The McGraw-Hill Companies

8051 Microcontroller: Hardware, Software & Applications

The McGraw-Hill Companies

ABOUT THE AUTHORS

V Udayashankara received his BE in Electronics and Communication Engineering (1984) from Sri Jayachamarajendra College of Engineering (SJCE), Mysore. He then obtained his ME and PhD from Indian Institute of Science (IISc), Bangalore. Currently, he is a Professor in the department of Instrumentation Technology of SJCE. He has more than 24 years of teaching experience. Dr Udayashankara has carried out many research projects in the area of Embedded Systems. His research interests also include Rehabilitation Engineering, Speech and EEG Signal Processing. He has authored more than 60 publications in journals and conferences related to these subjects.

M S Mallikarjunaswamy obtained his BE in Instrumentation Technology (1993) from Sri Jayachamarajendra College of Engineering (SJCE), Mysore and MTech in Industrial Electronics from University of Mysore (1999). Currently, he is an Assistant Professor in the department of Instrumentation Technology of SJCE. He has more than 14 years of teaching experience and his areas of interest include Embedded Architectures, VLSI and Biomedical Signal Processing. He has presented several papers in national and international conferences.

The McGraw-Hill Companies

8051 Microcontroller: Hardware, Software & Applications

V Udayashankara
Professor,
Dept. of Instrumentation Technology,
Sri Jayachamarajendra College of
Engineering, Mysore

M S Mallikarjunaswamy
Assistant Professor,
Dept. of Instrumentation Technology,
Sri Jayachamarajendra College of
Engineering, Mysore

Tata McGraw-Hill Publishing Company Limited
NEW DELHI

McGraw-Hill Offices

New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto

The McGraw-Hill Companies

Tata McGraw-Hill

Published by the Tata McGraw-Hill Publishing Company Limited,
7 West Patel Nagar, New Delhi 110 008.

8051 Microcontroller: Hardware, Software & Applications

ISBN 13: 978-0-07-008681-4
ISBN 10: 0-07-008681-8

Managing Director: *Ajay Shukla*
General Manager: Publishing—SEM & Tech Ed: *Vibha Mahajan*
Sponsoring Editor: *Shalini Jha*
Jr. Sponsoring Editor: *Nilanjan Chakravarty*
Jr. Editorial Executive: *Tina Jajoriya*
Sr. Copy Editor: *Dipika Dey*
Production Executive: *Suneeta S Bohra*

General Manager: Marketing—Higher Education & School: *Michael J Cruz*
Product Manager: SEM & Tech Ed: *Biju Ganesan*

Controller—Production: *Rajender P Ghansela*
Asst. General Manager—Production: *B L Dogra*

Published by the Tata McGraw-Hill Publishing Company Limited, 7 West Patel Nagar, New Delhi 110 008, typeset at Bukprint India, B-180A, Guru Nanak Pura, Laxmi Nagar-110 092 and printed at SDR Printers, A-28, West Jyoti Nagar, Loni Road, Shahdara, Delhi 110 094

Cover: SDR Printers

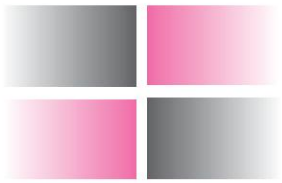
RALYCRLFRYCBB

This work is dedicated to

His Holiness

Dr Sri Sri Shivarathri Rajendra Mahaswamiji

—Authors



PREFACE

Latest developments in integrated-circuit technology and embedded processors have provided a strong impetus to the design of microcontroller-based systems. The 8051—a highly popular microcontroller—is used in a wide variety of control systems, telecom applications, robotics as well as in the automotive industry. Its standard form encompasses numerous standard on-chip peripherals (including timers, counters, and UARTs), 4 Kbytes of on-chip program memory and 256 bytes of data memory, making single-chip implementation possible. The study of architecture, instruction set, interfacing circuits and programs of the 8051 has become a challenging pursuit for many budding engineers.

WHY DID WE WRITE THIS BOOK?

This book has been written to fulfill the need of students pursuing degree or diploma courses in engineering as well as technical modules. In an undergraduate study of engineering, students are required to pursue a theory course on microcontrollers. Moreover, a laboratory course on microcontrollers is a part of the curricula of electrical science branches. A large number of students carry out their project work based on various aspects of microcontrollers. As teachers of engineering discipline for several years, we have observed and felt the need for a good textbook relevant to the requirement of students, for study of hardware, software and to develop necessary skills to excel in project work.

Thus, we envisaged this present book to provide a stimulating learning experience while facilitating students to become proficient in designing with the 8051. The book fulfills the objective of imparting knowledge about the complete hardware of microcontrollers, including their architecture, instruction set and interfaces, and also provides a depiction of the software used to develop programs with a good number of examples and applications. The treatment and coverage of topics are an outcome of more than a decade of teaching experience and laboratory work. After introducing the fundamentals and capabilities of the 8051, we have featured practical designs so that readers can develop in-depth understanding of the applications. Basic acquaintance with logic design and C programming will enable readers to easily grasp this book, without any prerequisite of prior knowledge of microprocessors.

WHO SHOULD READ IT?

8051 Microcontrollers: Hardware, Software and Applications is intended as an introductory course on microcontrollers with coverage of architecture, hardware interfaces, software and description of applications. It serves the purpose of a textbook for **students at the undergraduate level in engineering** such as Computer Science & Engineering, Electronics & Communication Engineering, and Electronics & Instrumentation Engineering. It can also be a valuable reference for students of BCA, MCA, diploma courses and technical training institutes, as well as practicing engineers interested in knowing more about the 8051. The objective-type and short-answer questions will benefit candidates preparing for competitive exams and technical interviews.

WHAT MAKES THIS BOOK OUTSTANDING?

The topics in this book—evolved based on our academic expertise of more than a decade—have been presented to cover hardware, software and applications. A special feature is the inclusion of abundant programming and interfacing examples; the example programs and hardware interfaces being a careful compilation from our laboratory work with students. After elucidating the basics of computers, microprocessors and microcontrollers, the text proceeds to explain the basics of the 8051 microcontroller, peripheral interfacing and then programming both in assembly and C language. Also, a description of software tools and their steps of usage have been provided to enhance the comprehension of the students.

This text is one-of-its-kind as it includes programs for interfacing experiments, both in 8051 assembly and C language. It follows an integrated approach to architecture and programming. Comprehensive coverage of important topics like A/D –D/A, and exhaustive discussion of software development tools like A51 Assembler and S51 Simulator, SC51 C compiler, µVision C Compiler and Simulator (SIDE 51) have been provided so that students gain in-depth knowledge about microcontrollers.

Salient Features

- Incorporates programs for interfacing experiments both in 8051 assembly and C language
- Offers a clear exposition of the fundamentals
- Writing style renders even complex topics easy to understand
- Text has a smooth flow and sections are properly interlinked to maintain continuity
- Topics illustrated with supportive diagrams and tables
- Students can implement the example programs to see how they work
- Includes interface experiments to give practical exposure
- Elucidates the hardware interface useful for laboratory work
- Discusses development tools

Throughout the book, the text is supplemented with **pedagogical aids** to provide better comprehension of concepts. **Chapter Objectives** provide a quick overview of the concepts that will be discussed in the chapter. Major topics like arithmetic instruction, instruction set, and timers/counters have been supported with **Solved Examples**. **Section-end Review Questions** have been provided to reinforce important points. **Summary** briefly reviews the highlights of each chapter and will be helpful for a quick appraisal during the examinations. Rich pool of pedagogy includes **Exercises (Multiple-choice Questions and Review Questions)** at the end of each chapter to help students test their understanding of concepts. **Programs** provided in this textbook are useful to conduct a laboratory course on the 8051 Microcontroller.

Pedagogical Features

- More than 200 Solved Examples
- More than 100 Multiple Choice Questions
- More than 250 Section-end Questions
- 200 Chapter-end Exercises

HOW IS THE BOOK ORGANISED?

Spanning over nine chapters, the structuring of this book provides a comprehensive understanding of software, hardware and applications of the 8051. Each chapter begins with objectives, followed by a thorough discussion of the concepts and culminates with a summary and plenty of exercises.

Chapter 1 covers the basics of computers, microprocessors, microcontrollers and related software. It explains concepts of RISC and CISC and also discusses embedded systems in detail. **Chapter 2** starts with the features, architecture and pin diagram of 8051. Memory organisation and external memory interfacing with examples are discussed in detail. It also explores the operation of stack with examples. Features of 8052 are also been elucidated. **Chapter 3** presents instruction syntax, data types and subroutine concepts. It aptly covers addressing modes supported by 8051 with examples. Instruction classification and instruction timings are also discussed with examples. After each group of instructions, examples to demonstrate their use are given. At the end of the chapter, a summary of the 8051 instruction set, syntax, examples and flag conditions are presented in a tabular format. **Chapter 4** covers programming concepts for 8051 with 30 examples. Issues of delay programs are aptly explored. **Chapter 5** provides an overview of software-development tools. Software tools SIDE 51 of SPJ systems are used to test the programs. The chapter also provides an overview of the development steps for evaluation software µVision of KEIL. **Chapter 6** introduces the basics of parallel I/O ports. Interfaces of push-button switches, matrix keyboard, LED, seven segments and LCD displays are covered with circuits, assembly and C programs. Interfacing of A/D and D/A converters are also discussed with circuits and programs. This chapter also includes operation and interface of stepper and dc motors with necessary assembly and C programs. **Chapter 7** covers concepts of interrupts, classification and priority of interrupts. It also discusses timer and counter functions. Operations of timer 0 and timer 1 are explained with programs to create time delay and to generate waveforms. **Chapter 8** is devoted to serial communication. It covers the basics and types of serial communication. Serial communication of the 8051 microcontroller is discussed with programming examples. This chapter introduces RS232 bus and plug connectors. It also incorporates MAX232IC to convert TTL signals to RS232 standards. **Chapter 9** covers the architecture and features of 8255A. Interfacing of 8255A with 8051, and also interfacing of external devices such as printers, stepper motors and D/A converters using 8255A are discussed with circuits and programs. This chapter also includes design of signal-conditioning circuits and minimum embedded systems using 8051. The six appendices include information on 89C60X2/61X2 Flash Microcontroller Data Sheets, ASCII Code Values, 74LS373 Latch Data Sheets, ADC0808 /0809 Data Sheets, DAC 0808 Data Sheets, 8052 Microcontroller data sheets, 80196 Architectural overview, and PIC 16F874 Microcontroller data sheets

WHAT DOES THE ONLINE LEARNING CENTER OFFER?

For answers to the multiple-choice questions and scheme description for project ideas given in the book, readers can also refer the exhaustive resource bank on the website at http://www.mhhe.com/udayashankara/mhsa.

Resources for Instructors

- Solution Manual
- Chapter-wise PowerPoint Slides
- Test Bank offers a Collection of Model Exam Questions

Resources for Students

- Chapter-wise Learning Objectives and Summary
- Sample Chapter—Chapter 1 on Introduction to Computer, Microprocessor and Microcontroller
- Block Schematic Diagrams and Brief Description of Projects given in the Book
- Additional Chapter-wise Exercises
- Multiple Choice Questions
- Real-life Experiments for 8051 Microcontroller Laboratory
- Web Links to Reference Material

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude to J S S Mahavidyapeetha for encouraging us to reach this milestone. We would also like to thank Dr B G Sangameshwara, Principal, Sri Jayachamarajendra College of Engineering, Mysore, for motivating us to write this book. Furthermore, we wish to extend our appreciation to Dr M Sukumar, Ms Roopa Nanjaiah, Mr Bharath Hegde and Mr Srinidhi R for their valuable comments and contributions.

Our grateful acknowledgement is also due to Advanced Electronic Systems (ALS), Bangalore and SPJ Systems for their useful suggestions, and granting us permissions to use the SIDE51 compiler. Our special thanks to KEIL Corporation for providing their µVision C-compiler evaluation software.

We are indebted to a number of colleagues and friends for their timely help in planning and execution of this book. We are also grateful to members of our family for the support and cooperation extended to us during the preparation of the manuscript. We would like to express our thanks to the entire team at McGraw-Hill Education India for efficient management of the project.

A note of acknowledgement is also due to the esteemed reviewers of this book for their valuable feedback.

Sanjeet Dwivedi	*Indira Gandhi Government Engineering College, Sagar*, Madhya Pradesh
Sangeeta Shukla	*Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal*, Madhya Pradesh
P K Malik	*SCR Institute of Engineering and Technology, Meerut*, Uttar Pradesh
A K Sen	*Kalinga Institute of Industrial Technology, Bhubaneswar,* Orissa
Ashok Kumar Turuk	*National Institute of Technology, Rourkela,* Orissa
Dhruba Basu	*West Bengal University of Technology, Kolkata,* West Bengal
Vivek Agarwal	*Indian Institute of Technology Bombay, Mumbai*, Maharashtra
H Renganathan	*Indian Institute of Technology Madras, Chennai,* Madras
G C Thyagarajan	*College of Technology, Coimbatore,* Tamil Nadu
K E Srinivas Murthy	*G Pulliah College of Engineering and Technology, Kurnool,* Andhra Pradesh
M S K Rayalu	*VR Siddhartha Engineering College, Vijayawada,* Andhra Pradesh

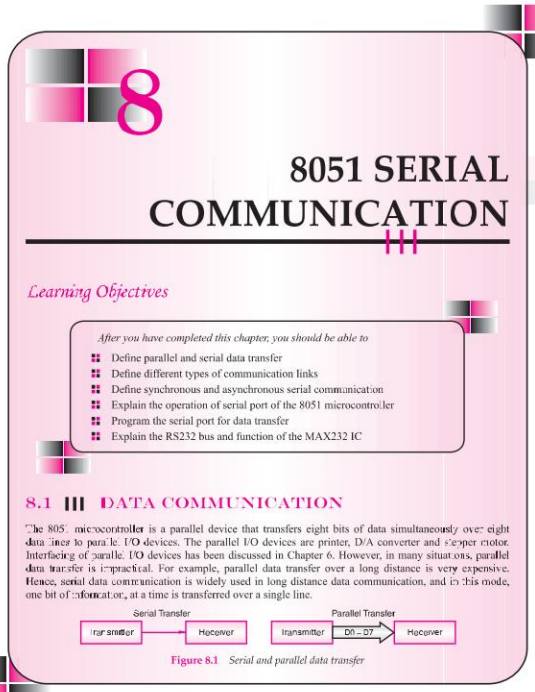
If readers have any suggestions and comments, they are welcome to contact us at tmh.csefeedback@gmail.com. Kindly mention the title and author names in the subject line.

V Udayashankara
M S Mallikarjunaswamy

The McGraw-Hill Companies

CONTENTS

VISUAL WALKTHROUGH



8

8051 SERIAL COMMUNICATION

Learning Objectives

After you have completed this chapter, you should be able to

- Define parallel and serial data transfer
- Define different types of communication links
- Define synchronous and asynchronous serial communication
- Explain the operation of serial port of the 8051 microcontroller
- Program the serial port for data transfer
- Explain the RS232 bus and function of the MAX232 IC

8.1 DATA COMMUNICATION

The 8051 microcontroller is a parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices. The parallel I/O devices are printer, D/A converter and stepper motor. Interfacing of parallel I/O devices has been discussed in Chapter 6. However, in many situations, parallel data transfer is impractical. For example, parallel data transfer over a long distance is very expensive. Hence, serial data communication is widely used in long distance data communication, and in this mode, one bit of information, at a time is transferred over a single line.

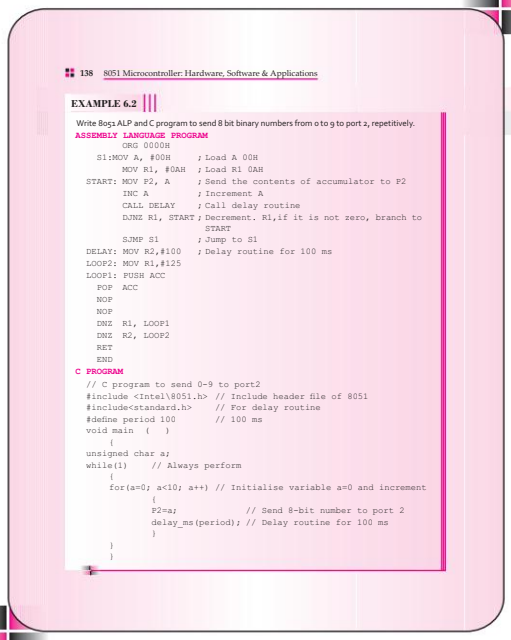
Serial Transfer: Transmitter → Receiver

Parallel Transfer: Transmitter → D0 – D7 → Receiver

Figure 8.1 *Serial and parallel data transfer*

Learning Objectives

Learning Objectives give an overview of the concepts that will be discussed in the chapter.



138 8051 Microcontroller: Hardware, Software & Applications

EXAMPLE 6.2

Write 8051 ALP and C program to send 8 bit binary numbers from 0 to 9 to port 2, repetitively.

ASSEMBLY LANGUAGE PROGRAM

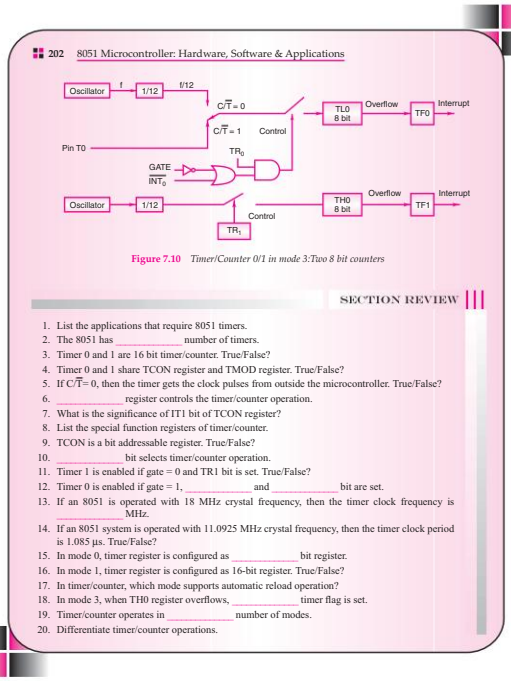
```
        ORG 0000H
   S1:MOV A, #00H      ; Load A 00H
      MOV R1, #0AH     ; Load R1 0AH
START: MOV P2, A       ; Send the contents of accumulator to P2
      INC A            ; Increment A
      CALL DELAY       ; Call delay routine
      DJNZ R1, START ; Decrement. R1,if it is not zero, branch to
                         START
      SJMP S1          ; Jump to S1
DELAY: MOV R2,#100     ; Delay routine for 100 ms
LOOP2: MOV R1,#125
LOOP1: PUSH ACC
  POP  ACC
  NOP
  NOP
  DNZ  R1, LOOP1
  DNZ  R2, LOOP2
  RET
  END
```

C PROGRAM

```
// C program to send 0-9 to port2
#include <Intel\8051.h> // Include header file of 8051
#include<standard.h>    // For delay routine
#define period 100      // 100 ms
void main  (   )
     {
unsigned char a;
while(1)     // Always perform
    {
    for(a=0; a<10; a++) // Initialise variable a=0 and increment
           {
           P2=a;             // Send 8-bit number to port 2
          delay_ms(period); // Delay routine for 100 ms
           }
    }
    }
```

Examples

A careful compilation of programming and interfacing examples both in 8051 assembly and C language strengthen understanding of concepts.

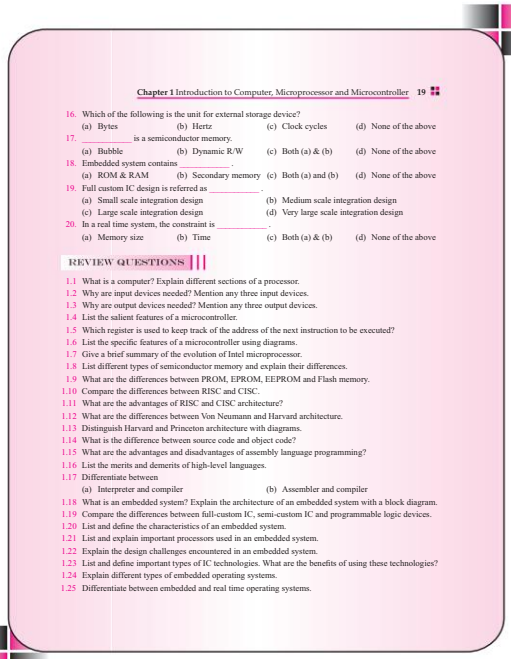


202 8051 Microcontroller: Hardware, Software & Applications

Figure 7.10 *Timer/Counter 0/1 in mode 3:Two 8 bit counters*

SECTION REVIEW

1. List the applications that require 8051 timers.
2. The 8051 has _____ number of timers.
3. Timer 0 and 1 are 16 bit timer/counter. True/False?
4. Timer 0 and 1 share TCON register and TMOD register. True/False?
5. If C/$\overline{T}$= 0, then the timer gets the clock pulses from outside the microcontroller. True/False?
6. _____ register controls the timer/counter operation.
7. What is the significance of IT1 bit of TCON register?
8. List the special function registers of timer/counter.
9. TCON is a bit addressable register. True/False?
10. _____ bit selects timer/counter operation.
11. Timer 1 is enabled if gate = 0 and TR1 bit is set. True/False?
12. Timer 0 is enabled if gate = 1, _____ and _____ bit are set.
13. If an 8051 is operated with 18 MHz crystal frequency, then the timer clock frequency is _____ MHz.
14. If an 8051 system is operated with 11.0925 MHz crystal frequency, then the timer clock period is 1.085 µs. True/False?
15. In mode 0, timer register is configured as _____ bit register.
16. In mode 1, timer register is configured as 16-bit register. True/False?
17. In timer/counter, which mode supports automatic reload operation?
18. In mode 3, when TH0 register overflows, _____ timer flag is set.
19. Timer/counter operates in _____ number of modes.
20. Differentiate timer/counter operations.



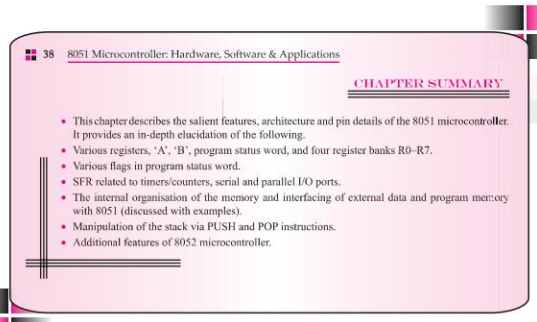
16. Which of the following is the unit for external storage device?
    (a) Bytes (b) Hertz (c) Clock cycles (d) None of the above
17. _____ is a semiconductor memory.
    (a) Bubble (b) Dynamic R/W (c) Both (a) & (b) (d) None of the above
18. Embedded system contains _____ .
    (a) ROM & RAM (b) Secondary memory (c) Both (a) and (b) (d) None of the above
19. Full custom IC design is referred as _____ .
    (a) Small scale integration design (b) Medium scale integration design
    (c) Large scale integration design (d) Very large scale integration design
20. In a real time system, the constraint is _____ .
    (a) Memory size (b) Time (c) Both (a) & (b) (d) None of the above

REVIEW QUESTIONS

1.1 What is a computer? Explain different sections of a processor.
1.2 Why are input devices needed? Mention any three input devices.
1.3 Why are output devices needed? Mention any three output devices.
1.4 List the salient features of a microcontroller.
1.5 Which register is used to keep track of the address of the next instruction to be executed?
1.6 List the specific features of a microcontroller using diagrams.
1.7 Give a brief summary of the evolution of Intel microprocessor.
1.8 List different types of semiconductor memory and explain their differences.
1.9 What are the differences between PROM, EPROM, EEPROM and Flash memory.
1.10 Compare the differences between RISC and CISC.
1.11 What are the advantages of RISC and CISC architecture?
1.12 What are the differences between Von Neumann and Harvard architecture.
1.13 Distinguish Harvard and Princeton architecture with diagrams.
1.14 What is the difference between source code and object code?
1.15 What are the advantages and disadvantages of assembly language programming?
1.16 List the merits and demerits of high-level languages.
1.17 Differentiate between
    (a) Interpreter and compiler (b) Assembler and compiler
1.18 What is an embedded system? Explain the architecture of an embedded system with a block diagram.
1.19 Compare the differences between full-custom IC, semi-custom IC and programmable logic devices.
1.20 List and define the characteristics of an embedded system.
1.21 List and explain important processors used in an embedded system.
1.22 Explain the design challenges encountered in an embedded system.
1.23 List and define important types of IC technologies. What are the benefits of using these technologies?
1.24 Explain different types of embedded operating systems.
1.25 Differentiate between embedded and real time operating systems.

Review Questions

Section-end and chapter-end review questions provide an opportunity to the students to reinforce their learning and gain confidence.


38 8051 Microcontroller: Hardware, Software & Applications

CHAPTER SUMMARY

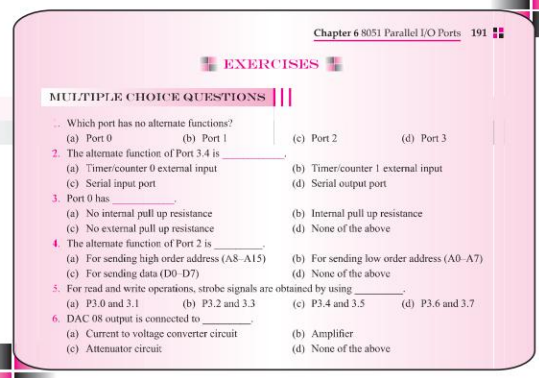
- This chapter describes the salient features, architecture and pin details of the 8051 microcontroller. It provides an in-depth elucidation of the following.
- Various registers, 'A', 'B', program status word, and four register banks R0–R7.
- Various flags in program status word.
- SFR related to timers/counters, serial and parallel I/O ports.
- The internal organisation of the memory and interfacing of external data and program memory with 8051 (discussed with examples).
- Manipulation of the stack via PUSH and POP instructions.
- Additional features of 8052 microcontroller.

Chapter Summary

Summary succinctly recapitulates the main concepts of the chapter, making it ideal for review.

Multiple Choice Questions

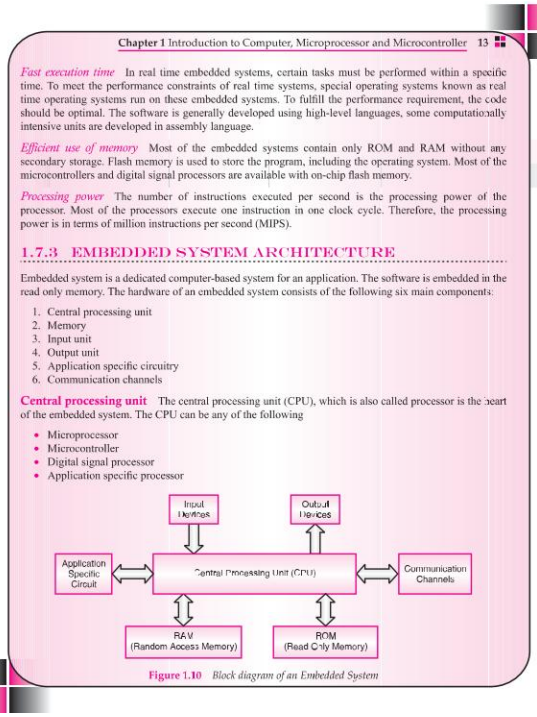
Numerous Multiple Choice Questions—a useful study tool for knowledge assessment—have been included.


Chapter 6 8051 Parallel I/O Ports 191

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. Which port has no alternate functions?
   (a) Port 0 (b) Port 1 (c) Port 2 (d) Port 3
2. The alternate function of Port 3.4 is _____.
   (a) Timer/counter 0 external input (b) Timer/counter 1 external input
   (c) Serial input port (d) Serial output port
3. Port 0 has _____.
   (a) No internal pull up resistance (b) Internal pull up resistance
   (c) No external pull up resistance (d) None of the above
4. The alternate function of Port 2 is _____.
   (a) For sending high order address (A8–A15) (b) For sending low order address (A0–A7)
   (c) For sending data (D0–D7) (d) None of the above
5. For read and write operations, strobe signals are obtained by using _____.
   (a) P3.0 and 3.1 (b) P3.2 and 3.3 (c) P3.4 and 3.5 (d) P3.6 and 3.7
6. DAC 08 output is connected to _____.
   (a) Current to voltage converter circuit (b) Amplifier
   (c) Attenuator circuit (d) None of the above


Chapter 1 Introduction to Computer, Microprocessor and Microcontroller 13

Fast execution time In real time embedded systems, certain tasks must be performed within a specific time. To meet the performance constraints of real time systems, special operating systems known as real time operating systems run on these embedded systems. To fulfill the performance requirement, the code should be optimal. The software is generally developed using high-level languages, some computationally intensive units are developed in assembly language.

Efficient use of memory Most of the embedded systems contain only ROM and RAM without any secondary storage. Flash memory is used to store the program, including the operating system. Most of the microcontrollers and digital signal processors are available with on-chip flash memory.

Processing power The number of instructions executed per second is the processing power of the processor. Most of the processors execute one instruction in one clock cycle. Therefore, the processing power is in terms of million instructions per second (MIPS).

1.7.3 EMBEDDED SYSTEM ARCHITECTURE

Embedded system is a dedicated computer-based system for an application. The software is embedded in the read only memory. The hardware of an embedded system consists of the following six main components:

1. Central processing unit
2. Memory
3. Input unit
4. Output unit
5. Application specific circuitry
6. Communication channels

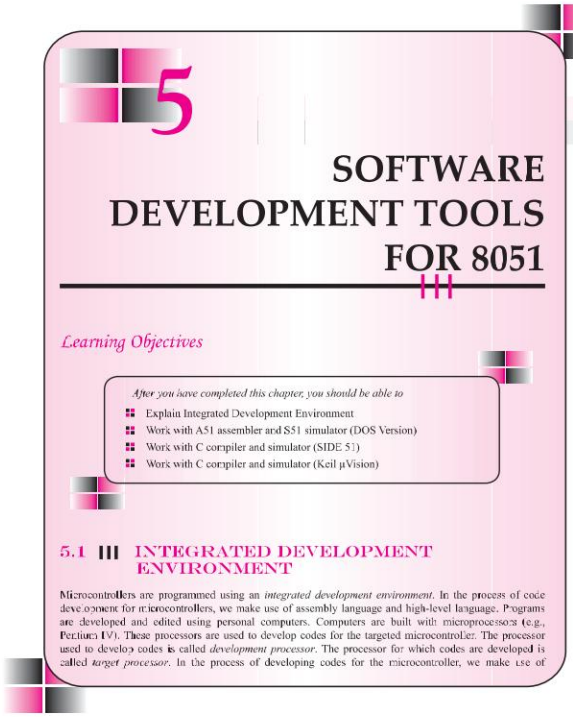
Central processing unit The central processing unit (CPU), which is also called processor is the heart of the embedded system. The CPU can be any of the following

- Microprocessor
- Microcontroller
- Digital signal processor
- Application specific processor

Figure 1.10 *Block diagram of an Embedded System*

An Overview Embedded Systems

Chapter 1 includes embedded system architecture, design challenges and embedded operating system.



5

SOFTWARE DEVELOPMENT TOOLS FOR 8051

Learning Objectives

After you have completed this chapter, you should be able to

- Explain Integrated Development Environment
- Work with A51 assembler and S51 simulator (DOS Version)
- Work with C compiler and simulator (SIDE 51)
- Work with C compiler and simulator (Keil µVision)

5.1 INTEGRATED DEVELOPMENT ENVIRONMENT

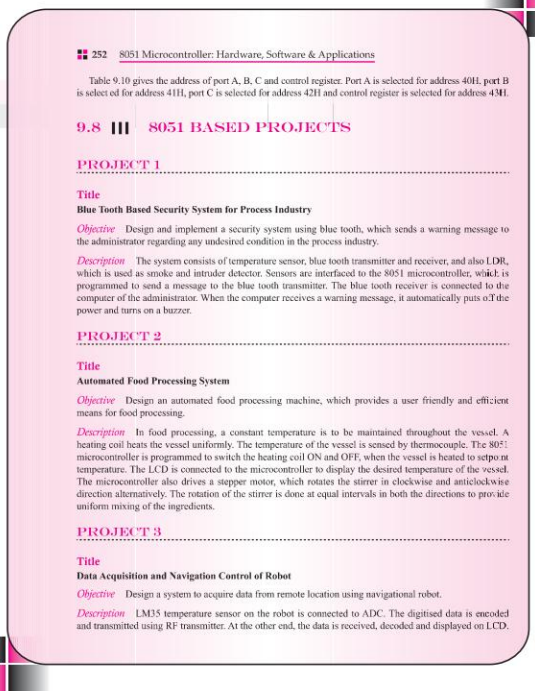
Microcontrollers are programmed using an *integrated development environment*. In the process of code development for microcontrollers, we make use of assembly language and high-level language. Programs are developed and edited using personal computers. Computers are built with microprocessors (e.g., Pentium IV). These processors are used to develop codes for the targeted microcontroller. The processor used to develop codes is called *development processor*. The processor for which codes are developed is called *target processor*. In the process of developing codes for the microcontroller, we make use of

Software Development Tools

In-depth discussion of software development tools like A51 assembler and S51 Simulator, SC51 C compiler, µVision C Compiler and Simulator (SIDE 51) provides comprehensive knowledge about microcontrollers.

8051 Based Projects

Chapter 9 devoted to the architecture and features of 8255A includes interfacing of 8255A with 8051 and interfacing of external devices such as printers, stepper motors and D/A converters using 8255A with circuits and programs.



252 8051 Microcontroller: Hardware, Software & Applications

Table 9.10 gives the address of port A, B, C and control register. Port A is selected for address 40H, port B is selected for address 41H, port C is selected for address 42H and control register is selected for address 43H.

9.8 8051 BASED PROJECTS

PROJECT 1

Title

Blue Tooth Based Security System for Process Industry

Objective Design and implement a security system using blue tooth, which sends a warning message to the administrator regarding any undesired condition in the process industry.

Description The system consists of temperature sensor, blue tooth transmitter and receiver, and also LDR, which is used as smoke and intruder detector. Sensors are interfaced to the 8051 microcontroller, which is programmed to send a message to the blue tooth transmitter. The blue tooth receiver is connected to the computer of the administrator. When the computer receives a warning message, it automatically puts off the power and turns on a buzzer.

PROJECT 2

Title

Automated Food Processing System

Objective Design an automated food processing machine, which provides a user friendly and efficient means for food processing.

Description In food processing, a constant temperature is to be maintained throughout the vessel. A heating coil heats the vessel uniformly. The temperature of the vessel is sensed by thermocouple. The 8051 microcontroller is programmed to switch the heating coil ON and OFF, when the vessel is heated to setpoint temperature. The LCD is connected to the microcontroller to display the desired temperature of the vessel. The microcontroller also drives a stepper motor, which rotates the stirrer in clockwise and anticlockwise direction alternatively. The rotation of the stirrer is done at equal intervals in both the directions to provide uniform mixing of the ingredients.

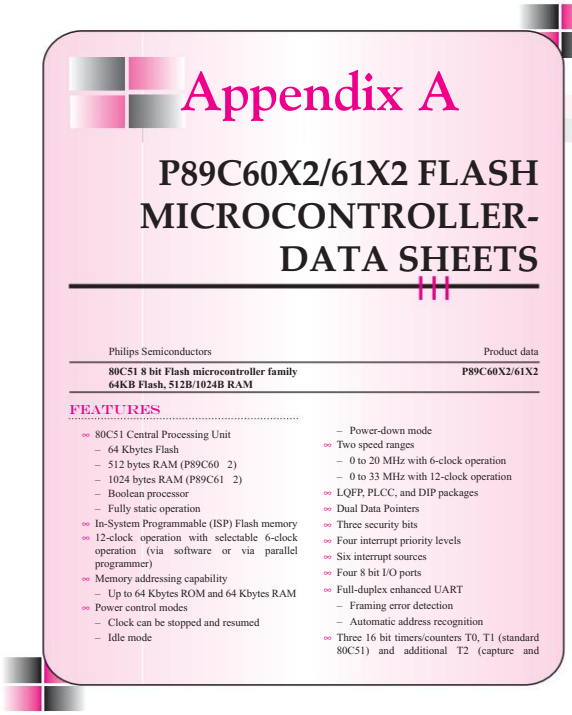
PROJECT 3

Title

Data Acquisition and Navigation Control of Robot

Objective Design a system to acquire data from remote location using navigational robot.

Description LM35 temperature sensor on the robot is connected to ADC. The digitised data is encoded and transmitted using RF transmitter. At the other end, the data is received, decoded and displayed on LCD.



Appendix A

P89C60X2/61X2 FLASH MICROCONTROLLER-DATA SHEETS

Philips Semiconductors — Product data

80C51 8 bit Flash microcontroller family 64KB Flash, 512B/1024B RAM — P89C60X2/61X2

FEATURES

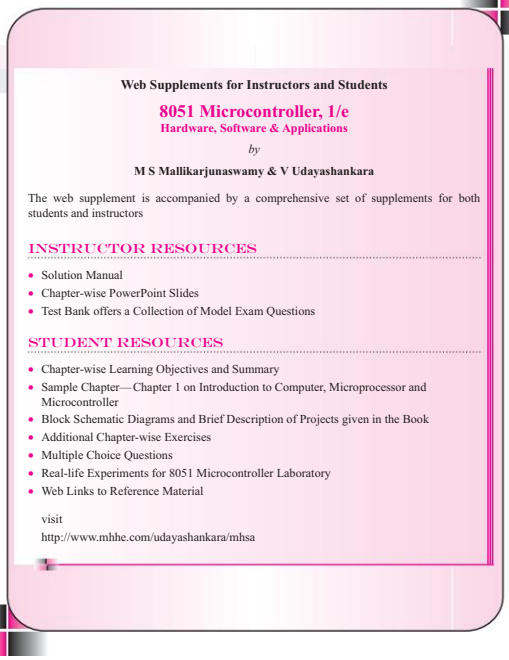
- 80C51 Central Processing Unit
  - 64 Kbytes Flash
  - 512 bytes RAM (P89C60 2)
  - 1024 bytes RAM (P89C61 2)
  - Boolean processor
  - Fully static operation
- In-System Programmable (ISP) Flash memory
- 12-clock operation with selectable 6-clock operation (via software or via parallel programmer)
- Memory addressing capability
  - Up to 64 Kbytes ROM and 64 Kbytes RAM
- Power control modes
  - Clock can be stopped and resumed
  - Idle mode
  - Power-down mode
- Two speed ranges
  - 0 to 20 MHz with 6-clock operation
  - 0 to 33 MHz with 12-clock operation
- LQFP, PLCC, and DIP packages
- Dual Data Pointers
- Three security bits
- Four interrupt priority levels
- Six interrupt sources
- Four 8 bit I/O ports
- Full-duplex enhanced UART
  - Framing error detection
  - Automatic address recognition
- Three 16 bit timers/counters T0, T1 (standard 80C51) and additional T2 (capture and

Appendices

Appendices include data sheets of relevant microcontrollers along with ASCII codes to apprise students of industry norms.

Online Learning Centre

A comprehensive, exclusive website http://www.mhhe.com/udayashankara/mhsa provides electronic resources for both instructors and students.



Web Supplements for Instructors and Students

8051 Microcontroller, 1/e

Hardware, Software & Applications

by

M S Mallikarjunaswamy & V Udayashankara

The web supplement is accompanied by a comprehensive set of supplements for both students and instructors

INSTRUCTOR RESOURCES

- Solution Manual
- Chapter-wise PowerPoint Slides
- Test Bank offers a Collection of Model Exam Questions

STUDENT RESOURCES

- Chapter-wise Learning Objectives and Summary
- Sample Chapter—Chapter 1 on Introduction to Computer, Microprocessor and Microcontroller
- Block Schematic Diagrams and Brief Description of Projects given in the Book
- Additional Chapter-wise Exercises
- Multiple Choice Questions
- Real-life Experiments for 8051 Microcontroller Laboratory
- Web Links to Reference Material

visit

http://www.mhhe.com/udayashankara/mhsa

The McGraw-Hill Companies

Web Supplements for Instructors and Students

8051 Microcontroller, 1/e

Hardware, Software & Applications

by

M S Mallikarjunaswamy & V Udayashankara

The web supplement is accompanied by a comprehensive set of supplements for both students and instructors

INSTRUCTOR RESOURCES

- Solution Manual
- Chapter-wise PowerPoint Slides
- Test Bank offers a Collection of Model Exam Questions

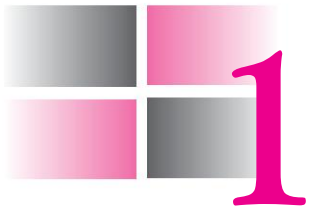
STUDENT RESOURCES

- Chapter-wise Learning Objectives and Summary
- Sample Chapter—Chapter 1 on Introduction to Computer, Microprocessor and Microcontroller
- Block Schematic Diagrams and Brief Description of Projects given in the Book
- Additional Chapter-wise Exercises
- Multiple Choice Questions
- Real-life Experiments for 8051 Microcontroller Laboratory
- Web Links to Reference Material

visit

http://www.mhhe.com/udayashankara/mhsa

The McGraw-Hill Companies



1 INTRODUCTION TO COMPUTER, MICROPROCESSOR AND MICROCONTROLLER

Learning Objectives

After you have completed this chapter, you should be able to

- Define or explain the following terms: Computer, CPU, Memory, Input unit, Output unit, Bus, RAM and ROM
- Explain the difference between microprocessor and microcontroller
- Compare features of commercial microcontrollers
- Explain the difference between Von Neumann (Princeton) architecture and Harvard architecture
- Explain the difference between RISC and CISC machines
- Explain the difference between machine language, assembly language and high level language programming of a computer
- Explain embedded system architecture, design challenges and embedded operating system

1.1 WHAT IS A COMPUTER?

A computer is a multipurpose programmable machine that reads binary instructions from its memory, accepts binary data as input, processes data according to those instructions and provides result as output. It is a programmable device, made up of *hardware* and *software*. The various components of the computer are called *hardware*. A set of instructions written for the computer to solve a specific task is called a *program* and a collection of programs is called *software*. The computer hardware consists of four main components:

1. *Central Processing Unit (CPU)*, which acts as the computer's brain.
2. *Input unit*, through which program and data can be entered into the computer.
3. *Output unit*, from which the results of computation can be displayed or viewed and
4. *Memory*, in which the programs and data are stored. Figure 1.1 shows a simple block diagram of a computer.

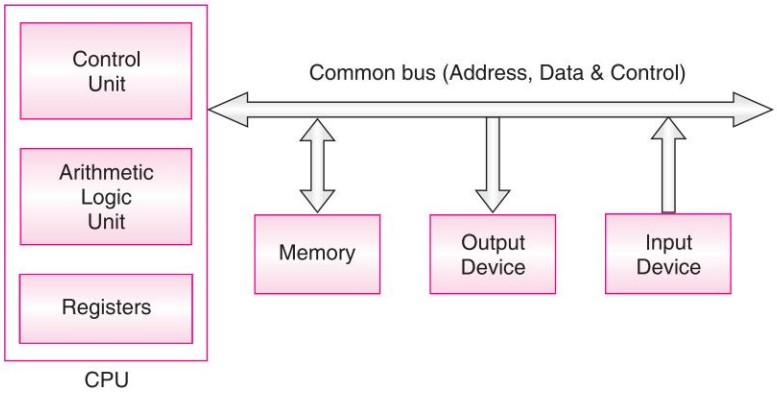


Figure 1.1 *Block diagram of a computer*

The processor communicates with the memory and input/output devices using three sets of lines called *buses—address bus, data bus*, and *control bus*. The bus is a communication path between the processor and the peripherals.

1.1.1 CENTRAL PROCESSING UNIT

The Central Processing Unit (CPU), which is also called the processor, can be further divided into three major parts:

Register File The register file consists of one or more registers. A register is a storage location in the CPU. It is used to hold data and/or a memory address during execution of an instruction.

Arithmetic Logic Unit The arithmetic logic unit is the computer's numerical calculator and logical operation evaluator. Under command from the control unit, it receives information from the memory. It analyses and rearranges the data and carries out the sequence of arithmetic and logical operations to accomplish the desired job.

Control Unit It controls and coordinates all the activities in the computer. It decodes the instructions and generates the necessary control signals for the execution of instructions. The system clock synchronises the

activities of the control unit. All CPU activities are measured by clock cycles. The control unit also contains a register called the Program Counter (PC), which contains the memory address of the next instruction to be executed.

1.1.2 MEMORY

The memory is a place where programs and data are stored. It is a storage device that stores instructions, data and intermediate results and provides that information to the other units of the computer. A computer contains Semiconductor, Magnetic or Optical memory. Semiconductor memory has been discussed in this chapter. Figure 1.2 shows the classification of semiconductor memory into two major types—Random Access Memory (RAM) and Read Only Memory (ROM).

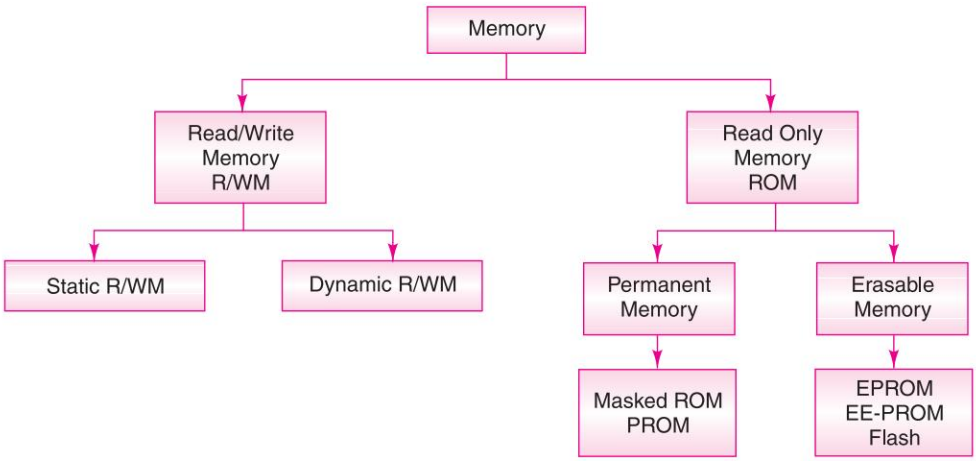


Figure 1.2 *Memory classification*

Random Access Memory Random access memory, popularly known as user memory, is used to store user program and data. It is also known as read/write (R/W) memory because it is equivalent to a group of addressable registers—you can either read the stored contents of memory location or write new contents into the memory location. The R/W memory is volatile, which means that when the power is turned off, the contents are lost. To communicate with the memory, the CPU should be able to

- Select the memory chip
- Select the register in the memory chip
- Read from or write into the register

Model of a typical read/write memory is shown in Fig. 1.3(a). The figure shows the chip select as $\overline{CS}$, write signal as $\overline{WR}$ and read signal as $\overline{RD}$—these are active low signals indicated by the bar. They help to select RAM and perform write and read operations. A flip-flop that can store one binary bit is called a *memory cell*. If more than one flip flop are grouped together, it is called a *register*.

In Fig. 1.3(b), the random access memory contains eight registers; each register contains eight memory cells and are arranged in a sequence. Here, the size of the memory is 8×8 bits or 8 bytes. To write into or read from any one of the registers, a specific register has to be selected. This is achieved by using a 3 to 8 decoder. Three address lines A2, A1 and A0 are required for the decoder. These input lines can have eight

different bits combination (000, 001, 010, 011, 100, 101, 110, 111), and each combination can select one of the registers.

Registers are made up of flip-flops and it stores bit as a voltage. *Dynamic memory* uses one MOS transistor and one capacitor to store one bit of information in the form of electrical charge. Static memories are designed to store binary information without needing periodic refreshes and require one flip-flop to store one bit of information (four to six transistors are needed to store one bit of information). Hence, it requires the use of more complicated circuitry for each bit. The advantage of dynamic memory is that a large number of transistor gates can be placed on the memory chip; thus it has high density and is faster than static memory. The disadvantage is that the charge (bit information) leaks and hence, it requires periodic refreshing, i.e. information needs to be read and written again after every few milliseconds.

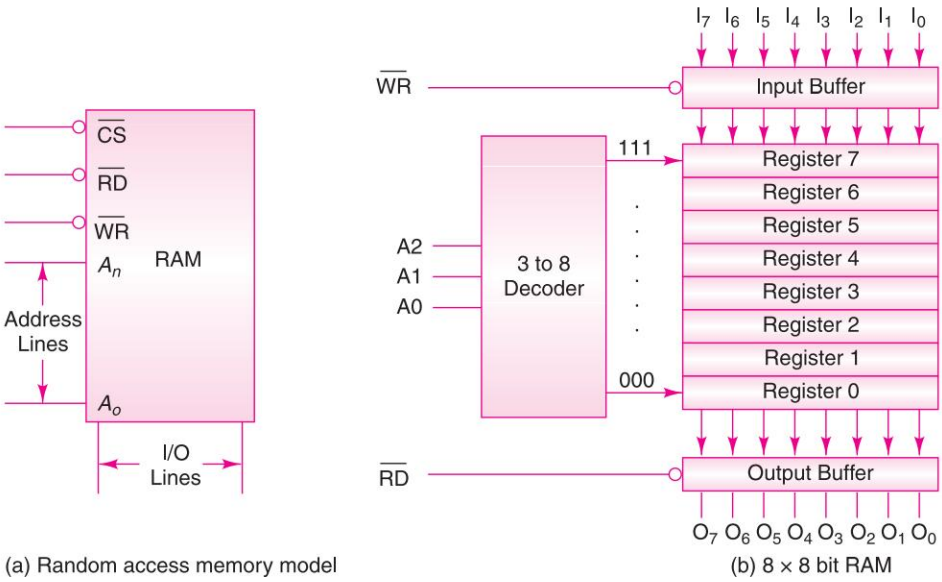


Figure 1.3 *Random Access Memory*

Read Only Memory The read only memory is the simplest kind of memory. It is equivalent to a group of registers and each store a word permanently. It is a nonvolatile memory which means that it retains the stored information even if the power is turned off. By applying control signal, we can read the contents of any memory location and if the processor attempts to write data to a ROM location, ROM will not accept the data. The model of a typical read only memory is shown in Fig. 1.4(a). The concept of ROM can be explained with the diodes arranged in a matrix format, as shown in Fig. 1.4(b). The horizontal lines are connected to the vertical lines using diodes to store '1'. The presence of diode stores '1' and the absence of diode stores '0'. When a register is selected, the voltage of that line goes high, and the output lines, where the diodes are connected also go high. When the memory register 010 is selected, the data byte 00000110 (06H) can be read at the data lines.

Two types of ROM are presently available, permanent ROM and erasable ROM. *Permanent* ROM includes two types of memory, Masked read only memory and Programmed read only memory.

Masked Read Only Memory (MROM) MROM is a type of ROM where binary instruction/data are stored during its fabrication. The semiconductor manufacturer places binary instruction/data in the memory

according to the request of the customer. MROM is used to hold microcontroller application program and constant data.

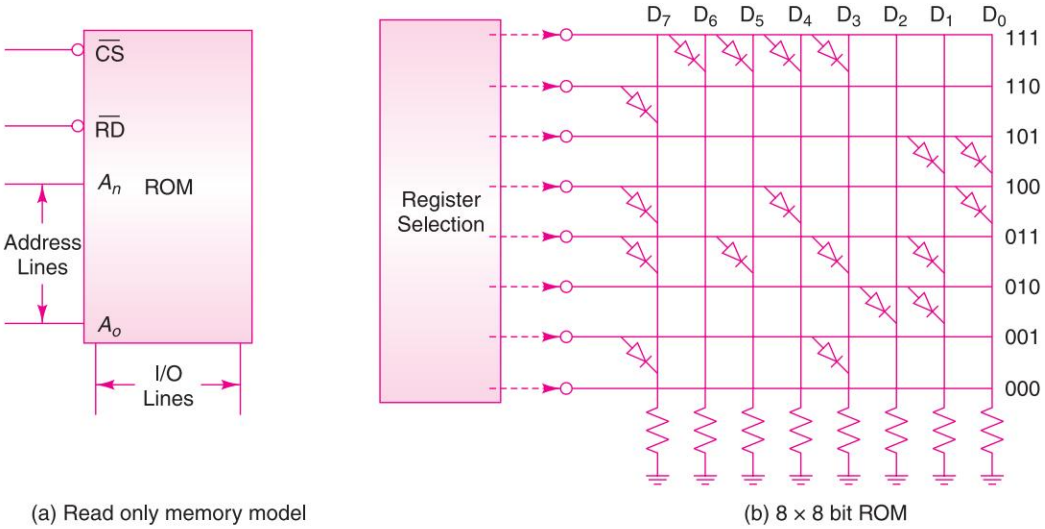


Figure 1.4 *Read Only Memory*

Programmed Read Only Memory (PROM) PROM is a type of read only memory that can be programmed in the field using a device called PROM programmer and here the programming is permanent. In other words, the stored contents cannot be erased.

Erasable memory includes three types of memory—*EPROM, EEPROM*, and *Flash memory*.

Erasable Programmable Read Only Memory (EPROM) The information stored in this memory is semi-permanent. Data is stored with PROM programmer. Subsequently, all the information can be erased by exposing the memory to ultraviolet light through a quartz window installed on top of the EPROM chip. It can then be reprogrammed.

Electrically Erasable Programmable Read Only Memory (EEPROM) EEPROM is a type of nonvolatile memory that can be erased by electrical signals and reprogrammed. EEPROM allows the user to selectively erase a single location, a row, or the whole chip. This feature requires a complicated programming circuitry. Because of this, EEPROM cannot achieve the density of EPROM technology.

Flash Memory Flash memory was invented to incorporate the advantages and avoid the drawback of EPROM and EEPROM technologies. It achieves the density of EPROM, and can be programmed and erased electrically like EEPROM. However, it does not allow individual location to be erased, but the user can erase the whole chip.

1.1.3 INPUT DEVICES

The input device transfers data and instructions in binary form, from the outside world to the CPU. The input device, also known as a peripheral is the means through which the user communicates data to the

computer. The user can enter instructions and data through an input device like a keyboard. Analog to digital converters, and switches can also be used as input devices.

1.1.4 OUTPUT DEVICES

It receives the stored result from the memory or from the CPU, converts it into a form that the user can understand and transfers this data to the outside world. Monitor, Liquid Crystal Display (LCD), seven-segment display, printer, etc. are used as output devices.

SECTION REVIEW

1. Name the four main components of a computer.
2. List three types of buses in a computer.
3. _____ register contains the memory address of the next instruction to be executed.
4. Differentiate volatile and nonvolatile memory.
5. List the signals that helps the CPU to communicate with the memory.
6. 8×8 bit memory stores _____ bit of information.
7. _____ number of address lines are required to address 16×8 bit memory.
8. Analog to digital converter is a _____ device.
9. LCD is a _____ device.
10. _____ memory can be erased electrically.

1.2 WHAT IS A MICROPROCESSOR?

In the late 1960s, the CPU was designed with discrete logic gates. As semiconductor technology advanced, it was possible to fabricate more than thousand gates on a single silicon chip—this came to be known as *Large-scale integration*. As technology moved from Small-scale Integration to Large-scale Integration, it became possible to build a whole CPU with its related timing functions on a single chip. This came to be known as *Microprocessor*. A computer that is designed using a microprocessor as its CPU, is known as a Microcomputer.

Intel Corporation announced the first 4 bit microprocessor 4004 in 1971. The number of bits refer to the number of binary digits that the microprocessor can manipulate in one operation. Soon after this, Intel developed an 8 bit microprocessor 8080 in 1974. The 8085 microprocessor that followed 8080, had few more additional features compared to 8080 architecture. The instruction sets of 8080 and 8085 are practically the same. The 8085 microprocessor has an 8 bit data bus, so it can read data from or write data to memory or I/O ports only 8 bit at a time. It has a 16 bit address bus, so it can address any one of 2^{16} or 65,536 memory locations. It operates with 3MHz clock signal. The limitations of 8085 microprocessor are

- It operates with low speed
- Less powerful addressing mode and instruction set
- Limited number of 8 bit general purpose registers
- Low memory (64 Kbytes) addressing capability

Because of these limitations, Intel Corporation announced the first 16 bit microprocessor 8086 in 1978. The 8086 microprocessor has 16 bit data bus and 20 bit address bus. It can read or write data to memory or I/O ports either 16 bit or 8 bit at a time and it can address any one of 1,048,576 (2^{20}) memory locations. Subsequently, Intel developed 16 bit processors such as 8088, 80186, 80188 and 80286. The Intel 8088 has the same instruction set and arithmetic unit as the 8086. It has 8 bit data bus and 20 bit address bus and can read or write data to memory or I/O devices only 8 bit of data at a time. 80186 and 80188 are an improved version of 8086 and 8088 respectively. Both the processors have a few additional instruction sets compared to 8086 instruction set. 80286 is an improved version of 80186. It is designed to be used as CPU in multitasking computers. It operates in real and virtual addressing mode. Later, in 1985, Intel developed 32 bit microprocessor 80386. The 80386 can address directly up to 4 gigabytes of memory. Intel's second generation of 32 bit microprocessor 80486 was available in the year 1989. Intel introduced a third generation 80586 (Pentium processor) in 1998. Motorola, Zilog, etc. also developed 8 bit, 16 bit and 32 bit Microprocessors. Microprocessors have been widely used after their invention. However, the following limitations of the microprocessor led to the invention of the microcontroller.

- A microprocessor requires external memory to execute a program.
- A microprocessor cannot be directly interfaced with I/O devices. Peripheral chips are needed to interface I/O devices.

SECTION REVIEW

1. The 8085 microprocessor has _____ bit data bus and _____ bit address bus.
2. List the limitations of the 8085 microprocessor.
3. The 8088 microprocessor has _____ bit data bus and _____ bit address bus.
4. The 80386 is a _____ bit microprocessor.
5. List the limitations of microprocessor.

1.3 WHAT IS A MICROCONTROLLER?

As technology moved from LSI to VLSI, it became possible to build the microprocessor, memory and I/O devices on a single chip. This came to be known as the 'Microcontroller.'

A microcontroller contains a microprocessor and also one or more of the following components.

- Memory
- Analog to Digital (A/D) converter
- Digital to Analog (D/A) converter
- Parallel I/O interface
- Serial I/O interface
- Timers and Counters

Figure 1.5 shows the block diagram of a typical microcontroller, which is a true computer on-chip. The first 4 bit microcontroller was developed by different companies like Hitachi, National, Toshiba, etc. Soon after this, 8 bit microcontrollers were developed by Intel, Motorola, Zilog, Philips, Microchip technology, etc.

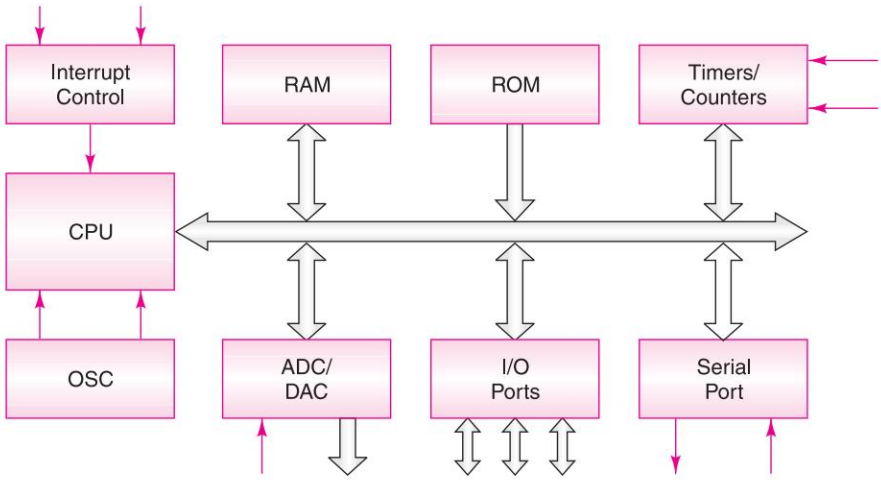


Figure 1.5 *Block diagram of a typical microcontroller*

1.3.1 APPLICATIONS OF MICROCONTROLLER

Microcontrollers have been widely used in home appliances such as refrigerators, washing machines and microwave ovens. It is used in displays, printers, keyboards, modems, charge card phones and also in automobile engines, etc. as controllers.

1.3.2 COMMERCIAL MICROCONTROLLER DEVICES

A brief overview of some commercial microcontrollers, PIC microcontrollers, Intel microcontrollers and Atmel microcontrollers is given in this section. Microcontrollers must be selected depending on the needs of a given application. Table 1.1 lists the various microcontrollers with important features like on-chip memory, number of timers, DMA, A/D converter and UART.

TABLE 1.1 *Microcontrollers with important features*

Device	Register Memory On-chip (bytes)	On-chip program memory ROM/EPROM	Speed MHz	No. of timers/ Counters	No. of I/O lines	On-chip Peripherals
8031 (MCS51-family)	128	ROM less	12	2	32	UART
8051 (MCS51-family)	128	4K ROM	12	2	32	UART
8052 (MCS51-family)	256	8K ROM	12	3	32	UART

(*Contd*)

(*Contd*)

8751 (MCS51-family)	256	8K EPROM	12	3	32	UART
87C58 (MCS51-family)	256	32K EPROM	12-24	3	32	UART
87C51GB (MCS51-family)	256	8K EPROM	12-16	3	48	UART, 8 channel ADC DMA
89C61x2 (MCS51-family)	1024	64K Flash	20-33	3	32	UART
AT89S8252 Atmel	256	8K Flash 2K EPROM	24	3	32	UART SPI
16C74 (Microchip)	192	4K ROM	20	3	32	USART 8 bit ADC SPI
16F874/877 (Microchip)	256	8K	20	3	32	USART 10 bit ADC

SECTION REVIEW

1. List the components in a microcontroller.
2. The 8051 is a _____ bit microcontroller.
3. List few applications of a microcontroller.
4. List few commercial microcontrollers with their important features.
5. _____ Intel microcontroller is ROM less.
6. The 8052 has _____ bytes of on-chip RAM and _____ bytes of on-chip ROM.
7. The 8751 has _____ number of I/O lines.
8. 16F874 microchip operates with speed _____ MHz.
9. List the features of ATMEL AT 89S252 microcontroller.
10. List the on-chip peripherals in an 8051 microcontroller.

1.4 VON NEUMANN (PRINCETON) AND HARVARD ARCHITECTURE

A microprocessor that fetches instruction and data using a single bus is called Von Neumann or Princeton architecture. In Von Neumann architecture, data memory (RAM) and Program memory (ROM) are connected by using single address and data bus as shown in Fig. 1.6.

In Harvard architecture, program memory and data memory are connected using separate address and data bus to achieve fast execution speed for a given clock rate as shown in Fig. 1.7. For example,

8051 microcontroller by Intel and PIC microcontroller by microchip have Harvard architecture. Motorola 68HC11 microcontroller has Von Neumann architecture.

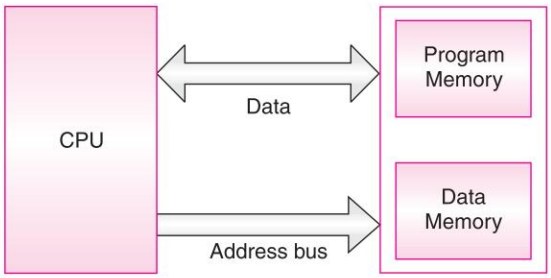


Figure 1.6 *Von Neumann architecture*

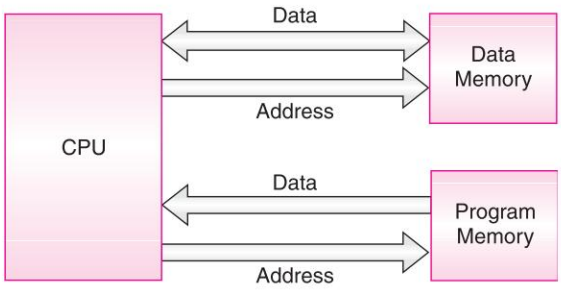


Figure 1.7 *Harvard architecture*

SECTION REVIEW

1. _____ architecture contains single address and data bus.
2. In _____ architecture, two memory access can be made in one instruction cycle.

