are intended to be machine-independent are called high-level languages. The list includes such languages as C, FORTRAN, BASIC, PASCAL, and COBOL. These languages have certain sets of rules and draw on symbols and conventions from English. Instructions written in these languages are known as statements rather than mnemonics. A program written in BASIC for a microcomputer with the Z80 microprocessor can generally run on another microcomputer with a different microprocessor.

Now the question is: How do words in English get converted into the binary languages of different microprocessors? The answer lies with another program called either a compiler or an interpreter. These programs accept English-like statements as their input, called the source code. The compiler or interpreter then translates the source code into the machine language compatible with the microprocessor being used in the system. This translation into the machine language is called the object code. Each microprocessor needs its own compiler or interpreter for each high-level language. The primary difference between a compiler and an interpreter is in the process of generating machine code. The compiler reads the entire program first and then generates the object code, while the interpreter reads one instruction at a time, produces its object code, and executes the instruction before reading the next instruction. Compiled programs are executed much faster than interpreted programs. M-Basic is a common example of an interpreter for the BASIC language. Compilers are generally used in such languages as C, FORTRAN, and PASCAL.

Compilers and interpreters require large memory space because each instruction in English requires several machine codes to translate that instruction into binary. On the other hand, there is a one-to-one correspondence between the assembly language mnemonics and the machine code. Thus, assembly language programs are compact and require less memory space; they are more efficient than the high-level language programs. The primary advantage of high-level languages is in troubleshooting programs, also known as debugging. It is much easier to find errors in a program written in a high-level language than to find them in a program written in assembly language.

In certain applications such as traffic control and appliance control, where programs are small and compact, assembly language is suitable. Similarly, in such real-time applications as converting a high-frequency waveform into digital data, program efficiency is critical. In real-time applications, events and time should closely match with each other without significant delay. Therefore, assembly language is highly desirable in these applications.

3.9. OPERATING SYSTEM

An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs. Time sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it. Operating systems are found on many devices that contain a computer from cellular phones and video game consoles to web servers and supercomputers.

The dominant general-purpose personal computer operating system is Microsoft Windows with a market share of around 76.45%. macOS by Apple Inc. is in second place (17.72%), and the varieties of Linux are collectively in third place (1.73%). In the mobile sector (including smartphones and tablets), Android's share is up to 72% in the year 2020. According to third quarter 2016 data, Android's share on smartphones is dominant with 87.5 percent with a growth rate of

10.3 % per year, followed by Apple's iOS with 12.1% with per year decrease in market share of 5.2 %, while other operating systems amount to just 0.3 percent. Linux distributions are dominant in the server and supercomputing sectors. Other specialized classes of operating systems (special-purpose operating systems), such as embedded and real-time systems, exist for many applications. Security-focused operating systems also exist. Some operating systems have low system requirements (e.g. light-weight Linux distribution). Others may have higher system requirements. Some operating systems require installation or may come pre-installed with purchased computers (OEM-installation), whereas others may run directly from media (i.e. live CD) or flash memory.

3.9.1. TYPES OF OPERATING SYSTEMS

Single-tasking and multi-tasking

A single-tasking system can only run one program at a time, while a multi-tasking operating system allows more than one program to be running concurrently. This is achieved by time sharing, where the available processor time is divided between multiple processes. These processes are each interrupted repeatedly in time slices by a task-scheduling subsystem of the operating system. Multi-tasking may be characterized in pre-emptive and cooperative types. In pre-emptive multitasking, the operating system slices the CPU time and dedicates a slot to each of the programs. UNIX like operating systems, such as Linux as well as non-Unix-like, such as Amiga OS support pre-emptive multitasking. Cooperative multitasking is achieved by relying on each process to provide time to the other processes in a defined manner. 16-bit versions of Microsoft Windows used cooperative multi-tasking; 32-bit versions of both Windows NT and Win9x used pre-emptive multi-tasking.

Single and multi user

Single-user operating systems have no facilities to distinguish users but may allow multiple programs to run in tandem. A multi-user operating system extends the basic concept of multi-tasking with facilities that identify processes and resources, such as disk space, belonging to multiple users, and the system permits multiple users to interact with the system at the same time. Time-sharing operating systems schedule tasks for efficient use of the system and may also include

accounting software for cost allocation of processor time, mass storage, printing, and other resources to multiple users.

Distributed

A distributed operating system manages a group of distinct, networked computers and makes them appear to be a single computer, as all computations are distributed (divided amongst the constituent computers).

Embedded

Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines with less autonomy (e.g. PDAs). They are very compact and extremely efficient by design and are able to operate with a limited amount of resources. Windows CE and Minix 3 are some examples of embedded operating systems.

Real-time

A real-time operating system is an operating system that guarantees to process events or data by a specific moment in time. A real time operating system may be single or multi-tasking, but when multitasking, it uses specialized scheduling algorithms so that a deterministic nature of behaviour is achieved. Such an event driven system switches between tasks based on their priorities or external events, whereas time sharing operating systems switch tasks based on clock interrupts.

Library

A library operating system is one in which the services that a typical operating system provides, such as networking, are provided in the form of libraries and composed with the application and configuration code to construct a unikernel a specialized, single address space, machine image that can be deployed to cloud or embedded environments

3.10 SUMMARY

In this unit, you have studied about microprocessor 8085 and its Pin diagram. You learnt in detail working and application of 8085. You have learnt about evolution of micro processors. You learnt through table about microprocessor, year of invention and in diagram. You learn about machine language also.

3.11 GLOSSARY

PGA - Pin Grid Array

MMX - MultiMedia eXtensions

EPIC - Explicitly Parallel Instruction Computing

SIMD - Single Instruction Multiple Data

ALU - Arithmetic and Logic Unit

MMU - Memory Management Unit

FPU - Floating Point Unit

3.12 REFERENCES

1. Microprocessor Architecture, Programming and Applications with the 8085 by Ramesh Gaonkar
2. Microprocessor and Microcontroller System By A. P. Godse
3. The 8085 Microprocessor: Architecture, Programming and Interfacing by B. S. Umashankar and K. Udaya Kumar.
4. Advanced Microprocessors and Peripherals by A. K. Ray
5. Online sources

3.13 SUGGESTED READINGS

1. Digital Fundamentals, 10th Ed, Floyd T L, Prentice Hall, 2009.
2. NPTEL
3. You Tube
4. Online tutorial
5. Byju's online study material

3.15 TERMINAL QUESTIONS

3.14.1 Short Answer type

1. Draw pin diagram of 8085.
2. Draw architecture 8085.
3. What do you understand by ALU?
4. Discuss control and timing unit.

UNIT 4

8085 MICROPROCESSOR

AND MEMORY INTERFACES

Structure

4.1 Introduction

4.2 Objectives

4.3. 8085 Microprocessor Architecture And Memory Interfacing

4.3.1. Architecture Of 8085 Microprocessor

4.3.2. Operations Of Microprocessor

4.3.2.1. The 8085 Instruction Format

4.3.3. The 8085 Addressing Modes

4.4. 8085 Instruction Set

4.5. Address Bus

4.5.1. Multiplexed Address/Data Bus

4.5.2. Control And Status Signals

4.5.3. Power Supply And Clock Frequency

4.6. Externally Initiated Signals, Including Interrupts

4.6.1. Serial I/O Ports

4.7 Microprocessor Communication And Bus Timings

4.7.1. Demultiplexing The Bus Ad_1 - Ad_0

- 4.8. Generating Control Signals
- 4.9. A Detailed Look At The 8085 Mpu And Its Architecture
 - 4.9.1. The Alu
 - 4.9.2. Timing And Control Unit
 - 4.9.3. Instruction Register And Decoder
 - 4.9.4. Register Array
- 4.10. Interrupts Of 8085
 - 4.10.1. Hardware And Software Interrupts
 - 4.10.2. Vectored And Non-Vectored Interrupts
 - 4.10.3. Maskable And Non-Maskable Interrupts
 - 4.10.4. Priority Of Interrupts
 - 4.10.5. Instruction For Interrupts
- 4.11. Timing Diagrams
- 4.12 Summary
- 4.13 Glossary
- 4.14 References
- 4.15 Suggested Readings
- 4.16 Terminal Questions
 - 4.16.1 Short Answer Type

4.1 INTRODUCTION

The 8085 microprocessor is a much improved version of its predecessor, the 8080A. The 8085 includes on its chip most of the logic circuitry for performing computing tasks and for communicating with peripherals. The architecture of 8085 consists of three main sections, ALU (Arithmetic and Logical Unit), timing and control unit and Registers. ALU performs all logical operations such as addition, subtraction etc. The data flow is controlled by timing and control unit.

An instruction is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts, one is task to be performed, called the operation code (opcode), and the second is the data to be operated on called the operand. An instruction is a binary pattern designed inside a microprocessor to perform a specific function. Each instruction is represented by 8 bit binary value. The 8085 has five interrupt signals that can be used to interrupt a program execution.

The timing and control unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals.

4.2 OBJECTIVES

After studying this unit, you will learn about-

- Microprocessor 8085
- Pin diagram 8085
- Interrupts
- Timing diagram

4.3. 8085 MICROPROCESSOR ARCHITECTURE AND MEMORY INTERFACING

The 8085 microprocessor is a much improved version of its predecessor, the 8080A. The 8085 includes on its chip most of the logic circuitry for performing computing tasks and for

communicating with peripherals. However, eight of its bus lines are **multiplexed**; that is, they are time-shared by the low-order address and data. This chapter discusses the 8085 architecture in detail and illustrates techniques for demultiplexing the bus and generating the necessary control signals.

It has 8 bit data bus and 16 bit address bus, thus it is capable of addressing 64 KB of memory.

1. It has 8 bit ALU 8 bit ALU that can perform 8 bit operations.
2. Lower order address bus is multiplexed with data bus to minimize the chip size.
3. The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package (shown the figure below) and uses +5 V for power. It can run at a maximum frequency of 3 MHz.
4. The 8085 has extensions to support new interrupts, with three maskable interrupts (RST 7.5, RST 6.5 and RST 5.5), one non-maskable interrupt (TRAP), and one externally serviced interrupt (INTR).
5. Three control signals are available on chip: (i) RD : it is a active low signal. Which indicate that the selected IO or Memory device is to be read and data is available on the data bus. (ii) WR : it is a active low signal which indicate that the data on the data bus are to be written into a selected memory or IO location. (iii) ALE : it is a +ve going pulse generated every time the 8085 begins an operation (machine cycle), which indicate that the bits on AD7-AD0 are address bits.
6. Three status signals are available on chip: (i) IO/M : this is a status signal used to differentiate between IO and Memory operations. If it is high then IO operation and If it is low then Memory operation. (ii) S1 and S0: status signals similar to IO/M, can identify various operations that are rarely used in the systems.

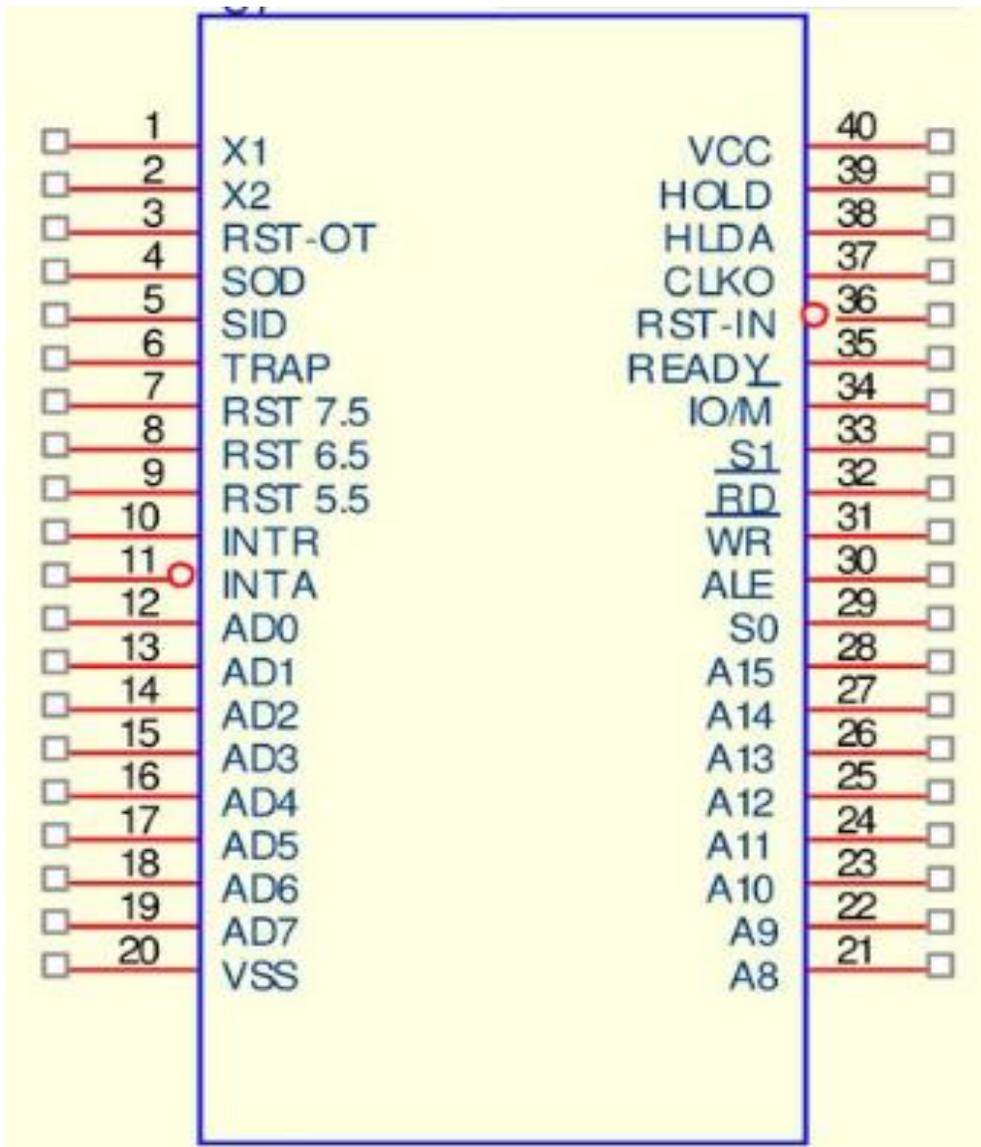


Fig. 4.1 A: PIN DIAGRAM of 8085

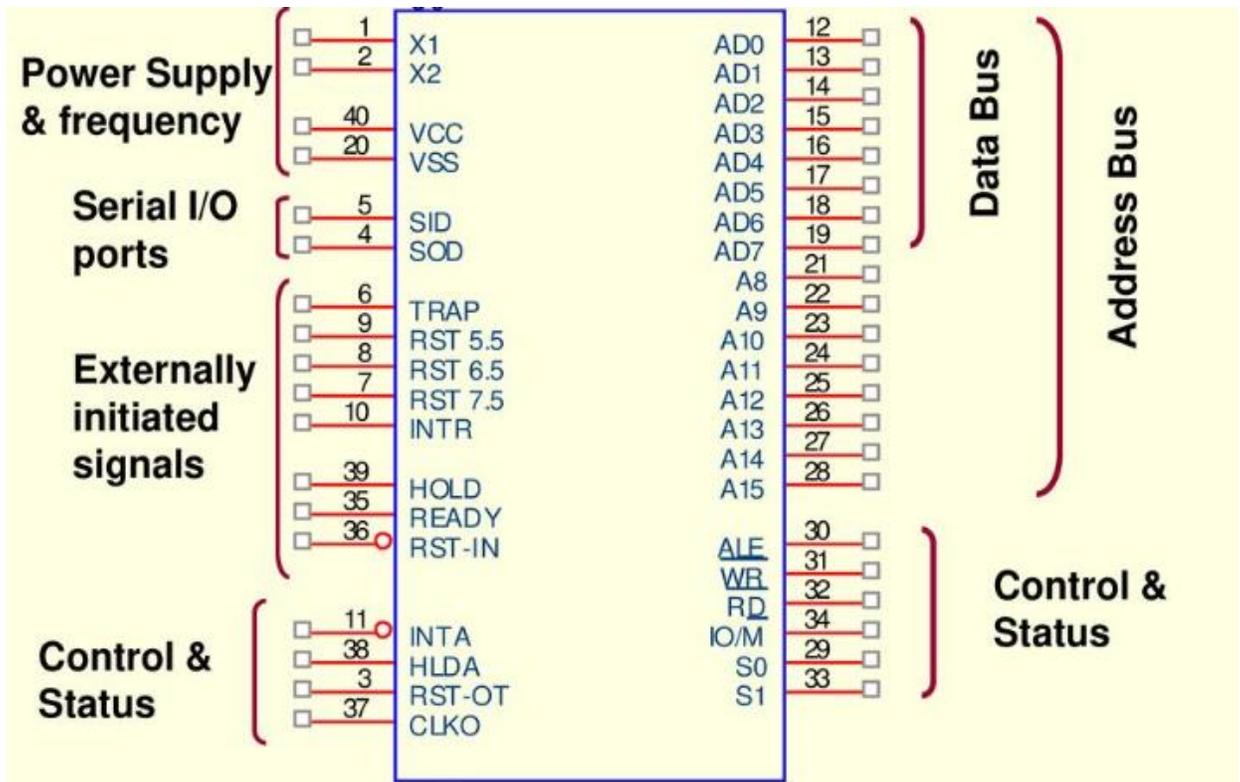


Fig. 4.1 B: PIN DIAGRAM of 8085

4.3.1. ARCHITECTURE OF 8085 MICROPROCESSOR

The architecture of 8085 consists of three main sections, ALU (Arithmetic and Logical Unit), timing and control unit and Registers (shown in the following figure).

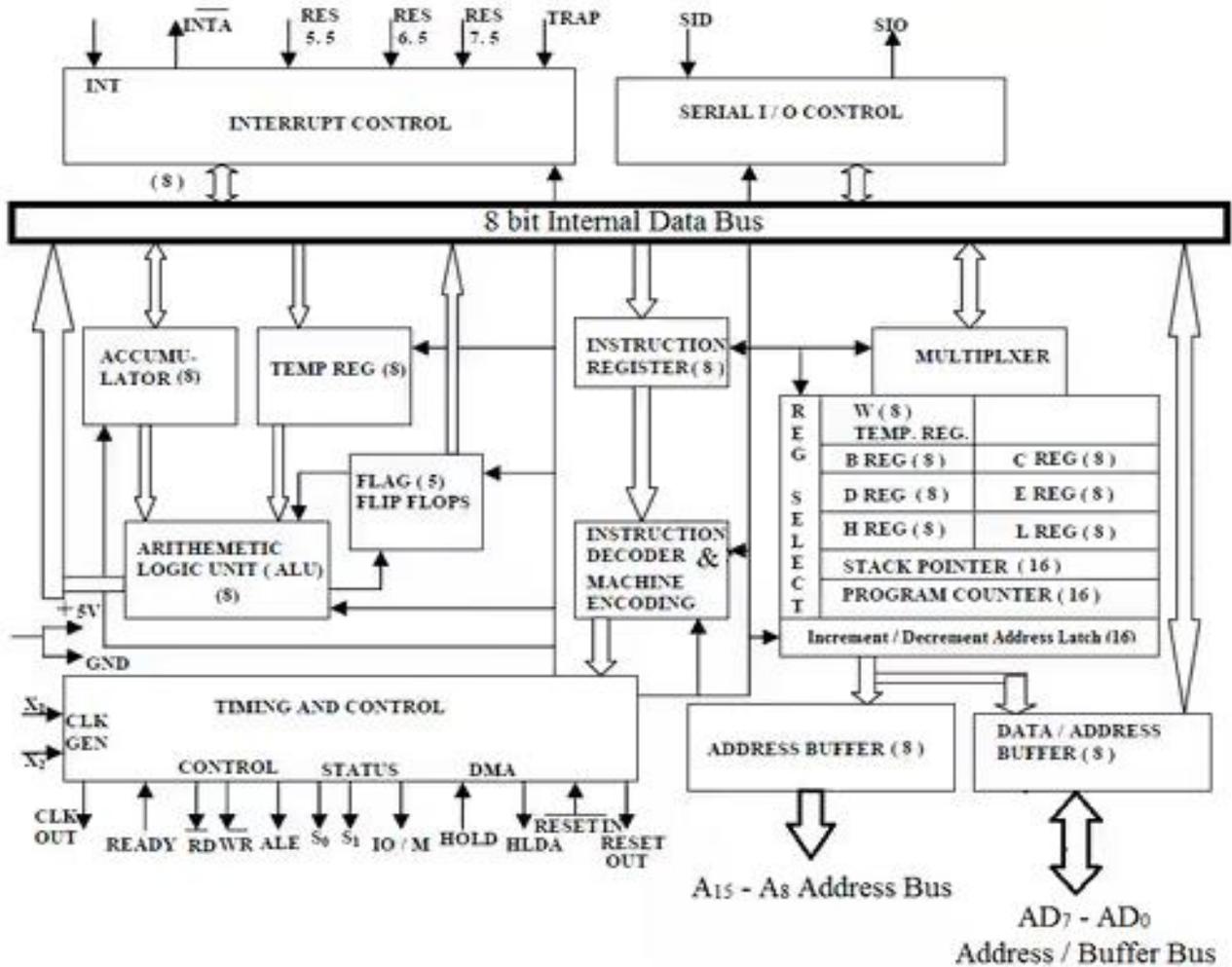


Fig. 4.2 ARCHITECTURE of 8085

ARITHMETIC AND LOGIC UNIT (ALU): The ALU performs the actual numerical and logical operations.

1. The ALU performs the following arithmetic and logical operations.
2. Addition, Subtraction
3. Logical AND, Logical OR, Logical Ex - OR
4. Complement (logical NOT)
5. Increment, Decrement
6. Left shift, Right shift

7. Clear, etc.

ALU includes the accumulator, the temporary register, the arithmetic and logic circuits and flags. It always stores result of operations in Accumulator.

TIMING & CONTROL UNIT: It generates timing and control signals, which are necessary for the execution of instructions.

- It controls data flow between CPU and peripherals (including memory).
- It provides status, control and timing signals, which are required for the operation of memory and I/O devices.

8085 SYSTEM BUS: Microprocessor communicates with memory and other devices (input and output) using three buses: Address Bus, Data Bus and Control Bus.

ADDRESS BUS: The Address bus consists of 16 wires. The size of the address bus determines the size of memory, which can be used. To communicate with memory the microprocessor sends an address on the address bus to the memory. Address bus is unidirectional, i.e., numbers only sent from microprocessor to memory.

DATA BUS: Bus is bidirectional. Size of the data bus determines what arithmetic can be done. Data bus also carries instructions from memory to the microprocessor.

Memory size = $2^A \times D$ where, A denotes the address lines, and D denotes the data lines.

CONTROL BUS: Control bus are various lines which have specific functions for coordinating and controlling μ P operations. The control bus carries control signals partly unidirectional, partly bidirectional. Control signals are things like read or write.

REGISTERS: 8085 has six general-purpose registers to store 8 bit data, these are identified as B, C, D, E, H and L . They can be combined as register pairs BC, DE and HL to perform some 16 bit operations.

ACCUMULATOR: The accumulator is an 8 bit register included as a part of Arithmetic Logic Unit (ALU). This register is used to store 8 bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator.

FLAG REGISTER: The ALU includes five flip-flops. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. The microprocessor uses these flags to test data conditions. The conditions (set or reset) of the flags are tested through the software instructions. The combination of the flag register and the accumulator is called Program Status Word (PSW) and PSW is the 16-bit unit for stack operation.

PROGRAM COUNTER (PC): This 16 bit register deals with sequencing the execution of instruction. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched.

STACK POINTER (SP): The stack pointer is also a 16 bit register used as a memory pointer. It points to a memory location in read-write memory, called the stack.

INSTRUCTION REGISTER/DECODER: Temporary store for the current instructions of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage.

MEMORY ADDRESS REGISTER: Holds address, received from PC of next program instruction.

CONTROL GENERATOR: It generates signal within μP to carry out the instructions which have been decoded.

REGISTER SELECTOR: This block controls the use of the register stack.

GENERAL PURPOSE REGISTERS: μP requires extra registers for versatility. It can be used to store additional data during a program.

4.3.2. OPERATIONS OF MICROPROCESSOR

The microprocessor performs the following four operations using address bus, data bus, and control bus:

Memory Read: Reads data (or instruction) from memory.

Memory Write: Writes data (or instruction) into memory.

I/O Read: Accepts data from input device.

I/O Write: Sends data to output device.

4.3.2.1. THE 8085 INSTRUCTION FORMAT

An instruction is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts, one is task to be performed, called the operation code (opcode), and the second is the data to be operated on called the operand. The 8085 instruction set is classified according to word size.

ONE-BYTE INSTRUCTIONS: A 1-byte instruction includes the opcode and operand in the same byte. Operands are internal registers and are coded into the instruction.

TWO-BYTE INSTRUCTIONS: In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode.

THREE-BYTE INSTRUCTIONS: In a three byte instruction, the first byte specifies the opcode and the following two bytes specify the 16-bit address. Note that, the second byte is the low-order address and the third byte is the high-order address.

4.3.3. THE 8085 ADDRESSING MODES

The various formats for specifying operands are called the addressing modes. For 8085, they are

IMMEDIATE ADDRESSING: Data is provided in the instruction. Load the immediate data to the destination provided.

REGISTER ADDRESSING: Data is provided through the registers.

DIRECT ADDRESSING: Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device.

INDIRECT ADDRESSING: Effective address is calculated by the processor and the contents of the address are used to form a second address. The second address is where the data is stored.

IMPLICIT ADDRESSING: In this addressing mode the data itself specifies the data to be operated upon.

4.4. 8085 INSTRUCTION SET

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. Each instruction is represented by 8 bit binary value. Instruction set can be categorized into 5 types:

DATA TRANSFER INSTRUCTIONS: These instructions are used to transfer data from one register to another register, from memory to register or register to memory. When an instruction of data transfer group is executed, data is transferred from the source to the destination without altering the contents of the source.

ARITHMETIC INSTRUCTIONS: These instructions are used to perform arithmetic operations such as addition, subtraction, increment or decrement of the content of a register or memory.

LOGICAL INSTRUCTIONS: These instructions are used to perform logical operations such as AND, OR, compare, rotate etc.

BRANCHING INSTRUCTIONS: These instructions are used to perform conditional and unconditional jump, subroutine call and return, and restart.

MACHINE CONTROL INSTRUCTIONS: These instructions control machine functions such as Halt, Interrupt, or do nothing. The microprocessor operations related to data manipulation can be summarized in four functions: copying data, performing arithmetic operations, performing logical operations, testing for a given condition and alerting the program sequence.

4.5. ADDRESS BUS

The 8085 has eight signal lines $A_1 - A_8$, which are unidirectional and used as the high order address bus.

4.5.1. MULTIPLEXED ADDRESS/DATA BUS

The signal lines $AD-AD_0$ are bidirectional: they serve a dual purpose. They are used the low-order address bus as well as the data bus. In executing an instruction, during Part of the cycle, these lines are used as the low-order address bus. During the later part of the cycle these lines are used as the data bus. (This is also known as multiplexing the bus. However, the low-order address bus can be separated from these signals by using a latch .

4.5.2. CONTROL AND STATUS SIGNALS

The group of signals includes two control signals (\overline{RD} and \overline{WR}), three status signals (IO/\overline{M} , S_1 and S_0) to identify the nature of the operation, and one special signal (ALE to indicate the beginning of the operation. These signals are as follows:

- **ALE-** Address Latch Enable: This is a positive going pulse generated every time the 8085 begins an operation (machine cycle); it indicates that the bits on $AD-AD_0$ are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines, A_1-AD_0 .
- **\overline{RD} -Read:** This is a Read control signal (active low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.
- **\overline{WR} -Write:** This is a Write control signal (active lo). This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.
- **IO/\overline{M} :** This is a status signal used to differentiate between I/O and memory operations. When it is high, it indicates an I/O operation; when is low, it indicates a memory operation. This signal is combined with \overline{RD} (Read) and \overline{WR} (Write) to generate I/O and memory control signals.

- **S₁ AND S₀:** These status signals, similar to IO/ \bar{M} , can identify various operations, but they are namely used in small system. (All the operations and their associated status signals are listed in Table 3.1 for reference.)

4.5.3. POWER SUPPLY AND CLOCK FREQUENCY

The power supply and frequency signals are as follows:

- **V_{cc}:** +5 V power supply
- **V_{ss}:** Ground References
- **X₁, X₂:** A crystal (or RC, LC network) is connected at these two pins. The frequency is internally divided by two; therefore, to operate a system at 3 MHz, the crystal should have a frequency of 6 MHz.
- **CLK (OUT)-Clock Output:** This signal can be used as the system clock for other devices.

4.6. EXTERNALLY INITIATED SIGNALS, INCLUDING INTERRUPTS

The 8085 has five interrupt signals that can be used to interrupt a program execution. One of the signals, INTR (Interrupt Request), is identical to the 8080A microprocessor interrupt signal (INT); the others are enhancements to the 8080A. The microprocessor acknowledges an interrupt request by the INTA (Interrupt Acknowledge) signal.

In addition to the interrupts, three pins-RESET, HOLD and READY-accept the externally initiated signals as inputs. To respond to the HOLD request, the 8085 has one signal called HLDA (Hold Acknowledge).

RESET IN: When the signal on this pin goes low, the program counter is set to zero, the buses are tri-stated, and the MPU is reset.

- **RESET OUT:** This signal indicates that the MPU is being reset. The signal can be used to reset other devices.

4.6.1. SERIAL I/O PORTS

The 8085 has two signals to implement the serial transmission: SID (Serial Input Data) and SOD (Serial Output Data). They will be discussed in Chapter 16 on serial I/O.

In this chapter, we will focus on the first three groups of signals; others will be discussed in later chapters.

4.7 MICROPROCESSOR COMMUNICATION AND BUS TIMINGS

Bus: A bus is a group of wires (lines) that carry similar information.

System Bus: A system bus is a group of wires used for communication between the microprocessor and peripherals.

Address Bus: It carries the address, which is a unique binary pattern used to identify a memory location or an I/O port.

Data Bus: The data bus is used to transfer data between memory and processor or between I/O device and processor.

Control Bus: The control bus carries control signals, which consist of signals for selection of memory or I/O device from the given address, direction of data transfer, and synchronization of data transfer in case of slow devices.

To understand the functions of various signals of the 8085, we should examine the process of communication (reading from and writing into memory) between the microprocessor and memory and the timings of these signals in relating to the system clock. The first step in the communication process is reading from memory or fetching an instruction. This can be easily understood using an analogy of how a package is picked up from your house by a shipping company such as Federal Express. The steps are as follows:

1. A courier gets the address from the office; he or she drives the pickup van, finds the street, and looks for your house number.
2. The courier rings the bell.
3. Somebody in the house opens the door and gives the package to the courier, and the courier returns to the office with the package.
4. The internal office staff disposes the package according to the instructions given by the customer.

Now let us examine the steps in the following example of how the microprocessor fetches or gets a machine code from memory.

4.7.1. DEMULTIPLEXING THE BUS AD₁-AD₀

The need for demultiplexing the bus AD₁-AD₀ becomes easier to understand after examining Figure 4.3. A This figure shows that the address on the high-order bus (20H) remains on the bus for three clock periods. However, the low-order address (05H) is lost after the first clock period. This address needs to be latched and used for identifying the memory address. If the bus AD₇-AD₀ is used to identify the memory location (2005H), the address will change to 204FH after the first clock period.

Figure 4.3 B shows a schematic that uses a latch and the ALE signal to demultiplex the bus. The bus AD₇-AD₀ is connected us the input to the latch 74LS373. The ALE signal is connected to the Enable (G) pin of the latch, and the Output control (\overline{OC}) signal of the latch is grounded.

Figure 4.3A shows that the ALE goes high during T₁. When the ALE is high, the latch is transparent; this means that the output changes according to input data. During T₁, the output of the latch is 05H. When the ALE goes low, the data byte 05H is latched until the next ALE, and the output of the latch represents the low-order address bus A₇-A₀ after the latching operation.