1.5 RISC AND CISC MACHINES

The microcontrollers with small instruction set are called *Reduced Instruction Set Computer (RISC) machines* and those with complex instruction set are called *Complex Instruction Set Computer (CISC) machines*. Intel 8051 microcontroller is an example of CISC machine and Microchip PIC16F87X is an example of RISC machine. Comparison of features of RISC and CISC is shown in Table 1.2.

TABLE 1.2 *Comparison of RISC and CISC machines*

RISC	CISC
1. Instruction takes one or two cycles	1. Instruction takes multiple cycles
2. Only load/store instructions are used to access memory	2. In addition to load and store instructions, memory access is possible with other instructions also.
3. Instructions executed by hardware	3. Instructions executed by the micro program
4. Fixed format instructions	4. Variable format instructions
5. Few addressing modes	5. Many addressing modes
6. Few instructions	6. Complex instruction set
7. Most of them have multiple register banks	7. Single register bank

SECTION REVIEW

1. CISC instruction takes multiple cycles. True or False?
2. RISC has variable format instruction. True or False?
3. CISC has multiple register banks. True or False?
4. RISC has few addressing modes and less number of instructions. True or False?

1.6 COMPUTER SOFTWARE

A set of instructions written in a specific sequence for the computer to solve a specific task is called a *program*, and *software* is a collection of programs. The program stored in the computer memory in the form of binary numbers is called *machine instructions*. The machine language program is called *object code*.

For example, in 8051 microcontroller, the instruction CLR A is represented by binary code 11100100 (E4 in hexadecimal). Because it is difficult to write programs in sets of 0's and 1's, assembly language was then developed to simplify the programming job. Machine language and assembly language are low level languages and both are microprocessor specific. Assembly language programs are written using assembly instructions. An assembly instruction is the mnemonic representation of machine instructions.

For example, in the instruction CLR A, CLR stands for clear and A represents the accumulator. This symbol suggests the operation of storing 00H in the accumulator. The assembly language program that a programmer enters is called the *source program* or *source code*. A software program called an *assembler* is then developed to translate the program (source code) written in assembly language into machine language (object code), which is compatible with the microprocessor being used in the system as shown in Fig. 1.8. Mnemonics are specific to microprocessors and each microprocessor has its own assembler.

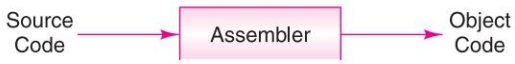


Figure 1.8 *Assembler*

The drawbacks in assembly language programming are:

- The programmer has to be very familiar with the processor of the computer in which the program is to be executed.
- It is difficult to understand an assembly language program without the use of comments.

To avoid the drawbacks of assembly language programming, high-level languages such as Fortran, Pascal, C and C++ were developed. A program written in high level language is also called a source program. A software program called a *compiler* or an *interpreter* was then developed to translate the source program into machine language as shown in Fig. 1.9.

Figure 1.9 *Compiler/Interpreter*

Thus, a compiler or an interpreter translates high level language program into machine language. Each microprocessor needs its own compiler or an interpreter for each high level language. The primary

difference between a compiler and an interpreter lies in the process of generating machine code. The compiler reads the entire program, translates it into object code and then it is executed by the processor. On the other hand, the interpreter takes one statement of a high level language program as input and translates it into object code and then executes. One of the major drawbacks of high level languages is that the machine code compiled from high level language program cannot run as fast as machine codes of assembly language program. For this reason, real time application programs are written in assembly language.

SECTION REVIEW

1. _____ languages are low level languages.
2. In assembly language, programs are written using mnemonics. True or False?
3. _____ software translates assembly language to object code.
4. List the drawbacks of assembly language programming.
5. Differentiate compiler and interpreter.

1.7 AN OVERVIEW OF EMBEDDED SYSTEM

In our daily lives, we are surrounded by number of embedded systems, such as TVs, DVD players, mobile phones, washing machines, digital cameras and automobiles. In this section, we will study definition of embedded systems, architecture of embedded systems, and embedded operating systems.

1.7.1 WHAT IS AN EMBEDDED SYSTEM?

An embedded system can be defined as a combination of computer hardware and software that does a specific job. Embedded systems are used in consumer electronics, food processing industry, chemical plants, cement plants, biomedical equipments, telecommunication and security. The embedded software that is executed for specific job is called *firmware*.

1.7.2 CHARACTERISTICS OF EMBEDDED SYSTEMS

Based on the processor and software, various types of embedded systems exist. Reliability, cost effectiveness, low power consumption, fast execution time, efficient use of memory, and processing power are the characteristics of most of the embedded systems.

Reliability Reliability of hardware and software is most important in embedded systems. The embedded system should reboot and rest by itself without human intervention during failure. The co-design of hardware and software is given importance in case of embedded system to meet these requirements.

Cost effectiveness The embedded systems are developed to meet the requirements of specific applications, in mass markets, and hence, keeping the product cost reasonable becomes essential. To meet the requirements, the system is developed using general-purpose processor during prototype. Then, an application specific integrated circuit is used to reduce the hardware and cost.

Low power consumption Many embedded systems are battery operated. Power consumption has to be limited to increase the life of the battery. This can be achieved by reducing the number of hardware components or by designing the processor to revert to sleep mode when no operation is to be performed.

Fast execution time In real time embedded systems, certain tasks must be performed within a specific time. To meet the performance constraints of real time systems, special operating systems known as real time operating systems run on these embedded systems. To fulfill the performance requirement, the code should be optimal. The software is generally developed using high-level languages, some computationally intensive units are developed in assembly language.

Efficient use of memory Most of the embedded systems contain only ROM and RAM without any secondary storage. Flash memory is used to store the program, including the operating system. Most of the microcontrollers and digital signal processors are available with on-chip flash memory.

Processing power The number of instructions executed per second is the processing power of the processor. Most of the processors execute one instruction in one clock cycle. Therefore, the processing power is in terms of million instructions per second (MIPS).

1.7.3 EMBEDDED SYSTEM ARCHITECTURE

Embedded system is a dedicated computer-based system for an application. The software is embedded in the read only memory. The hardware of an embedded system consists of the following six main components:

1. Central processing unit
2. Memory
3. Input unit
4. Output unit
5. Application specific circuitry
6. Communication channels

Central processing unit The central processing unit (CPU), which is also called processor is the heart of the embedded system. The CPU can be any of the following

- Microprocessor
- Microcontroller
- Digital signal processor
- Application specific processor

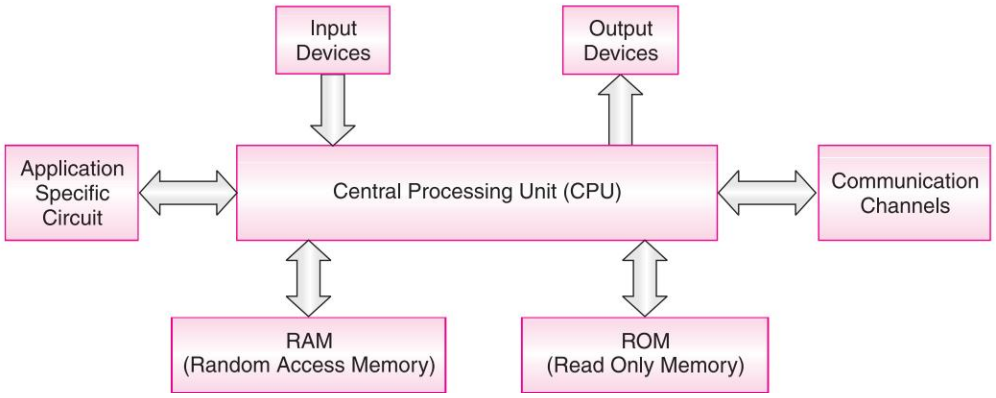


Figure 1.10 *Block diagram of an Embedded System*

Microprocessor Microprocessor is a single VLSI chip that has a CPU. It is used when large embedded software is to be executed. Microprocessors are powerful and result in faster processing of instructions. Table 1.3 lists the important microprocessors used in the embedded systems.

Microcontrollers Microcontrollers are also single VLSI chip, which have a CPU, memory, parallel ports, serial ports, analog to digital converters and timers. Microcontrollers are used when small-embedded software is to be executed and stored in its internal memory. Table 1.3 lists the important microcontrollers used in the embedded systems.

Digital signal processors Digital signal processors are used where signal processing is involved such as filtering, signal conversion from time to frequency domain, etc., and in applications like biomedical and speech signal processing. Table 1.3 lists the important digital signal processors used in the embedded systems.

Application specific processors In application specific processors, processors are designed for running the application specific tasks. They are the best choice where faster solution is required, as compared to microprocessor and microcontroller. Application specific processors are used in mobile phones, TV decoders, etc.

TABLE 1.3 *Important processors used in Embedded Systems*

Microprocessors	Microcontrollers	Digital Signal Processors
68HCxxx Motorola	68HC11xx Motorola	5600xx Motorola
80x86 Intel	8051, 80196, 89c61x2 Intel	TMS320Cxx Texas
SPARC SUN	PIC16F84, PIC16F874 Microchip	SHARC Analog Devices
Power PC IBM	ARM 7 ARM	ADSP 21xx Analog Devices

Memory An embedded system contains two types of semiconductor memory—random access memory (RAM) and read only memory (ROM). Random access memory stores user program and data. Firmware (embedded software) is stored in read only memory. When power is switched on, the processor executes the embedded software stored in read only memory.

Input Devices Some embedded systems have input devices like small keypad, analog to digital converters, switches whereas others do not have any input device for user interaction.

Output Devices Embedded systems have output devices like seven segment displays' light emitting diodes (LED) and liquid crystal displays (LCD) to display important parameters.

Application Specific Circuitry Depending on the application, embedded system contains transducers, sensors, amplifiers, and current to voltage conversion circuits. Function of these circuits is to convert incoming signal to match the range of Analog-to-Digital (A/D) converter.

Communication Interfaces An embedded system can interact with other embedded systems using serial communication bus like RS232, RS422 and IEEE bus.

1.7.4 FULL-CUSTOM PROCESSOR, SEMI-CUSTOM PROCESSOR AND PROGRAMMABLE LOGIC DEVICES

Processors are implemented on an integrated circuit (IC). IC, often called as chip consists of a set of transistors interconnected with other devices. There are different processes to fabricate semiconductors namely nMOS, pMOS and CMOS. The most widely used process is CMOS (complementary metal oxide semiconductor).

Semiconductors devices are built with different layers. The bottom-most layers are of transistors. The middle layers form logic components. The top-most layers connect these logic components with wires to build larger circuits like a processor.

Full-custom design Full-custom IC design is also referred to as Very Large Scale Integration (VLSI) design. Full-custom Design is the name given to the technique where the function and layout of every transistor is optimised. A microprocessor is an example of full custom IC. In this design, care has been taken to squeeze every last square of micron of chip space. Full-custom ICs are the most expensive to manufacture and to design.

Semi-custom design In case of semi-custom ICs, all of the logic cells are pre designed and some of the mask layers are custom designed. The masks for the transistor and gate levels are already built. The remaining design is to connect these gates to achieve a particular implementation. They provide good performance with much less cost than full-custom ICs.

Programmable logic devices In Programmable Logic Devices (PLDs), all of the logic cells are pre-designed and none of the mask layers are customised. There are two types in this category, namely Programmable Logic Devices (PLD) and Field Programmable Gate Array (FPGA). In case of PLDs, all layers already exist. One can purchase the IC and program it for intended design. The programming consists of creating or destroying connections between connected gates, either by blowing a fuse, or setting a bit in a programmable switch. A new inclusion to this family is Field Programmable Gate Arrays (FPGAs) that offer more general connectivity among blocks of logic, rather than just arrays of logic as with PLDs. These ICs are bigger in size, have higher unit cost and are slower compared to full-custom or semi-custom but provide reasonable performance with reduced design time.

1.7.5 DESIGN CHALLENGES IN AN EMBEDDED SYSTEM

The embedded system designers and developers have common design challenges like co-design, embedding an operating system, and optimising the code.

Co-design In an embedded system design, the task is implemented using hardware and software. The design engineer has to decide which module needs to be implemented in hardware and which module is to be realised by software. Hardware implementation imposes the constraints of size and cost. Software implementation imposes the constraints of memory size and performance. To design an efficient embedded system, co-design is a challenge.

Choice of operating system In the design of an embedded system, the designer has the option to develop the software in assembly and C without using an operating system. This will result in efficient code, saving processing power and memory. The other option is to use a readily available operating system, so that the designer can focus on application software development. This saves development time and cost.

Optimizing of code In the design of embedded system, memory and execution time are the main constraints. It is a challenge for the designer to optimise the code to meet the constraints.

1.7.6 EMBEDDED OPERATING SYSTEM

In the olden days, development of application embedded software was done in assembly language. Debugging and maintaining assembly language program were very difficult and time consuming. Hence, most of the development of application embedded software is now done using object oriented languages. Embedded operating systems are designed to work on semiconductor memory with limited processing power, since there is no secondary storage device in an embedded system. The advantages of using operating systems in embedded systems are

- The program can be written in a high-level language. Object oriented programming languages particularly C++ and Java are extensively used for embedded software development.
- The programmer can focus on application program rather than memory and I/O management program.

Operating systems used in embedded systems can be broadly divided into three types:

Non real time embedded operating systems These operating systems are suitable for soft real time embedded systems to perform tasks like memory management, I/O management, etc. They are not suitable for hard real time applications. Examples are embedded windows XP, and embedded Linux.

Real time operating systems These operating systems are suitable for hard real time operating systems. They are used to provide the necessary system calls for real time deadlines. Examples are OS/9, RT Linux, and Vx works.

Mobile/handheld operating systems Operating systems that are used to work in mobile environments are known as mobile operating systems. Examples are Embedded Windows NT, window CE, palm OS, and symfian OS.

In embedded systems, the operating system and the application software are integrated and stored in the memory of the embedded system.

1.7.7 REAL TIME EMBEDDED SYSTEMS

In most embedded systems, it is important to perform some of the computations in a timely manner. Such embedded systems in which strict deadlines are imposed to complete specific task are called *real time embedded systems*.

In real time systems, an additional programming is necessary to meet the deadline constraints in its functions. An operating system, which handles these multiple tasks with real time constraints, is called as *real time operating system* (RTOS). Note that real time system refers to embedded system that exhibits real time characteristics. For example, a cell phone decodes audio signal and converts digital signal to voice. All this takes place in a defined time period, else there will be a delay in reaching to the listener. Other systems that have timing requirements are process control system, robots, networks and multimedia systems.

SECTION REVIEW

1. Name few applications of an embedded system.
2. The embedded software executed for a specific job is called _____.
3. List the characteristics of an embedded system.
4. The processing power of an embedded system is in terms of MIPS. True or False?
5. List the components of an embedded system.
6. Name types of processors used in an embedded system.
7. Name few digital signal processors used in an embedded system.
8. An embedded system can interact with other embedded systems. True or False?
9. List the different processes used to fabricate semiconductors.
10. Full custom IC is also referred as _____ integration.
11. Differentiate between PLD and FPGA.
12. Name most common challenges in designing an embedded system.
13. List different types of operating system used in an embedded system.
14. The operating system used to work in a mobile equipment is known as _____.
15. Define real time embedded system (RTES).

CHAPTER SUMMARY

This chapter describes the major components of a computer system, namely CPU, memory and I/O devices. A memory is where software programs and data are stored. A computer may contain semiconductor, magnetic and optical memory.

This chapter further explains the following:

- Characteristics of volatile and nonvolatile memory
- Fundamental differences between microprocessor and microcontroller, application of microcontroller and types of microcontroller in the embedded market
- Difference between Von Neumann and Harvard architecture
- Computer languages such as high-level language, assembly language and machine language.

The chapter concludes with an overview of the embedded systems.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. Which was Intel's first microprocessor?
   (a) 8085 (b) 8086 (c) 4004 (d) 8088
2. The basic operations performed by a computer are _____ .
   (a) Arithmetic operations (b) Logical operations (c) Storage (d) All the above

3. Word length of 8085 microprocessor is _____ .
   (a) 4 bit (b) 8 bit (c) 16 bit (d) 32 bit
4. Programs stored in ROM are called _____ .
   (a) Hardware (b) Software (c) Firmware (d) None of these
5. Integrated circuits are classified on the basis of _____ .
   (a) Manufacturing company (b) Number of transistors
   (c) Types of microprocessor (d) None of these
6. A compiler _____ .
   (a) Translates assembly level language into machine language
   (b) Translates high level language into machine language
   (c) Translates high level language into assembly language
   (d) None of these
7. Machine language _____ .
   (a) Is the language in which programs are written using mnemonics
   (b) Is the only language understood by the microprocessor
   (c) Is the language in which programs are written using ASCII code
   (d) None of these
8. An assembler _____ .
   (a) Translates Mnemonics into machine language
   (b) Translates high level language into machine language
   (c) Translates ASCII code into machine language
   (d) None of these
9. A source program is a _____ .
   (a) Program written in a machine language
   (b) Program to be translated into assembly language
   (c) Program written using mnemonics
   (d) None of these
10. Which of the following is not an output device?
    (a) Scanner (b) Printer (c) Flat screen (d) Speaker
11. _____ address lines are required to address 8K × 8 bit memory
    (a) 13 (b) 14 (c) 10 (d) 16
12. PROM programmer can be used for _____ .
    (a) Erasing the contents of EPROM (b) Reconstructing the contents of EPROM
    (c) Duplicating the contents of EPROM (d) None of the above
13. Which of the following devices can understand the difference between data and programs?
    (a) Input device (b) Memory (c) Microprocessor (d) Output device
14. In a dynamic read/write memory, _____ is capable of storing a single binary bit.
    (a) Capacitor (b) Flip flop (c) Resistor (d) Inductor
15. A set of flip flops integrated together is called _____ .
    (a) Counter (b) Register (c) Adder (d) Subtractor

16. Which of the following is the unit for external storage device?
    (a) Bytes (b) Hertz (c) Clock cycles (d) None of the above
17. _____ is a semiconductor memory.
    (a) Bubble (b) Dynamic R/W (c) Both (a) & (b) (d) None of the above
18. Embedded system contains _____ .
    (a) ROM & RAM (b) Secondary memory (c) Both (a) and (b) (d) None of the above
19. Full custom IC design is referred as _____ .
    (a) Small scale integration design (b) Medium scale integration design
    (c) Large scale integration design (d) Very large scale integration design
20. In a real time system, the constraint is _____ .
    (a) Memory size (b) Time (c) Both (a) & (b) (d) None of the above

REVIEW QUESTIONS

1.1 What is a computer? Explain different sections of a processor.
1.2 Why are input devices needed? Mention any three input devices.
1.3 Why are output devices needed? Mention any three output devices.
1.4 List the salient features of a microcontroller.
1.5 Which register is used to keep track of the address of the next instruction to be executed?
1.6 List the specific features of a microcontroller using diagrams.
1.7 Give a brief summary of the evolution of Intel microprocessor.
1.8 List different types of semiconductor memory and explain their differences.
1.9 What are the differences between PROM, EPROM, EEPROM and Flash memory.
1.10 Compare the differences between RISC and CISC.
1.11 What are the advantages of RISC and CISC architecture?
1.12 What are the differences between Von Neumann and Harvard architecture.
1.13 Distinguish Harvard and Princeton architecture with diagrams.
1.14 What is the difference between source code and object code?
1.15 What are the advantages and disadvantages of assembly language programming?
1.16 List the merits and demerits of high-level languages.
1.17 Differentiate between
    (a) Interpreter and compiler (b) Assembler and compiler
1.18 What is an embedded system? Explain the architecture of an embedded system with a block diagram.
1.19 Compare the differences between full-custom IC, semi-custom IC and programmable logic devices.
1.20 List and define the characteristics of an embedded system.
1.21 List and explain important processors used in an embedded system.
1.22 Explain the design challenges encountered in an embedded system.
1.23 List and define important types of IC technologies. What are the benefits of using these technologies?
1.24 Explain different types of embedded operating systems.
1.25 Differentiate between embedded and real time operating systems.

2

THE 8051 MICROCONTROLLER

Learning Objectives

After you have completed this chapter, you should be able to

- Explain the features of the 8051 microcontroller
- Explain the architecture of the 8051 microcontroller
- Explain the pin diagram of the 8051 microcontroller
- Explain the memory organisation of the 8051 microcontroller
- Define the stack and explain its uses
- Explain the additional features of the 8052 microcontroller

2.1 FEATURES OF 8051

A microcomputer is a computer implemented on a very large scale integration chip. The 8051 is the first microcontroller of the MCS-51 family, introduced by Intel Corporation at the end of the 1980s. The 8051 family with its many enhanced members enjoys the largest market share, estimated to be about 40%, among the various microcontroller architectures. The architecture and pin diagram of the 8051 are presented in this chapter. The block diagram of the 8051 microcontroller is as shown in Fig. 2.1.

The salient features of the 8051 microcontroller are given below:

- 8 bit CPU
- On-chip clock oscillator
- 4 Kbytes of on-chip program memory
- 256 bytes of on-chip data random access memory

- 64 Kbytes of program memory address space
- 64 Kbytes of data memory address space
- 32 bidirectional I/O lines can be either used as four 8 bit ports or 32 individually addressable I/O lines
- Two 16 bit timers/counters
- 16 bit address bus multiplexed with port 0 and port 2 and 8 bit data bus multiplexed with port 0
- Full duplex asynchronous receiver transmitter
- Five-vector interrupt structure with two priority levels

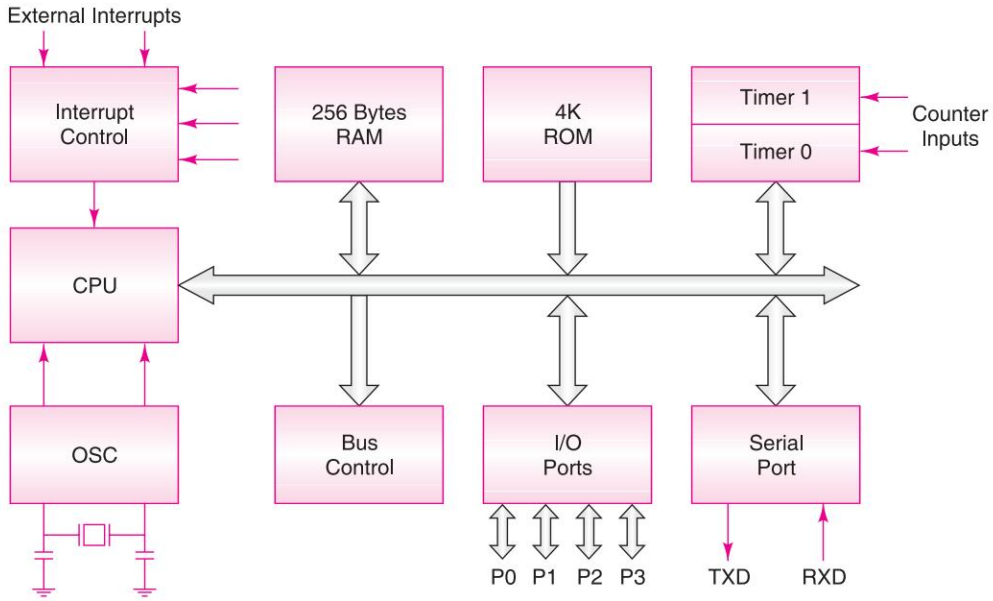


Figure 2.1 *Block diagram of 8051 Microcontroller*

2.2 ARCHITECTURE OF 8051

Figure 2.2 shows a functional block of the internal operations of an 8051 microcontroller. The 8051 includes an 8 bit CPU, memory, four 8 bit I/O ports, two timers/counters and a Universal Asynchronous Receiver Transmitter (UART).

2.2.1 PROCESSOR

The processor includes arithmetic and logic unit, instruction decoder and timing generation unit, Accumulator (A or Acc), B register and status register.

Arithmetic and Logic Unit The Arithmetic and Logic Unit (ALU) performs the computing functions. The accumulator is an 8 bit register. In arithmetic and logical operations, one of the operands is in 'A' register. After the arithmetic/logical operations, are performed, the result is stored in 'A' register and this affects various flags namely Carry (C), Auxiliary Carry (AC), Overflow (O), and Parity (P) of status register.

Instruction Decoder and Control The instruction decoder and control are parts of the timing and control unit. When an instruction is fetched from program 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. The timing generation and control unit synchronises all the microcontroller operations with the clock and generates control signals necessary for communication between the processor and peripherals.

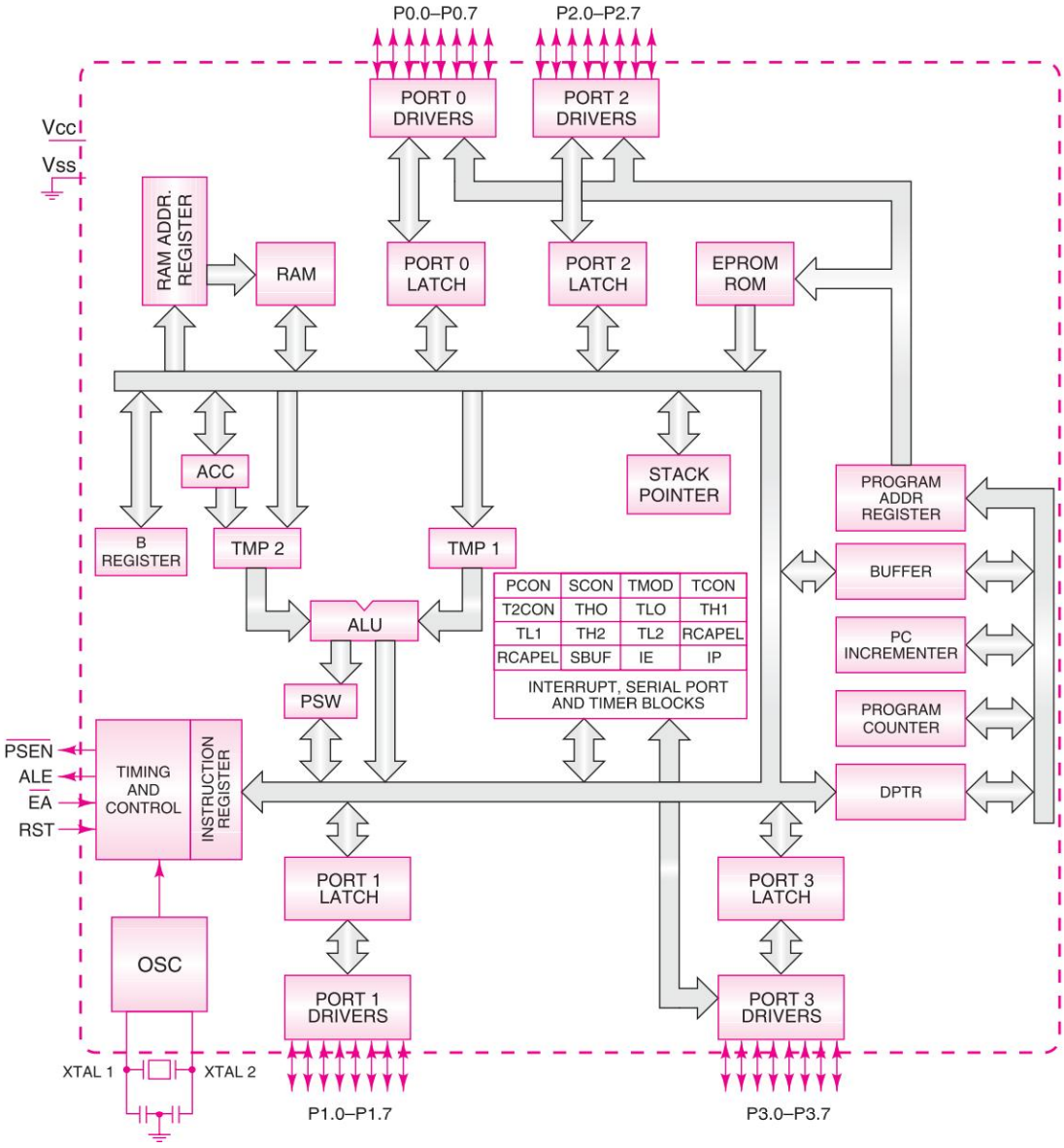


Figure 2.2 *Functional block diagram of 8051 (Courtesy Intel)*

CPU Registers

'A' Register (E0H) Similar to any Intel microprocessor, the 8051 has an 8 bit A and in the instruction, it is referred as 'A'. The accumulator is used in all arithmetic and logical operations and has direct connection to ALU. One of the operands is stored in the accumulator. After the operation is performed, the result is stored in the A. In multiplication operation, one of the 8 bit operands is stored in 'A' register. After the operation, it stores the lower byte of the result in 'A' register. In division operation, it holds an 8 bit dividend and after the operation, the quotient is stored in the accumulator. It is also used in indexed addressing mode to access information from program memory. 'A' is bit addressable register.

'B' Register (F0H) The 8 bit 'B' register is used during multiply and divide operations. In multiplication operation, one of the 8 bit operands is stored in 'B' register. After the operation, it stores the higher byte of the result in 'B' register. In division operation, it holds 8 bit divisor and after the operation the remainder is stored in 'B' register. For other instructions, it can be used as an 8 bit general purpose register. 'B' is bit addressable register.

Program Status Word (D0H) The 8 bit Program Status Word (PSW) register contains the arithmetic status of the ALU and the bank selects bits for the data memory. After the arithmetic and logic operations, the C, AC, P, and O flags of PSW register are set or reset according to the result. In subtraction, the C and AC bits operate as borrow and digit borrow flag respectively. PSW is bit addressable register

CY	AC	F0	RS1	RS0	OV	----	P
Bit 7							Bit 0

Bit 7: Carry/borrow bit When two 8 bit operands are added, the result may exceed 8 bit (may exceed 255 or FF H), and the 9th bit is copied in the carry bit. During subtraction, if the borrow occurs, the carry bits is set and otherwise, it is cleared.

Bit 6: Auxiliary carry/borrow bit This bit indicates a carry from the lower nibble (lower 4 bit) during 8 bit addition. If auxiliary carry flag is set, it means there is a carry from 3rd to 4th bit position. In subtraction, if there is a borrow from 4th bit to 3rd bit position, then AC is set, else it is cleared.

Bit 5: F0 Flag 0 is available to the user for general purpose.

Bit 4-3: RS1:RS0 Register Bank Select bits

11 = Selects register bank 3

10 = Selects register bank 2

01 = Selects register bank 1

00 = Selects register bank 0

Each bank contains eight 8 bit registers

Bit 2: OV OV flag is used to detect errors in signed arithmetic operations. When two signed numbers are added, if the result exceeds the destination, overflow flag is set, else it is reset. OV is set, if there is a carry from D6 to D7, but no carry from D7, or if there is a carry from D7 but no carry from D6 to D7.

Bit 1: Undefined flag

Bit 0: Parity flag P flag is set, if the result contains an even number of 1 bit, else it is reset, if the result contains an odd number of 1 bit.

Stack Pointer (81 H) Stack Pointer is an 8 bit register. It contains the address of the data item on the top of the stack. It is incremented before the data is stored. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialised to 07H after a reset. The operation and instruction associated with the stack will be discussed in detail in Chapter 3.

Data Pointer (DPH-83 H and DPL-82 H) The Data Pointer (DPTR) consists of two 8 bit registers—a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16 bit address. It is used to furnish address information for internal and external program memory and for external data memory.

Program Counter Program Counter (PC) is a 16 bit register. The 16 bit program counter specifies the address of the next instruction to be executed. After reset, the PC will be set to 0000H and the CPU will start executing the first instruction stored at program memory location 0000H. The 8051 fetches the instruction one byte at a time and after fetching, it increments the PC by 1.

2.2.2 MEMORY

The 8051 devices have 4 Kbytes on-chip program memory and 256 bytes of on-chip data random access memory. The program memory is used to hold the start up program that will be executed when the 8051 is powered up. The 8051's on-chip data random access memory is organised as follows:

First 128 bytes:	00H to 1FH	Register Banks
	20H to 2FH	Bit Addressable RAM
	30H to 7FH	General Purpose Registers
Next 128 bytes:	80H to FFH	Special Function Registers

The 256 bytes of internal memory are organised as shown in Fig. 2.3. Data memory is divided into two groups of memory size 128 bytes each. In the first 128 bytes, the user data can be stored in register banks, bit addressable RAM and in general purpose registers. The next 128 bytes are special function registers.

Figure 2.3 *Internal RAM organisation*

Register Banks (00H to 1FH) The lowest 32 bytes are grouped into four banks of eight registers. RS1 and RS0 of (bit 4-3) program status word select the bank. Each bank contains 8 general purpose registers R0–R7 (R0, R1, R2, R3, R4, R5, R6, and R7). These registers are similar to the general-purpose registers of the microprocessor. These registers are used in instructions such as:

ADD A, R2; adds the value contained in R2 to the accumulator

If RS1 = 0 and RS0 = 0, Bank 0 is selected, and then R2 is the memory location 02H of the Internal RAM. The following instruction has the same effect as the above instruction.

ADD A, 02H

Bit Addressable RAM (20H to 2FH) The 8051 supports a special feature which allows access to bit variables. The bit addressable area of the RAM is just 16 bytes of internal RAM located between 20H and 2FH. This is where individual memory

bits in internal RAM can be set or cleared. In total, there are 128 bits; addressed 00H to 7FH. Bit variables can have a value of 0 or 1.

A bit variable can be set with a command such as SETB and cleared with a command such as CLR. Example instructions are:

```
SETB 25H        ; sets the bit 25H (becomes 1)
CLR 25H         ; clears the bit 25H (becomes 0)
```

Note

1. *Bit 25H is actually bit 5 of internal RAM location 24H.*
2. *Bit addressing can also be performed on some of the SFR registers, which will be discussed later on.*

TABLE 2.1 *Bit address of internal RAM*

Byte Address	Bit Address 7-0	Byte Address	Bit Address *7 - 0*
2F	7F-78	27	3F-38
2E	77-70	26	37-30
2D	6F-68	25	2F-28
2C	67-60	24	27-20
2B	5F-58	23	1F-18
2A	57-50	22	17-10
29	4F-48	21	0F-08
28	47-40	20	07-00

General Purpose RAM (30H to 7FH) These 80 bytes of internal RAM memory are available for general-purpose data storage. Access to this area of memory is fast as compared to access to the main memory, and instructions with single byte operands use this area for storage. However, the system stack uses these 80 bytes and in practice, little space is left for general storage. The general purpose RAM can be accessed using direct or indirect addressing modes.

SFR Register The SFR registers are located within the internal memory in the address range 80H to FFH, as shown in Fig. 2.3. Not all locations within this range are defined. Each SFR has a very specific function. They have an address (within the range 80H to FFH) and a name, which reflects the purpose of the SFR. Although 128 bytes of the SFR address space is defined, yet only 21 SFR registers are defined in the standard 8051. Undefined SFR addresses should not be accessed, as this might lead to some unpredictable

results. Note that some of the SFR registers are bit addressable. SFRs are accessed just like normal internal RAM locations. CPU and internal peripheral modules use special function registers for controlling the desired operation of the device. The special function registers contain input and output ports, control register for interrupts, timers, serial ports, etc. as shown in Table 2.2.

2.2.3 DIGITAL I/O PORT AND PERIPHERALS

It contains four 8 bit parallel ports, named as port 0, port 1, port 2, and port 3. Ports are used to send or receive the data. Each bit of the port can be configured as an input or an output and also port 0 is used as low order address and data pins (AD0–AD7), Port 2 is used as high order address pins (A8–A15) and port 3 is used by timers, serial port, external interrupt and for sending control signals ($\overline{RD}$, $\overline{WR}$) to external data memory. It contains two versatile timers—timer 0 and timer 1. Both are 16 bit timers/counters and each timer consists of two 8 bit registers. It supports serial data transmission using universal asynchronous receiver transmitter.

TABLE 2.2 *Special function registers (Courtesy Intel)*

Symbol	Name	Address
*ACC	Accumulator	E0H
*B	B Register	F0H
*PSW	Program Status Word	D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	A0H
*P3	Port 3	B0H
*IP	Interrupt Priority Control	B8H
*IE	Interrupt Enable Control	A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8BH
TL1	Timer/Counter 1 Low Byte	8DH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

*bit addressable

SECTION REVIEW

1. The 8051 has _____Kbyte of program memory address space.
2. The 8051 has _____number of vector interrupts with _____ priority levels.
3. In division operation, an 8 bit divisor is stored in _____ register.
4. In division operation, _____ register is used to store quotient.
5. _____ flag detects errors in signed arithmetic operations.
6. _____ flag is set, if there is a borrow from fourth bit to third bit.
7. Stack pointer is a _____ bit register.
8. Stack pointer is initialised to _____ after reset.
9. Bit 1 of program status word is _____ flag.
10. _____ 16 bit register contains the address of program and external data memory.
11. _____location to _____ location is bit addressable space in internal RAM.
12. The 8051 contains _____ number of register banks and each register bank contains _____ number of 8 bit registers.
13. List bit addressable registers in the 8051 microcontroller.
14. Name any five byte-addressable registers.
15. _____ and _____ control signals are used to interface external data memory.
16. In the 8051, maximum power dissipation rating is _____.

2.3 PIN DIAGRAM OF 8051

The 8051 microcontroller is a 40 pin DIP as shown in Fig. 2.4. The crystal frequency is the basic clock frequency of the microcontroller. The 8051 requires a +5V single power supply and is designed for 1 MHz minimum clock frequency to 16 MHz maximum clock frequency. A brief discussion of these pins is given in this section.

Vcc (Pin no 40) Vcc pin is connected to +5 V power supply with rated current 125 mA. In 8051, maximum power dissipation rating is 1 W.

Vss (Pin no 20) Vss pin is connected to Ground reference.

XTAL2 (Pin no 18) The output of the crystal oscillator circuit is connected as shown in Fig. 2.5. 30 pF disc capacitors are recommended when 12 MHz quartz crystals is used. In case of external clock, clock is connected to XTAL2.

XTAL1 (Pin no 19) The input of the crystal oscillator circuit is connected to XTAL1. In case of external clock, this pin is connected to ground.

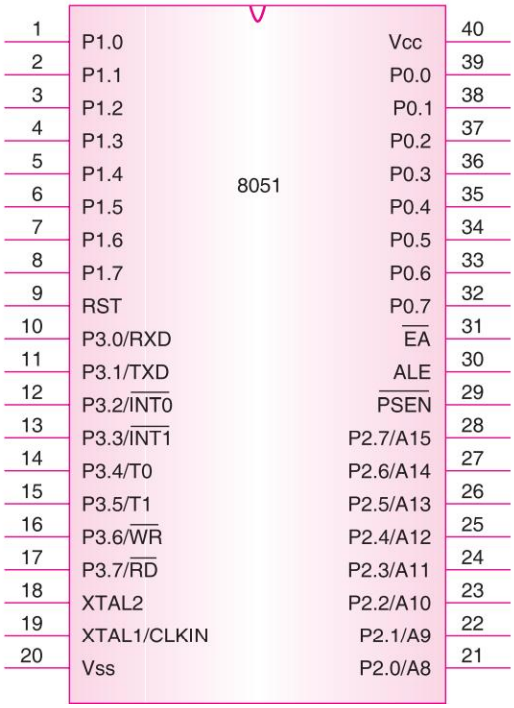


Figure 2.4 *Pin diagram of 8051*

RST (Pin no 9) The microcontroller provides a reset mechanism to establish initial conditions. The special function registers and few CPU registers must be initialised before the microcontroller can operate properly.

Reset circuit is as shown in Fig. 2.5. For resetting 8051, RST pin is made high for two machine cycles (24 oscillator periods). Table 2.3 lists the SFRs and their reset values.

Port 0 (Pins 32-39) Port 0 serves as true bi-directional I/O port, and also as low order address and data bus for external memory.

Port 1 (Pins 1-8) Port 1 has no dual functions. It serves as quasi bi-directional 8 bit I/O port.

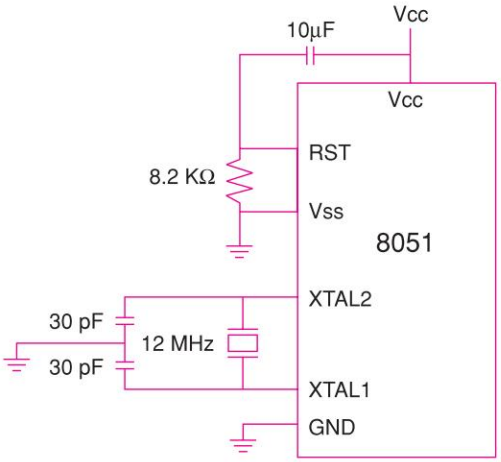


Figure 2.5 *8051 connected to 12 MHz crystal and reset circuit*

TABLE 2.3 *Reset values of the SFRs*

SFR	Reset Value
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
P0–P3	FFH
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
SCON	00H
IP	xxx00000B
IE	0xx00000B
PCON	0xxxxxxxB

x-unknown

Port 2 (Pins 21-28) Port 2 is also an 8 bit quasi bi-directional I/O port. The alternate use of port 2 is to serve as high order address bus for external memory.

Port 3 (Pins 10-17) Port 3 is also an 8 bit quasi bi-directional I/O port. The alternate functions of port 3 are related to external interrupts, serial ports, timers and read/write control signals for external data memory. Table 2.4 lists the alternate functions of port 3.

ALE (Pin 30) Address latch enable pin is used to demultiplex AD0–AD7 of port 0. This is a positive going pulse generated every time during external memory access. This signal is used primarily to latch low order address from the multiplexed bus and generate a separate set of eight address lines A0–A7 as shown in Fig. 2.6.

$\overline{\text{PSEN}}$ (Pin 29) Program strobe signal is the output control signal. It remains low while fetching external program memory. During the internal program execution, the condition of this pin is high.

$\overline{\text{EA}}$ (Pin 31) If external access ($\overline{\text{EA}}$) pin is held high, it selects internal program memory

for address 0000H to 0FFFH. Beyond this address (1000H to FFFFH), it selects external program memory as shown in Fig. 2.8(a). Else if this pin is held low, it selects only external program memory for address 0000H to FFFFH.

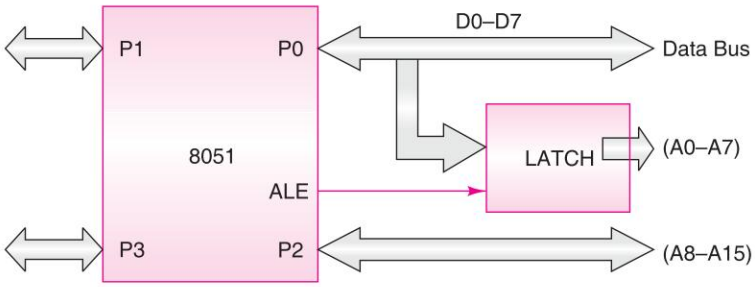


Figure 2.6 *Data and address multiplexing using ALE*

TABLE 2.4 *Alternate functions of port 3*

P3.0	Pin 10	RXD (serial input port)
P3.1	Pin 11	TXD (serial output port)
P3.2	Pin 12	$\overline{\text{INTO}}$ (external interrupt 0)
P3.3	Pin 13	$\overline{\text{INTI}}$ (external interrupt 1)
P3.4	Pin 14	T0 (timer/counter 0 external input)
P3.5	Pin 15	T1 (timer/counter 1 external input)
P3.6	Pin 16	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	Pin 17	$\overline{\text{RD}}$ (external data memory read strobe)

SECTION REVIEW

1. In an oscillator circuit, _____ disc capacitor is recommended when 12 MHz crystal is used.
2. After reset, the value of P0–P3 will be _____.
3. _____ port has no dual functions.
4. _____ port contains control signals for external data memory.
5. _____ pin is used to demultiplex AD0–AD7.
6. _____ to _____ is the address of internal program memory.
7. The 8051 contains _____ external interrupts.
8. _____ port serves as low order address bus and _____ port serves as high order address bus for external memory.
9. For resetting 8051, RST pin is made high for _____ oscillator periods.

2.4 MEMORY ORGANISATION

The 8051 devices have 4 Kbytes of on-chip program memory and 256 bytes of on-chip data random access memory. In addition to internal memory, 64 Kbytes of program memory and 64 Kbytes of data memory can be interfaced with 8051 using port 0, port 2, port 3.6, port 3.7, $\overline{\text{PSEN}}$ and $\overline{\text{EA}}$ pins. Program memory is accessed using the program counter (PC) and A, or DPTR and A. Data memory is accessed using DPTR, R0 and R1 register. In 8051, port 0 and port 2 provide 16 bit address to access external memory. Port 0 provides the lower 8 bit address (A0–A7) and 8 bit data (D0–D7). This is called address/data multiplexing. Port 2 provides the upper 8 bit address (A8–A15). ALE pin of 8051 and 74LS373 latch are used to demultiplex AD0–AD7. Data, address and control buses for 8051 to interface external memory are as shown in Fig. 2.7.

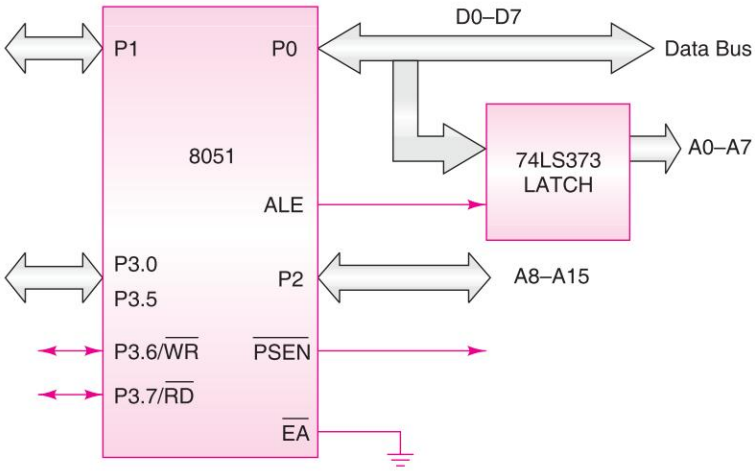


Figure 2.7 *Data bus, address bus and control signals of 8051*

When $\overline{\text{EA}}$ is connected to the ground, 8051 fetches instructions from external ROM by using $\overline{\text{PSEN}}$. Then, the address of external ROM is 0000H to FFFFH. When $\overline{\text{EA}}$ is connected to Vcc, then it fetches instructions from on-chip ROM 4 Kbytes. The address of on-chip ROM is 0000 to 0FFFH. For address 1000H to FFFFH, it fetches instructions from external ROM. In this mode, 60 Kbytes of external ROM can be interfaced with 8051 microcontroller as shown in Fig. 2.8(a). The data can be stored both in internal and 64 Kbytes of external data memory as shown in Fig. 2.8(b).

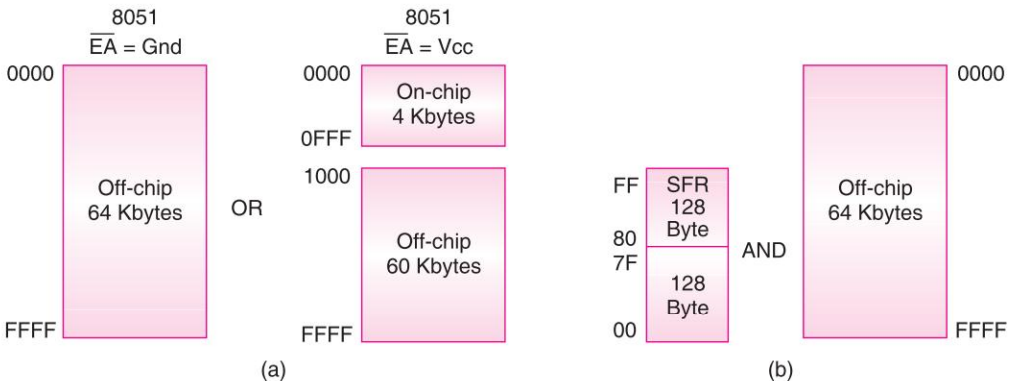


Figure 2.8(a) *On-chip and off-chip program Rom,* **(b)** *On-chip and off-chip data memory*

SECTION REVIEW

1. The 8051 device has _____ on-chip program memory and _____ on-chip data memory.
2. List the control signals used to interface the external memory.
3. _____ Kbytes of program memory and _____ Kbytes of data memory can be interfaced to 8051.
4. If $\overline{EA}$ is connected to Vcc, the address of on-chip ROM is _____ to _____.
5. _____ number of bytes is allocated for SFR in on-chip data memory.

2.5 EXTERNAL MEMORY INTERFACING

The 8051 devices have only 256 bytes of on-chip data random access memory. In applications where large amount of data random access memory is required, the external data random access memory is interfaced with 8051. Address, data, $\overline{RD}$ (P3.7) and $\overline{WR}$ (P3.6) pins are used to interface data RAM. P3.7 and P3.6 are connected to $\overline{RD}$ and $\overline{WR}$ pins of data RAM as shown in Fig. 2.9.

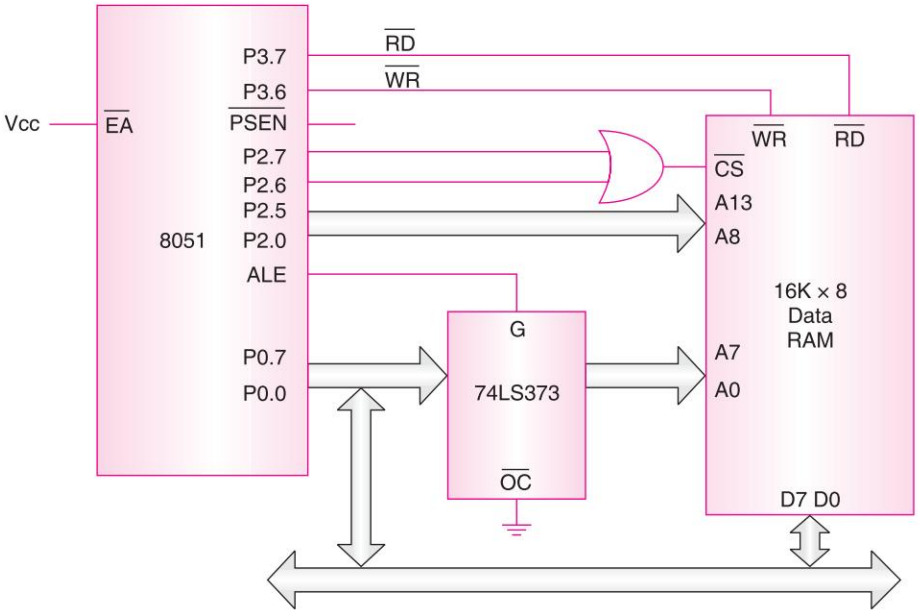


Figure 2.9 *Connection to external data RAM*

The memory chip requires 14 address lines (A13–A0) to decode 16384 × 8 registers. The remaining address lines—A15 and A14 pin—are connected through OR gate to $\overline{CS}$ pin of external data RAM. When the address lines A15 and A14 are low, then external data RAM is selected. The address of external data RAM is 0000H to 3FFFH. Since $\overline{EA}$ is connected to Vcc, internal program memory is selected for address 0000H to 0FFFH.

EXAMPLE 2.1

Design a microcontroller system using 8051 microcontroller, 4 Kbytes of ROM and 8 Kbytes of RAM. Interface the external memory such that the starting address of ROM is 1000H and RAM is C000H.

Figure 2.10 shows the interfacing of 4 Kbytes of ROM and 8 Kbytes of RAM. Then, $\overline{\text{PSEN}}$ is used as chip select pin for ROM containing program code, and $\overline{\text{RD}}$ (P3.7) is used as read control signal pin, and $\overline{\text{WR}}$ is used as write control signal for RAM as shown in Fig. 2.10.

Table 2.5 gives the memory address. ROM requires 12 address lines (A11–A0) to decode 4096 × 8 registers. The remaining address lines—A15, A14, A13, $\overline{\text{A12}}$ and $\overline{\text{PSEN}}$ pin —are connected through OR gate to $\overline{\text{CS}}$ and $\overline{\text{RD}}$ pin of ROM. When the address lines A15, A14, A13, $\overline{\text{PSEN}}$ are low and A12 is high, then ROM is selected. RAM requires 13 address lines (A12–A0) to decode 8192 × 8 registers. The remaining address lines— $\overline{\text{A15}}$, $\overline{\text{A14}}$ and A13—are connected through OR gate to $\overline{\text{CS}}$ pin of RAM. When the address lines A15, A14 are high and A13 is low, then RAM is selected. The address of ROM is 1000H to 1FFFH and RAM is C000H to DFFFH.

TABLE 2.5 *Memory address table*

Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ROM (4 K)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
							to									
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM (8 K)	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
							to									
	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1

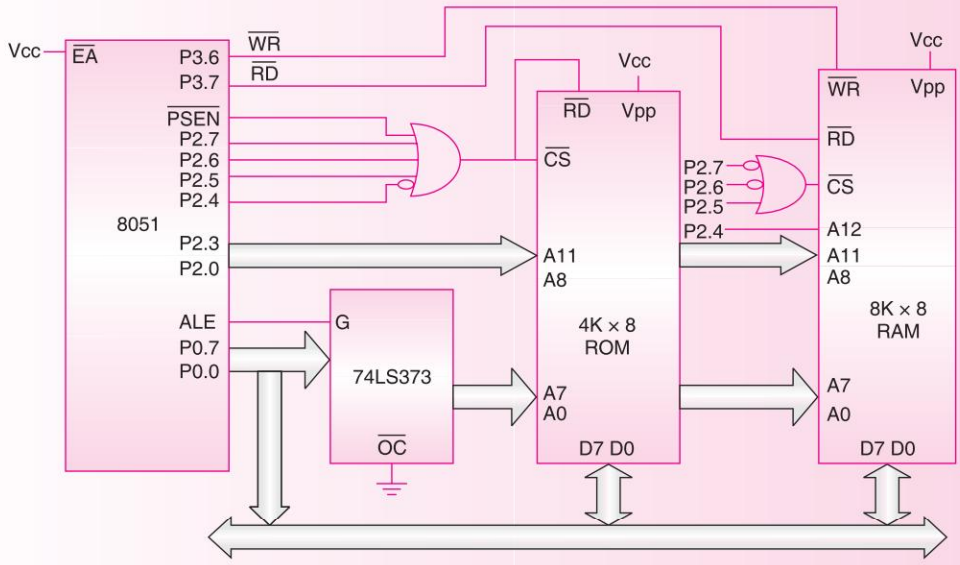


Figure 2.10 *Connection to external ROM and external RAM*

EXAMPLE 2.2

Design a microcontroller system using the 8051 microcontroller, 16 Kbytes of ROM and 32 Kbytes of RAM. Interface the memory such that the starting address of ROM is 0000H and RAM is 8000H.

Figure 2.11 shows the interfacing of 16 Kbytes of ROM and 32 Kbytes of data RAM. Then, $\overline{PSEN}$ is used as chip select pin for ROM containing program code, and $\overline{RD}$ (P3.7) is used as read control signal pin, and $\overline{WR}$ is used as write control signal for RAM as shown in Fig. 2.11.

Table 2.6 gives the memory address. ROM requires 14 address lines (A13–A0) to decode 16384 × 8 registers. The remaining address lines—A15, A14 and $\overline{PSEN}$ pin—are connected through OR gate to $\overline{CS}$ and $\overline{RD}$ pin of ROM. When the address lines A15, A14 and $\overline{PSEN}$ are low, then ROM is selected. The RAM requires 15 address lines (A14–A0) to decode 32768 × 8 registers. The address line A15 is connected through NOT gate to $\overline{CS}$ pin of data RAM. When the address line A15 is high, then data RAM is selected. The address of program ROM is 0000H to 3FFFH and data RAM is 8000H to FFFFH.

TABLE 2.6 *Memory address table*

Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ROM (16 K)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
							to									
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM (32 K)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
							to									
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

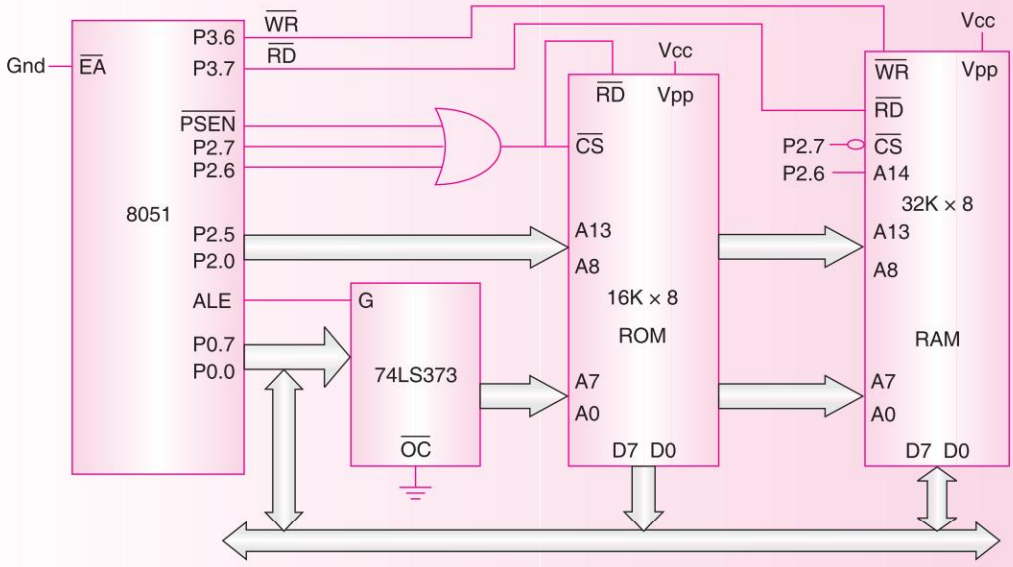


Figure 2.11 *Connection to external ROM and external RAM*

EXAMPLE 2.3

Design a microcontroller system using the 8051 microcontroller, 8 Kbytes Program ROM and 8 Kbytes of data ROM. Interface the memory such that the starting address of program ROM is 0000H and data ROM is E000H.

Figure 2.12 shows the interfacing of 8 Kbytes of program ROM and 8 Kbytes of data ROM. Then $\overline{PSEN}$ is used as chip select pin for ROM containing program code and $\overline{RD}$ ($P3.7$) is used as read control signal pin for ROM containing data as shown in Fig. 2.12. Table 2.7 shows the memory address. The memory chip requires 13 address lines (A12–A0) to decode 8192 × 8 registers.

TABLE 2.7 *Memory address table*

Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Program ROM (8 K)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
							to									
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
Data ROM (8 K)	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
							to									
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

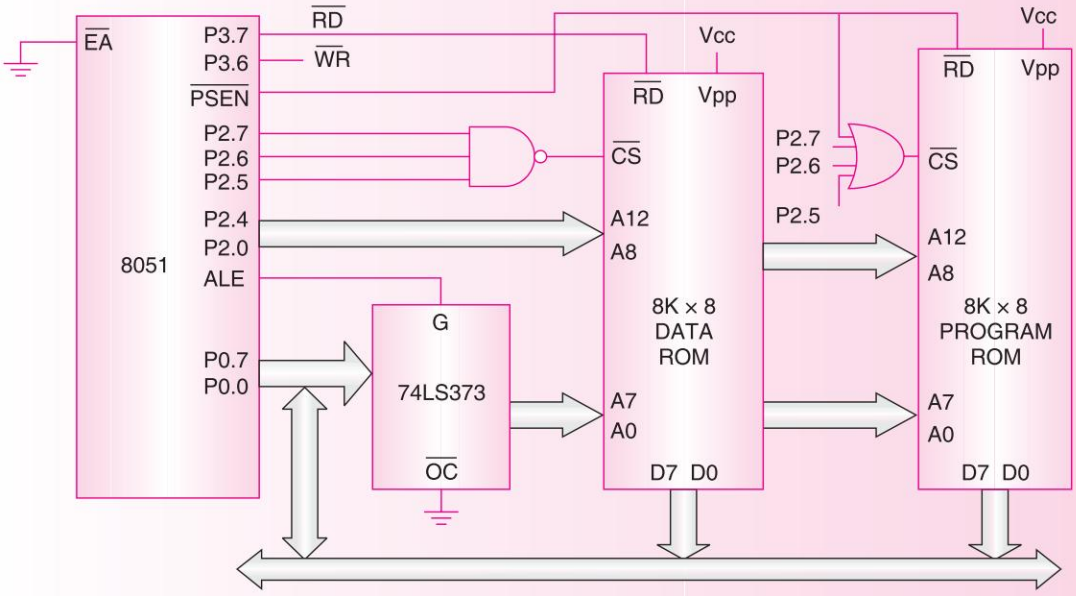


Figure 2.12 *Connection to external data ROM and external program ROM*

The remaining address lines—A15, A14, A13 and $\overline{\text{PSEN}}$ pin—are connected through OR gate to $\overline{\text{CS}}$ pin of program ROM. When the address lines A15, A14, A13 and $\overline{\text{PSEN}}$ are low, then program ROM is selected. The address lines A15, A14 and A13 are connected through NAND gate to $\overline{\text{CS}}$ pin of data ROM. When the address lines A15, A14 and A13 are high, then data ROM is selected. The address of program ROM is 0000H to 1FFFH and data ROM is E000H to FFFFH.

SECTION REVIEW

1. _____ number of address lines is used to decode 4 Kbytes of memory.
2. Mention the functions of $\overline{\text{EA}}$ pin in memory interfacing.
3. $\overline{\text{PSEN}}$ is used to select _____ memory.
4. _____maximum data memory and _____maximum program memory can be interfaced with the 8051.
5. 74LS373 is used to _____ address and data bus.

2.6 STACKS

A *stack* is a last-in-first-out (LIFO) memory. The stack allows all read and write operations to be carried out through one end and hence, the information that goes in last will come out first. The stack can be implemented in hardware or by software. Hardware stack is designed by using a set of high-speed registers in order to provide a fast response. The disadvantage of this approach is that the size of the stack is limited.

PIC microcontroller contains hardware stack and in the 8051 microcontroller, the stack is implemented by software. A software stack is a last-in-first-out data structure implemented by using a number of RAM locations. The software stack provides an unlimited stack size, that depends on the size of RAM but the disadvantage of this approach is slow response.

The address of the stack is contained in a register called the *stack pointer*. The stack pointer in 8051 is an 8 bit register. The 8051 has only 128 bytes of internal SRAM and can be used as stack space. Two instructions, namely PUSH and POP are usually used in stack operations. The PUSH operation is defined as writing to the top or bottom of the stack, whereas the POP operation means reading from the top or bottom of the stack. When stack is accessed from the bottom, the stack pointer is incremented in a push and decremented after a pop operation and when stack is accessed from the top, the stack pointer is decremented after a push and incremented after a pop operation. In the 8051 microcontroller, the stack is accessed from the bottom and the operation is as shown in Example 2.4 and 2.5.

EXAMPLE 2.4

Assume that the stack pointer points to memory location 3FH and the contents of the memory location 30H, 31H and 32H are 00,88 and FF respectively. Illustrate the stack contents after the execution of each of the following instructions.

```
PUSH 30H
PUSH 31H
PUSH 32H
```

The PUSH 30H instruction increments SP by 1 and then copies the data of memory location 30H to the location pointed by SP. Hence, 00 is copied to location 40H. If PUSH 31H instruction is executed,

SP is incremented by 1 and then content of location 31H is copied to the location 41H. Hence, 88 is copied to location 41H. If PUSH 32H instruction is executed, SP is incremented by 1 and then content of location 32H is copied to the location 42H. Hence, FF is copied to location 42H as shown in Fig. 2.13.

Figure 2.13 *Contents of the stack after 3 PUSH instructions*

EXAMPLE 2.5

Assume that the stack pointer points to the memory location 52H and the contents of the memory location 40H and 41H are 18 and 21 respectively Illustrate the operation of the stack after the execution of the following instructions.

POP 41H
POP 40H

POP 41H instruction copies the data pointed from the location pointed by SP to the memory location 41H and then decrements SP by 1. Hence, 88 is copied to location 41H. If POP 40H instruction is executed, then data pointed from the location pointed by SP is copied to the memory location 40H and then SP is decremented by 1. Hence, 00 is copied to location 40H as shown in Fig. 2.14.

Figure 2.14 *Contents of memory and stack pointer after 2 POP instructions*

SECTION REVIEW

1. What is a stack?
2. Name two instructions used in a stack operation.
3. In the 8051, the stack space is _____ bytes.
4. The 8051 contains a software stack. True/False?
5. In the 8051, the stack space is defined in _____.

2.7 8052 MICROCONTROLLER

The 8052 is an enhanced version of 8051 microcontroller. The pin diagram and instruction set of 8051 and 8052 are practically the same. The 8052 differs from 8051 by internal memory size and number of on-chip timers. There are two timers/counters in 8051 as discussed in Section 2.2, whereas the number of timers/counters in 8052 is three. As discussed in Section 2.2.2, the 8051 has 256 bytes of on-chip data RAM. Also, the data memory is divided into two groups of memory size 128 bytes each. The first 128 bytes of on-chip data RAM with address 00H–7FH are used as register banks, bit addressable RAM and scratch pad registers, and the remaining 128 bytes of on-chip RAM with address 80H–FFH are used as special function registers and are accessed by using direct addressing modes. The 8052 has 8 Kbytes of on-chip program memory. The program memory can be extended upto 64 Kbytes with external memory as shown in Fig. 2.15.

The 8052 has two on-chip data RAM. In addition to 256 bytes of on-chip data RAM of 8051, the 8052 has another 128 bytes of data RAM with address 80H–FFH as shown in Fig. 2.16. The address of second data RAM has the same address assigned to special function registers.

Direct addressing mode is used to access special function registers. To differentiate parallel address space, indirect addressing mode is used to access second data RAM. Registers R0 and R1 are used as pointers and instructions MOV A, @R0 and MOV @R0, A or MOV A, @R1 and MOV @R1, A are used to access second data RAM. Timer 2 of 8052 operates either as a timer or as an event counter like timer/counter 0 and timer/counter 1 and also operates in capture, auto load and as baud rate generator.

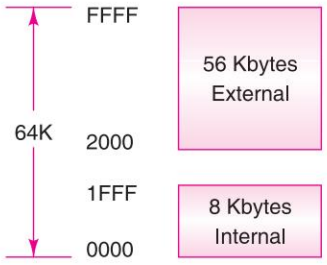


Figure 2.15 *Program memory organisation of 8052*

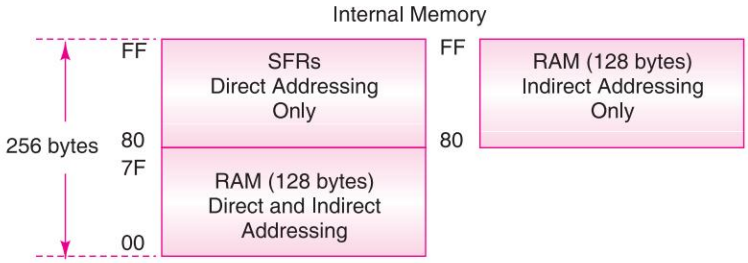


Figure 2.16 *Data memory organisation of 8052*

SECTION REVIEW

1. The 8052 has _____ bytes of internal data RAM.
2. Name the addressing modes used in internal data memory of the 8052.
3. The 8052 contains _____ number of timers.
4. The 8052 has _____ bytes of on-chip data memory and _____ bytes of on-chip program memory.

CHAPTER SUMMARY

- This chapter describes the salient features, architecture and pin details of the 8051 microcontroller. It provides an in-depth elucidation of the following.
- Various registers, 'A', 'B', program status word, and four register banks R0–R7.
- Various flags in program status word.
- SFR related to timers/counters, serial and parallel I/O ports.
- The internal organisation of the memory and interfacing of external data and program memory with 8051 (discussed with examples).
- Manipulation of the stack via PUSH and POP instructions.
- Additional features of 8052 microcontroller.

EXERCISES

MULTIPLE CHOICE QUESTIONS

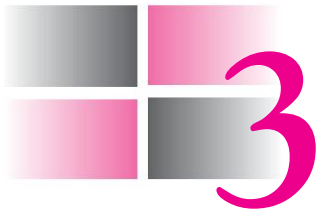
1. _____ special function register is bit addressable.
   (a) Timer/Counter control (b) Stack pointer
   (c) Serial data buffer (d) Data Pointer Low Byte (DPL)
2. _____ of internal RAM are bit addressable memory.
   (a) 128 bit (b) 64 bit
   (c) 32 bit (d) 256 bit
3. The 8051 contains.
   (a) 4 banks of 8 registers (b) 2 banks of 16 registers
   (c) 8 banks of 4 registers (d) 4 banks of 4 registers
4. _____ is used to detect errors in signed arithmetic operations.
   (a) AC flag (b) OV flag
   (c) CY flag (d) P flag
5. Stack pointer in the 8051 is.
   (a) 8 bit register (b) 4 bit register
   (c) 16 bit register (d) None of the above
6. The 8051 contains_____.
   (a) Four 8 bit parallel ports (b) Three 8 bit parallel ports
   (c) Three 16 bit parallel ports (d) None of the above
7. _____ special function register of 8051 is not bit addressable.
   (a) Accumulator (b) Program status word
   (c) Serial data buffer (d) Port 0

8. The alternate function of port 3.2 is.
   (a) Serial input port (b) External interrupt zero
   (c) Data memory write strobe (d) External interrupt 1
9. The 8051 operates with _____maximum clock frequency.
   (a) 12 MHz (b) 3 MHz
   (c) 16 MHz (d) 20 MHz
10. The 8051 operates with _____minimum clock frequency.
   (a) 1 MHz (b) 3 MHz
   (c) 2 MHz (d) 4 MHz
11. _____ pin is used to demultiplex address/data bus.
   (a) $\overline{EA}$ (b) ALE
   (c) $\overline{PSEN}$ (d) None of the above
12. 8051 devices have _____ on-chip program memory.
   (a) 1 Kbytes (b) 2 Kbytes
   (c) 3 Kbytes (d) 4 Kbytes
13. 8051 devices have _____ of on-chip data RAM.
   (a) 256 bytes (b) 1 Kbyte
   (c) 128 Kbytes (d) None of the above
14. _____ pin is the external data memory write strobe in 8051.
   (a) P3.4 (b) P3.5
   (c) P3.6 (d) P3.7
15. If $\overline{EA}$ = Vcc in 8051, then
   (a) Only external program ROM is selected
   (b) Only internal program ROM is selected
   (c) Both internal and external program ROM is selected
   (d) None of the above
16. _____ pin is used to select external program ROM.
   (a) $\overline{PSEN}$ (b) $\overline{EA}$
   (c) ALE (d) RESET
17. _____ number of minimum address lines required to select registers in 4 Kbytes of ROM.
   (a) 10 (b) 12
   (c) 14 (d) 16
18. In 8051, a stack is implemented in
   (a) Internal RAM (b) Internal ROM
   (c) External RAM (d) External ROM
19. The 8051 has maximum _____ of stack.
   (a) 128 bytes (b) 256 bytes
   (c) 64 bytes (d) 16 bytes

20. The stack in 8051 is _____.
    (a) Last-in-first-out (b) First-in-first-out
    (c) Last-in-last-out (d) None of the above

REVIEW QUESTIONS

2.1 Enlist the salient features of the 8051.
2.2 Describe the internal architecture of the 8051 microcontroller with a block schematic diagram.
2.3 Explain why the data pointer (DPTR) is 16 bit wide and the stack pointer is 8 bit wide in 8051? Justify.
2.4 Enlist the various flags in the PSW register. Discuss the function of each flag.
2.5 What is the difference between over flow and carry flag? Explain with an example.
2.6 List the special function register associated with
    (a) Interrupts (b) I/O ports (c) Timers/Counters
2.7 What are the functions of the following 8051 pins?
    (a) ALE (b) $\overline{\text{PSEN}}$ (c) $\overline{\text{EA}}$ (d) RST
2.8 Explain the oscillator circuit and timing of the 8051 microcontroller.
2.9 Explain the alternate functions of port 0, port 2 and port 3.
2.10 Explain the function of registers A and B in multiplication and division operations.
2.11 Explain the concept of memory banks.
2.12 Explain the structure of internal RAM of 8051.
2.13 Is it possible to address 8051 individual bits? What are the addresses of the bit addressable locations?
2.14 Explain what are the advantages in having a provision for both internal and external program memory.
2.15 Explain how bit addressing is distinguished from byte addressing in the 8051 microcontroller.
2.16 Explain how a stack is implemented in the 8051.
2.17 What is a stack? Explain the operation of a stack with an example.
2.18 Explain the functions of PUSH and POP with examples.
2.19 Interface RAM and ROM with 8051. Explain how to access them.
2.20 Interface 16 Kbytes of program ROM and 32 Kbytes of data ROM to the 8051 microcontroller, such that the starting address of program ROM is 4000H and data ROM is 0000H.
2.21 Interface 16 Kbytes of ROM and 8 Kbytes of RAM to the 8051 microcontroller such that the starting address of ROM is C000H and RAM is 8000H.
2.22 Interface 8 Kbytes of ROM and 4 Kbytes of RAM to 8051 microcontroller such that the starting address of ROM is 0000H and RAM is C000H.
2.23 Compare the features of the 8051 and 8052.
2.24 Explain the structure of internal RAM of the 8052 microcontroller.
2.25 List bit addressable special function registers of the 8051 microcontroller.



8051 ADDRESSING MODES AND INSTRUCTION SET

Learning Objectives

After you have completed this chapter, you should be able to

- Explain the instruction syntax and data types of the 8051
- Define a subroutine and Explain its uses
- Explain the addressing modes of the 8051
- Explain the instruction timings of the 8051
- Explain the instruction set of the 8051

3.1 INSTRUCTION SYNTAX

In assembly language, all operations and addresses can be identified by symbols. This frees the programmer from memorising or looking up the machine code for instructions and keeping track of the addresses of the entire data and instructions. 8051 instructions are divided into four fields as follows.

LABEL: OPCODE OPERAND; COMMENT

Label The label is the symbolic address for the instruction. As the program is assembled, the label will be given the value of the address in which the instruction is stored. This facilitates referencing of the instruction at any point in the given program. Of course, not all instructions will have labels. It is not necessary to define a symbol for the address of an instruction, unless that address is needed by a branch statement elsewhere in the program. For instance in Example 3.1, only one instruction is referred by a branch statement. A label can be any combination of upto 8 letters (A–Z), numbers (0–9) and period (.).

EXAMPLE 3.1

8051 program

```
MOV R5,# 05H    ; Load counter R5 = 05H
MOV A, 40H      ; Copy data from RAM location 40H to register A
MOV R0, #30H    ; Copy immediate data 30H to register R0
LOOP: ADD A, R0 ; Add contents of register R0 with contents of
                  register A
DJNZ R5, LOOP   ; Repeat addition until R5 is zero
END
```

Opcode The opcode field contains a symbolic representation of the operation. The operation tells the assembler what action the statement has to perform. The 8051 assembler converts the opcode into a unique machine language (binary code) that can be acted on by the 8051 internal circuitry. For instance, in Example 3.1, the mnemonics MOV and ADD are the opcodes. MOV will copy data from one location to register, or immediate data to register. ADD will add contents of one register to another register.

Operand The opcode specifies what action to perform, whereas the operand indicates where to perform the action. The operand field contains the address of the operand or the operand. For instance in Example 3.1, MOV A, 40H; the operand field contains two addresses. MOV will copy data from one location (source) to another location (destination). In MOV R0, #30H; the operand field contains operand 30H and it is moved to register R0.

Comment To improve program clarity, a programmer uses comments throughout the program. A comment always begins with a semicolon (;) and wherever we code it, the assembler assumes that all characters to its right are comments. A comment may contain any printable character, including a blank. We can insert a comment on a line all by itself or following an instruction on the same line. A comment appears only on a listing of an assembled program and generates no machine code.

SECTION REVIEW

1. Write the format of 8051 instructions.
2. _____special character is used to begin comments.

3.2 DATA TYPES

The 8051 microcontroller supports only 8 bit data. It supports both signed and unsigned 8 bit numbers. Unsigned numbers are defined as data in which all the bits are used to represent the data. For 8 bit, the data can be 00 to FFH or 00 to 255 in decimal. In signed number representation, the most significant bit identifies the number as positive or negative, and the remaining bits are used to represent magnitude. If the most significant bit is zero, the number is positive, or if it is one, then the number is negative. In negative numbers, the magnitude is represented in two's complement form as shown in Table 3.1. For 8 bit, the data can be from +127 to –128.

TABLE 3.1 *8051 data types*

Decimal	Hex	Binary
– 128	80	10000000
– 127	81	10000001
– 1	FF	11111111
0	00	00000000
+ 1	01	00000001
+ 127	7F	01111111

3.3 SUBROUTINES

Good program design is based on the concept of modularity—the partitioning of a large program into subroutines. A subroutine is a sequence of instructions stored in the memory at a specified address for performing repeatedly needed tasks. Subroutines are usually handled by special instructions, CALL and RET. The CALL instruction is of the form CALL address. The subroutine processing is as shown in Fig. 3.1.

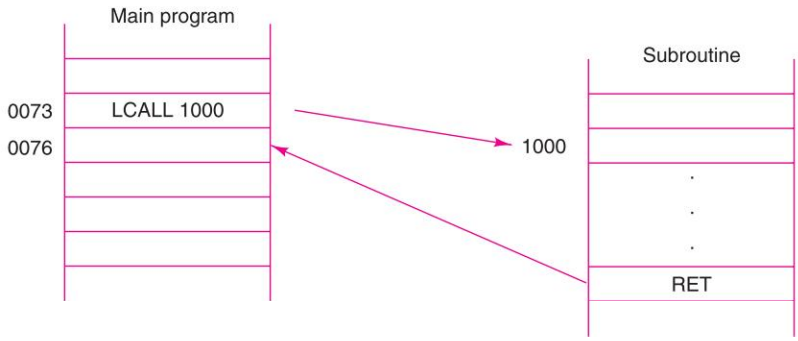


Figure 3.1 *Subroutine processing*

The address refers to the address of the subroutine. When CALL instruction is executed, the contents of the program counter are saved in the stack and the program counter is loaded with the address, which is a part of CALL instruction. The RET instruction is usually the last instruction in the subroutine. When this instruction is executed, the return address previously saved in the stack is retrieved and is loaded into the program counter. Thus, the control is transferred to the calling program.

SECTION REVIEW

1. List types of data supported by the 8051.
2. In signed numbers, _____ bit represents the sign.
3. In signed numbers, the magnitude of the negative numbers is represented in _____.
4. Name the instructions used in a subroutine.
5. When CALL instruction is executed, then the contents of the program counter are saved in _____.

3.4 ADDRESSING MODES

A microcontroller provides, for the convenience of the programmer, various methods for accessing data needed in the execution of an instruction. The various methods of accessing data are called *addressing modes*. The 8051 addressing modes can be classified into the following categories

- Immediate addressing
- Register addressing
- Direct addressing
- Indirect addressing
- Relative addressing
- Absolute addressing
- Long addressing
- Indexed addressing
- Bit inherent addressing
- Bit direct addressing

3.4.1 IMMEDIATE ADDRESSING

Immediate addressing means that the data is provided as part of the instruction (which immediately follows the instruction opcode).

EXAMPLE 3.2

Instruction MOV A, #99d moves the value 99 into the A (since we used 99d, data is in decimal) as shown in Fig. 3.2. The # symbol tells the assembler the immediate addressing mode is to be used.

Figure 3.2 *Immediate addressing*

3.4.2 REGISTER ADDRESSING

Register addressing mode involves the use of registers to hold the data to be manipulated. The lowest 32 bytes of the 8051 internal RAM are organised as four banks of eight registers. Only one bank is active at a time. Using names, R0 to R7 can access any active register. One of the eight general registers (R0 to R7) can be specified as the instruction operand. The assembly language documentation refers to a register generically as Rn.

EXAMPLE 3.3

Instruction MOV A, R5 copies contents of register R5 to A.

Here, the content of R5 is copied to the A as shown in Fig. 3.3. The advantage of register addressing is that the instruction tends to be short and is a single–byte instruction.

Figure 3.3 *Register addressing*

3.4.3 DIRECT ADDRESSING

Direct addressing mode is provided to allow us access to internal data memory, including Special Function Register (SFR). In direct addressing, an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode, data is obtained directly from the memory.

EXAMPLE 3.4

Consider the instruction: MOV A, 47H.

The instruction reads the data from internal RAM address 47H and copies this into the A as shown in Fig. 3.4.

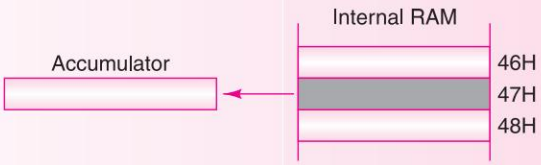


Figure 3.4 *Direct addressing*

3.4.4 INDIRECT ADDRESSING

Indirect addressing provides a powerful addressing capability, which needs to be appreciated. The indirect addressing mode uses a register to hold the actual address that will be used in data movement. Registers R0,

R1, and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R0 and R1 can hold 8 bit address and DPTR can hold 16 bit address.

EXAMPLE 3.5

The instruction which uses indirect addressing, is MOV A, @Ro.

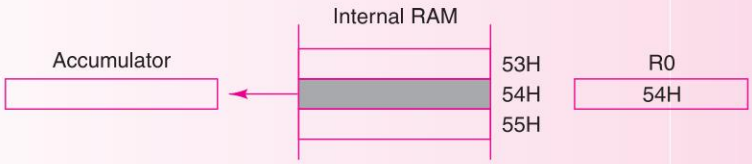


Figure 3.5 *Indirect addressing*

The @ symbol in the instruction indicates indirect addressing mode. Ro contains the address of the internal RAM location (54H). It copies the contents of memory location pointed by Ro into accumulator A.

MOVX A, @DPTR

It copies the contents of external data memory location pointed by DPTR into accumulator A.

3.4.5 INDEXED ADDRESSING

In indexed addressing, a separate register—either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of the offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are easily implemented with the help of index addressing.

EXAMPLE 3.6

Consider the instruction: MOVC A, @A+DPTR

MOVC is a move instruction, which copies data from the external code memory space. In this example, adding the content of the DPTR register to the contents of accumulator forms the address of the operand. Here, the DPTR value is referred as the base address and the accumulator value is referred as the index address. This instruction copies the contents of memory location pointed by the sum of the accumulator A and the data pointer DPTR into accumulator A.

MOVC A, @A+PC

This instruction copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.

The above–discussed addressing modes are illustrated as shown in Fig. 3.6

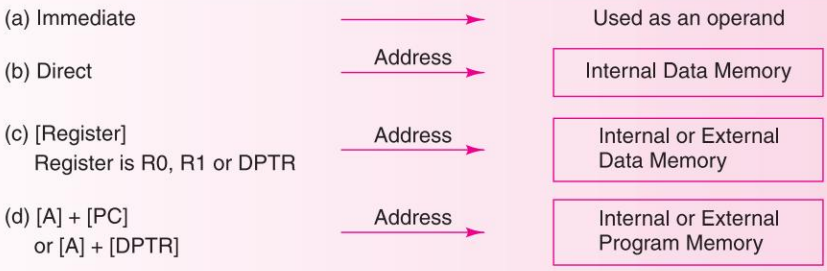


Figure 3.6 *Addressing modes (a) Immediate (b) Direct (c) Indirect (d) Indexed*

3.4.6 RELATIVE ADDRESSING

Relative addressing is used only with conditional jump instructions. The relative address, often referred to as an offset, is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to –128 locations. The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly.

EXAMPLE 3.7

Consider the example SJMP X.

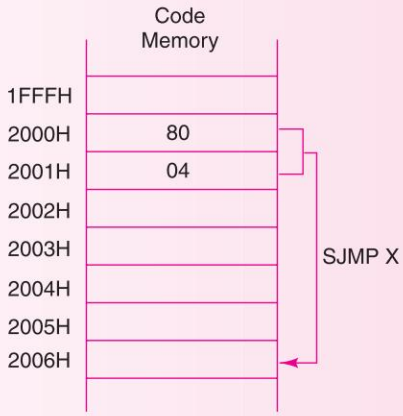


Figure 3.7 *Relative addressing mode*

PC is set to next instruction address: 2002H, when SJMP begins execution. The target address is then the sum of the PC + relative offset needed to reach X. If X is 4, then PC is set to 2002H + 4H = 2006H as shown in Fig. 3.7. The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

3.4.7 ABSOLUTE ADDRESSING

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions. These are 2 bytes instructions. The absolute addressing mode specifies the lowest 11 bit of the memory address as part of the instruction. The upper 5 bit of the destination address are the upper 5 bit of the current program counter. Hence, absolute addressing allows branching only within the current 2 Kbyte page of the program memory.

EXAMPLE 3.8

Consider the instruction

```
AJMP loop1
```

The instruction with the label loop1 will be executed after the current instruction.

```
ACALL loop1
```

Calls the subroutine that is started at the label loop1

3.4.8 LONG ADDRESSING

The long addressing mode within the 8051 is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space (2^{16} = 64K).

EXAMPLE 3.9

Consider the instruction

```
LJMP 5000H
```

16 bit branch address is specified in the instruction. The instruction with the address 5000H will be executed after the current instruction.

```
LCALL 6000H
```

16 bit subroutine address is specified in the instruction. It calls the subroutine, which starts at the address 6000H.

3.4.9 BIT INHERENT ADDRESSING

In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.

EXAMPLE 3.10

The following instruction illustrates bit inherent addressing.

```
CLR C     ; Clears the carry flag to 0
```

3.4.10 BIT DIRECT ADDRESSING

The RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.

EXAMPLE 3.11

The following instructions illustrate the bit direct addressing.

```
CLR 07H    ; Clears the bit 7 of 20H RAM space
SETB 07H   ; Sets the bit 7 of 20H RAM space.
```

SECTION REVIEW

1. List addressing modes of the 8051 microcontroller.
2. Immediate addressing is identified by _____ symbol in the assembler.
3. MOV A, R4 is an example for _____ addressing mode.
4. List the types of addressing modes in memory operations.
5. Direct addressing mode is used to access _____ memory.
6. List the registers used as data pointers in indirect addressing.
7. Name the instructions used to access internal data memory using indirect addressing mode.
8. In indexed addressing, _____ register is used to hold the base address and _____ register is used to hold the offset address.
9. MOV C instruction moves data from _____ memory.
10. In relative addressing mode, offset address is _____ .
11. SJMP 25 is an example of _____ addressing mode.
12. Differentiate SJMP and AJMP instructions.
13. CLRC instruction is an example for bit indirect addressing. True/False?
14. MOVXA@DPTR is an example for indirect addressing. True/False?
15. In bit direct addressing, bit values are _____ to _____ .
16. SETB 07H is an example for _____ addressing mode.

3.5 INSTRUCTION TIMINGS

The 8051 internal operations and external read/write operations are controlled by the oscillator clock input signal. T-state, Machine cycle and Instruction cycle are terms used in instruction timings. Let us define these terms.

T-state T-state is defined as one subdivision of the operation performed in one clock period. The terms 'T-state' and 'clock period' are often used synonymously.

Machine cycle Machine cycle in 8051 is defined as 12 oscillator periods. According to the Intel literature, a machine cycle consists of six states and each state lasts for two oscillator periods. The 8051, take one to four machine cycles to execute an instruction.

Instruction cycle Instruction cycle is defined as the time required for completing the execution of an instruction. The 8051 instruction cycle consists of one to four machine cycles.

EXAMPLE 3.12

If 8051 microcontroller is operated with 12 MHz oscillator, find the execution time for the following four instructions.

(a) ADD A, 45H
(b) SUBB A, #55H
(c) MOV DPTR, #2000H
(d) MUL AB

Since the oscillator frequency is 12 MHz, the clock period is

Clock period = 1/12 MHz = 0.08333 µs.

Time for 1 machine cycle = 0.08333 µs × 12 = 1 µs.

Execution timings are tabulated in Table 3.2

TABLE 3.2 *Instruction timing*

Instruction	Execution time in machine cycle	Execution time in µsec
ADD A, 45H	1 machine cycle	1 µs
SUBB A, #55H	2 machine cycles	2 µs
MOV DPTR, #2000H	2 machine cycles	2 µs
MUL AB	4 machine cycles	4 µs

SECTION REVIEW

1. If the 8051 operates with 6 MHz crystal, then the execution time for ADD A, # 45H instruction is _____.
2. If the 8051 operates with 3MHz oscillator, then the clock period is _____.

3.6 8051 INSTRUCTIONS

8051 instructions consist of 1 byte of opcode and 0 to 2 bytes of operands. The instructions use 8 bit register A, B, R0, R1, R2, R3, R4, R5, R6, R7 and also 16 bit registers, DPTR (data pointer) and PC (program counter). The instructions of 8051 can be broadly classified under the following headings:

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Bit manipulation instructions

The following notations are used in the description of the instruction.

Rn means, any of the eight registers (R0 to R7) in the selected banks.
Ri means either of the pointing registers (R0 or R1) in the selected banks.
M means internal data memory.
E.D.M. means external data memory.
P.M. means internal or external program memory.

3.6.1 DATA TRANSFER INSTRUCTIONS

In this group, the instructions perform data transfer operations of the following types.

1. Move the contents of a register A to Rn
2. Move the contents of a register Rn to A
3. Move an immediate 8 bit data to register A or to Rn or to a memory location
4. Move the contents of a memory location to A or A to a memory location using direct and indirect addressing
5. Move the contents of a memory location to Rn or Rn to a memory location using direct addressing
6. Move the contents of memory location to another memory location using direct and indirect addressing
7. Move the contents of an external memory to A or A to an external memory
8. Move the contents of program memory to A
9. Push and Pop instructions
10. Exchange instructions

Note

In this group of instructions, none of the flags of status register are affected.

Move the contents of a register Rn to A: MOV A, Rn This instruction copies the contents of register Rn to A. It is a 1 byte instruction.

EXAMPLE 3.13

```
                    MOV A, R2        [A] ← [R2]
                Before execution  After execution
[A]                   25                45
[R2]                  45                45
```

Move the contents of a register A to Rn: MOV Rn, A This instruction copies the contents of A to Rn. It is a 1 byte instruction.

EXAMPLE 3.14

```
                          MOV R3, A        [R3] ← [A]
                       Before execution After execution
              [A]            25               25
              [R3]           45               25
```

Move an immediate 8 bit data to register A or to Rn or to a memory location. This type of instruction loads the immediate data to A or to Rn or to the memory by using direct and indirect addressing.

```
MOV A, #data
```

EXAMPLE 3.15

```
                          MOV A, #23       [A] ← 23
                       Before execution After execution
              [A]            F5               23
```

```
MOV Rn, #data
```

EXAMPLE 3.16

```
                          MOV R2, #23      [R2] ← 23
                       Before execution After execution
              [R2]           F3               23
```

```
MOV direct, #data
```

EXAMPLE 3.17

```
                          MOV 30, #23     M [30] ← 23
                       Before execution After execution
            M [30]           F3               23
```

```
MOV @Ri, #data
```

EXAMPLE 3.18

```
                          MOV @R1, #23    M [[R1]] ← 23
                       Before execution After execution
              [R1]           34               34
               M [34]        F3               23
```

```
MOV DPTR, #16bitdata
```

EXAMPLE 3.19

```
                    MOV DPTR, #2300   [DPTR] ← 2300
                  Before execution After execution
[DPTR]                3422              2300
[DPL]                   22                00
[DPH]                   34                23
```

Move the contents of a memory location to A or A to a memory location using direct and indirect addressing

Direct addressing:

```
MOV A, direct
```

EXAMPLE 3.20

```
                     MOV A, 40       A ← M [40]
                  Before execution After execution
[A]                     23                55
M [40]                  55                55
```

```
MOV direct, A
```

EXAMPLE 3.21

```
                     MOV 45,A       M [45] ← [A]
                  Before execution After execution
[A]                     23                23
M [45]                  55                23
```

Indirect addressing:

```
MOV A, @Ri
```

EXAMPLE 3.22

```
                    MOV A,@R1     [A] ← M [[R1]]
                  Before execution After execution
[A]                     23                55
[R1]                    42                42
M[42]                   55                55
```

```
MOV @Ri, A
```

EXAMPLE 3.23

```
                    MOV @R1, A      M [[R1]] ← [A]
                 Before execution  After execution
[A]                    23                23
[R1]                   40                40
M[40]                  55                23
```

Move the contents of a memory location to Rn or Rn to a memory location using direct addressing

```
MOV Rn, direct
MOV direct, Rn
```

EXAMPLE 3.24

```
                    MOV R2, 50      R2 ← M [50]
                 Before execution  After execution
[R2]                   23                55
M [50]                 55                55
```

EXAMPLE 3.25

```
                    MOV 60,R3       M [60] ← R3
                 Before execution  After execution
[R3]                   23                23
M [60]                 55                23
```

Move the contents of a memory location to another memory location using direct and indirect addressing

```
MOV direct1, direct2
```

EXAMPLE 3.26

```
                    MOV 55,67     M [55] ← M [67]
                 Before execution  After execution
M [55]                 23                55
M [67]                 55                55
```

```
MOV direct, @Ri
```

EXAMPLE 3.27

```
                        MOV 57, @R0   M [57] ← M [[R0]]
                      Before execution After execution
[R0]                        30               30
M [30]                      33               33
M [57]                      55               33
```

Move the contents of an external data memory location pointed by Ri or DPTR to A or A to an external data memory pointed by Ri or DPTR.

```
MOVX A, @Ri
```

EXAMPLE 3.28

```
                          MOVX A, @R1       A ← E.D.M[[R1]]
                        Before execution    After execution
[A]                           23                  55
[R1]                          10                  10
E.D.M. [10]                   55                  55
```

```
MOVX @Ri,A
```

EXAMPLE 3.29

```
                          MOVX @R1, A     E.D. M [[R1]] ← [A]
                        Before execution    After execution
[A]                           23                  23
[R1]                          20                  20
E.D.M [20]                    55                  23
```

```
MOVX A, @DPTR
```

EXAMPLE 3.30

```
                          MOVX A, @DPTR     A ← E.D.M[[DPTR]]
                        Before execution    After execution
[A]                           23                  55
[DPTR]                       1000                1000
E.D.M. [1000]                 55                  55
```

```
MOVX @DPTR, A
```

EXAMPLE 3.31

```
                              MOVX @DPTR,A      E.D.M[[DPTR]] ← [A]
                            Before execution     After execution
[A ]                              23                   23
[DPTR]                          1000                 1000
EDM[1000]                         55                   23
```

Move code byte to accumulator: Copy the contents of program memory location pointed by A and DPTR or A and PC to A

```
MOVC A, @A+PC
```

EXAMPLE 3.32

```
                              MOVC A, @A+PC    [A] ← P.M [[A]+[PC]]
                            Before execution     After execution
[A]                               23                   55
[PC]                            1000                 1000
P.M.[1023]                        55                   55
```

```
MOVC A,@ A+DPTR
```

EXAMPLE 3.33

```
                              MOVC A, @A+DPTR   A ← P.M[[A]+[DPTR]]
                            Before execution     After execution
[A]                               23                   55
[DPTR]                          1000                 1000
P.M. [1023]                       55                   55
```

The above data transfer instructions and addressing is illustrated in Fig. 3.8.

Push and Pop instructions *Push instruction* increments SP by 1 and then copies the data at direct address to the location pointed by SP.

PUSH direct

Operation: [SP] ← [SP]+1

[[SP]] ← direct

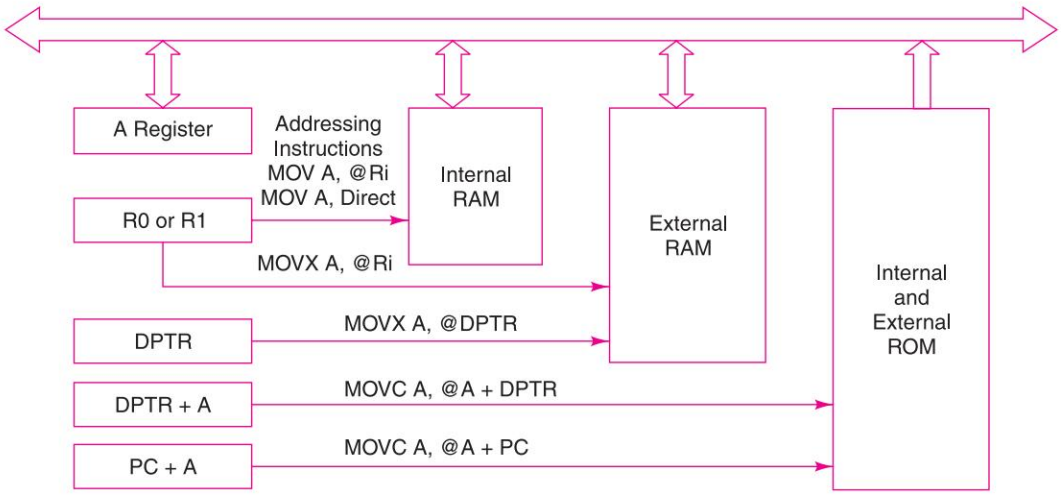


Figure 3.8 *Addressing using MOV, MOVX and MOVC*

EXAMPLE 3.34

```
                    PUSH 20H
                Before execution After execution
 M [20]              23              23
[SP]                 00              01
M [01]               99              23
M [00]               55              55
```

Pop instruction copies the data at the location pointed by SP to the direct address and then decrements SP by 1.

POP direct

```
Operation: direct ← [[SP]]
           [SP] ← [SP]-1
```

EXAMPLE 3.35

```
                    POP 20H
                Before execution   After execution
M [20]              23                 99
[SP]                01                 00
M [01]              99                 99
M [00]              55                 55
```

Exchange instructions

Exchange A with byte variable XCH loads the A with the contents of the indicated variable, at the same time, it writes the original A contents to the indicated variable.

```
XCH A, Rn
```

EXAMPLE 3.36

```
                    XCH A, R2          [A] ↔ [R2]
                 Before execution   After execution
[A]                    23                 55
[R2]                   55                 23
```

XCH A, direct

EXAMPLE 3.37

```
                    XCH A, 35          [A] ↔ M [35]
                 Before execution   After execution
[A]                    23                 55
M[35]                  55                 23
```

XCH A,@Ri

EXAMPLE 3.38

```
                    XCH A, @R0         [A] ↔ M[[R0]]
                 Before execution   After execution
[A]                    23                 33
[R0]                   55                 55
M [55]                 33                 23
```

Exchange digit XCHD exchanges the low–order nibble of the A bits (bits 3–0), with that of the internal RAM location indirectly addressed by the specified register.

XCHD A, @Ri

EXAMPLE 3.39

```
                    XCHD A, @R0        [A] ↔ M[[R0]]
                 Before execution   After execution
[A]                    29                 23
[R0]                   55                 55
M [55]                 33                 39
```

We will study following examples to become familiar with **data transfer group instructions**.

EXAMPLE 3.40

Write 8051 instructions to load data 25H to A and to register R2.

ALGORITHM

```
Step 1: Load data 25H to A
Step 2: Load data 25H to R2
```

Instructions are as follows

```
MOV A, #25H          ; Load data 25H to A
MOV R2, #25H         ; Load data 25H to R2
```

EXAMPLE 3.41

Write 8051 instructions to load data 30H to A and copy the contents of *A* to register R3.

ALGORITHM

```
Step 1: Load immediate data 30H to A
Step 2: Copy the contents A to R3
```

Instructions are as follows

```
MOV A, #30H   ; Load data 30H to A
MOV R3, A     ; Copy contents of A to R3
```

EXAMPLE 3.42

Write 8051 instructions to copy the contents of external data memory location pointed by DPTR register to internal data memory location pointed by Ro.

ALGORITHM

```
Step 1: Load 16 bit external memory address to DPTR
Step 2: Load 8 bit internal memory address to R0
Step 3: Copy contents of external memory to A
Step 4: Copy contents of A to internal memory
```

Instructions are as follows

```
MOV DPTR, #3000H ; Load 16 bit external memory address to DPTR
MOV R0, #25H     ; Load 8 bit internal memory address to R0
MOVX A, @DPTR    ; Copy contents of external memory to A
MOV @R0,A        ; Copy contents of A to internal memory
```

EXAMPLE 3.43

Write 8051 instructions to copy the contents of external program memory location 3025H to internal data memory location pointed by Ro.

ALGORITHM

```
Step 1: Load 16 bit address (3000H) to DPTR
Step 2: Load 8 bit address (25H) to A
Step 3: Internal memory address to R0
Step 4: Copy contents of external data memory to A
Step 5: Copy contents of A to internal memory
```

```
Instructions are as follows
  MOV DPTR, #3000H; Load address 3000H to DPTR
  MOV A, #25H     ; Load address 25H to A
  MOV R0, #25H    ; Load 8 bit internal memory address to R0
  MOVC A, @A+DPTR ; Copy contents of external program memory to A
  MOV @R0,A       ; Copy contents of A to internal memory
```

EXAMPLE 3.44

Write 8051 instructions to exchange contents of register R2 with the contents of A.

ALGORITHM

```
Step 1: Load data 35H to A
Step 2: Load data 57H to R2
Step 3: Exchange contents of A with R2
```

```
Instructions are as follows
  MOV A, #35H   ; Load data 35H to A
  MOV R2, #57H  ; Load data 57H to R2
  XCH A,R2      ; Exchange contents of A with R2
```

3.6.2 ARITHMETIC INSTRUCTIONS

The 8051 has powerful instructions in the arithmetic group compared to other microcontrollers. It can perform addition, subtraction, multiplication and division operations on 8 bit numbers.

ADD Group of Instructions

In this group, we have instructions to

1. Add the contents of A with immediate data with or without carry.
2. Add the contents of A with register Rn with or without carry.
3. Add the contents of A with contents of memory with or without carry using direct and indirect addressing.

Note

CY, AC, and OV flags are affected in this group.

Add the contents of accumulator with immediate data with or without carry

ADD A, #data Add the immediate data to accumulator A and store the result in A.

EXAMPLE 3.45

```
                        ADD A, #23      [A] ← 23 + [A]
                     Before execution   After execution
[A]                        23                 46
```

ADDC A, #data Add the immediate data and the contents of carry flag to accumulator A and store the result in A.

EXAMPLE 3.46

```
                       ADDC A, #23   [A] ← 23 + [A]+[CY]
                     Before execution   After execution
[A]                        20                 44
[CY]                        1                  0
```

Add the contents of accumulator with register Rn with or without carry

ADD A,Rn Add the contents of register Rn to accumulator A and store the result in A.

EXAMPLE 3.47

```
                        ADD A,R2      [A] ← [R2] + [A]
                     Before execution   After execution
[A]                        23                 78
[R2]                       55                 55
```

ADDC A, Rn Add the contents of register Rn and the contents of carry flag to accumulator A and store the result in A.

EXAMPLE 3.48

```
                        ADDC A,R2    [A] ← [R2] + [A]+[CY]
                    Before execution   After execution
[A]                        23                79
[R2]                       55                55
[CY]                        1                 0
```

Add the contents of accumulator with contents of memory with or without carry using direct and indirect addressing

ADD A, direct Add the contents of direct memory to accumulator A and store the result in A.

EXAMPLE 3.49

```
                        ADD A, 50     [A] ← M [50] + [A]
                    Before execution   After execution
[A]                        23                78
M [50]                     55                55
```

ADDC A, direct Add the contents of direct memory and the contents of carry flag to accumulator A and store the result in A.

EXAMPLE 3.50

```
                        ADDC A, 50    [A] ← M[50] + [A]+[CY]
                    Before execution     After execution
[A]                        23                79
M [50]                     55                55
[CY]                        1                 0
```

ADD A,@Ri Add the contents of memory location pointed to by Ri to accumulator A and store the result in A.

EXAMPLE 3.51

```
                        ADD A, @R0    [A] ← M [[R0]] + [A]
                    Before execution   After execution
[A]                        23                78
[R0]                       23                23
M [23]                     55                55
```

ADDC A,@Ri Add the contents of memory location pointed to by Ri and the contents of carry flag to accumulator A and store the result in A.

EXAMPLE 3.52

```
                    ADDC A, @R0   [A] ← M [[R0]] + [A] + [CY]
                 Before execution      After execution
[A]                    23                    79
[R0]                   23                    23
M [23]                 55                    55
[CY]                    1                     0
```

SUB Group of Instructions

In this group, we have instructions to

1. Subtract the immediate data and the contents of carry flag from A and store the result in A.
2. Subtract the contents of register Rn and the contents of carry flag from A and store the result in A.
3. Subtract the contents of memory and the contents of carry flag from A and store the result in A using direct and indirect addressing.

Note

CY, AC, and OV flags are affected in this group.

Subtract immediate data and the contents of carry flag from A and store the result in A

SUBB A, #data

EXAMPLE 3.53

```
                    SUBB A, #23    [A] ← [A] -23-[CY]
                 Before execution   After execution
[A]                    24                  00
[CF]                    1                   0
```

Subtract the contents register Rn and the contents of carry flag from A and store the result in A

SUBB A,R2

EXAMPLE 3.54

```
                    SUBB A,R2     [A] ← [A]-[CY]-[R2]
                 Before execution   After execution
[A]                    23                  CE
[R2]                   55                  55
[CY]                    0                   1
```

Subtract the contents of memory and the contents of carry flag from A and store the result in A using direct and indirect addressing

SUBB A, direct Subtract the contents of direct memory and the contents of carry flag from A and store the result in A.

EXAMPLE 3.55

SUBB A, 45 [A] ← [A]–M[45] –[CY]

	Before execution	After execution
[A]	23	D3
M [45]	50	50
[CY]	0	1

SUBB A, @Ri Subtract the contents of memory location pointed by Ri and the contents of carry flag from A and store the result in A.

EXAMPLE 3.56

SUBB A, @R0 [A] ← [A] –M [[R0]] –[CY]

	Before execution	After execution
[A]	55	4F
[R0]	23	23
M [23]	05	05
[CY]	1	0

Multiplication

MUL AB Multiplies the unsigned 8 bit integer in the A and register B. Low order byte of the result is stored in the A and high order byte of the result is stored in B. After the execution, the carry flag is cleared.

EXAMPLE 3.57

MUL AB [B][A] ← [B] × [A]

	Before execution	After execution
[A]	23	9F
[B]	55	0B

Division

DIV AB Divides the unsigned 8 bit integer in the A by the unsigned 8 bit integer in register B. The quotient is stored in the A and the remainder is stored in B register. The carry and overflow flag are cleared. If the content of B is zero, then OV is set.

EXAMPLE 3.58

```
                    DIV AB        Q -[A] R-[B] ← [A] / [B]
              Before execution       After execution
[A]                 23                    07
[B]                 05                    00
```

Decimal Adjustment After Addition

DA A When two Binary Coded Decimal (BCD) numbers are added, the answer is a non–BCD number. To get the result in BCD, DA A instruction is executed after adding two BCD numbers and storing the result in an A. The working of DA A is as follows.

- If lower nibble of A ≤ 9 and auxiliary flag is 0, then the lower nibble will not be altered.
- If lower nibble is greater than 9 or auxiliary flag is set, then 6 is added to lower nibble and after addition, if there is a carry, then upper nibble is incremented by one.
- If upper nibble of A ≤ 9 and carry flag is 0, then the upper nibble will not be altered.
- If upper nibble is greater than 9 or carry flag is set, then 6 is added to upper nibble and after addition, if there is a carry, carry flag is set.

Operation:

IF [(A3 – 0) > 9 V (AC) = 1]
THEN (A3 – 0) ← (A3 – 0) + 6, (A7 – 4) ← (A7 – 4) + (AC)
IF [(A7 – 4) > 9 V (CY) = 1]
THEN (A7 – 4) ← (A7 – 4) + 6

EXAMPLE 3.59

```
          Before      After execution    After execution
        execution     of ADD A, R1          of DA A
[A]        23             78                  78
[R1]       55             55                  55
```

EXAMPLE 3.60

	Before execution	After execution of ADD A, R1	After execution of DA A
[A]	53	AB	11
[R1]	58	58	58
[CY]	0	0	1

Clear A

CLR A [A] ← 0, Flags are not affected.

EXAMPLE 3.61

CLR A

	Before execution	After execution
[A]	23	00

Increment and Decrement Instructions

INC <byte> INC increments the indicated variable by 1. Flags are not affected. Register, direct and register indirect addressing modes are used.

INC A Increments the contents of A by 1: [A] ← [A]+1

EXAMPLE 3.62

INC A

	Before execution	After execution
[A]	FF	00

INC Rn

INC R5 Increments the contents of register 5 by 1: [R5] ← [R5]+1

EXAMPLE 3.63

INC R5

	Before execution	After execution
[R5]	FA	FB

INC direct

INC 50 Increments the contents of memory location 50 by 1.

EXAMPLE 3.64

```
                              INC 50          M [50] ← M [50]+1
                          Before execution    After execution
                M [50]          0A                  0B
```

INC @Ri

INC @Ro Increments the contents of memory location using indirect addressing mode by 1.

M [[Ro]] ← M[[Ro]]+1

EXAMPLE 3.65

```
                              INC @R0
                          Before execution    After execution
                [R0]              65                  65
                M[65]             0F                  10
```

INC DPTR

INC DPTR Increments the 16 bits data pointer by 1: [DPTR] ← [DPTR] +1

EXAMPLE 3.66

```
                              INC DPTR
                          Before execution    After execution
              [DPTR]           0F00                0F01
```

DEC <byte> DEC decrements the indicated variable by 1. Flags are not affected. Register, direct and register indirect addressing modes are used.

DEC A Decrements the contents of A by 1: [A] ← [A] – 1

EXAMPLE 3.67

```
                              DEC A
                          Before execution   After execution
                [A]               0F                 0E
```

DEC Rn

DEC R5 Decrements the contents of register 5 by 1: [R5] ← [R5] – 1

EXAMPLE 3.68

```
                              DEC R5
                        Before execution   After execution
[R5]                          FA                 F9
```

DEC direct

DEC 6o Decrements the contents of memory location 6o by 1.

M [6o] ← M [6o]–1

EXAMPLE 3.69

```
                              DEC 60
                        Before execution   After execution
M [60]                        0A                 09
```

DEC @Ri

DEC @Ro Decrements the contents of memory location using indirect addressing mode by 1.

M [[Ro]] ← M [[Ro]]–1

EXAMPLE 3.70

```
                             DEC @R0
                        Before execution   After execution
[R0]                          61                 61
M [61]                        0F                 0E
```

We will study following examples to become familiar with **arithmetic group instructions**.

EXAMPLE 3.71

Write 8051 instructions to add contents of A with the contents of register R2 and store the result in the A (assume result is 8 bit).

ALGORITHM

```
Step 1: Load data 35H to A
Step 2: Load data 57H to R2
Step 3: Add contents of A with R2
```

```
Instructions are as follows
        MOV A, #35H  ; Load data 35H to A
        MOV R2, #57H ; Load data 57H to R2
        ADD A,R2     ; Add contents of A with R2
```

EXAMPLE 3.72

Write 8051 instructions to subtract contents of register R2 from the contents of A and store the result in the A (assume result is 8 bit).

ALGORITHM

```
        Step 1: Load data 45H to A
        Step 2: Load data 27H to R2
        Step 3: Clear the carry flag
        Step 4: Subtract contents of R2 and CY flag from A
Instructions are as follows
        MOV A, #45H  ; Load data 45H to A
        MOV R2, #27H ; Load data 27H to R2
        CLR C        ; Clear carry flag
        SUBB A,R2    ; SUB contents of R2 and CY flag from A
```

EXAMPLE 3.73

Write 8051 instructions to multiply contents of register B with the contents of A and store the result in locations 50H and 51H. Assume that the least significant byte of result is stored in 50H.

ALGORITHM

```
        Step 1: Load data 45H to A
        Step 2: Load data 27H to B
        Step 3: Multiply contents of A with B
        Step 4: Store least significant byte of result in 50H
        Step 5: Store most significant byte of result in 51H
Instructions are as follows
        MOV A, #45H  ; Load data 45H to A
        MOV B, #27H  ; Load data 27H to B
        MUL AB       ; Multiply contents of A and B
        MOV 50,A     ; Store LSB of result in 50H
        MOV 51,B     ; Store MSB of result in 51H
```

EXAMPLE 3.74

Write 8051 instructions to divide the contents of an A with the contents of B register and store the remainder in 60H and quotient in 61H.

ALGORITHM

```
Step 1: Load data 45H to A
Step 2: Load data 27H to B
Step 3: Divide contents of A by B
Step 4: Store remainder in 60H
Step 5: Store quotient in 61H
```

Instructions are as follows

```
MOV A, #45H  ; Load data 45H to A
MOV B, #27H  ; Load data 27H to B
DIV AB       ; Divide contents of A with B
MOV 60,B     ; Store remainder in 60H
MOV 61,A     ; Store quotient in 61H
```

EXAMPLE 3.75

Write 8051 instructions to add two BCD numbers and store the result in BCD in register R1.

ALGORITHM

```
Step 1: Load data 09 to A
Step 2: Add data 07 to A
Step 3: Convert the result to BCD
Step 4: Store the result in R1
```

Instructions are as follows

```
MOV A, #09   ; Load data 09 to A
ADD A, #07   ; Add data 07 to A
DAA          ; Convert result to BCD
MOV R1,A     ; Move contents of R1 to A
```

3.6.3 LOGICAL INSTRUCTIONS

Logical instructions are very useful in input/output operations. They can perform various operations such as AND, OR, EX-OR, Complement, Swap and Rotate. In this group, we have instructions for the following.

1. AND the contents of A with immediate data or with contents of register or with contents of memory using direct and indirect addressing mode.
2. OR the contents of A with immediate data or with contents of register or with contents of memory using direct and indirect addressing mode.

3. EX-OR the contents of A with immediate data or with contents of register or with contents of memory using direct and indirect addressing mode.
4. Complement the contents of A.
5. Swap the upper and lower nibble of A.
6. Rotate instructions: In this group we have instructions for the following.
   - Rotate left contents of A
   - Rotate right contents of A
   - Rotate left including carry contents of A
   - Rotate right including carry contents of A

Note

In AND, OR, EX-OR and SWAP, instructions flags are not affected.

AND Instructions

AND the contents of A with immediate data

ANL A, #data This instruction AND the contents of an A with immediate data and stores the result in the A.

EXAMPLE 3.76

```
                          ANL A, #02      [A] ← [A] Λ02
                       Before execution  After execution
[A]                          24               00
```

AND the contents of A with contents of register Rn

ANL A, Rn This instruction AND the contents of an A with contents of register Rn and stores the result in the A.

EXAMPLE 3.77

```
                          ANL A,R2      [A] ← [A] Λ [R2]
                       Before execution  After execution
[A]                          24               00
[R2]                         00               00
```

AND the contents of A with contents of memory using direct addressing

ANL A, direct This instruction AND the contents of an A with contents of memory using direct addressing mode and stores the result in the A.

EXAMPLE 3.78

```
                        ANL A, 67     [A] ← [A] Λ M[67]
                    Before execution  After execution
[A]                        24               00
M [67]                     00               00
```

AND the contents of A with contents of memory using indirect addressing

ANL A, @Ri This instruction AND the contents of an A with contents of memory using indirect addressing mode and stores the result in the A.

EXAMPLE 3.79

```
                        ANL A,@R0    [A] ← [A] Λ M[[R0]]
                    Before execution  After execution
[A]                        24               00
[R0]                       55               55
M [55]                     00               00
```

AND the contents of memory using direct addressing with the contents of A

ANL direct,A This instruction AND the contents of memory using direct addressing mode with the contents of an A and stores the result in memory.

EXAMPLE 3.80

```
                        ANL 76, A    M[76] ← M[76] Λ [A]
                    Before execution  After execution
[A]                        00               00
M [76]                     55               00
```

AND the contents of memory using direct addressing with immediate data

ANL direct, #data This instruction AND the contents of memory using direct addressing mode with immediate data.

EXAMPLE 3.81

```
                      ANL 76, #55   M[76] ← M[76] Λ 55
                    Before execution  After execution
M [76]                     53               51
```

OR Instructions

OR the contents of A with immediate data

ORL A, #data This instruction OR the contents of A with immediate data and stores the result in the A.

EXAMPLE 3.82

```
                          ORL A, #22      [A] ← [A] V 22
                       Before execution  After execution
            [A]              24               26
```

OR the contents of A with contents of register Rn

ORL A, Rn This instruction OR the contents of an A with contents of register Rn and stores the result in the A.

EXAMPLE 3.83

```
                          ORL A, R2      [A] ← [A] V [R2]
                       Before execution  After execution
            [A]              24               24
            [R2]             00               00
```

OR the contents of A with contents of memory using direct addressing

ORL A, direct This instruction OR the contents of A with contents of memory using direct addressing mode and stores the result in the A.

EXAMPLE 3.84

```
                          ORL A, 50     [A] ← [A] V M [50]
                       Before execution  After execution
            [A]              24               2C
          M [50]             08               08
```

OR the contents of A with contents of memory using indirect addressing

ORL A, @ Ri This instruction OR the contents of an A with contents of memory using indirect addressing mode and stores the result in the A.

EXAMPLE 3.85

```
                        ORL A, @R0   [A] ← [A] V M [[R0]]
                    Before execution   After execution
[A]                        24                25
[R0]                       55                55
M[55]                      01                01
```

OR the contents of memory using direct addressing with the contents of A

ORL direct, A This instruction OR the contents of memory using direct addressing mode with the contents of an A and stores the result in memory.

EXAMPLE 3.86

```
                        ORL 76, A    M[76] ← M[76] V [A]
                    Before execution   After execution
[A]                        00                00
M [76]                     55                55
```

OR the contents of memory using direct addressing with immediate data

ORL direct, #data This instruction OR the contents of memory using direct addressing mode with immediate data.

EXAMPLE 3.87

```
                        ORL 76, #55   M[76] ← M[76] V 55
                    Before execution   After execution
M [76]                     53                57
```

EX-OR Instructions

EX-OR the contents of A with immediate data

XRL A, #data This instruction Exclusive-OR the contents of an A with immediate data and stores the result in the A.

EXAMPLE 3.88

```
                        XRL A, #22       [A] ← [A] ∀ 22
                    Before execution   After execution
A                          24                06
```

EX-OR the contents of A with contents of register Rn

XRL A, Rn This instruction Exclusive–OR the contents of an A with contents of register Rn and stores the result in the A.

EXAMPLE 3.89

```
                    XRL A, R2      [A] ← [A] ∀ [R2]
                Before execution  After execution
[A]                    24              24
[R2]                   00              00
```

EX-OR the contents of A with contents of memory using direct addressing

XRL A, direct This instruction Exclusive–OR the contents of an A with contents of memory using direct addressing mode and stores the result in the A.

EXAMPLE 3.90

```
                    XRL A, 55     [A] ← [A] ∀ M [55]
                Before execution  After execution
[A]                    24              25
M [55]                 01              01
```

EX-OR the contents of A with contents of memory using indirect addressing

XRL A, @Ri This instruction Exclusive-OR the contents of an A with contents of memory using indirect addressing mode and stores the result in the A.

EXAMPLE 3.91

```
                    XRL A,@R0     [A] ← [A] ∀ M[[R0]]
                Before execution  After execution
[A]                    24              04
[R0]                   55              55
M[55]                  20              20
```

EX-OR the contents of memory using direct addressing with the contents of A

XRL direct, A This instruction EX-OR the contents of memory using direct addressing mode with the contents of A and stores the result in memory.

EXAMPLE 3.92

```
                          XRL 76, A    M[76] ← M[76]  ∀ [A]
                      Before execution  After execution
[A]                         02                02
M [76]                      55                57
```

EX-OR the contents of memory using direct addressing with immediate data

XRL direct, #data This instruction EX-OR the contents of memory using direct addressing mode with immediate data.

EXAMPLE 3.93

```
                          XRL 76, #55   M[76] ← M [76]  ∀ 55
                      Before execution  After execution
M [76]                      53                56
```

Complement Contents of A

This instruction performs logical inversion of the contents of A. It changes 0's to 1's and 1's to 0's. It performs the NOT operation.

EXAMPLE 3.94

```
                            CPL A
                      Before execution  After execution
[A]                         FF                00
```

Swap the Upper and Lower Nibble of A

This instruction interchanges the low and high order nibbles of the A. The operation can also be thought of as a 4 bit rotate instruction. No flags are affected.

EXAMPLE 3.95

```
                           SWAP A        [A3-0] ↔ [A7-4]
                      Before execution  After execution
[A]                         F8                8F
```

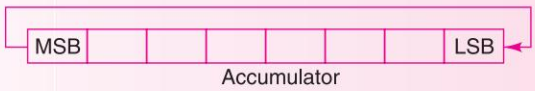
Rotate Instructions: In this group, we have instructions for the following.

- Rotate left contents of A
- Rotate right contents of A
- Rotate left including carry contents of A
- Rotate right including carry contents of A

Rotate left contents of A Rotate A left one place.

EXAMPLE 3.96

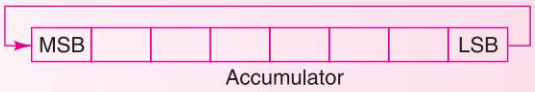
```
                    RL A
            Before execution After execution
[A]              F8               F1
```



Rotate right contents of A Rotate A right one place.

EXAMPLE 3.97

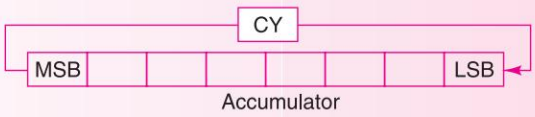
```
                    RR A
            Before execution After execution
[A]              F8               7C
```



Rotate left including carry contents of A Rotate A left one place through carry.

EXAMPLE 3.98

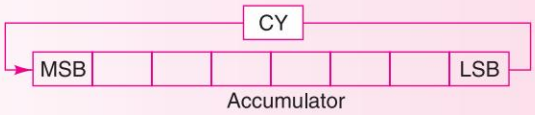
```
                    RLC A
            Before execution After execution
[A]              F8               F 1
[CY]              1                1
```



Rotate right including carry contents of A Rotate A right one place through carry.

EXAMPLE 3.99

```
                RRC A
          Before execution After execution
[A]             F8              FC
[CY]            1               0
```



We will study following examples to become familiar with *logical group instructions.*

EXAMPLE 3.100

Write 8051 instructions to mask most significant bit (MSB) and least significant bit (LSB) of A.

ALGORITHM

```
        Step 1: Load data 85H to A
        Step 2: Mask MSB and LSB using AND instruction
Instructions are as follows
        MOV A, #85H    ; Load data 85H to A
        ANL A, #7EH    ; Mask MSB and LSB of A
```

EXAMPLE 3.101

Write 8051 instructions to set LSB+1 and LSB of A.

ALGORITHM

```
        Step 1: Load data 80H to A
        Step 2: Set LSB+1 and LSB using OR instruction
Instructions are as follows
        MOV A, #80H    ; Load data 80H to A
        ORL A, #03H    ; set LSB+1 and LSB of A
```

EXAMPLE 3.102

Write 8051 instructions to swap the upper and lower nibble of location 20H.

ALGORITHM

```
        Step 1: Load contents of 20H to A
        Step 2: Swap contents of A
        Step 3: Copy contents of A to 20H
Instructions are as follows
        MOV A, 20H     ; Load contents of 20H to A
        SWAP A         ; Swap lower and upper nibble of A
        MOV 20H, A     ; Load contents of A to 20H
```

3.6.4 BRANCH (JUMP) INSTRUCTIONS

8051 instructions have very powerful branch instructions. When a branch instruction is executed by the 8051, a jump occurs, which can be a forward or a backward jump. It can perform branch unconditionally or based on a flag value. The 8051 supports two types of jump instructions.

- Unconditional Jump Instructions
- Conditional Jump Instructions

Unconditional Jump Instructions

The execution of this instruction always results in a branch. The destination address is provided as a part of the instruction.

LJMP addr16 LJMP causes an unconditional branch to the indicated address, by loading the program counter with 16 bits address i.e. the second and third byte of the instruction respectively.

EXAMPLE 3.103

```
LJMP 1200        [PC] ← 1200
```

AJMP addr11 AJMP causes an unconditional branch to the indicated address, by loading the 11 bit address to 0 to 10 bits of the program counter. The destination must therefore be within the same 2K blocks.

EXAMPLE 3.104

```
AJMP 200       [PC 10-0] ← 200
```

SJMP rel The branch destination is computed by adding the signed displacement to the contents of program counter. Therefore, it branches from 128 bytes preceding this instruction to 127 bytes following it.

EXAMPLE 3.105

```
SJMP 25     [PC] ← [PC] + 2 + 25
```

JUMP Indirect The contents of an A is added (8 bit unsigned) with 16 bit contents of data pointer and this 16 bit is loaded to program counter. Neither the A nor the data pointer is altered and no flags are affected.

EXAMPLE 3.106

```
JUMP @A+DPTR:   [PC] ← [A]+[DPTR]
```

Compare and jump if not equal (CJNE <dest–byte>, <src–byte>,rel) CJNE compares the magnitude of the first two operands, and branches if their values are not equal. Branch address is computed by adding the signed displacement to the contents of Program Counter (PC). The carry flag is set, if unsigned integer value of <dest–byte> is less than the unsigned integer value of <src–byte>, else carry flag is cleared.

CJNE A, direct, rel:

Operation: [PC] ← [PC]+3
IF [A] ≠ M[direct]
THEN [PC] ← [PC]+displacement
IF [A] < M[direct]
THEN [CY] ← 1
ELSE [CY] ← 0

CJNE A, #data,rel

Operation: [PC] ← [PC]+3
IF [A] ≠ #data
THEN [PC] ← [PC]+displacement
IF [A] < #data
THEN [CY] ← 1
ELSE [CY] ← 0

CJNE Rn,#data,rel

Operation: [PC] ← [PC]+3
IF [Rn] ≠ #data
THEN [PC] ← [PC]+displacement
IF [Rn] < #data
THEN [CY] ← 1
ELSE [CY] ← 0

CJNE @Ri,#data,rel

Operation: [PC] ← [PC]+3
IF M [[Ri]] ≠ # data
THEN [PC] ← [PC]+displacement
IF M[[Ri]] < # data
THEN [CY] ← 1
ELSE [CY] ← 0

Decrement and jump if not zero (DJNZ <byte>,<rel–addr>) DJNZ decrements the contents of the memory or register by 1, and if the resulting value is not zero, it branches to the relative address indicated by the second operand. No flags are affected and the branch destination is computed by adding the signed displacement value to the contents of program counter.

DJNZ Rn,rel

Operation: [PC] ← [PC] + 2
[Rn] ← [Rn] – 1
IF [Rn] ≠ 0
THEN [PC] ← [PC]+ displacement

DJNZ direct,rel

Operation: [PC] ← [PC] + 2
M[direct] ← M[direct] – 1
IF M[direct] ≠ 0
THEN [PC] ← [PC]+displacement

Jump if A is zero (JZ rel) If A is zero, branch to the address indicated; otherwise execute the next instruction. The branch address is computed by adding the signed displacement to the contents of PC, after incrementing the PC twice. Flags are not affected.

Operation: [PC] ← [PC]+2
IF [A] = 0 THEN [PC] ← [PC] + displacement

Jump if A is not zero (JNZ rel) If A is not zero, branch to the address indicated; otherwise execute the next instruction. The branch address is computed by adding the signed displacement to the contents of PC, after incrementing the PC twice. Flags are not affected.