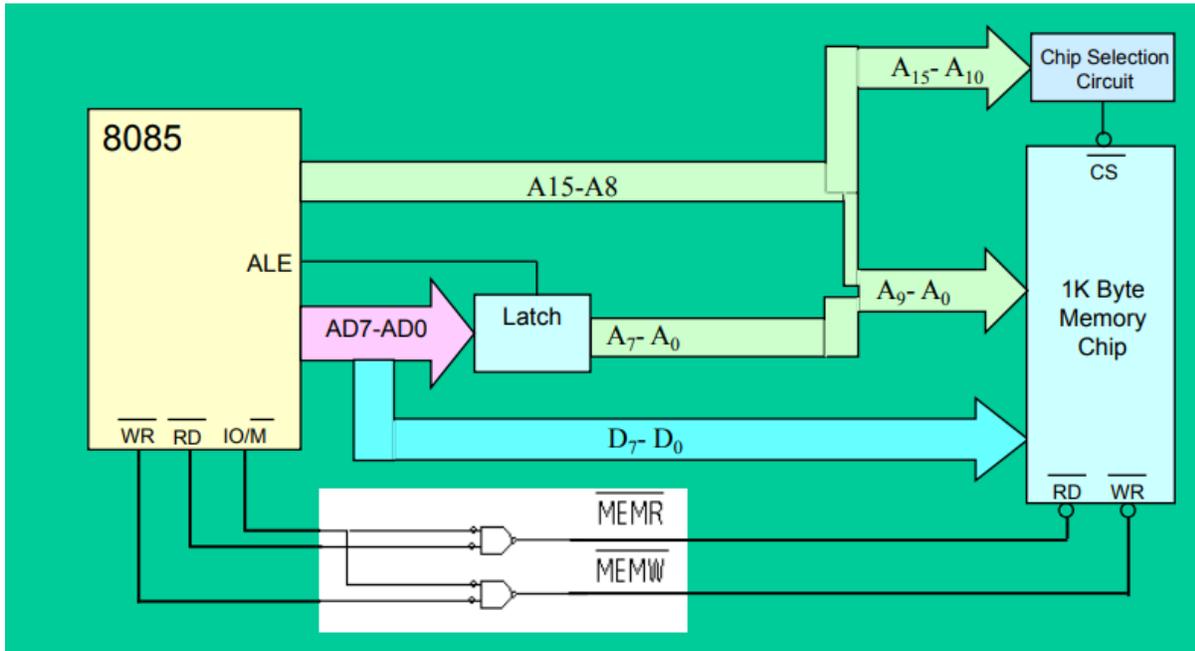


FIG 4.3. A: SCHEMATIC OF LATCHING LOW-ORDER ADDRESS BUS

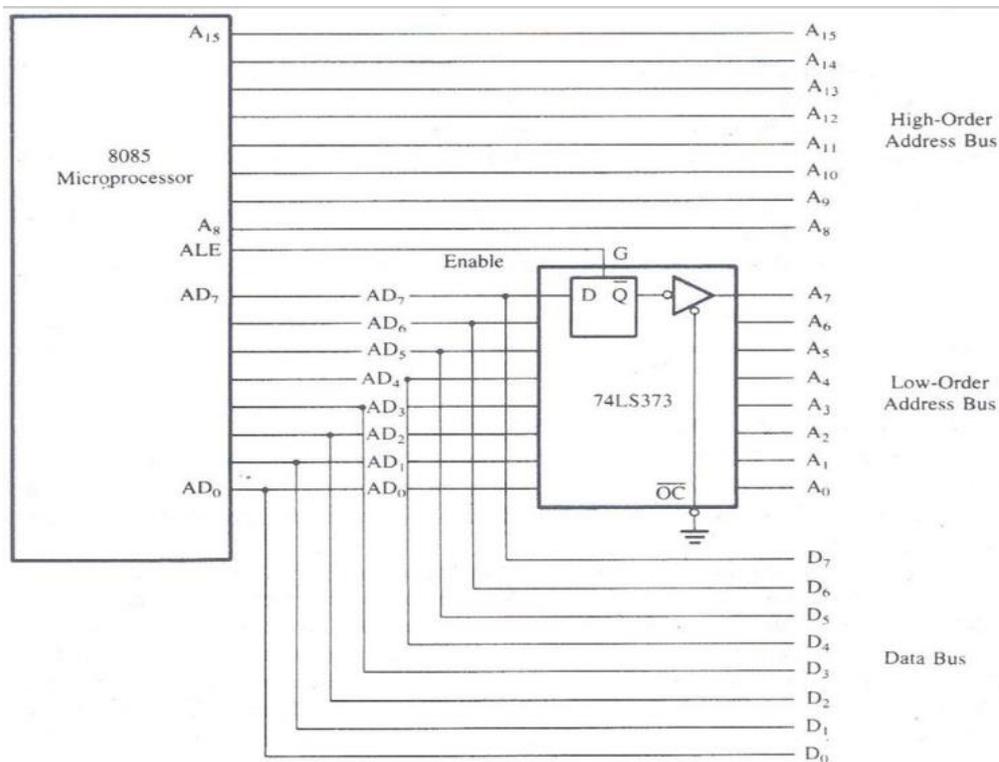


FIG 4.3. B: SCHEMATIC OF LATCHING LOW-ORDER ADDRESS BUS

Intel has circumvented the problem of demultiplexing the low-order bus by special devices such as the 8155 (256 bytes of R/W memory I/O which is potable with the 8085 multiplexed bus. These devices internally demultiplex the bus the ALE signal.

We can make the following observations:

1. The machine code 4FH (0100 1000) is a one-byte instruction that copies the of the accumulator into register C.
2. The 8085 microprocessor requires one external operation-fetching a machine from memory location 2005H.
3. The entire operation-fetching, decoding, and executing-requires four clock periods.

Now we can define three terms-instruction cycle, machine cycle and T-state. We use these terms later for examining timings of various 8085 operations

Instruction cycle is defined as the time required to complete the execution of an instruction. The 8085 instruction cycle consists of one to six machine cycles or one to many operations.

Machine cycle is defined as the time required completing one operation of accessing memory. I/O, or acknowledging an external request. This cycle may consist of three to six T-states. In Figure 3.3, the instruction cycle and the machine cycle are the same.

T-state is defined as one subdivision of the operation performed in one clock period. These subdivisions are internal states synchronized with the system clock, and each T-state is precisely equal to one clock period. The terms T-state and clock period are often used synonymously.

4.8. GENERATING CONTROL SIGNALS

Figure 4.4 shows the \overline{RD} (Read) as a control signal. Because this signal is used both for reading memory and for reading an input device, it is necessary to generate two different Read signals; one for memory and another for input. Similarly, two separate Write signals must be generated.

Figure 4.5 shows that four different control signals are generated by combining the signals \overline{RD} , \overline{WR} and IO/\overline{M} . The signal IO/\overline{M} goes low for the memory operation. This signal is AND_{ed} with \overline{RD} and \overline{WR} signals by using the 74LS32 quadruple two-input OR gates, as shown in Figure 4.4. The OR gates are functionally connected as negative NAND gates. When both input signals go low, the outputs of the gates go low and generate \overline{MEMR} (Memory Read) and \overline{MEMW} (Memory Write) control signals. When the IO/\overline{M} signal goes high, it indicates the peripheral I/O operation. Figure 4.4 shows that this signal is complemented using the Hex inverter 74LS04 and AND_{ed} with the \overline{RD} and \overline{WR} signals to generate \overline{IOR} (I/O Read) and \overline{IOW} (I/O Write) control signals.

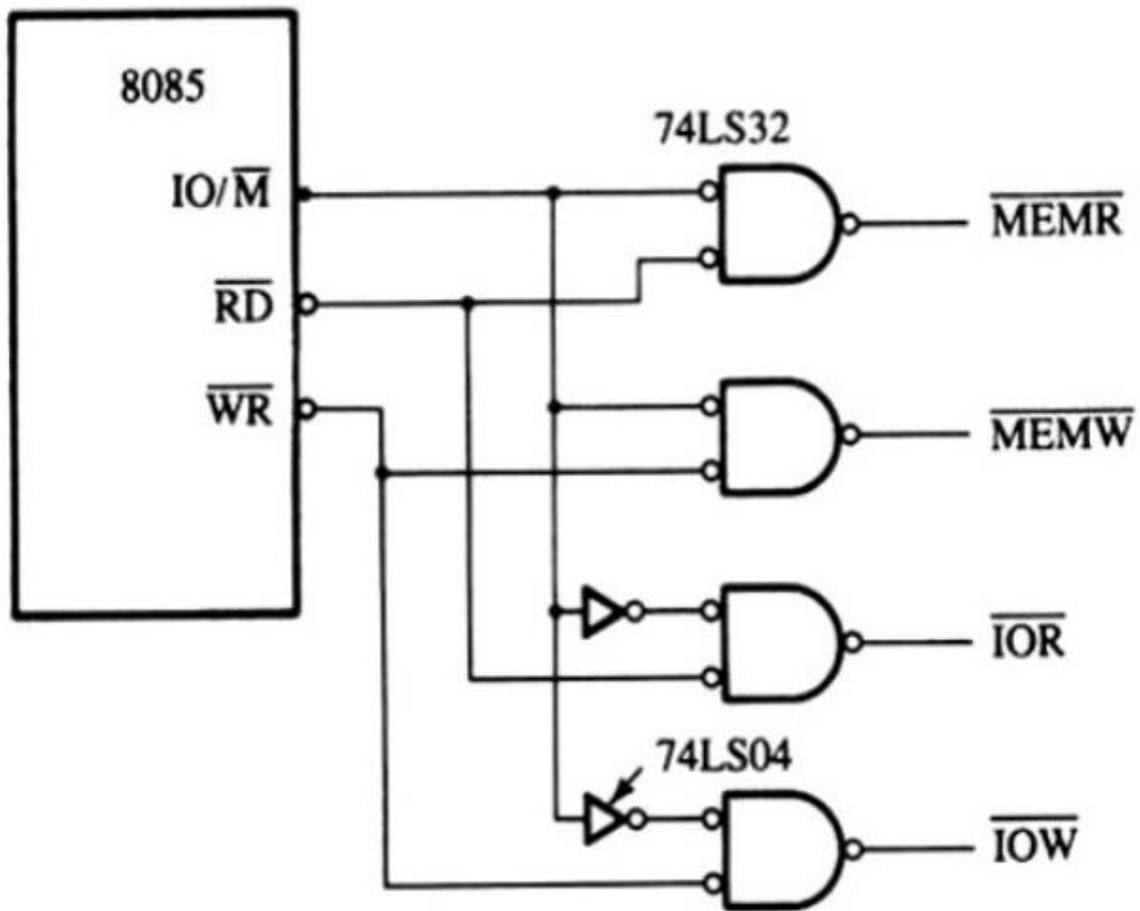


FIG. 4.4 SCHEMATIC TO GENERATE READ/WRITE CONTROL SIGNALS FOR MEMORY AND I/O.

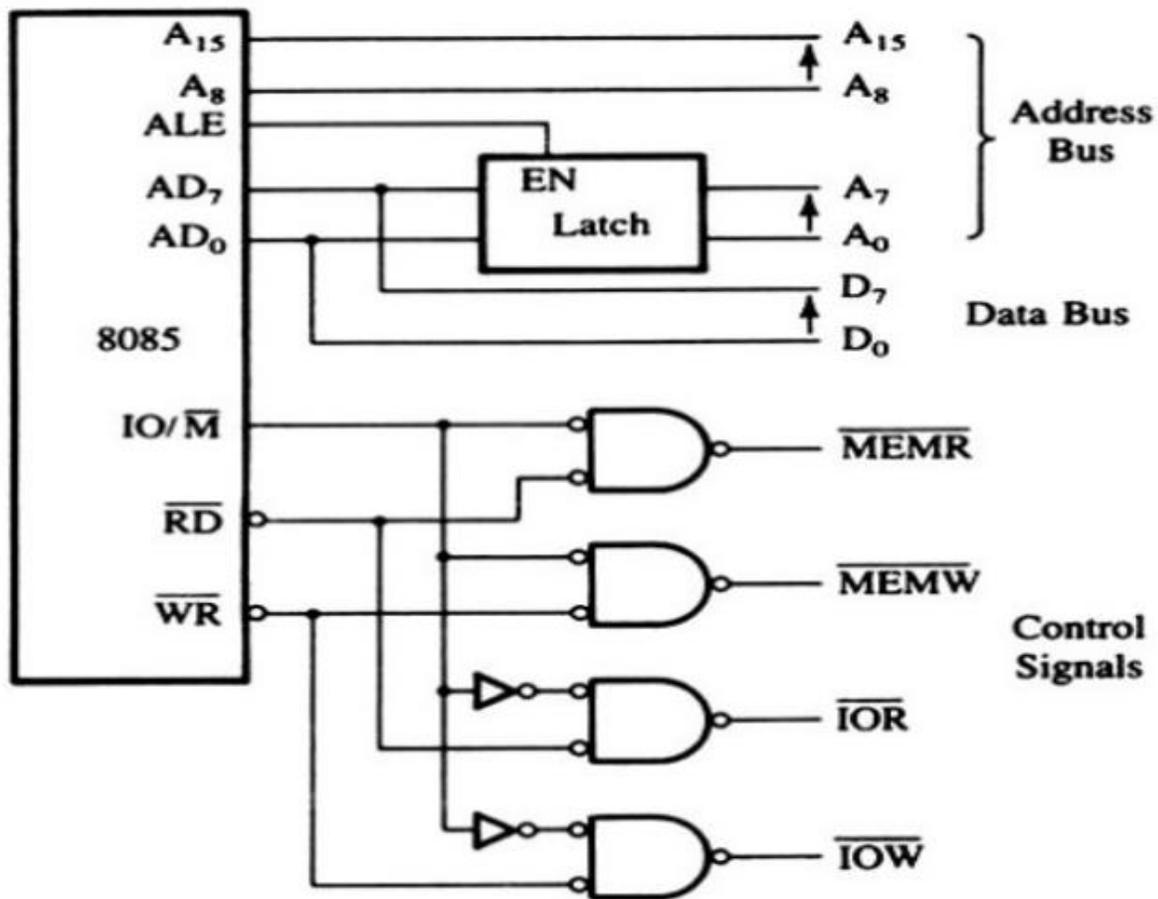


FIG. 4.5: - 8085 DEMULTIPLEXED ADDRESS AND DATA BUS WITH CONTROL SIGNALS

To demultiplex the bus and to generate the necessary control signals, the 8085 microprocessor requires a latch and logic gates to build the MPU. This MPU can be interfaced with any memory or I/O.

4.9. A DETAILED LOOK AT THE 8085 MPU AND ITS ARCHITECTURE

8085 includes the ALU (Arithmetic/Logic Unit), Timing and Control Unit, Instruction Register and Decoder, Register Array. Interrupt Control and Serial I/O Control.

4.9.1. THE ALU

The arithmetic/logic unit performs the computing functions; it includes the accumulator, the temporary register, the arithmetic and logic circuits and five flags. The temporary register is used to hold data during an arithmetic/logic operation. The result is stored in the accumulator, and the flags (flip-flops) are set or reset according to the result of the operation.

The flags are affected by the arithmetic and logic operations in the ALU. In most of these operations, the result is stored in the accumulator. Therefore, the flags generally reflect data conditions in the accumulator-with some exceptions. The descriptions and conditions of the flags are as follows:

- **S-Sign flag:** After the execution of an arithmetic or logic operation, if bit D_7 of the result (usually in the accumulator) is 1, the Sign flag is set. This flag is used with signed numbers. In a given byte, if D_7 is 1, the number will be viewed as a negative number, if it is 0, the number will be considered positive. In arithmetic operations with signed numbers, bit D_7 is reserved for indicating the sign, and the remaining seven bits are used to represent the magnitude of a number. See Appendix A2 for a discussion of signed numbers.)
- **Z-Zero flag:** The Zero flag is set if the ALU operation results in 0, and the flag is reset if the result is not 0. This flag is modified by the results in the accumulator as well as in the other registers.
- **AC-Auxiliary Carry flag:** In an arithmetic operation, when a carry is generated by digit D_3 and passed on to digit D_4 , the AC flag is set. The flag is used only internally for BCD (binary-coded decimal) operations and is not available for the programmer to change the sequence of a program with a jump instruction.
- **P-Parity flag:** After an arithmetic or logical operation, if the result has an even number of 1s, the flag is set. If it has an odd number of 1s, the flag is reset.(For example, the data byte 0000 0011 has even parity even if the magnitude of the number is odd.)
- **CY-Carry flag:** If an arithmetic operation results in a carry, the Carry flag is set; otherwise it is reset. The Carry flag also serves as a borrow flag for subtraction.

The bit positions reserved for these flags in the flag register are as follows:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
S	Z		AC		P		CY

Among the five flags, the AC flag is used internally for BCD arithmetic; the instruction set does not include any conditional jump instructions based on the AC flag. Of the remaining four flags, the Z and CY flags are those most commonly used.

4.9.2. TIMING AND CONTROL UNIT

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals.

The control signals are similar to a sync pulse in an oscilloscope. The \overline{RD} and \overline{WR} signals are sync pulse indicating the availability of data on the data bus.

4.9.3. INSTRUCTION REGISTER AND DECODER

The instruction register and the decoder are part of the ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow. The instruction register is not programmable and cannot be accessed through any instruction.

4.9.4. REGISTER ARRAY

The programmable registers were discussed in the last chapter. Two additional registers, called temporary registers W and Z, are included in the register array. These registers are used to hold 8-bit data during the execution of some instructions. However, because they are used internally, they are not available to the programmer.

4.10. INTERRUPTS OF 8085

When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating CALL signal and after executing sub-routine by generating RET signal again program control is transferred to main program from where it had stopped.

When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Interrupts can be classified into various categories based on different parameters:

4.10.1. HARDWARE AND SOFTWARE INTERRUPTS

When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as Hardware Interrupts. There are 5 Hardware Interrupts in 8085 microprocessor. They are – INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Software Interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 microprocessor. They are – RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.

4.10.2. VECTORED AND NON-VECTORED INTERRUPTS

Non-Vectored Interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. INTR is the only non-vectored interrupt in 8085 microprocessor.

4.10.3. MASKABLE AND NON-MASKABLE INTERRUPTS

Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 microprocessor.

Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. TRAP is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

4.10.4. PRIORITY OF INTERRUPTS

When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.

4.10.5. INSTRUCTION FOR INTERRUPTS

ENABLE INTERRUPT (EI) – The interrupt enable flip-flop is set and all interrupts are enabled following the execution of next instruction followed by EI. No flags are affected. After a system reset, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to enable the interrupts again (except TRAP).

DISABLE INTERRUPT (DI) – This instruction is used to reset the value of enable flip-flop hence disabling all the interrupts. No flags are affected by this instruction.

SET INTERRUPT MASK (SIM) – It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line. First the required value is loaded in accumulator then SIM will take the bit pattern from it.

READ INTERRUPT MASK (RIM) – This instruction is used to read the status of the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by loading into the A register a byte which defines the condition of the mask bits for the interrupts. It also reads the condition of SID (Serial Input Data) bit on the microprocessor.

4.11. TIMING DIAGRAMS

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

INSTRUCTION CYCLE: The time required to execute an instruction is called instruction cycle.

MACHINE CYCLE: The time required to access the memory or input/output devices is called machine cycle.

T-STATE: The machine cycle and instruction cycle takes multiple clock periods. A portion of an operation carried out in one system clock period is called as T-state.

MACHINE CYCLES OF 8085

The 8085 microprocessor has 5 basic machine cycles. They are

Opcode fetch cycle (4T)

Memory read cycle (3 T)

Memory write cycle (3 T)

I/O read cycle (3 T)

I/O write cycle (3 T)

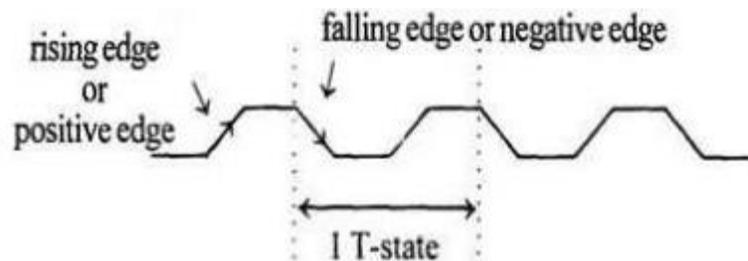


FIG 4.6: CLOCK SIGNAL

Signal 1.Opcode fetch machine cycle of 8085:

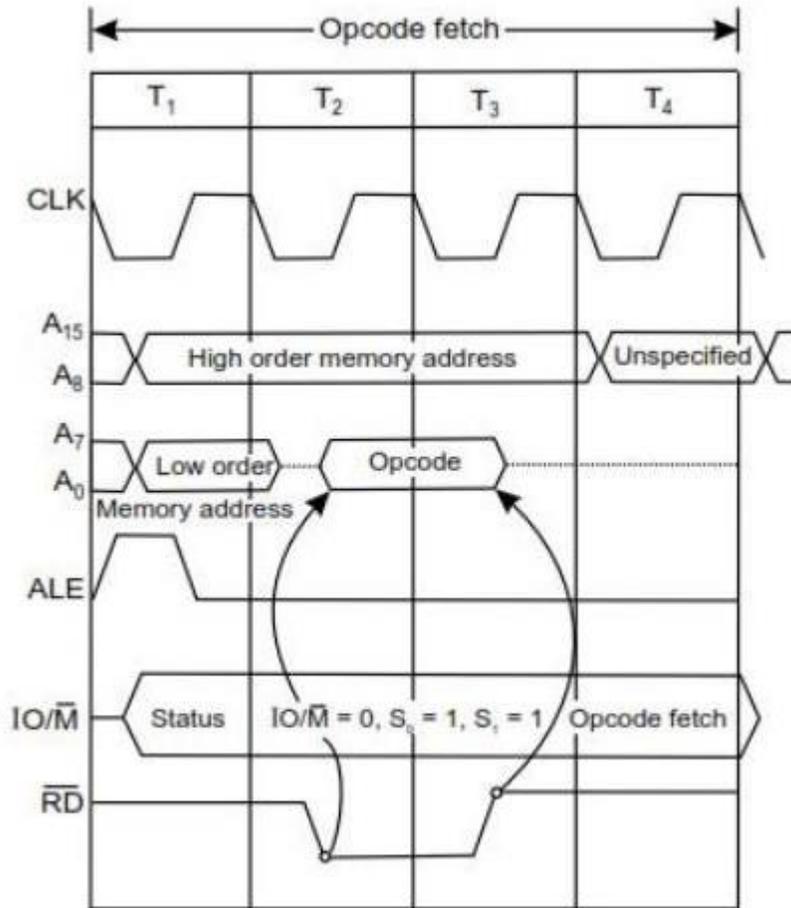


FIG 4.7: OPCODE FETCH CYCLE

Each instruction of the processor has one byte opcode. The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory. Hence, every instruction starts with opcode fetch machine cycle. The time taken by the processor to execute the opcode fetch cycle is 4T. In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

Memory Read Machine Cycle of 8085: The memory read machine cycle is executed by the processor to read a data byte from memory. The processor takes 3T states to execute this cycle. The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.

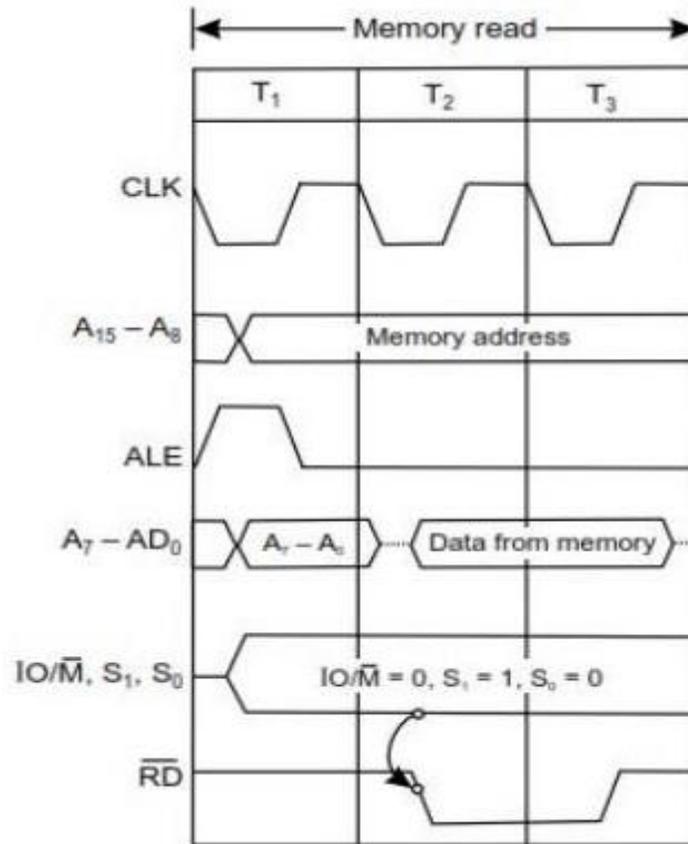


FIG 4.8: MEMORY READ MACHINE CYCLE

Cycle 3. Memory Write Machine Cycle of 8085

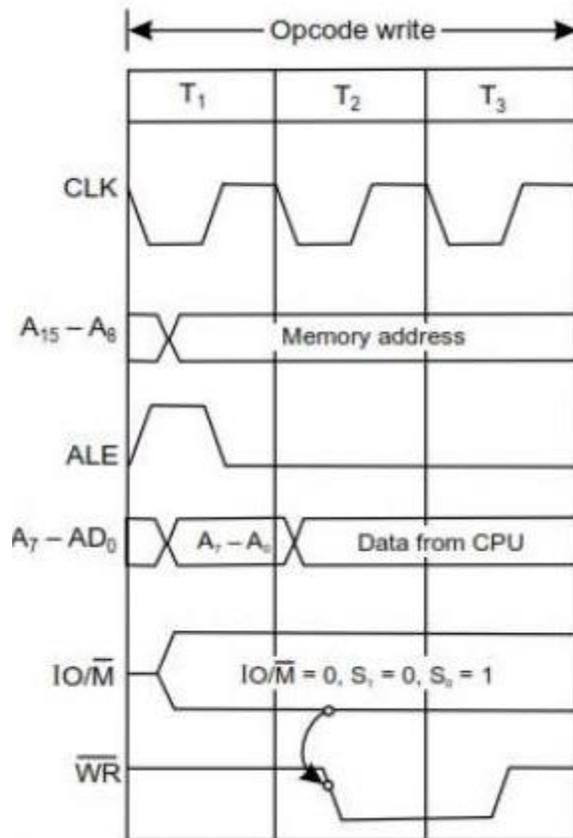


FIG 4.9: MEMORY WRITE MACHINE CYCLE

The memory write machine cycle is executed by the processor to write a data byte in a memory location. The processor takes, 3T states to execute this machine cycle.

I/O Read Cycle of 8085

The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system. The processor takes 3T states to execute this machine cycle. The IN instruction uses this machine cycle during the execution.

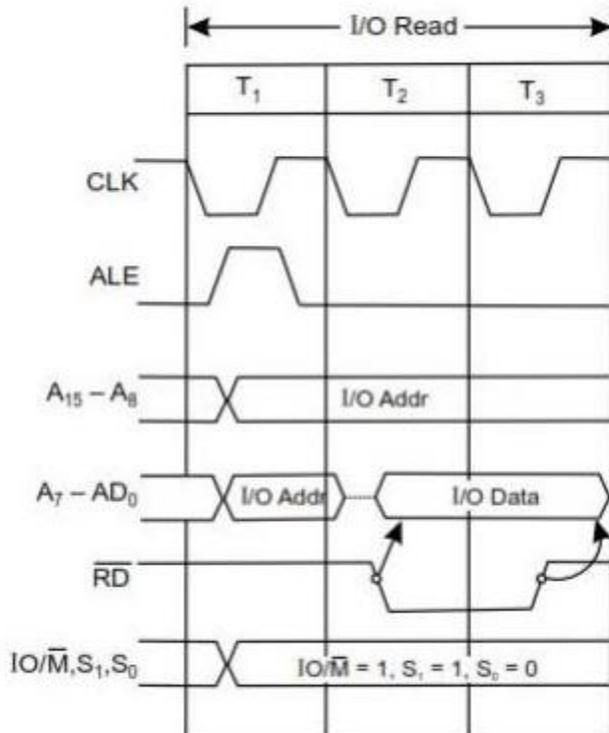


FIG 4.10: I/O READ CYCLE

4.12 SUMMARY

In this unit, you have studied about microprocessor 8085 and its Pin diagram. You learnt in detail working and application of 8085.

4.13 GLOSSARY

Base of a number system: - The number of distinct symbols (digits) used in a number system.

Bits: - Digits in binary number system are called bits.

Byte: - A group of eight bits.

Code: - A system of representation of numeric, alphabets or special characters in a binary form for processing and transmission using digital techniques.

ALU: Arithmetic Logic Unit

4.14 REFERENCES

1. Microprocessor Architecture, Programming and Applications with the 8085 by Ramesh Gaonkar
2. Microprocessor and Microcontroller System By A. P. Godse
3. The 8085 Microprocessor: Architecture, Programming and Interfacing by B. S. Umashankar and K. Udaya Kumar.
4. Advanced Microprocessors and Peripherals by A. K. Ray
5. Online sources

4.15 SUGGESTED READINGS

1. Digital Fundamentals, 10th Ed, Floyd T L, Prentice Hall, 2009.
2. NPTEL
3. You Tube
4. Online tutorial
5. Byju's online study material

4.16 TERMINAL QUESTIONS

4.16.1 Short Answer type

1. Draw pin diagram of 8085.
2. Draw architecture 8085.
3. What do you understand by ALU?
4. Discuss control and timing unit.

UNIT 5

INTERFACING I/O DEVICE

Structure

5.1. Introduction

5.2. Objective

5.3. Basic Interface Concept

5.3.1. Interface

5.3.1.1. Hardware Interfaces

5.3.1.2. Software Interfaces

5.4. Interfacing Output Display

5.4.1. Input Output Interfacing 8085 Microprocessor

5.4.1.1. Input Port

5.4.1.2. Output Port

5.5 Input Output Interfacing Techniques

5.5.1. I/O Device Selection

5.5.2. Interfacing Input Device

5.5.3. Memory Mapped I/O

5.5.3.1. Interfacing Of I/O Port With Memory Mapped I/O

5.6. Comparison Between Memory Mapped I/O And I/O Mapped I/O

5.7. Input Output Transfer Techniques

5.8. Isolated I/O

5.9. Direct Memory Access (Dma)

5.10. Modes Of Operation

5.10.1. Burst Mode

5.10.2. Cycle Stealing Mode

5.10.3. Transparent Mode

5.11. I/O Interface (Interrupt And Dma Mode)

5.11.1. Mode Of Transfer

5.12. Direct Memory Access

5.13. 8253 Programmable Interval Timer

5.13.1. Pin Diagram Of 8253

5.14 Summary

5.15 Glossary

5.16 References

5.17 Suggested Readings

5.18 Terminal Questions

5.18.1 Short Answer Type

5.1. INTRODUCTION

In computing, an interface is a shared boundary across which two or more separate components of computer system exchange information. A standard interface, such as SCSI, decouples the design and introduction of computing hardware, such as I/O devices, from the design and introduction of other components of a computing system, thereby allowing users and manufacturers great flexibility in the implementation of computing systems. Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. The most of the microprocessors support isolated I/O system. It partitions memory from I/O, via software, by having instructions that specifically access (address) memory, and others that specifically access I/O.

As a CPU needs to communicate with the various memory and input-output devices (I/O) as we know data between the processor and these devices flow with the help of the system bus.