Operation: [PC] ← [PC]+2
IF [A] ≠ 0 THEN [PC] ← [PC] + displacement

Conditional Jump Instructions

In this group, the instruction performs a branch, based on the condition of a single status flag. The status flag used in this instruction is only a carry flag.

Jump if carry is set (JC rel) If the carry flag is set, branch to the address indicated; otherwise execute the next instruction. Flags are not affected. The branch destination is computed by adding the signed displacement to the contents of PC, after incrementing the contents of PC by 2.

Operation: [PC] ← [PC]+2
IF [CY] = 1 THEN (PC) ← (PC)+displacement

Jump if carry is not set (JNC rel) If the carry flag is not set, branch to the address indicated; otherwise execute the next instruction. Flags are not affected. The branch destination is computed by adding the signed displacement to the contents of PC, after incrementing the contents of PC by 2.

Operation: [PC] ← [PC]+2
IF [CY]=0 THEN [PC] ← [PC]+displacement

We will study following examples to become familiar with *JUMP instructions*.

EXAMPLE 3.107

Write 8051 instructions to decrement the contents of R2 until it becomes zero.

ALGORITHM

```
Step 1: Load data to register R2
Step 2: Decrement R2 until it becomes zero.
```

Instructions are as follows

```
MOV R2, #FFH         ; Load FFH to R2
LOOP: DJNZ R2, LOOP ; decrement R2 until it becomes zero
```

EXAMPLE 3.108

Write 8051 instructions to rotate the contents of A left by two positions.

ALGORITHM

```
Step 1: Load data to A
Step 2: Load 02 to register R2
Step 3: Rotate contents of A left by one position
Step 4: Decrement R2 and repeat Step 3 until R2 becomes zero
```

Instructions are as follows

```
MOV A, #25H   ; Load 25H to A
MOV R2, #02H  ; Load 02H to R2
LOOP: RLA     ; Rotate contents of A left by one position
DJNZ R2, LOOP; Decrement R2 and branch to LOOP until R2 becomes
               zero
```

EXAMPLE 3.109

Write 8051 instructions to compare the contents of A with the contents of 20H. If contents are equal, store 00H in 21H, else store FFH in 21H.

ALGORITHM

```
Step 1: Load data 25H to A
Step 2: Load data 15H to memory location 20H
Step 3: Load 00H to register R3
Step 4: Clear carry flag
Step 5: Sub contents of location 20H and contents of CY flag from
        contents of A
Step 6: If contents are equal, branch to Step 8
Step 7: Decrement R3, then R3 contains FFH
Step 8: Store contents of R3 in location 21H
```

```
   MOV A, #25H          ; Load data 25H to A
   MOV 20, #15H         ; Load data 15H to location 20H
   MOV R3,#00           ; Load data 00H to R3
   CLR C                ; Clear carry flag
   SUBB A, 20H          ; Sub contents of location 20H and C flag
                          from the contents of A
   JZ LOOP1             ; If result is zero, branch to LOOP1
   DEC R3               ; Decrement R3
   LOOP1: MOV 21H, R3 : If result is zero, store 00 in location 21H,
                          else store FF
```

EXAMPLE 3.110

Write 8051 instructions to add two 8 bit numbers and store 16 bit results in location 20H and 21H.

ALGORITHM

```
Step 1: Load data 77H to A
Step 2: Load data 99H to register R2
Step 3: Load 00H to register R3
Step 4: Add contents of register R2 with the contents of A
Step 5: Store contents of A (LSB result) in memory location 20H
Step 6: If carry flag is reset, branch to Step 8
Step 7: Increment R3, then R3 contains 01H
Step 8: Store contents of R3 in location 21H
```

```
MOV A, #77H          ; Load data 77H to A
MOV R2, #99H         ; Load data 99H to R2
MOV R3,#00           ; Load data 00H to R3
ADD A,R2             ; Add contents of A with R2
MOV 20H,A            ; Store contents of A in location 20H
JNC LOOP1            ; If carry flag is reset, branch to LOOP1
INC R3               ; Increment R3
LOOP1: MOV 21H, R3 ; Store contents of R3 in location 21H
```

3.6.5 SUBROUTINE CALL AND RET INSTRUCTIONS

As discussed in Section 3.3 subroutines are handled by

- CALL instructions
- RET instructions

CALL Instructions

The 8051 provides two types of CALL instructions.

LCALL addr16 This instruction is called *long call instruction* and it unconditionally calls a subroutine located at the indicated address. This is a 3 byte instruction. The instruction increments program counter by 3 and generates the address of the next instruction and saves this address in a stack. The program counter is then loaded with the 16 bit address of the LCALL instruction. Flags are not affected.

Operation: [PC] ← [PC]+3
[SP] ← [SP]+1
[[SP]] ← [PC_{7-0}]
[SP] ← [SP]+1
[[SP]] ← [PC_{15-8}]
[PC] ← addr 15–0

ACALL addr11 This instruction is called *absolute call instruction* and it unconditionally calls a subroutine located at the indicated address. This is a 2 byte instruction. The instruction increments the program counter by 2, and generates the address of the next instruction and saves this address in a stack. The program counter (PC_{10-0}) is then loaded with 11 bit address of the LCALL instruction. Flags are not affected.

Operation: [PC] ← [PC]+2
[SP] ← [SP]+1
[[SP]] ← [PC_{7-0}]
[SP] ← [SP]+1
[[SP]] ← [PC_{15-8}]
[PC_{10-0}] ← addr10–0

RET Instructions

The 8051 provides two types of RET instruction

RET: Return from subroutine RET instruction pops the top two contents of the stack and is loaded to PC. It decrements the stack pointer by 2 and flags are not affected.

Operation: $[PC_{15-8}] \leftarrow [[SP]]$
$[SP] \leftarrow [SP] - 1$
$[PC_{7-0}] \leftarrow [[SP]]$
$[SP] \leftarrow [SP] - 1$

RETI: Return from interrupt RETI instruction pops the top two contents of the stack and is loaded to PC. It decrements the stack pointer by 2 and flags are not affected. It restores the interrupt logic at the same priority level to accept the request from I/O devices.

Operation: $[PC_{15-8}] \leftarrow [[SP]]$
$[SP] \leftarrow [SP] - 1$
$[PC_{7-0}] \leftarrow [[SP]]$
$[SP] \leftarrow [SP] - 1$

3.6.6 BIT MANIPULATION INSTRUCTION

The 8051 supports a single bit operation. The internal RAM contains 128 addressable bits (20H to 2FH) 00 to 7FH. SFR supports 128 addressable bits and all the I/O ports are bit addressable. Each I/O line can be treated as a separate single bit port. The 8051 provides bit manipulation instructions to perform operations such as and, or, set, clear, complement and also conditional bit manipulation jump instructions.

And

AND contents of carry flag with Boolean value of the source bit or with complemented Boolean value of the source bit.

ANL C,bit: $[CY] \leftarrow [CY] \wedge [bit]$
ANL C,/bit: $[CY] \leftarrow [C\ Y] \wedge [\overline{bit}]$

OR

OR contents of carry flag with Boolean value of the source bit or with complemented Boolean value of the source bit.

ORL C,bit: $[CY] \leftarrow [C\ Y] \vee [bit]$
ORL C,/bit: $[CY] \leftarrow [C\ Y] \vee [\overline{bit}]$

CLR bit

The indicated bit is cleared. No flags are affected. CLR can operate on the carry flag or any directly addressable bit.

CLR bit: $[bit] \leftarrow 0$
CLR P1.2: $[P1.2] \leftarrow 0$
CLR C: $[CY] \leftarrow 0$

CPL bit

Bit variable specified is complemented. Mentioned bit, which had been a one, is changed to zero and vice-versa.

CPL bit: [bit] ← [$\overline{bit}$]
CPL P1.5: [P1.5] ← [$\overline{p1.5}$]
CPL C: [CY] ← [$\overline{C}$]

Bit Manipulation Branch Instructions

It checks the condition of the bit; if the condition is satisfied then it jumps to the address indicated; otherwise executes the next instruction. Instruction increments the program counter by 3 and then branch address is computed by adding the signed displacement to the contents of PC.

JB bit, rel: Jump if direct bit is set

Operation: [PC] ← [PC]+3
IF [bit] = 1 THEN [PC] ← [PC]+displacement

JNB bit, rel: Jump if direct bit is not set

Operation: [PC] ← [PC]+3
IF [bit] = 0 THEN [PC] ← [PC]+displacement

JBC bit, rel: Jump if direct bit is set and clear bit.

Operation: [PC] ← [PC]+3
IF [bit] = 1 THEN [PC] ← [PC]+displacement, [bit] ← 0

We will study following examples to become familiar with *bit manipulation instructions*

EXAMPLE 3.111

Write 8051 instructions to set carry flag and to complement MSB of 20H.

ALGORITHM

```
Step 1: Clear carry flag
Step 2: Complement carry flag, carry flag is set
Step 3: Complement MSB of register 20H
```

Instructions are as follows

```
CLR C          ; Clear carry flag
CPL C          ; Complement the carry flag, carry flag is set
CPL 07H        ; Complement MSB of register 20H
```

EXAMPLE 3.112

Write 8051 instructions to complement MSB of 22H. If bit is 1, store FFH in 40H, else store 00H.

ALGORITHM

```
Step 1: Load 00H to register R3
Step 2: Complement MSB of register 22H
Step 3: If bit=0, branch to step 5
Step 4: Decrement R3, then R3 contains FFH
Step 5: Store contents of R3 in location 40H
```

```
MOV R3, #00          ; Load data 00H to R3
CPL 17H              ; Complement MSB of register 22H
JNB 17H, LOOP1       ; If bit is 0, then branch to LOOP1
DEC R3               ; Decrement R3, then R3 contains FFH
LOOP1: MOV 40H, R3   ; Store contents of R3 in location 40H
```

SECTION REVIEW

1. List the types of data transfer instructions.
2. MOV R2, 50 is an example for _____ addressing.
3. MOVX instruction is used to move data from _____ RAM.
4. List various MOV instructions.
5. If [SP]=00, after PUSH operation [SP]= _____.
6. _____ instruction exchanges low order nibble of the A with internal RAM location.
7. List the flags affected by arithmetic instructions.
8. After multiplication operation, the carry flag is _____.
9. After division operation, OV flag is _____.
10. List the steps in working of DAA instruction.
11. Write 8051 instructions to implement DEC DPTR instruction.
12. _____ instruction is used to MASK bits of A.
13. List the logical type instructions which affect the flag registers.
14. _____ instruction interchanges low and high order nibble of the A.
15. _____ instruction is used to rotate contents of A left through carry flag.
16. List the types of branch instructions.

17. List the operation of JC 25 instruction.
18. _____ instruction restores the interrupt logic at the same priority level.
19. List bit manipulation logical instructions.
20. List bit manipulation branch instructions.

3.7 INSTRUCTION SET SUMMARY

DATA TRANSFER INSTRUCTIONS

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
MOV A,Rn	MOV A,R1	[A] ← [R1]	Copy contents of Register R1 to *A*	1	1	–	–	–
MOV A,direct	MOV A, 10H	[A] ← M [10]	Move direct byte to ACC	2	1	–	–	–
MOV A,@Ri	MOV A, @R1	[A] ← M [[R1]]	Move indirect byte to ACC	1	1	–	–	–
MOV A,#data	MOV A, #21	[A] ← 21	Move immediate data to ACC	2	1	–	–	–
MOV Rn,A	MOV R5,A	[R5] ← [A]	Move ACC to register	1	1	–	–	–
MOV Rn,direct	MOV R5, 10H	[R5] ← [10]	Move direct byte to register	2	2	–	–	–
MOV Rn,#data	MOV R5, #21	[R5] ← 21	Move immediate data to register	2	1	–	–	–
MOV direct,A	MOV 10, A	M[10] ← [A]	Move ACC to direct byte	2	1	–	–	–
MOV direct,Rn	MOV 10, R3	M[10] ← [R3]	Move register to direct byte	2	2	–	–	–
MOV direct,direct	MOV 10, 20	M[10] ← M[20]	Move direct byte to direct byte	3	2	–	–	–
MOV direct,@ Ri	MOV 10, @R0	M[10] ← M[[R0]]	Move indirect RAM to direct byte	3	2	–	–	–
MOV direct,#data	MOV 10, #23	M[10] ← 23	Move immediate data to direct byte	3	2	–	–	–
MOV @Ri,A	MOV @R0,A	M[[R0]] ← [A]	Move ACC to indirect RAM	1	1	–	–	–
MOV @ Ri,direct	MOV @R1,10	M[[R1]] ← M[10]	Move direct byte to indirect RAM.	2	2	–	–	–
MOV @ Ri,#data	MOV @R0,#23	M[[R0]] ← 23	Move immediate data to indirect RAM	2	1	–	–	–
MOV DPTR,#data16	MOV DPTR, #2300	[DPTR] ← 2300	Load data pointer with 16 bit constant	3	2	–	–	–
MOVC A, @ A + DPTR	MOVC A, @ A + DPTR	[A] ← PM[[A+DPTR]]	Move code byte at ACC+DPTR to ACC	1	2	–	–	–

(*Contd*)

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
MOVC A,@ A+PC	MOVC A,@ A+PC	[A] ← PM[[A+PC]]	Move code byte at ACC+PC to ACC	1	2	–	–	–
MOVX A,@Ri	MOVX A, @R1	[A] ← EDM [[R1]]	Move external RAM to ACC	1	2	–	–	–
MOVX @Ri,A	MOVX @R0, A	EDM [[R0]] ← A	Move ACC to external RAM	1	2	–	–	–
MOVX A,@ DPTR	MOVX A,@ DPTR	[A] ← EDM [[DPTR]]	Move external RAM to ACC	1	2	–	–	–
PUSH direct	PUSH 20	[[SP]] ← M[20]	Push direct byte to stack	2	2	–	–	–
POP direct	POP 10	M[10] ← [[SP]]	Pop direct byte from stack	2	2	–	–	–
XCH A,Rn	XCH A, R4	[A] ↔ [R4]	Exchange register with ACC	1	1	–	–	–
XCH A,direct	XCH A, 20	[A] ↔ M [20]	Exchange direct byte with ACC	2	1	–	–	–
XCH A,@Ri	XCH A, @R1	[A] ↔ M [[R1]]	Exchange indirect RAM with ACC	1	1	–	–	–
XCHD A, @Ri	XCHD A, @R0	[ALB] ↔ M [[R0]] LB	Exchange low order digit indirect RAM with ACC	1	1	–	–	–

ARITHMETIC INSTRUCTIONS

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
ADD A, Rn	ADD A, R2	[A] ← [A] + [R2]	Add register to ACC	1	1	X	X	X
ADD A, direct	ADD A, 10	[A] ← [A] + M[10]	Add direct byte to ACC	2	1	X	X	X
ADD A, @Ri	ADD A, @R0	[A] ← [A]+ M[[R0]]	Add indirect RAM to ACC	1	1	X	X	X
ADD A, #data	ADD A, #23	[A] ← [A] + 23	Add immediate data to ACC	2	1	X	X	X
ADDC A, Rn	ADDC A, R3	[A] ← [A] + [R3] + [C]	Add register to ACC with carry	1	1	X	X	X
ADDC A, direct	ADDC A,10	[A] ← [A]+[C]+ M[10]	Add direct byte to ACC with carry	2	1	X	X	X
ADDC A, @Ri	ADDC A,@R1	[A] ← [A]+[C]+ M[[R1]	Add indirect RAM to ACC with carry	1	1	X	X	X
ADDC A, #data	ADDC A,#23	[A] ← [A] +23+[C]	Add immediate data to ACC with carry	2	1	X	X	X
SUBB A, Rn	SUBB A,R6	[A] ← [A] – [C] – [R6]	Subtract register from ACC with borrow	1	1	X	X	X
SUBB A, direct	SUBB A,10	[A] ← [A] – [C] – M[10]	Subtract indirect RAM from ACC with borrow	2	1	X	X	X

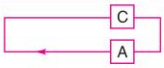
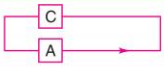
(*Contd*)

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
SUBB A, @Ri	SUBB A,@R1	[A] ← [A] – [C] – M[[R1]]	Subtract indirect RAM from ACC with borrow	1	1	X	X	X
SUBB A, #data	SUBB A, #23	[A] ← [A] – [C] – 23	Subtract immediate data from ACC with borrow	2	1	X	X	X
INC A	INC A	[A] ← [A] +1	Increment ACC	1	1	–	–	–
INC Rn	INC R3	[R3] ← [R3] +1	Increment register	1	1	–	–	–
INC direct	INC 10	M[10] ← M[10] +1	Increment direct byte	2	1	–	–	–
INC @Ri	INC @R0	M[[R0]] ← M[[R0]] +1	Increment direct RAM	1	1	–	–	–
DEC A	DEC A	[A] ← [A] – 1	Decrement ACC	1	1	–	–	–
DEC Rn	DEC R7	[R7] ← [R7] – 1	Decrement register	1	1	–	–	–
DEC direct	DEC 10	M[10] ← M[10]– 1	Decrement direct byte	2	1	–	–	–
DEC @Ri	DEC @R0	M[[R0]] ← M[[R0]] –1	Decrement indirect RAM	1	1	–	–	–
INC DPTR	INC DPTR	[DPTR] ← [DPTR] + 1	Increment data pointer	1	2	–	–	–
MUL AB	MUL AB	[B] [A] ← [A] x[B]	Multiply A and B	1	4	0	X	–
DIV AB	DIV AB	[A] [B] ← [A] /[B]	Divide A by B	1	4	0	X	–
DAA	DAA	$[A_{DEC}] \leftarrow [A_{BIN}]$	Decimal adjust ACC	1	1	X	–	–

LOGICAL INSTRUCTIONS

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
ANL A,Rn	ANL A,R3	[A]←[A]Λ[R3]	AND register to ACC	1	1	–	–	–
ANL A,direct	ANL A,20	[A]←[A]ΛM[20]	AND direct byte to ACC	2	1	–	–	–
ANL A,@Ri	ANL A,@R0	[A]←[A]ΛM[[R0]]	AND indirect RAM to ACC	1	1	–	–	–
ANL A,#data	ANL A,#27	[A]←[A]Λ27	AND immediate data to ACC	2	1	–	–	–
ANL direct,A	ANL 10,A	M[10] ← M[10]Λ[A]	AND ACC to direct byte	2	1	–	–	–
ANL direct, #data	ANL 10,#27	M[10] ← M[10] Λ27	AND immediate data to direct byte	3	2	–	–	–
ORL A,Rn	ORL A,R3	[A] ← [A] ⋁ [R3]	OR register to ACC	1	1	–	–	–
ORL A,direct	ORL A,10	[A] ← [A] ⋁ M[10]	OR direct byte to ACC	2	1	–	–	–
ORL A,@Ri	ORL A,@R0	[A] ← [A] ⋁ M[[R0]]	OR indirect RAM to ACC	1	1	–	–	–

(*Contd*)

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
ORL A,#data	ORL A,#23	[A] ← [A] ∨ 23	OR immediate data to ACC	2	1	–	–	–
ORL direct,A	ORL 10,A	M[10] ← M[10] ∨ [A]	OR ACC to direct byte	2	1	–	–	–
ORL direct,#data	ORL 10,#23	M[10] ← M[10] ∨ 23	OR immediate data to direct byte	3	2	–	–	–
XRL A,Rn	XRL A,R5	[A] ← [A] ∀ [R5]	XOR register to ACC	1	1	–	–	–
XRL A,direct	XRL A,10	[A] ← [A] ∀ M[10]	XOR direct byte to ACC	2	1	–	–	–
XRL A,@Ri	XRL A,@R1	[A] ← [A] ∀ [[R1]]	XOR indirect RAM to ACC	1	1	–	–	–
XRL A,#data	XRL A,#32	[A] ← [A] ∀ 32	XOR immediate data to ACC	2	1	–	–	–
XRL direct,A	XRL 10,A	M[10] ← M[10] ∀ [A]	XOR ACC to direct byte	2	1	–	–	–
XRL direct,#data	XRL 10,#32	M[10] ← M[10] ∀ 32	XOR immediate data to direct byte	3	2	–	–	–
CLR A	CLR A	[A] ← 0	Clear the ACC	1	1	–	–	–
CPL A	CPL A	[A] ← $[\overline{A}]$	Complement the ACC	1	1	–	–	–
RL A	RL A		Rotate the ACC left	1	1	–	–	–
RLC A	RLC A		Rotate the ACC left through carry	1	1	X	–	–
RR A	RR A		Rotate the ACC right	1	1	–	–	–
RRC A	RRC A		Rotate the ACC right through Carry	1	1	X	–	–
SWAP A	SWAP A	$A_{LSBN} \leftrightarrow A_{MSBN}$	Swap nibbles in the ACC	1	1	–	–	–

BRANCH INSTRUCTIONS

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
ACALL addr11	ACALL 200	STACK ← [PC]+2 $[PC]_{11-0}$ ← 200	Absolute call within 2K page	2	2	–	–	–
LCALL addr16	LCALL 1200	STACK ← [PC]+3 [PC] ← 1200	Absolute call (long call)	3	2	–	–	–
RET	RET	[PC] ← STACK	Return from subroutine	1	2	–	–	–
RETI	RETI	[PC] ← STACK & Restores interrupt logic	Return from interrupt	1	2	–	–	–

(*Contd*)

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
AJMP addr11	AJMP 200	$[PC]_{11-0} \leftarrow 200$	Absolute jump within 2K page	2	2	–	–	–
LJMP addr16	LJMP 1200	[PC] ← 1200	Absolute jump (long jump)	3	2	–	–	–
SJMP rel8	SJMP 25	[PC] ← [PC]+2+25	Relative jump within +/– 127 bytes (short jump)	2	2	–	–	–
JMP @A+DPTR	JMP @A+DPTR	[PC] ← [A]+[DPTR]	Jump direct relative to DPTR	1	2	–	–	–
JZ rel8	JZ 25	[A]=0[PC] ← [PC] + 2 +25h	Jump if ACC is zero	2	2	–	–	–
JNZ rel8	JNZ 25	[A] ≠ 0[PC] ← [PC] + 2 +25h	Jump if ACC is NOT zero	2	2	–	–	–
CJNE A,direct, rel8	CJNE A,25,30	[A] ≠ M[25] [PC] ← [PC] +3 +30h	Compare direct byte to ACC, jump if NOT equal	3	2	X	–	–
CJNE A,#data, rel8	CJNE A,#45,50h	[A] ≠ 45 [PC] ← [PC] +3 +50h	Compare immediate to ACC, jump if NOT equal	3	2	X	–	–
CJNE Rn,#data, rel8	CJNE R5,#45,25h	[R5] ≠ 45 [PC] ← [PC] +3 +25h	Compare immediate to register, jump if NOT equal	3	2	X	–	–
CJNE@ Ri,#data, rel 8	CJNE @ R0,48,27h	M[[R0]] ≠ 48 [PC] ← [PC] +3 +27h	Compare immediate to indirect, jump if NOT equal	3	2	X	–	–
DJNZ Rn, rel8	DJNZ R3,25	[R3]–1 ≠ 0 [PC] ← [PC]+2+25	Decrement register, jump if NOT zero	2	2	–	–	–
DJNZ direct, rel8	DJNZ 10,25	M[10]–1 ≠ 0 [PC] ←[PC]+3+25	Decrement direct byte, jump if NOT zero	3	2	–	–	–
NOP	NOP	[PC] ← [PC]+1	No operation (skip to next instruction)	1	1	–	–	–

BIT MANIPULATION INSTRUCTIONS

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
CLR C	CLR C	[C] ← 0	Clear carry flag	1	1	0	–	–
CLR bit	CLR 67	M[MSB2C] ← 1	Clear direct bit	2	1	–	–	
SETB C	SETB C	[c] ← 1	Set carry flag	1	1	1	–	–
SETB bit	SETB 48H	M[LSB 29] ← 0	Set direct bit	2	2	–	–	
CPL C	CPL C	[C] ← [$\overline{C}$]	Complement carry flag	1	1	X	–	–
CPL bit	CPL P1.0	[P1.0] ← [$\overline{P1.0}$]	Complement direct bit	2	1	–	–	–
ANL C,bit	ANL C,P2.5	[C] ← [C] $\vee$ [P2.5]	AND direct bit to carry	2	2	X	–	–

(*Contd*)

MNEMONIC	EXAMPLE	OPERATION	DESCRIPTION	BYTES	CYS	CY	OV	AC
ANL C,/bit	ANL C,/P2.5	[C] ← [C] ∨ $\overline{[P2.5]}$	AND complement of direct bit to carry	2	2	X	–	–
ORL C,bit	ORL C,P2.2	[C] ← [C] ∨ [P2.5]	OR direct bit to carry	2	2	X	–	–
ORL C,/bit	ORL C,/P2.2	[C] ← [C] ∨ $\overline{[P2.5]}$	OR complement of direct bit to carry	2	2	X	–	–
MOV C,bit	MOV C,P2.4	[C] ← [P2.4]	Move direct bit to carry	2	1	X	–	–
MOV bit,C	MOV P2.4, C	[P2.4] ← [C]	Move carry to direct bit	2	2	–	–	–
JC rel	JC rel	C=1 [PC] ← [PC]+2+rel	Jump if carry is set	2	2	–	–	–
JNC rel	JNC rel	C=0 [PC] ← [PC]+2+rel	Jump if carry is NOT set	2	2	–	–	–
JB bit,rel	JB PSW.2 rel	Bit=1, [PC] ← [PC]+3+rel	Jump if direct bit is set	3	2	–	–	–
JNB bit,rel	JNB P2.3,rel	Bit=0, [PC] ← [PC]+3+rel	Jump if direct bit is NOT set	3	2	–	–	–
JBC bit,rel	JBC P1.1,rel	Bit=1, [PC] ← [PC]+3+rel [bit] ← 0	Jump if direct bit is set and clear that bit	3	2	–	–	–

CHAPTER SUMMARY

This chapter presents an in depth understanding of the following.

- Instruction syntax, data types and subroutines.
- The various addressing modes of 8051. It covers Immediate Addressing, Register Addressing, Direct Addressing, Indirect Addressing, Relative Addressing, Absolute addressing, Long Addressing, Indexed Addressing, Bit Inherent Addressing and Bit Direct Addressing.
- Classification of 8051 instructions with examples.
- 8051 instruction set is summerized in a table with example, operation and description.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. _____ instruction is an example for direct addressing mode.
   (a) MOV A, @R1 (b) MOV A, #21H (c) MOV A, 10H (d) MOV R5, A
2. An alternate instruction for CLR C is _____.
   (a) CLR PSW.0 (b) CLR PSW.7 (c) CLR PSW.2 (d) CLR PSW.5

3. The 8051 supports _____.
   (a) Unsigned 8 bit numbers (b) Signed 8 bit numbers
   (c) Both (a) and (b) (d) None of the above
4. Direct addressing mode is used in _____.
   (a) Internal data memory
   (b) External data memory
   (c) Internal program memory
   (d) External program memory
5. MOVC A, @A+DPTR is example of _____.
   (a) Immediate addressing
   (b) Direct addressing
   (c) Indirect addressing
   (d) Indexed addressing
6. SETB 07H instruction is an example of_____.
   (a) Bit inherent addressing (b) Bit direct addressing
   (c) Bit immediate addressing (d) Bit indirect addressing
7. AJMP instruction is an example of _____.
   (a) Absolute addressing (b) Long addressing
   (c) Indexed addressing (d) Relative addressing
8. In the 8051, decimal data –127 is represented in binary as _____.
   (a) 10000001 (b) 01111111
   (c) 10000000 (d) 11111111
9. Execution time for MUL A, B instruction is _____.
   (a) 1 Machine cycles (b) 2 Machine cycles
   (c) 3 Machine cycles (d) 4 Machine cycles
10. If the 8051 is operated with 12 MHz, the clock period is _____.
    (a) 1 µs (b) 0.08333 µs (c) 0.8333 µs (e) None of the above
11. If the 8051 operates with 12 MHz, the clock signal execution time for the instruction MUL AB is _____.
    (a) 1 µs (b) 2 µs (c) 3 µs (e) 4 µs
12. To mask MSB of the A, we must ANL it with _____.
    (a) 7FH (b) 80H (c) FFH (d) 85H
13. To set LSB+1 bit of the A, we must ORL it with _____.
    (a) 02H (b) 01H (c) 04H (d) 08H
14. [A] = 29H, [R0] = 55H and M[55] = 33H. After execution of instruction XCHG A, @ R0, the contents of A will be _____.
    (a) 92H (b) 55H (c) 33H (d) 23H

15. [A] = 24H and [CY] = 1. After execution of instruction SUBB A, #23H, the contents of A will be.
    (a) 01H (b) 02H (c) 00H (d) 23H
16. INC R5 instruction affects _____.
    (a) CY flag (b) AC flag (c) OV flag (d) None of the above
17. If [A] =F8 before execution of instruction RRA, then after execution, [A] is _____.
    (a) 7CH (b) FCH (c) F0H (d) F1H
18. If AJMP instruction is executed, then the destination must be
    (a) Within the same 2 Kbytes
    (b) Within the same 4 Kbytes
    (c) Within the same 8 Kbytes
    (d) Within the same 16 Kbytes
19. In JNC rel instruction, the operation is _____.
    (a) Operation: [PC] ← [PC]+2, IF [CY]=1 THEN [PC] ← [PC]+displacement
    (b) Operation: [PC] ← [PC]+3, IF [CY]=1 THEN [PC] ← [PC]+displacement
    (c) Operation: [PC] ← [PC]+2, IF [CY]=0 THEN [PC] ← [PC]+displacement
    (d) Operation: [PC] ← [PC]+3, IF [CY]=0 THEN [PC] ← [PC]+displacement
20. In JB bit, rel instruction the operation is _____.
    (a) Operation: [PC] ← [PC]+3, IF [bit] = 1 THEN [PC] ← [PC]+displacement
    (b) Operation: [PC] ← [PC]+2, IF [bit] = 1 THEN [PC] ← [PC]+displacement
    (c) Operation: [PC] ← [PC]+3, IF [bit] = 0 THEN [PC] ← [PC]+displacement
    (d) Operation: [PC] ← [PC]+2, IF [bit] = 0 THEN [PC] ← [PC]+displacement

REVIEW QUESTIONS

3.1 What are the different addressing modes supported by the 8051? Explain with examples.

3.2 State and explain the addressing modes used in each of the following instructions.
    (a) MOV A,#25H (b) MOV A,@Ri (c) MOV R2,40H
    (d) MOVC A,@A+PC (e) DA A (f) LCALL 2000H

3.3 Explain with an example, the addressing mode used to access program memory.

3.4 Explain with an example, the addressing mode used to access internal data memory.

3.5 Define
    (a) Instruction cycle (b) Machine cycle (c) T-state

3.6 Find the contents of an A and carry flag after the execution of the following instructions. Assume [A] = 25H and [C] = 1 before the execution of instructions.

```
ADD A,#25H
SUBB A,#50H
ADDC A,#0FCH
```

3.7 Find the contents of A after the execution of the following instructions.
(a) ORL A,#75H
(b) XRL A,#0F5H
(c) ANL A,#23H

3.8 With an example, explain the functions of rotate instruction.

3.9 Explain the difference between AJMP, LJMP and SJMP instructions.

3.10 With an example, explain the function of CJNE A, #data, rel instruction.

3.11 With an example, explain how bit level XOR operation can be done in the 8051.

3.12 Explain with an example, how the conditional jump instructions are useful in implementing 'for' loop in a program.

3.13 Explain the difference between MOV, MOVX and MOVC instruction with examples.

3.14 Explain the functions of the following instruction with examples.
(a) DIV A,B (b) CLR A (c) MOVX A,@Ri
(d) XCHD A,@R0 (e) SUBB A,@R0 (f) JNZ 24

3.15 Explain the difference between LCALL and ACALL instruction.

3.16 Explain the instructions that access only the bit wise operand with examples.

3.17 List and explain the instructions related to stack with an example.

3.18 Write a sequence of instructions that clear the AC flag.

3.19 Write an instruction that clears bit 3 of RAM location (23h) without affecting any other bits.

3.20 Calculate the time required to execute the following instructions if the clock frequency is 11.0592 MHz.
(a) CLR A (b) LJMP 3000H (c) MOV 10,R3

3.21 Suppose an 8051 is operating under the control of an external crystal oscillator running at 16 MHz, how much time does it take to execute the following instructions?
(a) MUL AB (b) ANL 10,#27 (c) MOV @R1,10 (d) DAA

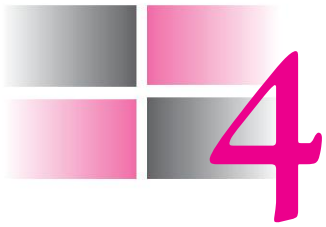
3.22 Find the values of A and B after the execution of the MULAB instruction, if they contain the following values.
(a) 44H and 78H (b) 67H and 6DH

3.23 What will be the value of the carry flag after the execution of each of the following instructions? Assume A contains 6AH and the carry flag is 1 before the execution of each instruction.
(a) ADD A,#45H (b) ADD A,#77H (c) SUBB A,#60H (d) SUBB A,#8FH

3.24 What is the last instruction in the interrupt service routine? What does this instruction do?

3.25 With reasons, explain why SP should pre-increment on a PUSH and post-decrement on a POP instruction in the 8051.

3.26 List the differences and similarities between CALL-RET and PUSH-POP instructions.



8051 ASSEMBLY PROGRAMMING

Learning Objectives

After you have completed this chapter, you should be able to

- Explain the structure of an assembly language program
- Use assembler directives to allocate memory blocks, define constants, etc.
- Write assembly programs to perform simple arithmetic operations
- Write assembly programs to set up time delays and to calculate time delays in a given loop

4.1 ASSEMBLY LANGUAGE PROGRAMS

In the previous chapter, we have discussed architecture, addressing modes and the instruction set of the 8051 microcontroller. In this chapter, we will write assembly language programs. The 8051 assembly program consists of two sections, namely, assembly language program and assembler directives. An *assembly language program* consists of a sequence of statements that tell the microcontroller to perform the desired operations. *Assembler directives* instruct the assembler on how to process subsequent assembly language instructions. Each assembler uses various directives and the assembler directives discussed here correspond to 8051 Assembler.

4.2 ASSEMBLER DIRECTIVES

Assembler directives appear just like instructions in an assembly language program, but they tell the assembler to do something other than creating the machine code for an instruction. In assembly language programming, the assembler directives instruct the assembler to

- process subsequent assembly language instructions
- define program constants
- reserve space for variables

Each assembler uses various directives. The following are the widely used 8051 assembler directives.

ORG (origin)

The ORG directive is used to indicate the starting address. It can be used only when the program counter needs to be changed. The number that comes after ORG can be either in hex or in decimal.

EXAMPLE 4.1

```
        ORG 2000H           ; Set program counter 2000
```

EQU and SET

The EQU and SET directives assign numerical value or register name to the specified symbol name. EQU is used to define a constant without storing information in the memory. The symbol defined with EQU should not be redefined, whereas the SET directive allows redefinition of symbols at a later stage.

EXAMPLE 4.2

```
        Pointer  SET  R1      ; use R1 as pointer
        Counter EQU R3        ; use R3 as counter
        N  EQU 35H
        MOV R3, #N            ; 35h is  stored in R3
```

DB (define byte)

The DB directive is used to define an 8 bit data. DB directive initialises memory with 8 bit values. The numbers can be in decimal, binary, hex or in ASCII formats. For decimal, the 'D' after the decimal number is optional, but for binary and hexadecimal, 'B' and 'H' are required. For ASCII, the number is written in quotation marks (' ').

EXAMPLE 4.3

```
        DATA1: DB  40H        ; hex
        DATA2: DB  01011100B  ; binary
        DATA3: DB  48         ; decimal
        DATA4: DB  'HELLOW'   ; ASCII
```

END

The END directive signals the end of the assembly module. It indicates the end of the program to the assembler. Any text in the assembly file that appears after the END directive is ignored. If the END statement is missing, the assembler will generate an error message.

4.3 ASSEMBLY LANGUAGE PROGRAMS

EXAMPLE 4.4

Write a program to add the values of locations 50H and 51H and store the result in locations 52H and 53H.

ALGORITHM

```
  Step 1: Load the memory contents 50H into A
  Step 2: ADD the memory contents 51H with contents of A
  Step 3: Store the contents of A in 52H
  Step 4: Store the contents of carry flag in 53H.
  The program is as follows
ORG 0000H   ; Set program counter 0000H
MOV A,50H   ; Load the contents of memory location 50H into A
ADD A,51H   ; Add the contents of memory location 51H with contents of A
MOV 52H,A   ; Save the least significant byte of the result in location
              52H
MOV A, #00  ; Load 00H into A
ADDC A, #00 ; Add the immediate data and the contents of carry flag to A
MOV 53H,A   ; Save the most significant byte of the result in location 53
END
```

EXAMPLE 4.5

Write a program to subtract the values of locations 51H from 50H and store the result in location 52H. If the result is positive, store 00H, else store 01H in 53H.

ALGORITHM

```
Step 1: Load the memory contents 50H into A A
Step 2: Clear the carry flag
Step 3: Subtract the memory contents 51H from the contents of A A
Step 4: Store the contents of A A in 52H
Step 5: Check the contents of carry flag, if the contents of carry flag
        is 0 (result is positive) store 00H, else if the contents of
        carry flag is 1 (result is negative) store 01H in 53H.
```

```
ORG 0000H     ; Set program counter 0000H
MOV  A,50H    ; Load the contents of memory location 50H into A
CLR C         ; Clear the borrow flag
SUBB  A,51H   ; Subtract the contents of memory location 51H from
                content of A
MOV 52H,A     ; Store the result  in location 52H
MOV A,#00     ; Load 00H into A
ADDC A,#00    ; Add the immediate data and the contents of carry flag to A
MOV 53H,A     ; Save the most significant byte of the result in location 53
END
```

EXAMPLE 4.6

Write a program to store data FFH into RAM memory locations 50H to 58H using direct addressing mode.

```
ORG 0000H      ; Set program counter 0000H
MOV A, #0FFH   ; Load FFH into A
MOV 50H,A      ; Store contents of A in location 50H
MOV 51H,A      ; Store contents of A in location 51H
MOV 52H,A      ; Store contents of A in location 52H
MOV 53H,A      ; Store contents of A in location 53H
MOV 54H,A      ; Store contents of A in location 54H
MOV 55H,A      ; Store contents of A in location 55H
MOV 56H,A      ; Store contents of A in location 56H
MOV 57H,A      ; Store contents of A in location 57H
MOV 58H,A      ; Store contents of A in location 58H
END
```

EXAMPLE 4.7

Write a program to store data FFH into RAM memory locations 50H to 58H using indirect addressing mode.

```
       ORG 0000H       ; Set program counter 0000H
       MOV A, #0FFH    ; Load FFH into A
       MOV R0, #50H    ; Load pointer, R0=50H
       MOV R5, #08H    ; Load counter, R5=08H
Start: MOV @R0,A       ; Copy contents of A to internal data RAM pointed
                         by R0
       INC R0          ; Increment pointer
       DJNZ R5, start  ; Repeat until R5 is zero
       END
```

EXAMPLE 4.8

Write a program to add two 16 bit numbers stored at locations 51H–52H and 55H–56H and store the result in locations 40H, 41H and 42H. Assume that the least significant byte of data and the result is stored in low address and the most significant byte of data or the result is stored in high address.

```
The program is as follows
ORG 0000H   ; Set program counter 0000H
MOV  A,51H  ; Load the contents of memory location 51H into A
ADD  A,55H  ; Add the contents of memory location 55H with contents of A
MOV 40H,A   ; Save the least significant byte of the result in location
              40H
MOV  A,52H  ; Load the contents of memory location 52H into A
ADDC  A,56H ; Add the contents of 56H and cy flag with contents of A
MOV 41H,A   ; Save the second byte of the result in location 41H
MOV A,#00   ; Load 00H into A
ADDC A,#00  ; Add the immediate data 00H and the contents of carry flag
              to A
MOV 42H,A   ; Save the most significant byte of the result in location 42H
END
```

EXAMPLE 4.9

Write a program to subtract a 16 bit number stored at locations 51H–52H from 55H–56H and store the result in locations 40H and 41H. Assume that the least significant byte of data or the result is stored in low address. If the result is positive, then store 00H, else store 01H in 42H.

```
The program is as follows
ORG 0000H   ; Set program counter 0000H
MOV  A,55H  ; Load the contents of memory location 55H into A
CLR C       ; Clear the borrow flag
SUBB  A,51H ; Sub the contents of memory location 51H from contents of A
MOV 40H,A   ; Save the least significant byte of the result in location 40H
MOV  A,56H  ; Load the contents of memory location 56H into A
SUBB  A,52H ; Sub the content of memory location 52H from the contents of A
MOV 41H,A   ; Save the most significant byte of the result  in location 41H
MOV A, #00  ; Load 00H into A
ADDC A, #00 ; Add the immediate data and the contents of carry flag to A
MOV 42H, A  ; If result is positive, store 00H, else store 01H in location 42H
END
```

EXAMPLE 4.10

Write a program to add two Binary Coded Decimal (BCD) numbers stored at locations 60H and 61H and store the result in BCD at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in low address.

```
The program is as follows
ORG 0000H  ; Set program counter 0000H
MOV  A,60H ; Load the contents of memory location 60H into A
ADD  A,61H ; Add the contents of memory location 61H with contents of A
DA A       ; Decimal adjustment of the sum in A
MOV 52H, A ; Save the least significant byte of the result in location 52H
MOV A,#00  ; Load 00H into A
ADDC A,#00 ; Add the immediate data and the contents of carry flag to A
MOV 53H,A  ; Save the most significant byte of the result in location 53
END
```

EXAMPLE 4.11

Write a program to clear 10 RAM locations starting at RAM address 1000H.

```
The program is as follows
      ORG 0000H          ; Set program counter 0000H
      MOV  DPTR, #1000H  ; Copy address 1000H to DPTR
      CLR A              ; Clear A
      MOV R6, #0AH       ; Load 0AH to R6
again:MOVX @DPTR,A       ; Clear RAM location pointed by DPTR
      INC DPTR           ; Increment DPTR
      DJNZ R6, again     ; Loop until counter R6 = 0
      END
```

EXAMPLE 4.12

Write a program to clear 10 RAM locations starting at RAM address 10H.

```
  The program is as follows
       ORG 0000H      ; Set program counter 0000H
       MOV  R0, #10H  ; Copy address 10H to R0
       CLR A          ; Clear A
       MOV R6,#0AH    ; Load 0AH to R6
```

```
again: MOV @R0,A       ; Clear RAM location pointed by R0
       INC R0          ; Increment R0
       DJNZ R6, again  ; Loop until counter R6 = 0
       END
```

EXAMPLE 4.13

Write a program to compute 1 + 2 + ... + N (say 15) and save the sum at 70H

```
The program is as follows
    ORG 0000H           ; Set program counter 0000H
    N EQU 15
    MOV R0, #00         ; Clear R0
    CLR A               ; Clear A
again: INC R0           ; Increment R0
    ADD A, R0           ; Add the contents of R0 with contents of A
    CJNE R0, #N, again  ; Loop until counter, R0 = N
    MOV 70H,A           ; Save the result in location 70H
    END
```

EXAMPLE 4.14

Write a program to multiply two 8 bit numbers stored at locations 70H and 71H and store the result at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in low address.

```
The program is as follows
ORG 0000H ; Set program counter 0000H
MOV A,70H ; Load the contents of memory location 70H into A
MOV B,71H ; Load the contents of memory location 71H into B
MUL AB    ; Perform multiplication
MOV 52H,A ; Save the least significant byte of the result in location 52H
MOV 53H,B ; Save the most significant byte of the result in location 53
END
```

EXAMPLE 4.15

Write a program to divide contents of 70H from contents of 71H (Assume contents of 70H is greater or equal to contents of 71H). Store the remainder at memory location 53H and the quotient at memory location 52H.

```
The program is as follows
ORG 0000H      ; Set program counter 0000H
MOV A,70H      ; Load the contents of memory location 70H into A
MOV B,71H      ; Load the contents of memory location 71H into B
DIV AB         ; Perform division
MOV 52H,A      ; Save the quotient in location 52H
MOV 53H,B      ; Save the remainder in location 53H
END
```

EXAMPLE 4.16

Ten 8 bit numbers are stored in internal data memory from location 50H. Write a program to increment the data.

SOLUTION

Assume that ten 8 bit numbers are stored in internal data memory from location 50H, hence R0 or R1 must be used as a pointer.

```
The program is as follows
       ORG 0000H       ; Set program counter 0000H
       MOV R0,#50H     ; Load pointer, R0=50H
       MOV R3,#0AH     ; Load counter, R3=0AH
Loop1: INC @R0         ; Increment contents of internal data RAM pointed
                         by R0
       INC R0          ; Increment pointer
       DJNZ R3, loop1  ; Repeat until R3 is zero
      END
```

EXAMPLE 4.17

Write a program to store four 8 bit numbers in internal data RAM. Add these numbers and store the result in 55H and 56H. Assume that the least significant byte of the result is stored in low address.

```
The program is as follows
     ORG 0000H     ; Set program counter 0000H
     MOV 41H,#55H     ;     Store the first number in location 41H
     MOV 42H,#76H ; Store the second number in location 42H
     MOV 43H,#1AH ; Store the third number in location 43H
     MOV 44H,#9FH ; Store the fourth number in location 44H
     MOV R0,#41H  ; Store the first number  address 41H in R0
     MOV R5,#04H  ; Store the number 04H in R5
     CLR C        ; Clear the carry flag
     MOV 56H,#00H ; Store the number 00H in location 56H
```

```
      MOV 55H,#00H ; Store the number 00H in location 55
Loop: MOV A,55H    ; Store the contents of location 55H in A
      ADD A,@R0    ; Add contents of memory with the contents of A
      MOV 55H,A    ; Store the contents of A in location 55H
      MOV A,#00H   ; Store the number 00H in A
      ADDC A,56H   ; Add contents of 56H and the contents of carry flag
                     to A
      MOV 56H,A    ; Store the contents of A in location 56H
      INC R0       ; Increment contents of register R0
      DJNZ R5,Loop ; Decrement R5, if it is not zero, branch to loop
      END
```

EXAMPLE 4.18

Write a program to find the average of five 8 bit numbers. Store the result in 55H. (Assume that after adding five 8 bit numbers, the result is 8 bit only)

```
The program is as follows
      ORG 0000H    ; Set program counter 0000H
      MOV 40H,#05H ; Store the first number in location 40H
      MOV 41H,#55H ; Store the second number in location 41H
      MOV 42H,#06H ; Store the third number in location 42H
      MOV 43H,#1AH ; Store the fourth number in location 43H
      MOV 44H,#09H ; Store the fifth number in location 44H
      MOV R0,#40H  ; Store the first number address 40H in R0
      MOV R5,#05H  ; Store the number 05H in R5
      MOV B,R5     ; Store the number 05H in B
      CLR A        ; Clear the A
Loop: ADD A,@R0    ; Add contents of memory with the contents of A
      INC R0       ; Increment contents of register R0
      DJNZ R5,Loop ; Decrement R5, if it is not zero, branch to loop
      DIV AB       ; Divide the result by 5 (A)/(B)
      MOV 55H,A    ; Save the quotient  in location 55H
      END
```

EXAMPLE 4.19

Write a program to find the cube of an 8 bit number.

```
The program is as follows
ORG 0000H    ;  Set program counter 0000H
MOV R1,#N    ;  Load number to R1
MOV A,R1     ;  Load number to A
MOV B,R1     ;  Load number to B
```

```
MUL AB       ;  Square is computed
MOV R2,B     ;  Copy Contents of B(MSB of square) to R2
MOV B,R1     ;  Load number to B
MUL AB       ;  Multiply LSB of square with B
MOV 50,A     ;  Store result in 51H and 50H
MOV 51,B
MOV A,R2     ;  Copy MSB of square to A
MOV B,R1     ;  Copy number to B
MUL AB       ;  Multiply MSB of square with B
ADD A,51H    ;  Add LSB of result to location 51H
MOV 51H,A    ;  Store result in 52H and 51H
MOV 52H,B
MOV A,#00H   ;  Clear A
ADDC A,52H   ;  Add contents of CF to contents of 52H
MOV 52H,A    ;  Cube is stored at locations 52, 51 and 50
END
```

EXAMPLE 4.20

Write a program to multiply two 16 bit unsigned numbers and store the result at 60H–63H. Numbers are stored at 50H–51H and 52H–53H respectively. Assume that the least significant byte of data is stored in low address.

The 8051 provides only one multiplication instruction—MUL AB that multiplies the unsigned 8 bit integers of registers A and B. As shown in Fig. 4.1, to multiply two 16 bit unsigned numbers, the multiplier and the multiplicand must be broken down into 8 bit numbers as follows.

M = MH ML and N = NH NL

where MH, ML, NH, and NL are the upper and lower 8 bit of M and N respectively. Four 8 bit multiplications must be performed and then partial products are added together.

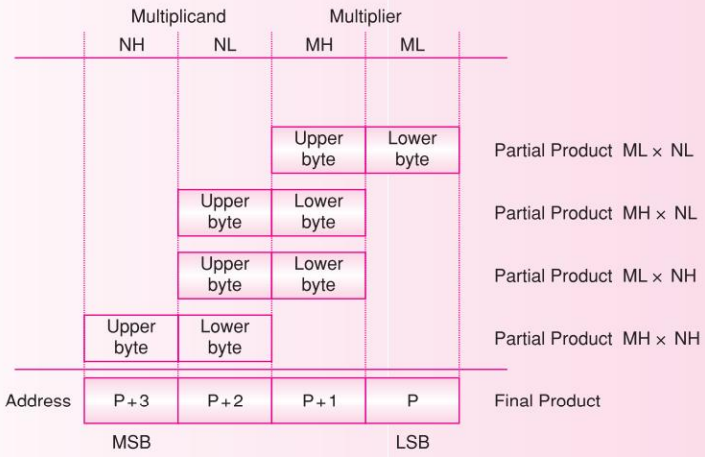


Figure 4.1 *16 bit by 16 bit multiplication*

```
The program is as follows
ORG 0000H      ; Set program counter 0000H
MOV A,50H      ; Place LSB of Multiplier in A
MOV B,52H      ; Place LSB of Multiplicand in B
MUL AB         ; Compute product
MOV 60H,A      ; Store LSB of the result
MOV 61H,B      ; Store MSB of the result
MOV A,51H      ; Place MSB of Multiplier in A
MOV B,53H      ; Place MSB of Multiplicand in B
MUL AB         ; Compute product
MOV 62H,A      ; Store LSB of the result
MOV 63H,B      ; Store MSB of the result
MOV A,51H      ; Place MSB of Multiplier in A
MOV B,52H      ; Place LSB of Multiplicand in B
MUL AB         ; Compute product
ADD A,61H      ; Add the result to P+1
MOV 61H,A      ; Store LSB of the result
MOV A,B
ADDC A,62H     ; Add the result to P+2
MOV 62H,A
MOV A,63H
ADDC A,#0H
MOV 63H,A      ; Store MSB of the result in P+3
MOV A,50H      ; Place LSB of Multiplier in A
MOV B,53H      ; Place MSB of Multiplicand in B
MUL AB         ; Compute product
ADD A,61H      ; Add the result to P+1
MOV 61H,A      ; Store LSB of result in P+1
MOV A,B
ADDC A,62H     ; Add the result to P+2
MOV 62H,A      ; Store the result in P+2
MOV A,63H
ADDC A,#0H     ; Add the result to P+3
MOV 63H,A      ; Store the result in P+3
END
```

EXAMPLE 4.21

Write a program to exchange the lower nibble of data present in external memory 6000H and 6001H.

```
The program is as follows
ORG 0000H         ; Set program counter 0000H
MOV DPTR, #6000H  ; Copy address 6000H to DPTR
MOVX A, @DPTR     ; Copy contents of 6000H to A
MOV R0,#45H       ; Load pointer, R0=45H
MOV @R0,A         ; Copy contents of A to internal data RAM pointed by R0
INC DPL           ; Increment pointer
MOVX A,@DPTR      ; Copy contents of 6001H to A
XCHD A,@R0        ; Exchange lower nibble of A with RAM pointed by R0
MOVX @DPTR,A      ; Copy contents of A to 6001H
DEC DPL           ; Decrement pointer
MOV A,@R0         ; Copy contents of internal data RAM pointed by R0 to A
MOVX @DPTR,A      ; Copy contents of A to data RAM pointed by DPTR
END
```

EXAMPLE 4.22

Write a program to count the number of 1's and 0's of 8 bit data stored in location 6000H.

```
The program is as follows
       ORG 0000H         ; Set program counter 0000H
       MOV DPTR, #6000H  ; Copy address 6000H to DPTR
       MOVX A, @DPTR     ; Copy number to A
       MOV R0,#08        ; Copy 08 in R0
       MOV R2,#00        ; Copy 00 in R2
       MOV R3,#00        ; Copy 00 in R3
       CLR C             ; Clear carry flag
 BACK: RLC A             ; Rotate contents of A through carry flag
       JC NEXT           ; If CF = 1, branch to next
       INC R2            ; If CF = 0, increment R2
       AJMP NEXT2
NEXT:  INC R3            ; If CF = 1, increment R3
NEXT2: DJNZ R0,BACK      ; Repeat until R0 is zero
       END
```

EXAMPLE 4.23

An 8 bit code word is stored in location 1000H of external data memory. Code word is valid, if three MSBs are zero and it contains two ones in the remaining five bits. If code word is valid, store FF, else store 00 in 1001H.

SOLUTION

Code word is valid if three MSBs are zero and it contains two ones in the remaining five bits, for example—00010010, then code word is valid.

The program is as follows

```
      ORG 0000H           ; Set program counter 0000H
      MOV DPTR, #1000H    ; Copy address 1000H to DPTR
      MOVX A, @DPTR       ; Copy number to A
      MOV R1,A            ; Copy number to R1
      INC DPTR
      MOV R2,#05          ; Copy 05 in R2
      MOV R3,#00          ; Copy 00 in R3
      MOV R4,#00          ; Copy 00 in R4
      CLR C               ; Clear carry flag
      ANL A,#0EOH         ; Mask lower 5 bit
      CJNE A,#00,LOOP1    ; If first condition fails, branch to LOOP1
      MOV A,R1            ; Check the second condition
BACK: RRC A
      JNC LOOP2
      INC R3
LOOP2:DNZ R2,BACK
      MOV A,R3
      CJNE A, #02H, LOOP1 ; If second condition fails, branch to LOOP1
                            and store 00H
      DEC R4              ; Code word is valid, to store FF, decrement R4
LOOP1:MOV A,R4
      MOVX @DPTR,A
      END
```

EXAMPLE 4.24

Ten 8 bit numbers are stored in external data memory from location 5000H. Write a program to transfer a block of data to location 6000H.

SOLUTION

Assume ten 8 bit numbers are stored in external data memory from location 5000H, hence DPTR must be used as a pointer.

```
The program is as follows
        ORG 0000H     ; Set program counter 0000H
        MOV R4,#00H   ; Load 00H to R4
        MOV R5,#50H   ; Load 50H to R4
        MOV R2,#00H   ; Load 00H to R2
        MOV R3,#60H   ; Load 60H to R4
        MOV R6,#0AH   ; Load counter, R6=0AH
Loop1:  MOV DPL,R4    ; Copy contents of R5 and R4 to DPTR
        MOV DPH,R5
        MOVX A,@DPTR  ; Copy contents of data RAM pointed by DPTR to A
        MOV DPL,R2    ; Copy contents of R3 and R2 to DPTR
        MOV DPH,R3
        MOVX @DPTR,A  ; Copy contents of A to data RAM pointed by
DPTR
        INC R4        ; Increment R4
        INC R2        ; Increment R2
        DJNZ R6,Loop1 ; Repeat until R6 is zero
        END
```

EXAMPLE 4.25

Write a program to shift a 24 bit number stored at 57H–55H to the left logically four places. Assume that the least significant byte of data is stored in lower address.

```
The program is as follows
        ORG 0000H       ;  Set program counter 0000H
        MOV R1,#04      ;  Set up loop count to 4
again:  MOV A,55H       ;  Place the least significant byte of data in A
        CLR C           ;  Clear the carry flag
        RLC A           ;  Rotate contents of A (55H) left by one position
                           through carry
        MOV 55H,A
        MOV A,56H
        RLC A           ;  Rotate contents of A (56H) left by one position
                           through carry
        MOV 56H,A
        MOV A,57H
        RLC A           ;  Rotate contents of A   (57H) left by one
                           position through carry
        MOV 57H,A
        DJNZ R1, again  ;  Repeat until R1 is zero
        END
```

EXAMPLE 4.26

Write a program to find the smallest number of an array of N 8 bit unsigned numbers (N is an 8 bit number). The starting address of the array is at 2000H and stores the result in 2500H.

SOLUTION

N 8 bit numbers are stored in external data memory from location 2000H, hence DPTR must be used as a pointer.

ALGORITHM

```
 Step 1: Initialise data in memory
 Step 2: Initialise R4 with N-1
 Step 3: Initialise DPTR with address of the array
 Step 4: Load the element pointed by DPTR into A and increment DPTR
 Step 5: Load the next element to register TEMP
 Step 6: Compare contents of A with contents of TEMP
 Step 7: If A is greater, then copy the next elements into A
 Step 8: Increment the pointer and decrement R5 register
 Step 9: Check if all the elements have been compared in the array
         (i.e. R5 = 0?)
Step 10: No, go to step 5
Step 11: If R4 is zero, terminate the program
```

The program is as follows

```
       ORG 0000H           ; Set program counter 0000H
       TEMP EQU 40H
          N EQU 04H        ; Array count
       MOV R4,#N-1         ; Load N-1 to R4
       MOV DPTR,#2000H     ; Store the starting address of the array in DPTR
       MOVX A,@DPTR        ; Copy first number to A
LOOP1: MOV R1,A            ; Copy contents of A to R1
again: INC DPTR            ; Increment DPTR
       MOVX A,@DPTR        ; Get the next number to A
       MOV TEMP,A          ; Copy the next number to TEMP
       MOV A,R1
       CJNE A,TEMP, LOOP2  ; (A)≠(TEMP) branch to LOOP2
       SJMP LOOP3          ; (A)=(TEMP) branch to LOOP3
LOOP2: JC LOOP3            ; (A)<(TEMP) branch to LOOP3
       MOV A,TEMP          ; (A)>(TEMP) copy contents of TEMP to A
LOOP3: DJNZ R4,LOOP1       ; Repeat until R4 is zero
       END
```

EXAMPLE 4.27

N 8 bit numbers are stored in external data memory. Write a program to arrange the numbers in ascending order.

ALGORITHM

```
Step 1: Initialise data in memory
Step 2: Initialise R4 with N-1
Step 3: Initialise DPTR with address of the array
Step 4: Copy the contents of R4 to R5 register
Step 5: Load the element pointed by DPTR into A and the next element
        to register TEMP
Step 6: Compare contents of A with contents of TEMP
Step 7: If A is greater, then exchange the two elements in the
        memory
Step 8: Increment the pointer and decrement R5 register
Step 9: Check if all the elements have been compared in the array
        (i.e. R5 = 0?)
Step 10: No, go to step 5
Step 11: Yes, decrement R4; if it is not zero, go to step 3
Step 12: If R4 is zero, terminate the program
```

The program is as follows

```
       ORG 0000H           ; Set program counter 0000H
       TEMP  EQU  40H
       N     EQU  04H      ; Array count
       MOV R4,#N-1         ; Load N-1 to R4
LOOPS: MOV DPTR,#2000H     ; Store the starting address of the array in DPTR
       MOV R5,R4
LOOP1: MOVX A,@DPTR        ; Copy contents of memory pointed by DPTR to A
       MOV R1,A            ; Copy contents of A to R1
       INC DPTR            ; Increment DPTR
       MOVX A,@DPTR        ; Get the next number to A
       MOV TEMP,A          ; Copy the next number to TEMP
       MOV A,R1
       CJNE A,TEMP, Loop2  ; (A) ≠ (TEMP) branch to LOOP2
       SJMP LOOP3          ; (A) = (TEMP) branch to LOOP3
LOOP2: JC LOOP3            ; (A) < (TEMP) branch to LOOP3
       MOVX @DPTR,A        ; (A) > (TEMP) exchange contents of two locations
      MOV A,TEMP
       DEC DPL
```

```
        MOVX @DPTR,A
        INC DPL
LOOP3:  DJNZ R5,LOOP1  ;  Repeat until R5 is zero
        DJNZ R4,LOOPS  ;  Repeat until R4 is zero
        END
```

EXAMPLE 4.28

N 8 bit numbers are stored in internal data memory. Write a program to arrange the numbers in descending order.