The 8253 Programmable Interval Timer/counter specifically designed for use in real-time application for timing and counting function such as binary counting, generation of accurate time delay, generation of square wave, rate generation, hardware/software triggered strobe signal, one-shot signal of desired width, etc.

5.2. OBJECTIVES

After studying this unit, you should be able to-

- Learn about semiconductor memories and their classification based on their operation
- Be familiar with different other types of memories
- Utilize the memory organization and its expansion.
- Know the different storage devices with their applications
- Know about the memory address
- Solve problems based on the memory size, number of address lines, etc.

5.3. BASIC INTERFACE CONCEPT

5.3.1. INTERFACE

In computing, an interface is a shared boundary across which two or more separate components of computer system exchange information. The exchange can be between software, computer

hardware, peripheral devices, humans, and combinations of these. Some computer hardware devices, such as a touch screen, can both send and receive data through the interface, while others such as a mouse or microphone may only provide an interface to send data to a given system.

5.3.1.1. HARDWARE INTERFACES

Hardware interfaces exist in many components, such as the various buses, storage devices, other I/O devices, etc. A hardware interface is described by the mechanical, electrical, and logical signals at the interface and the protocol for sequencing them (sometimes called signaling). A standard interface, such as SCSI, decouples the design and introduction of computing hardware, such as I/O devices, from the design and introduction of other components of a computing system, thereby allowing users and manufacturer's great flexibility in the implementation of computing systems. Hardware interfaces can be parallel with several electrical connections carrying parts of the data simultaneously or serial where data are sent one bit at a time.

5.3.1.2. SOFTWARE INTERFACES

A software interface may refer to a wide range of different types of interface at different "levels": an operating system may interface with pieces of hardware. Applications or programs running on the operating system may need to interact via data streams, filters, and pipelines and in object oriented programs; objects within an application may need to interact via methods.

5.4. INTERFACING OUTPUT DISPLAY

5.4.1. INPUT OUTPUT INTERFACING 8085 MICROPROCESSOR

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. User can give information to the microprocessor using keyboard and user can see the result or output information from the microprocessor with the help of display device. The transfer of data between keyboard and microprocessor, and microprocessor and display device is called Input Output Interfacing 8085 Microprocessor or I/O data transfer. This data transfer is done with the help of I/O ports.

5.4.1.1. INPUT PORT

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer as shown in the Fig. 1. This buffer is a tri-state buffer and its output is available only when enable signal is active.

When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command.

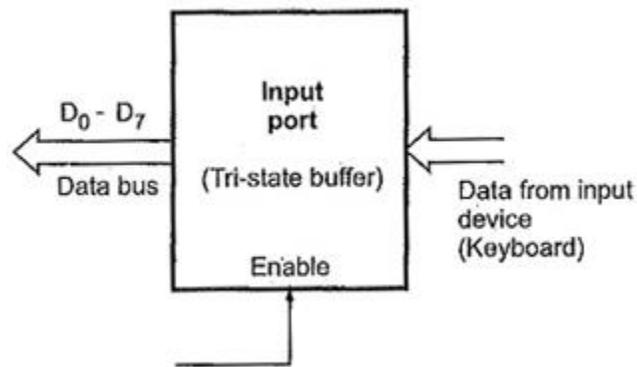


Fig. 5.1.

5.4.1.2. OUTPUT PORT

It is used to send data to the output device such as display from the microprocessor. The simplest form of output port is a latch. The output device is connected to the microprocessor through latch as shown in the **Fig.5. 2.**

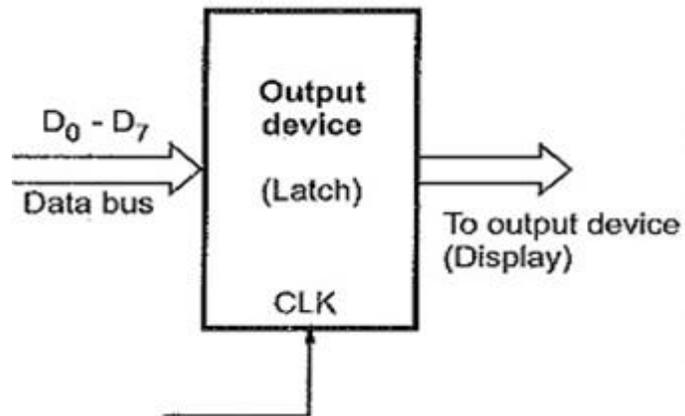


Fig.5.2

When microprocessor wants to send data to the output device, it puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.

5.5 INPUT OUTPUT INTERFACING TECHNIQUES

The most of the microprocessors support isolated I/O system. It partitions memory from I/O, via software, by having instructions that specifically access (address) memory, and others that specifically access I/O. When these instructions are decoded by the microprocessor, an appropriate control signal is generated to activate either memory or I/O operation. In 8085, IO/M signal is used for this purpose. The 8085 outputs a logic '1' on the IO/M line for an I/O operation and a logic '0' for memory, operation. In 8085, it is possible to connect 64 Kbyte memory and 256 I/O ports in the system since 8085 sends 16 bit address for memory and 8 bit address for I/O. I/O devices can be Input Output Interfacing Techniques to an 8085A system in two ways:

I/O Mapped I/O

Memory mapped I/O

In I/O mapped I/O, the 8085 uses IO/M signal to distinguish between I/O read/write and memory read/write operations. The 8085 has separate instructions IN and OUT for I/O data transfer. When 8085 executes IN or OUT instruction, it places device address (port number) on the demultiplexed

low order address bus as well as the high order address bus. In other words, we can say that higher order address bus duplicates the contents of demultiplexed low-order address bus, when 8085 microprocessor executes an IN or OUT instruction. For example, if the device address is 60H then the contents on A₁₅ to A₀ will be as follows:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0

Here, A₈ follows A₀, A₉ follows A₁ and so on, as shown below.

A7	A6	A5	A4	A3	A2	A1	A0	Device Address
A15	A14	A13	A12	A11	A10	A9	A8	
0	1	1	0	0	0	0	0	60H

The instruction IN inputs data from an input device (such as keyboard) into the accumulator and the instruction OUT send the contents of the accumulator to an output device such as LED display. These are two byte instructions. The second byte of the instruction specifies the address or the port number of an I/O device. As it is a byte, the address or port number can be any of the 256 combinations of eight bits, from 00H to FFH. Therefore, the 8085 can communicate with 256 different I/O devices. When we want to Input Output Interfacing Techniques, it is necessary to assign a device address or a port number.

5.5.1. I/O Device Selection

As mentioned earlier, the 8085 gives 8 bit I/O address. This means it can select one of the 256 I/O ports. To select an appropriate I/O device, it is necessary to do following things.

1. Decode the address to generate unique signal corresponding to the device address on the bus.
2. When device address signal and control signal (IOR or IOW) both are low, generate device select signal.
3. Use device select signal to activate the Input Output Interfacing Techniques.

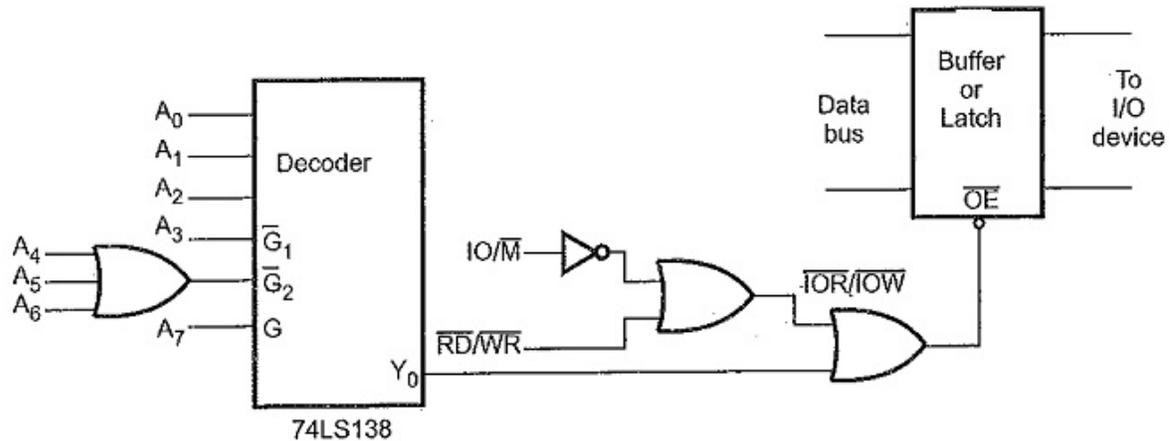


Fig.5.3: Absolute decoding circuit for I/O Devices

5.5.2. INTERFACING INPUT DEVICE

The microprocessor 8085 accepts 8 bit data from the input device such as keyboard, sensors, transducers etc. Fig. 4 shows the circuit diagram to Input Output Interfacing Techniques (buffer) which is used to read the status of 8 switches. The address for this input device is 80H as device select signal goes low when address is 80H.

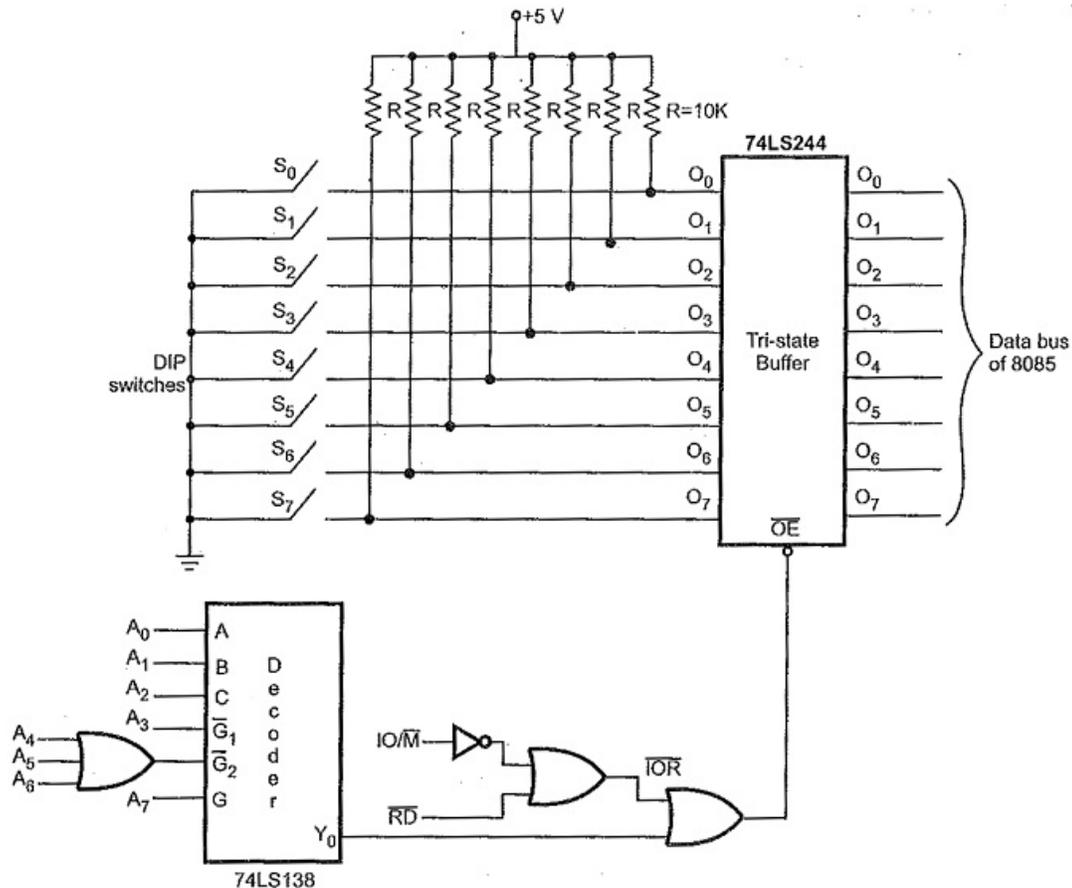


Fig.5.4: Circuit Diagram to Interface Input Port

5.5.3. MEMORY MAPPED I/O

In memory mapped I/O, the I/O devices are assigned and identified by 16 bit addresses. The memory related instructions transfer the data between an I/O device and the microprocessor, as long as I/O port is assigned to the memory address space rather than to the I/O address space. The register associated with the I/O port is simply treated as a memory location. Thus I/O device becomes a part of the system's memory map and hence its name. In memory-mapped I/O every instruction that refers to a memory location can control I/O. The source and destination of the data is limited with I/O mapped I/O, since for an IN instruction the destination register is always the accumulator, and for the OUT instruction the source register is always the accumulator. However, for memory mapped I/O there are number of sources and destinations.

5.5.3.1. Interfacing of I/O port with memory mapped I/O

In memory mapped I/O, MEMR (memory read) and MEMW (memory write) control signals are required to control the data transfer between I/O device and microprocessor. As 8085 gives 16 bit memory address, it is necessary to decode 16 bit memory address to generate device select signal in case of memory mapped I/O.

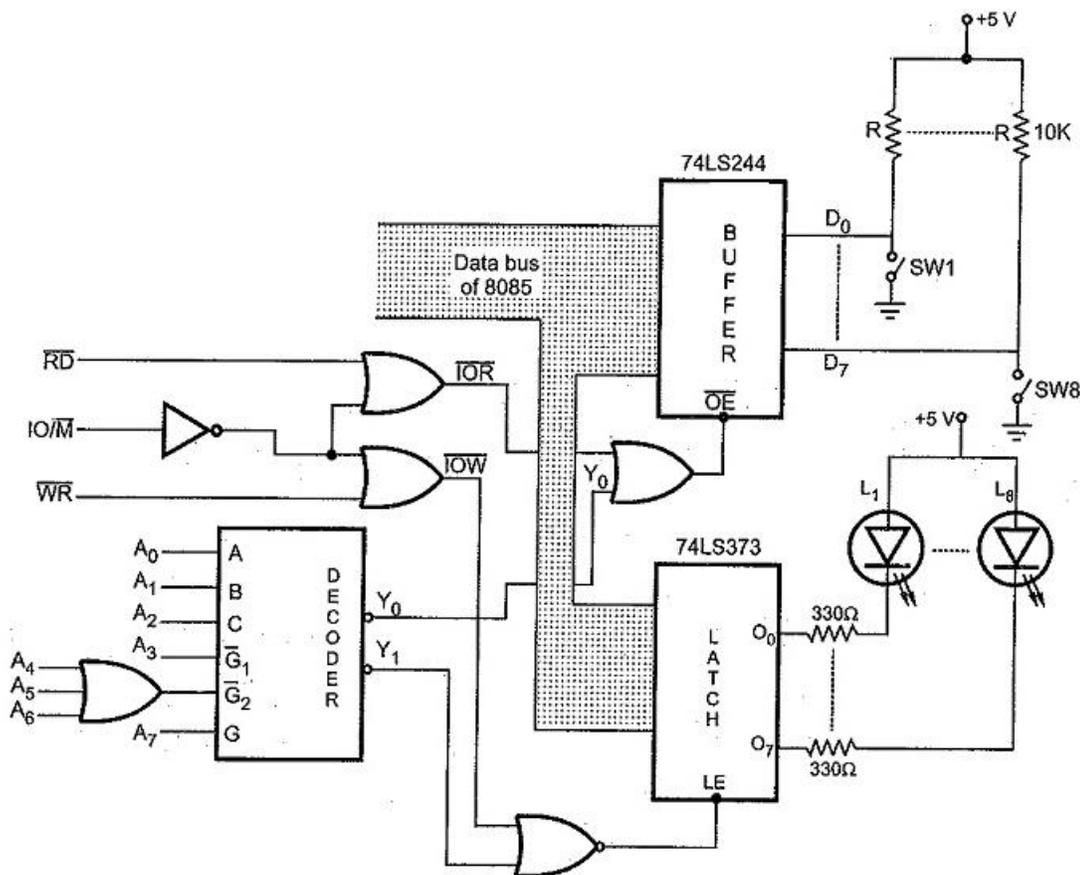


Fig.5.5: Interfacing of I/O port with memory mapped I/O

5.6. Comparison between Memory Mapped I/O and I/O Mapped I/O

Memory mapped I/O and mapped i/o have following differences:

Memory mapped I/O	I/O mapped I/O
Maximum number of I/O devices are 65536	Maximum number of I/O devices are 256
Decoding 16 bit address may require more hardware	Decoding 8 bit address will require less hardware
Execution speed is 13 T state	Execution speed is 10 T state
Data transfer is between any register and I/O device.	Data transfer is between accumulator and I/O device.

5.7. Input Output Transfer Techniques

In Input Output Transfer Techniques, the system requires the transfer of data between external circuitry and the microprocessor. In this section, we will discuss different ways of I/O transfer.

1. Program controlled I/O or Polling control .
2. Interrupt program controlled I/O or interrupt driven I/O.
3. Hardware controlled I/O.
4. I/O controlled by Handshake signals.
5. I/O controlled by ready signal.

1. Program controlled I/O or Polling control:

In program controlled I/O, the transfer of data is completely under the control of the microprocessor program. This means that the data transfer takes place only when an input output transfer techniques instructions executed. In most of the cases it is necessary to check whether the

device is ready for data transfer or not. To check this, microprocessor polls the status bit associated with the I/O device.

2. Interrupt program controlled I/O or Interrupt driven I/O:

In interrupt program controlled approach, when a peripheral is ready to transfer data, it sends an interrupt signal to the microprocessor. This indicates that the Input Output Transfer Techniques is initiated by the external I/O device. When interrupted, the microprocessor stops the execution of the program and transfers the program control to an interrupt service routine. This interrupt service routine performs the data transfer. After the data transfer, it returns control to the main program at the point it was interrupted.

3. Hardware controlled I/O:

To increase the speed of data transfer between memory and I/O, the hardware controlled I/O is used. It is commonly referred to as direct memory access (DMA). The hardware which controls this data transfer is commonly known as DMA controller. The DMA controller sends a HOLD signal, to the microprocessor to initiate data transfer. In response to HOLD signal microprocessor releases its data, address and control buses to the DMA controller. Then the data transfer is controlled at high speed by the DMA controller without the intervention of the microprocessor. After data transfer, DMA controller sends low on the HOLD pin, which gives the control of data, address, and control buses back to the microprocessor. This type of data transfer is used for large data transfers. This technique is explained in more details in chapter 8.

4. I/O Control by handshake signals:

The handshake signals are used to ensure the readiness of the I/O device and to synchronize the timing of the data transfer. In this data transfer, the status of handshaking signals is checked between the microprocessor and an I/O device and when both are ready, the actual data is transferred.

5. I/O control by READY signal:

This technique is used to transfer data between slower I/O device and the microprocessor. In some applications, speed of I/O system is not compatible with the microprocessor's timings. This means that it takes longer time to read/write data. In such situations, the microprocessor has to confirm whether a peripheral is ready to transfer data or not. If READY pin is high, the peripheral is ready otherwise 8085 enters WAIT State or WAIT States. These WAIT states elongate the read/write cycle timings and prepare 8085 microprocessor to communicate with slower I/O devices

5.8. ISOLATED I/O

As a CPU needs to communicate with the various memory and input-output devices (I/O) as we know data between the processor and these devices flow with the help of the system bus. There are three ways in which system bus can be allotted to them:

1. Separate set of address, control and data bus to I/O and memory.
2. Have common bus (data and address) for I/O and memory but separate control lines.
3. Have common bus (data, address, and control) for I/O and memory.

In first case it is simple because both have different set of address space and instruction but require more buses.

Isolated I/O –

Then we have Isolated I/O in which we Have common bus (data and address) for I/O and memory but separate read and write control lines for I/O. So when CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O. As the address space of memory and I/O is isolated and the name is so. The address for I/O here is called ports. Here we have different read-write instruction for both I/O and memory.

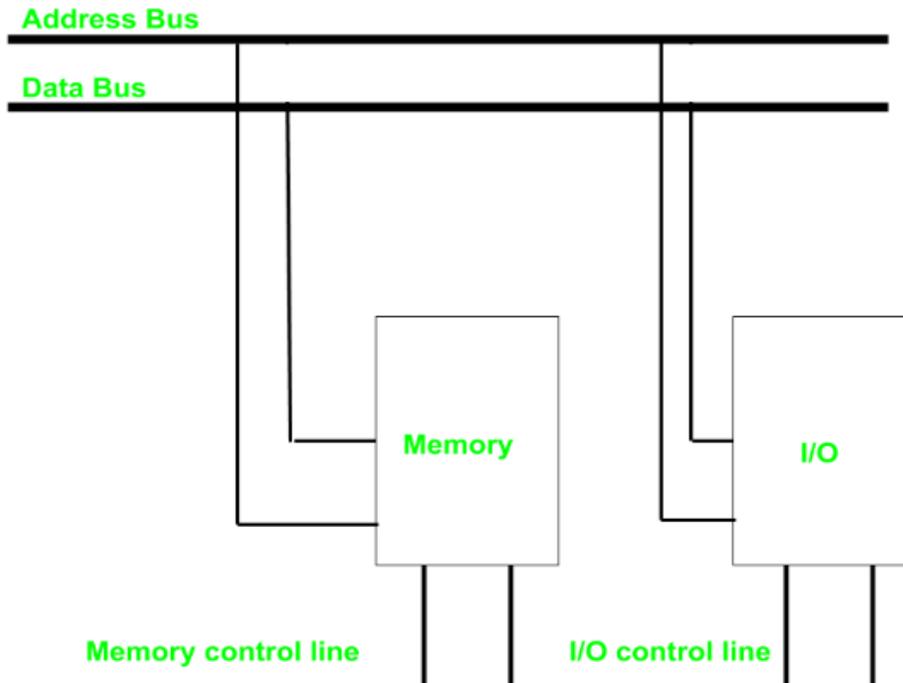


Fig.5.6: Isolated I/O

5.9. DIRECT MEMORY ACCESS (DMA)

Direct memory access (DMA) is a feature of computer systems and allows certain hardware subsystems to access main system memory independently of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller (DMAC) when the operation is done.[citation needed] This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing

circuitry inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or scatter gather operations, from the CPU to a dedicated DMA engine. An implementation example is the I/O Acceleration Technology. DMA is of interest in network-on-chip and in-memory computing architectures.

Standard DMA, also called third-party DMA, uses a DMA controller. A DMA controller can generate memory addresses and initiate memory read or write cycles. It contains several hardware registers that can be written and read by the CPU. These include a memory address register, a byte count register, and one or more control registers. Depending on what features the DMA controller provides, these control registers might specify some combination of the source, the destination, the direction of the transfer (reading from the I/O device or writing to the I/O device), the size of the transfer unit, and/or the number of bytes to transfer in one burst

5.10. MODES OF OPERATION

5.10.1. BURST MODE

In burst mode, an entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time. The mode is also called "Block Transfer Mode".

5.10.2. CYCLE STEALING MODE

The cycle stealing mode is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using BR (Bus Request) and BG (Bus Grant) signals, which are the two signals controlling the interface between the CPU and the DMA controller. However, in cycle stealing mode, after one unit (e.g. byte) of data transfer, the control of the system bus is deasserted to the CPU via BG. It is then continually requested again via BR, transferring one unit (e.g. byte) of data per request, until the entire block of data has been

transferred. By continually obtaining and releasing the control of the system bus, the DMA controller essentially interleaves instruction and data transfers. The CPU processes an instruction, then the DMA controller transfers one data value, and so on. Data is not transferred as quickly, but CPU is not idled for as long as in burst mode. Cycle stealing mode is useful for controllers that monitor data in real time.

5.10.3. TRANSPARENT MODE

Transparent mode takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance. In transparent mode, the DMA controller transfers data only when the CPU is performing operations that do not use the system buses. The primary advantage of transparent mode is that the CPU never stops executing its programs and the DMA transfer is free in terms of time, while the disadvantage is that the hardware needs to determine when the CPU is not using the system buses, which can be complex. This is also called "Hidden DMA data transfer mode".

5.11. I/O Interface (Interrupt and DMA Mode)

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

5.11.1. MODE OF TRANSFER

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.

2. Interrupt- initiated I/O.
3. Direct memory access (DMA).

Now let's discuss each mode one by one.

Programmed I/O: It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

Interrupt- initiated I/O: Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

5.12. DIRECT MEMORY ACCESS

The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has

no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

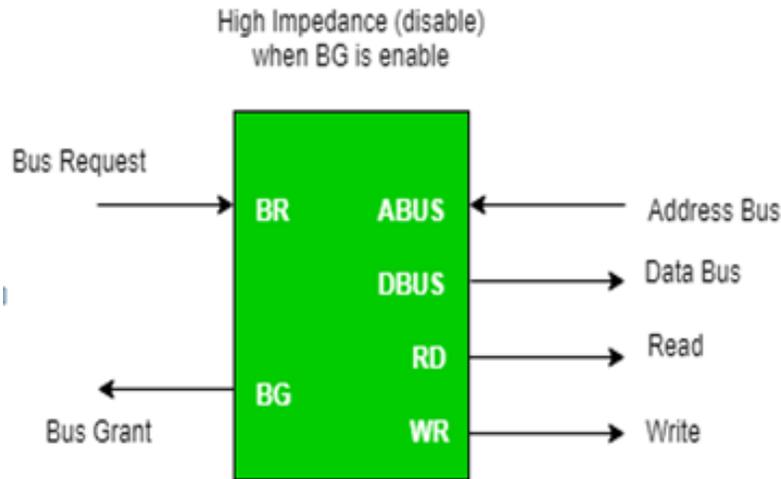


Fig.5.7: CPU Bus Signals for DMA Transfer

Bus Request: It is used by the DMA controller to request the CPU to relinquish the control of the buses.

Bus Grant: It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

5.13. 8253 PROGRAMMABLE INTERVAL TIMER

In the process control system or automation industry, a number of operations are generally performed sequentially. Between two operations, a fixed time delay is specified. In a microprocessor-based system, time delay can be generated using software. Sequences of operations are also performed based on software. Therefore time delay, sequence and counting can be done under the control of a microprocessor. These most common problems can be solved using the 8253 in any microcomputer system.

The 8253 Programmable Interval Timer/counter specifically designed for use in real-time application for timing and counting function such as binary counting, generation of accurate time delay, generation of square wave, rate generation, hardware/software triggered strobe signal, one-

shot signal of desired width, etc. The function of 8253 timer is that of a general purpose, multi-timing element which can be treated as an array of I/O ports in the system software.

The generation of accurate time delay using software control or writing instruction is possible. But instead of writing instructions for time delay loop, the 8253 timer may be used for this. The programmer configures the 8253 as per requirements. When the counters of the 8253 are initializing with the desired control word, the counter operates as per requirement. Then a command is given to the 8253 to count out the delay and interrupt the CPU. At the instant it has completed its tasks, the output will be obtained from the output terminal. Multiple delays can easily be implemented by assignment of priority levels in the microprocessor.

The counter/timer can also be used for non-delay in nature such as Programmable Rate Generator, Event Counter, Binary Rate Multiplier, Real Time Clock, Digital One-Shot, and Complex Motor Controller. The 8253 operates in the frequency range of dc to 2.6 MHz while the 8253 uses NMOS technology. The 8253 is compatible to the 8085 microprocessor. Generally, 8253 Programmable Interval Timer can be operating in the following modes.

- Mode 0 Interrupt on terminal count
- Mode 1 Programmable one-shot
- Mode 2 Rate generator
- Mode 3 Square-wave generator
- Mode 4 Software triggered mode
- Mode 5 Hardware triggered mode

5.13.1. PIN DIAGRAM OF 8253

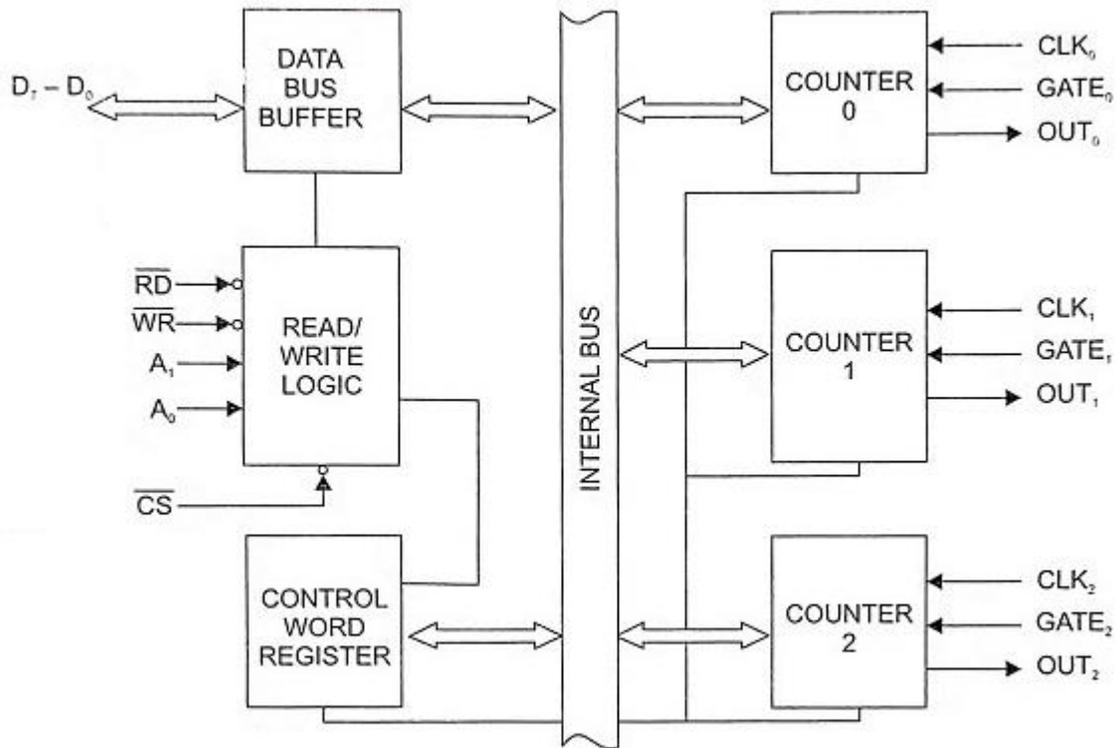


Fig.5.8: Schematic block diagram of Intel 8253 timer/counter

The 8253 timer is a 24-pin IC and operates at +5 V dc. It consists of three independent programmable 16-bit counters: Counter 0, Counter 1, and Counter 2. Each counter operates as a 16-bit down counter and each counter consists of clock input, gate input and output as depicted in Fig. 8. The schematic block diagram is given in Fig. 8. The gate input is used to enable the counting process. Therefore the starting of counting may be controlled by external input pulse in gate terminal. After gate triggered, the counter starts count down. When the counter has completed counting, output signal would be available at the out terminal.

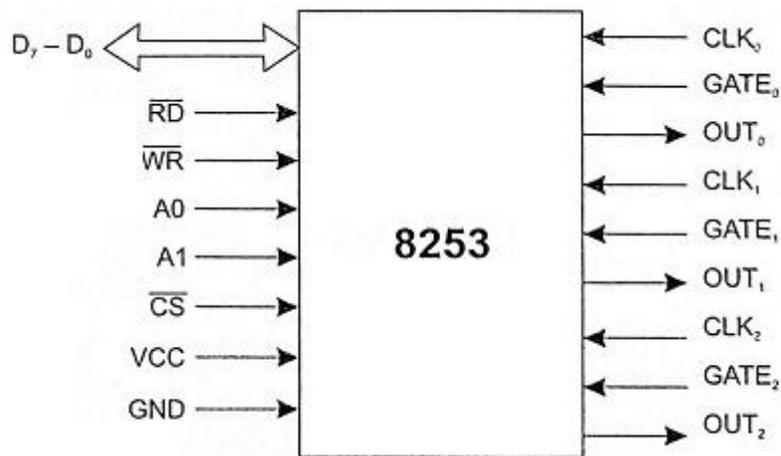


Fig.5.9: Pin diagram of 8253

The programmer can program 8253 using software in any one of the six operating modes: Mode 0, Mode 1, Mode 2, Mode 3, Mode 4, and Mode 5. The pin diagram of 8253 Programmable Interval Timer is shown in Fig 5. 9 and Fig. 10. The functional descriptions of pins are as follows:

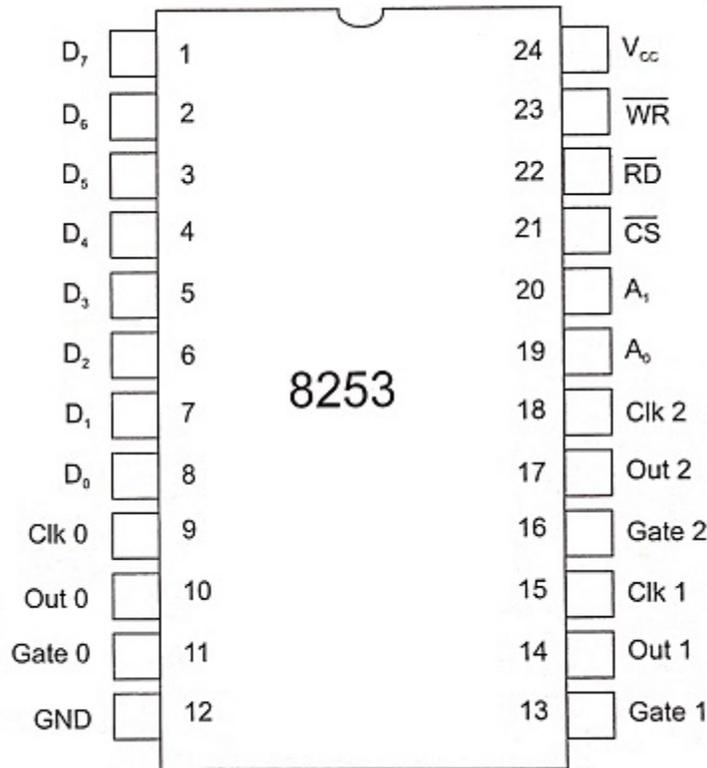


Fig.5.10: Pin diagram of 8253

\overline{RD} (Read) When this pin is low, the CPU is inputting data in the counter.

\overline{WR} (Write) When this is low, the CPU is outputting data in the form of mode information or loading of counters.

A_0, A_1 These pins are normally connected to the address bus. The function of these pins is used to select one of the three counters to be operated and to address the control word registers for mode selection as given below:

A1	A0	Selection of Counters and Control word register
0	0	Counter 0
0	1	Counter 1
1	0	Counter 1
1	1	Control Word Register

\overline{CS} Chip select A ‘low’ on \overline{CS} input enables the 8253. No reading or writing operation will be performed until the device is selected. The \overline{CS} input signal is not used to control the actual operation of the counters.

Data Bus Buffer The 3-state, bi-directional, 8-bit buffers exist in 8253. These buffers are used to interface the 8253 to the systems data bus D₀-D₇ lines. Data can be transmitted or received by the buffer upon execution of input and output CPU instructions. The data bus buffer has three basic functions, namely, programming the Modes of the 8253, loading the count registers and reading the count values.

D₀-D₇ Bi-directional Data Bus There are eight data lines through which the control word will be written in the control word register of 8253 counter/timer during programming. The counter will be written and read through the data bus.

Read/Write Logic The read/write Logic accepts inputs from the system bus and in turn Gate generates control signals for operation of 8253. This is enabled by \overline{CS} . Therefore, no operation can take place to change the function unless the device has been selected by the system logic. Table shows the various functions of 8253 Programmable Interval Timer based on the status of pins associated with read/write logic.

Function of 8253

\overline{CS}	\overline{RD}	\overline{WR}	A1	A0	Function
0	1	0	0	0	Load Counter 0
0	1	0	0	1	Load Counter 1
0	1	0	1	0	Load Counter 2
0	1	0	1	1	Write Mode word
0	0	1	0	0	Read Counter 0
0	0	1	0	1	Read Counter 1
0	0	1	1	0	Read Counter 2
0	0	1	1	1	No operation 3 State
1	X	X	X	X	Disable 3 State
0	1	1	X	X	No operation 3 State

CLK₀, CLK₁, CLK₂ CLK₀, CLK₁ and CLK₂ are clock for Counter 0, Counter 1 and Counter 2 respectively. The countdown of the counter takes place on each high to low transition of clock input.

GATE₀, GATE₁, GATE₂ GATE₀, GATE₁ and GATE₂ are gate terminals of Counter 0, Counter 1 and Counter 2 respectively. The function of the GATE in different modes is illustrated in Table

Different mode of operation corresponding to gate signal

Single status mode	Low of going law	Rising	High
0	Disable counting	-	Enable counting
1	-	Initiating counting	-
2	Disable counting	Initiating counting	Enable counting
3	Disable counting	Initiating counting	Enable counting
4	Disable counting	-	Enable counting
5	-	Initiating counting	-

OUT₀, OUT₁, OUT₂ OUT₀, OUT₁, OUT₂ are output terminals of Counter 0, Counter 1 and Counter 2 respectively. The output of the 8253 timer depends Upon the mode of operation.

5.14 SUMMARY

In this unit, you have studied about microprocessor 8085 and its Pin diagram. You learnt in detail working and application of 8085.

5.15 GLOSSARY

I/O: - Input/Output

IC: Integrated Circuit

5.16 REFERENCES

1. Microprocessor Architecture, Programming and Applications with the 8085 by Ramesh Gaonkar

2. Microprocessor and Microcontroller System By A. P. Godse
3. The 8085 Microprocessor: Architecture, Programming and Interfacing by B. S. Umashankar and K. Udaya Kumar.
4. Advanced Microprocessors and Peripherals by A. K. Ray
5. Online sources

5.17 SUGGESTED READINGS

1. Digital Fundamentals, 10th Ed, Floyd T L, Prentice Hall, 2009.
2. NPTL
3. You Tube
4. Online tutorial
5. Byju's online study material

5.18 TERMINAL QUESTIONS

5.18.1 Short Answer type

1. Compare between Memory Mapped I/O And I/O Mapped I/O?
2. What do you understand by Input Output Transfer Techniques?
3. What do understand by Modes Of Operation? Discuss about Burst Mode, Cycle Stealing Mode and Transparent Mode?
- 4, What do you understand Direct Memory Access?
5. Draw Pin Diagram of 8253.

UNIT 6: 8085 MICROPROCESSORS PROGRAMMING

6.1 Introduction

6.2 Objectives

6.3 The 8085 Programming Model

6.3.1. General Purpose Registers

6.3.2. Specific Purpose Registers

6.3.3 Memory Registers

6.4. Instruction Set Classification

6.4.1 Data Transfer Operation

6.4.2. Arithmetic Operations

6.4.3. Logical Operations

6.4.4. Branching Operations

6.3.5. Machine Control Operations

6.5. Instruction Format

6.5.1 How an Instruction Is Formed

6.5.2 Instruction Word Size

6.6. Introduction To Addressing Modes

6.6.1 Immediate Addressing

6.6.2. Register Addressing

6.6.3. Direct Addressing

6.6.4. Register Indirect Addressing

6.6.5. Implicit Addressing

6.7. Instruction Set

6.8. Example

6.9. Summary

6.10. Questions

6.1 INTRODUCTION

A microprocessor is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory accepts binary data as input and processes data according to those instructions and provides results as output. The input and output of microprocessor is either memory or any input-output device.

Microprocessor can be described with its internal architecture and programming model. Internal architecture specifies the hardware structure which has been studied in the previous units. The programming model is writing an assembly language program using mnemonics for a specific task. An assembly language program is a set of instructions. To write a program, we must be familiar with the instructions set available with the microprocessor and hardware parts which can be accessed. This unit describes the programming model for 8085.

6.2 OBJECTIVES

After studying this unit, you should be able to

- Explain the function of the registers in 8085 programming model
- Description of flags and their role
- Define and explain the addressing mode
- Describe instruction and instruction set
- Classification of instruction set
- Explain the functions of Data transfer instructions
- Describe the functions of Machine control instructions
- To write some program in assembly language

6.3 THE 8085 PROGRAMMING MODEL

The 8085-programming model includes six registers, one accumulator, and one flag register. In addition, it also includes two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows:

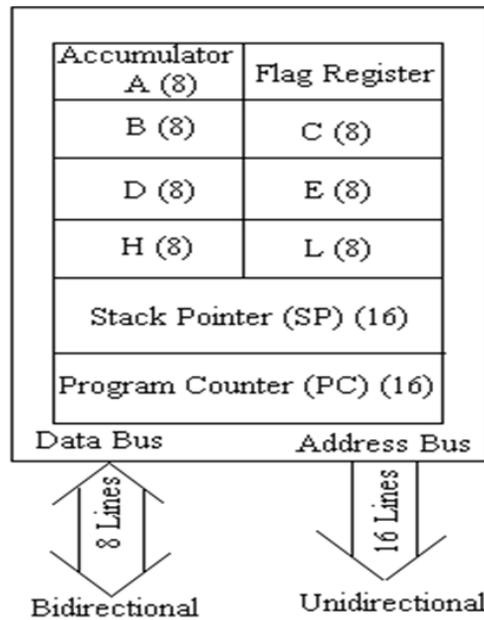


Figure 6.1: Programming Model of 8085

6.3.1 GENERAL PURPOSE REGISTERS

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H, and L. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. These registers are used to store or copy temporary data, by using instructions, during the execution of the program.

6.3.2 SPECIFIC PURPOSE REGISTERS

Accumulator:

The accumulator is an 8-bit register (can store 8-bit data) that is the part of the arithmetic and logical unit (ALU). After performing arithmetical or logical operations, the result is stored in accumulator. Accumulator is also defined as register A.

Flag register:

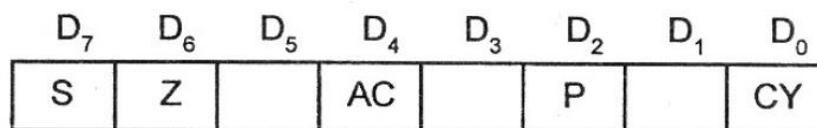


Figure 6.2: Flag register

The flag register in ALU is 8 bit register whose only 5 bit positions are used. These bits are set (1) or reset (0) after an arithmetic and logical operation performed in ALU, according to data conditions of the result in the accumulator and other registers.

They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; their bit positions in the flag register are shown in Figure 2.

The most commonly used flags are Zero, Carry, and Sign. This flag is used internally for BCD (Binary-Coded decimal Number) operations. The microprocessor uses these flags to test data conditions.

Carry Flag (CY) Bit position B₀	: It is set to 1, if an arithmetic operation results in a Carry/ Borrow otherwise reset.
Parity Flag (P) Bit position B₂	: It is set to 1, if the result has an even number of 1s, otherwise it is reset
Auxiliary Carry Flag (AC) Bit position B₄	: It is set to 1, if a carry is generated by digit D ₃ and is passed to digit D ₄ . Used only to perform BCD operations
Zero Flag (Z) Bit position B₆	: It is set to 1 when the result is zero, otherwise it is reset. It helps in determining, if two numbers are equal or not.
Sign Flag (S) Bit position B₇	: It is set to 1, if bit D ₇ of the result is 1 (the stored number is negative), otherwise it is reset. S flags signify the sign (Positive/Negative) of stored number.

6.3.3 MEMORY REGISTERS

There are two 16-bit registers used to hold memory addresses. The size of these registers is 16 bits because the memory addresses are 16 bits. They are:-

Program Counter: This register is used to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer: The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in read/write memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

6.4 INSTRUCTION SET CLASSIFICATION

The 8085 Instruction classification can be categorized into five different groups based on the nature of function of the instructions.

- **Data transfer operations;**
- **Arithmetic operations;**
- **Logical operations;**
- **Branching operations; and**
- **Machine-control operations.**

6.4.1 DATA TRANSFER OPERATION

The data transfer instructions, copies data from a location called a source to another location called a destination. These location are either register or memory location. The content of source remains same. These instructions do not affect the flag register of the processor. These operations can be broadly classified in four types:

Types	Example
Register to Register	MOV H, L; copies the contents of Reg L to Reg H
Specific data to Register / Memory location	MVI B, 55H; copies data 55H to Reg B
Between Register to Memory Location	MOV B, M; copies the contents of Memory location by SP to Reg B
Between an I/O Device and the Accumulator	IN 04H, copies the data at port 04H to accumulator

6.4.2 ARITHMETIC OPERATIONS

The arithmetic instructions perform mathematical operations such as addition, subtraction, increment, and decrement on the data in registers or memory.

- **Addition:** Any 8-bit number, or the contents of a register, or the contents of a memory location can be added to the contents of the accumulator and the resulted sum is stored in the accumulator.
In 8085, no two other registers can be added directly, i.e. the contents of B and C registers cannot be added directly. To add two 16-bit numbers the 8085 provides DAD instruction. It adds the data within the register pair to the contents of the HL register pair and resulted sum is stored in the HL register pair.
- **Subtraction:** Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the result is stored in the accumulator. The resulted borrow bit is stored in the carry flag:

- **Increment/Decrement** : The 8085 has the increment and decrement instructions to increment and decrement the contents of any register, memory location or register pair by 1.

6.4.3 LOGICAL OPERATIONS

The logical instructions provided by 8085 perform logical, rotate, compare and complement operations with the contents of the accumulator.

- **Logical** - Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, Ored, or Exclusive-ORed with the contents of the accumulator. The results are stored in the accumulator.
- **Rotate**- Each bit in the accumulator can be shifted either left or right to the next position.
- **Compare**- Any 8-bit number, or the contents of a register, or a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.
- **Complement** - The contents of the accumulator can be complemented. All 0s are replaced by 1s and all 1s are replaced by 0s.

6.4.4. Branching Operations

This group of instructions alters the sequence of the program execution, either unconditionally or under certain test conditions. These instructions include branch instructions, subroutine call and return instructions and restart instructions.

Jump - Conditional jumps are an important aspect of the decision-making process in the programming. These instructions test for a certain conditions (e.g., Zero or Carry flag) and alter the program sequence when the condition is met. In addition, the instruction set includes an instruction called *unconditional jump*.

Call, Return, and Restart - These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine. The conditional Call and Return instructions also can test condition flags.

6.4.5. Machine Control Operations

These instructions control machine functions such as Halt, Interrupt, or do nothing.

6.5. INSTRUCTION FORMAT

An instruction is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: opcode and operand

Opcode : The operation code (OPCODE) field in the instruction specifies the operation to be performed. This is a specified binary code.

Operand: The data on which operation is to be performed, called the **operand**. The operand is specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

Internal Registers, Memory Location, and Register pair have their specific code. The Codes are as follows:

Code	Register	Code	Register pair
000	B	00	BC
001	C	01	DE
010	D	10	HL
011	E	11	AF
100	H		
101	L		
111	A		
110	Memory		

6.5.1 How an instruction is formed

- Task: perform the addition of data stored in Register B with Accumulator and result in Accumulator.

Here, Opcode is Addition and Operand are Reg B and A. Opcode assigned for Addition in 8085 is 10000. This is first part of Instruction.

Operand is B and A which have codes 000 and 111, respectively. In 8085, to perform most of Arithmetic and Logical operations, Accumulator is set as default register. So, operand for A will be implicit and code of B is operand.

Combine Opcode (10000) and Operand (000) which is 1000 0000 (80H in hexadecimal). Instruction formed to Addition is ADD B (Hex Code 80H.)

6.5.2 Instruction word size

The Instruction Format of 8085 is classified on the basis of instruction length as of one, two and three byte instructions. First part of instruction is opcode which can be from one bit to one byte length. The 8085 instruction set, according to word size are listed below

1. One-word or 1-byte instructions
2. Two-word or 2-byte instructions
3. Three-word or 3-byte instructions

*In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

- **One-Byte Instructions**

A 1-byte instruction share opcode and operand in same one byte. Operand(s) are internal register or memory and are coded into the instruction. For example:

Task	Opcode (binary code)	Operand (binary code)	Instruction in Binary
Add the contents of register B to the contents of the accumulator. (ADD B)	ADD (10000)	Register B (000), A is implicit	10000 000, Hex code 80H
Add the contents of register B to the contents of the accumulator. ADD M	ADD (10000)	Memory M (110) Memory address is implicit	10000 110, Hex code 86H
Copy the contents of the Register E in the Register D. MOV D, E	MOV (01)	Reg D (010), E(011)	01 010 011, Hex code 53H
Invert (compliment) each bit in the accumulator	CMA	Implicit	0010 1111, Hex code2FH

- **Two-Byte Instructions**

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is an 8 bit data byte immediately following the opcode. For example:

Task	Opcode	Operand	Hex Code	
Load an 8-bit data byte in the	MVI	8 bit data	3E	First Byte

accumulator			Data	Second Byte
Add an 8-bit data byte with to contents of accumulator.	ADI	8 bit data	C6 Data	First Byte Second Byte
Perform ANDing an 8-bit data byte with the accumulator	ANI	8 bit data	E6 Data	First Byte Second Byte

- **Three-Byte Instructions**

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address. For Example:

Task	Opcode	Operand	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H memory address	C3 85 20	First byte Second Byte Third Byte
Load an 16-bit data byte in the Register pair (LXI B)	LXI	Register pair, 16-bit data	01 Data Data	First byte Second Byte Third Byte

6.6. INTRODUCTION TO ADDRESSING MODES

Every instruction consists of two parts. That is what the operation is to be performed (called operation code or opcode), and on what this operation is to be performed (called operands). For example, in case of $A + B$, the sign is the opcode, and A & B are the operands.

Each instruction performs an operation on the particular data called operand. An operand should be specified for an instruction to be executed. The operand may be in the general purpose register, accumulator or in a memory location. The way in which the operand is specified in an instruction is called addressing mode. The microprocessor 8085 uses following type of addressing modes:

- Immediate addressing
- Register addressing
- Direct addressing
- Register indirect addressing
- Implicit addressing

6.6.1 Immediate Addressing

In immediate addressing mode, the operands are specified in the instruction itself. The instruction format of instructions with immediate addressing mode is given in the figure below:-

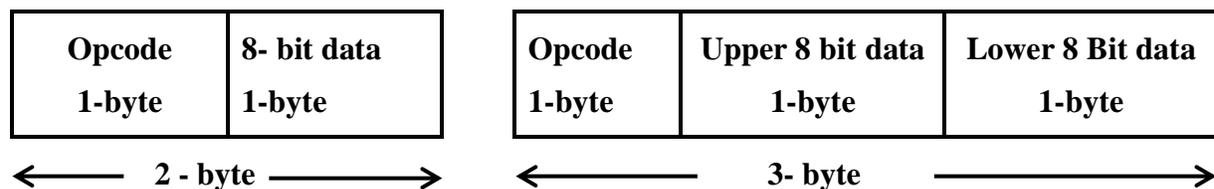


Figure 6.3: Instruction format of the instruction with immediate addressing modes

Examples of immediate addressing mode are –

- MVI A, 05H: Move 05H in the accumulator.
- ADI 06: Add 06H to the contents of the accumulator.
- LXI H, 2500H: Load HL pair with 2500H

6.6.2. Register Addressing

In register addressing mode, the operands are in the general purpose registers. The registers are specified in the instructions. The format of instructions with register addressing mode is given in the figure below:-



Figure 6.4: Instruction format of the instruction with register addressing mode

Most of the instructions that uses register to register addressing deal with 8-bit values. However, some of these instructions deal with 16-bit register pairs. For example, the SPHL instruction moves the contents of the H and L registers to SP.

Examples of register addressing mode are as follows:-

- MOV A, B: This instruction moves the contents of register B to register A or accumulator.
- ADD B: Add the contents of B register to accumulator.

6.6.3. Direct Addressing

In this addressing mode one of the operand is data stored in the memory. The memory address of the data is directly given in the instruction itself. The instruction format of direct addressing mode is in Figure 6.5 below.

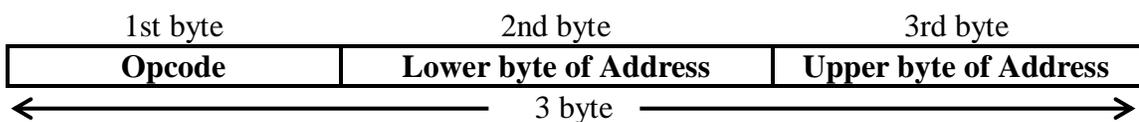


Figure 6.5: Instruction format of the instruction with direct addressing mode

Instructions that include a direct address require three-bytes of storage. The first one for the instruction code, and next two for the 16-bit address in case of memory related operations.

Examples of direct addressing mode are as follows:-

- STA 2500H: Store the content of the accumulator to memory location 2500H.
- IN 02H: Input the data from input port 02H in accumulator.

6.6.4. Register Indirect Addressing

This addressing mode is used only in concern with memory. In register indirect addressing mode, the address of the operand (data) is specified by a register pair. The instruction format of direct addressing mode is given, in the figure 6.6 below:-

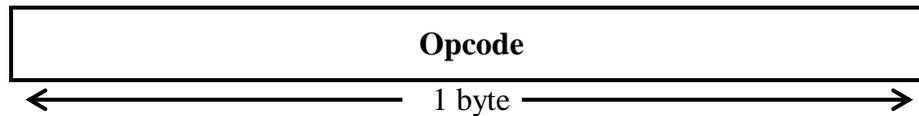


Figure 6.6: Instruction format of the instruction with register indirect addressing mode

Example of indirect addressing mode is as follows:

- LXI H, 2500H: Load HL pair with 2500H.
- MOV A, M: Move the contents of memory to the accumulator.
- HLT: Stop the program.

6.6.5. Implicit Addressing

There are certain instructions which operate on the content or the accumulator directly; these instructions do not require specifying the operands. The instruction format of implicit addressing mode is given in the figure 6.7 below.

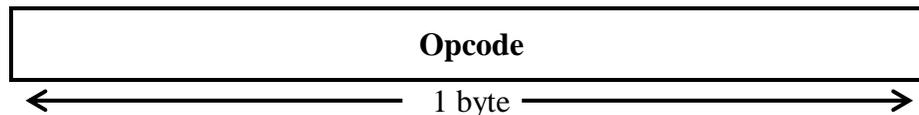


Figure 6.7 : Instruction format of the instruction with implicit addressing mode

Examples of implicit addressing mode are as follows:

- CMA: Compliment the contents of the accumulator.
- RAR: Rotate the contents of the accumulator to the right by one bit.
- RAL: Rotate the contents of the accumulator to the left by one bit.

6.7. INSTRUCTION SET

- **Data Transfer (Copy) Instructions**

Opcode	Operand	operation	Explanation
MOV	Rd, Sc M, Sc Dt, M	Copy from the source (Sc) to the destination (Dt)	This instruction copies the contents of the source register into the destination register without any alteration. Example – MOV H, L
MVI	Rd, data M, data	Move immediate 8-bit data	The 8-bit data is stored in the destination register or memory. Example – MVI B, 55H
LDA	16-bit address	Load the accumulator from memory	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the Accumulator. Example – LDA 2034H
STA	16-bit address	copy the accumulator data to memory	The contents of the Accumulator are copied into the memory location specified by the operand. This is a 3-Byte instruction, the second Byte specifies the low-order address and the third Byte specifies the high-order address. Example – STA AB00H
OUT	8-bit port address	Output the data from the Accumulator to a port with 8 bit address	The contents of the Accumulator are copied into the I/O port specified by the operand. Example – OUT 01H
IN	8-bit port address	Input data to Accumulator from a port with 8-bit address	The contents of the input port designated in the operand are read and loaded into the Accumulator. Example – IN 04H

▪ Arithmetic Instructions

ADD	R M	Add register or memory, to the Accumulator	The contents of the register or memory are added to the contents of the Accumulator and the result is stored in the Accumulator. Example – ADD B.
ADC	R M	Add register to the Accumulator with carry	The contents of the register or memory & the Carry flag are added to the contents of the Accumulator and the result is stored in the Accumulator. Example – ADC B
ADI	8-bit data	Add the immediate to the Accumulator	The 8-bit data is added to the contents of the Accumulator and the result is stored in the Accumulator. Example – ADI 55H

Opcode	Operand	operation	Explanation
ACI	8-bit data	Add the immediate to the Accumulator with carry	The 8-bit data and the Carry flag are added to the contents of the Accumulator and the result is stored in the Accumulator. Example –ACI 55H
SUB	R M	Subtract the register or the memory from the Accumulator	The contents of the register or the memory are subtracted from the contents of the Accumulator, and the result is stored in the Accumulator. Example – SUB B
SBB	R M	Subtract the source and borrow from the Accumulator	The contents of the register or the memory &the Borrow flag are subtracted from the contents of the Accumulator and the result is placed in the Accumulator. Example – SBB B
SUI	8-bit data	Subtract the immediate from the Accumulator	The 8-bit data is subtracted from the contents of the Accumulator & the result is stored in the Accumulator. Example –SUI 55H
SBI	8-bit data	Subtract the immediate from the Accumulator with borrow	The 8-bit data is subtracted to the contents of the Accumulator and the result is stored in the Accumulator. Example –SBI 55H
INR	R M	Increment the register or the memory by 1	The contents of the designated register or the memory are incremented by 1 and their result is stored at the same place. Example – INR B
DCR	R M	Decrement the register or the memory by 1	The contents of the designated register or memory are decremented by 1 and their result is stored at the same place. Example – DCR B

▪ **Logic Instructions**

ANA	R M	Logical AND register or memory with the Accumulator	The contents of the Accumulator are logically AND with the contents of the register or memory, and the result is placed in the Accumulator.
ANI	8-bit data	Logical AND immediate with the Accumulator	The contents of the Accumulator are logically AND with the 8-bit data and the result is placed in the Accumulator.
XRA	R M	Exclusive OR register or memory with the Accumulator	The contents of the Accumulator are Exclusive OR with the contents of the register or memory, and the result is placed in the Accumulator.

Opcode	Operand	operation	Explanation
XRI	8-bit data	Exclusive OR immediate with the Accumulator	The contents of the Accumulator are Exclusive OR with the 8-bit data and the result is placed in the Accumulator.
ORA	R M	Logical OR register or memory with the Accumulator	The contents of the Accumulator are logically OR with the contents of the register or memory, and result is placed in the Accumulator.
ORI	8-bit data	Logical OR immediate with the Accumulator	The contents of the Accumulator are logically OR with the 8-bit data and the result is placed in the Accumulator.
CMA	None	Complement Accumulator	The contents of the Accumulator are complemented. No flags are affected.
CMC	None	Complement carry	The Carry flag is complemented. No other flags are affected.
STC	None	Set Carry	Set Carry

▪ **JUMP Instructions**

JMP	16-bit address	Jump unconditionally	The program sequence is transferred to the memory address given in the operand.
JC	16-bit address	Jump on Carry flag, If CY=1	The program sequence is transferred to the memory address given in the operand based on the Carry flag of the PSW.
JNC	16-bit address	Jump on no Carry, If CY=0	
JP	16-bit address	Jump on Sign Flag, if S=0	The program sequence is transferred to the memory address given in the operand based on the Sign flag of the PSW.
JM	16-bit address	Jump on No Sign Flag, if S=1	
JZ	16-bit address	Jump on zero flag, if Z=1	The program sequence is transferred to the memory address given in the operand if zero flag status is satisfied.
JNZ	16-bit address	Jump on no zero flag Z=0	
JPE	16-bit address	Jump on parity even, if P=1	The program sequence is transferred to the memory address given in the operand if Parity flag condition is satisfied.
JPO	16-bit address	Jump on parity odd, if P=0	

▪ **Machine control instructions**

NOP	None	No operation	No operation is performed, i.e., the instruction is fetched and decoded only.
HLT	None	Halt and enter wait state	The CPU finishes executing the current instruction and stops further execution. An interrupt or reset is necessary to exit from the halt state.

6.8. EXAMPLES:

Example 1: It is desired to clear the accumulator contents of 8085 Microprocessor. Explain the possible instructions for this purpose.

Solution: To clear the accumulator contents, the possible instructions are:

- (i) MVI A, 00 H: Two byte instruction and no flag gets affected.
- (ii) XRA A: One byte instruction, CY flag will be reset and all other flags will be modified as per the result.
- (iii) ANI 00 H Two byte instruction, CY flag will be reset and all other flags will be modified as per the result.
- (iv) SUB A One byte instruction and all flags will be affected as per the result.

Example 2: Write an assembly program to add two numbers

Program: Using Register

```

MVI D, 8BH      ; Load 83H data in register D
MVI C, 6FH      ; Load 6FH data in register C
MOV A, C        ; Copy data from Register C into Accumulator
ADD D           ; Add contents of D to contents A
OUT PORT1       ; Display result at port1
HLT             ; End of Program

```

Program : without using Register

```

MVI A, 03H      ; Load Data 03 to A
ADI 02H         ; Add 02H data to A
STA 2200H       ; Store the data at memory location 2200H
HLT             ; End of Program

```

Example 3: Addition of Two Numbers Stored at Two Memory Locations (2100H & 2101H) in the RAM

Program:

LDA 2100H	Load the contents of memory location 2100H to the accumulator
MOV B,A	Copy the contents of accumulator into register B
LDA 2101H	Load the contents of memory location 2101H to the accumulator
ADD B	Add the contents of register B to the contents of accumulator and store the result in accumulator itself
STA 2200H	Store the contents of accumulator to the memory location 2200H in the RAM
HLT	Halt (Stop executing)

Example 4: Write a program in mask off the least significant 4 bits of a given hexadecimal number in 8085. The answer should be stored in memory location 2200 H. Let the given number is B3 H.

Solution. The binary equivalent of B3 H is 1011 0011. The masking of the 4 LSB 0011 means to make 0011 to 0000. However, the four most significant bits should not be changed. This can be done if the given number is ANDed with F0 H (1111 0000). In doing so when 4 most significant bits are ANDed with 1111 no change will be there, but 4 least significant bits will be 0000 as required.

MVI A, B3 H	Loads the number B3H to accumulator
ANI F0 H	ANDs the accumulator with F0H and answer is loaded to accumulator.
STA 2200 H	Store the contents of accumulator to the memory location 2200H in the RAM

HLT Halt (Stop executing)

Example 5: Write a program to interchange (swap) the contents of two memory locations 2100 H and 2101 H.

Solution.

LDA 2100H	Loads the content of Memory location 2100H into accumulator.
MOV B, A	Moves the content of Accumulator to Register B
LDA 2101 H	Loads the content of Memory location 2101H into accumulator.
STA 2100 H	Store the content of Accumulator to Memory location 2100 H.
MOV A, B	Moves the content of Register B to Accumulator
STA 2101 H	Store the content of Accumulator to Memory location
HLT	

6.9. SUMMARY

The important concept in this chapter can be summarized as follows:

- The 8085-programming model includes six 8 bit registers, one accumulator, and one flag register. In addition, it also includes two 16-bit registers: the stack pointer and the program counter.
- 8085 microprocessors operations are classified into five groups: data transfer, arithmetic, logical, branch and machine control.
- An instruction is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: opcode and operand. Opcode and operand specifies operation to be performed and data on which operation is performed, respectively.
- The 8085 instruction set, according to word size can be classified as 1-byte , 2-byte , and 3-byte instructions

- The way operand is accessed to execute an instruction is called, addressing mode. 8085 have five addressing mode: Immediate, Register, Direct, Register indirect, and Implicit addressing.
- The instruction set of simple instructions is introduced in this chapter.

6.10. QUESTIONS

1. Identify the addressing mode (8085) for the following instructions:
 - a. MVI A, 23H
 - b. ADD B
 - c. IDA 2000 H

2. Specify the contents of the registers and the flag status as the following instructions are executed.


```
MVI A, 00H

MVI B, F8H
MOV C, A
MOV D, B
HLT
```

3. Specify the output of the PORT 17 H if the following instructions are executed and specify the contents of the register before and after execution.


```
MVI B, 02H
MOV A, B
MOV C, A
MVI D, 17H
OUT PORT
HLT
```

4. Write an instruction to load 00H in the accumulator, then decrement the accumulator and display the answer on the output port 01H.