The program is as follows

```
        ORG 0000H           ; Set program counter 0000H
        TEMP  EQU  40H
        N     EQU  04H      ; Array count
        MOV R4,#N-1         ; Load N-1 to R4
LOOPS:  MOV R0,#41H         ; Store the starting address of the array in R0
        MOV R5,R4
LOOP1:  MOV A,@R0           ; Copy contents of memory pointed by R0 to A
        MOV R1,A            ; Copy contents of A to R1
        INC R0              ; Increment R0
        MOV A,@R0           ; Get the next number to A
        MOV TEMP,A          ; Copy the next number to TEMP
        MOV A,R1
        CJNE A,TEMP, LOOP2 ; (A) ≠ (TEMP) branch to LOOP2
        SJMP LOOP3          ; (A) = (TEMP) branch to LOOP3
LOOP2:  JNC LOOP3           ; (A) > (TEMP) branch to LOOP3
        MOV @R0,A           ; (A) < (TEMP) exchange contents of two locations
        MOV A,TEMP
        DEC R0
        MOV @R0,A
        INC R0
LOOP3:  DJNZ R5,LOOP1       ; Repeat until R5 is zero
        DJNZ R4,LOOPS       ; Repeat until R4 is zero
        END
```

EXAMPLE 4.29

Write a program to convert a binary number stored in location 6000H to BCD and store the result in 51H–50H.

```
The program is as follows
       ORG 0000H          ;  Set program counter 0000H
       MOV DPTR, #6000H   ;  Store the address of the memory in DPTR
       MOVX A, @DPTR      ;  Copy contents of memory pointed by DPTR to A
       MOV R0,#00H        ;  Clear R0
       CJNE A,#09, NEXT   ;  (A) ≠ 09 branch to NEXT
       AJMP LAST          ;  (A) = 09 branch to LAST
 NEXT: JC LAST            ;  (A) < 09 branch to LAST
       CJNE A,#63, NEXT1  ;  (A) ≠ 63H branch to NEXT1
       AJMP NEXT2         ;  (A) = 63H branch to NEXT2
NEXT1: JC NEXT2           ;  (A) < 63H branch to NEXT2
       MOV B,#64H         ;  (A) > 63H divide contents of A by 100 (64H)
       DIV AB             ;  divide contents of A by contents of B
       MOV R0,A
       MOV A,B
NEXT2: CJNE A,#09H,NEXT3  ;  (A) ≠ 09 branch to NEXT3
       AJMP LAST          ;  (A) = 09 branch to LAST
NEXT3: JC LAST            ;  (A) < 09 branch to LAST
       MOV B,#0AH         ;  (A) > 09 divide contents of A by 10 (0A H)
       DIV AB             ;  After division, Q is in A and R is in B
       SWAP A             ;  Get Q in upper nibble of A
       ORL A,B            ;  Get R in lower nibble of A
 LAST: MOV 51H,R0         ;  Store result in 51H and 50H.
       MOV 50H,A
       END
```

EXAMPLE 4.30

Write a program to convert BCD number stored in location 6000H to binary and store the result in 5100H.

```
The program is as follows
ORG 0000H          ;  Set program counter 0000H
MOV DPTR, #6000H   ;  Store the address of the memory in DPTR
MOVX A, @DPTR      ;  Copy contents of memory pointed by DPTR to A
ANL A,#0FH         ;  Mask upper nibble of A
MOV R1,A           ;  Copy contents of A in R1
MOVX A, @DPTR      ;  Copy contents of memory pointed by DPTR to A
```

```
ANL A,#F0H          ;  Mask lower nibble of A
SWAP A              ;  Exchange upper and lower nibble of A
MOV B,A
MOV A,#0AH
MUL AB              ;  Multiply upper nibble with 10 (0AH)
ADD A,R1            ;  Add lower nibble of A with result
MOV DPTR, #5100H
MOVX @DPTR, A       ;  Store result in 5100H
END
```

EXAMPLE 4.31

Write a program to convert an ASCII number stored in location 7000H of external data RAM to hex. Store the result in 5000H.

```
The program is as follows
       ORG 0000H           ; Set program counter 0000H
       MOV DPTR, #7000H    ; Store the address of the memory in DPTR
       MOVX A, @DPTR       ; Copy contents of memory pointed by DPTR to A
       CJNE A,#39H, NEXT1  ; (A) ≠ 39H branch to NEXT1
       AJMP NEXT2          ; (A) = 39H branch to NEXT2
NEXT1: JC NEXT2            ; (A) < 39H branch to NEXT2
       CLR C
       SUBB A,#37H         ; (A) > 39H Sub 37H
       AJMP NEXT5          ; Branch to NEXT5
NEXT2: CLR C
       SUBB A,#30H         ; (A) < 39H Sub 30H
NEXT5: MOV DPTR, #5000H
       MOVX @DPTR, A       ; Store the result in 5000H
       END
```

EXAMPLE 4.32

Write a program to convert hex number stored in location 7000H of external data RAM to ASCII. Store the result in 5000H.

```
The program is as follows
       ORG 0000H          ; Set program counter 0000H
       MOV DPTR, #7000H   ; Store the address of the memory in DPTR
       MOVX A, @DPTR      ; Copy contents of memory pointed by DPTR to A
       CJNE A,#9H, NEXT1  ; (A) ≠ 9H branch to NEXT1
```

```
        AJMP NEXT2          ; (A) = 9H branch to NEXT2
NEXT1:  JC NEXT2            ; (A) < 9H branch to NEXT2
        ADD A,#07H          ; (A) > 9H add 37H
NEXT2:  ADD A,#30H          ; (A) < 9H add 30 H
        MOV DPTR, #5000H
NEXT5:  MOVX @DPTR,A        ; Store the result in 5000H
        END
```

EXAMPLE 4.33

Write a program to convert an 8 bit BCD number stored in location 7000H of external data RAM to ASCII. Store the result in locations 5000H and 5001H.

```
The program is as follows
ORG 0000H           ;  Set program counter 0000H
MOV DPTR, #7000H    ;  Copy address 7000H to DPTR
MOVX A, @DPTR       ;  Copy number to A
MOV DPTR, #5000H    ;  Copy address 5000H to DPTR
MOV R2,A            ;  Copy BCD data in R2
ANL A,#0FH          ;  Mask the upper nibble
ORL A,#30H          ;  Convert the lower nibble to ASCII
MOVX @DPTR,A        ;  Store the result in  5000H
MOV A,R2            ;  Get the original data
ANL A,#0F0H         ;  Mask the lower nibble
SWAP A
ORL A,#30H          ;  Convert the upper nibble to ASCII
INC DPTR            ;  Increment DPTR
MOVX @DPTR,A        ;  Store the result in  5001H
END
```

EXAMPLE 4.34

Write a program to convert an ASCII number stored in locations 7000H and 7001H of external data RAM to packed BCD. Store the result in location 5000H.

```
The program is as follows
ORG 0000H           ; Set program counter 0000H
MOV DPTR, #7000H    ; Copy address 7000H to DPTR
MOVX A, @DPTR       ; Copy contents of 7000H to A
CLR C               ; Clear carry flag
```

```
SUBB A,#30H          ; Sub 30H, convert number to BCD
MOV R2,A
INC DPTR
MOVX A, @DPTR        ; Copy contents of 7001H to A
CLR C
SUBB A,#30H          ; Sub 30H, convert number to BCD
SWAP A
OR A,R2              ; Convert result to packed BCD
MOV DPTR, #5000H     ; Copy address 5000H to DPTR
MOVX @DPTR,A         ; Store the result in  5000H
END
```

EXAMPLE 4.35

Write a program to convert binary code stored in location 6000H to gray code. Store the result in location 5000H.

ALGORITHM

1. Copy data from memory to A and R0.
2. Shift contents of A right by one position and store 0 in MSB.
3. EX-OR contents of A with R0 and store the result in memory.

The program is as follows

```
ORG 0000H          ;  Set program counter 0000H
MOV DPTR, #6000H   ;  Copy address 6000H to DPTR
MOVX A, @DPTR      ;  Copy number to A
MOV DPTR, #5000H   ;  Copy address 5000H to DPTR
MOV R0,A           ;  Copy data in R0
CLR C              ;  Clear carry flag
RRC A              ;  Rotate contents of A right by one position
XRL A,R0           ;  ER-OR contents of A with R0
MOVX @DPTR,A       ;  Store result in location 5000H
END
```

EXAMPLE 4.36

Two 8 bit numbers are stored in locations 1000H and 1001H of external data memory. Write a program to find the Greatest Common Divisor (GCD) of two numbers and store the result in location 2000H.

ALGORITHM

```
Step 1 : Initialise external data memory with data and DPTR with address
Step 2 : Load A and TEMP with the operands
Step 3 : Are the two operands equal? If yes, go to step 9
Step 4 : Is (A) greater than (TEMP)? If yes, go to step 6
Step 5 : Exchange (A) with (TEMP) such that A contains the bigger number
Step 6 : Perform division operation (contents of A with contents of TEMP)
Step 7 : If the remainder is zero, go to step 9
Step 8 : Move the remainder into A and go to step 4
Step 9 : Save the contents of TEMP in memory and terminate the program
```

The program is as follows

```
       ORG 0000H          ; Set program counter 0000H
       TEMP EQU 70H
       TEMP1 EQU 71H
       MOV DPTR, #1000H   ; Copy address 1000H to DPTR
       MOVX A, @DPTR      ; Copy First number to A
       MOV TEMP,A         ; Copy First number to temp
       INC DPTR
       MOVX A, @DPTR      ; Copy Second number to A
LOOPS: CJNE A,TEMP,LOOP1  ; (A) ≠ (TEMP) branch to LOOP1
       AJMP LOOP2         ; (A) = (TEMP) branch to LOOP2
LOOP1: JNC LOOP3          ; (A) > (TEMP) branch to LOOP3
       MOV TEMP1,A        ; (A) < (TEMP) exchange (A) with (TEMP)
       MOV A,TEMP
       MOV TEMP,TEMP1
LOOP3: MOV B,TEMP
       DIV AB             ; Divide (A) by (TEMP)
       MOV A,B            ; Move remainder to A
       CJNE A,#00,LOOPS   ; (A) ≠ 00 branch to LOOPS
LOOP2: MOV A,TEMP
       MOV DPTR,#2000H
       MOVX @DPTR,A       ; Store the result in 2000H
       END
```

4.4 TIME DELAY CALCULATIONS

The 8051 instructions execution time is calculated either by the number of machine cycles or the oscillator periods. A machine cycle depends on the frequency of the oscillator periods connected to the 8051 and is equal to 12 oscillator periods. The frequency of the 8051 family can vary from 1 MHz to 30 MHz. The number of machine cycles for 8051 instructions can be found from instruction set summary (Section 3.7). By knowing the execution time for each instruction, delay programs to generate different delays can be created.

EXAMPLE 4.37

Write a program to create a delay of 1 ms. Assume that the oscillator frequency is 12 MHz.

SOLUTION

```
Oscillator period = 1/12 MHz
1 Machine cycle = (1/12 MHz) × 12 = 1μs
Execution time for the following instruction sequence is 8 × 1 μs
= 8 μs
Start: PUSH ACC          2 Machine cycles
       POP ACC           2 Machine cycles
       NOP               1 Machine cycle
       NOP               1 Machine cycle
       DJNZ R1,Start     2 Machine cycles
To create a delay of 1 ms, the program must be executed 125 times
     = 1 ms / 8 μs = 1000 μs /8 μs = 125
Program to create 1 ms is as follows
     MOV R1,#125   2 Machine cycles
Start:             PUSH ACC     2 Machine cycles
     POP ACC       2 Machine cycles
     NOP           1 Machine cycle
     NOP           1 Machine cycle
     DJNZ R1,Start 2 Machine cycles
Actual time is 2 μs more than 1 ms. The instruction MOV R1,#125
takes 2 Machine cycles to execute. This instruction is the overhead
for setting up the delay loop.
```

EXAMPLE 4.38

Write a program to create a delay of 0.1 second or 100 ms. Assume that the oscillator frequency is 12 MHz.

SOLUTION

```
Oscillator period = 1/12 MHz
1 Machine cycle = (1/12 MHz) × 12 = 1 μs
By repeating the previous program 100 times, a time delay of 0.1
second can be created.
     = (100 ms / 1 ms) = 100
Program to create 0.1 second is as follows
     MOV R2,#100   2 Machine cycles
```

```
Start1: MOV R1,#125        2 Machine cycles
Start:  PUSH ACC           2 Machine cycles
        POP ACC            2 Machine cycles
        NOP                1 Machine cycle
        NOP                1 Machine cycle
        DNZ R1,Start       2 Machine cycles
        DNZ R2, Start1     2 Machine cycles
```

Actual time is 202 µs more than 0.1 second. The instruction MOV R1,#125 and MOV R2,#100 take 2 Machine cycles to execute. Therefore the total overhead is 202 µs.

EXAMPLE 4.39

Write a program to create a delay of 1 second or 1000 ms. Assume that the oscillator frequency is 12 MHz.

SOLUTION

Oscillator period = 1/12 MHz

1 Machine cycle = (1/12 MHz) × 12 = 1 µs

By repeating the previous program loop 10 times, a time delay of 1 second can be created i.e. (1000 ms/100 ms) = 10

Program to create 1 second delay is as follows

```
        MOV R3,#10         2 Machine cycles
Start2: MOV R2,#100        2 Machine cycles
Start1: MOV R1,#125        2 Machine cycles
Start:  PUSH ACC           2 Machine cycles
        POP ACC            2 Machine cycles
        NOP                1 Machine cycle
        NOP                1 Machine cycle
        DNZ R1,Start       2 Machine cycles
        DNZ R2, Start1     2 Machine cycles
        DNZ R3, Start2     2 Machine cycles
```

Actual time is (2 + 20 + 2500) µs = 2522 µs more than 1 second. The instruction MOV R1,#125, MOV R2,#100 and MOV R3,#10 take 2 Machine cycles to execute. Therefore the total overhead is 2522 µs.

EXAMPLE 4.40

Find the time delay for the following subroutine, assuming a crystal frequency of 11.0592 MHz.

```
                    MOV R2,#25      2 Machine cycles
            Start:  NOP             1 Machine cycle
                    NOP             1 Machine cycle
                    NOP             1 Machine cycle
                    NOP             1 Machine cycle
                    DNZ R2, Start   2 Machine cycles
                    RET             1 Machine cycle
```

SOLUTION

Oscillator period = 1/11.0592 MHz

1 Machine cycle = (1/11.0592MHz) × 12 = 1.085 μs

Time delay inside the start loop is [25 (1 + 1 + 1 + 1 + 2)] × 1.085 μs
= 150 × 1.085 μs = 162.75 μs.

Adding 3 Machine cycles of MOV R2, #25 and RET instructions outside the loop = 3 × 1.085 μs = 3.255 μs

Total Delay = 162.75 + 3.255 μs = 166.005 μs.

Note: For 8052 microcontroller, the calculation of time and delay programs remain same as discussed.

CHAPTER SUMMARY

Programming environment of the 8051 has been elucidated in this chapter. Assembler directives are explained with examples. A proper study of this chapter will enable you to write simple programs such as addition, subtraction, multiplication, arranging numbers in ascending/descending order, etc. All instructions take some amount of time to execute. Writing delay programs and calculation of time taken by the processor to execute the programs have been discussed with examples.

EXERCISES

REVIEW QUESTIONS

1. What are assembler directives? Explain any two directives.
2. Explain the significance of the following assembler directives in assembly language programming.
   (a) ORG (b) EQU (c) DB
3. Write a program to subtract a 16 bit number stored in locations Y and Y + 1 from a 16 bit number stored in locations X and X + 1. Store the result in locations Z and Z + 1.

4. Write a program to store the following numbers in bank 0.
   98H, 10101100B, 22D, A4H, 11100011B, 99D, FBH and 60D
5. Write a program to copy the contents of successive memory locations (30H to 37H) to bank 0, using direct and indirect addressing mode.
6. Write a program to find the number of 1s in an 8 bit number stored at memory location 6000H.
7. Write a program to find the sum of two 32 bit numbers. The bytes of the first number are stored in 40H–43H and the bytes of the second number are stored in 50H–53H. Store the result in memory locations starting from 60H.
8. Write a program to subtract the number in location 30H from the number in location 31H by using 2's complement subtraction method. Store the result in location 33H. Store 00H in location 34H, if the result is positive, else store 01, if the result is negative.
9. Write a program to find number of 1s and 0s in an 8 bit number stored at memory location 50H.
10. Write a program to find whether an 8 bit number at location 40H is 2 out of 5 code. The number is valid, if the three MSBs are 1 and in the remaining 5 bit, at least 2 bit are 1. If code is valid, store FFH at 41H, else store 00H.
11. Can you call a subroutine using JMP? Explain with an example and corresponding assembly language program.
12. Write a program to store the numbers 00H to 05H at locations 30H to 35H using PUSH instructions. Store these values at locations 40H to 45H using POP instructions.
13. Write a program to find the sum of N BCD numbers stored at successive locations starting from 41H. Store the result in BCD in locations 51H and 50H.
14. Write a program to multiply two 32 bit numbers stored at internal data memory 43H–40H and 48H–45H. Store the result at 57H–50H.
15. Write a program to swap the last element of an array with the first element, the next to last element with the second element, etc. Assume that the array has ten 8 bit numbers and the array starts at 40H of internal data memory.
16. An array contains signed twenty 8 bit numbers. Write a program to compute the sum of the positive numbers of an array and store the result in external data memory.
17. An array contains ten 8 bit numbers that are stored starting from location 1000H of external data memory. Write a program to find the average of the square of each element.
18. Write a program for addition of two N byte BCD numbers.
19. Write a program to compute $\sum_{I=1}^{N} X_I Y_I$, where X_I and Y_I are 8 bits numbers and N = 8.
20. Two 8 bit numbers are stored in external data memory. Write a program to find LCM of these two numbers.
21. An array contains twenty 8 bit numbers that are stored starting from location 1000H of external data memory. Write a program to add all the even numbers in an array and store the result in 2001H and 2000H.
22. Write a program to find the number of elements that are divisible by 4 from an array of twenty 8 bit numbers. The array starts from 3000H of external data memory.
23. Write a program to compute HCF of two 16 bit numbers.
24. Write a subroutine to implement insertion sort algorithm to arrange the numbers in ascending or descending order.

25. Write a program to find the square root of an 8 bit number.
26. Write a program to check whether the number stored in location 45H is a palindrome. Store FFH in location 46H, if it is a palindrome, else store 00H.
27. Write a program to create a delay of 200 msec. Assume that the oscillator frequency is 11.0592 MHz.
28. Write a program to create a delay of 50 ms. Assume that the oscillator frequency is 6 MHz.
29. Write a subroutine to create a delay of 20 ms. Assume that the oscillator frequency is 11.0592 MHz. Using subroutine, write a program to create a delay of 0.1 second.
30. Write a subroutine to create a delay of 50 ms. Assume that the oscillator frequency is 12 MHz. Using subroutine, write a program to create a delay of 0.1 second.

5

SOFTWARE DEVELOPMENT TOOLS FOR 8051

Learning Objectives

After you have completed this chapter, you should be able to

- Explain Integrated Development Environment
- Work with A51 assembler and S51 simulator (DOS Version)
- Work with C compiler and simulator (SIDE 51)
- Work with C compiler and simulator (Keil µVision)

5.1 INTEGRATED DEVELOPMENT ENVIRONMENT

Microcontrollers are programmed using an *integrated development environment*. In the process of code development for microcontrollers, we make use of assembly language and high-level language. Programs are developed and edited using personal computers. Computers are built with microprocessors (e.g., Pentium IV). These processors are used to develop codes for the targeted microcontroller. The processor used to develop codes is called *development processor*. The processor for which codes are developed is called *target processor*. In the process of developing codes for the microcontroller, we make use of

softwares, namely assembler, debugger, simulator and compiler. These softwares enable us to write, edit and debug assembly and C programs. We simulate the functionality of a microcontroller using simulator software on the computer. The simulator enables us to view various registers and memory contents after running the program. Intermediate results in various registers and memory can also be verified using a single step mode. Finally, machine codes are downloaded to the memory of the target processor. The development processor, software and target processor together are called Integrated Development Environment (IDE).

In Circuit Debugger

The developed program is downloaded to the microcontroller program memory (EPROM/FLASH) using serial communication (RS232) and run on the hardware. The software program establishes serial communication between development processor and target processor to download the code. After burning of code, the program is executed on the hardware. In case of errors in the expected output, the program can be modified and reloaded again to the target microcontroller. This is called *In Circuit Debugging*. The hardware setup and connectors used for this purpose is called *In Circuit Debugger (ICD)*.

The program written using high-level language is compiled to the machine language of the microcontroller. The software to edit, debug, simulate and compile is usually available as one package, e.g. compiler SIDE51 from SPJ systems and compiler µVision of Kiel corp. A51 assembler and S51 simulator are DOS-based assembler and simulator respectively. The programs written in C (.c files) or assembly (.asm) are converted to machine language (.hex files) and loaded to the on-chip program memory of the microcontroller. Development software generates .lst, .obj, and .hex files. Software like Flash magic is used to burn the desired hex file to the program memory.

5.2 A51 ASSEMBLER AND S51 SIMULATOR

Earlier versions of software for the code development are based on DOS (e.g., A51 such assembler). The following are the steps to develop the code.

To create an .asm file, compile, load and run

1. Restart the computer in MS-DOS mode. Change the working directory to 8051. Create your file and type in the mnemonics and save the file with .asm extension.
   (e.g., add.asm)
2. To compile the program,
   8051>a51 add.asm
   When the .asm file is compiled, .lst and .obj files are generated by the assembler.
3. To load and run the program,
   8051>s51
   The simulator window appears.
   - Select ‘previous machine’ option.
   - Enter ‘book.bss’ option (this initialises the simulator for loading the .obj file).
   - Select ‘load’ option, Enter the file name with .obj extension (e.g., add.obj).
   - Select the option ‘Run’. This will run the program using simulator software and generates .hex file.
   - When you run the program, contents of various registers of 8051 and memory locations are displayed in the window. Verify the results of your program.
4. To download the .hex codes to the program memory of 8051, change the working directory to b30drv.
   8051>b30drv
   Select options F10, and then 3. Enter file name add.obj.

5.3 SC51 C COMPILER (SIDE 51)

SIDE 51 is a windows-based compiler developed by SPJ Systems. The following are the steps to develop the code.

Starting the IDE

Click on START button and select Programs/SPJ Systems SC51/SIDE51. A window as shown in Fig. 5.1 appears on the screen.

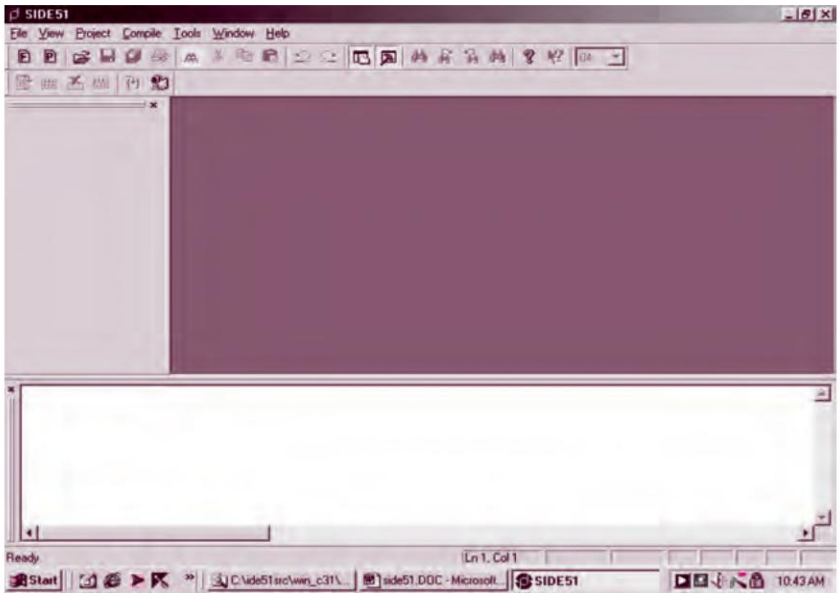


Figure 5.1 *Window to start IDE (Courtesy SPJ Systems)*

Creating a New Project

A project is a file in which SIDE51 stores all information related to an application. For example, it stores the name(s) of 'C' and/or assembler source file(s), memory model to be used and other options for compiler, assembler and linker.

To open an existing project file, select Project/Open Project from the menu.

To create a new project, select Project/New Project from the menu.

Changing Project Settings

To change the project settings (such as adding or removing 'C' and/or assembler source file(s), changing memory model, etc.), select Project/Settings from the menu. The screen displays the status as shown in Fig. 5.2.

There are 3 tabs in the project settings window.

1. Source Files This tab is automatically selected when you open the Project Settings window. When this tab is selected, you can see the list of files that are part of this project. This list is divided into groups of C files, ASM files, OBJ files and LIB files. To remove a file from the project, select the filename by clicking

on it and then press the 'Remove File' button. To add a file to the project, press the 'Add File' button, select desired file and then press 'Open' button. The SIDE51 allows adding up to 16 C files, 16 ASM files, 16 OBJ files and 16 LIB files in a project.

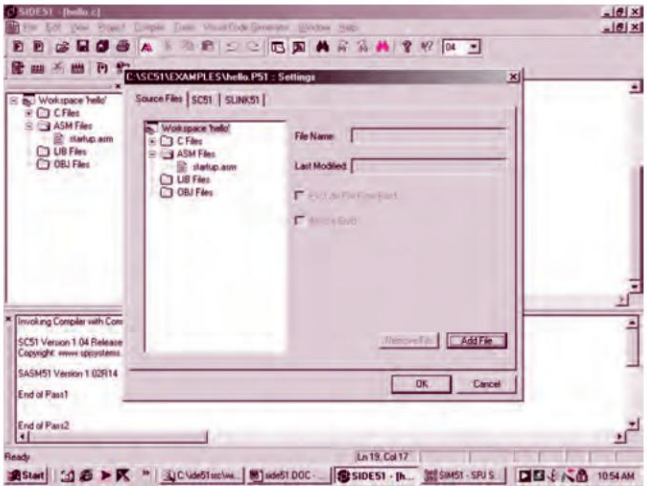


Figure 5.2 *Window to change project settings (Courtesy SPJ Systems)*

2. SC51 This tab lists some options for the SC51 compiler. You can select the 'Target micro-controller' by selecting the desired manufacturer's name from the list, and then selecting the desired 8051 derivative name from the list of micro-controllers. Depending on the processor selected, the amount of internal data memory (128 or 256 bytes) is automatically selected. However, you may change it, if there is a need. You can also enter the crystal frequency used in the target (this information may later be used by the simulator). You may check the 'Generate debug info' option, if you wish to use the simulator. You also have the option to check the 'Include C source lines in generated ASM file'.

3. SLINK51 This tab lists some options for SLINK51 linker. You may enter the start and end address of xdata memory. If you need the ROM image file (.BIN) file, you should check the option 'Generate BIN file'.

Set the necessary options in all the three tabs and then, press 'Ok' button. This will save the changed settings in the project file. To discard the changes made, press 'Cancel' button.

Opening an Existing Project

Select Project/Open Project from the menu. Select the path where you had installed SC51 and select the 'Examples' folder. You will see a list of project files (Example . P51).

Compiling the Program

To compile the program, select Compile/Build or Compile/Re-build All from the menu. Doing this may invoke one or more of these applications: C compiler, Assembler, and Linker. If there are no errors, then Intel HEX format file (.HEX) and optionally .BIN file will be produced. The error and warning messages produced by compiler, assembler, and linker will be displayed in the error window. If there are errors, correct the errors and repeat the process until all errors are removed.

Running the Program

To run (i.e. debug) the program, select Tools/Simulator from the menu. Doing this will invoke the simulator program. The window as shown in Fig. 5.3 appears on the screen.

- To monitor memory locations in the external RAM, select View / External RAM watch.
- To monitor memory locations in the internal RAM, select View / Internal RAM watch.
- To monitor the status of connected peripheral devices, select View / I/O watch from the menu.
- To monitor the internal registers / SFRs of 8051, select View / SFR watch from the menu.
- To monitor 'C' program variables, select View / C variable watch from the menu.

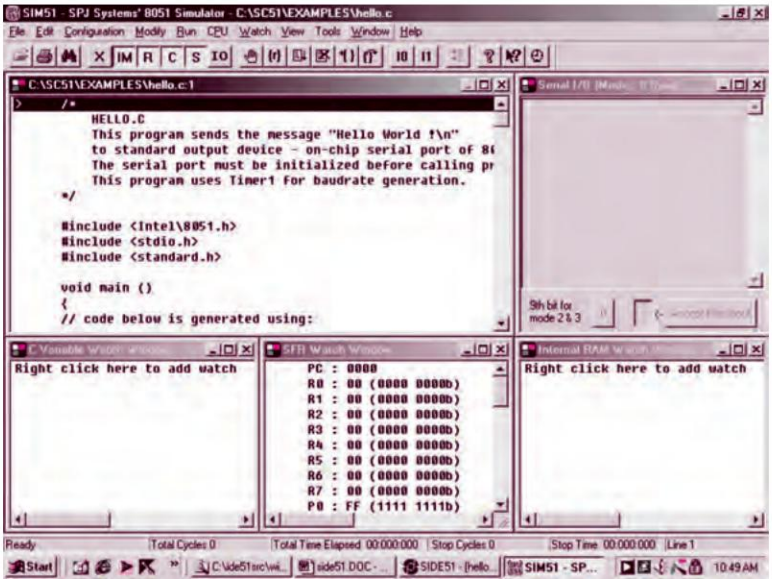


Figure 5.3 *Window while running a program (Courtesy SPJ Systems)*

Once the required window is visible, run the program either in continuous mode or in single step mode.

To execute the program in single step mode, select Run followed by Single step from the menu or press F7. This executes a statement at a time. On the contrary, if you select Run followed by Run from the menu (or press CTRL+F7), the program will run in continuous mode. To stop the program execution, select Run followed by Terminate program (or press CTRL+F7).

For simulation of the program, the summarised steps are given below.

1. Enter 'SIDE 51' software.
2. Click open 'Project' menu in the window.
3. Click on 'New project'.
4. Give the project file name as **name. P51** (.P51 extension is created by default).
5. In settings window, select Intel device as 8051 and click OK.
6. In the workspace window, you can observe that the default .C file and .asm files are created. Remove the default .C and .asm files.
7. Click on 'add file ' and enter the file name as name.asm or name.c.
8. Open the file, enter the program and save the file.

9. Open compile menu and click on build (or F7).
10. Correct the errors indicated, if any and save.
11. Open 'tools' menu and click open the simulator (F5).
12. Open 'run' menu and run the program. Observe Special Function Register (SFR) contents.
13. Close the simulator window.

5.4 µVISION C COMPILER AND SIMULATOR

µ Vision3 Overview

The µVision3 IDE is a Windows-based software development platform that combines a robust editor and project manager. µVision3 integrates all tools including C compiler, macro assembler, linker/locator, and HEX file generator. µVision3 helps expedite the development process of embedded applications.

Create Project File Folder and Specify Project Name

To create a new project file, from µVision3 menu, select Project/New/µVision Project…. The window appears as shown in Fig. 5.4. This opens a standard Windows dialog that asks you for the new project file name. Use a separate folder for each project. To get a new empty folder, use the icon 'Create New Folder' in the dialog.

Select this folder and enter the file name for the new project, i.e., Project1. µVision3 creates a new project file with the name PROJECT1.UV2 that contains a default target and file group name. You can see these names in the Project Workspace–Files.

Select Microcontroller from Device Database

When you create a new project, µVision3 asks you to select a CPU for the project. The Select Device dialog box shows the µVision3 device database as shown in Fig. 5.5. Select the microcontroller, e.g., Philips 89C62x.

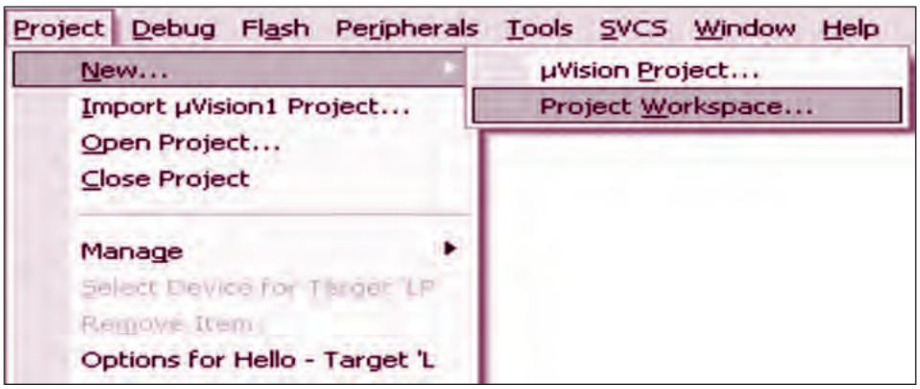


Figure 5.4 *Window to create a project (Courtesy Keil Corp. Inc.)*

Add Source Files to Project

Once the source file is created, add .asm or .c file to your project. µVision3 offers several ways to add source files to a project as shown in Fig. 5.6.

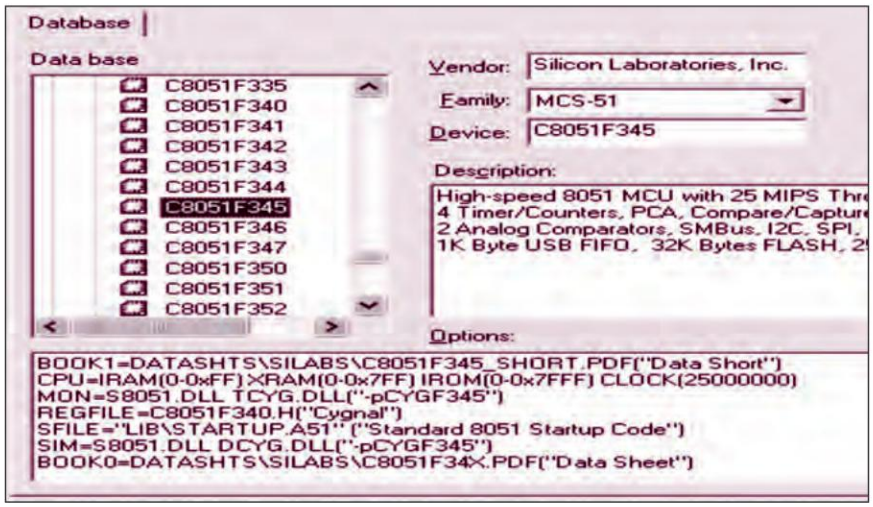


Figure 5.5 *Window to select target microcontroller (Courtesy Keil Corp. Inc.)*

For example, you can select the file group in the Project Workspace and Files page and click the right mouse key to open a local menu. The option 'Add Files' opens the standard files dialog. Select the file MAIN.C just created.

Build Project

To build a project, go to Options/Target, translate all source files and link the application with a click on the Build Target toolbar icon. When you build an application with syntax errors, µVision3 will display errors and warning messages in the Output Window–Build page as shown in Fig. 5.7. A double click on a message line opens the source file on the correct location in a µVision3 editor window.

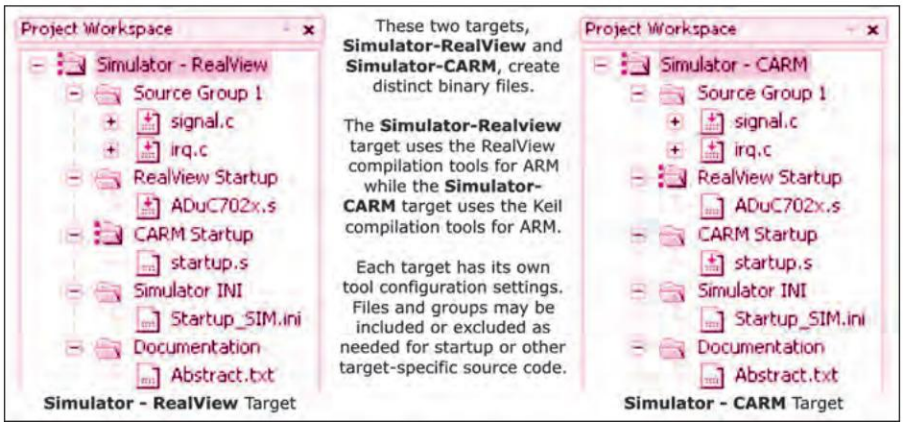


Figure 5.6 *Window to add source files to the project (Courtesy Keil Corp. Inc.)*

- Modify the existing source code or add new source files to the project. The Build Target toolbar button translates only modified or new source files and generates the executable file. µVision3

maintains a file dependency list and knows all the included files used within a source file. Even the tool options are saved in the file dependency list, so that µVision3 can rebuild files when it is needed. With the Rebuild Target command, all source files are translated, regardless of modifications.

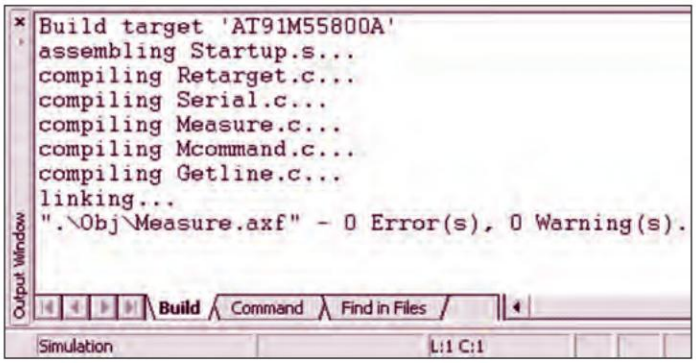


Figure 5.7 *Messages after compile/build of the program (Courtesy Keil Corp. Inc.)*

- Test programs with the µVision3 Debugger. The µVision3 Debugger offers two operating modes—simulator allows verifying application on PC, or Target Debugging with an Evaluation Board.
- To program your application into Flash ROM, µVision3 integrates command-line driven Flash Utilities ULINK USB JTAG. Create a HEX file to use Flash programming utilities.

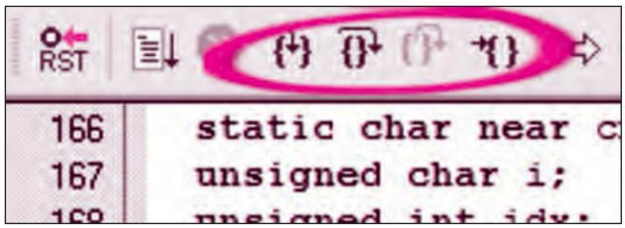


Figure 5.8 *Single stepping of the program (Courtesy Keil Corp. Inc.)*

Single Stepping

µVision supports various methods of single stepping through your application. The various options are available on the menu as shown in Fig. 5.8.

Click the 'Step Into' button on the toolbar to execute a single instruction or line of code (single-step). 'Step Into' single-steps into all called functions.

Click the 'Step Over' button on the toolbar to execute a single instruction or line of code (single-step). 'Step Over' executes all called functions without stepping into them.

Click the 'Step Out' button on the toolbar to step out of a function and return to the caller.

Click the 'Run to Cursor' button on the toolbar to begin executing your target program until the current cursor line is reached.

5.5 BURNING THE HEX FILE TO PROGRAM MEMORY

Click open the flash magic software. A window as shown in Fig. 5.9 appears on the screen. Select the Com 1/Com2 port to which the target microcontroller is connected. Enter the baud rate say 9600. Select the target device, e.g., 89F61x2. Oscillator frequency is 11.056 MHz.

Browse the target hex file (example test.hex). Click start icon on the window. Observe the message window at the bottom. It erases and then programs the flash memory. When burning of the program is completed, the message 'finished' is displayed. Then, the target board is ready for the intended application.

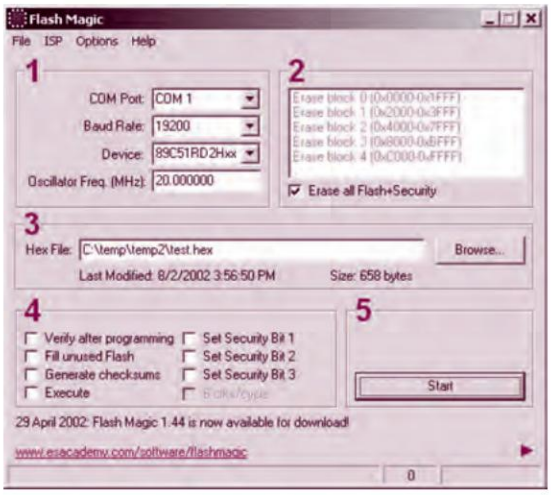


Figure 5.9 *Burning of hex file to the program memory (Courtesy Flash magic)*

CHAPTER SUMMARY

In this chapter, components of Integrated Development Environment (IDE) and In Circuit Debugger (ICD) have been discussed. It also explains the steps to work with assembler, simulator software with DOS and windows-based operating systems.

EXERCISES

REVIEW QUESTIONS

1. What is an Integrated Development Environment?
2. What is the role of In Circuit Debugger (ICD) in development environment?
3. What is a simulator? Explain its importance in code development?

6

8051 PARALLEL I/O PORTS

Learning Objectives

After you have completed this chapter, you should be able to

- Explain the data transfer synchronisation methods between the CPU and I/O interface
- Explain 8051 parallel I/O ports
- Input data from simple switches and output data to light emitting diodes (LED)
- Input data from matrix keyboard and output data to seven segment display
- Input data from matrix keyboard and output data to LCD display
- Use D/A converter to generate digital waveforms
- Interface multi channel 8 bit A/D converter to the 8051
- Interface serial A/D converter to the 8051
- Explain the operation and interfacing of stepper motor
- Explain the operation and interfacing of DC motor

6.1 BASIC I/O CONCEPTS

This chapter explores interfacing of peripheral devices, through which one interacts with a microcontroller. Examples of peripheral devices include switches, light emitting diodes, printers, keyboards, and liquid crystal displays. The speed and characteristics of these devices are different from that of the processor, so they cannot be connected to the processor directly. Hence, interface chips are needed to resolve the difference between processor and peripheral devices. The main function of an interface chip is to synchronize data transfer between the processor and an I/O device.

In case of input operation, the input devices place data in the data register of the interface chip and then, the processor reads the data. In the output operation, the processor writes data into the data register in the interface chip and the data register holds data until the output device fetches this data. An interface chip has data pins that are connected to the processor data bus, and I/O port pins that are connected to the I/O device. Data transfer between I/O devices takes place in serial (bits by bits) or in parallel (multiple bits). Data is generally transferred serially, in long distance communication and low speed devices. High speed I/O devices mainly use parallel data transfer. We will discuss only interfacing of parallel I/O in this chapter.

6.2 PORT STRUCTURES AND OPERATION

The 8051 has 32 I/O pins that are further divided into four ports—port 0, port 1, port 2 and port 3. These I/O pins allow the 8051 to monitor and control other devices. Port 0, port 2 and port 3 pins are multiplexed with an alternate function. Each port consists of a D latch, an output driver and an input buffer. To access external program and data memory the output driver of port 0 and port 2, and the input buffer of port 0 are used. In this application, low order address lines (A_7–A_0) and data bus (D_7–D_0) are multiplexed on port 0. Port 2 outputs the high-order address lines (A_{15}–A_8). Port 3 pins are used as external interrupt, for serial transfer, timer and external memory control signals as listed in Table 6.1. Port 1 pins have no dual functions. Port 3 is basically same as port 1 except that its, extra circuit allows dual functions. The output drivers of port 0 and port 2 are switchable to internal address/data (AD_0–AD_7) and address bus (A_8–A_{15}) respectively by an internal control signal during access of external memory. Ports 1, 2 and 3 have internal pull ups and port has open drain output. Open drain in Metal Oxide Semiconductors (MOS) means open collector in Transistor Transistor Logic (TTL).

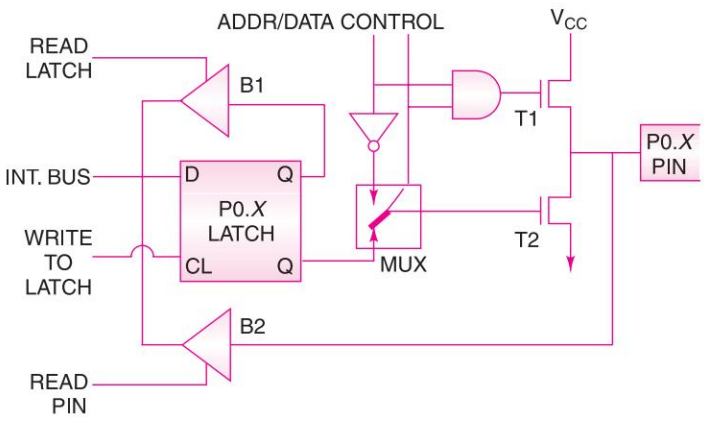


Figure 6.1 *PORT 0*

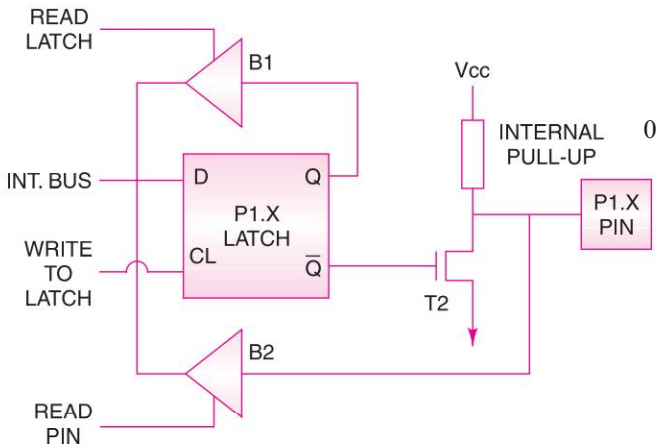


Figure 6.2 *PORT 1*

Input/Output Ports

All 8051 ports have both D-latch and buffer as shown in Fig. 6.1 to 6.4. All the ports can be defined as input or output port. Port 0 has no internal pull up resistors (as shown in Fig. 6.1), it is

simply an open drain. Now by writing a '1', to corresponding port 0 latch, both the transistors are OFF and that causes the pin to float in a high impedance state. When port 0 is used for simple data I/O, then external pull up resistor is required.

Output port All the bits of port register are physically connected with data bus. When the data is written to port register, the data bit is latched to the D flip flop in the port circuit. The processor generates $\overline{WR}$ signal, when an instruction moves the data to the port. $\overline{WR}$ signal is connected as active low clock of the D flip flop. The data from the data bus line is latched to the D flip-flop. If input is 0 (D = 0), then Q = 0 and $\overline{Q}$ = 1 and transistor T2 is on and the condition of the output pin is 0. If input is 1 (D = 1), then Q = 1 and $\overline{Q}$ = 0 and transistor T2 is off and the condition of the output pin is 1.

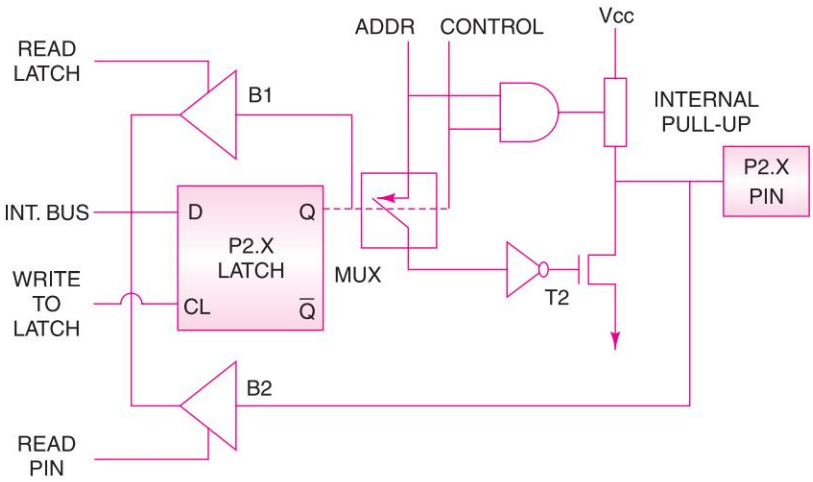


Figure 6.3 *PORT 2*

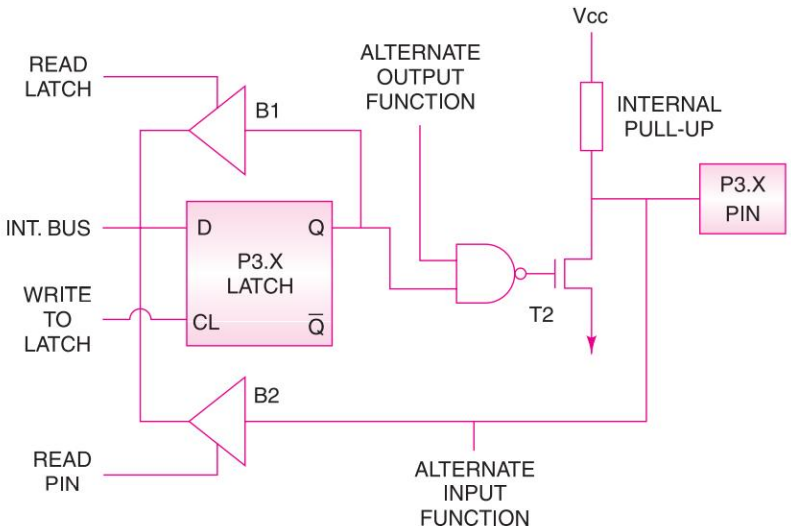


Figure 6.4 *PORT 3 (Courtesy Intel)*

Input port While reading the ports, there are two possibilities:

1. Reading the input pin
2. Reading the latch

Reading the input pin To define any port of the 8051 as input, we must first write '1' to that port bit. By writing '1' to that port bit, Q = 1 and $\overline{Q} = 0$. In port 0 and port 1, Q is connected to the transistor gate (T2); in port 2, Q is connected through not gate to T2; and in port 3, Q is connected through nand gate to T2. In all the cases, the transistor (T2) is turned off, the pin is simply pulled high by the pull up resistors, and connected to the input buffer (B2). When we read the input port, it activates B2 and brings the data from the pin into the CPU internal bus.

Reading the latch When we read the latch, it activates tristate buffer (B1) and brings the data from the Q latch into the CPU internal bus.

Read modify write Some of the instructions like INC P2 and DEC P2 will read a latch, modify and rewrite it to the latch. These instructions read the latch instead of reading the port pin. This happens for those instructions where the destination operand is a port or a port bit. These instructions are called 'read modify write instructions'. The reason behind reading a latch rather than a port pin is to avoid possible misinterpretation of voltage level at the port pin, when a port bit is driving an external circuit.

Alternate Functions

Port 0 Port 0 is used as an address/data bus to external memory; internal control signals switch the AD0–AD7 to the gates of the field effect transistor. Logic 1 on an address bit will turn the upper FET on and the lower FET off and provides logic 1 at the pin. When the address bit is 0, the lower FET is on and the upper FET is off and provides logic 0 at the pin. The address latch latches the address into the external circuit. After Address Latch Enable (ALE) pulse, port 0 is configured as data bus to read/write data from the external memory. When Port 0 is configured as an input, logic 1 is automatically written by the internal control logic to all port 0 latches.

Port 2 The alternate function of port 2 is to send high order address byte (A8–A15). The logic 1 on the address bit will turn off T2 and provides logic 1 at the output pin. The logic 0 on the address bit will turn on T2 and provides logic 0 at the output pin.

Port 3 Port 3 functions similar to port 1 and also, it is used by internal peripherals as listed in Table 6.1. Each pin of port 3 can be programmed individually as I/O or as one of the alternate functions.

We can also access single pin of all the four ports using bits address as given in Table 6.2.

As discussed under Section 1.6, the drawback in assembly language

TABLE 6.1 *Port pin alternate functions*

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (timer/counter 0 external input)
P3.5	T1 (timer/counter 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

programming is that it is difficult and time consuming to write a program. Writing programs in a high level language such as C is easier. For interfacing I/O devices, programs are given both in assembly language and C. We will discuss few examples to become familiar with the syntax.

TABLE 6.2 *Bit address of the ports*

Port 0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
Address	87H	86H	85H	84H	83H	82H	81H	80H
Port 1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
Address	97H	96H	95H	94H	93H	92H	91H	90H
Port 2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
Address	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H
Port 3	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
Address	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H

EXAMPLE 6.1

Write 8051 Assembly Language Program (ALP) and C program to send 8 bit hex number to port 0.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #0CH     ;Load A hex number 0C
        MOV P0, A       ;Send the data to port 0
        END
```

C PROGRAM

```
        // C program to send hex data to port 0
        #include <Intel\8051.h>
        void main (   )
             {
         unsigned char a;
             a=0x0c;    //Initialise a variable to hex value
             P0=a;          //Send hex number to port 0
             }
```

EXAMPLE 6.2

Write 8051 ALP and C program to send 8 bit binary numbers from 0 to 9 to port 2, repetitively.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
   S1:MOV A, #00H      ; Load A 00H
      MOV R1, #0AH     ; Load R1 0AH
START: MOV P2, A       ; Send the contents of accumulator to P2
      INC A            ; Increment A
      CALL DELAY       ; Call delay routine
      DJNZ R1, START   ; Decrement. R1,if it is not zero, branch to
                         START
      SJMP S1          ; Jump to S1
DELAY: MOV R2,#100     ; Delay routine for 100 ms
LOOP2: MOV R1,#125
LOOP1: PUSH ACC
  POP  ACC
  NOP
  NOP
  DNZ  R1, LOOP1
  DNZ  R2, LOOP2
  RET
  END
```

C PROGRAM

```
// C program to send 0-9 to port2
#include <Intel\8051.h> // Include header file of 8051
#include<standard.h>    // For delay routine
#define period 100      // 100 ms
void main  (   )
    {
unsigned char a;
while(1)     // Always perform
    {
    for(a=0; a<10; a++) // Initialise variable a=0 and increment
            {
            P2=a;              // Send 8-bit number to port 2
            delay_ms(period); // Delay routine for 100 ms
            }
    }
    }
```

EXAMPLE 6.3

Write 8051 ALP and C program to toggle 8 bit of port 1.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #00H        ; Load A 00H
START:  MOV P1, A          ; Send the contents of A to P1
        CPL A              ; Complement contents of A
        CALL DELAY         ; Call delay routine
        AJMP START         ; Branch to start
DELAY:  MOV R2,#100        ; Delay routine for 100 ms
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to toggle LED's at port 1
#include <Intel\8051.h>
#include<standard.h>      // For delay routine
#define period 100        // 100 ms

void main (   )
      {
      while(1)                    // Always perform
            {
            P1=0xFF;              // To send 1s to port1
            delay_ms(period);
            P1=0x00;              // To send 0s to port 1
            delay_ms(period);
            }
      }
```

EXAMPLE 6.4

Write 8051 ALP and C program to toggle alternate bits at port 1.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #0AAH      ;Load A AAH
START:  MOV P1, A         ;Send the contents of A to P1
        CPL A             ;Complement contents of A
        CALL DELAY        ;Call delay routine
        AJMP START        ;Branch to start
DELAY:  MOV R2,#100       ;Delay routine
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to toggle alternate bits of port 1
#include <Intel\8051.h>
#include<standard.h>  // For delay routine
#define period 100    // 100 ms
void main ( )
    {
    while(1)          // Always perform
          {
          P1=0x55;          // To send 01010101 to port 1
          delay_ms(period);
          P1=0xAA;          // To send 10101010 to port 1
          delay_ms(period);
          }
    }
```

EXAMPLE 6.5

Write 8051 ALP and C program to toggle MSB bit of port 1.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        CLR P1.7          ;Clear the port 1.7
START:  CALL DELAY        ;Call delay routine
        CPL P1.7          ;Complement contents of port 1.7
        AJMP START        ;Branch to START
DELAY:  MOV R2,#100       ;Delay routine
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to toggle MSB bit of port 1
#include <Intel\8051.h>
#include<standard.h>      // For delay routine
#define period 100        // 100 ms
BIT disp1 P1.7       // Identify port P1.7 as disp1
void main (   )
   {
   while(1)                    // Always perform
          {
          disp1=1;             // Send 1 to port P1.7
          delay_ms(period);
          disp1=0;             // Send 0 to port P1.7
          delay_ms(period);
          }
   }
```

EXAMPLE 6.6

Write 8051 ALP and C program to toggle LSB bit of port 0.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        CLR P0.0              ;Clear the port 0.0
START:  CALL DELAY            ;Call delay routine
        CPL P0.0              ;Complement contents of port 0.0
        AJMP START            ;Branch to START
DELAY:  MOV R2,#100           ;Delay routine
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to toggle LSB bit of port 0
#include <Intel\8051.h>
#include<standard.h>     // For delay routine
#define period 100       // 100 ms
BIT disp1 P0.0           // Declare port P0.0 as disp1
void main ( )
      {
   while(1)                    // Always perform
         {
         disp1=1;                    // Send 1 to port P0.0
         delay_ms(period);
         disp1=0;                    // Send 0 to port P0.0
         delay_ms(period);
         }
   }
```

EXAMPLE 6.7

Write 8051 ALP and C program to left shift data at port 1 repetitively.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #01H        ;Load A 01H
        MOV R1, #07H       ;Load R1 07H
START:  MOV P1, A          ;Send the contents of A to P1
        RL A               ;Rotate contents of A left by one position
        CALL DELAY         ;Call delay routine
        DJNZ R1, START     ;Branch to START
        AJMP S1            ;Branch to S1
DELAY:  MOV R2,#100        ;Delay routine
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to left shift the data at port 1
#include <Intel\8051.h>
#include<standard.h>      // For delay routine
#define period 100        // 100 ms
void main ( )
{
unsigned char x;
P1=0x01;                  // Send a number to port 1
while(1)                  // Always perform
        {
        for(x=0;x<8;x++)
                {
                P1=P1<<1;  // Shift data at port 1 to left by one bit.
                delay_ms(period);
                }
        }
}
```

EXAMPLE 6.8

Write 8051 ALP and C program to receive 8 bits data from port P0 and P1. Perform AND operation of the received data and send the result to port P2.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV P0, #0FFH       ;Define port 0 as input
        MOV P1, #0FFH       ;Define port 1 as input
        MOV A, P1           ;Read data from port1
        MOV R1, A           ;Copy contents of A to R1
        MOV A, P0           ;Read data from port 0
        ANL A, R1           ;or cont. of A with R1
        MOV P2,A            ;send the contents of A to P2
        END
```

C PROGRAM

```
        // C program to AND 8 bits data of port 0 and port 1 and to
        send result to port 2
        #include <Intel\8051.h>
        unsigned char input1;
        unsigned char input2;
        unsigned char result;
        void main (   )
        {
        while(1)                    // Do always
        {

        input1=P1;                  //Read 8 bit data at port P1
        input2=P0;                  //Read 8 bit data at port P0
        result=input1 & input2;     //AND operation
        P2=result;                  //Send result to P2
        }

        }
```

EXAMPLE 6.9

Write 8051 ALP and C program to receive 1 bit data from port P0.0 and P1.3. Perform AND operation of the received bits and send the result to port P2.0.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV P0, #0FFH         ;Define port 0 as input
```

```
        MOV P1, #0FFH          ;Define port 1 as input
        MOV A, P1              ;Read data from port1
        RR A                   ;Rotate contents of A, 3 times right
        RR A
        RR A
        MOV R1, A              ;Copy contents of A in R1
        MOV A, P0              ;Read data from port 0
        ANL A, R1              ;Perform AND operation
        MOV P2, A              ;Send the result to P2.0
        END
```

C PROGRAM

```
        //C program AND P0.0 and P1.3 send result at P2.0
        #include <Intel\8051.h>
        BIT input1 P0.0        //Declare port P0.0 as input1
        BIT input2 P1.3        //Declare port P1.3 as input2
        BIT output P2.0        //Declare port P2.0 as output
        void main ( )

              {
              while(1)         // Always perform
                    {
                    output=input1 & input2;  // AND operation
                    }
              }
```

EXAMPLE 6.10

Write 8051 ALP and C program to receive 1 bit data from port P0.0 and P1.2. Perform OR operation of the received bit and send the result to port P2.7.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV P0, #0FFH     ;Define port 0 as input
        MOV P1, #0FFH     ;Define port 1 as input
        MOV A, P1         ;Read data from port 1
        RR A              ;Rotate contents of A right by two positions
        RR A
        MOV R1, A         ;Copy contents of A to R1
        MOV A, P0         ;Read data from port 0
```

```
        ORL A, R1              ;Perform OR operation
        RR A                   ;Rotate right to get LSB in MSB
        MOV P2, A              ;Send the result to P2.7
        END
```

C PROGRAM

```
        // C program OR P0.0 and P1.2 send result at P2.7
        #include <Intel\8051.h>
        BIT  input1 P0.0       //Declare port P0.0 as input1
        BIT input2 P1.2        //Declare port P1.2 as input2
        BIT output P2.7        //Declare port P2.7 as output

        void main ( )
               {
               while(1)
                      {
                      output=input1|input2;
                      }
               }
```

EXAMPLE 6.11

Write 8051 ALP and C program to send ASCII character data ('A', '@', '!', '*') through port P0, P1, P2 and P3 respectively.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #41H        ;Load ASCII code for A  in hex 41H to A
        MOV P0, A          ;Send the data to port 0
        MOV A, #40H        ;Load ASCII code for @  in hex 40H to A
        MOV P1, A          ;Send the data to port 1
        MOV A, #21H        ;Load ASCII code for ! in hex 21H to A
        MOV P2, A          ;Send the data to port 2
        MOV A, #2AH        ;Load ASCII code for * in hex 2AH to A
        MOV P3, A          ;Send the data to port 3
        END
```

C PROGRAM

```
        // C program to send ASCII values of the characters
        #include <Intel\8051.h>
        const unsigned char input[4]={ 'A', '@', '!', '*'}; //
        declare ASCII data array
```

```
void main (  )

{
P0=input [0];        //Send ASCII value to P0
P1=input [1];        //Send ASCII value to P1
P2=input [2];        //Send ASCII value to P2
P3=input [3];        //Send ASCII value to P3
}
```

EXAMPLE 6.12

Write 8051 ALP and C program to read port 1. If the received data is equal to 20H, send FFH to port 2, otherwise send 00H to port 3.

ASSEMBLY LANGUAGE PROGRAM

```
      ORG 0000H
      MOV P1, #0FFH      ;Define port 1 as input
      MOV A, P1          ;Read data from port 1
      CLR C              ;Clear carry flag
      SUBB A, #20H       ;Compare (A) with 20H using sub operation
      JZ LOOP1           ;If data is equal, branch to Loop1
      MOV A, #00H        ;If data is not equal, sends 00 to P3
      MOV P3, A
      SJMP LOOP2         ;Branch to Loop2
LOOP1: MOV A, #0FFH      ;Send FF to port2
      MOV P2, A
LOOP2: NOP
      END
```

C PROGRAM

```
// C program to read port 1 and send data to port 2 and port 3
#include <Intel\8051.h>
unsigned char a;
unsigned char b=0x00;        //Declare variable b=00H
unsigned char c=0xFF;        //Declare variable c=FFH

void main (  )
```

```
{

a=P1;                    //Read port P1 to a
     if (a!=20)
     P3=b;               // Send 00H to port P3
     else
     P2=c;               // Send FFH to port P2

      }
```

SECTION REVIEW

1. The 8051 has _____ I/O pins.
2. In long distance communication, data is transferred _____.
3. List ports which have internal pull up resistors.
4. List ports which have alternate functions.
5. Port 1 has no dual functions. True/ False?
6. List pins of port 3, which are used as serial ports.
7. _____ port is used as write strobe and _____ port is used as read strobe for external data memory.
8. Open drain in MOS means _____ in TTL.
9. Port 3.2 is used as an external interrupt. True/False?

6.3 INTERFACING PUSH BUTTON SWITCHES AND LEDS

EXAMPLE 6.13

Write an ALP to monitor and sense the push button keys and display the key value by corresponding LED.

In this problem, 8 push button keys are connected to port 1 and 8 LED's are connected to port 0 as shown in Fig. 6.5. Port 1 is defined as input by storing FFh in port 1 and port 0 is used as output. When a push button key is pressed, it bounces (makes and breaks) a few times before it makes a firm contact.

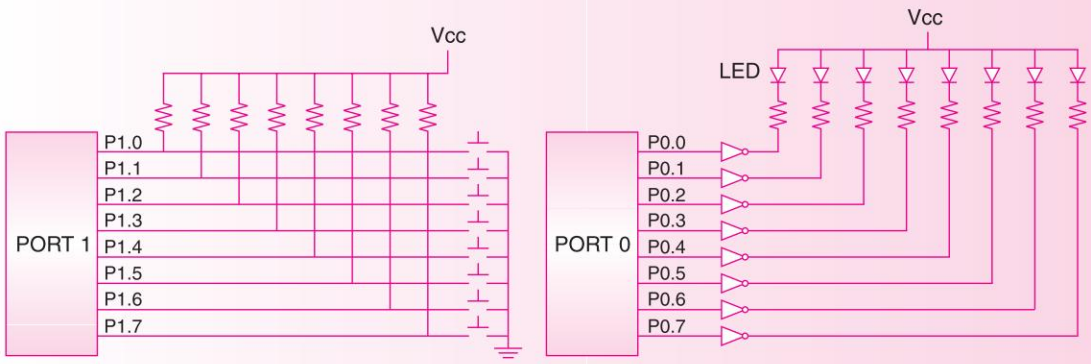


Figure 6.5 *Push button key and LED interface*

The solution to this problem is to wait for 10 to 20 ms until the key is settled and check the key again. The display contains 8 common cathode LEDs. Interfacing eight keys of push button keyboard and eight LEDs using ports 1 and 0 is as shown in Fig. 6.5. Programming of this problem can be divided into the following steps:

1. Check if a key is pressed
2. Debounce the key
3. Identify the key in binary format
4. Display the key condition using 8 LEDs.

PROGRAM IS AS FOLLOWS:

```
        MOV P1,#0FF          ;Make P1 input
START:  MOV A,P1             ;Read data from Port 1
        CJNE A,#0FFH,CHECK1  ;Key pressed branch to check1
        SJMP START           ;Branch to start
CHECK1: ACALL DELAY          ;Call delay
        MOV A,P1             ;Read data from Port 1
        CPL  A               ;Complement A
        MOV P0,A             ;Send the data to LED
        AJMP START           ;Branch to start
DELAY:  MOV R6,#20h          ;Delay program, R6 = 20h
NEXT1:  MOV R7,#0FFH         ;R7 = FFH
NEXT2:  DJNZ R7, NEXT2       ;Stay until R7 becomes 0
        DJNZ R6, NEXT1       ;Dec. R6, if it not zero, branch to NEXT1
        RET
        END
```

SECTION REVIEW

1. What is a key bounce?
2. What is the software solution to a key bounce?
3. Name the types of LED.

6.4 INTERFACING MATRIX KEYBOARD AND SEVEN-SEGMENT DISPLAY

EXAMPLE 6.14

Write ALP and C program to read a key from matrix keyboard and to display key value using seven-segment display.

Matrix keyboard is commonly used as an input device, when more keys are required. A matrix keyboard reduces the number of connections, e.g, a keyboard with 16 keys, arranged in a 4 × 4 matrix as shown in Fig. 6.6. It requires eight lines for the microcontroller to make all the connections instead of 16 lines, if the keys are connected in a linear format. When a key is pressed, it shorts one row and one column; otherwise, the row and column do not have any connections. For interfacing 4 × 4 matrix keyboard, it requires 4 bit port for row lines and 4 bit port for column lines. 4 bit port (P1.3–P1.0) connected to row lines, are defined as output port and the remaining 4 bit port (P2.3–P2.0) connected to column lines, are defined as input port as shown in Fig. 6.6.

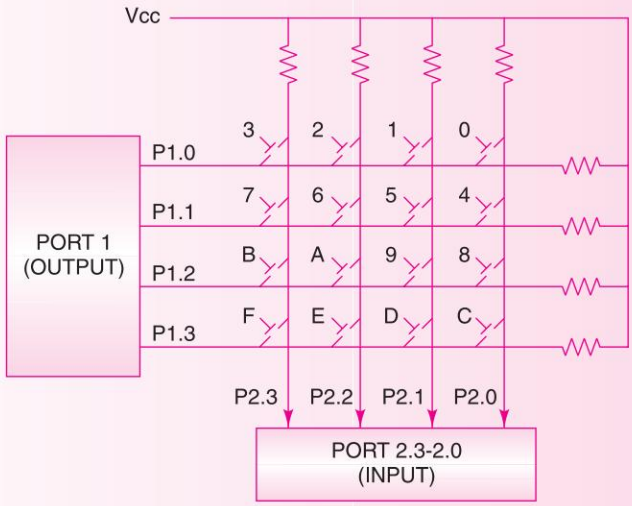


Figure 6.6 *Matrix keyboard connection to ports*

The seven-segment display consists of seven LED segments a, b, c, d, e, f, g and h for decimal point (dp). The seven-segment display comes in either common cathode or common anode form. In a common anode display, the anodes of all seven LED's are shorted. A segment is ON whenever a OV is applied to the corresponding segment's input and common anode is connected to +5V. In a common cathode display all, cathodes of seven segments are shorted. A segment is ON whenever a +5V is applied to the corresponding segment's input and common cathode is connected to ground. BCD to seven-segment codes for common cathode display are given in Table 6.3

TABLE 6.3 *BCD to seven segment codes*

BCD Digit	Segments								Corresponding hex Number
	h	g	f	e	d	c	b	a	
0	0	0	1	1	1	1	1	1	3FH
1	0	0	0	0	0	1	1	0	06H
2	0	1	0	1	1	0	1	1	5BH
3	0	1	0	0	1	1	1	1	4FH
4	0	1	1	0	0	1	1	0	66H
5	0	1	1	0	1	1	0	1	6DH
6	0	1	1	1	1	1	0	1	7DH
7	0	0	0	0	0	1	1	1	07H
8	0	1	1	1	1	1	1	1	7FH
9	0	1	1	0	0	1	1	1	67H
A	0	1	1	1	0	1	1	1	77H
B	0	1	1	1	1	1	1	1	7FH
C	0	0	1	1	1	0	0	1	39H
D	0	0	1	1	1	1	1	1	3FH
E	0	1	1	1	1	0	0	1	79H
F	0	1	1	1	0	0	0	1	71H

Circuit to interface matrix keyboard 4 × 4 and 4 common cathode seven-segment display are as shown in Fig. 6.6 and 6.8. In Fig. 6.8, the common cathode of each display is connected to the collector of a transistor. Port 2.4 and 2.5 are connected to a decoder, and the decoder output is connected to transistors. Depending on the P2.4 and P2.5 port values, a particular transistor will be turned on and driven into the saturation region, allowing the selection of the seven-segment display.

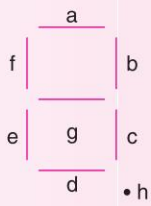


Figure 6.7 *Seven-segment display*

Programming of this problem is to read a key from matrix keyboard and display key value using seven-segment display, can be divided into the following steps:

1. Check whether all the keys are open.
2. Check a key closure.
3. Debounce the key.
4. Identify the key.
5. Find the binary code for the key.
6. Find the seven-segment code for the binary code.
7. Send seven-segment code to display.

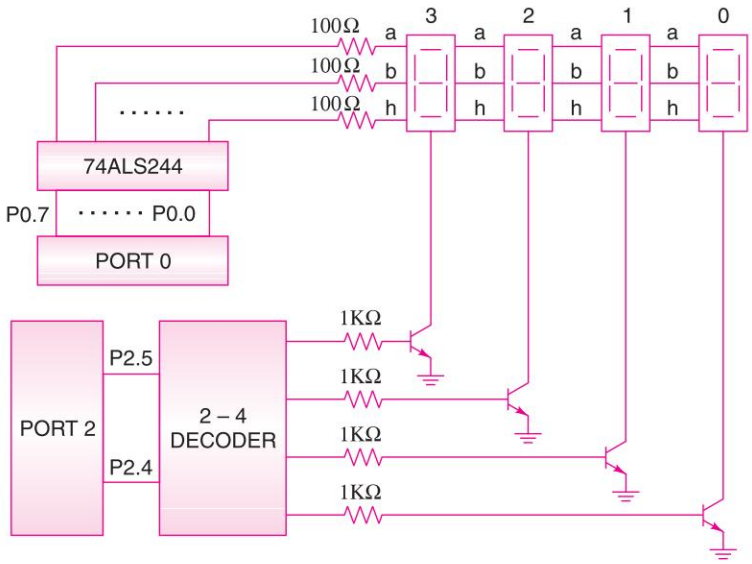


Figure 6.8 *Drivers for seven-segment displays*

PROGRAM IS AS FOLLOWS:

```
        ORG 0000H
        SEL_DIGIT EQU 80H          ;Init SEL_DIGIT to port 0
        SEL_SEG EQU 0A0H           ;Init SEL-SEG to port 2
        KEY_ROW EQU 90H            ;Init KEY_ROW to port 1
        KEY_COL EQU 0A0H           ;Init KEY_COL to port 2
        MOV KEY_COL, #0Fh          ;Define KEY_COL as input port
    START:MOV KEY_ROW,#00          ;Ground all the row lines
        MOV A,KEY_COL              ;Read all column lines
        ANL A,#0Fh                 ;Mask unused bits
        CJNE A,#0FH,CHECK1         ;Key pressed branch to check1
        SJMP START                 ;Branch to start till key pressed
    CHECK1:ACALL DELAY             ;Wait debounce time
         MOV A,KEY_COL             ;Check key closure
```

```
    ANL A,#0Fh              ; Mask unused bits
    CJNE A,#0Fh,NEXT        ; Key pressed branch to next to find row
    SJMP START              ; Branch to start till key pressed
NEXT:MOV KEY_ROW,#0Eh       ; Ground row 0
    MOV A,KEY_COL           ; Read all column lines
    ANL A,#0Fh              ; Mask unused bits
    CJNE A,#0Fh,SCAN_ROW0   ; If key is in row 0, branch to SCAN_ROW0
    MOV KEY_ROW,#0Dh        ; Ground row 1
    MOV A,KEY_COL           ; Read all column lines
    ANL A,#0Fh              ; Mask unused bits
    CJNE A,#0Fh,SCAN_ROW1   ; If key is in row 1, branch to SCAN_ROW1
                            MOV KEY_ROW,#0Bh  ;Ground row 2
    MOV A,KEY_COL           ; Read all column lines
    ANL A,#0Fh              ; Mask unused bits
    CJNE A,#0Fh,SCAN_ROW2   ; If key is in row 2, branch to SCAN_ROW2
    MOV KEY_ROW,#07h        ; Ground row 2
    MOV A,KEY_COL           ; Read all column lines
    ANL A,#0Fh              ; Mask unused bits
    CJNE A,#0Fh,SCAN_ROW3   ; If key is in row 3, branch to SCAN_ROW3
    SJMP CHECK1             ; If none of the keys are pressed,
                            branch to check1
  SCAN_ROW0:MOV DPTR,#ROW0  ; Copy row 0 address to DPTR
  SJMP FIND_KEY             ; Find column
  SCAN_ROW1:MOV DPTR,#ROW1  ; Copy row 1 address to DPTR
  SJMP FIND_KEY             ; Find column
  SCAN_ROW2:MOV DPTR,#ROW2  ; Copy row 2 address to DPTR
  SJMP FIND_KEY             ; Find column
  SCAN_ROW3:MOV DPTR,#ROW3  ; Copy row 3 address to DPTR
  SJMP FIND_KEY             ; Find column
FIND_KEY: RRC A             ; Find column and update DPTR
  JNC FOUND_KEY
  INC DPTR
  SJMP FIND_KEY
FOUND_KEY: MOV SEL_SEG,#0Fh ; Select display
  CLR A                     ; Clear accumulator
  MOVC A,@A+DPTR            ; Get the seven-segment code
  MOV SEL_DIGIT,A           ; Send the data to display
  LJMP START                ; Branch to start
DELAY: MOV R6,#20h          ; Delay program, R6 = 20h
NEXT6: MOV R7,#0FFh         ; R7 = FFh
```

```
NEXT7: DJNZ R7,NEXT7          ;Stay until R7 becomes 0
DJNZ R6,NEXT6                 ;Stay until R6 becomes 0
RET
ROW0: db 3Fh, 06h, 5bh, 4Fh   ;Seven-segment code for 0,1,2,3
ROW1: db 66h, 6dh, 7dh, 07h   ;Seven-segment code for 4,5,6,7
ROW2: db 7Fh, 67h, 77h, 7Ch   ;Seven-segment code for 8,9,A,B
ROW3: db 39h, 5Eh, 79h, 71h   ;Seven-segment code for C,D,E,F
END
```

C PROGRAM

```
#include <Intel\8051.h>
#define key_row P1              //Define port 1 for row scan
unsigned char i=0,temp1,temp2,key,var;
    const unsigned char
    seg[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,
    0x07,0x7f,0x67,0x77,0x7c,0x39,0x5e,0x79,0x71};
    // Declaration of segment table for seven-segment display
    void main ()
    {
    while(1)
    {
            P2 = 0x0f;       // Configure port 2 lower as input port
            key_row =0x00;   // Configure port 1 as output port

            do
            {
            temp1 = P2 & 0x0f;
         }while(temp1==0x0f);// Wait till the key is pressed
         while(i<100)
    {
         i=i+1;
    }// De bouncing
    P2= 0x0f;

    var = 0xf7;// For scanning rows
    temp2=0; // Key count

    for(i=0;i<4;i++)
    {
         var = var<<1;
         key_row =var>>4; // Scan rows
         temp1 = P2&0x0f; // Key Input

    if(temp1!=0x0f)
```

```
    {
        // To check which key is pressed
        switch (temp1){
           case 0x0e:
                  P0 = seg[temp2+0]; //Display on seven-segment
                                     break;
        case 0x0d:
                  P0 = seg[temp2+1]; // Display on seven-segment
                                     break;
        case 0x0b:
                  P0 = seg[temp2+2]; // Display on seven-segment
                                     break;
        case 0x07:
                  P0 = seg[temp2+3]; // display on 7 segment
                                     break;
        }

    }
        temp2=temp2+4;     // To go to next row
    }   //End of For loop
}
}       // end of main
```

EXAMPLE 6.15

Write ALP and C program for hexadecimal up counter (o-F)

The program for hexadecimal up counter is as follows:

```
   ORG 0000h
  TEMP EQU 40h
  LOOP1:MOV A,#00h    ;Clear A
 LOOP:MOV TEMP,A      ;Store A in TEMP
   LCALL DISPLAY      ;Call display-to-display counter value
   LCALL DELAY        ;Call delay
   MOV A,TEMP         ;Copy TEMP value to A
   ADD A,#01h         ;Increment counter value
   CJNE A,#10h,LOOP   ;Compare counter value with 10h,< branch to loop
   LJMP LOOP1         ;Else branch to loop1
   DISPLAY:ANL A,#0Fh;Display program to display counter value
   MOV DPTR,#TABLE
   MOVC A,@A+DPTR
```

```
  MOV P0,A
  CLR P2.4
  CLR P2.5
  RET

DELAY:MOV R2,#05h      ;Delay program
LOOP7:MOV R3,#0FFh
LOOP3:MOV R4,#0FFh
LOOP2:NOP
  NOP
  NOP
  DJNZ R4,LOOP2
  DJNZ R3,LOOP3
  DJNZ R2,LOOP7
  RET

  TABLE:DB 3Fh,06h,5Bh,4Fh,66h,6Dh,7Dh,07h,7Fh,6Fh,77h,7Fh,39h,3Fh,
  79h,71h
  END
```

C PROGRAM

```
#include <Intel\8051.h>

idata unsigned int temp=0;
unsigned char count;
BIT disp1 P2.4          // Assign port 2.4 for display
BIT disp2 P2.5          // Assign port 2.5 for display
Const unsigned char
value[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,
0x07,0x7f,0x67,0x77,0x7c,0x39,0x5e,0x79,0x71};// Segment code for
                                                 display

void main ( )
{
    disp1=0;
    disp2=0;//Select display
    while(1)
    {
         count=0;
         do
         {
```

```
            P0=value[count];//Data to display
            while(temp<0xFFFF){
                    temp=temp+1;
        }                   //Generate delay
        temp=0;
        count=count+1;      // Increment counter by 1
        }while(count<0x10);  //End of loop
    }
}       //End of main
```

EXAMPLE 6.16

Write ALP and C program for Decimal up counter (0-9).

The program for decimal up counter is as follows:

```
     ORG 0000h
     TEMP EQU 40h
    LOOP1:CLR A          ;Clear A
LOOP:MOV TEMP,A          ;Store A in TEMP
    LCALL DISPLAY        ;Call display to display counter value
    LCALL DELAY          ;Call delay
    MOV A,TEMP           ;Copy TEMP value to A
    ADD A,#01h           ;Increment counter value
    DA A
    CJNE A,#10,LOOP      ;Compare counter value with 10,< branch to loop
    LJMP LOOP1           ;Else branch to loop1
    DISPLAY:ANL A,#0Fh ;Display program to display counter value
     MOV DPTR,#TABLE
     MOVC A,@A+DPTR
     MOV P0,A
     CLR P2.4
     CLR P2.5
     RET
DELAY:MOV R2,#05h        ;Delay program
LOOP7:MOV R3,#0FFh
LOOP3:MOV R4,#0FFh
LOOP2:NOP
    NOP
```

```
     NOP
     DJNZ R4,LOOP2
     DJNZ R3,LOOP3
     DJNZ R2,LOOP7
     RET

TABLE:DB 3Fh,06h,5Bh,4Fh,66h,6Dh,7Dh,07h,7Fh,6Fh
     END
```

C PROGRAM

```
#include <Intel\8051.h>

idata unsigned int temp=0;
unsigned char count;
BIT disp1 P2.4            // Assign port 2.4 for display
BIT disp2 P2.5                  // Assign port 2.5 for display
const unsigned char
value[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67,
0x77,0x7c,0x39,0x5e,0x79,0x71};// Segment code for display

void main ()
{
      disp1=0; //Select display
      disp2=0;
      while(1)
      {
            count=0;
            do
            {
                P0=value[count];
                while(temp<0xFFFF){
                     temp=temp+1;         //Generate Delay

      }
                temp=0;
                count=count+1;// Increment counter by 1
            }while(count<0x0A);          // Loop until 10 counts
      }
}           //End of main
```

EXAMPLE 6.17

Write ALP and C program for hexadecimal down counter (F-o).

The program for hexadecimal down counter is as follows:

```
      ORG 0000h
      TEMP EQU 40h
      LOOP1:MOV A,#0Fh
   LOOP:MOV TEMP,A       ;Store A in TEMP
      LCALL DISPLAY      ;Call display to display counter value
      LCALL DELAY        ;Call delay
      MOV A,TEMP         ;Copy TEMP value to A
      CLR C
      SUBB A,#01h        ;Decrement counter value
      CJNE A,#0FFh,LOOP  ;Comp counter value with FFh,< branch to loop
      LJMP LOOP1         ;Else branch to loop1
      DISPLAY:ANL A,#0Fh ;Display program to display counter value
      MOV DPTR,#TABLE
      MOVC A,@A+DPTR
      MOV P0,A
      CLR P2.4
      CLR P2.5
      RET

 DELAY:MOV R2,#05h       ;Delay program
 LOOP7:MOV R3,#0FFh
 LOOP3:MOV R4,#0FFh
 LOOP2:NOP
       NOP
       NOP
       DJNZ R4,LOOP2
       DJNZ R3,LOOP3
       DJNZ R2,LOOP7
       RET

 TABLE:DB 3Fh,06h,5Bh,4Fh,66h,6Dh,7Dh,07h,7Fh,6Fh,77h,7Fh,39h,3Fh,
 79h,71h
       END
```

C PROGRAM

```
#include <Intel\8051.h>
idata unsigned int temp=0;
unsigned char count;
```

```
        BIT disp1 P2.4        // Assign port 2.4 for display
        BIT disp2 P2.5        // Assign port 2.5 for display
        const unsigned char
        value[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
        0x7f,0x67,0x77,0x7c,0x39,0x5e,0x79,0x71};// Segment code for
        seven-segment display
        void main ()
        {
        disp1=0;
        disp2=0;
        while(1)
        {
               count=0x0f;
               do
        {
               P0=value[count];
               while(temp<0xFFFF){
                      temp=temp+1;// Generate delay
               }
               temp=0;
               count=count-1;// Decrement count by 1
        }while(count>=0);    // Loop until count is 0
        }
}       // end of main
```

EXAMPLE 6.18

Write ALP and C program for decimal down counter (9-0).

The program for decimal down counter is as follows:

```
        ORG 0000h
        TEMP EQU 40h
 LOOP1:MOV A,#09h
   LOOP:MOV TEMP,A       ;Store A in TEMP
        LCALL DISPLAY    ;Call display to display counter value
        LCALL DELAY      ;Call delay
        MOV A,TEMP       ;Copy TEMP value to A
        CLR C
        SUBB A,#01h      ;Decrement counter value
        CJNE A,#0FFh,LOOP;Compare counter value with FFh,< branch to Loop
```

```
        LJMP LOOP1        ;Else branch to Loop1

 DISPLAY:ANL A,#0Fh       ;Display program to display counter value
        MOV DPTR,#TABLE
        MOVC A,@A+DPTR
        MOV P0,A
        CLR P2.4
        CLR P2.5
        RET

 DELAY:MOV R2,#05h        ;Delay program
 LOOP7:MOV R3,#0FFh
 LOOP3:MOV R4,#0FFh
 LOOP2:NOP
       NOP
       NOP
       DJNZ R4,LOOP2
       DJNZ R3,LOOP3
       DJNZ R2,LOOP7
       RET

TABLE: DB 3Fh,06h,5Bh,4Fh,66h,6Dh,7Dh,07h,7Fh,6Fh
       END
```

C PROGRAM:

```
#include <Intel\8051.h>

idata unsigned int temp=0;
unsigned char count;
BIT disp1 P2.4            // Assign port 2.4 for display
BIT disp2 P2.5            // Assign port 2.5 for display
const unsigned char
value[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67,
0x77, 0x7c,0x39,0x5e,0x79,0x71};// Segment code for display

void main ()
{
        disp1=0;
        disp2=0;
        while(1)
```

```
                {
                        count=0x09;
                do
                {
                        P0=value[count];
                        while(temp<0xFFFF){
                             temp=temp+1;  // Generate delay
                }
                temp=0;
                count=count-1;              // Decrement count by 1
                }while(count>=0);
        }
}       // end of main
```

SECTION REVIEW

1. Name the types of seven-segment display.
2. _____ seven-segment code is used to display 8 in common cathode display.
3. 71H is seven-segment code to display F in common cathode display. True/ False?
4. _____ number of I/O pins are required to interface 3 4 matrix keyboard.
5. In matrix keyboard, the ports for row lines are defined as output ports. True/ False?
6. To detect the key press in a matrix keyboard, one row at a time is grounded. True/ False?
7. In 4 4 matrix keyboard, if 0 number key is pressed, then the condition of the column line in hexadecimal will be _____.
8. List the steps to detect a key in 4 4 matrix keyboard.

6.5 INTERFACING MATRIX KEYBOARD AND LIQUID CRYSTAL DISPLAY (LCD)

EXAMPLE 6.19

Write ALP and C program to read a key from a matrix keyboard and display the key value using liquid crystal display (LCD).

The matrix keyboard interfacing is as discussed in Section 6.4. The LCD is interfaced by using Port 0 and Port 2.4–2.6 as shown in Fig. 6.9 and the pin details of the LCD are given in Table 6.4. When voltage is applied across the LCD segment, electrostatic field is created that aligns crystals in the liquid. This alignment allows light to pass through the segment. LCD needs a driving circuit. LCD and

driving circuits are integrated in LCD display module. The display has one command register and one data register. Data to be displayed should be in ASCII. Data and command registers are differentiated by the RS input. The interfacing circuit allows the microcontroller to send commands and data to the LCD and also to read the status. The commands for LCD are given in Table 6.5.

TABLE 6.4 *Pin details of LCD*

Pin	Symbol	I/O	Description
1	Vss	---	Ground
2	Vcc	--	+5 V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/ $\overline{W}$	I	R/ $\overline{W}$ = 0 for write, R/ $\overline{W}$ = 1 for read
6	E	I	Enable
7	D0	I/O	The 8 bit data bus
8	D1	I/O	The 8 bit data bus
9	D2	I/O	The 8 bit data bus
10	D3	I/O	The 8 bit data bus
11	D4	I/O	The 8 bit data bus
12	D5	I/O	The 8 bit data bus
13	D6	I/O	The 8 bit data bus
14	D7	I/O	The 8 bit data bus

Programming to read a key from matrix keyboard and display the key value using liquid crystal display can be divided into the following steps:

- Initialise LCD
- Find the key value
  1. Check whether all keys are open
  2. Check a key closure
  3. Debounce the key
  4. Identify the key
- Find the binary code for the key
- Send the data to LCD
  1. Find the ASCII code for the binary
  2. Send the command code to LCD
  3. Check busy-bit for sending the data
  4. Send the ASCII code to LCD

TABLE 6.5 *Commands of LCD*

Code (Hex)	Commands to LCD Instruction Register
1	Clear Display Screen
2	Return Home
4	Decrement Cursor
6	Increment Cursor
5	Shift Display Right
7	Shift Display Left
8	Display Off, Cursor Off
A	Display Off, Cursor On
C	Display On, Cursor Off
E	Display On, Cursor Blinking
F	Display On, Cursor Blinking
10	Shift Cursor Position To Left
14	Shift Cursor Position To Right
18	Shift The Entire Display To The Left
1C	Shift The Entire Display To The Right
80	Force Cursor To Beginning of 1st line
C0	Force Cursor To Beginning of 2nd line
38	2 Lines and 5 × 7 Matrix

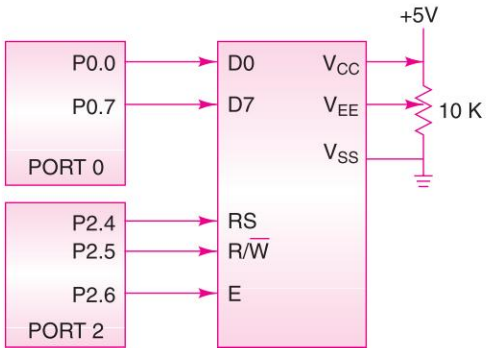


Figure 6.9 *LCD Interface connections*

THE PROGRAM IS AS FOLLOWS:

```
    ORG 0000h
    LCDDATA EQU 80h          ;Init LCD data lines to port 0
    LCDCTRL EQU 0A0h         ;Init LCD control lines to port 2.4, 2.5
                             and 2.6
    KEY_ROW EQU 90h          ;Init KEY_ROW to port 1
    KEY_COL EQU 0A0h         ;Init KEY_COL to lower port 2
    BUSY_BIT EQU 87h         ;Init busy-bit to 87H
    REG_SEL EQU 0A4h         ;Init LCD control line REG_SEL to port 2.4
    READ_WRITE EQU 0A5h      ;Init LCD control line READ_WRITE to port 2.5
    ENABLE EQU 0A6h          ;Init LCD control line ENABLE to Port 2.6
MAIN:ACALL LCD_INIT          ;Call LCD_INIT and initialise LCD
BACK:ACALL FIND_KEY_PRESS    ;Call FIND_KEY_PRESS and get the key value
    ACALL DISP_KEY           ;Call DISP_KEY and display key value
    SJMP BACK
LCD_INIT:MOV A,#38h          ;Init LCD 2 lines, 5x7 matrix
    CALL CMD_WRITE           ;Call CMD_WRITE
    MOV A,#0Ch               ;Display ON cursor OFF
    ACALL CMD_WRITE          ;Call CMD_WRITE
    MOV A,#01h               ;Clear display screen
    ACALL CMD_WRITE          ;Call CMD_WRITE
    MOV A,#06h               ;Increment cursor
    ACALL CMD_WRITE          ;Call CMD_WRITE
    MOV A,#80h               ;Force cursor to the beginning of the
                              first line
    ACALL CMD_WRITE          ;Call CMD_WRITE
    RET
CMD_WRITE:ACALL CHK_BUSY_BIT  ;Check LCD ready
    MOV LCDDATA,A             ;Send command code
    CLR REG_SEL               ;RS=0 for command
    CLR READ_WRITE            ;R/W = 0 to write
    SETB ENABLE               ;E = 1
    CLR ENABLE                ;E = 0, to latch
    RET
DATA_WRITE:ACALL CHK_BUSY_BIT ;Check LCD ready
    MOV LCDDATA, A            ;Send data
     SETB REG_SEL             ;RS=1 for data
    CLR READ_WRITE            ;R/W = 0 to write
    SETB ENABLE               ;E = 1
    CLR ENABLE                ;E = 0, to latch
    RET
```

```
CHK_BUSY_BIT:CLR ENABLE          ;Check LCD ready, E = 0
    SETB BUSY_BIT                ;BUSY_BIT = 1
    CLR REG_SEL                  ;RS = 0
    SETB READ_WRITE              ;R/W = 1
CHK_BUSY:CLR ENABLE              ;E = 0
    SETB ENABLE                  ;E = 1
    JB BUSY_BIT, CHK_BUSY        ;Stay until busy flag = 0
    CLR ENABLE                   ;E = 0
    RET
FIND_KEY_PRESS:MOV KEY_COL, #0Fh ;Define KEY_COL as input port
START:MOV KEY_ROW, #00h          ;Ground all the row lines
    MOV A, KEY_COL               ;Read all the column lines
    ANL A, #0Fh                  ;Mask unused bits
    CJNE A, #0Fh, CHECK1         ;Key pressed branch to check1
    SJMP START                   ;Branch to start till key pressed
CHECK1:LCALL DELAY               ;Wait debounce time
    MOV A,KEY_COL                ;Read all the column lines
    ANL A,#0Fh                   ;Mask unused bits
    CJNE A,#0Fh,NEXT             ;Key pressed branch to NEXT
    SJMP START                   ;Branch to start till key pressed
 NEXT:MOV KEY_ROW, #0Eh          ;Ground row 0
    MOV A, KEY_COL               ;Read all column lines
    ANL A, #0Fh                  ;Mask unused bits
    CJNE A, #0Fh, SCAN_ROW0      ;If key is in row 0, branch to SCAN_ROW0
    MOV KEY_ROW, #0Dh            ;Ground row 1
    MOV A, KEY_COL               ;Read all column lines
    ANL A, #0Fh                  ;Mask unused bits
    CJNE A, #0Fh,SCAN_ROW1       ;If key is in row 1, branch to SCAN_ROW1
    MOV KEY_ROW, #0Bh            ;Ground row 2
    MOV A, KEY_COL               ;Read all column lines
    ANL A, #0Fh                  ;Mask unused bits
    CJNE A, #0Fh,SCAN_ROW2       ;If key is in row 1, branch to SCAN_ROW2
    MOV KEY_ROW, #07h            ;Ground row3
    MOV A, KEY_COL               ;Read all column lines
    ANL A, #0Fh                  ;Mask unused bits
    CJNE A, #0Fh,SCAN_ROW3       ;If key is in row 1, branch to SCAN_ROW3
    SJMP START                   ;If none of the keys are pressed,
                                 branch to start
SCAN_ROW0:MOV DPTR, #ROW0        ;Copy ROW0 address to DPTR
    SJMP FINDKEY                 ;Find column
SCAN_ROW1:MOV DPTR, #ROW1      ;Copy ROW1 address to DPTR
```

```
    SJMP FINDKEY                ;Find column
SCAN_ROW2:MOV DPTR, #ROW2       ;Copy ROW2 address to DPTR
    SJMP FINDKEY                ;Find column
SCAN_ROW3:MOV DPTR, #ROW3       ;Copy ROW3 address to DPTR
    SJMP FINDKEY                ;Find column
FINDKEY:RRC A                   ;Find column and update DPTR
    JNC FOUND_KEY
    INC DPTR
    SJMP FINDKEY
FOUND_KEY:RET
DISP_KEY:MOV A,#80h  ;Force cursor to the beginning of the first line
    ACALL CMD_WRITE  ;Call CMD_WRITE
     CLR A           ;Clear A
     MOVC A, @A+DPTR ;Get ASCII code from the code memory
     ACALL DATA_WRITE ;Send the ASCII code to LCD
     RET
DELAY:MOV R6,#20h    ;Delay program
NEXT6:MOV R7,#0FFh
NEXT7:DJNZ R7,NEXT7
     DJNZ R6,NEXT6
     RET
ROW0: dB '0', '1', '2', '3'  ;ASCII code for 0,1,2,3
ROW1: dB '4','5','6','7'     ;ASCII code for 4,5,6,7
ROW2: dB '8','9','A','B      ;ASCII code for 8,9,A,B
ROW3: dB 'C','D','E','F'     ;ASCII code for C,D,E,F
END
```

C PROGRAM

```
#include <Intel\8051.h>
#define key_row P1

BIT reg_sel P2.4
BIT read_write P2.5
BIT enable P2.6
BIT busy_bit 0x87         // port initialisation

unsigned char i=0;
idata unsigned char temp;
idata unsigned char temp2,temp1;
unsigned char var;
```

```
idata unsigned char lcdvalue;
const unsigned char seg[16]={'0','1','2','3','4','5','6','7',
'8','9','A','B','C','D','E','F'};
      // ASCII look up table
void data_write();
void cmd_write();
void chk_busy();

void main ()
{
// Initialise LCD display
       temp= 0x38; // Command to select 2 lines and 5x7 matrix display
       cmd_write();
       temp= 0x0c; // Command to select display on and cursor off
       cmd_write();
       temp= 0x01; // Command to clear display
       cmd_write();
       temp= 0x06; // Command to increment cursor
       cmd_write();
       temp= 0x80; // Command to force cursor to the beginning of
                      the first line
       cmd_write();
while(1)
{
       P2 = 0x0f;
       key_row =0x00;
       do
       {
              temp1 = P2 & 0x0f;
       }while(temp1==0x0f);// Wait till the key is pressed

       while(i<100)
       {
              i=i+1;
       }// De bouncing
       P2= 0x0f;

       var = 0xf7;// For Scanning rows
       temp2=0; // Key Count
```

```
        for(i=0;i<4;i++)
        {
                var = var<<1;
                key_row =var>>4; // Scan rows
                temp1 = P2&0x0f; // Key input
        if(temp1!=0x0f)
        {
                switch (temp1){
                        case 0x0e:
                              lcdvalue=  (seg[temp2+0]);
                              break;
                        case 0x0d:
                              lcdvalue=(seg[temp2+1]);
                              break;
                        case 0x0b:
                              lcdvalue= (seg[temp2+2]);
                              break;
                        case 0x07:
                              lcdvalue= (seg[temp2+3]);
                              break;
                }

                }
                        temp2=temp2+4;    // Move to next row

                }       //End of for loop

                        temp= 0x80; // Command to force cursor to the
                                      beginning of the first line
                        cmd_write();
                        data_write();     //Write key value to LCD
                }
}
// Function to write command to LCD
void cmd_write()

{
        chk_busy();// Check LCD is Available
        P0=temp;        // Write command
```

```
        reg_sel=0;
        read_write=0;
        enable=1;
        enable=0;//Latch
}
// Function to write data to LCD
void data_write()
{
        chk_busy(); //Check if LCD is available
        P0 = lcdvalue; //Write data
        reg_sel =1;
        read_write =0;
        enable =1;
        enable =0; //Latch
}
// Function to check if LCD is free
void chk_busy()
{
        enable=0;
        busy_bit =1;
        reg_sel =0;
        read_write =1;
        do
        {
                enable=0;
                enable =1;
        }while (busy_bit);    //Wait until busy_bit is cleared
        enable =0;
}
```

SECTION REVIEW

1. _____ pin of LCD selects the command and data register.
2. List the registers in LCD.
3. RS and E pins of LCD are input pin. True/False?
4. Name the control pins of LCD.
5. For LCD to display letters and numbers, the data is in ASCII. True/False?

6.6 INTERFACING D/A CONVERTER USING PARALLEL PORTS

EXAMPLE 6.20

Interface DAC 08 and write ALP and C program to generate square wave.

Digital to analog converters are required when a digital code must be converted to an analog signal. Figure 6.10 shows interfacing of DAC 08, with reference current 2 mA by using +5 V power supply and 2.5 KΩ resistance. It has eight digital input lines and an output line for analog signal. The number of data bits decide the resolution of the DAC. The output pin I_{out} is connected to a current to voltage converter circuit. The eight input lines can assume eight input combinations from 00000000 to 11111111. The range of D/A converter is 0 to +5 V. Then, for input 00000000, the output of D/A converter is 0 V; for input 10000000, the output is +2.5 V; and for input 11111111, the output is +5 V.

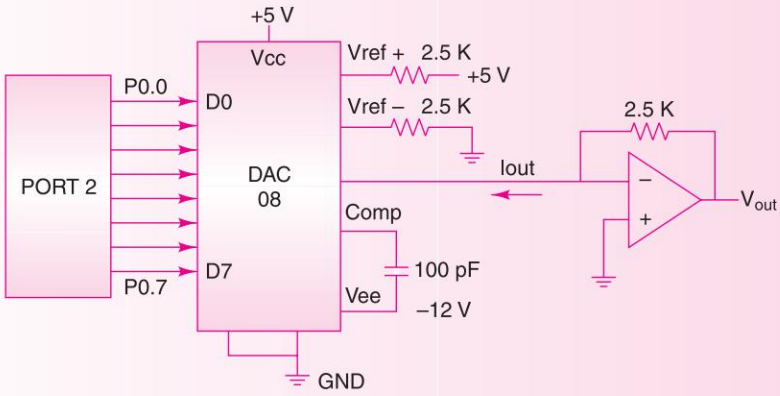


Figure 6.10 *8051 connection with DAC 08*

Generation of Square Wave

In this problem, since D/A converter is an output device, port 2 is used as an output port. Connect the output of D/A converter to an oscilloscope. Delay program decides the 'on period' and 'off period' of square wave. Program to generate a square wave is as follows:

```
        ORG 0000h
        CLR A                ;Clear A
BACK:   MOV P2,A             ;Send data to port 2 (to DAC)
        LCALL DELAY          ;Delay decides period of square wave
        CPL A                ;Complement content of A
        SJMP BACK            ;Branch to BACK
DELAY:                       MOV R1,#40h  ;Delay program
NEXT2:                       MOV R2,#0FFh
NEXT1:                       DJNZ R2, NEXT1
        DJNZ R1, NEXT2
        RET
        END
```

C PROGRAM:

```
#include <Intel\8051.h>
#include <standard.h>
# define PERIOD 250          // Define delay time 250 ms
void main ()
{
unsigned char a;
a=0xFF;
while (1)
{
P2=a; //Out data to port2
delay_ms (PERIOD);          // Delay function defined in standard.h for
                            delay in ms
a=~a;         // Toggle

}
}      // End of main
```

EXAMPLE 6.21

Interface DAC 08 and write ALP and C program to generate triangular wave.

Generation of Triangular Wave

In this problem, since D/A converter is an output device, port 2 is used as an output port. To generate a triangular wave, increment input to D/A from 00H to FFH, and then decrement from FFH to 00H. The program to generate triangular wave is as follows:

```
        ORG 00h
        CLR A
 BACK:  MOV P2,A              ;Send data to port 2 (to DAC)
        LCALL DELAY              ;DAC conversion time
        ADD A,#01h            ;Increment contents of A
        CJNE A,#0FFh,BACK     ;Increment contents of A up to FF (+5V)
BACK1:  MOV P2,A              ;Send data to port 2 (to DAC)
        LCALL DELAY              ;DAC conversion time
        CLR C                 ;Clear carry flag
        SUBB A,#01h           ;Decrement contents of A
        CJNE A,#00h, BACK1    ;Decrement contents of A up to 00 (+0V)
        SJMP BACK
DELAY:  MOV R1,#0Fh           ;Delay program
NEXT:   DJNZ R1,NEXT
        RET
```

C PROGRAM

```
#include <Intel\8051.h>
#include <standard.h>
void main ()
      {
      unsigned char a;
      while (1)
         {
               for (a=0; a<0xFF; a++) // To generate positive ramp
         {
             P2=a;
             delay_ms (1);
         }
             for (a=0xFF; a>0; a--)   // To generate negative ramp
         {
             P2=a;
             delay_ms (1);
             // Delay function defined in standard.h for delay in ms
             }
         }
     }          // End of main
```

EXAMPLE 6.22

Interface DAC o8 and write ALP and C program to generate saw tooth waveform.

Generation of Saw Tooth Wave

To generate saw tooth wave, increment input to D/A from ooH to FFH, and then the program will be in loop. Program to generate saw tooth wave is as follows:

```
      ORG 0000h
       CLR A
BACK: MOV P2,A          ;Send data to port 2 ( to DAC)
      INC A             ;Increment contents of A
      LCALL DELAY       ;DAC conversion time
      SJMP BACK         ;Branch to BACK
DELAY:MOV R1, #0Fh      ;Delay program
 NEXT:DJNZ R1, NEXT
      RET
```

C PROGRAM

```
#include <Intel\8051.h>
#include <standard.h>
void main ()
       {
```

```
unsigned char a;
while (1)
    {
        for (a=0; a<0xFF; a++)// To generate positive ramp

{
        P2=a;
        delay_ms (1);
        // Delay function defined in standard.h for delay in ms
}
    }
}   // End of main
```

EXAMPLE 6.23

Interface DAC 08 and write ALP and C program to generate sine wave.

Generation of Sine Wave

Full-scale output +5 V is achieved when all the data inputs of the DAC are high. To generate a sine wave, we first need a table whose values represent the magnitude of the sine angles between 0 and 360 degree angles. V_{out} of DAC for various sine degrees is calculated as follows.

$$V_{out} = 2.5\,V + (2.5\,V \times \text{Sin}\,\theta)$$

Table 6.6 shows the angles, the sine values, the voltage magnitude and the integer values representing the voltage magnitude for each angle with 10 degree increment.

TABLE 6.6 *Angle and voltage magnitude for Sine wave*

Angle θ	sin θ	V_{out}	Values to DAC	Angle θ	sin θ	V_{out}	Values to DAC
0	0	2.5	32	190	–0.173	2.067	27
10	0.173	2.935	38	200	–0.342	1.645	21
20	0.342	3.355	43	210	–0.500	1.250	16
30	0.500	3.750	48	220	–0.642	0.895	12
40	0.642	4.105	53	230	–0.766	0.585	8
50	0.766	4.415	57	240	–0.866	0.335	4
60	0.866	4.665	60	250	–0.939	0.152	2
70	0.939	4.847	62	260	–0.984	0.040	1
80	0.984	4.96	63	270	–1.000	0.000	0
90	1.000	5.0	64	280	–0.984	0.040	1
100	0.984	4.96	63	290	–0.939	0.152	2
110	0.939	4.847	62	300	–0.866	0.335	4
120	0.866	4.665	60	310	–0.766	0.585	8
130	0.766	4.415	57	320	–0.642	0.895	12
140	0.642	4.105	53	330	–0.500	1.250	16
150	0.500	3.750	48	340	–0.342	1.645	21
160	0.342	3.355	43	350	–0.173	2.067	27
170	0.173	2.935	38	360	0.000	2.5	32
180	0.000	2.5	32				

Program to generate Sine wave is as follows:

```
        ORG 00h
   BACK1: MOV R2,#25h   ; Define R2 as counter
          MOV DPTR,#TABLE
    BACK: CLR A
          MOVC A,@A+DPTR ; Get the data from the memory pointed by DPTR
           MOV P2,A      ; Send data to port 0
           LCALL DELAY
            INC DPTR
 DJNZ R2, BACK           ; Decrement counter of R2, if it is not 0,
                         branch to BACK
     AJMP BACK1          ; Branch to BACK1
  DELAY: MOV R1,#0Fh     ; Delay program
    NEXT:DJNZ R1,NEXT
      RET

 TABLE: dB 32, 38, 43, 48, 53, 57, 60, 62, 63, 64, 63, 62, 60, 57,
       53, 48, 43, 38, 32, 27, 21, 16,
        dB  12, 08, 04, 02, 01, 00, 01, 02, 04, 08, 12, 16, 21, 27,
       32
         END
```

C PROGRAM

```
#include <Intel\8051.h>
const unsigned char val [37]={32, 38, 43, 48, 53, 57, 60, 62, 63,
64, 63, 62, 60, 57, 53, 48, 43, 38, 32, 27, 21, 16, 12, 08, 04, 02,
01, 00, 01, 02, 04, 08, 12, 16, 21, 27};// Look up table of sine
unsigned char count;
void main ()
       {
            while (1)
            {
            count=0;
            while (count<37)      // up to 360
            {
            P2=val[count]; // Out data from table to port 2
            count=count+1; // Increment for next data
            }
            }
       } // End of main
```

SECTION REVIEW

1. In DAC 08, the output is current. True/False?
2. The reference current in DAC 08 is 2 mA. True/False?
3. DAC 08 has eight input data lines. True/False?
4. DAC 08 output pin I_{out} is connected to amplifier circuit. True/False?
5. In DAC 08, the input signal is analog. True/ False?

6.7 INTERFACING A/D CONVERTER USING PARALLEL PORTS

EXAMPLE 6.24

Write ALP and C program to start an A/D conversion and store the result in accumulator.

Analog to digital converters are required when an analog signal must be converted to a digital code. Figure 6.11 shows the circuit to inter-face an eight-channel, 8 bit A/D converter. It has eight input lines, three address lines (A B C) to select the input, ALE pin to latch the address and eight digital output lines. SC is for Start Conversion, EOC is for End Of Conversion and OE is for Output Enable to perform read operation. If $V_{ref}(+)$ = 5V and $V_{ref}(-)$ = GND, then the range of A/D converter is 0 to +5 V and the step size is 5V/256 = 19.53 mV. The eight output lines can assume eight output combinations from 00000000 to 11111111. For 0V input, the output of A/D converter is 00000000 (00H); for input +2.5 V the output is 10000000 (80 H); and for input +5 V, the output is 11111111 (FF H).

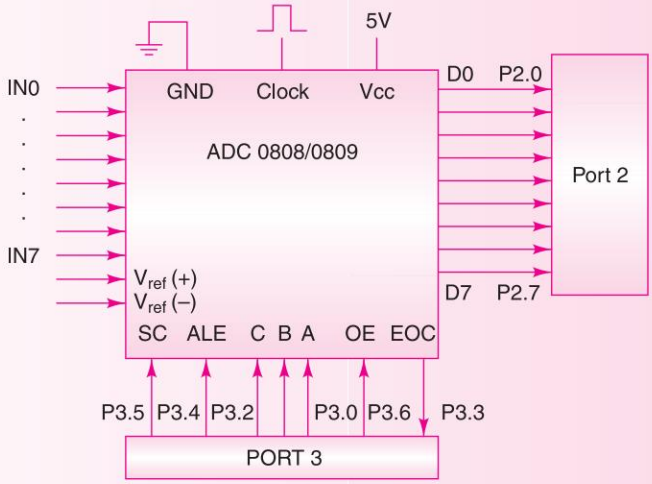


Figure 6.11 *ADC0808/0809*

In this problem, since A/D converter is an input device, Port 2 should be defined as an input port, P3.0 – P3.2 and P3.4 – P3.6 are used as output and P3.3 is defined as input. 2 seven-segment displays

are connected by using port 0, port 1.0 and port 1.1. Port 1.0 and port1.1 select the display. Port 0, port 1.1. and port 1.2 are used as output.

The program to start A/D conversion and display the result is as follows.

```
       ORG 0000H
        SEL_DIGIT EQU 80h  ; Send data to display using port 0
        SEL_SEGMENT EQU 90h; Display is selected using P1.1 and P1.0
        AD_OE BIT P3.6     ; Output Enable is assigned to AD_OE
        AD_SC BIT P3.5     ; Start Conversion is assigned to AD_SC
        AD_EOC BIT P3.3    ; EOC is assigned to AD_EOC
        AD_ALE BIT P3.4    ; ALE is assigned to AD_ALE
        AD0 BIT P3.0       ; Multiplexer channel addressing
        AD1 BIT P3.1
        AD2 BIT P3.2
        MOV P2,#0FFh       ; Make P2 input
        MOV P3,#08h        ; Make P3.3 (EOC pin) input
        CLR AD_ALE         ; ALE =0
        CLR AD0            ; Channel 0 is selected
        CLR AD1
        CLR AD2
        SETB AD_ALE        ; ALE =1, latch the address
  MAIN: SETB AD_OE         ; Output buffer is enabled
        CLR AD_SC          ; Send pulse to start conversion
        CALL DELAY
        SETB AD_SC
        CALL DELAY
        CLR AD_SC
  LOOP1:MOV A,P3           ; Wait for end of conversion
        ANL A,#08h
        CJNE A,#08h,LOOP1
        MOV A,P2           ; Read the data
        MOV R2,A
  LOOPD:MOV A,R2
        ANL A,#0FH
        MOV DPTR,#TABLE
        MOVC A,@A+DPTR
        MOV SEL_DIGIT,A    ; Send the lower nibble data to display1
        CLR P1.0
        SETB P1.1
```

```
        CALL DELAY
        MOV A,R2
        SWAP A
        ANL A,#0FH
        MOVC A,@A+DPTR
        MOV SEL_DIGIT,A ; Send the upper nibble data to display2
        SETB P1.0
        CLR P1.1
        AJMP MAIN
DELAY:  MOV R7,#100     ; Delay program
        D1:NOP
        NOP
        DJNZ R7,D1
        RET
TABLE: db 3Fh, 06h, 5bh, 4Fh, 66h, 6dh, 7dh, 07h, 7Fh, 67h, 77h, 7Ch,
       39h, 5Eh, 79h, 71h
       END
```

C PROGRAM

```
#include <Intel\8051.h>
          // Port initialisation
BIT AD_OE P3.6;   // Output Enable
BIT AD_SC P3.5;   // Start Conversion
BIT AD_EOC P3.3;  // End Of Conversion
BIT AD_ALE P3.4;  // ALE signal
BIT AD0 P3.0;
BIT AD1 P3.1;
BIT AD2 P3.2; // Channel selection bits
BIT AD3 P1.0;
BIT AD4 P1.1;
unsigned char value,dig1,dig2,temp,i;
const unsigned char
seg[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67,0x77,0
x7c,0x39,0x5e,0x79,0x71};
        // Segment codes for Seven-segment display
void delay(unsigned int count);
void main ()
{
   P2=0xff;
   P3= 0x08;
   AD_ALE=0;
```

```
   // Select channel 0
   AD0=0;
   AD1=0;
   AD2=0;
   AD_ALE=1;// Generate ALE
   while(1)
   {
          AD_OE=1;
   // Generate start of conversion pulse
          AD_SC=0;
          delay(100);
          AD_SC=1;
          delay(0x100);
          AD_SC=0;
   Do {
          }while(!AD_EOC);// Wait till end of conversion is detected
          value =P2; // Read ADC output
          dig1=value&0x0f; // Display lower nibble in display 1
          P0=seg[dig1];
          AD4=0;
          AD5=1;
          while(i<100)
          {
                 i=i+1;
          }
          i=0;
          AD4=1;
          AD5=0;
          dig2=value&0xf0; // Display higher nibble in display 2
          dig2=dig2>>4;
          P0=seg[dig2];
                 }
}

   // Function to generate delay
void delay(unsigned int count)
{
        unsigned int j;
        for(j=0;j<count;j++)
        {
        }
}
```

SECTION REVIEW

1. In ADC0808, input signal is analog. True/False?
2. ADC0809 has three address lines to select the input lines. True/False?
3. If range of 8 bit A/D converter is 0 to +5 V, the step rise is _____.
4. If range of 8 bit A/D converter is 0 to +5 V for input +1.25 V, the output is _____.
5. What is the function of ALE and EOC pin in ADC0808?

6.8 INTERFACING SERIAL A/D CONVERTER

EXAMPLE 6.25

Write ALP and C program to start serial A/D conversion (ADC 1031) and store the result in registers.

A/D converter as discussed in Section 6.7 is of parallel type. Do to D7 data lines of the A/D converter are connected to an 8 bit port of the 8051 microcontroller.

Figure 6.12 shows the circuit to interface ADC1031. ADC1031 serial ADC chip from National semiconductor is 10 bit (Do–D9) serial A/D converter and most significant bit is shifted out first through data output pin (Do). Serial clock (S_{CLK}) input is used to bring data out; the falling edge shifts the data on data output pin (Do). The clock frequency applied to C_{CLK} pin can be from 700 KHz to 4 MHz. It operates with +5V power supply.

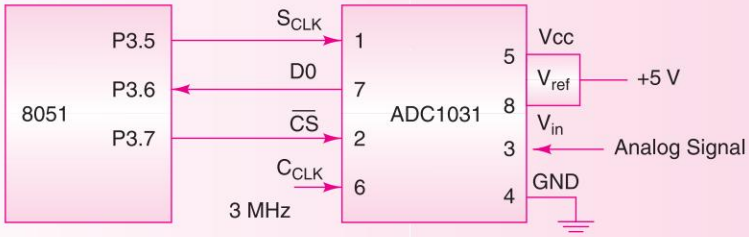


Figure 6.12 *Interfacing serial A/D*

In this problem, Do is connected to P3.6. P3.6 should be defined as input port whereas P3.5 and P3.7 should be defined as output port. 3 MHz clock signal is connected to C_{CLK} pin. Assume that 12 MHz crystal is connected to the 8051 microcontroller and 3 MHz clock signal is connected to C_{CLK} pin of ADC 1031.

The program to get 10 bits data and store the data in registers R4 and R3 is as follows:

```
     ORG 0000H
     CLR A          ; Clear A
     CLR P3.7       ; Select ADC 1031
```

```
    SETB P3.6       ; Define P3.6 as input
    MOV R2,#2       ; R2 is used as a counter to get MSB and MSB-1.
L1: SETB P3.5       ; Create falling edge in pin S_CLK
    CLR P3.5
    MOV A, P1.6     ; Copy data of A/D converter to LSB of A
    RLC A           ; Shift the bit
    DJNZ R2, L1     ; Get MSB and MSB-1 of A/D converter
    MOV R4,A        ; Store the data in R4
    CLR A           ; Clear accumulator
    MOV R2,#08      ; R2 is used as counter to get remaining 8 bit
L2: SETB P3.5       ; Create falling edge in pin S_CLK
    CLR P3.5
    MOV A,P1.6      ; Copy data of A/D converter to LSB of A
    RLC A           ; Shift the bit
    DJNZ R2, L2     ; Get the remaining 8 bit of A/D converter
    MOV R3,A        ; Store the data in R3
    SETB P3.7       ; Disable ADC 1031
    END
```

C PROGRAM

```
#include<intel /8051.h>
BIT AD_SEL  P3.7
BIT AD_INP  P3.6
BIT AD_EDGE P3.5
Unsigned char count, data1, data2
Void main ( )
    {
        AD_SEL=0;  // Select ADC
        AD_INP=1; // Configure input
        For (count=0; count<10; count++)
        {
        AD_EDGE=1;
        AD_EDGE=0; // Create falling edge
        If (count<2)
        {
        DATA1= ADC_INP;
        DATA1<<=1;
        } else
        {
```

```
        DATA2= ADC_INP;
        DATA2<<=1;
        }
        }     // End of loop
        AD_SEL=1;// Disable ADC
}       // End of main
```

SECTION REVIEW

1. ADC 1031 is a serial A/D converter. True/False?
2. ADC 1031 is _____ bit serial A/D converter.
3. In ADC 1031, the falling edge of S_{CLK} shifts the data on data output pin. True/False?
4. ADC 1031 operates with clock frequency _____ to _____.
5. Differentiate the functions of S_{CLK} and C_{CLK} pin of ADC 1031.

6.9 INTERFACING STEPPER MOTOR

EXAMPLE 6.26

Interface stepper motor and write ALP and C program to rotate the stepper motor in the direction given in the program.

Stepper motors, also known as stepping or step motors, are essentially incremental motion devices. If rotor rotates 90° in each step (from one pole segment to another), it is called a full step. In Fig. 6.13, the rotor rotates 90° in each step.

The stepper motor may also be operated with a half step. A half step occurs when the rotor is moved to 45°. Figure 6.14 illustrates the half-stepping sequence. The common step angles of stepper motor are 1.8°, 7.5° and 15°. If step size is 1.8°, then to complete one rotation or 360 degrees, it requires 200 steps. The step angle and the stepping code can be obtained from the stepper motor manufacturer, and the rotation speed depends on the delay program between the stepping codes.

The circuit to interface stepper motor is as shown in Fig. 6.15. The stepper motor can be driven directly by the transistors. Transistors are used to supply higher current to the motor. The microcontroller outputs the drive pattern to make the motor rotate. The diodes in Fig. 6.15 are called *fly back diodes* and are used to protect the transistors from reverse biases.

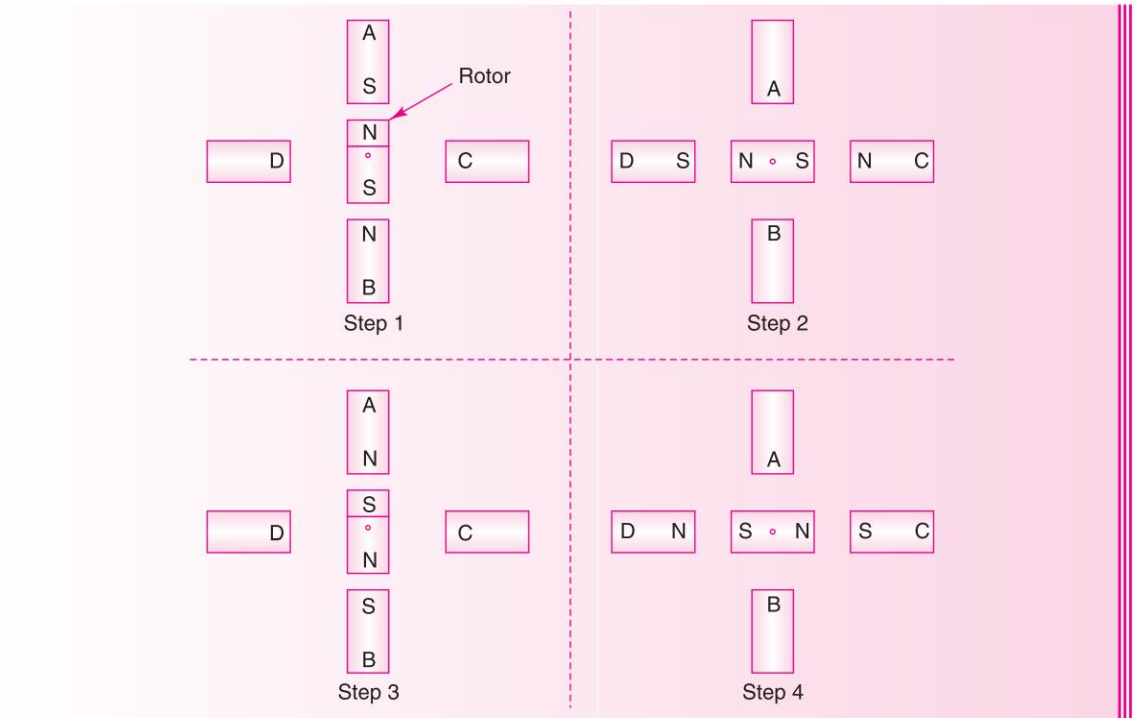


Figure 6.13 *Full step operation of step motor*

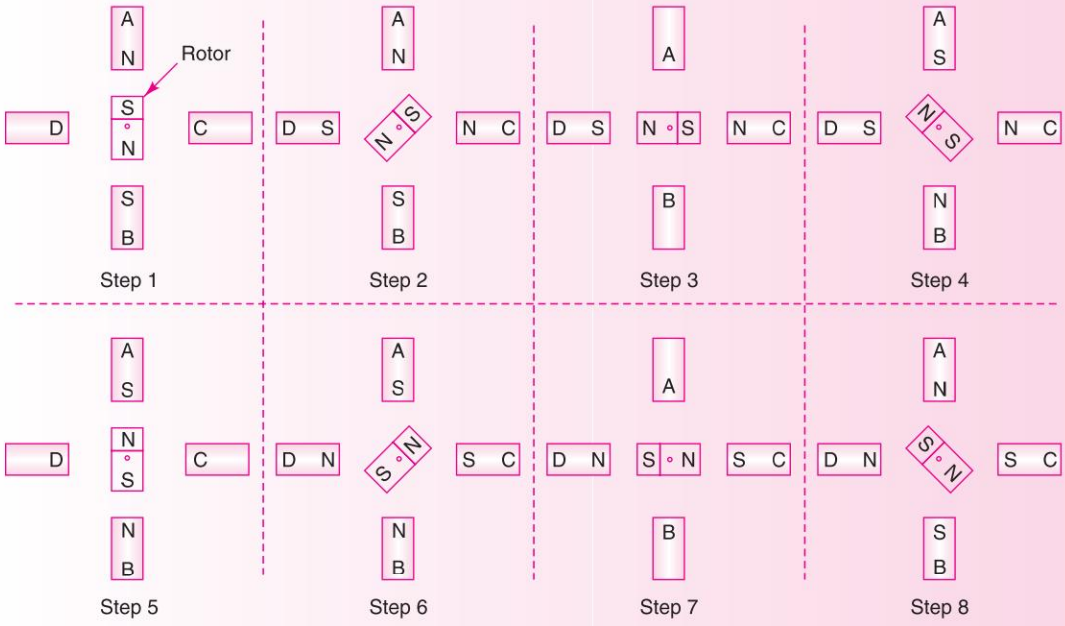


Figure 6.14 *Half step operation of stepper motor*

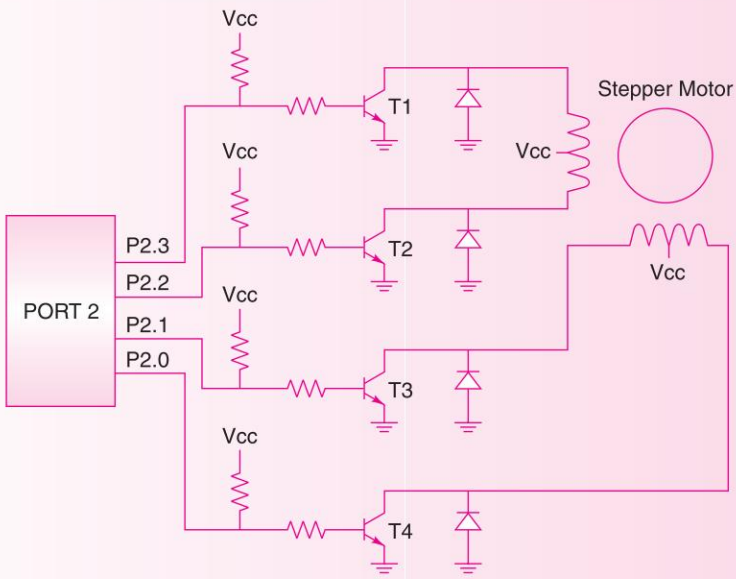


Figure 6.15 *Driver circuit and port interface for stepper*

In this problem, since the stepper motor is an output device, port 2 is used as an output port.

The program to rotate stepper motor is as follows.

```
     ORG 0000h
LOOP:MOV P2,#03h          ; Load step sequence 03
     ACALL DELAY          ; Call delay
     MOV P2,#09h          ; Load step sequence 09
     ACALL DELAY          ; Call delay
     MOV P2,#0Ch          ; Load step sequence 0C
     ACALL DELAY          ; Call delay
     MOV P2,#06h          ; Load step sequence 06
     ACALL DELAY          ; Call delay
     AJMP LOOP            ; Repeat the above procedure
DELAY:  MOV R5,#0FFh      ; Delay program
DELAY1: MOV R7,#0FFh
     S1: DJNZ R7,S1
         DJNZ R5,DELAY1
         RET
          END
```

EXAMPLE 6.27

Interface the stepper motor using port 2 and a switch using P1.6. Write ALP and C program to rotate the stepper motor in clockwise direction, if switch is ON, else in anticlockwise direction.

PROGRAM

```
      ORG 0000
      MOV P1,#40H       ; Define P1.6 as Input
  LOOP2:MOV A,P1        ; Read Port 1
      ANL A,#40H        ; Mask all the bits, except D6
      CJNEA,#00h,LOOP1 ; If switch is off, branch to LOOP1
      MOV P2,#03H       ; Switch is on, send step sequence for clockwise
      ACALL DELAY
      MOV P2,#09H
      ACALL DELAY
      MOV P2,#0CH
      ACALL DELAY
      MOV P2,#06H
      ACALL DELAY
      AJMP LOOP2
  LOOP1:MOV P2,#06H       ; Switch is off, send step sequence for
                            anticlockwise
      ACALL DELAY
      MOV P2,#0CH
      ACALL DELAY
      MOV P2,#09H
      ACALL DELAY
       MOV P2,#03H
      ACALL DELAY
      AJMP LOOP2
  DELAY: MOV R5,#0FFH     ; Delay program
  DELAY1: MOV R7,#0FFH
   S1: DJNZ R7,S1
       DJNZ R5,DELAY1
       RET
       END
```

C PROGRAM

```
  #include <Intel\8051.h>
  #define phasea        0x03   // Data for stepper motor excitation
  #define phaseb        0x09   // Data for stepper motor excitation
```

```
#define phasec       0x0c   // Data for stepper motor excitation
#define phased       0x06   // Data for stepper motor excitation

void clockwise(void);
void anticlockwise(void);
void delayl(void);
int i;
unsigned char test;

void main ( )
{
//  sense switch

          P1 = 0xFF;
          P1 |= 0x40;
//Depending on the switch position, the stepper motor will rotate
clockwise or anticlockwise
          while(1)
          {

          test = P1;
          test = test & 0x40;

          if(test == 0x40)
          {
          clockwise();
          }
          else
          {
          anticlockwise();
          }
        }  // end of while(1)

}  // end of main()

// Function for clockwise rotation
// Excitation sequence for clockwise a b c d
void clockwise(void)
{
```

```
            P2 = phasea;                        delayl();
            delayl();

            P2 = phaseb;
            delayl();
            delayl();

            P2 = phasec;
            delayl();
            delayl();

            P2 = phased;
            delayl();
            delayl();
}

// Anticlockwise program
// Excitation sequence for anticlockwise d c b a

void anticlockwise(void)
{
            P2 = phased;
            delayl();
            delayl();

            P2 = phasec;
            delayl();
            delayl();

            P2 = phaseb;
            delayl();
            delayl();

            P2 = phasea;
            delayl();
            delayl();

}
// Function for delay in between steps
```

```
void delay1(void)
{
                              delay_ms(250);
//for(i=0;i<=10000;i++)
}
```

SECTION REVIEW

1. If step size is 1.8°, then to complete one rotation, it requires 200 steps. True/False?
2. If step size is 7.5°, then to complete two rotations in clockwise, it requires _____ steps.
3. If step size is 1.8° and stepper motor is operated with half step, then it requires _____ steps to complete one rotation.
4. The function of driving circuit in stepper motor is to amplify the current. True/False?
5. What is the function of fly back diodes in driving circuit?

6.10 INTERFACING DC MOTOR

EXAMPLE 6.28

Interface DC motor and write ALP and C program to rotate the DC motor for a given speed.

DC motor is used extensively in control system applications because the speed and torque can be precisely controlled. The difference between stepper motor and DC motor is that DC motor has a permanent magnetic field. When a voltage and subsequent current flow are applied to the armature, the motor begins to rotate. The DC voltage applied across the armature determines the speed of rotation. The circuit to interface DC motor is as shown in Fig. 6.16.

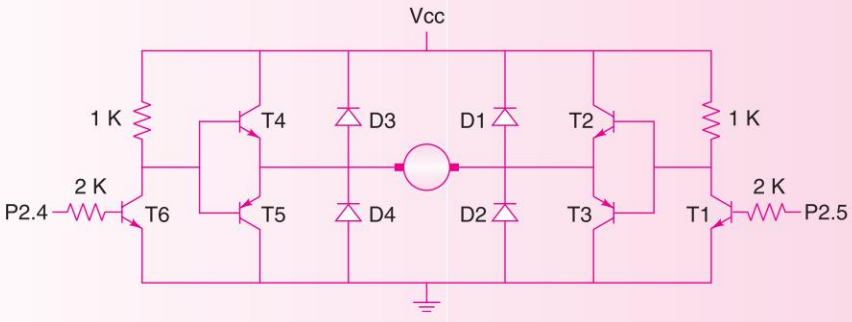


Figure 6.16 *Motor drive circuit*

DC motor speed and direction of rotation is controlled using port pins—P2.4 and P2.5. The circuit utilises complementary pair NPN/PNP, transistors T4/T5 and T2/T3. Motor is on, if a PWM signal is applied to P2.4; and if P2.5 is held at logic 0, when the PWM is at logic 1, T6 collector will be low; so T4 is OFF and T5 is ON. Since P2.5 is held at logic 0, the collector of T1 will go high, so T2 is ON and T3 is OFF. The motor rotates in reverse direction, if P2.4 is held at logic 0; and if a PWM signal is applied to P2.5, then T4 will be ON and T5 will be OFF. When PWM is at logic 1, transistor T2 is OFF and T3 is ON giving a reverse conduction path. Motor is OFF if both P2.4 and P2.5 are held at logic 0. The diodes in the figure are used to protect the transistors from reverse biases.

The program to control the speed of DC motor is as follows:

```
    ORG 0000H
    CLR  P2.5                ; Make P2.5 low
  NEXT:SETB P2.4             ; Make P2.4 high
    MOV R4, #0FH
    B1:MOV R5,#0FFH
    B2:MOV R7,#0FFH
    B3:DJNZ R7,B3
    DJNZ R5,B1
        DJNZ R4,B1
        CLR  P2.4            ; Make P2.4 low
        MOV R4, #05H
    B1:MOV R5,#05H
    B2:MOV R7,#05H
    B3:DJNZ R7,B3
        DJNZ R5,B1
        DJNZ R4,B1
        SJMP NEXT
        END
```

C PROGRAM

```
#include <Intel\8051.h>
#include<standard.h>
#define DTIME 10                // Delay for 10 ms
BIT out P2.4                    // Port for dc motor
void main ()
{
   while(1)
```

```
    {
        out=1;
        delay_ms(DTIME);        // Generate pulse of 10 ms
        out =0;
        delay_ms(DTIME);
    }
}
```

Note

All C Programs are executed by using SIDE 51 compiler (SPJ). To execute programs by using µvision compiler (KEIL Corp), few changes specific to the compiler need to be done.