5. Write a instructions to perform each operation
 - a. clear the accumulator
 - b. Add 48H
 - c. Subtract 93H
 - d. Add 68 H
 - e. Display the result in register B

6. write the mnemonic of an instruction that will set bit 6 of the accumulator without changing any of the other bits in the register. (**Ans.: ORI 40 H**)
7. What will be contents of accumulator and flags (CY, S, P and Z), after the execution of ADD D instruction; if A = C3 H and D = 3D H
8. What will be contents of accumulator and flags (CY, S, P and Z), after the execution of SUB D instruction; if A = C3 H and D = 3D H.
9. What will be contents of memory location 2500 H and flags (CY, S, P and Z), after the execution of the following program:

```
MVI C, C8 H  
  
MVI A, 11 H  
  
ADD C  
  
STA 2500 H  
  
HLT
```
10. Write a program to load a number 79 H in register B and mask off all bits except A₅ and A₂ bits. The result is to be transferred to register C.

*******End of chapter*******

Unit 7: Assembly Language Programming

7.1. Introduction

7.2. Objectives

7.3. Assembly Language Programming

7.4. Flow Chart

7.5. Writing an Assembly Language Program

7.6. Programming Techniques

7.6.1. Continuous Loop

7.6.2. Conditional Loop

7.7. Additional Instructions Of 8085

7.8. Questions

7.1 INTRODUCTION

Microprocessor programming is to write a set of instructions arranged in the specific sequence to complete the specific task. It tells the microprocessor what it has to do. To do this, the programmer must write the instructions in a language which processor understands. All processor understands only machine language which is written in binary numbers.

Machine language and Hex code instructions are very difficult to understand for the programmer. Hence for programmer, the instructions of microprocessor are made in the form of English abbreviation (short form). These instructions are name as Assembly Language instructions or mnemonics. The combinations of different mnemonics are known as Assembly Language Program and it is a low level language. In this unit, we will discuss some assembly language program and how to write assembly language program (ALP). Also, additional instructions from instruction set will be introduced.

7.2 OBJECTIVES

After studying this unit, you should be able to-

1. Describe the assembly language programming and coding rules.
2. Description of flow chart and its importance
3. Steps to write an assembly language program
4. Explanation of programming techniques as looping, counting , and indexing
5. Additional instructions of 8085 instruction set
6. To write complex program in assembly language

7.3. ASSEMBLY LANGUAGE PROGRAMMING

Just as the English language has its rules of grammar, assembly language has certain coding rules. The source line is the assembly language equivalent of a sentence. And a complete program is like a story.

The programming of the problem in any microprocessor is written in assembly language. The assembly language is written in mnemonics. The mnemonics are the initials or short form of the English word of the operation to be performed by the instruction. Assembly language statements are written in standard format as given below:

0Label Mnemonic Operand Comment

Label A label is a symbol name or group of symbols whose value is the location where the instruction is assembled. The label can be one to six alphanumeric characters, alphabetic or the special characters '?' or '@'. Following are the acceptable labels. NEXT, BACK, DELAY, LOOP20etc. Labels are always optional.

Mnemonic Short form of the operation to be performed or instruction code

Operand Operand is the data on which the operation is performed. It can be a data, memory address, register or port address.

Comment It is written to explain the program to the user. Although it is optional yet you should use them liberally since it is easier to debug and maintain a well-documented program. The comment statement is started with the semicolon

Machine language program are written in hexa-decimal numbers only, or in other words instructions, data, memory locations are represented by Hexa decimal / binary numbers which is very difficult to understand by user. The comments are not the part of the machine language program.

Machine language is difficult to understand by user but necessary for processor. Conversion of hex code to mnemonics makes it simple to user. This is shown through following example 7.1.

Example 7.1: Write assembly language and Machine language program using the instructions of 8085 microprocessor of the following statement.

Load the contents of memory locations 2100 H and 2101 H in B-register and C register, respectively. The content of memory locations 2100 H and 2101H are 16 H and 19 H respectively.

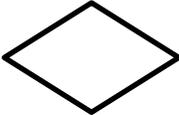
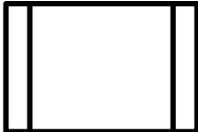
Assembly Language Program				Machine language program	
Label	Mnemonic	Operand	Comment	Memory Address	Content
	LDA	2100H	Loads the content of Memory location 2100 H into accumulator	2000 H 2001 H 2002 H	3A H 00 H 21 H
	MOV	B, A	Copy the content of accumulator to B-register	2003 H	47 H
	LDA	2101H	Loads the content of Memory location 2101 H into accumulator.	2004 H 2005 H 2006 H	3A H 01 H 21 H

MOV	C, A	Copy the content of accumulator to C-register	2007 H	4F H
HLT			2008 H	76 H
			2100 H	16 H
			2101 H	19 H

From the above example, the difference between Assembly language and Machine language it is clear that chances of error is higher as it contains only hexa decimal numbers. Therefore, ALP is preferred for writing programs.

7.4. FLOW CHART

A flowchart is simply a graphical representation of flow of data to complete the solution of a problem. It shows steps in sequential order and is widely used in presenting the flow of algorithms or processes. Typically, a flowchart shows the steps as boxes of various kinds, and their order by connecting them with arrows. The commonly used symbols used in flow chart are given below:

Oval : It indicates start or stop operation.	
Arrow : It indicates flow with direction.	
Parallelogram: It indicates input/output operation.	
Rectangle: It indicates process operation.	
Diamond: It indicates decision making operation.	
Circle with alphabet: It indicates continuation. A: Any alphabet	
Double sided Rectangle: It indicates execution of pre-defined process (subroutine).	

7.5. WRITING AN ASSEMBLY LANGUAGE PROGRAM

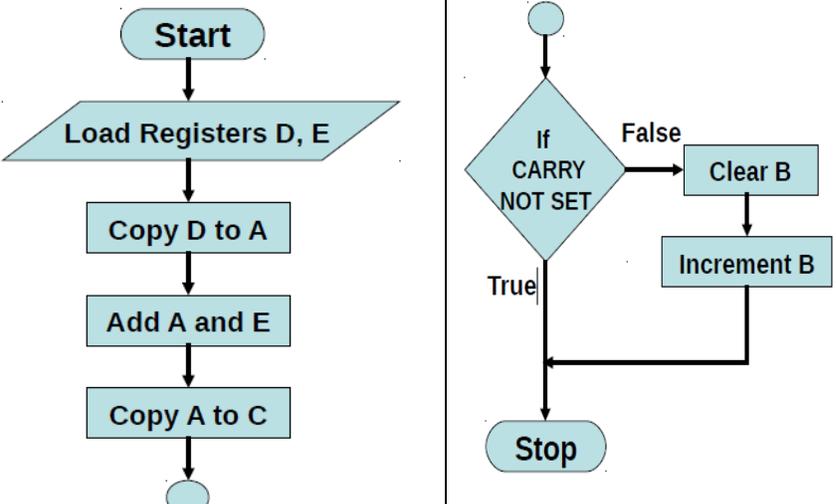
Like any other coding language, an assembly language also follows almost same steps to write a program. An Assembly Language Program is written by using the following steps:

1. **Analyze the problem:** This means in the programming is to find out which task is to be performed and what will be expected outcome. This is also called as specifying the problem.
2. **Develop program Logic:** in this step, programmer develop logic to perform the specified task.
3. **Write an Algorithm:** Algorithm defines step-by-step rules/instructions for logic to complete the specified task in order to get the expected results.
4. **Make a Flowchart:** It is basically the graphical representation of various actions which are to be performed to complete the specified task in order to get the expected results.
5. **Coding:** it is implementation of various actions by writing step by step instruction available with the microprocessor.
6. **Debugging:** Once the program or a part of program is coded, the next step is debugging the code. Debugging is the process of testing the [code](#) to see if it does the given task. If program is not working properly, debugging process helps in finding and correcting errors.

Example 7.2 : Here, an example have been taken to explain the above steps in assembly language programming.

Write a program in 8085 Assembly language to add two 8- bit numbers and save carry to register, if generated.

1. Analyze the problem	<p>Result of addition of two 8-bit numbers can be 9-bit</p> <pre style="text-align: center;"> 1001 1000 (98H) + 1001 1001 (99H) Sum 1 0011 0001 (131H) </pre> <p>The 9th bit in the result is called CARRY bit.</p> <p>Here the task is addition and outcome is 8 or 9 bit binary number.</p>
-------------------------------	--

2. Program Logic	<p>How 8085 does it?</p> <p>1001 1000 8 bit data is stored in Register E 1001 1001 8 bit data is stored in Register D Cy=1 0011 0001 Sum is stored in A and Sets CY flag 3 1 H</p> <p>Two registers are required to store SUM, if Carry is generated.</p>	
3. Algorithm	<ol style="list-style-type: none"> 1. Load two numbers in registers D, E 2. Add them 3. Store 8 bit result in C 4. Check CARRY flag 5. If CARRY flag is SET , Store CARRY in register B If not, then move to stop 6. Stop 	
4. Flow Chart	 <pre> graph TD Start([Start]) --> Load[/Load Registers D, E/] Load --> CopyD[Copy D to A] CopyD --> Add[Add A and E] Add --> CopyA[Copy A to C] CopyA --> End1(()) End1 --> Decision{if CARRY NOT SET} Decision -- True --> Stop([Stop]) Decision -- False --> ClearB[Clear B] ClearB --> IncB[Increment B] IncB --> Stop </pre>	
	<p>Fig.7.1.: Flow chart of Addition of two number with carry</p>	
5. Assembly Language Program	<ol style="list-style-type: none"> 1. Load registers D, E 	<pre>MVI D, 02H MVI E, 03H</pre>
	<ol style="list-style-type: none"> 2. Copy register D to A 	<pre>MOV A, D</pre>
	<ol style="list-style-type: none"> 3. Add register E to A 	<pre>ADD E</pre>
	<ol style="list-style-type: none"> 4. Copy A to register C 	<pre>MOC C,A</pre>
	<ol style="list-style-type: none"> 5. Check: if Carry generated: No 	<pre>JNC : LAST</pre>
	<ol style="list-style-type: none"> 6. Yes: Clear Register B and Increment by 1 	<pre>MVI B, 00H INR B</pre>
	<ol style="list-style-type: none"> 7. Stop processing 	<pre>LAST: HLT</pre>
6. Debugging	<p>If the program does not give the expected result, then its output is checked by step by step to find bug. Generally, it is performed for</p>	

	very large program. Mostly the Error in program appears when hex codes of machine languages are entered into hardware.
--	--

7.6 PROGRAMMING TECHNIQUES

In earlier programming examples, programming was simple and non-repetitive type. However, in copying thousands of data bytes from one memory location to another location or addition of data stored in 50 memory locations require repeating of instructions.

To remove this repetition in a program, a programming technique called looping is incorporated in 8085. Looping is used to instruct the microprocessor to repeat tasks. This task is accomplished by using jump instructions.

Loops can be classified as:

1. Continuous loop
2. Unconditional loop

7.6.1 Continuous Loop

A continuous loop repeats a task continuously. This is set up by using the unconditional jump instruction. A program with a continuous loop does not stop repeating the tasks until the system is reset. Typical examples are electronics watches, temperature recording of a city etc

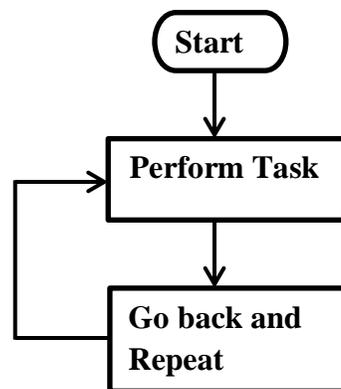


Fig.7.2 Flow chart of Continuous Loop

7.6.2 Conditional Loop

A conditional loop repeats the task until specific condition is met. This is set up by using a conditional jump instruction. The specific condition is set by flags (Z, CY, P, and S) in 8085. These loops include counting and indexing.

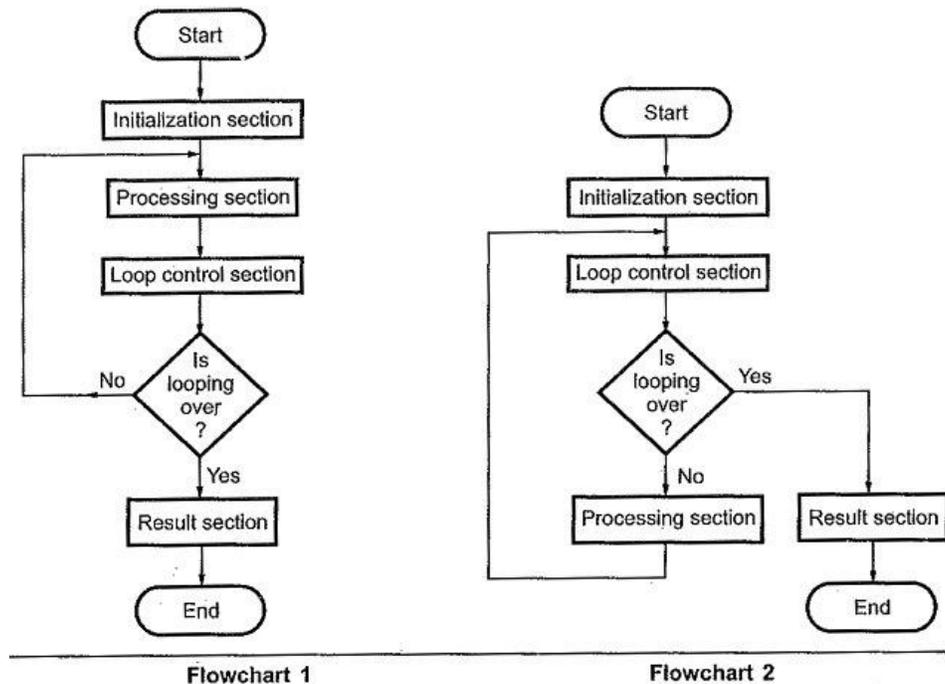


Fig.7.3 flow chart of Conditional Loop

Conditional Loop and Counter:

Sometimes, a conditional loop can not be established using arithmetic or logical operation. For example, addition of ten numbers. In executing this program, it is very difficult to find condition whether addition is completed or not? To perform such task, counter and flags are used to form conditional loop which is called counting.

Counting is performed as follows:

- a. Counter is set up by loading an appropriate count in a register.
- b. Counting is performed by either increment or decrement the counter.
- c. Loop is set up by a conditional jump instruction.
- d. End of counting is indicated by a flag.

Conditional Loop, Counter and Indexing:

This is another type of loop which includes counter and indexing. Indexing is pointing of referencing objects with sequential numbers. Data bytes are stored in memory locations and those data bytes are referred to by their memory locations.

7.7 ADDITIONAL INSTRUCTIONS OF 8085

All five types of simple instructions for Data transfer operations, Arithmetic operations, Logical operations, Branching operations, and Machine-control operations are introduced in previous unit. Some instructions are developed in 8085 microprocessor to solve more complex problems. These additional instructions for 16 bit operations are included in this section.

a. Data transfer instructions

Instructions	Description
LDA 16-bit address	Load the Accumulator Direct with contents of memory location specified by a 16-bit address Example – LDA 2050H The contents of a memory location 2050, are copied to the Accumulator.
LDAX Register pair (B/D)	Load the Accumulator indirect with the memory location specified by Register pair B/D Example – LDAX B The contents of memory location pointed by the register pair B, are copied to the Accumulator
LXI Reg. pair, 16-bit data	Load the register pair immediate with 16 bit data Example – LXI B, 3225H The instruction loads data 3225H in the designated register pair B. Here, 32 and 25 load in Reg. B and C, respectively
STA 16-bit address	Store the Accumulator Direct to the memory location specified by a 16-bit address Example – STA 2050H The contents of Accumulator are copied to the memory location 2050.
STAX Register pair (B/D)	Store Accumulator indirect with the memory location specified by Register pair B/D Example – STAX D The contents of accumulator are copied to the memory location pointed by the register pair B.

LHLD 16-bit address	<p>Load H and L registers direct The instruction copies the contents of the memory location pointed out by the address into register L and copies the contents of the next memory location into register H. Example – LHLD 2000H Contents of memory location 2000H and 2001H are copied to register L and H, respectively.</p>
SHLD 16-bit address	<p>Store H and L registers direct The contents of register L are stored into the memory location specified by the 16-bit address and the contents of H register are stored into the next memory location. The contents of registers HL are not altered. Example – SHLD 2000H Contents of Register L and H are copied to memory location 2000H and 2001H, respectively.</p>
XCHG	<p>Exchange H and L with D and E The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.</p>
SPHL	<p>Copy H and L registers to the stack pointer The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.</p>
XTHL	<p>Exchange H and L with top of stack The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1).</p>

b. Arithmetic instructions

Instructions	Description
DAD Register pair	<p>Add the register pair to H and L registers The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD B</p>
INX Register pair	<p>Increment register pair by 1 The contents of the designated register pair are incremented by 1 and their result is stored at the same place. Example – INX B</p>
DCX Register pair	<p>Decrement the register pair by 1 The contents of the designated register pair are decremented by 1 and their result is stored at the same place. Example – DCX D</p>

DAA	<p>Decimal adjust Accumulator</p> <p>The contents of the Accumulator are changed from a binary value to two 4-bit BCD digits.</p> <ul style="list-style-type: none"> • If the value of the low-order 4-bits in the Accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. • If the value of the high-order 4-bits in the Accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.
------------	--

Example 7.3:What will be the value of accumulator, CY and AC flags after the execution of the following program?

```
MVI A, 38 H
ADI 87 H
DAA
HLT
```

Solution.

```
MVI A, 38 H; copy 38H to A   A = 38 H 0 0 1 1 1 0 0 0
ADI 87 H; directly add 87H   87 H  1 0 0 0 0 1 1 1
DAA                          1 0 1 1 1 1 1 1; Lower nibble is more than 9 so, AC = 1
                                0 0 0 0 0 1 1 0; add 6 to lower bit
                                1 1 0 0 0 1 0 1; Upper nibble is more than 9, so, CY=1
                                0 1 1 0 0 0 0 0; add 6 to lower bit
                                CY= 1 0 0 1 0 0 1 0 1; so answer is 125(IN DECIMAL) with Carry
                                and Auxiliary Carry flag SET
```

c. Logical instructions

CMP R/M	<p>Compare the register or memory with the Accumulator</p> <p>The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:</p> <p>if (A) < (R/M): carry flag is set if (A) = (R/M): zero flag is set if (A) > (R/M): carry and zero flags are reset</p> <p>Example: CMP B or CMP M</p>
----------------	---

CPI8-bit data	<p>Compare immediate with the Accumulator The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows: if (A) < data: carry flag is set if (A) = data: zero flag is set if (A) > data: carry and zero flags are reset Example: CPI 75H</p>
RLC	<p>Rotate Accumulator left Each binary bit of the Accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7.</p>

Example 7.4: Illustrate the execution of instructions RLC, assume that Accumulator have data AA H and carry flag is reset.

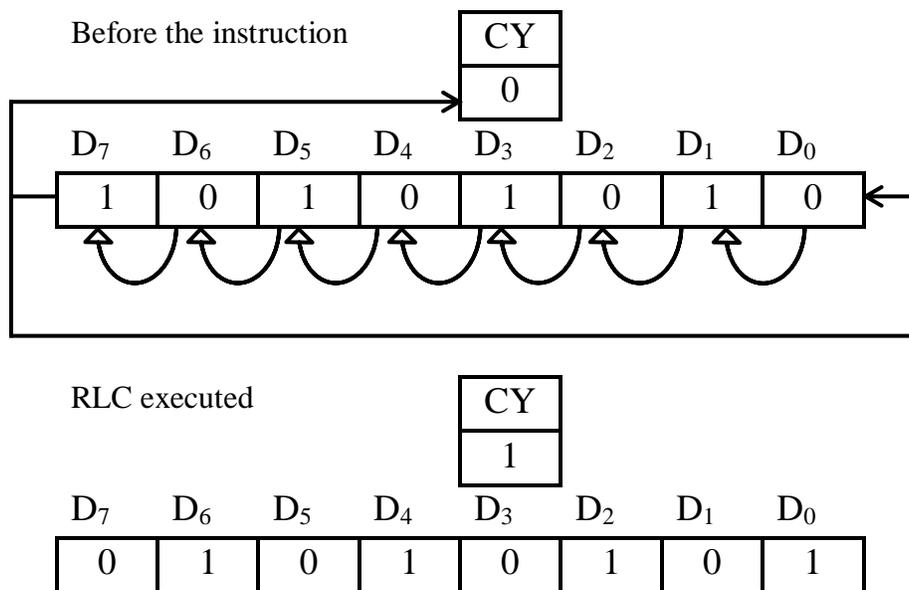


Fig. 7.4, Illustration of the execution of instruction RLC

After the execution of RLC, the data in Accumulator is 55H and Carry flag is SET.

RAL	<p>Rotate the Accumulator left through carry Each binary bit of the Accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.</p>
------------	--

Example 7.5: Illustrate the execution of instructions RLC, assume that Accumulator have data AA H and carry flag is reset.

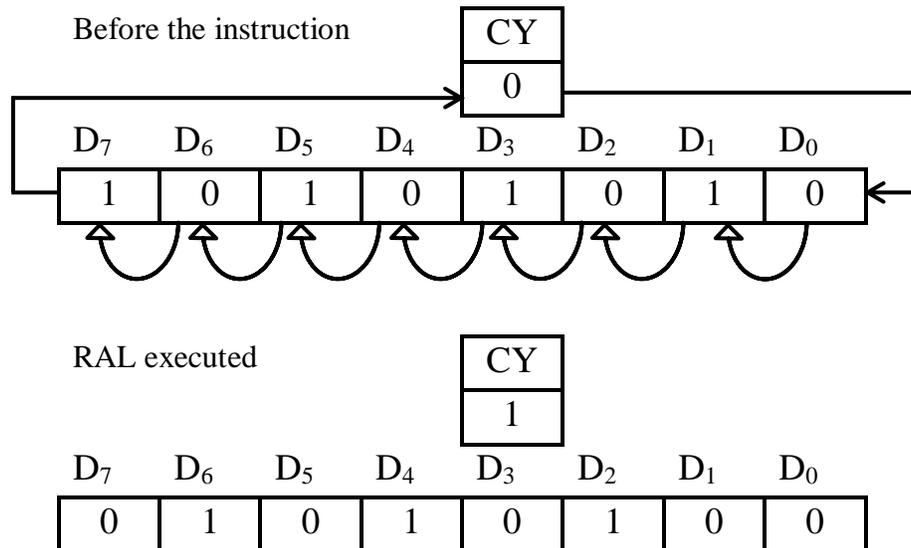


Fig.7.5: Illustration of the execution of instruction RAL

After the execution of RAL, the data in Accumulator is 54H and Carry flag is SET.

RAR	Rotate the Accumulator right through carry Each binary bit of the Accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.
RRC	Rotate the Accumulator right Each binary bit of the Accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0.

Example 7.6: Write a program in assembly language for 8085 Microprocessor to find the sum of a series $1+2+3+\dots+10$. (or sum of first 10 natural numbers).

Solution.

Label	Instructions	Comments
	MVI B, 01H	;Load the first number of series to register B.
	MVI C, 00 H	;Clears register C.

```

LOOP MOV A, C    ; Saves the contents of C register to accumulator.
      ADD B      ; Add the contents of B register with the contents of accumulator.
      MOV C, A   ; Save the partial sum to C register.
      INR B     ; increment the contents of B-register.
      CPI 0B H  ; Compare with natural number 11.
      JNZ LOOP  ; if less than 11 go back to LOOP.
      MOV A, C  ; Copy the answer to accumulator.
      STA 2500 H ; Store the answer to memory location.
      HLT

```

Example 7.7: Write a program for the multiplication of two numbers stored at memory locations 2100H and 2101H. Store the result at 2200H. Assume that the result does not overflow.

Solution :

```

      LDA 2100H  ; Load the contents of memory location 2100H (multiplicand) to A
      MOV B, A   ; Copy the contents of accumulator into register B
      LDA 2101H  ; Load the contents of memory location 2101H (multiplier) to A
      MOV C, A   ; Copy the contents of accumulator into register C
      XRA A     ; Clear the accumulator
NEXT:  ADD B     ; Add the contents of register B to the contents of A
      DCR C     ; Decrease the contents of register C by one
      JNZ NEXT  ; If zero flag is not set, then jump to the label NEXT
      STA 2200H ; Store the contents of A to the memory location 2200H in RAM
      HLT      ; Halt (Stop executing)

```

Example 7.8: Write a program to sort given 10 numbers from memory location 2200H in the ascending order.

Solution:

```

                MVI B, 09      ;Load Register B to initialize counter 1
START  LXI H, 2200H    ; Initialize memory pointer by HL pair
                MVI C, 09H    ;Load Register C to initialize counter 2
BACK: MOV A, M      ; Copy the contents of memory location pointed by HL to A
        INX H        ; Increment HL pair by 1
        CMP M        ; Compare number with next number
        JC SKIP      ; If less, don't interchange
        JZ SKIP      ; if equal, don't interchange
        MOV D, M     ;Copy the contents of memory location pointed by HL to D
        MOV M, A     ; Copy the contents of A to memory location pointed by HL
        DCX H        ; Decrement HL pair by 1
        MOV M, D     ;Copy the contents of Reg D to memory location pointed by HL
        INX H        ;Increment HL pair by 1
SKIP: DCR C         ; Decrement counter 2
        JNZ BACK     ; If not zero, repeat
        DCR B        ; Decrement counter 1
        JNZ START    ; Jump if not zero to start
        HLT          ; Terminate program execution

```

7.8. QUESTIONS

1. Write a program that adds the three numbers in locations 2500 H, 2501 H and 2502 H, and puts the sum in 2503 H.
2. Write a program to count the number of 1's in a given 8-bit data.
3. Write a program that interchanges the values in locations 2500H and 2501 H.
4. Write an assembly language program of 8085 to fill the RAM area from 2500 H to 25FF H with a byte 33 H.
5. Write a program that computes the sum $1 + 2 + 3 + \dots + n$, where n is the value in Register H. The sum should be placed in 2500 H onwards.
6. Write a program to add a series of 10 numbers. These numbers are stored in consecutive memory locations. The result should be stored in the memory locations following the input data. What should be the size of the result location? What if you were to add a series of 1000 numbers? How much result storage space would be sufficient? Write a report.
7. What will be contents of accumulator and flag register, after the execution of following program:


```

MVI A, 47 H
MVI B, 37 H
ADD B
DAA
HLT

```
8. Sixteen bytes of data are stored in memory locations 2001 H to 2010 H. Write an assembly language program of 8085 to transfer this block of data to 2006 H to 2015 H.
(Hint: In this case data will be transferred in the reverse order otherwise some of the data will coincide.)
9. Write an ALP for 8085 to find the smallest of the three numbers stored in memory locations starting at 2301 H. Store the smallest number in 2304 H.
10. Write an ALP for 8085 to find the smallest number among a series of 16 numbers stored in the memory locations starting at 2301 H. The number 16 (10 H) is stored in 2300 H. The smallest number should be stored in 2401 H.
11. Write an ALP to find 13 terms of Fibonacci series. The terms of the series are to be stored in the memory locations starting at 2501 H.
12. Write an ALP for 8085 to shift an 8-bit number left by two bits. The number is stored in memory location 2501 H and the result is to be stored in 2502 H memory location. (Hint: Keep the number in accumulator and use ADD A instruction twice.)
13. Write an ALP for 8085 to shift a 16-bit number left by two bits. The number is stored in memory locations 2501 H and 2502 H. The result is to be stored in 2503 H and 2504 H memory locations.

Unit 8- Counters time delay, Stack and code conversion

8.1 Introduction

8.2 Objectives

8.3. Counters And Time Delay

8.3.1 Time Delay Using One Register

8.3.2. Time Delay Using A Register Pair

8.3.3 Time Delay Using A Loop Within A Loop Technique

8.3.4 Additional Techniques For Time Delay

8.3.5 Hexadecimal Counter

8.4. Stack

8.5. Subroutine

8.6. Binary To Ascii Code Conversion

8.7. Bcd To Binary Code Conversion

8.8. Bcd To Binary Code Conversion

8.1 INTRODUCTION

Counters are needed to perform an event at a specific time. They are commonly used in applications such as traffic light signals, digital clocks, timing alerts, serial data transfer and other event driven functions. These counters and time delays are designed using microprocessor programming. Basically looping and counting techniques are used to design counters and time delays. A counter is designed by loading a count in a register. This count is decided by delay required and clock frequency. Then, a loop is set up either to decrement or increment the count for a down/up-counter.

The stack and the subroutine offer a great deal of flexibility in writing programs. A stack is a particular group of memory locations in the Read and Write memory that is used for temporary storage of data during the execution of a program. The starting memory location of the stack is defined in the main program, and this space is reserved at an isolated part of memory. A subroutine is a group of instructions that performs a part of main program which can be used repeatedly (e.g., time delay or arithmetic operation). The subroutine is written as a separate unit, apart from the main program, and the microprocessor transfers the program execution from the main program to the subroutine whenever it is called to perform the task. After the completion of the subroutine task, the microprocessor returns to the main program. The subroutine technique eliminates the need to write a subtask repeatedly; thus, it uses memory more efficiently

8.2 OBJECTIVES:

After studying this unit, you should be able to-

1. Describe the counters and timing delays
2. Write programs for implementing counter and timing delay using loop and counting techniques
3. Calculate the time delay in a given loop
4. Write program to turn on/off specific bits for different functions at a given interval.
5. Define the stack, the stack pointer (register), and the program counter, and their uses

6. Illustrate the concepts of subroutines in Assembly Language Program
7. Concept of various code conversions used in microprocessor programming

8.3. COUNTERS AND TIME DELAY

Counters are used primarily to keep track of events in any module which are used to start/ stop or execute actions and time delays are important in setting up reasonably accurate timing between two events. The process of designing counters and time delays using microprocessor instructions is much more flexible and require less time in implementation than its hardware implementation.

In microprocessor, counters are designed by loading an appropriate number into a register or register pair, depending on the time delay required. Then, this register or pair is decremented or incremented by one, until it reaches to final value by setting up a loop with a conditional jump instruction using flag status. The flow chart of a counter is shown in fig.8.1.

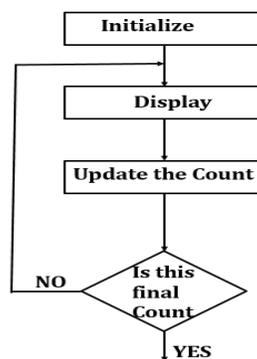


Fig. 8.1: Flowchart of Time Delay

Time Delay is the certain delay interval which is created by executing instructions (Software programs) to keep a track between events. Time delay can be designed through executing group of instructions number of times. Flow chart as an example of time delay is shown in Fig. (8.2)

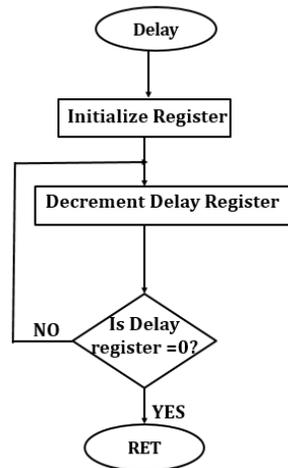


Fig. 8.2:Flowchart of Time Delay

Calculating Time Delay.

Each instruction passes through different combinations of Opcode Fetch, Memory Read, and Memory Write cycles. Knowing the combinations of cycles, we can calculate the time required to complete an instruction. The formula for calculating the time delay is given by:-

$$\text{Time Delay} = \text{No. of T-states} * \text{Clock Period}$$

For example, “MVI A, 23H” instruction uses 7 T-States and if Microprocessor is operating at 2 MHz frequency. So, time delay to execute this instruction will be:

$$\text{Time delay} = 7 * 0.5 \mu\text{s} = 3.5 \mu\text{s}.$$

Time delay has two components 1. Time delay created due to instructions execution not in the loop (T_{NL}) and 2. Time delay created due to execution of instructions in the loop (T_L).

$$\text{So, total time delay} = T_{NL} + T_L$$

Instructions are added outside the loop to make minor variation in time delay. For Large delay requirement instructions are added in loop.

Time delay in microprocessor can be implemented using following techniques:

- d. Time Delay Using One Register
- e. Time Delay Using a Register Pair

- f. Time Delay Using a Loop within a Loop Technique
- g. Additional techniques

8.3.1 Time Delay Using One Register

In this type of counter, the time delay is created by storing data in a single register.

Example: for a clock frequency of 4MHz of 8085 microprocessor, calculate the time delay of the following program?

MVI B, FF H

Loop: DCR B

JNZ Loop

Solution:

Clock frequency of the system $f = 4 \text{ MHz}$

Clock period $T = 1/f = 1/4 \times 10^{-6} = 0.25 \mu\text{s}$

Instructions	No of T-States required	No of times executed	Part of loop	Execution time
MVI B, FF H	7	01	No	$7 \times 1 \times 0.25 = 1.75 \mu\text{s}$
DCR B	4	FF H (255 in Decimal)	Yes	$4 \times 255 \times 0.25 = 255.00 \mu\text{s}$
JNZ	10/7	FF H (255 in Decimal)	Yes	$(10 \times 254 + 7 \times 1) \times 0.25 = 636.75 \mu\text{s}$
Total execution time = $1.75 + 255.00 + 636.75 = 893.50 \mu\text{s}$				

In above example, Instructions MVI B, FFH is not a part of Loop and hence executed once. Instructions DCR B and JNZ are part of loop. So, these are executed 255 times. Also, Instructions JNZ have 10/7 execution states which means that if condition is False, then it is executed in 10 T and if condition is true then executed in 7 T states. So, JNZ instructions took 10 T States 254 times and 7 T states in last time execution.

8.3.2. Time Delay Using a Register Pair

The data in single register is restricted to 8-bit only and time delay is limited to small value. The time delay can be considerably increased by setting a loop and using a register pair which can store a 16-bit number (maximum FFFFH). But, the register pair instructions does not set zero flag and

without setting the flags, Jump instructions cannot take check the required conditions. Therefore, additional instructions are used to set Zero flag.

Example: for a clock frequency of 2MHz of 8085 microprocessor, calculate the time delay of the following program?

LXI D, 4567H

Loop: DCXD

MOV A, E

ADD D

JNZ Loop

Solution:

Clock frequency of the system $f = 2 \text{ MHz}$

Clock period $T = 1/f = 1/2 \times 10^{-6} = 0.5\mu\text{s}$

Instructions	T-States required	No of times executed	Part of loop	Execution time	
LXI D, 4567H	10	01	No	$10 \times 1 \times 0.5$	5.0
DCXD	6	17767	Yes	$6 \times 17767 \times 0.5$	
MOV A, E	4	17767			
ADD D	4	17767			
JNZ	10/7	17767	Yes	$(10 \times 254 + 7 \times 1) \times 0.25 =$ $636.75\mu\text{s}$	
Total execution time = $1.75 + 255.00 + 636.75 = 893.50\mu\text{s}$					

In this example, Instructions LXI D, 4567H is not a part of Loop and hence executed once. Instructions DCXD, MOV A,E , and ADD D are part of loop. So, these are executed 17767 times. Instructions JNZ is also in loop but it is executed 17766 times with 10 T states and one time with 7 T states.

8.3.3 Time Delay Using a Loop within a Loop Technique

A larger time delay can also be achieved by using two loops; one loop inside the other loop, as shown in Figure 8.3(a). In this method, first register is used in the inner loop (LOOP1) and

Second register is used for the outer loop (LOOP2). The delay count will be the multiplication of data in these two registers. The following example illustrates the delay in this method.

Example: for a clock frequency of 2MHz of 8085 microprocessor, calculate the time delay of the following program?

DELAY CALCULATIONS

```

                MVI B,38H

LOOP2:         MVI C,FFH

LOOP1:         DCR C

                JNZ LOOP1

                DCRB

                JNZ LOOP2

```

Instructions	T-States required	Part of loop	No of times executed	Total T States used
MVI B, 38H	7	No	01	7.0
MVI C, 38H	7	Yes Loop2	38 H= (56) ₁₀	56*7= 392
DCR C	4	Yes,Loop2 and Loop1	38H * FFH= 56*255=	57120
JNZ	10/7	Yes Loop2 and Loop1	38H * FFH	14120 142744
DCR C	4	Yes,Loop2	38H	56*4= 224
JNZ	10/7	Yes,Loop2	38H	56= 55*10+7=557

The total delay should include the execution time of the first instruction (MVI B,7T); however, the delay outside these loops is insignificant. The time delay can be increased considerably by using register pairs in the above example.

Similarly,

The desired time delay can be obtained by using any or all available registers.

8.3.4 Additional Techniques for Time Delay

- Addition of NOP Instruction in the loop: the time delay within a loop can be increased by using NOP (No Operation) instruction that will not affect the program. NOP is executed in four T-states and will add four T-States in the delay loop to increase the delay.
- Using register pair in both loops: for getting large delay 16 bit data in both loops using register pairs can be obtained.

Generation of counters or time delay using software programming have the following disadvantage:

- a. Underutilization of microprocessor as it is used to generate clock or time delay instead of performing computation.
- b. Exact calculation of time delay is tedious and difficult to achieve.
- c. Poor accuracy in real time applications

8.3.5 HEXADECIMAL COUNTER

Hexadecimal counters in 8085 counts in binary and displays the result in hexadecimal form to external display through Input-Output ports. It has two different parts (i) main counting part and (ii) the delay part. A delay subroutine is used to generate delay between each number while counting. The functioning of the hexadecimal counter is explained with the help of the following program.

Example 7.1: Write a program to count continuously in hexadecimal from FFH to 00H in a system with a 0.5 μ s clock period. Set up a one millisecond (ms) delay between each count and display the numbers at one of the output ports with port address 40H.

The example has two parts; the first is to set up down counter to display count from FFH to 00H, and the second is to design a one millisecond delay between two counts.

The first part is implemented using the hexadecimal counter in outer loop. One register, Say B is set up by loading 00H as starting number which will not be part of loop. which a register with an appropriate starting number and decrementing it until it becomes zero (shown by the outer loop in the flowchart, Figure 8.2). After zero count, the register goes back to FF because decrementing zero results in a (-1), which is FF in 2's complement.

The one millisecond (ms) delay between each count is set up by using the procedure explained previously in Section 8.1.1- Time Delay Using One Register. Figure 8.2 is identical with the inner loop of the flowchart shown in Figure 8.6. The delay calculations are shown later.

```
MVI B,00H
```

```
NEXT: DCR B
```

```

MVI C,COUNT
DELAY:DCR C
JNZ DELAY
MOV A,B
OUT PORT#
JMP NEXT

```

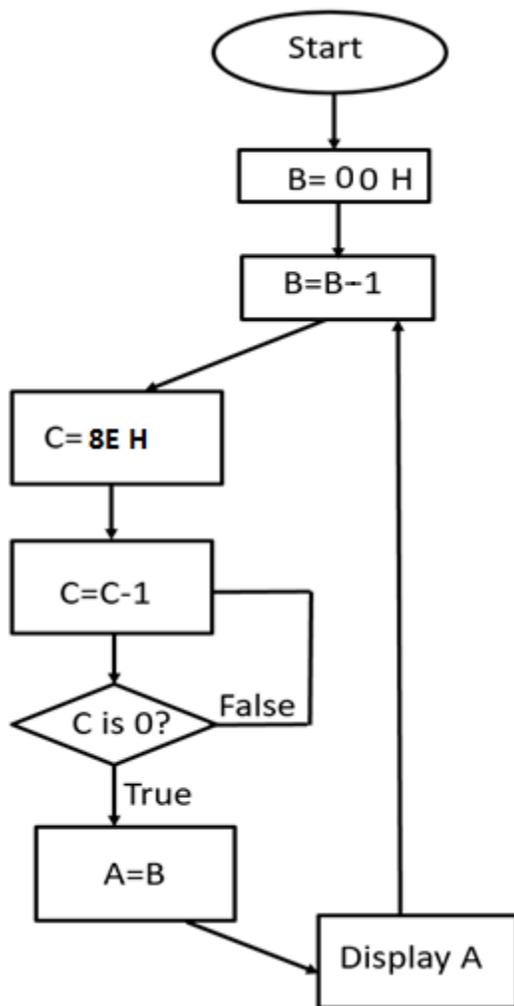


Fig. 8.3

8.4. STACK

The stack in 8085 microprocessor is a reserved area of memory to store temporary data during the execution of a program. The highest address of the stack is stored in 16 bit register called as the

stack pointer (SP). Instructions LXI,SP 16 bit address is used to load 16-bit memory address in the stack pointer register. The difference between the stack and the stack pointer is that the stack is an area of memory; the stack pointer is the address of the last value pushed onto the stack. Usually, the stack is used for storing data when subroutines are called. The stack is a last-in-first-out, i.e., LIFO structure so the last thing stored in the stack is the first thing retrieved.

In 8085, following three instructions are used

1. LXI SP 16 bit memory address: Load Stack Pointer
2. PUSH Register pair; Store Register Pair on Stack
3. POP Register Pair; Retrieve Register Pair from Stack

1. LXI SP 16 bit memory address

This instruction loads a 16-bit memory address in the stack pointer register of the microprocessor. This is a 3-byte data transfer instruction. Therefore, does not affect the flags.

For example, LXI SP, 2099H, loads the stack pointer register with the memory address 2099H, and the storing of data bytes begins at 2098H and continues in reversed numerical order (decreasing memory addresses such as 2098H, 2097H, etc.). Therefore, as a general practice, the stack is initialized at the highest available memory location to prevent the overlapping of program and temporary data storage which can destroy the program.

2. PUSH Register pair (B/D/H/PSW)

- It copies the contents of the specified register pair on the stack
- This is a 1-byte instruction
- The operands B, D, and H represent register pairs BC, DE, and HL, respectively
- The operand PSW represents Program Status Word, meaning the contents of the accumulator and the flags.

Instruction Execution

PUSH B : a. Decrement the contents of Stack pointer (SP) by 1

- b. Copy the contents of register B to the memory location pointed to by SP
- c. Again, decrement the contents of SP by 1
- d. Copy the contents of register C to the memory location pointed to by SP

In Fig.xx execution of instruction PUSH B is illustrated. Initially, Stack pointer is pointing to memory address FFFF H. In execution of PUSH B, SP is decremented by 1 and becomes FFFE H. The contents of register B, 12H is copied to memory location FFFE H. Then, SP is decremented by one and becomes FFFD H. Now, The contents of register C, F3H is copied to memory location FFFD H.

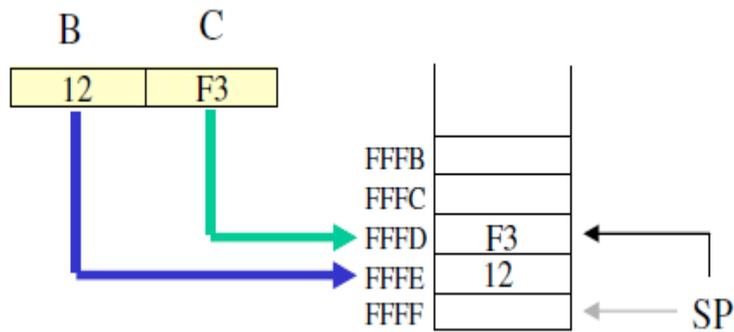


Fig.8.4. Illustrates the operation of Instruction PUSH B

3. POP Register pair (B/D/H/PSW)

- It retrieve data to Register Pair from Stack
- This is a 1-byte instruction
- The operands B, D, and H represent register pairs BC,DE, and HL, respectively
- The operand PSW represents Program Status Word,meaning the contents of the accumulator and the flags.

Instruction Execution

- POP D :**
- a. Copy the contents of the memory location pointed to by SP to register E
 - b. Increment the contents of Stack pointer (SP) by 1
 - c. Copy the contents of the memory location pointed to by SP to register D
 - d. Increment the contents of Stack pointer (SP) by 1.

In Fig.xx execution of instruction POP D is illustrated. Initially, Stack pointer is pointing to memory address FFFD H. In execution of POP D, The contents of memory location FFFD H which is F3H is copied to register E. The contents of Stack pointer (SP) is incremented by 1 and becomes FFFE H. Now, The contents of memory location FFFE H which is 12H is copied to register D. Then, The contents of Stack pointer (SP) is incremented by 1 and becomes FFFFH

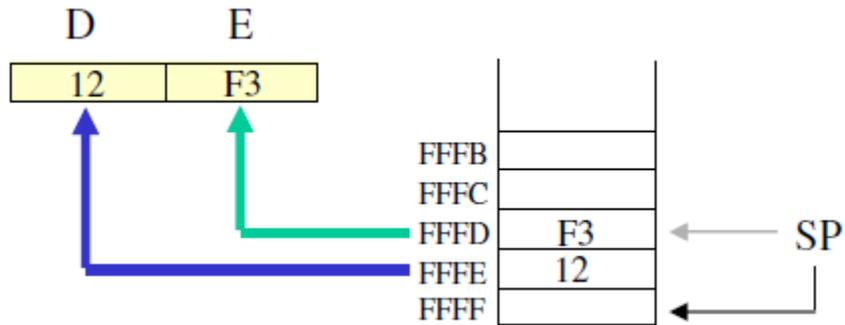


Fig.8.5. Illustrates the operation of Instruction POPD

Example: Write a program to complete the following operation:

- Initialize the stack pointer register at FFA0H.
- Load register pairs B, D, and H with data 0237H, 1242H, and 4087H, respectively.
- Push the contents of the register pairs B, D, and H on the stack.
- POP the contents of Register pairs B, D, and H from the stack.

Solution:

Program	SP	Memory location	B	C	D	E	H	L
LXI SP, FFA0H	FFA0	-	-	-	-	-	-	-
LXI B, 0237H	FFA0	-	02H	37H	-	-	-	-
LXI D, 1242H	FFA0	-	02H	37H	12H	42H	-	-
LXI H, 4087H	FFA0	-	02H	37H	12H	42H	40H	87H
PUSH B	FF9E	FF9F:02H FF9E:37H	02H	37H	12H	42H	40H	87H
PUSH D	FF9C	FF9D: 12H	02H	37H	12H	42H	40H	87H

		FF9C: 42H						
PUSH H	FF9A	FF9A: 40H FF9B: 87H	02H	37H	12H	42H	40H	87H
POP B	FF9C		40H	87H	12H	42H	40H	87H
POP D	FF9E		40H	87H	12H	42H	40H	87H
POP H	FFA0		40H	87H	12H	42H	02H	37H

Explanation:

LXI SP, FFA0H	Initilias the SP to FFA0
LXI B, 0237H	Load the Rp (B&C) with B=02H and C= 37H
LXI D, 1242H	Load the Rp (D&E) with B=12H and C= 42H
LXI H, 4087H	Load the Rp (H&L) with B=40H and C= 87H
PUSH B	Copy the contents of B to M(FF9F):02H and C to M(FF9E):37H
PUSH D	Copy the contents of D to M(FF9D):12H and E to M(FF9C):42H
PUSH H	Copy the contents of H to M(FF9B):40H and Lto M(FF9A):87H
POP B	Copy the contents of M(FF9A) to C: 87H and M(FF9B) to B:40H
POP D	Copy the contents of M(FF9C) to E: 42H and M(FF9D) to B:12H
POP H	Copy the contents of M(FF9E) to L: 37H and M(FF9F) to H:02H

8.5. SUBROUTINE

A **subroutine** is a program written to perform a specific function that is repeatedly used during the execution of main program. This program can be written to implement delays, data transfer, data sorting and other applications. Subroutines programs are written once, avoid repetition of the same instructions, separately from the main program, and are called by the main program when needed. Hence, save the memory space.

In 8085, following two instructions are used to implement subroutines:

1. CALL : call a subroutine
2. RET (return to main program from a subroutine)

CALL 16 bit memory address

- It calls subroutines unconditionally that transfers the program sequence to a subroutine address
- This is a 3-byte instruction
- Saves the contents of the Program Counter (the address of the next instruction which will be executed after return to main program) on the stack
- Decrements the stack pointer register by two to store 16 bit address in program Counter
- This instruction is accompanied by a return instruction in the subroutine

RET

- It return from Subroutine Unconditionally to the next address of the main program
- This is a 1-byte instruction
- Inserts the two bytes from the top of the stack into the program counter and increments the stack pointer register by two

These call and return functions are classified into two categories:

- unconditional Call
- conditional Call
- unconditional Return
- conditional return

(a) Unconditional Call instruction is executed without checking the status of flags in the main program. CALL 16 bit memory address is the format for unconditional call instruction. After execution of this instruction, value of Program counter is updated with subroutine address and executes the subroutine. The value of the PC (Program Counter) is transferred to the memory stack and the value of SP (Stack Pointer) is decremented by two.

(b) Conditional Call instructions transfer the program control to the subroutine and the value of PC is pushed into stack only if the condition is satisfied.

(c) Unconditional Return instruction:RET is the instruction used to mark the end of the subroutine. It has no parameter. After execution of this instruction program control is transferred back to the main program from where it had stopped. The value of the PC (Program Counter) is retrieved from the memory stack and the value of SP (Stack Pointer) is incremented by 2.

(d) Conditional Return instruction:By these instructions program control is transferred back to the main program and the value of PC is popped from the stack only, if the condition is satisfied.

Different types of subroutine instructions are:-

INSTRUCTION	COMMENT
CC16-bit address	Call at address, if CY (carry flag) = 1
CNC16-bit address	Call at address, if CY (carry flag) = 0
CZ16-bit address	Call at address, if ZF (zero flag) = 1
CNZ16-bit address	Call at address, if ZF (zero flag) = 0

INSTRUCTION	COMMENT
CPE16-bit address	Call at address, if PF (parity flag) = 1
CPO16-bit address	Call at address, if PF (parity flag) = 0
CN16-bit address	Call at address, if SF (signed flag) = 1
CP16-bit address	Call at address, if SF (signed flag) = 0
RC	Return from subroutine if cy (carry flag) = 1
RNC	Return from subroutine if cy (carry flag) = 0
RZ	Return from subroutine if ZF (zero flag) = 1
RNZ	Return from subroutine if ZF (zero flag) = 0
RPE	Return from subroutine if PF (parity flag) = 1
RPO	Return from subroutine if PF (parity flag) = 0
RN	Return from subroutine if SF (signed flag) = 1
RP	Return from subroutine if SF (signed flag) = 0

8.6. BINARY TO ASCII CODE CONVERSION

Alphanumeric codes: A computer is a binary machine, to communicate with the computer in alphanumeric letters and decimal numbers, translation codes are necessary. The commonly used

code is Known as ASCII (American Standard Codes for Information Interchange). It is a 7-bit code with 128 combinations and each combination from 01H to 7FH is organized to represent the 26 letters of the English alphabet (both in lower and upper cases); numbers from 0 to 9; and various punctuation marks or machine command. For example.30H to 39H represents 0 to 9 in decimal , 41H to 5AH represents A to Z, 21H to 2FH represents various symbols and 61H to 7AH represents a to z.

Conversion of Binary number to ASCII Code

Step 1: Convert bit binary number into Hexadecimal equivalent

Step2: Find the ASCII Code from table

Example: find the ASCII text conversion of the following string of binary numbers:

01110111 01101111 01110010 01100100

Step 1: make the group of eight binary digits :01110111, 01101111, 01110010, 01100100

Step 2: Look up the Hexadecimal numbers 77H, 6FH, 72H, and 64H in the ASCII table.

77H, 6FH, 72H, and 64H represent w, o, r, and d, respectively. Therefore, the ASCII text for the given binary number is “word”.

8.7. BCD to binary code conversion

The users of the most microprocessor-based products demand the output display in decimal numbers. The decimal output is simple to understand and do not need any conversion. The data processing inside the microprocessor is performed in binary. Therefore, it is necessary to convert the binary results into their equivalent Binary Coded Decimal numbers before they are displayed.

Convert Binary number 11011010 into BCD Number

1. Convert the binary number 1101 1010 into decimal number	$=1*2^7+1*2^6+0*2^5+1*2^4+1*2^3+0*2^2+1*2^1+0*2^0$ $=128+64+0+16+8+0+2+0$ $=218$	
2. Identify the digit of decimal number	218 First digit 2, second digit 1, third digit 8	The conversion can be performed as follows: Step 1: If the number is less than 100, go to Step 2; otherwise, divide by 100 or subtract

3. Conversion of each digit into 4 bit binary number	2= 0010 1=0001 8=1000	100 repeatedly until the remainder is less than 100. The quotient is the MSB of BCD digit Step 2: If the number is less than 10, go to Step 3; otherwise, divide by 10 repeatedly until the remainder is less than 10. The quotient is Step 3: The remainder from Step 2 is ones digit in binary.
4. BCD number is	(0010 0001 1000)	

Example: Write a main program and a conversion subroutine to convert the binary number stored at memory address 6000H into its equivalent BCD number. Store the result from memory location 6100H onwards.

Source Program:

LDA 6000H : Get the binary number in accumulator

CALL SUBROUTINE : Call subroutine

HLT : Terminate program execution

Subroutine to convert binary number into its equivalent BCD number:

SUBROUTINE:

MVI B, 64H : Load divisor decimal 100 in B register

MVI C, 0AH : Load divisor decimal 10 in C register

MVI D, 00H : Initialize Digit 1 MSB Of BCD

MVI E, 00H : Initialize Digit 2; BCD2

STEP1: CMP B : Check if number < Decimal 100

JC STEP 2 : if yes go to step 2

SUB B : Subtract decimal 100

INR E : update quotient

JMP STEP 1 : go to step 1

STEP2: CMP C : Check if number < Decimal 10

JC STEP 3 : if yes go to step 3
SUB C : Subtract decimal 10
INR D : Update quotient
JMP STEP 2 : Continue division by 10
STEP3: STA 6100H : Store Digit 0
MOV A, D : Get Digit 1
STA 6101H : Store Digit 1
MOV A, E : Get Digit 2
STA 6102H : Store Digit 2

8.8. BCD to binary code conversion

The **Seven segment display** is mostly used for the digital display in calculators, digital counters, digital clocks, measuring instruments, etc. to display both the numbers and characters.

When a BCD number is to be displayed by a seven-segment LED, it is necessary to convert the BCD number to its seven-segment code. The code is determined by hardware considerations such as common-cathode or common-anode LED; the code has no direct relationship to binary numbers. Therefore, to display a BCD digit at a seven-segment LED, the **table look-up technique** is used. In the table look-up technique, the codes of the digits to be displayed are stored sequentially in memory. The conversion program locates the code of a digit based on its

A set of data having time in 24 hours format and room temperature for a day are stored in memory locations starting at 8000H, respectively. For example, at 1A.M., temperature is 20°C. The lookup tables for 0 to 9 digits for a common-cathode LED are stored in memory locations starting at 7000H, and the Output-Buffer memory is reserved at 9000H. Write a program to display time and temperature. Data stored in memory is in BCD format and of 2 digit number.

Program:

LXI H, 7000 H: Initialize lookup table pointer

LXI D, 8000 H: Initialize source memory pointer

LXI B, 9000 H: Initialize destination memory pointer

BACK: LDAX D: data is copied from memory location pointed by reg pair D,E to accumulator

ANI F0 H

RRC

RRC

RRC

RRC

MOV L, A: A point to the 7-segment code

MOV A, M: Get the 7-segment code

STAX B: Store the result at destination memory location

INX B

LDAX D

ANI 0F H

MOV L, A: A point to the 7-segment code

MOV A, M: Get the 7-segment code

STAX B: Store the result at destination memory location

INX D: Increment source memory pointer

INX B: Increment destination memory pointer

MOV A, C

CPI 30H: Check for last number

JNZ BACK: If not repeat

HLT: End of program

BCD Addition and Subtraction:

In 8085, input/output data are displayed in decimal numbers. If , this data requires addition or subtraction then it is beneficial to perform arithmetic operation directly in BCD codes. The addition of two BCD numbers may not represent an appropriate BCD value. So, if value is greater than 9, then add 6 to get the correct BCD digit.

Example, perform the addition of $(34)_{BCD}$ and $(36)_{BCD}$

	By Human	By Processor								
carry		0	1	1	0		1	0	0	0
Ist number	$(34)_{BCD}$	0	0	1	1		0	1	0	0
IInd Number	$(36)_{BCD}$	0	0	1	1		0	1	1	0
Sum	$(70)_{BCD}$	0	1	1	0		1	0	1	0
		6					A			

The microprocessor cannot differentiate between BCD and Binary numbers because both are string of zeroes and ones and it adds numbers in binary. So, there is difference in answer. $(70)_{BCD}$ by human and 6AH by processor. In BCD addition, any number larger than 9 (from A to F) is invalid and needs to be adjusted by adding 6 in binary.

The Hex number A can be adjusted as a BCD number by adding 6 in binary.

The BCD adjustment in an 8-bit binary register can be shown as follows:

$$\begin{array}{rcl}
 A & = & 0000\ 1010 \\
 +6 & = & 0000\ 0110 \\
 \text{Sum} & = & 0001\ 0000
 \end{array}$$

This carry is added to 2nd BCD digit and answer become $(70)_{BCD}$.

In this example, the carry is generated after the adjustment of the least significant four bits for the BCD digit and is again added to the adjustment of the most significant four bits.

A special instruction called DAA (Decimal Adjust Accumulator) performs the function of adjusting a BCD sum in the 8085 instruction set. This instruction uses the Auxiliary Carry flip flop (AC) to sense that the value of the least four bits is larger than 9 and adjusts the bits to the BCD value. Similarly, it uses the Carry flag (CY) to adjust the most significant four bits. However, the AC flag is used internally by the microprocessor; this flag is not available to the programmer through any Jump instruction.

INSTRUCTION

DAA: Decimal Adjust Accumulator

- This is a 1-byte instruction
- It adjusts an 8-bit number BCD sum.
- It uses the AC and the CY flags to perform the adjustment and all flags are affected
- It must be emphasized that instruction DAA
- It works only with addition when BCD numbers are used; does not work with subtraction.

BCD subtraction:

The subtraction is performed using by adding the 9's complement in BCD numbers. Hence,
 $A - B = A + [9\text{'s complement of } B]$

Following procedure is followed:

- IF RESULT > 9, Add 0110 to make valid BCD number
- If MOST SIGNIFICANT CARRY is produced then the result is positive and the end around carry must be added.
- IF MOST SIGNIFICANT CARRY is 0 [i.e. NO CARRY] then the result is negative and we get the 9's comp. of the result.

Example:

$$(a) \ 8 - 3 = 8 + [9\text{'s COMP. OF } 3] = 8 + 6$$

Ist number: 8		1000	
9's complement of: 3		0110	
Sum with invalid BCD		1110 > 1001 so add 0110	invalid BCD
	1	0100	End around carry, add carry to number
Result		0101	

$$(a) \ 38 - 93 = 8 + [9\text{'s COMP. OF } 3] = 8 + 6$$

Ist number: 38		0011 1000	
9's complement of 93 : 06		0000 0110	
Sum with invalid BCD		0011 1110 0110	invalid BCD so add 0110
	0	0100 0100	No carry, take 9's complement

Result	0	0101 0101	
--------	---	-----------	--

QUESTIONS:

1. Specify the number of times the following loops are executed.

- | | | |
|--|--|---|
| a. MVI A,17H
LOOP: ORAA
RAL
JNC LOOP | b. MVIA, 17H
LOOP: RAL
ORAA
JNC LOOP | c. LXI B,1000H
LOOP: DCX B
NOP
JNZ LOOP |
|--|--|---|

2. Calculate the COUNT to obtain a 100 μ s loop delay, and express the value in Hex.(Use the clock frequency of 4MHz.)

MVI B,COUNT

LOOP: NOP 4

NOP 4

DCRB 4

JNZ LOOP 10/7

3. Design an up-down counter to count from 0 to 9 and 9 to 0 continuously with a 1.5-second delay between each count, and display the count at one of the outputports. Draw a flowchart and show the delay calculations.
4. Write a program to turn a light on and off every 5 seconds. Use data bit D7 to operate the light.
5. Write a main program and a conversion subroutine to convert the binary number stored at Register H into its equivalent BCD number. Store the result from memory location in 6000H and 6001H.

6. Explain the functions of the following routines:

- | | |
|---|---|
| a. LXI SP,209FH
MVI C,00H
PUSH B
POP PSW
RET | b. LXI SP,STACK
PUSH B
PUSH D
POP B
POP D
RET |
|---|---|

7. A set of ASCII Hex digits is stored in the Input-Buffer memory. Write a program to convert these numbers into binary. Add these numbers in binary, and store the result in the Output-Buffer memory.
8. A set of ten BCD readings is stored in the Input Buffer. Convert the numbers into binary and add the numbers. Store the sum in the Output Buffer; the sum can be larger than FFH.
9. Write a program to add 2-BCD numbers where starting address is **2000** and the numbers are stored at **2500** and **2501** memory addresses and store sum into **2502** and carry into **2503** memory address.

*******End of chapter*******

UNIT 9

ADVANCED MICROPROCESSOR

Structure

9.1 Introduction

9.2. Objectives

9.3. Intel 8255

9.3.1. Applications

9.3.2. Function

9.3.3. Operational Modes Of 8255

9.3.3.1. Bit Set/Reset (Bsr) Mode

9.3.3.1.2. Mode 0 - Simple I/O

9.3.3.1.3. Mode 0 – Input Mode

9.3.3.1.4. Mode 0 - Output Mode

9.3.1.5. Mode 1 - Strobed Input/Output Mode

9.3.1.6. Input Handshaking Signals

9.3.1.7. Output Handshaking Signals

9.3.1.8. Mode 2 - Strobed Bidirectional Input/Output Mode

9.3.2. Internal Block Diagram Of 8255

9.3.3. Pins & Signals Of 8255

9.4. Intel 8259

9.4.1. Functional Description

9.4.2. Programming Considerations Dos And Windows

9.5. Intel 8086

9.5.1. The First X86 Design

9.6. Intel 80286

9.6.1. Architecture

9.6.1.1. Protected Mode

9.6.1.2. Support Components

9.7. Intel 80386 (I386)

9.7.1. Architecture

9.7.2. Data Types

9.8. Architecture Of 80486 Microprocessor

9.9. Pentium

9.9.1. Pentium-Branded Processors

9.9.1.1. P5 Microarchitecture Based

9.9.1.2. P6 Microarchitecture Based

9.9.1.3. Pentium Pro

9.9.1.4. Pentium Ii

9.9.1.5. Pentium Iii

9.9.1.6. Pentium 4

9.10 Summary

9.11 Glossary

9.12 References

9.13 Suggested Readings

9.14 Terminal Questions

9.14.1 Short Answer Type

9.1. INTRODUCTION

The **Intel 8255** (or **i8255**) Programmable Peripheral Interface (PPI) chip was developed and manufactured by Intel in the first half of the 1970s for the Intel 8080 microprocessor.

The Intel 8259 is a Programmable Interrupt Controller (PIC) designed for the Intel 8085 and Intel 8086 microprocessors. The initial part was 8259, a later A suffix version was upward compatible and usable with the 8086 or 8088 processor.

The 8259 may be configured to work with an 8080/8085 or an 8086/8088. On the 8086/8088, the interrupt controller will provide an interrupt number on the data bus when an interrupt occurs. The interrupt cycle of the 8080/8085 will issue three bytes on the data bus (corresponding to a CALL instruction in the 8080/8085 instruction set).