```
replace header #include <Intel\8051.h> with  # include <reg51.h>
replace        bit disp1 P1.5 with sbit disp1= P1^5 or sbit
               disp1=0X95.
```

SECTION REVIEW

1. DC motor has _____ magnetic field.
2. In DC motor, the voltage applied across the armature determines the speed of rotation. True /False?
3. The speed of DC motor is proportional to armature voltage. True/False?
4. Mention the important differences between stepper and DC motor.
5. The speed of the DC motor is controlled by pulse width modulation. True/False?

CHAPTER SUMMARY

In this chapter, we have elucidated the following important concepts:

- Structure and functions of parallel ports—port 0, port 1, port 2 and port 3.
- Alternate functions of port 0, port 1 and port 3.
- Circuits and programs to interface real world devices such as LED, LCD, and seven-segment display, and keyboard.
- The programming and interfacing of ADC and DAC.
- Basic operations of stepper motor and DC motor.
- Circuits and programs to interface stepper motor with microcontroller

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. Which port has no alternate functions?
   (a) Port 0 (b) Port 1 (c) Port 2 (d) Port 3
2. The alternate function of Port 3.4 is _____.
   (a) Timer/counter 0 external input (b) Timer/counter 1 external input
   (c) Serial input port (d) Serial output port
3. Port 0 has _____.
   (a) No internal pull up resistance (b) Internal pull up resistance
   (c) No external pull up resistance (d) None of the above
4. The alternate function of Port 2 is _____.
   (a) For sending high order address (A8–A15) (b) For sending low order address (A0–A7)
   (c) For sending data (D0–D7) (d) None of the above
5. For read and write operations, strobe signals are obtained by using _____.
   (a) P3.0 and 3.1 (b) P3.2 and 3.3 (c) P3.4 and 3.5 (d) P3.6 and 3.7
6. DAC 08 output is connected to _____.
   (a) Current to voltage converter circuit (b) Amplifier
   (c) Attenuator circuit (d) None of the above
7. Consider 4 rows × 4 columns keypad, connected by using P0.0 to P0.3 and P0.4 to P0.7. The key values are 0 to F respectively. When key 0 is pressed, the values at port P0.7 to P0.4 values will be _____.
   (a) 1110 (b) 0111 (c) 1011 (d) 1101
8. To display 'A' using common cathode seven-segment display, the code required is _____.
   (a) 77H (b) 7FH (c) 79H (d) 06H
9. If the step angle is 1.8 degree for a stepper motor, then the number of steps required to complete half rotation would be _____.
   (a) 200 (b) 100 (c) 360 (d) 180
10. If the range of an 8 bit A/D converter is 0 to +5 V, then the step size is _____.
    (a) 19.53 mv (b) 2.5 V (c) 0.625V (d) 256 mv
11. The driver circuit for stepper motor functions as _____.
    (a) Buffer (b) Voltage amplifier (c) Attenuator (d) None of the above
12. Which of the following ports in 8051 does not require external pull-up resistors to function as an I/O port _____.
    (a) Port 1, 2 and 3 (b) Port 0, 1 and 2 (c) Port 0 and 2 (d) Port 0 and 3
13. The number of analog input channel for ADC 0809 is _____.
    (a) 8 (b) 9 (c) 1 (d) 4
14. ADC 0808 is an _____ bit converter.
    (a) 8 (b) 4 (c) 1 (d) 16
15. _____ Pin of ADC 0808 indicates that the analog signal is converted to digital.
    (a) EOC (b) SC (c) OE (d) ALE
16. DAC 08 is _____ digital to analog converter.
    (a) 8 bit (b) 1 bit (c) 4 bit (d) None of the above
17. The output pins of DAC 08 _____.
    (a) Sources the current (b) Sinks the current (c) Both a and b (d) None of the above

18. _____ value is sent to 0–10V DAC to generate 7.5 V.
(a) 192 (b) 238 (c) 128 (d) 255
19. Stepper motor rotates by one step when the _____ .
(a) Current is transferred from one coil to next coil.
(b) Current is switched OFF in the next coil.
(c) Current is switched ON in the next coil.
(d) When the current is switched ON in all the coils.
20. _____ ADC provides the best resolution.
(a) 4 bit (b) 8 bit (c) 12 bit (d) 16 bit

REVIEW QUESTIONS

6.1 With a diagram, explain the operation of port 0.
6.2 Explain the alternate functions of port 3.
6.3 Explain two types of reading operations in 8051 ports.
6.4 Explain the alternate functions of port 0 and port 2.
6.5 Write a program to implement binary up counter (count up from 00H to FFH) with a delay of 1 second. Output the count value to port 0.
6.6 Write a program to implement binary down counter (count down from FFH to 00H) with a delay of 0.5 second. Output the count value to port 1.
6.7 Write a program to implement decimal up counter (count up from 00 to 99) with a delay of 1 second. Output the count value to port 2.
6.8 Write a program to implement decimal down counter (count down from 99 to 00) with a delay of 1 second. Output the count value to port 2.
6.9 Write a program to toggle the bits of port 1 with a delay of 10 ms.
6.10 Write a program to generate a square wave of 50% duty cycle at bit 0 of port 1.
6.11 Write a program to display values from 00–49 four times and then stop.
6.12 Write a program to display 0 to 9 in a seven-segment display card, which contains four seven-segment displays.
6.13 Write a program to display ADC output value using an LCD.
6.14 Write a program to rotate stepper motor in clockwise direction to complete ten rotations and then rotate in anticlockwise direction.
6.15 Write a program to generate stair case waveform using 8 bit DAC.
6.16 Write a program to implement stopwatch.
6.17 Construct an 8051-based system to read 4 analog inputs one after the another and generate the following outputs.

If Input 1 > Input 2	output square wave of 2 KHz
(Input1 + Input 2) > Input 3	Saw tooth wave of 500 Hz.
(Input1+Input2+Input3) > Input 4	Pulse output with 75% duty cycle
Otherwise	out constant DC

Select suitable ADC and DAC.
6.18 Write a program to implement real time clock using an LCD.
6.19 Write a program to rotate DC motor with different speeds in clockwise direction.
6.20 Interface an 8 bit DAC to the 8051 microcontroller and write a program to generate negative ramp waveform.

7

8051 INTERRUPTS AND TIMERS/COUNTERS

Learning Objectives

After you have completed this chapter, you should be able to

- Explain interrupts and their classification
- Explain 8051 interrupt structure
- Understand the timers/counters of the 8051
- Comprehend the operation of Timer 0 and Timer 1 in various modes
- Write programs for timers/counters

7.1 BASICS OF INTERRUPTS

Interrupt is an input to a processor, whereby an external device or a peripheral can inform the processor that it is ready for communication. When peripheral devices activate an interrupt signal, the processor branches to a program called *interrupt service routine*. This program is written by the user for performing tasks that the interrupting device wants the processor to execute. After executing the interrupt service routine, the processor returns to the main program as shown in Fig. 7.1.

When peripheral devices interrupt the processor, branching takes place to interrupt service subroutine. Before branching, the actions taken by the processor are as follows:

1. It completes the execution of current instruction.
2. Program status word register value is pushed onto the stack.
3. Program counter value is pushed onto the stack.
4. Interrupt flag is reset.
5. Program counter is loaded with Interrupt Service Subroutine (ISS) address.

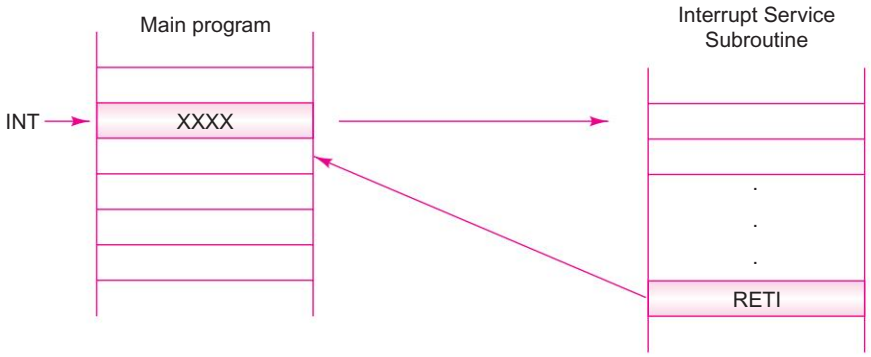


Figure 7.1 *Interrupt service subroutine processing*

When program counter is loaded with interrupt service subroutine address, branching to ISS takes place. During execution of ISS, interrupts are disabled because interrupt flag is reset. The ISS is ended with RETI instruction. The execution of RETI instruction results in the following:

1. POP from the stack top to the program counter.
2. POP from the stack top to program status word register.

Thus, after executing the interrupt service routine, processor returns to the main program with program status word register value unchanged.

7.1.1 CLASSIFICATION OF INTERRUPTS

Interrupt is classified into two types—*external and internal interrupt*. Peripheral devices via the microcontroller interrupt pins initiate external interrupts. Internal interrupts are activated by the peripherals of the microcontroller and by the execution of interrupt instructions.

7.1.2 INTERRUPT MASKABILITY

Interrupt requests are classified into two categories—*maskable and non-maskable interrupts*. Microprocessors and microcontrollers have the option to disable the interrupts. These types of interrupts are called maskable interrupts. Other types of interrupts, which the processor cannot disable, are called non-maskable interrupts.

7.1.3 INTERRUPT VECTOR

The term 'interrupt vector' refers to the starting address of the interrupt service routine. The processor needs to determine the starting address of the interrupt service routine before it can provide service. The interrupt vector can be determined by one of the following methods:

Vectored Interrupts In this method, the starting address of the interrupt service routine is predefined when the microcontroller is designed. These types of interrupts are called vectored interrupts.

Non-vectored Interrupts In this method, when the microcontroller receives the interrupt signal from the external devices, the processor completes the current instruction and sends a signal called $\overline{\text{INTA}}$ interrupt acknowledge (active low). After receiving the $\overline{\text{INTA}}$ signal, external hardware sends the interrupt vector to the microcontroller. These types of interrupts are called *non-vectored interrupts*.

7.2 8051 INTERRUPT STRUCTURE

The 8051 provides five vectored interrupt sources. There are two external interrupts, external interrupt 0 ($\overline{\text{INT0}}$) and external interrupt 1 ($\overline{\text{INT1}}$) and three internal interrupts, 2 timer interrupt and a serial port interrupt. When an interrupt is generated, the contents of pogram status word register and program counter (PC) are pushed onto the stack. Vector address as shown in Table 7.1 is loaded in the program counter. As it branches to interrupt service routine, the interrupt flag of that particular interrupt is cleared by the hardware. In 8051, the interrupt flags are IE0, IE1, TF0, TF1, RI and TI. The interrupt service routine is ended with RETI instruction. The RETI instruction will POP from the stack top to the program counter and program status word register, and set the interrupt flag of that particular interrupt. The processor will start executing from where the main program was interrupted.

TABLE 7.1 *Interrupt service routines*

Interrupt Source	Vector Address	Priority within Level
External Interrupt 0	0003H	Highest
Timer 0 Interrupt	000BH	↑
External Interrupt 1	0013H	
Timer 1 Interrupt	001BH	
Serial Port Interrupt	0023H	Lowest

Interrupt Enable Each of the interrupt sources can be individually enabled or disabled by setting or clearing a bit in Interrupt Enable (IE) register. Figure 7.2 shows the IE register in the 8051. This register also contains a global disable bit, which can be cleared to disable all the interrupts.

Interrupt Priority Priority of the interrupt can be programmed by setting or clearing a bit in the Interrupt Priority (IP) register. When more than one interrupt is enabled, the user first programs the interrupt priority register. Figure 7.3 shows the Interrupt Priority (IP) register in the 8051.

If two interrupt requests of different priority levels are received simultaneously, the request of highest priority level is serviced. If interrupt requests of same priority levels are received simultaneously, an internal polling sequence as shown in Table 7.1, determines which interrupt should be serviced first. Thus, within each priority level there is a second priority structure determined by the polling sequence.

MSB							LSB
EA	–	–	ES	ET1	EX1	ET0	EX0

Enable bit = 1 enables the interrupt
Enable bit = 0 disables the interrupt

Symbol	Position	Function
EA	IE.7	If EA = 0, disables all interrupts If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit
—	IE.6	reserved*
—	IE.5	reserved*
ES	IE.4	Serial Port Interrupt Enable Bit
ET1	IE.3	Timer 1 Overflow Interrupt Enable Bit
EX1	IE.2	External Interrupt Enable Bit
ET0	IE.1	Timer 0 Overflow Interrupt Enable Bit
EX0	IE.0	External Interrupt 0 Enable Bit

*These reserved bits are used in other MCS-51 devices.

Figure 7.2 *IE (Interrupt Enable) Register in 8051*

MSB							LSB
–	–	–	PS	PT1	PX1	PT0	PX0

Priority bit = 1 Assigns high priority
Priority bit = 0 Assigns low priority

Symbol	Position	Function
—	IP.7	reserved*
—	IP.6	reserved*
—	IP.5	reserved*
PS	IP.4	Serial Port Interrupt Priority Bit
PT1	IP.3	Timer 1 Interrupt Priority Bit
PX1	IP.2	External Interrupt 1 Priority Bit
PT0	IP.1	Timer 0 Interrupt Priority Bit
PX0	IP.0	External Interrupt 0 Priority Bit

*These reserved bits are used in other MCS-51 devices.

Figure 7.3 *(Interrupt Priority) Register in 8051*

SECTION REVIEW

1. The 8051 has 5 vectored interrupt sources. True/False?
2. Differentiate vectored and non-vectored interrupts.
3. Differentiate maskable and non-maskable interrupts.
4. List the steps taken by the processor after receiving the interrupt.
5. List the functions of RETI instructions.
6. The 8051 contains two external interrupts and 3 internal interrupts. True/False?
7. External interrupt 0 has the highest priority. True/False?
8. When the 8051 receives interrupt through $\overline{\text{INT1}}$ pin, it branches to _____ address.
9. EX0 bit of IE register enables external interrupt 0. True/False?
10. The 8051 has _____ priority levels in interrupts.

7.3 TIMERS AND COUNTERS

In a microcontroller, physical time is represented by the count value of a timer. By interpreting the count value of a timer properly, many timer applications can be realised. There are many applications that require a dedicated timer system, including

- Time reference
- Time delay creation
- Pulse width, period and frequency measurement
- Periodic interrupt generation (to remind the processor to perform routine tasks)
- Waveform generation

A basic timer consists of a register that can be read from or written to by the processor and is driven by some frequency source. The register is usually of 8 or 16 bit. The timer/counter clock source is either microcontrollers clock or an external clock. The value of the register can be read or a new value can be written during the processor operation. When the counter overflows (in 8 bit—FF to 00), an interrupt is generated.

7.4 8051 TIMERS/COUNTERS

The 8051 has two timers, Timer 0 and Timer 1. Both are 16 bit timers/counters. Both timers consist of two 8 bit registers. Timer 0 consists of two 8 bit registers—TH0 and TL0 and Timer 1 consists of two 8 bit registers, TH1 and TL1. Both share the timer control (TCON) register and timer mode register (TMOD). The timer registers are as shown in Fig. 7.4.

A timer can be used to create time delay, or as a counter to count the external events happening outside the micrcontroller that occurred within a time interval. The only difference between counter and timer is the source of the clock pulse as shown in Fig. 7.5. The C/$\overline{\text{T}}$ bit in the TMOD register decides the source of the clock for the timer. If C/$\overline{\text{T}}$ = 1, then timer is used as counter and gets the clock from outside the microcontroller, else if C/$\overline{\text{T}}$ = 0, then timer gets the clock pulses from the crystal.

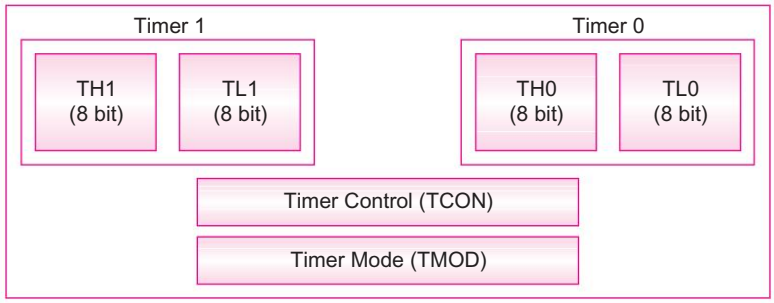


Figure 7.4 *Timer registers*

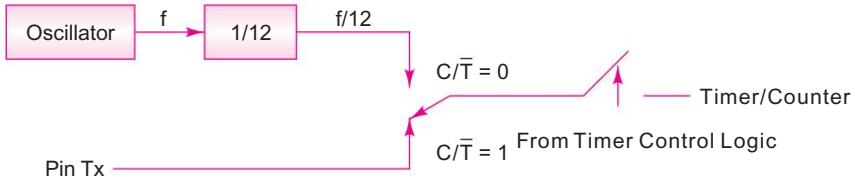


Figure. 7.5 *Timer or Counter operation*

When used as a timer, the clock pulse is obtained from the oscillator through divide by 12 d circuit. When used as counter, pin T0 (P3.4) supplies pulses to counter 0, and pin T1 (P3.5) supplies pulses to counter 1. If 12 MHz crystal is connected to the 8051, then the clock period will be equal to 1 μs. Registers are incremented after 1 μs. In the counter function, the registers are incremented in response to a transition from 1 to 0, at external input pin P3.4 for counter 0 and P3.5 for counter 1. When the registers overflow from FFFF h to 0000 h, it sets a flag (TF flag) and generates an interrupt. Timer control register controls the timer/counter operation. Figure 7.6 shows TCON register. This register is an 8 bit register.

Bit 7 Bit 0

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	Timer 1 overflow flag set when timer/counter overflows
TR1	Timer 1 run control bit
TF0	Timer 0 overflow flag set when timer/counter overflows
TR0	Timer 0 run control bit
IE1	Interrupt 1
IT1	Timer 1 interrupt, ITI = 0 low level triggered, ITI = 1 falling edge trigger
IE0	Interrupt 0
IT0	Timer 0 interrupt, IT0 = 0 low level triggered, IT0 = 1 falling edge trigger

Figure 7.6 *Timer Control Register (TCON)*

TR0 and TR1 flags turn the timer ON or OFF. The upper 4 bit are used to store the TF and TR bits of both Timer 0 and Timer 1. The lower 4 bit are used for controlling the interrupt. TCON is a bit addressable register. Instructions SETB TCON.6 and CLR TCON.6 can be used to store 1 and 0 in TR1 flag of TCON register. Figure 7.7 shows Timer mode control register (TMOD). The lower 4 bit are used for controlling Timer 0 and the upper 4 bit are used for controlling Timer 1. C/$\overline{T}$ selects timer or counter operation and M1 and M0 select the mode.

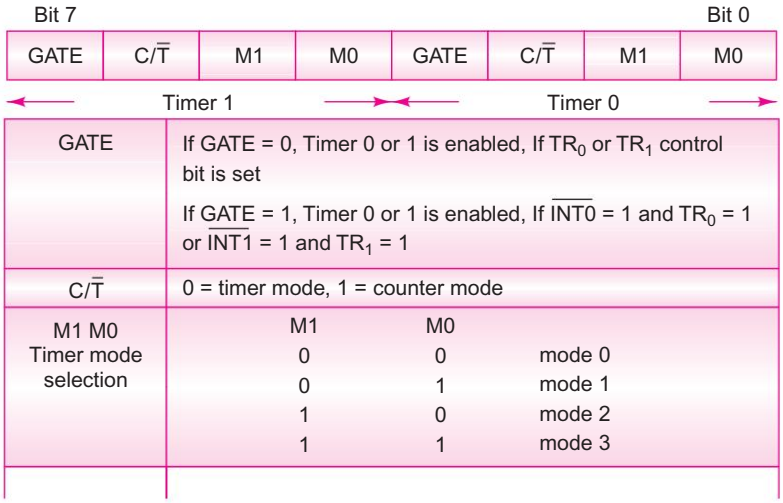


Figure 7.7 *Timer Mode Control Register (TMOD)*

EXAMPLE 7.1

If an 8051-based system is controlled by the following crystal frequencies, find the timer clock frequency and its period.

1. 12 MHz
2. 18 MHz
3. 11.0592 MHz

SOLUTION

1. Timer clock frequencies = 1/12 × 12 MHz = 1 MHz
   Period T = 1/1MHz = 1 μs.
2. Timer clock frequencies = 1/12 × 18 MHz = 1.5 MHz
   Period T = 1/1.5 MHz = 0.667 μs.
3. Timer clock frequencies = 1/12 × 11.0592 MHz = 921.6 KHz
   Period T = 1/921.6 KHz = 01.085 μs

7.5 TIMER/COUNTER OPERATION MODES

In addition to the timer or counter selection, both timer 0 and timer 1 have operating modes. Modes are selected by using Timer Mode Control Register (TMOD). Figure 7.7 shows TMOD register. The timers may operate in any one of the four modes that are determined by mode bits, M1 and M0, in TMOD register.

MODE 0 Figure 7.8 shows the mode 0 operation for timer 0 and timer 1. In this mode, timer register is configured as a 13 bit register. The 13 bit timer register consists of all 8 bit of THx and the lower 5 bit of TLx register. The upper 3 bit of TLx register are ignored. As the counter changes from all 1's to 0's, it sets the timer interrupt flag, TFx. The timer is enabled when TRx = 1, and either gate = 0 or INTx = 1.13 bit registers can hold values between 0000H to 1FFFH in TH and TL. When the register reaches its maximum 1FFFH, it rolls over to 0000H i.e. when the register overflows, then it sets the timer interrupt flag (TF1 for timer 1 and TF0 for timer 0). Example 7.2 gives the program to initialise timer 0 in mode 0.

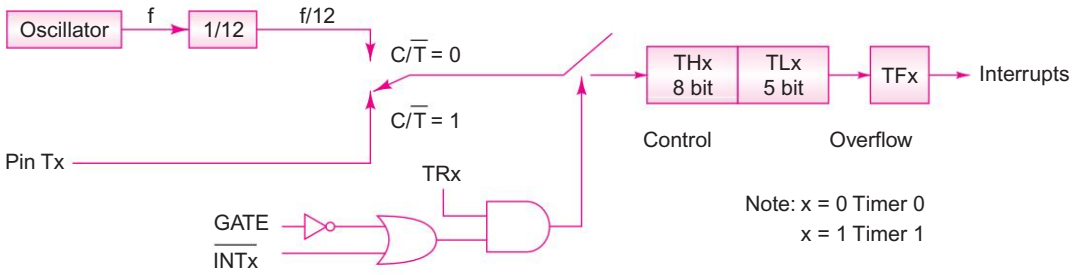


Figure 7.8 *Timer/Counter 0/1 in mode 0*

For mode 0, loading TLx and THx with values derived from the formula can create a specific time delay.

Time delay = [12 × [5110 – Init Value]]/ Freq

Where Init Value = TLx + [256 × THx]

Where the values of THx and TLx are in decimal, the values must be converted to hexadecimal, then it must be loaded to THx and TLx registers.

EXAMPLE 7.2

Write 8051 program to initialise Timer 0 and Timer 1 in mode 0. The external pin 12 (INT0) controls Timer 0, and Timer 1 is fully controlled by TR1.

SOLUTION

```
MOV TMOD,#08h ; Timer 0 and 1 in mode 0 and timer 0 is controlled by
              pin INT 0
SETB TR1      ; Start timer 1
SETB TR0      ; Start timer 0
END
```

Mode 1 Mode 1 is the same as mode 0 except that the timer register uses all 16 bit. In this mode, THx and TLx are cascaded. For mode 1, loading TLx and THx with values derived from the formula can create a specific time delay.

Time delay = [12 × [65,536 – Init Value]]/ Freq

where Init Value = TLx + [256 × THx]

If the values of THx and TLx are in decimal, the values must be converted to Hexadecimal, then it must be loaded to THx and TLx registers.

Mode 2 The operation is same for timer 0 and timer 1. In this mode, timer register is configured as TLx. When TLx overflows from FFh to 00H it, sets the flag TFx and it loads TLx with the contents of THx. The reload leaves THx unchanged. In this mode, timer 1or timer 0 supports the automatic reload operation. The timer control logic is same as mode 0 or mode 1 as shown in Fig. 7.9. Example 7.3 gives the program to initialise timer 0 in mode 2.

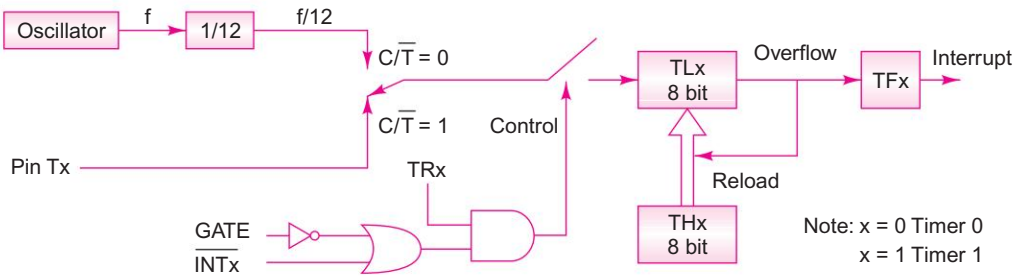


Figure 7.9 *Timer/Counter 0/1 in mode 2:8 bit auto reload*

EXAMPLE 7.3

Write 8051 program to initialise Timer 0 in mode 2. Load TH0 with preset value, 55H and load TL0 with starter value, 55H.

SOLUTION

```
MOV TMOD, #02h    ; Load TMOD to operate timer 0 in mode 2
MOV TH0,#55h      ; Load TH0 with preset value to be reloaded
MOV TL0,#55h      ; Load TL0 with starting value = preset value
SETB TR0          ; Start timer 0
END
```

Mode 3 Timer 0 in mode 3 establishes TL0 and TH0 as two separate registers as shown in Fig. 7.10. TL0 uses Gate, TR0, INT0 and TF0 control bits of timer 0 and TH0 uses TR1 and TF1 control bits of timer 1.

When TL0 overflows from FF to 00, then it sets the timer flag TF0. If control bit TR1 is set, TH0 receives the timer clock (oscillator divided by 12). When the counter TH0 overflows from FF to 00, then it sets the flag TF1.

Timer 1 may still be used in mode 0, mode 1 and mode 2, but it neither interrupts the processor nor sets the flag. When timer 0 is in mode 3, timer 1 can be used for baud rate generation in serial communication.

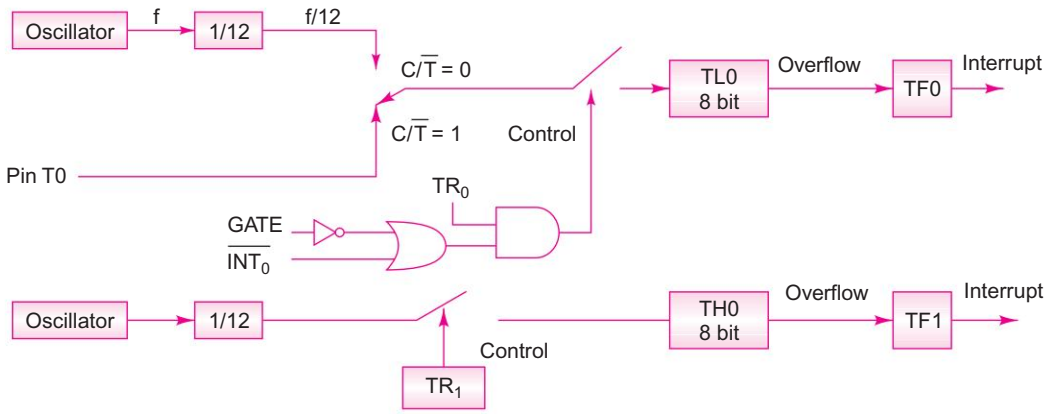


Figure 7.10 *Timer/Counter 0/1 in mode 3:Two 8 bit counters*

SECTION REVIEW

1. List the applications that require 8051 timers.
2. The 8051 has _____ number of timers.
3. Timer 0 and 1 are 16 bit timer/counter. True/False?
4. Timer 0 and 1 share TCON register and TMOD register. True/False?
5. If C/$\overline{T}$= 0, then the timer gets the clock pulses from outside the microcontroller. True/False?
6. _____ register controls the timer/counter operation.
7. What is the significance of IT1 bit of TCON register?
8. List the special function registers of timer/counter.
9. TCON is a bit addressable register. True/False?
10. _____ bit selects timer/counter operation.
11. Timer 1 is enabled if gate = 0 and TR1 bit is set. True/False?
12. Timer 0 is enabled if gate = 1, _____ and _____ bit are set.
13. If an 8051 is operated with 18 MHz crystal frequency, then the timer clock frequency is _____ MHz.
14. If an 8051 system is operated with 11.0925 MHz crystal frequency, then the timer clock period is 1.085 µs. True/False?
15. In mode 0, timer register is configured as _____ bit register.
16. In mode 1, timer register is configured as 16-bit register. True/False?
17. In timer/counter, which mode supports automatic reload operation?
18. In mode 3, when TH0 register overflows, _____ timer flag is set.
19. Timer/counter operates in _____ number of modes.
20. Differentiate timer/counter operations.

7.6 PROGRAMMING 8051 TIMERS

EXAMPLE 7.4

Assume an oscillator running at 12 MHz controls an 8051 micrcontroller. Write a subroutine to create a time delay of 20 ms.

SOLUTION

Step 1: If C/$\overline{T}$ = 0, then clock source to timer 1 = oscillator /12
= 12 MHz/12 = 1 MHz.
Step 2: For 20 ms, 20 ms/1 µs = 20,000.
Step 3: Timer register (TH1 and TL1) value = (65,536-20,000) = 45,536 d
= B1E0 H.

Algorithm to create a delay of 20 ms is as follows:

Step 1: Configure timer 1 to operate in mode 1 and choose oscillator/12 as the clock input (C/$\overline{T}$ = 0), Gate = 0 and TF1 = 0.
Step 2: Place value B1E0h, into timer 1 register and wait until the overflow flag is set to 1.

PROGRAM

```
Main Program        Subroutine
•              DELAY: MOV TMOD,#10h    ; Set up timer 1 in mode 1
•                     CLR TF1          ; Clear timer 1 overflow flag
•                     CLR ET1          ; Clear timer 1 interrupt flag
LCALL DELAY           MOV TH1, #B1h    ; Store upper byte of count
•                     MOV TL1, #E0h    ; Store lower byte of count
                      SETB TR1         ; Enable timer 1
•                  WAIT: JNB TF1, WAIT ; Wait until TF1 is set to 1
                      RET
```

EXAMPLE 7.5

Assume an oscillator running at 12 MHz controls an 8051 micrcontroller. Write a subroutine to create a time delay of 1 second.

SOLUTION

Step 1: If C/$\overline{T}$ = 0, then clock source to timer 1 = oscillator/12 = 12 MHz/12 = 1 MHz.
Step 2: For 50 ms, 50 ms/1 µs = 50,000.

Step 3: Timer register (TH1 and TL1) value = (65,536-50,000) = 15,536 d = 3CAF H.

Step 4: For 1 second, 1 second/50 ms = 20, if it is repeated 20 d times, we get 1 second.

Algorithm to create a delay of 1 second is as follows:

Step 1: Configure timer 1 to operate in mode 1 and choose oscillator/12 as the the clock input ($C/\overline{T}$ = 0), Gate = 0 and TF1 = 0.

Step 2: Place value 3CAF into timer 1 register and wait until the overflow flag is set to 1.

Step 3: Repeat step 20d (14 h) times.

PROGRAM

```
   Main Program          Subroutine
  •                DELAY1: MOV R1, #14h ; Repeat the delay 20 times
                   MOV TMOD, #10h       ; Setup timer 1 in mode 1
                   CLR ET1              ; Disable timer 1 interrupt
                 LOOP2: CLR TF1         ; Clear timer 1 overflow flag
  LCALL DELAY1   MOV TH1, #3C h         ; Place 15536 into timer register
  •                MOV TL1, #0AF h
                   SETB TR1             ; Enable timer 1 to operate
  •              WAIT: JNB TF1, WAIT    ; Wait until TF1 is set
                   DJNZ R1, LOOP2
                   RET
```

EXAMPLE 7.6

Assume an oscillator running at 12 MHz controls an 8051 micrcontroller. Write a program to generate 2 KHz square wave on port 1.0 (pin 1).

SOLUTION

To generate a square wave on port 1.0 (pin 1) is as follows

Step 1: If $C/\overline{T}$ = 0, then clock source to timer 1 = oscillator/12 = 12 MHz/12 = 1 MHz.

Step 2: Period of square wave: T = 1/f = 1/2KHz = 500 µs.

Step 3: Duration of high and low portion of the pulse is = 250 µs.

Step 4: Timer register value is: 250 µs/1 µs = 250 and (65,536 – 250) = 65,286 d = FF06 h is loaded to TH and TL register.

PROGRAM

```
MOV TMOD,#10h        ; Setup timer 1 in mode 1
CLR ET1              ; Disable timer 1 interrupt
LOOP: CLR TF1        ; Clear timer 1 overflow flag
```

```
MOV TH1, #0FFh        ; Place 65,286 into timer register
MOV TL1, #06h
SETB TR1              ; Enable timer 1 to operate
WAIT: JNB TF1, WAIT   ; Wait until TF1 is set
CLR TR1               ; Stop timer 1
CPL P1.0              ; Complement P1.0 to get high and low
SJMP LOOP
```

EXAMPLE 7.7

Assume an oscillator running at 12 MHz controls an 8051 micrcontroller. Write a program to generate 4 KHz square wave on port 1.2 (pin 3) using timer 0 in auto reload mode.

SOLUTION

To generate a square wave on port 1.2 (pin 3) is as follows

Step 1: If $C/\overline{T}$ = 0, then clock source to timer 1 = oscillator/12 = 12 MHz/12 = 1 MHz.

Step 2: Period of square wave: T = 1/f = 1/4KHz = 250 µs.

Step 3: Duration of high and low portion of the pulse = 125 µs.

Step 4: Timer register value is : 125 µs/1µs = 125 and (256 – 125) = 131 d (83 h) is loaded to TH0 and TL0 register .

PROGRAM

```
  MOV TMOD, #02h      ; Setup timer 0 in auto reload mode
  CLR TF0             ; Clear timer 0 overflow flag
  CLR ET0             ; Disable timer 0 interrupt
  LOOP:MOV TH0, #83h  ; Place 83 h into timer register
  MOV TL0, #83h
  SETB TR0            ; Enable timer 0 to operate
  BACK: JNB TF0, BACK ; Wait until TF0 is set
  CLR TR0             ; Stop timer 1
  CPL P1.2            ;     Complement P1.2 to get high and low
  CLR TF0             ; Clear timer flag1
SJMP BACK
```

EXAMPLE 7.8

Assume an oscillator running at 11.0592 MHz controls an 8051 microcontroller. Write a program to generate 2 KHz square wave on port 1.3 (pin 4) using timer 0 interrupt.

SOLUTION

To generate a square wave on pin 4 of port 1.3 is as follows

Step 1: If $C/\overline{T}$ = 0, then clock source to timer 1 = oscillator/12 = 11.0592 MHz/12 = 0.9216 MHz.

Step 2: Period of square wave: T = 1/f = 1/2KHz = 500 µs.

Step 3: Duration of high and low portion of the pulse = 250 µs.

Step 4: Timer register value is: 250 µs/1.085 µs = 230 and (65,536 – 230)= 65,306 d = FF1A h is loaded to TH and TL register.

PROGRAM

```
  ORG 0000
  LJMP MAIN                    ; Bypass interrupt vector table
  ORG 000B                     ; ISR for timer 0
  CLR TR0                      ; Disable timer 0
  CPL P1.3                     ; Complement P1.3 to get high and low
  MOV TL0,#1Ah
  MOV TH0,#0FFh
  RETI                         ; Return from interrupt service routine
  ORG 0030
  MAIN: MOV TMOD, #00000001B ; Timer 0 in mode 1
     MOV TLO,#1Ah              ; Place 65306 into timer register
     MOV TH0,#0FFh
     CLR TF0                   ; Clear TF flag
     MOV IE,#82h               ; Enable timer 0 interrupt
  LOOP1:SETB TR0               ; Start timer 0
  HERE: SJMP HERE
     AJMP LOOP1
```

EXAMPLE 7.9

Assume an oscillator running at 11.0592 MHz controls an 8051 micrcontroller. Write a program to generate 1 KHz square wave from port 1.1 using timer 0.

SOLUTION

To generate a square wave from port 1.1 is as follows

Step 1: If $C/\overline{T}$ = 0 then clock source to timer 1 = oscillator /12 = 11.0592 MHz/12 = 0.9216 MHz.

```
Step 2: Period of square wave: T = 1/f = 1/1 KHz = 1 ms.
Step 3: Duration of high and low portion of the square wave = 500 µs.
Step 4: Timer register value is: 500 µs/1.085 µs = 461 (to nearest
        whole number) and ( 65,536 - 461 ) = 65,075 d = FF1A h is
        loaded to TH and  TL register.
```

C PROGRAM

```
#include <Intel\8051.h>
#define on 1
#define off 0
bit Squarewavepin =P1.1; //Pin 1 of port 1
void delay1KHz( ) ;//Delay on( ) returns nothing and takes nothing

main( ){            // Start the program
  TMOD = 0x01;      // Timer 0 : Gate = 0, C/T = 0, M1 = 0, M0 = 1; mode 1
  While (1){                // Do for ever
    Squarewavepin = on;          // P1.1 set to 1
    delay 1KHz ( );          // Wait for on time
    Squarewavepin = off;     // P1.1 set to 0
    delay 1KHz ( );     // Wait for off time
      }                      // While ( )
  }                          // Main ( )
void delay1KHz ( ){
    TH0 =0X FF;
    TL0 = 0X1A ;
    TR0 = on;           //Set TR0 of TCON to run timer 0
    While (!TF0);       //Wait for timer 0 to set the flag TF0?
    TR0 = off;          //Stop the timer 0
    TF0 = off;          //Clear the TF0
  }
                        //Delay ( )
```

EXAMPLE 7.10

Assume an oscillator running at 11.0592 MHz controls an 8051 microcontroller. Write a program to generate 5 KHz square wave from port 1.5 using timer 1.

SOLUTION

Solution to generate a square wave from port 1.5 is as follows

Step 1: If C/$\overline{T}$ = 0, then clock source to timer 1 = oscillator /12 = 11.0592 MHz/12 = 0.9216 MHz.

Step 2: Period of square wave: T = 1/f = 1/5 KHz = 0.2 ms.

Step 3: Duration of high and low portion of the square wave = 0.1 ms.

Step 4: Timer register value is: 100 µs/1.085 µs = 92 (to nearest whole number) and (65,536 – 92) = 65,444 d = FF5A h is loaded to TH and TL register.

C PROGRAM

```
#include <Intel\8051.h>
#define on 1
#define off 0
bit Squarewavepin P1.5; //Pin 5 of port1
void delay1KHz( );      //Delay on( ) returns nothing and takes
                        nothing
   main( ){             // Start the program
   TMOD = 0100; // Timer0 : Gate = 0, C/T =0, M1 = 0, M0 = 1; mode 1
      While (1){               // Do for ever
        Squarewavepin = on;        // P1.5 set to 1
        delay 5KHz ( );        // Wait for on time
        Squarewavepin = off;       // P1.5 set to 0
        delay 5KHz ( );        // Wait for off time
          }                        // While ( )
       }                           // Main ( )
void delay5KHz ( )
          {
        TH1 = 0XFF ;
        TL1 = 0X5A ;
        TR1 = on;                  //Set TR0 of TCON to run timer 1
        While (!TF1);              //Wait for timer 1 to set the
                                   flag TF1
        TR1 = off;                 //Stop the timer 1
        TF1 = off;                 //Clear the TF1
        }                              //Delay ( )
```

CHAPTER SUMMARY

In this chapter, we have discussed the interrupts and timers/counters. There are two 16 bit timers in the 8051. Timers support 4 different modes. This chapter covers applications of interrupts and timers/counters with relevant examples.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. _____ memory address in the interrupt vector table is assigned to INT0.
   (a) 000BH (b) 0003H (c) 0013H (d) 001BH
2. The 8051 has _____ external interrupt(s).
   (a) 1 (b) 2 (c) 3 (d) 4
3. Which is the highest priority interrupt in the 8051?
   (a) Ext. Int. 0 (b) Timer 0 interrupt (c) Serial port interrupt (d) Ext. Int. 1
4. Which is the lowest priority interrupt in the 8051?
   (a) Ext. Int. 0 (b) Timer 0 interrupt (c) Serial port interrupts (d) Ext. Int. 1
5. _____ memory address in the interrupt vector table is assigned to Timer 0.
   (a) 000BH (b) 0023H (c) 0003H (d) 001BH
6. Which bit of the IE register is used to set/reset the timer 0 overflow interrupt?
   (a) bit 0 (b) bit 1 (c) bit 2 (d) bit 3
7. Which port of the 8051 is used as external interrupt 0 (INT0)?
   (a) P3.2 (b) P3.4 (c) P3.3 (d) P3.5
8. Bit EA enables _____.
   (a) All maskable interrupts (b) Only timer interrupts
   (c) Only external interrupt (d) Only serial interrupt
9. Polling of next ISR to be executed is done _____.
   (a) After completion of each ISR instruction (b) After completion of RETI instruction
   (c) At all instances (d) None of the above
10. Number of registers associated with Timer 0 and 1 are _____.
    (a) 2 (b) 4 (c) 6 (d) 8
11. _____ timer register is bit addressable.
    (a) TH0 (b) TH1 (c) TCON (d) TMOD
12. If an 8051-based system is controlled by a crystal frequency of 18 MHz, then timer clock frequency is _____.
    (a) 1 MHz (b) 1.5 MHz (c) 18 MHz (d) 1.8 MHz

13. If an 8051-based system is controlled by a crystal frequency of 12 MHz, then timer clock period is.
    (a) 1 µs (b) 0.5 µs (c) 12 µs (d) 6 µs
14. Timers of the 8051 operate in _____ as 8 bit auto reload mode.
    (a) Mode 0 (b) Mode 1 (c) Mode 2 (d) Mode 3
15. The 8051 contains _____ internal interrupts.
    (a) Three (b) Two (c) Five (d) None of the above
16. The 8051 contains _____ external interrupts.
    (a) Three (b) Two (c) Five (d) None of the above
17. In mode 0, timer 1 register overflows when the register reaches _____ .
    (a) 1FFF (b) FFFF (c) FF (d) None of the above
18. Timer 0 in mode 3 uses _____ .
    (a) TL0 and TH0 as two separate registers (b) only TLO register
    (c) Only THO register (d) TL0 and TH0 as single register
19. Timer 0 interrupt can be programmed as _____ .
    (a) Low level triggered or falling edge triggered (b) Only low level triggered
    (c) Only falling edge triggered (d) None of the above
20. Reset pin acts as _____ .
    (a) Non-maskable vectored interrupt (b) Non-maskable, non-vectored interrupt
    (c) Maskable vectored interrupt (d) Maskable, non-vectored interrupt

REVIEW QUESTIONS

7.1 What is an interrupt? Explain interrupt structure of the 8051 microcontroller.

7.2 Distinguish between
    (a) Maskable and non-maskable interrupts
    (b) Vectored and non-vectored interrupts

7.3 What are the various SFRs required to use 8051 interrupts?

7.4 Explain two level interrupt priority. If two requests of interrupts are received simultaneously, how are these handled by the 8051?

7.5 Discuss the various timer modes supported by the 8051 and write a program to initialise timer 0 in auto reload mode.

7.6 Explain the functions of two hardware interrupt pins—INT0 and INT1 of the 8051 with its concerned registers. Give an application which uses these pins.

7.7 How does the 8051 determine which interrupt to service when there are several pending interrupts?

7.8 What is the last instruction in most interrupt service routines? What does this instruction do?

7.9 List the various interrupts of the 8051 and their corresponding vector addresses.

7.10 Assume IP register is set by the instruction MOV IP,#00001100B. Discuss the sequence in which the interrupts are serviced.

7.11 Write a program using INT0 to receive data from port 2 and send it to port 1.

7.12 List the features of timer 0 and timer 1.

7.13 Distinguish counting and timing requirements. Explain the modes of operation of timer/counter of the 8051 with a diagram.

7.14 Write a program using timer 1 for turning ON and OFF the LED connected to P0.5 every second.

7.15 Write a program using timer 0 for generating a square wave of 2 KHz.

7.16 Write the instructions to

(a) Enable timer 1 interrupt and external hardware interrupt 1

(b) Disable timer 1 interrupt

(c) To disable all the interrupts

7.17 Explain the functions of TCON.0, TCON.1, TCON.2, and TCON.3 in the execution of external interrupt 0 and 1.

7.18 Explain the operation of timer 1 in interrupt mode with the necessary program.

7.19 Generate a square wave with frequency 5 KHz on port 1.0 using timer 1.

7.20 Write a program to implement up counter 00–99. The counter is implemented after every one second and the delay program is implemented by using timer 1.

8

8051 SERIAL COMMUNICATION

Learning Objectives

After you have completed this chapter, you should be able to

- Define parallel and serial data transfer
- Define different types of communication links
- Define synchronous and asynchronous serial communication
- Explain the operation of serial port of the 8051 microcontroller
- Program the serial port for data transfer
- Explain the RS232 bus and function of the MAX232 IC

8.1 DATA COMMUNICATION

The 8051 microcontroller is a parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices. The parallel I/O devices are printer, D/A converter and stepper motor. Interfacing of parallel I/O devices has been discussed in Chapter 6. However, in many situations, parallel data transfer is impractical. For example, parallel data transfer over a long distance is very expensive. Hence, serial data communication is widely used in long distance data communication, and in this mode, one bit of information, at a time is transferred over a single line.

Figure 8.1 *Serial and parallel data transfer*

8.2 BASICS OF SERIAL DATA COMMUNICATION

In serial data communication, 8 bit data is converted to serial bits using a parallel in serial out shift register, and then it is transmitted over a single data line. The data byte is always transmitted with least significant bit first.

8.2.1 COMMUNICATION LINKS

Serial communication is classified into three types of communication links as shown in Fig. 8.2.

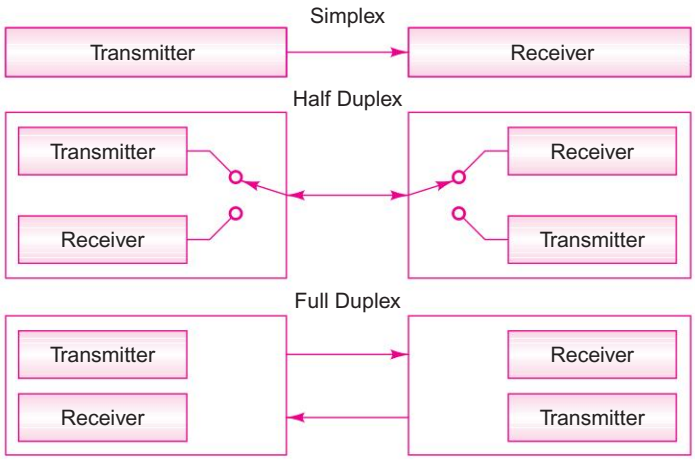


Figure 8.2 *Simplex, Half duplex and Full duplex data transfer*

Simplex In simplex transmission, the line is dedicated for transmission. The transmitter sends and the receiver receives the data..

Half duplex In half duplex, the communication link can be used for either transmission or reception. Data is transmitted in only one direction at a time.

Full duplex If the data is transmitted in both ways at the same time, it is a full duplex, i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.

8.2.2 TYPES OF SERIAL DATA COMMUNICATION

Serial data communication uses two types of communication

- Synchronous serial data communication
- Asynchronous serial data communication

Synchronous Serial Data Communication In synchronous serial data communication, transmitter and receiver are synchronised. It uses a common clock signal to synchronise the receiver and the transmitter, as

shown in Fig. 8.3. The figure shows the transmission of data; first the sync character and then, the data is transmitted. This format is generally used for high-speed transmission.

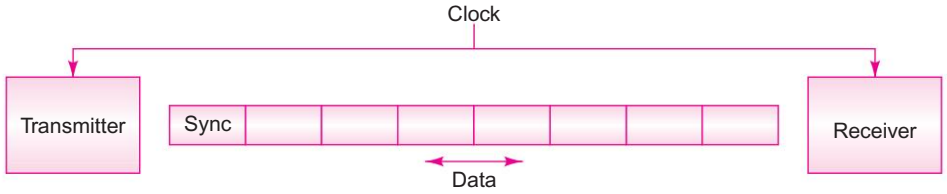


Figure 8.3 *Synchronous transmission format*

Asynchronous Serial Data Communication In asynchronous serial data communication, different clock sources are used for transmitter and receiver. In this mode, data is transmitted with start and stop bits. Transmission begins with start bit, followed by data and then stop bit. Figure 8.4 shows the transmission of 10 bit in the asynchronous format: one start bit, eight data bit and one stop bit. For error checking purpose, parity bit is included just prior to stop bit.

Figure 8.4 *Asynchronous transmission format*

8.2.3 BAUD RATE

The rate at which the bits are transmitted (bits/second) is called *baud or transfer rate*. The baud rate is the reciprocal of the time to send 1 bit. In asynchronous transmission, baud rate is not equal to number of bits per second. This is because, each byte is preceded by a start bit and followed by parity and stop bit. For example, in synchronous transmission, if data is transmitted with 9600 baud, it means that 9600 bits are transmitted in one second. For one bit, transmission time = 1 second/9600 = 0.104 ms.

SECTION REVIEW

1. The 8051 supports parallel and serial data transfer. True/False?
2. In serial data transfer, _____ bit is transmitted first.
3. In half duplex, data is transmitted in only one direction at a time. True/False?
4. If transmission and reception are done simultaneously, then the serial communication is called _____ .
5. Mention the advantages and disadvantages of parallel data communication.
6. List the types of serial data communication.
7. In _____ serial communication, first sync character and then the data is transmitted.
8. In asynchronous communication, _____ bit is included for error checking purpose.
9. The rate at which the bits are transmitted is called _____ .
10. In asynchronous transmission, the baud rate is not equal to number of bits/second. True/False?

8.3 8051 SERIAL COMMUNICATION

The 8051 supports a full duplex serial port. Full duplex means that it can transmit and receive a byte simultaneously. The 8051 has TXD (pin 11 or P3.1) and RXD (pin 10 or P3.0) pins for transmission and reception of serial data respectively. These pins are TTL compatible. The 8051 transfers and receives data serially with different baud rates. Three special function registers support serial communication, namely, SBUF Register, SCON Register and PCON Register.

SBUF Register SBUF is an 8 bit register. It has separate SBUF registers for data transmission and for data reception. These two registers are accessed by the same name, SBUF. One of the registers is write only and used to hold data to be transmitted via TXD pin. The other is read only and holds the received data from external source via RXD pin. For a byte of data to be transferred via the TXD line, it must be placed in the SBUF register. Similarly, SBUF holds the 8 bit data received by the RXD pin.

SCON Register The contents of SCON register are shown in Fig. 8.5. This register contains mode selection bits, serial port interrupt bit (TI and RI) and also the ninth data bit for transmission and reception (TB8 and RB8).

Bit 7 Bit 0

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 SM1	Serial communication mode selection bits SM0 SM1 Mode Description Baud rate 0 0 Mode 0 8 bit shift register mode fosc/12 0 1 Mode 1 8 bit UART Variable (set by timer 1) 1 0 Mode 2 9 bit UART fosc/32 or fosc/164 1 1 Mode 3 9 bit UART variable (set by timer 1)
SM2	In modes 2 and 3, if set, this will enable multiprocessor communication
REN	Enables serial reception
TB8	This is the 9th data bit that is transmitted in modes 2 and 3
RB8	9th data bit that is received in modes 2 and 3, it is not used in mode 0 In mode 1, if SM2 = 0, then RB8 is the stop bit that is received
TI	Transmit interrupt flag, set by hardware must be cleared by software
RI	Receive interrupt flag, set by hardware must be cleared by software

Figure 8.5 *Serial Control Register (SCON)*

PCON Register The smod bit (bit 7) of PCON register controls the baud rate in asynchronous mode transmission.

8.4 SERIAL COMMUNICATION MODES

The serial port can operate in four modes. SM0 and SM1 are D7 and D6 bits of SCON register, and these two bits determine the 4 serial modes. Serial port runs in both synchronous and asynchronous mode.

Mode 0 In mode 0, the serial port runs in synchronous mode. In this mode, data is transmitted and received by RXD pin, and TXD pin is used for clock output. In this mode, processor clock is used and the baud rate is 1/12 of the oscillator frequency. Eight bits are transmitted/received with least significant bit first. Figure 8.6 shows the timing for mode 0 data transmission. This mode is also called shift register mode.

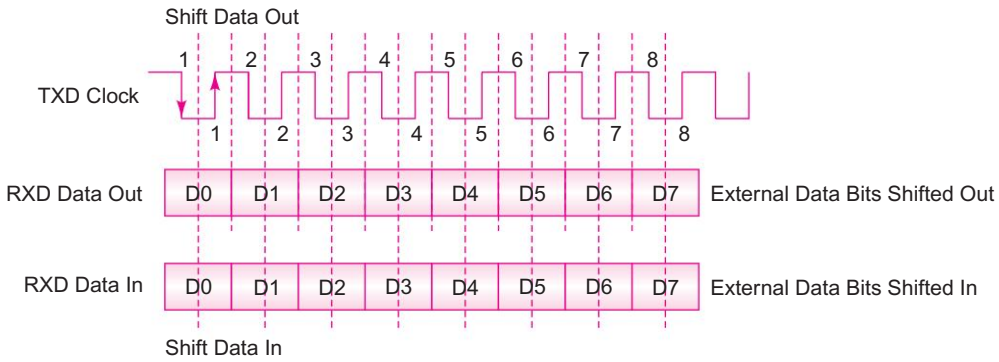


Figure 8.6 *Mode 0 Timing diagram*

In the remaining three modes, the data transmission is asynchronous and provides two different ways of clocking as well as 8 and 9 bit data transfer.

Mode 1 In this mode, SBUF becomes a 10 bit full duplex receiver/transmitter that may receive and transmit data at the same time. In this mode, ten bit are transmitted or received—1 start bit, 8 data bits and 1 stop bit. The interrupt flag TI is set, once all the ten bits have been sent. On reception, the stop bit goes into bit RB8 of the SCON register. The baud rate is variable and is determined by Timer 1 overflow rate. The value of SMOD of PCON register is as follows

$$\text{Baud rate} = [\,2^{\text{smod}}/32] \times [\text{Timer 1 overflow rate}]$$

Timer 1 interrupt should be disabled and is configured in auto-reload mode. In this case, if upper nibble of TMOD register is loaded with value 2H, then the baud rate is given by the formula.

$$\text{Baud rate} = [2^{\text{smod}} / 32] \times [\text{oscillator frequency}] / [12 \times [256 - [\text{TH1}]]]$$

For example, if TH1 contents are 230d, SMOD bit in PCON is 0 and if 12 MHz is the oscillator frequency, then the baud rate is 1201. To get exactly 1200 baud, the oscillator frequency must be 11.059 MHz. Table 8.1 lists the baud rates and how they can be obtained from Timer 1.

Mode 2 Mode 2 is similar to mode 1 except that eleven bits are transmitted or received—1 start bit, 8 data bits, a programmable ninth data bit, and 1 stop bit. During transmission, the ninth data bit is copied from bit TB8 of SCON and on reception, the ninth data bit goes into bit RB8 of the SCON register, while stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 of the oscillator frequency and depends on the SMOD bit of PCON register.

$$\text{Baud rate} = [2^{\text{smod}} / 64] \times [\text{oscillator frequency}]$$

TABLE 8.1 *Timer 1 generated commonly used baud rates*

Baud Rate	fosc	SMOD	Timer 1 C/$\overline{T}$	Timer 1 Mode	Timer 1 Reload Value
Mode 0, Max: 1000K	12 MHz	X	X	X	X
Mode 2, Max: 375K	12 MHz	1	X	X	X
Modes 1,3:62.5K	12 MHz	1	0	2	FFH
19.2K	11.059 MHz	1	0	2	FDH
9.6K	11.059 MHz	0	0	2	FDH
4.8K	11.059 MHz	0	0	2	FAH
2.4K	11.059 MHz	0	0	2	F4H
1.2K	11.059 MHz	0	0	2	E8H
137.5	11.986 MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

Mode 3 Mode 3 is identical to mode 2 except that the baud rate is determined as in mode 1. In mode 3, eleven bits are transmitted or received, a start bit, 8 data bits, a programmable ninth data bit and a stop bit.

SECTION REVIEW

1. Name the SFRs that support serial communication.
2. _____ pin is used for transmission and _____ pin is used for reception in 8051 serial communication.
3. The 8051 has two SBUF registers. True/False?
4. _____ register controls the baud rate in asynchronous mode of transmission.
5. 9 bit data is transmitted in mode 2 and mode 3 using TB8 bit of SCON register. True/False?
6. _____ bit of SCON enables serial reception.
7. _____ interrupt flag is used during serial transmission.
8. Serial interrupt has lowest priority in 8051 interrupts. True/False?
9. In mode 0 of serial communication, baud rate is Fosc/12. True/False?
10. If SMOD bit of PCON is zero, then baud rate in mode 2 of serial communication is oscillator frequency/64. True/False?

8.5 SERIAL COMMUNICATION PROGRAMMING

EXAMPLE 8.1

Write a subroutine to initialise 8051 serial port to operate with the following parameters:

- Disable interrupt for transmission and receiving
- Baud rate 9600
- One start bit, eight data bits and one stop bit; and to enable receiving and transmission

```
Step 1: Disable transmit and receive interrupt, clear bit 4 of IE register
Step 2: To set baud rate = 9600, the following parameters are selected
     a. Use 11.059 MHz crystal oscillator
     b. Choose mode 2 operation for timer 1
     c. Load 0010xxxx b to TMOD register
     d. Load reload value FD H to TH1 and TL1
     e. Clear SMOD bit of the PCON register
Baud rate = [2^smod / 32] x [oscillator frequency] / [12 x (256 – (TH1))]
Step 3: Value 01010000 b is written in SCON to operate serial port
in mode 1 and transmission and receiver is enabled.
CLR IE.4          ; Disable serial port interrupt
MOV TMOD,#20H     ; Choose mode 2 operation for timer 1
MOV TL1,#0FDH     ; Set the initial value
MOV TH1,#0FDH     ; Set the reset value
MOV PCON,#7FH     ; Clear MSB of PCON
SETB TR1          ; Start timer 1
MOV SCON,#50H     ; Operate serial port in mode 1
RET
```

EXAMPLE 8.2

Write a subroutine to initialise 8051 serial port to operate in mode 0 for transmission.

```
CLR IE.4             ; Disable serial port interrupt
MOV SCON, #00H       ; Operate serial port in mode 0
MOV SBUF, #44H       ; Load SBUF
CLR TI               ; Clear TI bit
LOOP: JNB TI, LOOP   ; Wait for the last bit to transfer
RET
```

EXAMPLE 8.3

Write a subroutine to initialise 8051 serial port to operate in mode 1 for transmission and timer 1 in auto-reload mode.

```
CLR IE.4          ; Disable serial port interrupt
MOV TMOD, #20H    ; Choose mode 2 operation for timer 1
MOV SCON, #40H    ; Operate serial port in mode 1
MOV TL1, #0FDH    ; Set the initial value
MOV TH1, #0FDH    ; Set the reset value
SETB TR1          ; Start timer 1
MOV SBUF, #56H    ; Load SBUF
CLR TI            ; Clear TI bit
LOOP: JNB TI LOOP ; Wait for last bit to transfer
RET
```

EXAMPLE 8.4

Write a program to initialise 8051 serial port to operate in mode 1 for receiving a serial byte through RXD pin, send the received data to port 2 and operate timer 1 in auto reload mode.

```
CLR IE.4             ; Disable serial port interrupt
MOV TMOD, #20H       ; Choose mode 2 operation for timer 1
MOV SCON, #50H       ; Operate serial port in mode 1
MOV TL1, #0FDH       ; Set the initial value
MOV TH1, #0FDH       ; Set the reset value
SETB TR1             ; Start timer 1
AGAIN: CLR RI        ; Clear RI bit
LOOP1: JNB RI, LOOP1 ; Wait for last bit to receive
MOV A, SBUF          ; Save incoming data in A
MOV P2, A            ; Send to port 2
SJMP AGAIN           ; Keep getting data
```

EXAMPLE 8.5

An 8051 micrcontroller has an oscillator frequency of 11.0592 MHz. Using timer 1 and configuring the UART in mode 1, write a C program that transmits ASCII character D at a baud rate of 9600.