The 8086 was sequenced using a mixture of random logic and microcode and was implemented using depletion-load nMOS circuitry with approximately 20,000 active transistor (29,000 counting all ROM and PLA sites). It was soon moved to a new refined nMOS manufacturing process called HMOS (for High performance MOS) that Intel originally developed for manufacturing of fast static RAM products. This was followed by HMOS-II, HMOS-III versions, and, eventually, a fully static CMOS version for battery powered devices, manufactured using Intel's CHMOS processes. The original chip measured 33 mm² and The performance increase of the 80286 over the 8086 (or 8088) could be more than 100% per clock cycle in many programs (i.e., a doubled performance at the same clock speed).minimum feature size was 3.2 μm.

Pentium is a brand used for a series of x86 architecture compatible microprocessors produced by Intel. The original Pentium was released in 1993. After that, the Pentium II and Pentium III were released. In the case of Atom architectures, Pentiums are the highest performance implementations of the architecture. Pentium processors with Core architectures prior to 2017 were distinguished from the faster, higher-end i-series processors by lower clock rates and disabling some features, such as hyper threading, virtualization and sometimes L3 cache.

9.2. OBJECTIVES

After studying this unit, you will learn about-

- Intel 8255
- Operational Modes Of 8255

- Intel 8259
- Intel 80286
- Intel 80386 (I386)
- Pentium

9.3. INTEL 8255

The **Intel 8255** (or **i8255**) Programmable Peripheral Interface (PPI) chip was developed and manufactured by Intel in the first half of the 1970s for the Intel 8080 microprocessor. The 8255 provides 24 parallel input/output lines with a variety of programmable operating modes.

The 8255 is a member of the MCS -85 Family of chips, designed by Intel for use with their 8085 and 8086 microprocessors and their descendants. It was first available in a 40-pin DIP and later a 44-pin PLCC packages. It found wide applicability in digital processing systems and was later cloned by other manufacturers. The 82C55 is a CMOS version for higher speed and lower current consumption.

The functionality of the 8255 is now mostly embedded in larger VLSI processing chips as a sub-function. A CMOS version of the 8255 is still being made by Renesas but mostly used to expand the I/O of microcontrollers.

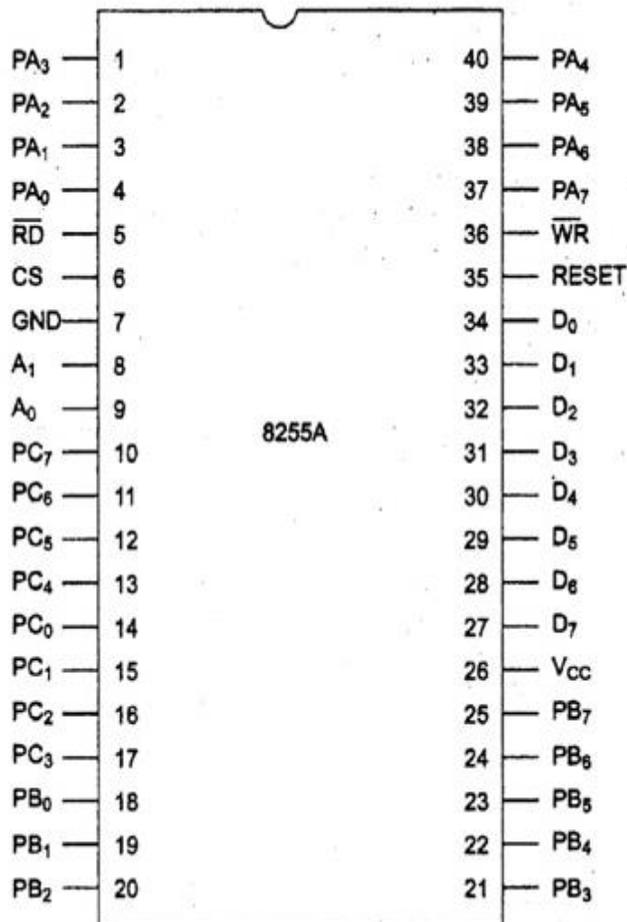


FIG. 9.1: PINOUT OF I8255

9.3.1. APPLICATIONS

The 8255 was widely used in many microcomputer/microcontroller systems and home computers such as the SV-328 and all MSX models. The 8255 was used in the original IBM - PC, PC/XT, PC/jr and clones, along with numerous homebuilt computers such as the N8VEM.

9.3.2. FUNCTION

The 8255 gives a CPU or digital system access to programmable parallel I/O. The 8255 has 24 input/output pins. These are divided into three 8-bit ports (A, B, C). Port A and port B can be used as 8-bit input/output ports. Port C can be used as an 8-bit input/output port or as two 4-bit input/output ports or to produce handshake signals for ports A and B.

The three ports are further grouped as follows:

1. Group A consisting of port A and upper part of port C.
2. Group B consisting of port B and lower part of port C.

Eight data lines (D0–D7) are available (with an 8-bit data buffer) to read/write data into the ports or control register under the status of the **RD** (pin 5) and **WR** (pin 36), which are active-low signals for read and write operations respectively. Address lines A₁ and A₀ allow to access a data register for each port or a control register, as listed below:

A ₁	A ₀	Port selected
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Control Register

The control signal chip select **CS** (pin 6) is used to enable the 8255 chip. It is an active-low signal, i.e., when CS = 0, the 8255 is enabled. The **RESET** input (pin 35) is connected to the RESET line of system like 8085, 8086, etc., so that when the system is reset, all the ports are initialized as input lines. This is done to prevent 8255 and/or any peripheral connected to it from being destroyed due to mismatch of port direction settings. As an example, consider an input device connected to 8255 at port A. If from the previous operation, port A is initialized as an output port and if 8255 is not reset before using the current configuration, then there is a possibility of damage of either the input device connected or 8255 or both, since both 8255 and the device connected will be sending out data.

The control register (or the control logic, or the command word register) is an 8-bit register used to select the modes of operation and input/output designation of the ports.

9.3.3. OPERATIONAL MODES OF 8255

There are two basic operational modes of 8255

- Bit Set/Reset mode (BSR mode).
- Input/Output mode (I/O mode).

The two modes are selected on the basis of the value present at the D_7 bit of the control word register. When $D_7 = 1$, 8255 operates in I/O mode, and when $D_7 = 0$, it operates in the BSR mode.

9.3.3.1. BIT SET/RESET (BSR) MODE

The Bit Set/Reset (BSR) mode is available on port C only. Each line of port C ($PC_7 - PC_0$) can be set or reset by writing a suitable value to the control word register. BSR mode and I/O mode are independent and selection of BSR mode does not affect the operation of other ports in I/O mode.

- D_7 bit is always 0 for BSR mode.
- Bits D_6 , D_5 and D_4 are don't care bits.
- Bits D_3 , D_2 and D_1 are used to select the pin of Port C.
- Bit D_0 is used to set/reset the selected pin of Port C.

Selection of port C pin is determined as follows:

D3	D2	D1	Bit/pin of port C selecte
0	0	0	PC
0	0	1	PC1
0	1	0	PC2
0	1	1	PC3

1	1	0	PC6
1	1	1	PC7

As an example, if it is needed that PC₅ be set, then in the control word,

1. Since it is BSR mode, **D₇ = '0'**.
2. Since D₄, D₅, D₆ are not used, assume them to be '0'.
3. PC₅ has to be selected, hence, **D₃ = '1', D₂ = '0', D₁ = '1'**.
4. PC₅ has to be set, hence, **D₀ = '1'**.

Thus, as per the above values, 0B (Hex) will be loaded into the Control Word Register (CWR).

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	0	1	1

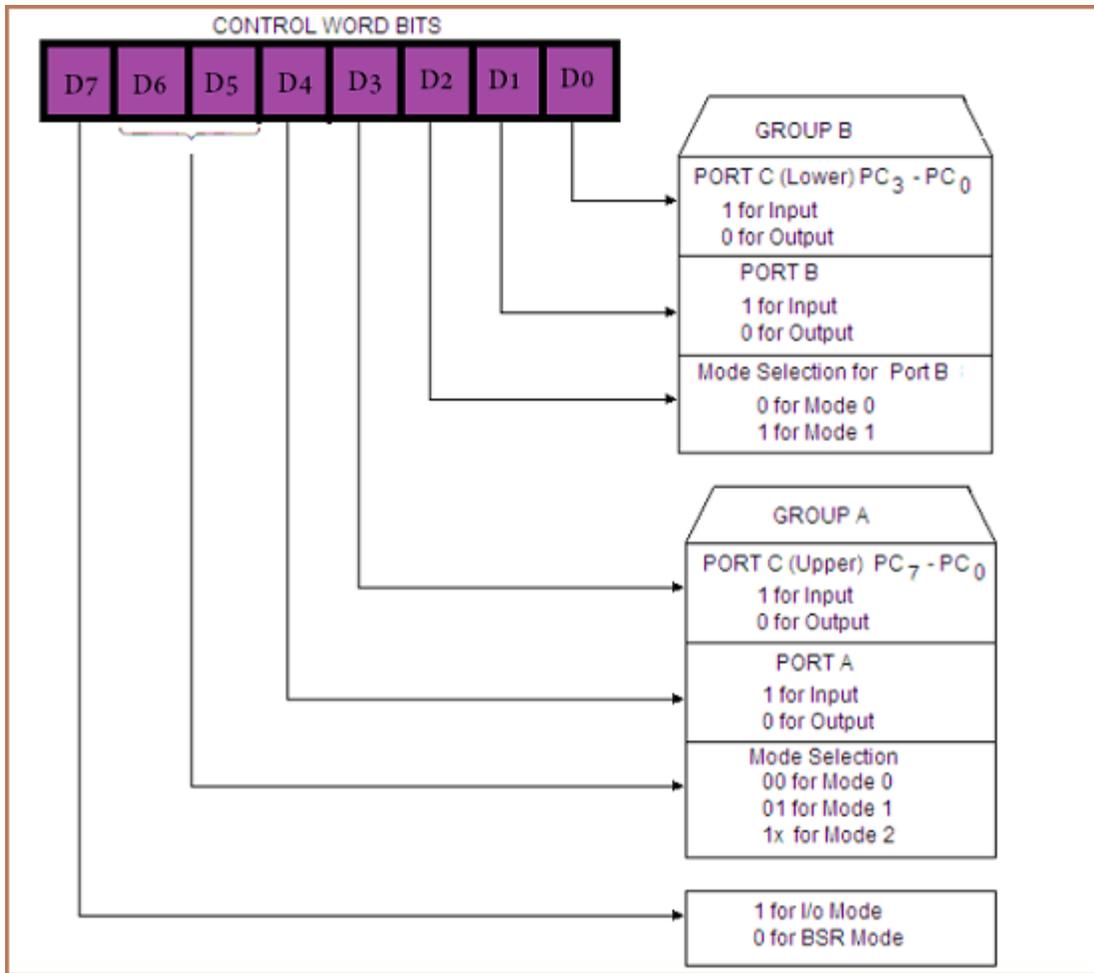


FIG. 9.2: 8255 control word for I/O MODE

9.3.3.1.2. MODE 0 - SIMPLE I/O

In this mode, the ports can be used for simple I/O operations without handshaking signals. Port A, port B provide simple I/O operation. The two halves of port C can be either used together as an additional 8-bit port, or they can be used as individual 4-bit ports. Since the two halves of port C are independent, they may be used such that one-half is initialized as an input port while the other half is initialized as an output port.

The input/output features in mode 0 are as follows:

1. Output ports are latched.
2. Input ports are buffered, not latched.
3. Ports do not have handshake or interrupt capability.
4. With 4 ports, 16 different combinations of I/O are possible.

'Latched' means the bits are put into a storage register (array of flip-flops) which holds its output constant even if the inputs change after being latched.

The 8255's outputs are latched to hold the last data written to them. This is required because the data only stays on the bus for one cycle. So, without latching, the outputs would become invalid as soon as the write cycle finishes.

The inputs are not latched because the CPU only has to read their current values, then store the data in a CPU register or memory if it needs to be referenced at a later time. If an input changes while the port is being read then the result may be indeterminate.

9.3.3.1.3. MODE 0 – INPUT MODE

- In the input mode, the 8255 gets data from the external peripheral ports and the CPU reads the received data via its data bus.
- The CPU first selects the 8255 chip by making CS low. Then it selects the desired port using A_0 and A_1 lines.
- The CPU then issues an RD signal to read the data from the external peripheral device via the system data bus.

9.3.3.1.4. MODE 0 - OUTPUT MODE

- In the output mode, the CPU sends data to 8255 via system data bus and then the external peripheral ports receive this data via 8255 port.
- CPU first selects the 8255 chip by making CS low. It then selects the desired port using A_0 and A_1 lines.
- CPU then issues a WR signal to write data to the selected port via the system data bus. This data is then received by the external peripheral device connected to the selected port.

9.3.1.5. MODE 1 - STROBED INPUT/OUTPUT MODE

When we wish to use port A or port B for handshake (strobed) input or output operation, we initialize that port in mode 1 (port A and port B can be initialized to operate in different modes, i.e., for e.g., port A can operate in mode 0 and port B in mode 1). Some of the pins of port C function as handshake lines.

For port B in this mode (irrespective of whether is acting as an input port or output port), PC0, PC1 and PC2 pins function as handshake lines.

If port A is initialized as mode 1 input port, then, PC3, PC4 and PC5 function as handshake signals. Pins PC6 and PC7 are available for use as input/output lines.

The mode 1 which supports handshaking has following features:

1. Two ports i.e. port A and B can be used as 8-bit i/o ports.
2. Each port uses three lines of port c as handshake signal and remaining two signals can be used as I/O ports.
3. Interrupt logic is supported.
4. Input and Output data are latched.

9.3.1.6. INPUT HANDSHAKING SIGNALS

1. IBF (Input Buffer Full) - It is an output indicating that the input latch contains information.
2. STB (Strobed Input) - The strobe input loads data into the port latch, which holds the information until it is input to the microprocessor via the IN instruction.
3. INTR (Interrupt request) - It is an output that requests can interrupt. The INTR pin becomes logic 1 when the STB input returns to a logic 1, and is cleared when the data are input from the port by the microprocessor.
4. INTE (Interrupt enable) - It is neither an input nor an output; it is an internal bit programmed via the port PC4 (port A) or PC2 (port B) bit position.

9.3.1.7. OUTPUT HANDSHAKING SIGNALS

1. OBF (Output Buffer Full) - It is an output that goes low whenever data are output(OUT) to the port A or port B latch. This signal is set to logic 1 whenever the ACK pulse returns from the external device.
2. ACK (Acknowledge)-It causes the OBF pin to return to a logic 1 level. The ACK signal is a response from an external device, indicating that it has received the data from the 82C55A port.
3. INTR (Interrupt request) - It is a signal that often interrupts the microprocessor when the external device receives the data via the signal. this pin is qualified by the internal INTE(interrupt enable) bit.
4. INTE (Interrupt enable) - It is neither an input nor an output; it is an internal bit programmed to enable or disable the INTR pin. The INTE A bit is programmed using the PC6 bit and INTE B is programmed using the PC2 bit.

9.3.1.8. MODE 2 - STROBED BIDIRECTIONAL INPUT/OUTPUT MODE

Only port A can be initialized in this mode. Port A can be used for bidirectional handshake data transfer. This means that data can be input or output on the same eight lines (PA0 - PA7). Pins PC3 - PC7 are used as handshake lines for port A. The remaining pins of port C (PC0 - PC2) can be used as input/output lines if group B is initialized in mode 0 or as handshaking for port B if group B is initialized in mode 1. In this mode, the 8255 may be used to extend the system bus to a slave microprocessor or to transfer data bytes to and from a floppy disk controller. Acknowledgement and handshaking signals are provided to maintain proper data flow and synchronisation between the data transmitter and receiver.

9.3.2. INTERNAL BLOCK DIAGRAM OF 8255

The ports are grouped as Group A and Group B. The group A has port A, port C upper and its control circuit. The group B has port B, port C lower and its control circuit. The Read/Write control logic requires six control signals. These signals are given below.

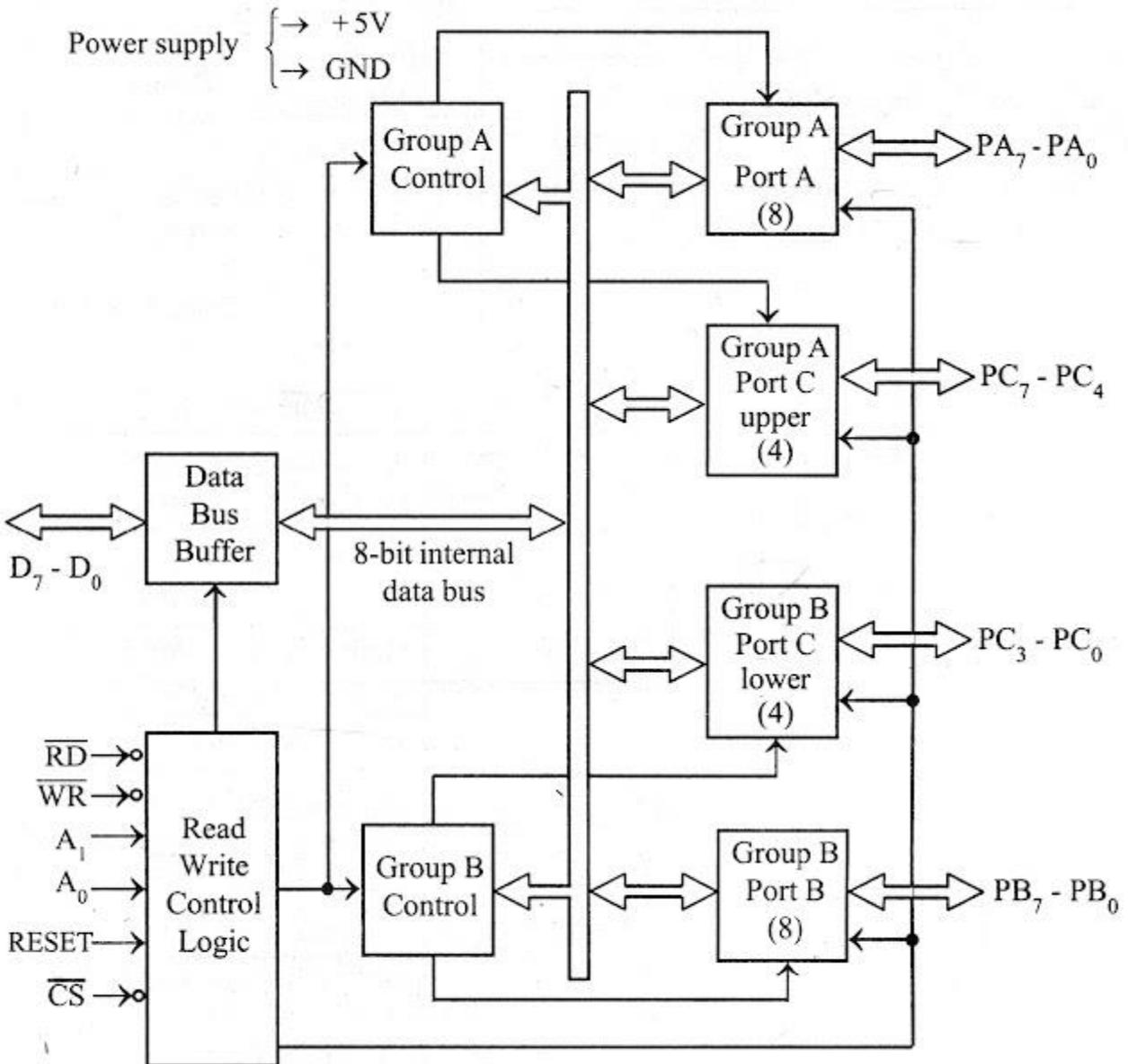


FIG. 9.3: INTERNAL BLOCK DIAGRAM

RD (Read): This control signal enables the read operation. When this signal is LOW, the microprocessor reads data from a selected I/O port of the 8255A.

RD (Read): This control signal enables the read operation. When this signal is LOW, the microprocessor reads data from a selected I/O port of the 8255A.

WR (Write): This control signal enables the write operation. When this signal goes LOW, the microprocessor writes into a selected I/O port or the control register.

RESET: This is an active HIGH signal. It clears the control register and set all ports in the input mode.

CS, A0 and A1: These are device select signals. The CS is connected to the decoder in the system. A0 and A1 are generally connected to A0 and A1 of the processor. (Alternatively, A0 and A1 can be connected to any two-address lines of the processor). 8255 can be either Memory mapped or I/O mapped in the system. A0 and A1 address lines can be made to select any one of the following four internal devices as shown on right side.

INTERNAL ADDRESS		DEVICE SELECTED
A1	A0	
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Control Register

9.3.3. PINS & SIGNALS OF 8255

The pin description of 8255 is shown in figure below. It has 40 pins and requires a single +5V supply.

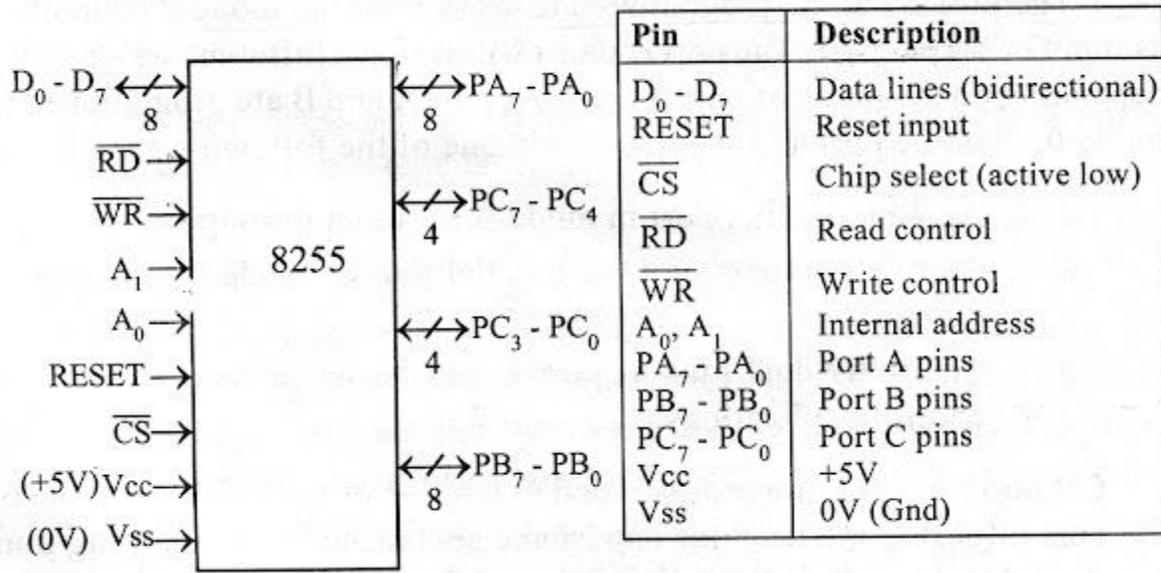


FIG. 9.4: PIN AND SIGNAL DIAGRAM

9.4. INTEL 8259

The Intel 8259 is a Programmable Interrupt Controller (PIC) designed for the Intel 8085 and Intel 8086 microprocessors. The initial part was 8259, a later A suffix version was upward compatible and usable with the 8086 or 8088 processor. The 8259 combines multiple interrupt input sources into a single interrupt output to the host microprocessor, extending the interrupt levels available in a system beyond the one or two levels found on the processor chip. The 8259A was the interrupt controller for the ISA bus in the original IBM PC and IBM PC AT.

The 8259 was introduced as part of Intel's MCS 85 family in 1976. The 8259A was included in the original PC introduced in 1981 and maintained by the PC/XT when introduced in 1983. A second 8259A was added with the introduction of the PC/AT. The 8259 has coexisted with the Intel APIC Architecture since its introduction in Symmetric Multi – Processor PCs. Modern PCs have begun to phase out the 8259A in favor of the Intel APIC Architecture. However, while not

anymore a separate chip, the 8259A interface is still provided by the Platform Controller Hub or Southbridge chipset on modern x86 motherboards.

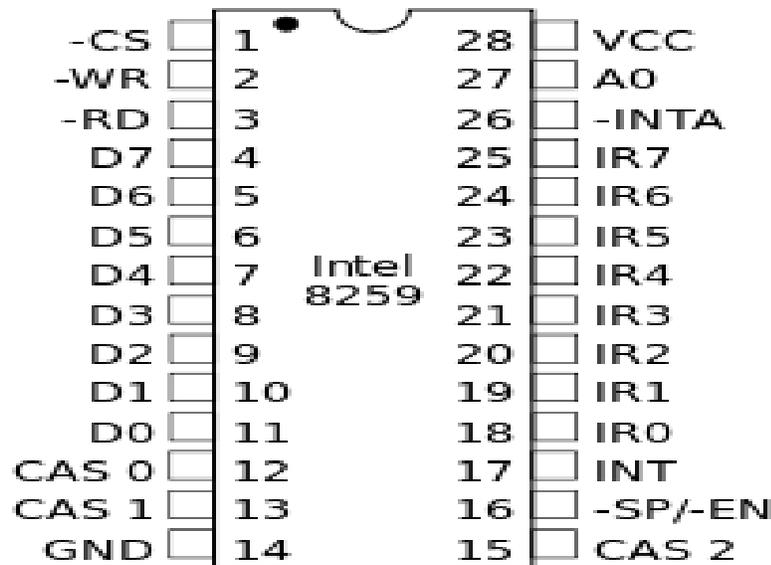


FIG. 9.5: PINOUT

9.4.1. FUNCTIONAL DESCRIPTION

The main signal pins on an 8259 are as follows: eight interrupt input request lines named IRQ0 through IRQ7, an interrupt request output line named INTR, interrupt acknowledgment line named INTA, D0 through D7 for communicating the interrupt level or vector offset. Other connections include CAS0 through CAS2 for cascading between 8259s.

Up to eight slave 8259s may be cascaded to a master 8259 to provide up to 64 IRQs. 8259s are cascaded by connecting the INT line of one slave 8259 to the IRQ line of one master 8259.

End of interrupt (EOI) operations support specific EOI, non-specific EOI, and auto-EOI. A specific EOI specifies the IRQ level it is acknowledging in the ISR. A non-specific EOI resets the IRQ level in the ISR. Auto-EOI resets the IRQ level in the ISR immediately after the interrupt is acknowledged.

Edge and level interrupt trigger modes are supported by the 8259A. Fixed priority and rotating priority modes are supported.

The 8259 may be configured to work with an 8080/8085 or an 8086/8088. On the 8086/8088, the interrupt controller will provide an interrupt number on the data bus when an interrupt occurs. The interrupt cycle of the 8080/8085 will issue three bytes on the data bus (corresponding to a CALL instruction in the 8080/8085 instruction set).

The 8259A provides additional functionality compared to the 8259 (in particular buffered mode and level-triggered mode) and is upward compatible with it.

9.4.2. PROGRAMMING CONSIDERATIONS DOS AND WINDOWS

Programming an 8259 in conjunction with DOS and Microsoft Windows has introduced a number of confusing issues for the sake of backwards compatibility, which extends as far back as the original PC introduced in 1981.

The first issue is more or less the root of the second issue. DOS device drivers are expected to send a non-specific EOI to the 8259s when they finish servicing their device. This prevents the use of any of the 8259's other EOI modes in DOS, and excludes the differentiation between device interrupts rerouted from the master 8259 to the slave 8259.

The second issue deals with the use of IRQ2 and IRQ9 from the introduction of a slave 8259 in the PC/AT. The slave 8259's INT output is connected to the master's IR2. The IRQ2 line of the ISA bus, originally connected to this IR2, was rerouted to IR1 of the slave. Thus the old IRQ2 line now generates IRQ9 in the CPU. To allow backwards compatibility with DOS device drivers that still set up for IRQ2, a handler is installed by the BIOS for IRQ9 that redirects interrupts to the original IRQ2 handler.

On the PC, the BIOS (and thus also DOS) traditionally maps the master 8259 interrupt requests (IRQ0-IRQ7) to interrupt vector offset 8 (INT08-INT0F) and the slave 8259 (in PC/AT and later) interrupt requests (IRQ8-IRQ15) to interrupt vector offset 112 (INT70-INT77). This was done despite the first 32 (INT00-INT1F) interrupt vectors being reserved by the processor for internal exceptions (this was ignored for the design of the PC for some reason). Because of the reserved vectors for exceptions most other operating systems map (at least the master) 8259 IRQs (if used on a platform) to another interrupt vector base offset.

9.5. INTEL 8086

The **8086** (also called **iAPX 86**) is a 16-bit microprocessor chip designed by Intel between early 1976 and June 8, 1978, when it was released. The Intel 8088, released July 1, 1979, is a slightly modified chip with an external 8-bit data bus (allowing the use of cheaper and fewer supporting ICs), and is notable as the processor used in the original IBM PC design.

The 8086 gave rise to the x86 architecture, which eventually became Intel's most successful line of processors. On June 5, 2018, Intel released a limited-edition CPU celebrating the 40th anniversary of the Intel 8086, called the Intel Core i7-8086K.

In 1972, Intel launched the 8008, the first 8-bit microprocessor. It implemented an instruction set designed by Datapoint Corporation with programmable CRT terminals in mind, which also proved to be fairly general-purpose. The device needed several additional ICs to produce a functional computer, in part due to it being packaged in a small 18-pin "memory package", which ruled out the use of a separate address bus (Intel was primarily a DRAM manufacturer at the time).

Two years later, Intel launched the 8080, employing the new 40-pin DIL packages originally developed for calculator ICs to enable a separate address bus. It has an extended instruction set that is source-compatible with the 8008 and also includes some 16-bit instructions to make programming easier. The 8080 device was eventually replaced by the depletion-load based 8085 (1977), which sufficed with a single +5 V power supply instead of the three different operating voltages of earlier chips. Other well known 8-bit microprocessors that emerged during these years are Motorola 6800 (1974), General Instruction PIC 16X (1975), MOS Technology 6502 (1975), Zilog Z80 (1976), and Motorola 6809 (1978).

9.5.1. THE FIRST X86 DESIGN

The 8086 project started in May 1976 and was originally intended as a temporary substitute for the ambitious and delayed iAPX 432 project. It was an attempt to draw attention from the less-delayed 16-bit and 32-bit processors of other manufacturers Motorola, Zilog and National Semiconductor.

Whereas the 8086 was a 16-bit microprocessor, it used the same micro architecture as Intel's 8-bit microprocessors (8008, 8080, and 8085). This allowed assembly language programs written in 8-

bit to seamlessly migrate. New instructions and features such as signed integers, base offset addressing, and self-repeating operations were added. Instructions were added to assist source code compilation of nested functions in the ALGOL family of languages, including Pascal and PL/M. According to principal architect this was a result of a more software-centric approach. Other enhancements included microcode instructions for the multiply and divide assembly language instructions. Designers also anticipated coprocessors, such as 8087 and 8089, so the bus structure was designed to be flexible.

The first revision of the instruction set and high level architecture was ready after about three months, and as almost no CAD tools were used, four engineers and 12 layout people were simultaneously working on the chip. The 8086 took a little more than two years from idea to working product, which was considered rather fast for a complex design in 1976–1978.

The 8086 was sequenced using a mixture of random logic and microcode and was implemented using depletion-load nMOS circuitry with approximately 20,000 active transistor (29,000 counting all ROM and PLA sites). It was soon moved to a new refined nMOS manufacturing process called HMOS (for High performance MOS) that Intel originally developed for manufacturing of fast static RAM products. This was followed by HMOS-II, HMOS-III versions, and, eventually, a fully static CMOS version for battery powered devices, manufactured using Intel's CHMOS processes. The original chip measured 33 mm² and minimum feature size was 3.2 μm.

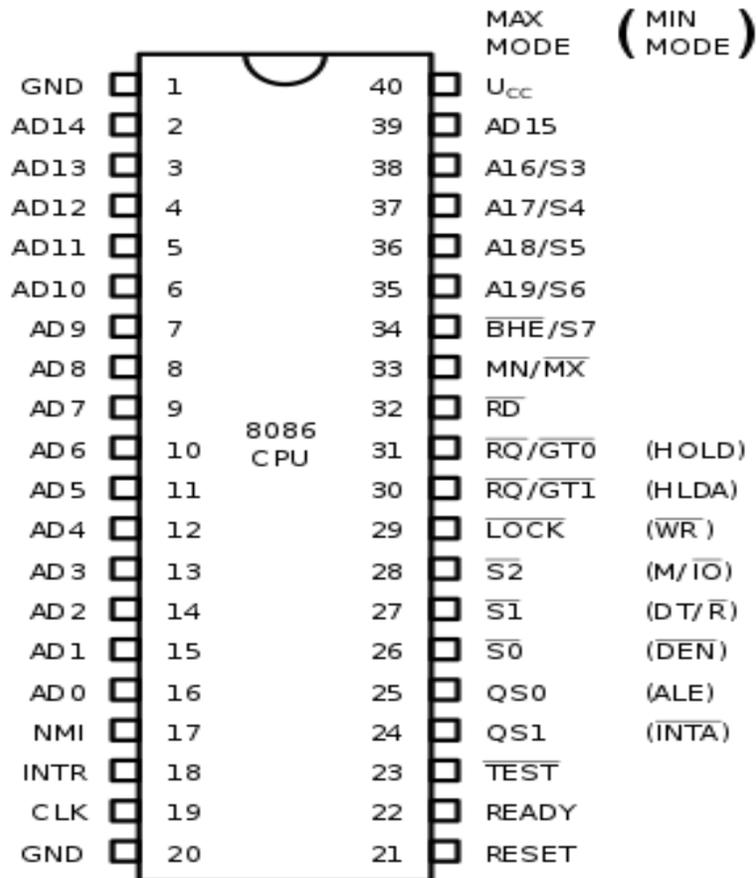


FIG. 9.6: THE 8086 PIN

9.6. INTEL 80286

The Intel 80286 (also marketed as the **iAPX 286** and often called **Intel 286**) is a 16-bit microprocessor that was introduced on February 1, 1982. It was the first 8086-based CPU with separate, non-multiplexed address and data buses and also the first with memory management and wide protection abilities. The 80286 used approximately 134,000 transistors in its original nMOS incarnation and, just like the contemporary 80186, it could correctly execute most software written for the earlier Intel 8086 and 8088 processors.

The 80286 was employed for the IBM PC/AT, introduced in 1984, and then widely used in most PC/AT compatible computers until the early 1990s.

9.6.1. ARCHITECTURE

Intel did not expect personal computers to use the 286. The CPU was designed for multi user systems with multi asking applications, including communications (such as automated PBXS) and real time control. It had 134,000 transistors and consisted of four independent units: the address unit, bus unit, instruction unit, and execution unit, organized into a loosely coupled (buffered) pipeline, just as in the 8086. It was produced in a 68-pin package, including PLCC (plastic leaded chip carrier), LCC (leadless chip carrier) and PGA (pin grid array) packages.

The performance increase of the 80286 over the 8086 (or 8088) could be more than 100% per clock cycle in many programs (i.e., a doubled performance at the same clock speed). This was a large increase, fully comparable to the speed improvements seven years later when the i486 (1989) or the original Pentium (1993) were introduced. This was partly due to the non-multiplexed address and data buses, but mainly to the fact that address calculations were less expensive. They were performed by a dedicated unit in the 80286, while the older 8086 had to do effective address computation using its general ALU, consuming several extra clock cycles in many cases. Also, the 80286 was more efficient in the pre fetch of instructions, buffering, execution of jumps, and in complex micro coded numerical operations such as MUL/DIV than its predecessor.

The 80286 included, in addition to all of the 8086 instructions, all of the new instructions of the 80186: ENTER, LEAVE, BOUND, INS, OUTS, PUSHA, POPA, PUSH immediate, IMUL immediate, and immediate shifts and rotates. The 80286 also added new instructions for protected mode: ARPL, CLTS, LAR, LGDT, LIDT, LLDT, LMSW, LSL, LTR, SGDT, SIDT, SLDT, SMSW, STR, VERR, and VERW. Some of the instructions for protected mode can (or must) be used in real mode to set up and switch to protected mode, and a few (such as SMSW and LMSW) are useful for real mode itself.

The Intel 80286 had a 24-bit address bus and as such had a 16 MB physical address space, compared to the 1 MB address space of prior x86 processors. It was the first x86 processor to support virtual memory supporting up to 1 GB via segmentation. However, memory cost and the initial rarity of software using the memory above 1 MB meant that until late in its production 80286 computers rarely shipped with more than one megabyte of RAM. Additionally, there was a performance penalty involved in accessing extended memory from real mode as noted below.

9.6.1.1. PROTECTED MODE

The 286 was the first of the x 86 CPU families to support protected virtual-address mode, commonly called "protected mode". In addition, it was the first commercially available microprocessor with on-chip MMU capabilities (systems using the contemporaneous Motorola 68010 and NS320xx could be equipped with an optional MMU controller). This would allow IBM compatibles to have advanced multitasking OSes for the first time and compete in the Unix dominated server/ workstation market. Several additional instructions were introduced in the protected mode of 80286, which are helpful for multitasking operating systems.

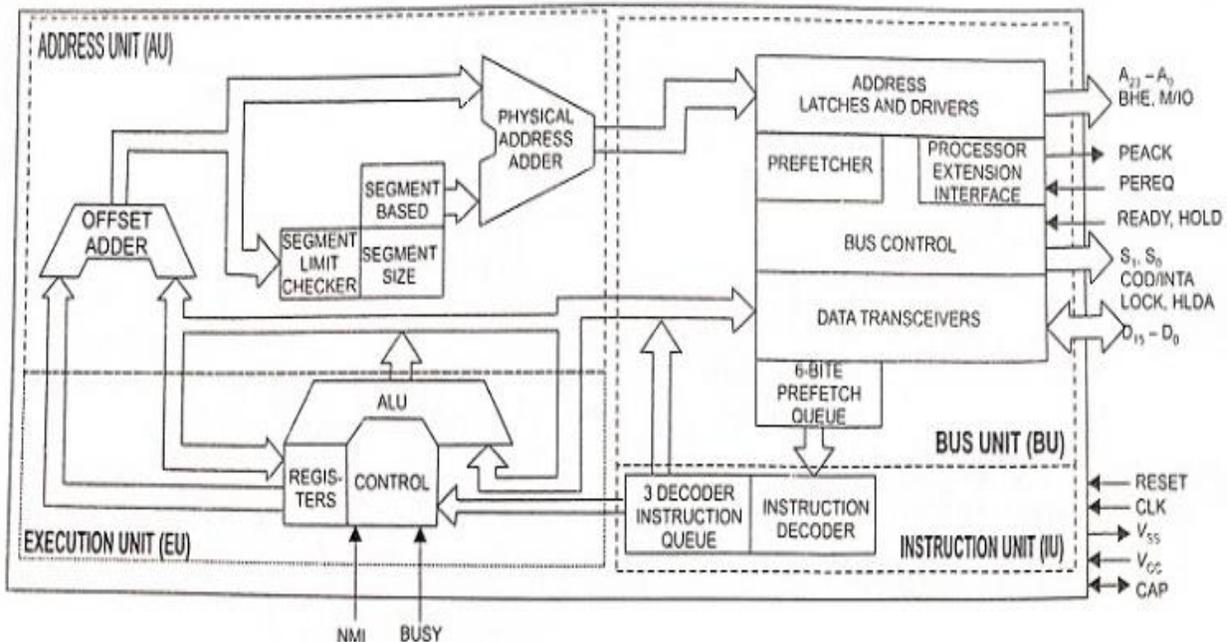


FIG.9.7: INTERNAL BLOCK DIAGRAM OF 80286 MICROPROCESSOR

Another important feature of 80286 is the prevention of unauthorized access. This is achieved by:

- Forming different segments for data, code, and stack, and preventing their overlapping.
- Assigning privilege levels to each segment. Segments with lower privilege levels cannot access segments with higher privilege levels.

In 80286 (and in its co-processor Intel 80287), arithmetic operations can be performed on the following different types of numbers:

- Unsigned packed decimal
- unsigned binary,
- unsigned unpacked decimal,
- signed binary,
- floating point number

By design, the 286 could not revert from protected mode to the basic 8086-compatible real address mode without a hardware-initiated reset. In the PC/AT introduced in 1984, IBM added external circuitry, as well as specialized code in the ROM BIOS and the 8042 peripheral microcontroller to enable software to cause the reset, allowing real-mode reentry while retaining active memory and returning control to the program that initiated the reset. (The BIOS is necessarily involved because it obtains control directly whenever the CPU resets.) Though it worked correctly, the method imposed a huge performance penalty.

In theory, real-mode applications could be directly executed in 16-bit protected mode if certain rules (newly proposed with the introduction of the 80286) were followed; however, as many DOS programs did not conform to those rules, protected mode was not widely used until the appearance of its successor, the 32-bit Intel 80386, which was designed to go back and forth between modes easily and to provide an emulation of real mode within protected mode. When Intel designed the 286, it was not designed to be able to multitask real-mode applications; real mode was intended to be a simple way for a bootstrap loader to prepare the system and then switch to protected mode; essentially, in protected mode the 80286 was designed to be a new processor with many similarities to its predecessors, while real mode on the 80286 was offered for smaller-scale systems that could benefit from a more advanced version of the 80186 CPU core, with advantages such as higher clock rates, faster instruction execution (measured in clock cycles), and unmultiplexed buses, but not the 24-bit (16 MB) memory space.

To support protected mode, new instructions have been added: ARPL, VERR, VERW, LAR, LSL, SMSW, SGDT, SIDT, SLDT, STR, LMSW, LGDT, LIDT, LLDT, LTR, CLTS. There are also new exceptions (internal interrupts): invalid opcode, coprocessor not available, double fault,

coprocessor segment overrun, stack fault, segment overrun/general protection fault, and others only for protected mode.

9.6.1.2. SUPPORT COMPONENTS

This list of bus interface components that connect to Intel 80286 microprocessor.

- 82258 Advanced Direct Memory Access Controller – Transfer rate of 8MB per second, supports up to 32 sub channels, mask and compare, verify, translation, and assembly/disassembly operation that are being processed simultaneously. It also support 16MB addressing range.
- 82C284 Clock Generator and Driver - Intel second sourced this 82284 version to Fujitsu Limited around 1985. The Intel branded chipset was available in 20-pin PLCC in sampling at first quarter 1986.
- 82288 Bus Controller
- 82289 Bus Arbiter

9.7. INTEL 80386 (i386)

The Intel **386**, originally released as **80386** and later renamed **i386**, is a 32-bit microprocessor introduced in 1985. The first versions had 275,000 transistors and were the CPU of many workstations and high-end personal computers of the time. As the original implementation of the 32-bit extension of the 80286 architecture, the i386 instruction set, programming model, and binary encodings are still the common denominator for all 32-bit x86 processors, which is termed the i386-architecture, x86, or IA -32 , depending on context.

The 32-bit i386 can correctly execute most code intended for the earlier 16-bit processors such as 8086 and 80286 that were ubiquitous in early PCs. (Following the same tradition, modern 64-bit x86 processors are able to run most programs written for older x86 CPUs, all the way back to the original 16-bit 8086 of 1978.) Over the years, successively newer implementations of the same architecture have become several hundreds of times faster than the original 80386 (and thousands of times faster than the 8086). A 33 MHz 80386 was reportedly measured to operate at about 11.4 MIPS.

Development of i386 technology began in 1982 under the internal name of P3. The tape out of the 80386 development was finalized on July 1985. The 80386 was introduced as pre-production samples for software development workstation in October 1985. Manufacturing of the chips in significant quantities commenced in June 1986, along with the first plug-in device that allowed existing 80286-based computers to be upgraded to the 386, the Translator 386 by American Computer and Peripheral. Main boards for 80386-based computer systems were cumbersome and expensive at first, but manufacturing was justified upon the 80386's mainstream adoption. In May 2006, Intel announced that i386 production would stop at the end of September 2007. Although it had long been obsolete as a personal computer CPU, Intel and others had continued making the chip for embedded systems. Such systems using an i386 or one of many derivatives are common in aerospace technology and electronic musical instruments, among others. Some mobile phones also used (later fully static CMOS variants of) the i386 processor, such as y 950 and Nokia 9000 communicator. Linux continued to support i386 processors until December 11, 2012; when the kernel cut 386-specific instructions in version 3.8.

9.7.1. ARCHITECTURE

The processor was a significant evolution in the x86 architecture, and extended a long line of processors that stretched back to the Intel 8008. The predecessor of the 80386 was the Intel 80286, a 16-bit processor with a segment-based memory management and protection system. The 80386 added a three-stage instruction pipeline which it brings up to total of 6-stage instruction pipeline, extended the architecture from 16-bit to 32-bit, and added an on-chip memory management unit. This paging translation unit made it much easier to implement operating systems that used virtual memory. It also offered support for register debugging.

The 80386 featured three operating modes: real mode, protected mode and virtual mode. The protected mode, which debuted in the 286, was extended to allow the 386 to address up to 4 GB of memory. With the addition of segmented addressing system, it can expand up to 64 terabytes of virtual memory. The all new virtual 8086 mode (or VM86) made it possible to run one or more real mode programs in a protected environment, although some programs were not compatible. It features scaled indexing and 64-bit barrel shifter.

The ability for a 386 to be set up to act like it had a flat memory model in protected mode despite the fact that it uses a segmented memory model in all modes was arguably the most important

feature change for the x86 processor family until AMD released x86. Several new instructions have been added to 386: BSF, BSR, BT, BTS, BTR, BTC, CDQ, CWDE, LFS, LGS, LSS, MOVSX, MOVZX, SETcc, SHLD, SHRD.

Two new segment registers have been added (FS and GS) for general-purpose programs, single Machine Status Word of 286 grew into eight control registers CR0–CR7. Debug registers DR0–DR7 were added for hardware breakpoints. New forms of MOV instruction are used to access them.

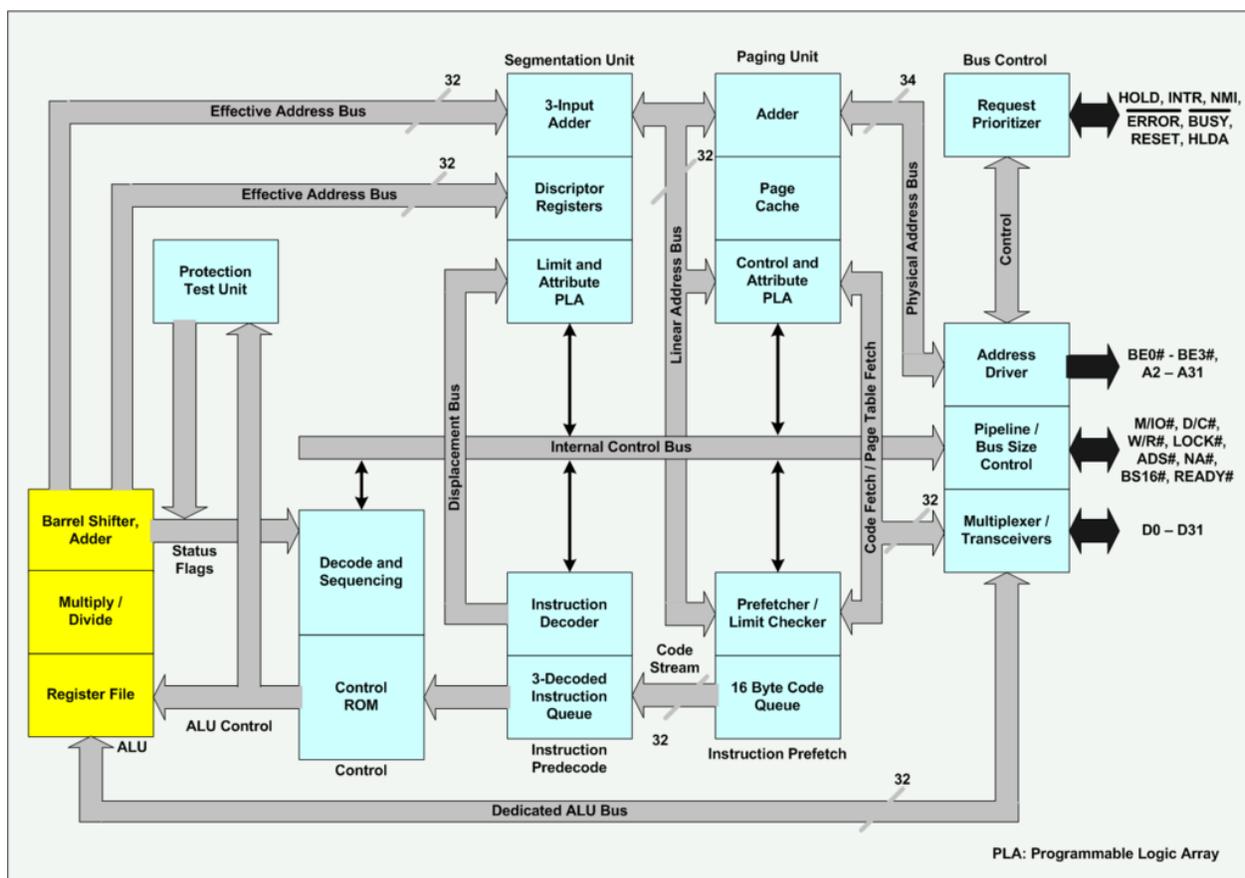


FIG. 9.8: ARCHITECTURE

9.7.2. DATA TYPES

The following data types are directly supported and thus implemented by one or more i386 machine instructions ; these data types are briefly described here.

- Bit (Boolean value), bit field (group of up to 32 bits) and bit string (up to 4 Gbit in length).
- 8-bit integer (byte), either signed (range $-128..127$) or unsigned (range $0..255$).
- 16-bit integer, either signed (range $-32,768..32,767$) or unsigned (range $0..65,535$).
- 32-bit integer, either signed (range $-2^{31}..2^{31}-1$) or unsigned (range $0..2^{32}-1$).
- Offset, a 16- or 32-bit displacement referring to a memory location (using any addressing mode).
- Pointer, a 16-bit selector together with a 16- or 32-bit offset.
- Character (8-bit character code).
- String, a sequence of 8-, 16- or 32-bit words (up to 4 Gbit in length).
- BCD, decimal digits (0..9) represented by unpacked bytes.
- Packed BCD, two BCD digits in one byte (range 0..99).

9.8. ARCHITECTURE OF 80486 MICROPROCESSOR

The 80486DX is a 32-bit processor. [Figure 11.46](#) shows the simplified block diagram of 80486 and the internal architecture of 80486 Microprocessor is depicted in [Fig. 11.47](#).

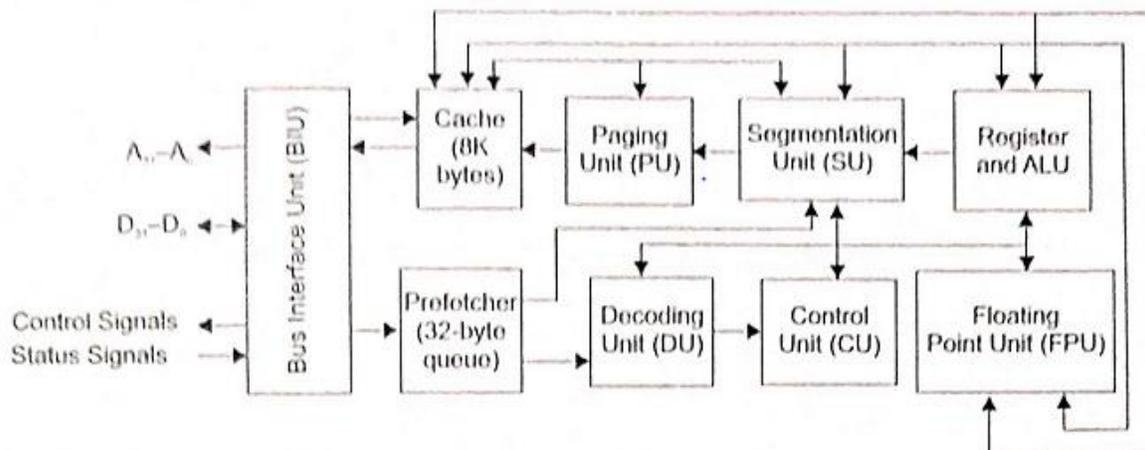


FIG. 9.9: SIMPLIFIED BLOCK DIAGRAM OF 80486

The architecture of Intel's 80486 can be divided into three different sections such as

- Bus interface unit (BIU),
- Execution and control unit (EU), and
- Floating-point unit (FU).

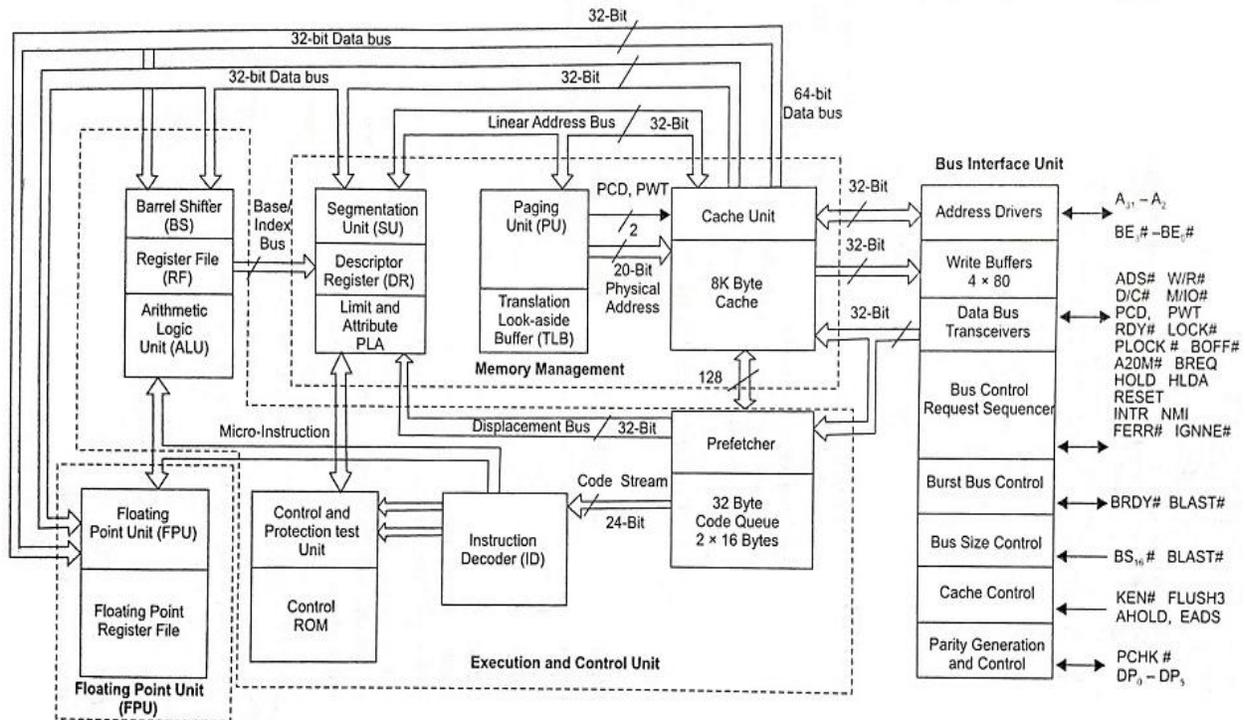


FIG. 9.10: INTERNAL ARCHITECTURE OF 80486 MICROPROCESSOR

Bus Interface Unit (BIU) The bus interface unit is used to organize all the bus activities of the processor. The address driver is connected with the internal 32-bit address output of the cache and the system bus. The data bus transceivers are interconnected between the internal 32-bit data bus and system bus. The write data buffer is a queue of four 80-bit registers and is able to hold the 80-bit data which will be written to the memory. Due to pipelined execution of the write operation, data must be available in advance. To control the bus access and operations, the following bus control and request sequencer signals \overline{ADS} , W/\overline{R} , D/\overline{C} , M/\overline{IO} , PCD , PWT , \overline{RDY} , \overline{LOCK} , \overline{PLOCK} , $\overline{BÖFF}$, $\overline{A20M}$, $BREQ$, $HOLD$, $HLDA$, $RESET$, $INTR$, NMI , \overline{FERR} and \overline{IGNNE} are used.

Execution Unit (EU) and Control Unit (CU) The burst control signal updates the processor that the burst is ready. This signal works as a ready signal in the burst cycle. The \overline{BLAST} output shows that the previous burst cycle is over. The bus size control signals $\overline{BS16}$ and $\overline{BS8}$ indicates dynamic bus sizing. The cache control signals \overline{KEN} , FLUSH, AHOLD and \overline{EADS} are used to control the cache control unit.

The parity generation and control unit generates the parity and carries out the checking during the processor operation. The boundary scan control unit of the processor performs boundary scan tests operation to ensure the correct operation of all components of the circuit on the mother board.

The pre fetcher unit fetches the codes from the memory and arranges them in a 32-byte code queue. The function of the instruction decoder is to receive the code from the code queue and then decodes the instruction code sequentially. The output of the decoder is fed to the control unit to derive the control signals, which are used for execution of the decoded instructions. Before execution, the protection unit should check all protection norms. If there is in any violation, an appropriate exception is generated.

The control ROM stores a microprogram to generate control signals for execution of instructions. The register bank and ALU are used for their usual operation just like they perform in 80286. The barrel shifter is used to perform the shift and rotate algorithms. The segmentation unit, descriptor registers, paging unit, translation look aside buffer and limit and attribute PLA are worked together for the virtual memory management. These units also provide protection to the op-codes or operand in the physical memory.

Floating point unit (FPU) The floating-point unit and register banks or FPU communicate with the bus interface unit (BIU) under the control of memory management unit (MMU), through a 64-bit internal data bus. Generally, the FPU is used for mathematical data processing at very high speed as compared to the ALU.

9.9. PENTIUM

Pentium is a brand used for a series of x86 architecture compatible microprocessors produced by Intel. The original Pentium was released in 1993. After that, the Pentium II and Pentium III were released.

In their form as of March 2022, Pentium processors are considered entry-level products that Intel rates as "two stars", meaning that they are above the low-end Atom and Celeron series, but below the faster Intel core lineup, and workstation/server Xenon series.

As of 2017, Pentium processors have little more than their name in common with earlier Pentiums, which were Intel's flagship processor for over a decade until the introduction of the Intel Core line in 2006. They are based on both the architecture used in Atom and that of Core processors. In the case of Atom architectures, Pentiums are the highest performance implementations of the architecture. Pentium processors with Core architectures prior to 2017 were distinguished from the faster, higher-end i-series processors by lower clock rates and disabling some features, such as hyper threading, virtualization and sometimes L3 cache.

9.9.1. PENTIUM-BRANDED PROCESSORS

9.9.1.1. P5 MICROARCHITECTURE BASED

The original Intel P5 or Pentium and Pentium MMX processors were the superscalar follow-on to the 80486 processor and were marketed from 1993 to 1999. Some versions of these were available as Pentium over drive that would fit into older CPU sockets.

Core p	Process	Clock rates	L1 cache	FSB	Socket	Release date
P5	0.8 μm	60–66 MHz	16 KB	60–66 MHz	Socket 4	Mach 1993
P54C	0.6 μm	75–120 MHz	16 KB	50–66 MHz	Socket 5	October 1994
P54CS	0.35 μm	133–200 MHz	16 KB	60–66 MHz	Socket 7	June 1995
P55C	0.35 μm	120–233 MHz	32 KB	60–66 MHz	Socket 7	January 1997
Tillamook	0.25 μm	166–300 MHz	32 KB	66 MHz	Socket 7	August 1997

9.9.1.2. P6 MICROARCHITECTURE BASED

In parallel with the P5 microarchitecture, Intel developed the P6 micro architecture and started marketing it as the Pentium Pro for the high-end market in 1995. It introduced out of order execution and an integrated second-level cache on dual-chip processor package. The second P6 generation replaced the original P5 with the Pentium II and rebranded the high-end version as Pentium II Xeon. It was followed by a third version named the Pentium III and Pentium III Xeon respectively. The Pentium II line added the MMX instructions that were also present in the Pentium MMX.

Versions of these processors for the laptop market were initially named Mobile Pentium II and Mobile Pentium III, later versions were named Pentium III-M. Starting with the Pentium II, the Celeron brand was used for low-end versions of most Pentium processors with a reduced feature set such as a smaller cache or missing power management features.

9.9.1.3. PENTIUM PRO

Core	Process	Clock rates	L2 cache	FSB	Socket	Release date
P6	0.5 μm	150 MHz	256 KB	60–66 MHz	Socket 8	November 1995
P6	0.35 μm	166–200 MHz	256 -1024 KB	60–66 MHz	Socket 8	

9.9.1.4. PENTIUM II

Core	Process	Clock rates	L2 cache	FSB	Socket	Release date
------	---------	-------------	----------	-----	--------	--------------

Klamath	0.35 μm	233- 300 MHz	512 KB	66 MHz	Slot 1	May 1997
Deshutes	0.25 μm	266- 450 MHz	512 KB	60- 100 MHz	Slot 1	January 1998
Tonga	0.25 μm	233- 300 MHz	512 KB	66 MHz	MMC-2	April 1998
Dixon	0.25 μm	266- 366 MHz	256 KB	66 MHz	MMC-2	January 1999

9.9.1.5. PENTIUM III

Core	Process	Clock rates	L2 cache	FSB	Socket	Release date
Katmai	0.25 μm	450- 600 MHz	512 KB	100- 133 MHz	Slot 1	February 1999
Coppermine	0.18 μm	400- 1.13 GHz	256 KB	100- 133 MHz	Slot 1, Socket 370, BGA2, μ PGA2	October 1999
Tualatin	0.13 μm	700- 1.4 GHz	512 KB	100- 133 MHz	Socket 370, BGA2, μ PGA2	April 1998

9.9.1.6. PENTIUM 4

Core	Process	Clock rates	L2 cache	FSB	Socket	Release date
Willamette	180 nm	1.3- 2.0 GHz	256 KB	400 MT/s	Socket 423, Socket 478	November 2000
Northwood	130 nm	1.6-3.4 GHz	512 KB	400 MT/s- 800 MT/s	Socket 478	January 2002

Gallatin	130 nm	3.2– 3.46 GHz	800– 1066 MT/s	800– 1066 MT/s	Socket 478, LGA 775	November 2003
Prescott	90 nm	2.4– 3.8 GHz	1 MB	533 MT/s– 800 MT/s	Socket 478, LGA 775	February 2004
Prescott- 2M	90 nm	2.8– 3.8 GHz	2 MB	800– 1066 MT/s	LGA 775	February 2005
Cedar Mill	65 nm	3 GHz	2 MB	800 MT/s	LGA 775	January 2006

9.10 SUMMARY

In this unit you learnt about The **Intel 8255** (or **i8255**) Programmable Peripheral Interface (PPI) chip, Programmable Interrupt Controller (PIC) designed for the Intel 8085 and Intel 8086 microprocessors. In this unit you have learnt also Operational Modes Of 8255, Intel 8259, Intel 80286, Intel 80286, Intel 80286 and Pentium.

9.11 GLOSSARY

Bus interface unit (BIU),

Execution and control unit (EU)

Floating point unit (FU).

9.12 REFERENCES

1. Advanced 80386 programming techniques by James L Turley
2. Microprocessor and Microcontroller System By A. P. Godse
3. The 8085 Microprocessor: Architecture, Programming and Interfacing by B. S. Umashankar and K. Udaya Kumar.
4. Advanced Microprocessors and Peripherals by A. K. Ray

5. Online sources

6. Electronics desk

9.13 SUGGESTED READINGS

1. Digital Fundamentals, 10th Ed, Floyd T L, Prentice Hall, 2009.

2. NPTEL

3. You Tube

4. Online tutorial

5. Byju's online study material

9.14 TERMINAL QUESTIONS

9.14.1 Short Answer type

1. Draw pin diagram of 80286.

2. Draw architecture 80386.

3. Draw pin diagram of 80486.