```
#include <Intel\8051.h>
# main ( ) {                    //Start the program
          SCON = 0x42;          //Serial mode 1
          TMOD = 0x20;          // Timer 1 in mode 2
          TH1 = 0xFA;           // Baud rate = 9600
          TL1 = 0xFA;
          TR1 = 1;              //Start Timer1
          While (1) {
                 SBUF = 'D';    // Load 'D' into serial buffer
                 While ( !TI );// Wait for completion of transmission
                 TI = 0;        // Clear transmission flag
              }                 // While (1)
          }                     // End of the program
```

8.6 RS232

RS232 is the most widely used serial I/O interfacing standard. The RS232 standard was published by the Electronic Industry Association (EIA) in 1960. The COM1 and COM2 ports in IBMPC are RS232 compatible ports. In RS232, 1 is represented by –3 to –25 V and 0 is represented by +3 to +25 V. In a microcontroller, serial TXD and RXD lines are TTL compatible i.e. 1 and 0 are represented by +5 V and 0 V. For this reason, in order to connect a microcontroller to RS232 bus, voltage converters are used. MAX 232 IC is commonly used to convert the TTL logic levels to the RS232 voltage levels. The significance of the number 232 is that 2 is transmission line, 3 is receiving line, and 7 (2+3+2) is signal ground line. In RS232, ground line is common to the transmitter and receiver, and they are usable up to one meter without any shield.

8.6.1 RS232 PLUG CONNECTORS

Basic data communication link is shown in Fig. 8.7. The communication link consists of Data Terminal Equipment (DTE) and an associated modem (DCE) at each end. The function of modem is to process digital information received from the computer into a form suitable for analog transmission. Also, it receives analog signal and processes it into digital information.

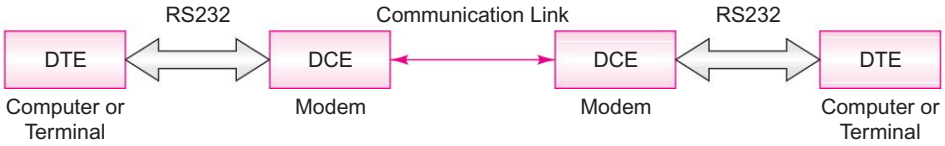


Figure 8.7 *Data communication system*

RS232 uses a 25 pin plug connector for all interface circuits and is commonly referred to as the DB-25 pin connector. DB-25P refers to the plug connector (male) and DB-25S refers to the socket connector (female). Since all the 25 pins are not used in PC, IBM introduced DB-9 connector. These plug connectors are shown in Fig. 8.8

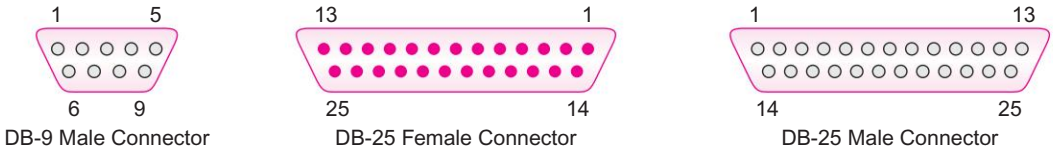


Figure 8.8 *DB-25 and DB-9 connectors*

A microcontroller with minimum three lines—TXD, RXD and ground—can be connected to another microcontroller as shown in Fig. 8.9. The remaining pins in the connector are used for handshaking signals. In this section, the functions of important signals in RS232 bus will be discussed.

Figure 8.9 *Interface of two microcontrollers with minimum signals*

TABLE 8.2 *RS232 pins (DB-25 and DB-9)*

DB-25	DB-9	I/O	Pin Description
1		x	Protective Ground
2	3	I	Transmitted Data
3	2	O	Received Data
4	7	I	Request To Send
5	8	O	Clear To Send
6	6	O	Data Set Ready
7	5	x	Signal Ground
8	1	O	Received Line Signal Detector
9		x	Reserved For Data Set Testing
10		x	Reserved For Data Set Testing
11		x	Unassigned
12		O	Secondary Rcvd Line Signal Detector
13		O	Secondary Clear To Send
14		I	Secondary Transmitted Data

(*Contd*)

(*Contd*)

DB-25	DB-9	I/O	Pin Description
15		O	Transmission Signal Element Timing
16		O	Secondary Received Data
17		O	Receiver Signal Element Timing
18		x	Unassigned
19		I	Secondary Request To Send
20	4	I	Data Terminal Ready
21		O	Signal Quality Detector
22	9	O	Ring Indicator
23		I/O	Data Signal Rate Selector
24		I	Transmit Signal Element Timing
25		x	Unassigned

Transmitted Data Data Terminal Equipment (DTE) transmits data through this pin.

Received Data Data Terminal Equipment (DTE) receives data through this pin.

Signal Ground This circuit establishes a common ground reference potential for all interface circuits.

Data Terminal Ready (DTR) When Data Terminal Equipment (DTE) is turned on, it sends active low signal DTR and indicates that DTE is ready for communication. If DTE is not ready for communication, then this signal is not activated.

Data Set Ready (DSR) When modem is turned on, it sends an active low signal DSR and indicates that it is ready for communication.

Request To Send (RTS) DTE asserts this signal to its associated DCE (modem) when it has data to transmit. RTS is an active low output from DTE and an input to the modem.

Clear To Send (CTS) In response to RTS, the modem sends out signal CTS to DTE and indicates that it is ready to receive the data.

Data Carrier Detect (DCD) DCD is an output from the modem (DCE) and an input to DTE. The modem asserts signal DCD to inform DTE that a valid carrier has been detected.

Ring Indicator (RI) RI is an output from the modem (DCE) and an input to DTE. This signal indicates that the telephone is ringing and is used when DTE is in charge of answering the phone.

8.6.2 MAX232

The 8051 microcontroller has two pins TXD and RXD, specifically used for transmitting and receiving data serially. In microprocessors and microcontrollers, the pins are TTL compatible. Therefore, it requires a line driver such as MAX232 IC to convert TTL levels to RS232 voltage levels, and vice versa.

MAX232 MAX232 is 16 pin IC and requires +5 V power supply. It converts RS232 voltage levels to TTL voltage levels, and vice versa. As shown in Fig. 8.10, it has two sets of line drivers for transferring and receiving the data.

T1 IN or T2 IN pins in the TTL side are connected to TXD pin of the microcontroller, while T1 OUT or T2 OUT pins in the RS232 side are connected to RXD pin of RS232 DB connector. Similarly, R1 IN or R2 IN pins in the RS232 side are connected to TXD pin of RS232 DB connector and R1 OUT or R2 OUT pins in the TTL side are connected to RXD pin of the microcontroller.

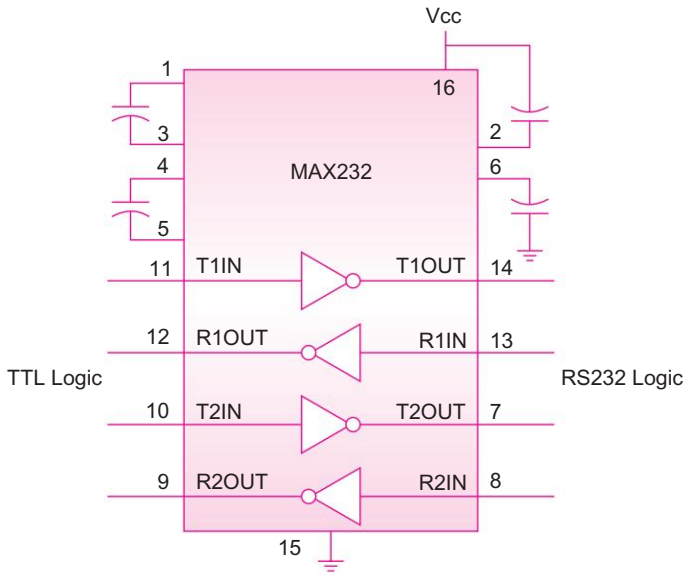


Figure 8.10 *MAX232 pin diagram*

MAX233 IC is a 20 pin IC and requires +5 V. MAX233 IC also converts RS232 voltage levels to TTL voltage levels and vice versa. As shown in Fig. 8.11, external capacitors are not required in MAX233, but it is expensive compared to MAX232.

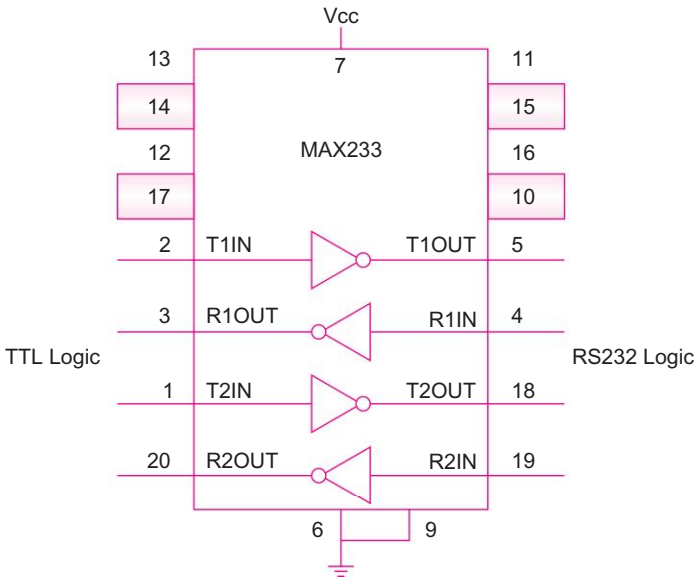


Figure 8.11 *MAX233 pin diagram*

SECTION REVIEW

1. Which voltage levels are used to represent 1 and 0 in an RS232 bus?
2. In an RS232 bus, pin no 2 is a transmission line. True/False?
3. What is the function of modem in data communication system?
4. _____ signal is used when data terminal equipment is in charge of answering the phone.
5. MAX232 converts RS232 voltage levels to TTL voltage levels and vice versa. True/False?

CHAPTER SUMMARY

In this chapter, we have discussed fundamentals and types of serial communication. The 8051 supports full duplex serial I/O, which is a need in many desired applications. UART with programming examples for serial communication and interfacing the 8051 with RS232 connectors have been presented in this chapter.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. _____ port is used for transmission of serial data.
   (a) P3.0 (b) P3.1 (c) P3.2 (d) P3.3
2. _____ type of communication is used in long distance.
   (a) Serial communication (b) Parallel communication
   (c) Both serial and parallel communication (d) None of the above
3. In _____ , communication link can be used for either transmission or reception.
   (a) Simplex (b) Half duplex (c) Full duplex (d) None of the above
4. Asynchronous transmission begins with _____ .
   (a) Start bit (b) Stop bit (c) Parity bit (d) Sync bit
5. Synchronous transmission transmits data with _____.
   (a) Start bit and stop bit (b) Start and parity bit (c) Sync and parity bit (d) First sync bit
6. Baud means _____ .
   (a) Bits transmitted in one second (b) Bits transmitted in one minute
   (c) Bytes transmitted in one second (d) Bytes transmitted in one minute
7. If data is transmitted with 9600 baud, then transmission rate is _____ .
   (a) 0.104 ms (b) 1.04 ms (c) 10.4 ms (d) None of the above

8. Baud rate is dependent on timer 1 _____ .
   (a) Mode 1 and mode 3 (b) Mode 2 and mode 3
   (c) Mode 0 and mode 2 (d) Mode 0 and mode 1
9. 9th data bit is transmitted in mode 2 and mode 3 using _____ .
   (a) TB8 bit of SCON (b) TI bit of SCON (c) SM0 bit of SCON (d) None of the above
10. _____ register holds the serial data interrupt flag.
   (a) SCON (b) PCON (c) IE (d) IP
11. If timer 1 operates in mode 2, with reload value FAH, SMOD=0 and Fosc = 11.059 MHz, then baud rate is _____ .
   (a) 4.8 K (b) 19.2 K (c) 1.2 K (d) 9.6 K
12. In mode 2, baud rate is calculated using _____ .
   (a) Baud rate = [2^{smod} / 64] × [oscillator frequency]
   (b) Baud rate = [2^{smod} / 32] × [oscillator frequency]
   (c) Baud rate = [2^{smod} / 64] × [Timer 1 overflow rate]
   (d) Baud rate = [2^{smod} / 32] × [Timer 1 overflow rate]
13. PS bit in _____ register must be set to give the serial data interrupt highest priority.
   (a) IP (b) IE (c) SCON (d) PCON
14. PCON register controls the baud rate in _____ .
   (a) Synchronous transmission
   (b) Asynchronous transmission
   (c) In synchronous and asynchronous transmission
   (d) None of the above
15. 9th data bit is transmitted in mode 2 and mode 3 using _____ .
   (a) SCON register (b) SBUF register (c) PCON register (d) Accumulator
16. In mode 0, baud rate is _____ .
   (a) 1/12th of the oscillator frequency (b) 1/6th of the oscillator frequency
   (c) Oscillator frequency (d) None of the above
17. RS232 bus is used _____ .
   (a) For serial transmission (b) For parallel transmission
   (c) For both serial and parallel transmission (d) None of the above
18. In RS232, logic 1 is represented by _____ .
   (a) –3 to –25 V (b) +3 to +25 V (c) +5 V (d) None of the above
19. The significance of number 3 in RS232 bus is _____ .
   (a) is transmission line (b) is receiving line (c) is ground (d) None of the above
20. DB-25 is.
   (a) 25-pin connector (b) 20-pin connector (c) 9-pin connector (d) 22-pin connector

REVIEW QUESTIONS

8.1 List the merits and demerits of parallel and serial communication.

8.2 Differentiate between
- (a) Simplex and half duplex
- (b) Half duplex and full duplex

8.3 Distinguish between synchronous and asynchronous communication.

8.4 Explain the role of Timer 1 in serial communication.

8.5 List the SFRs and their functions used in serial communication.

8.6 Write a program to receive serial data and store it in internal data RAM.

8.7 Explain the functions of the.
- (a) SBUF Register (b) SCON Register (c) PCON Register

8.8 Write a program to receive 8 bits data through port 1 and transfer this data serially.

8.9 Find the baud rate, when it operates in mode 1, oscillator frequency = 12 MHz and SMOD = 0 in the following.
- (a) [TH1] = 20H (b) [TH1] = 30H (c) [TH1] = 45H (d) [TH1] = 90H

8.10 Find the baud rate, when it operates in mode 1, oscillator frequency = 11.059 MHz and SMOD = 1 in the following.
- (a) [TH1] = 25H (b) [TH1] = 35H (c) [TH1] = 45H (d) [TH1] = 75H

8.11 Find the baud rate, when it operates in mode 2, oscillator frequency = 11.059 MHz and SMOD = 0.

8.12 Find the baud rate, when it operates in mode 2, oscillator frequency = 16 MHz and SMOD = 1.

8.13 Interface two 8051 microcontrollers with minimum signals.

8.14 Write a subroutine to initialise 8051 with the following specifications.
- (a) Enable interrupt for transmission
- (b) Baud rate is 4800
- (c) One start bit, eight data bits and one stop bit

8.15 Connect two 8051 microcontrollers using transmitter and receiver pins of serial port. Write a program to transmit data from one 8051 to another.

8.16 List and explain the important signals in RS232 bus.

8.17 Give the scheme to convert micrcontroller signals to RS232 compatible signals.

8.18 Explain the role of Max232/233 in serial communication.

8.19 Write a program to transfer the numbers 0 to F serially.

8.20 Serial data is being transmitted through RS232 with a baud rate of 10 KHz
- (a) Estimate the total time taken to transmit one byte of information in each protocol, including all framing/addressing/error correcting.
- (b) Estimate the time required by each, if 80 bytes are to be sent, optimising each transmission in any way possible.

9

8255A PROGRAMMABLE PERIPHERAL INTERFACE

Learning Objectives

After you have completed this chapter, you should be able to

- List the ports of the 8255A Programmable Peripheral Interface (PPI) and explain their features
- Explain various operating modes of the 8255A
- Design an interface circuit to interface the 8255A with the 8051 microcontroller
- Interface the 8255A with external devices such as printer, stepper motor and D/A converter
- Understand the semiconductor sensors and signal conditioning circuits
- Understand the design of minimum embedded systems

The 8051 microcontroller has only four 8 bit ports. If external memory is required, then port 0, port 2, port 3.6 and port 3.7 of the 8051 are used for external memory interfacing and also port 3 is used by external interrupts, serial ports and timers (as discussed in Section 2.4). If more I/O ports are needed, then 8255A Programmable Peripheral Interface (PPI) chip is interfaced with the 8051 microcontroller to expand the number of parallel ports.

9.1 FEATURES OF 8255A

- It is a 40 Pin DIP chip.
- It has three 8 bit ports—Port A, Port B and Port C.

- Port A can be programmed as either input or output port, with or without handshake signals, and also it can be programmed as bi-directional port.
- Port B can be programmed as input or output port, and with or without handshake signals.
- Port C is grouped in two 4 bit ports: PC4–PC7 (Port C upper) and PC3–PC0 (Port C lower)—each can be programmed as input or output port. Port C lines can be individually set or reset to generate control signals for controlling external I/O devices.

9.2 ARCHITECTURE OF 8255A

The internal organisation of the 8255A is shown in Fig. 9.1(a) and pin diagram of 8255A is shown in Fig. 9.1(b). It has two 8 bit ports—Port A and Port B, two 4 bit ports—Port C upper and Port C lower, data bus buffer, and read/write control logic. Port can be programmed to function as an input or an output port. When the ports are defined as output, they act as latches. When the ports are defined as input, they act as buffers. Group A controls port A and upper port C, and Group B controls port B and lower port C.

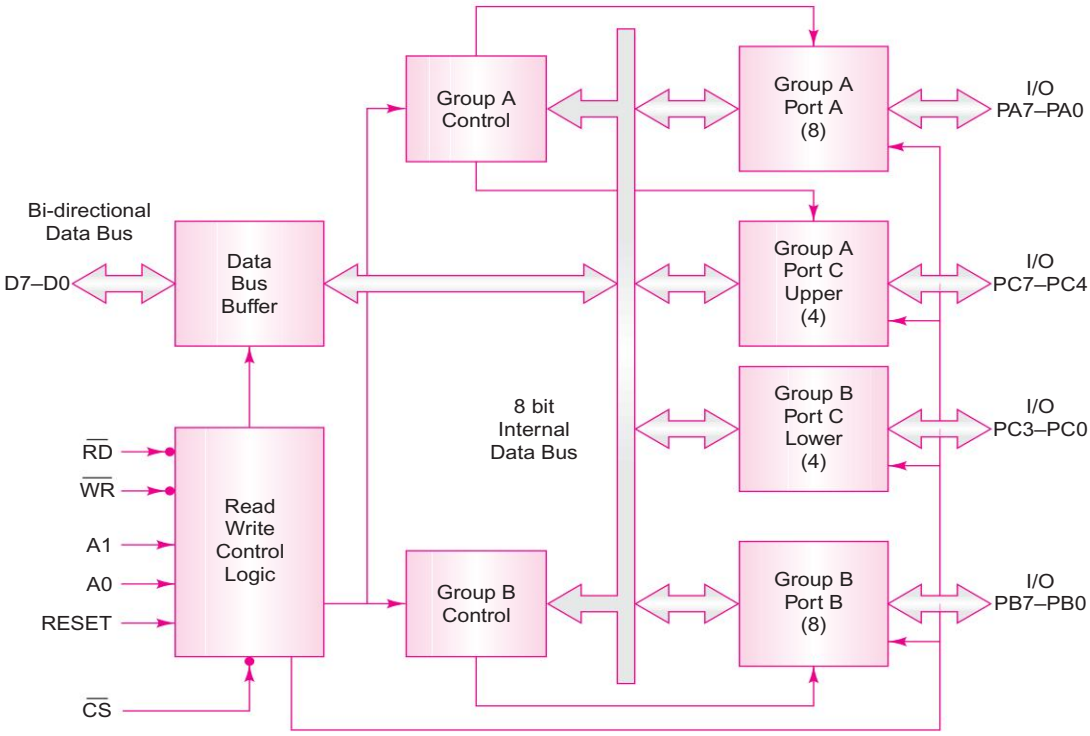


Figure 9.1(a) *Block diagram of Intel 8255A programmable peripheral interface*

Read/write control logic has six lines. Their functions are as follows:

- $\overline{CS}$ Signal is the master select—when this pin is low, it selects the entire chip.
- A1 and A0 select the specific ports and control register as shown in Table 9.1.

TABLE 9.1 *Selection of specific ports and control register*

$\overline{CS}$	A1A0	Port
0	00	A
0	01	B
0	10	C
0	11	Control Register

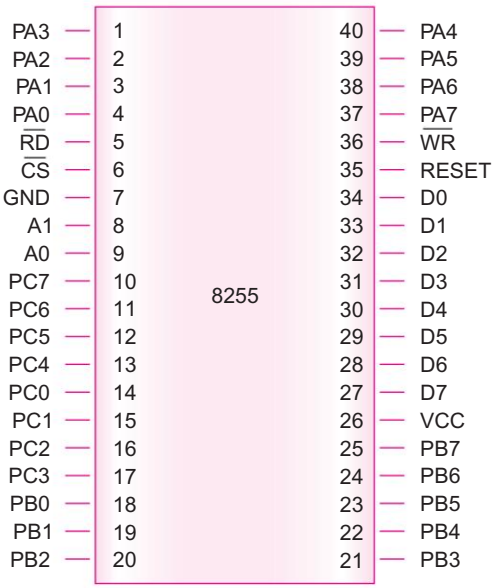


Figure 9.1(b) *Pin diagram of 8255A*

- $\overline{RD}$, an active low control signal, is input to the 8255A and enables the read operation. When the signal is low, the microcontroller reads the data from a selected I/O port.
- $\overline{WR}$, an active low control signal, is input to the 8255A and enables the write operation. When the signal is low, the microcontroller writes the data into a selected I/O port or control register.
- RESET is an active high signal; when this signal goes high, it clears the control register and all ports (port A, port B and port C) are defined as input.

Control Register

Control register is an 8 bit register, and its content is called *control word*. Control register controls the over all operations of the 8255A. Control register is divided into two blocks—Group A control and Group B control. Group A control, controls the port A and upper port C, and group B control, controls the port B and lower port C. Figure 9.2 shows the functions of control word, and control register must be programmed to select the operations of Port A, B and C. The 8255 operates in BSR mode or I/O mode as shown in Fig. 9.2.

If D7 = 0, port C operates in the bit set/reset (BSR) mode. In this mode, any of the eight bits of port C is selected by using D3, D2 and D1 bits and it is set or reset by D0 bit as shown in Fig. 9.3. These can be used to generate strobe signals for controlling external devices.

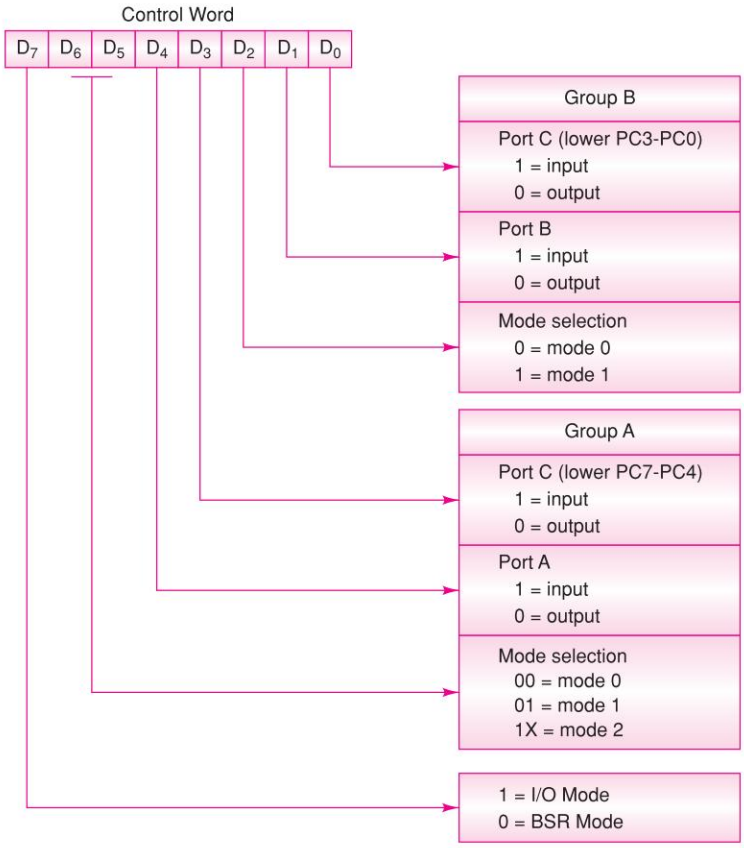


Figure 9.2 *Control word format of 8255A (Courtesy Intel)*

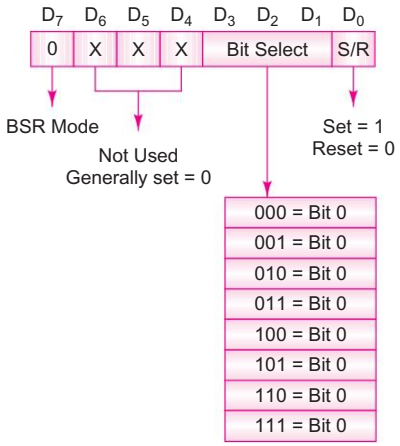


Figure 9.3 *Control word format in BSR mode (Courtesy Intel)*

Bit D7 = 1 selects I/O functions, then bits D6–D0 determine I/O function operation in three modes—mode 0, mode 1 and mode 2 as shown in Fig. 9.2.

Mode 0

In this mode, ports are used for simple input and output operations. No handshaking is required for I/O operations. Port A and B can be programmed as simple input/output 8 bit port, and port C can be programmed as simple input/output 4 or 8 bit port.

Mode 1

In mode 1, Port A and B can be used as input or output port in handshake mode, and port C is used to generate or accept these handshake signals. Port A uses three bits on port C, and port B uses three bits on port C to generate or accept handshake signals. The remaining two bits on port C are used as general purpose I/O.

Mode 1 Input When port A and B are configured as input port, as shown in Fig. 9.4, port A uses PC5, PC4 and PC3, and port B uses PC2, PC1 and PC0 for handshake signals. The remaining PC6 and PC7 are used as general purpose I/O. The functions of the handshake signals are as shown in Fig. 9.5.

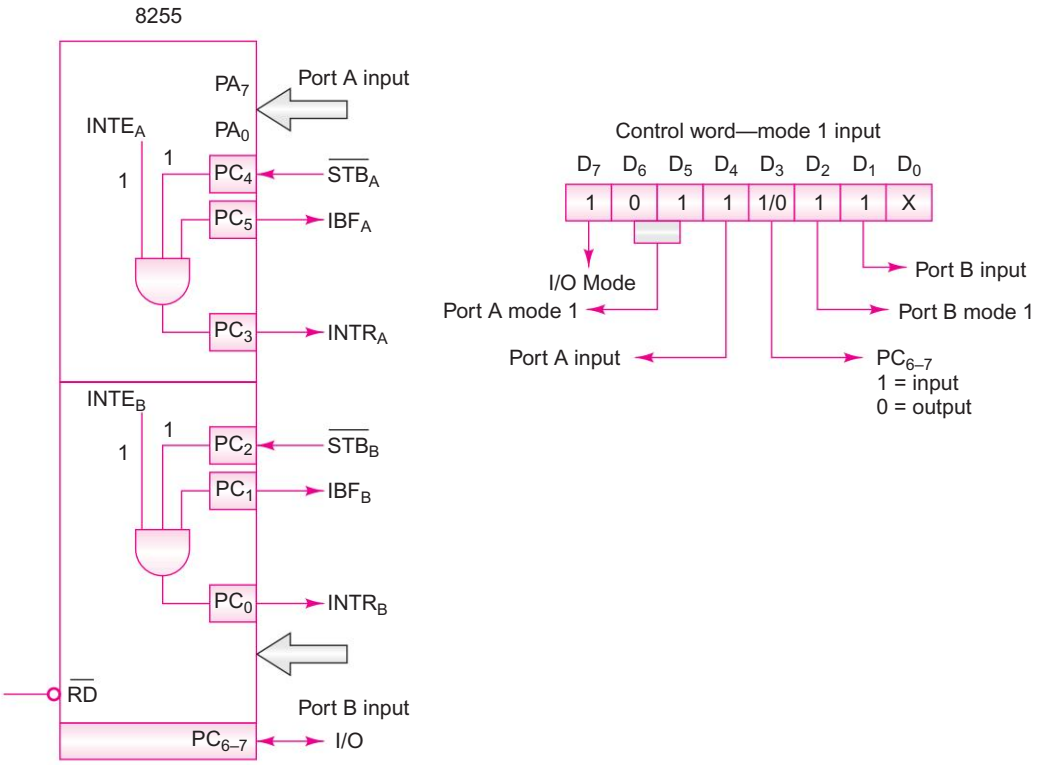


Figure 9.4 *8255A Model: Input configuration (Courtesy Intel)*

$\overline{\text{STB}}$ (Strobe Input) This is an active low signal. When the buffer is not full, an input device generates this signal. The input device places the data on input port and then pulses the $\overline{\text{STB}}$ signal. In response to $\overline{\text{STB}}$, the 8255 generates IBF and INTR as shown in Fig. 9.5.

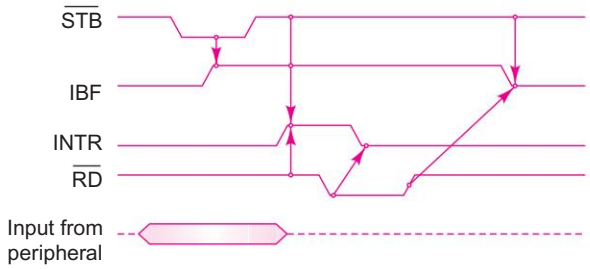


Figure 9.5 *Timing diagram of 8255 in mode 1 for strobed input (Courtesy Intel)*

IBF (Input Buffer Full) When the input receives the data, the 8255A sends the acknowledgement using an IBF line. The IBF line is reset, when processor reads the data.

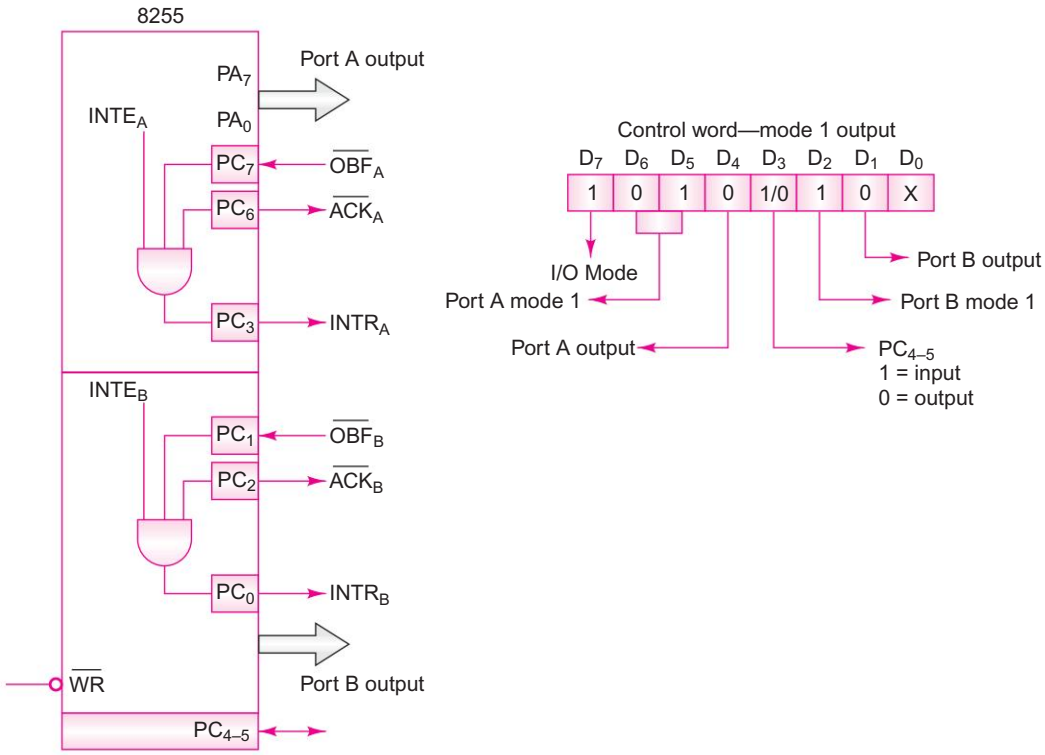


Figure 9.6 *8255A model: Output configuration (Courtesy Intel)*

INTR (Interrupt Request) This is an output signal; the 8255A asserts the interrupt request signal INTR and when the processor reads the data, then this is reset by the falling edge of $\overline{RD}$ signal.

INTE (Interrupt Enable) Port A uses an internal flip-flop $INTE_A$ and port B uses an internal flip $INTE_B$. These flip-flops are used to enable or disable the INTR signal. $INTE_A$ and $INTE_B$ are enabled or disabled using PC4 and PC2 respectively.

Mode 1 output When port A and B are configured as output port as shown in Fig. 9.6. port A uses PC3, PC6 and PC7, and port B uses PC0, PC1 and PC2 for handshake signals. The remaining PC4 and PC5 are used as general purpose I/O. The functions of the handshake signals are as shown in Fig. 9.7.

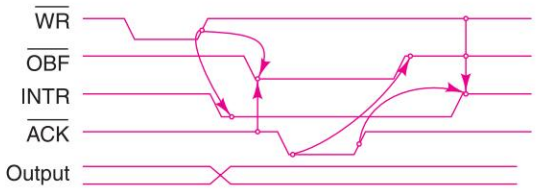


Figure 9.7 *Timing diagram of 8255A for strobed output (Courtesy Intel)*

INTR (Interrupt Request) When the output buffer is not full, then this signal is set. It is used to interrupt the processor to request the data for output and it is reset by the falling edge of $\overline{WR}$.

$\overline{OBF}$ (Output Buffer Full) The $\overline{OBF}$ goes low, when the processor writes data into the output port, and goes high again, after the 8255 receives the $\overline{ACK}$ signal from the output device.

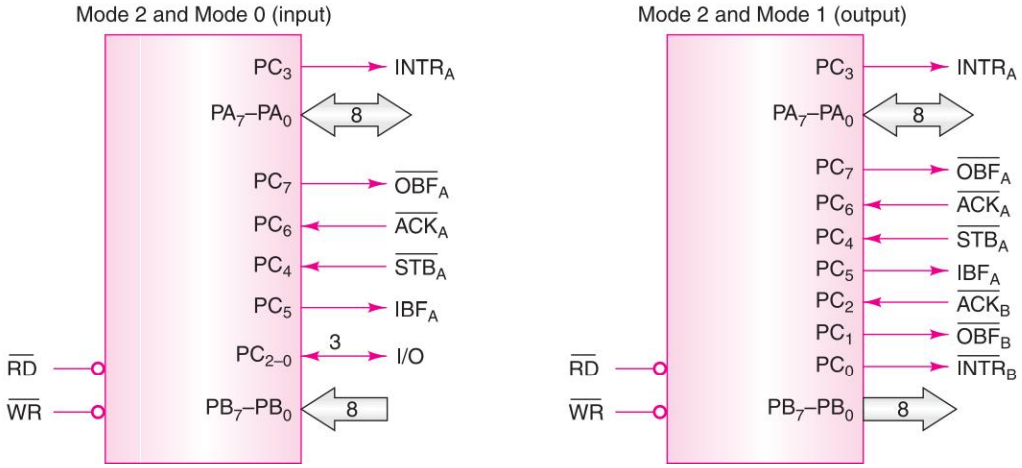
$\overline{ACK}$ When the output device receives the data from the 8255, it asserts $\overline{ACK}$ signal to the 8255, in order to acknowledge the receipt of the data.

INTE (Interrupt Enable) Port A uses an internal flip-flop $INTE_A$ and port B uses an internal FLIP-FLOP $INTE_B$. These flip-flops are used to enable or disable the INTR signal. $INTE_A$ and $INTE_B$ are enabled or disabled using PC6 and PC2 respectively.

Mode 2

In mode 2, Port A can be configured as an 8 bit bi-directional port, and 5 bit of port C are used as handshaking signals. In this mode, in both input and output configurations, port A acts as latch. Port B can be used either in mode 0 (simple I/O) or in mode 1 (I/O with handshake) as shown in Fig. 9.8.

When port B operates in mode 0, the remaining three signals of port C can be used as simple I/O, and when port B operates in mode 1, the remaining three signals of port C are used as handshake signals for port B. The functions of the handshake signals for Port A are as shown in Fig. 9.9.



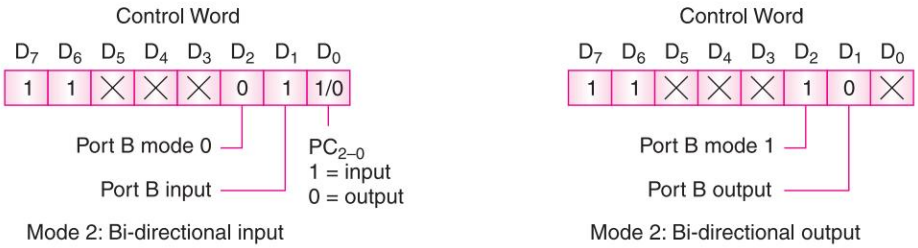


Figure 9.8 *8255A mode2 input/output configuration (Courtesy Intel)*

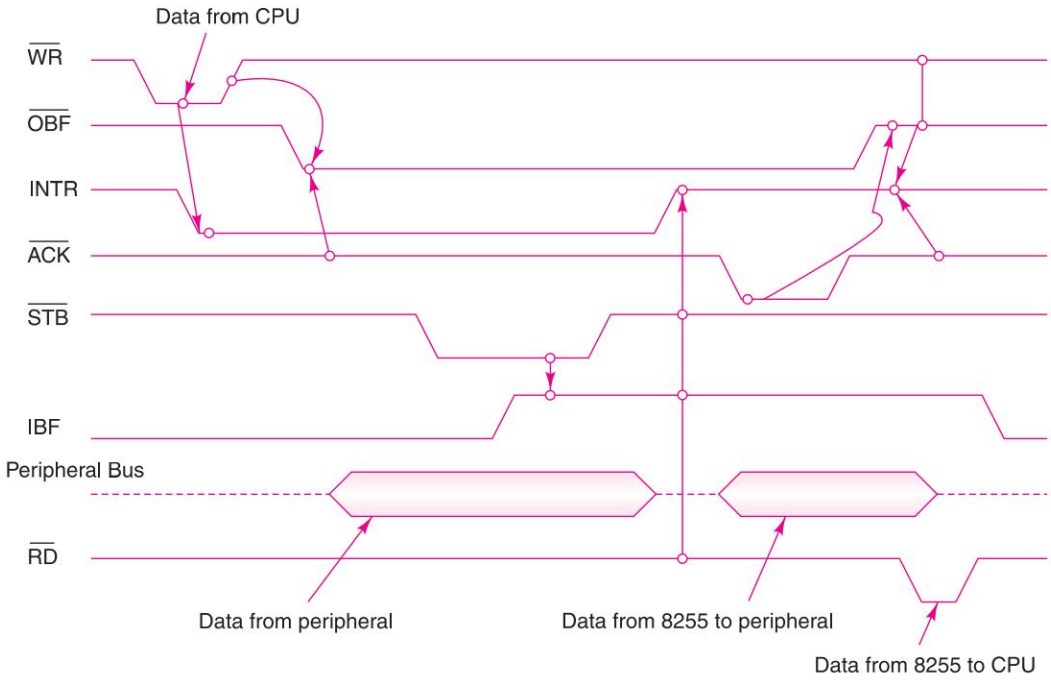


Figure 9.9 *Timing diagram for 8255A mode 2 (Courtesy Intel)*

$INTR_A$ (Interrupt Request) High on this pin can be used to interrupt the CPU for both input and output.

Data Output

$\overline{OBF}_A$ Low on this pin indicates that processor has written data to port A.

$\overline{ACK}_A$ Low on this pin indicates that the O/P device has received the data from 8255A port or peripheral has acknowledged the receipt of the data.

Data Input

$\overline{STB}_A$ (Strobe Input) Low on this pin, generated by the input device, indicates that it has transmitted 8 bit data.

IBF (Input Buffer Full) High on this pin indicates that data has been loaded into the input latch.

SECTION REVIEW

1. List the various ports of the 8255A.
2. _____ port of the 8255A operates as simple output port (without hand shaking).
3. _____ port of the 8255A is used to generate or accept handshake signals.
4. _____ port is used in BSR mode.
5. In mode 1, port A and B can be used as input or output port in handshake mode. True/False?
6. In mode 2, _____ port can be configured as an 8 bit bi-directional port.
7. In mode 2, _____ port can be used either in mode 0 and mode 1.
8. In mode 2, port A uses five bits of _____ as handshake signals.
9. Find the control word, if all the ports are defined as output port.
10. Find the control word, if all the ports are defined as input port.

9.3 I/O ADDRESSING

The 8255A can be interfaced with the processor by two methods, namely,

- Isolated I/O
- Memory mapped I/O

Few microprocessors have dedicated instructions for input and output operations. This method is called *isolated I/O* or *I/O mapped I/O*. In this approach, IN instruction reads the data from input device into the A of the microprocessor and OUT instruction writes the data from the A of the microprocessor to the output device.

In *Memory mapped I/O*, the microprocessor uses the same memory read and write instructions for I/O read and write operations. In this method, if the input device is interfaced for address 5000H, then the address 5000H is placed in the DPTR register. The move instruction MOVX A, @DPTR is then executed to get the data from the input device. If output device is interfaced for address 4000H, then the 4000H address is placed in the DPTR register. The move instruction MOVX @DPTR, A is then executed to output the data from A to output device. Intel processors 8085, 8086, etc. use both isolated I/O and memory mapped I/O method. The 8051 microcontroller has no separate I/O instructions, and uses only memory mapped I/O for interfacing peripherals.

9.4 INTERFACING 8255A WITH 8051

The 8051 uses memory mapped I/O for interfacing the 8255A—same as interfacing RAM memory. Since memory mapped space is used to access I/O device, MOVX instruction is used to access the 8255. If 16 address lines (A15 to A0) are used to interface the 8255, then base address is 16 bit and 16 bit address is stored in DPTR register. Instructions MOVX A, @DPTR and MOVX @DPTR, A are executed. If 8 address lines are used to interface the 8255, then base address is 8 bit and 8 bit address is stored in R0 or R1 register and instructions MOVX A, @R0 and MOVX @R0, A or MOVX A, @R1 and MOVX @R1, A are executed. Port 0 provides the lower 8 bit address and 8 bit data (D0–D7). Port 2 provides the upper 8 bit address (A8–A15). ALE pin of the 8051 and 74LS373 latch are used to demultiplex AD0–AD7. Data pins of D0–D7 of the 8255 are connected directly to Port 0 (data pins) of the 8051 microcontroller. Two address lines, A0 and A1 (latch O/P) are connected directly to the address lines of A0 and A1 of 8255A. The remaining address lines are used to select 8255A (connected to $\overline{CS}$ pin of 8255A).

EXAMPLE 9.1

Interface the 8255A with the 8051 microcontroller such that the control register is selected for address 1003H. Find the address of port A, port B and port C.

SOLUTION

The control register is selected for address 1003H. Table 9.2 gives the condition of the address lines A15–A0 for the ports A, B and C, and the control register.

TABLE 9.2 *Address table*

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Port
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	Port A
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	Port B
0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	Port C
0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	Control register

The address of port A is 1000H, port B is 1001H, port C is 1002H and control register is 1003H. $\overline{RD}$ (P3.7) and $\overline{WR}$(P3.6) are connected to $\overline{RD}$ and $\overline{WR}$ pins of the 8255A as shown in Fig. 9.10. Address lines A1 and A0 are connected directly to A1 and A0 pins of the 8255. The remaining address lines—A15, A14, A13, A12, A11…A2 are connected to decoder 74LS138. Y0 output line of the decoder is connected to $\overline{CS}$ pin of the 8255. Data pins of the 8255 are connected directly to data bus of the 8051.

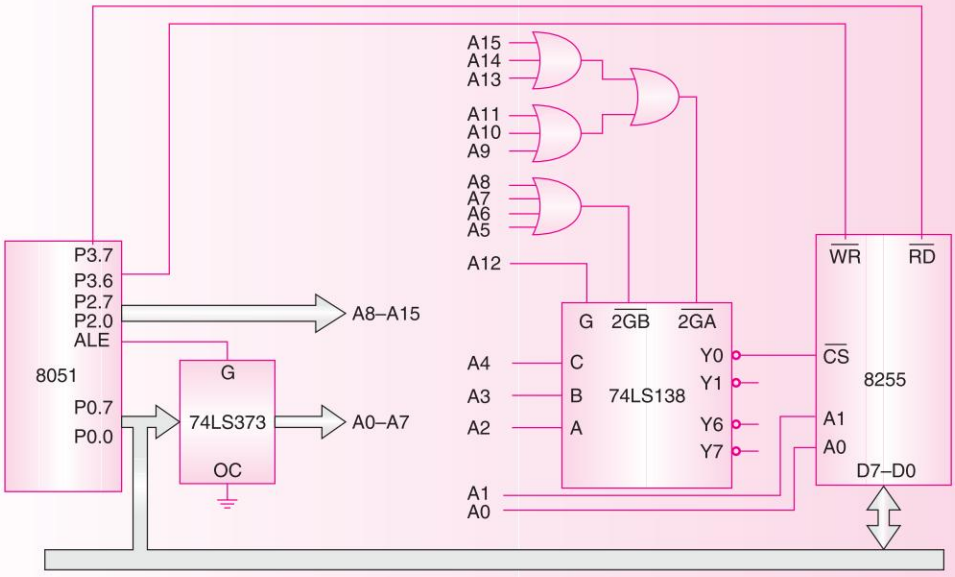


Figure 9.10 *Interfacing 8255A*

EXAMPLE 9.2

Interface the 8255 with the 8051 microcontroller such that port A is selected for address C000H, port B is selected for address C002H, Port C for address C004H and control register for address C006H.

SOLUTION

Table 9.3 gives the condition of address lines A15 to A0 for port A, port B, port C and control register. Conditions of address lines A2 and A1 are 00 for port A, 01 for port B, 10 for port C, and 11 for control register.

TABLE 9.3 *Address table*

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Port
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Port A
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Port B
1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	Port C
1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	Control register

The interfacing connection is as shown in Fig. 9.11. Address lines A2 and A1 of the 8051 microcontroller are connected to A1 and A0 lines of the 8255. The remaining address lines—A15, A14, A13, A12… A3, A0 are connected to 74LS138 decoder. Y0 output line of decoder is connected to $\overline{CS}$ pin of the 8255 as shown in Fig. 9.11.

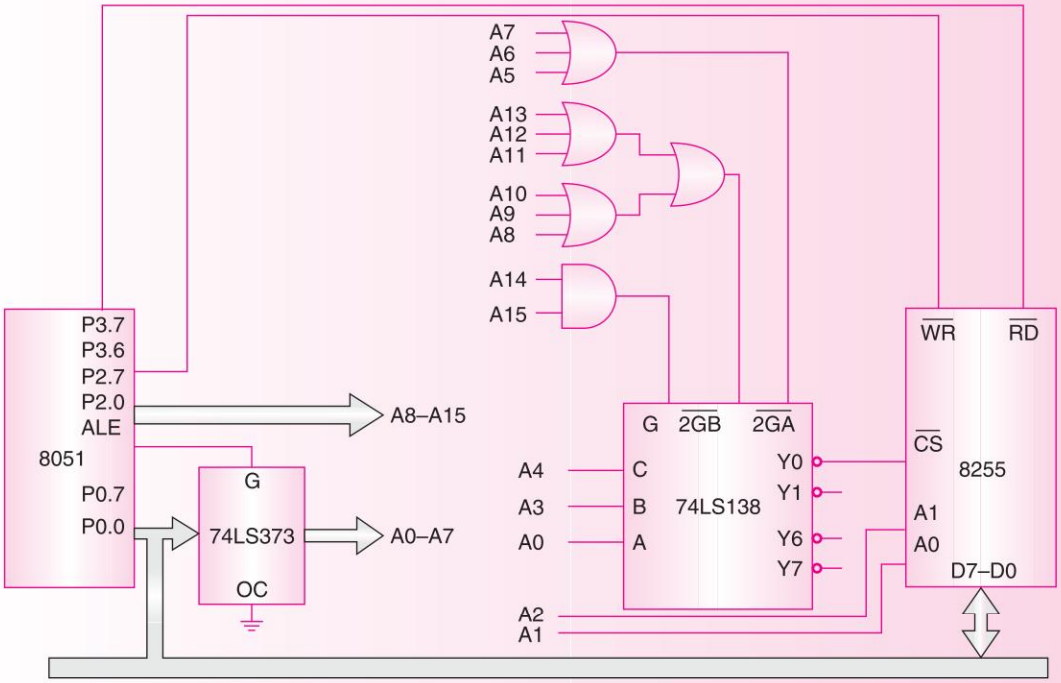


Figure 9.11 *Interfacing 8255A*

EXAMPLE 9.3

Interface two 8255A with the 8051 microcontroller such that port A of 8255(1) is selected for address 2000H and port A of 8255(2) is selected for address 4000H.

SOLUTION

Table 9.4 gives the condition of address lines A15 to A0 for 8255(1) and 8255(2). The condition of A1 and A0 is same for port A, port B, port C and control register in both 8255.

TABLE 9.4 *Address table*

	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Port
8255(1)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Port A
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	Port B
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	Port C
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	Control register
8255(2)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Port A
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Port B
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Port C
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	Control register

As shown in Fig. 9.12, A1 and A0 address lines are connected directly to A1 and A0 pins of 8255(1) and 8255(2). The remaining address lines—A15, A14, A13, A12…A2 are connected to 74LS138 decoder. Y1 output line of the decoder is connected to $\overline{CS}$ of 8255(1) and Y2 output line of the decoder is connected to $\overline{CS}$ of 8255(2).

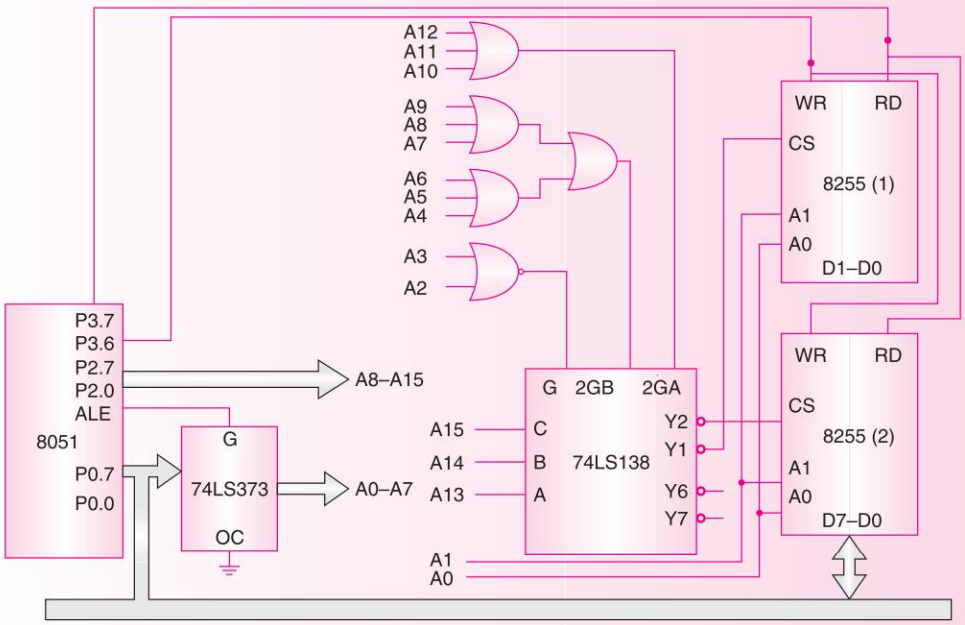


Figure 9.12 *Interfacing 8255*

EXAMPLE 9.4

Interface the 8255A with the 8051 microcontroller such that the control register is selected for address 03H. Find the address of port A, port B and port C.

SOLUTION

The control register is selected for address 03H. Table 9.5 gives the condition of the address lines A7–A0 for the ports A, B and C, and the control register.

TABLE 9.5 *Address table*

A7	A6	A5	A4	A3	A2	A1	A0	Port
0	0	0	0	0	0	0	0	Port A
0	0	0	0	0	0	0	1	Port B
0	0	0	0	0	0	1	0	Port C
0	0	0	0	0	0	1	1	Control register

The address of port A is 00H, port B is 01H, port C is 02H and control register is 03H. $\overline{RD}$ (P3.7) and $\overline{WR}$ (P3.6) are connected to $\overline{RD}$ and $\overline{WR}$ pins of the 8255 as shown in Fig. 9.13. Address lines A1 and A0 are connected directly to A1 and A0 pins of the 8255A. The remaining address lines—A7…A2 are connected through OR gate to $\overline{CS}$ pin of the 8255A. Data pins of 8255A are connected directly to data bus of 8051 as shown in Fig. 9.13.

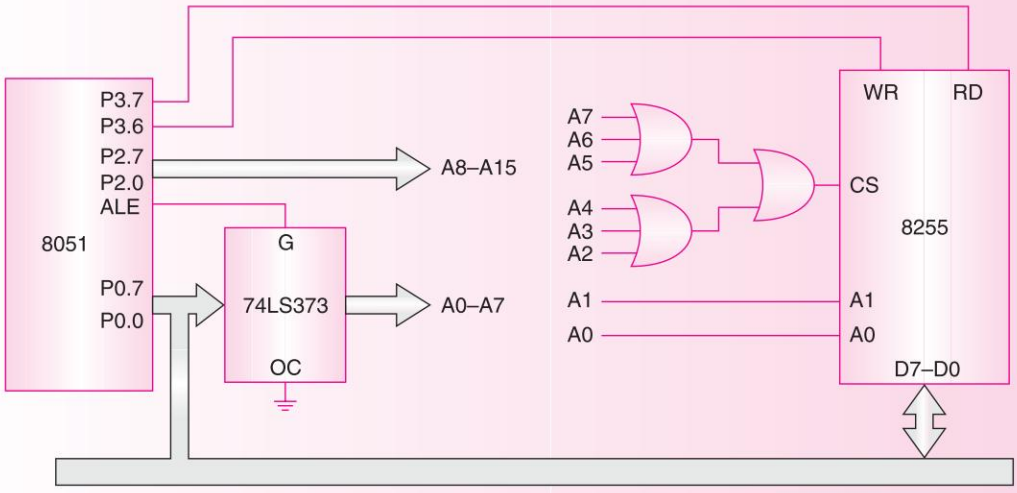


Figure 9.13 *Interfacing 8255*

EXAMPLE 9.5

Interface the 8255 with the 8051 microcontroller such that port A is selected for address CoH, port B is selected for address C2H, Port C for address C4H and control register for address C6H.

SOLUTION

Table 9.6 gives the condition of address lines A7 to A0 for port A, port B, port C and control register. Conditions of address lines A2 and A1 are 00 for port A, 01 for port B, 10 for port C, and 11 for control register.

TABLE 9.6 *Address table*

A7	A6	A5	A4	A3	A2	A1	A0	Port
1	1	0	0	0	0	0	0	Port A
1	1	0	0	0	0	1	0	Port B
1	1	0	0	0	1	0	0	Port C
1	1	0	0	0	1	1	0	Control register

The interfacing connection is as shown in Fig. 9.14. Address lines A2 and A1 of the 8051 microcontroller are connected to A1 and A0 lines of the 8255. The remaining address—lines A7…A3, A0 are connected using NAND and OR gate to $\overline{CS}$ pin of 8255 as shown in Fig. 9.14.

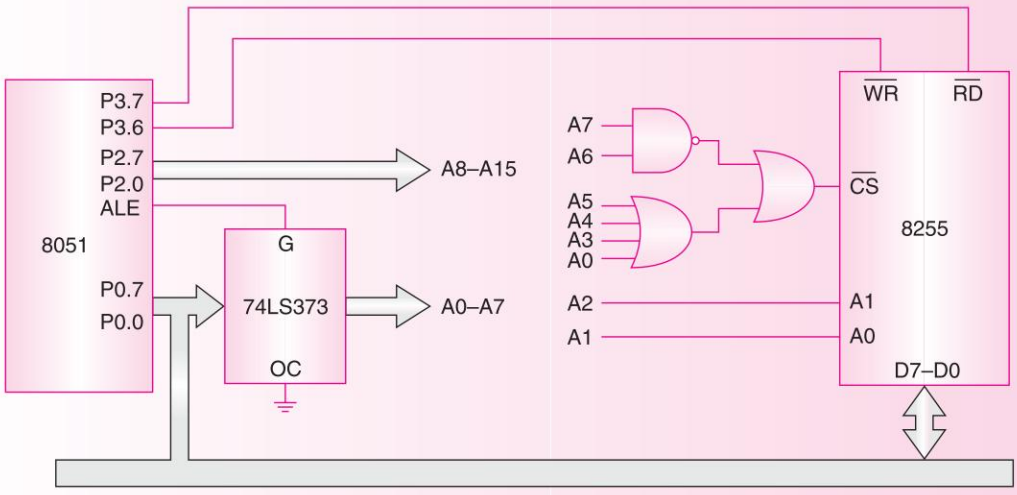


Figure 9.14 *Interfacing 8255*

SECTION REVIEW

1. The 8051 uses memory mapped I/O for interfacing the 8255A. True/False?
2. If 16 address lines are used to interface the 8255A, then _____ register of the 8051 is used to store the address.
3. Name the registers of the 8051, if 8 address lines are used to interface the 8255A.
4. Name the instructions of the 8051, if 8 address lines are used to interface the 8255A.
5. When the 8255A is reset, all the ports A, B and C are defined as _____.
6. The control register of the 8255A is selected, if the condition of pin A1 and A0 is _____.
7. Port B of the 8255A is selected, if the condition of pin A1 and A0 is _____.
8. If A2–A7 address lines are connected through NAND gate to $\overline{CS}$ pin of the 8255A, then control register address is _____ .

9.5 I/O DEVICES INTERFACING WITH 8051 USING 8255A

I/O devices are interfaced using port A or port B or port C. If the peripheral is an input device, then the port is defined as *input port*. If the peripheral is an output device, then the port is defined as *output port*. If I/O devices are 8 bit, then port A or B or C is used. If I/O devices are 4 bit, then upper port C or lower port C is used. We will discuss 4 bit and 8 bit I/O devices interfacing in this section.

9.5.1 INTERFACING PUSH BUTTON SWITCHES AND LED USING 8255A

EXAMPLE 9.6

Interface four push button switches to PC0–PC3 and display the key value by four LEDs connected to PC4–PC7.

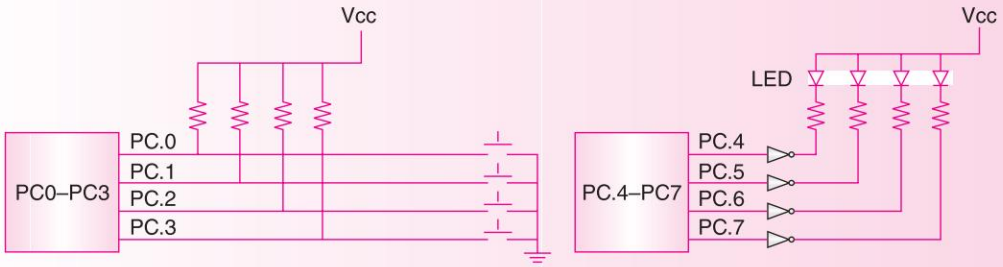


Figure 9.15 *Push button key and LED interface*

SOLUTION

In this problem, 4 push button switches are connected to lower port C—lower port C must be defined as input port, and 4 LEDs are connected

to upper port C—upper port C must be defined as output port. The control word is

100X00X1 ------- 81H

The programming can be divided into the following categories:

1. Define lower port C as input port and upper port C as output port.
2. Check if a key is pressed.
3. Debounce the key.
4. Identify the key in binary format.
5. Display the key condition using four LEDs.

Assume that the 8255A is interfaced to the 8051 as shown in Fig. 9.13. Then Port A address is 00H, Port B address is 01H, Port C address is 02H and control register address is 03H.

```
        MOV A,#81H          ; Move control word to A
        MOV R1,03H          ; Load control register address to R1
        MOVX @R1,A          ; Move contents of A to control register
        MOV R1,02H          ; Load port C address to R1
START:  MOVX A,@R1          ; Move contents of lower port C to A
        ANL A,#0FH          ; Mask the upper nibble
        CJNE A,#0FH,CHECK ; Key pressed branch to CHECK
        SJMP START          ; Branch to START, still key is pressed
CHECK:  ACALL DELAY         ; Call delay
        SWAP A              ; Exchange upper and lower nibble of A
        CPL A               ; Complement A
        MOVX @R1,A          ; Move contents of A to upper port C
        SJMP START          ; Branch to START
DELAY:  MOV R6, #20H        ; Delay program
NEXT 2: MOV R7, # OFFH
NEXT 1: DJNZ R7, NEXT 1
        DJNZ R6, NEXT 2
        RET
        END
```

9.5.2 INTERFACING STEPPER MOTOR USING 8255A

EXAMPLE 9.7

Interface stepper motor using port A and write a program to rotate stepper motor in clockwise direction.

SOLUTION

The circuit to interface stepper motor is as shown in Fig. 9.16. The functions of driver circuit are discussed in section 6.9. The

stepper motor is connected to PA0–PA3 as shown in Fig. 9.16. Port A must be defined as output port. The control word is

1000X0XX ------- 80H

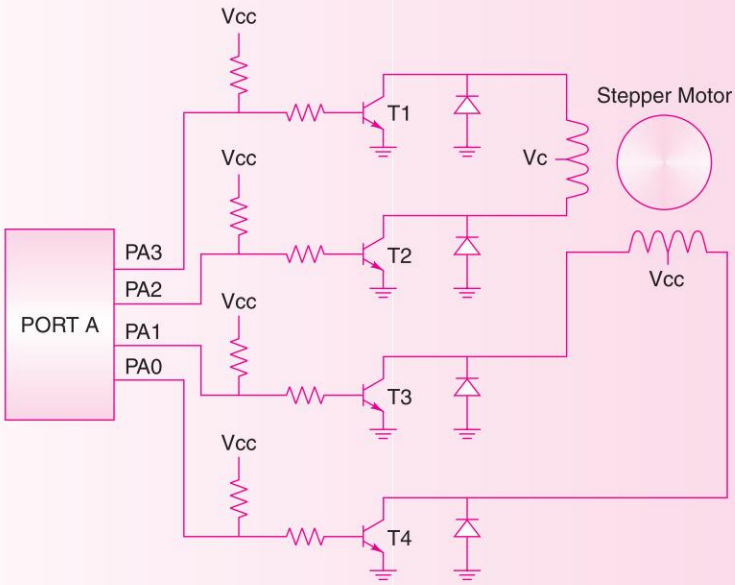


Figure 9.16 *Driver circuit and port interface for stepper*

Assume that the 8255 is interfaced to the 8051 as shown in Fig. 9.11. Then Port A address is C000H, Port B address is C002H, Port C address is C004H and control register address is C006H.

```
      MOV A,#80H          ; Move control word to A
      MOV DPTR,#0C006H    ; Load control register address to DPTR
      MOVX @DPTR,A        ; Move contents of A to control register
      MOV DPTR,#0C000H    ; Load port A address to DPTR
LOOP1:MOV A,#03H          ; Move step sequence 03 to A
      MOVX @DPTR,A        ; Move contents of A to port A
      ACALL DELAY         ; Call delay program
      MOV A,#09H          ; Move step sequence 09 to A
      MOVX @DPTR,A        ; Move contents of A to port A
      ACALL DELAY         ; Call delay program
      MOV A,#0CH          ; Move step sequence 0C to A
      MOVX @DPTR,A        ; Move contents of A to port A
      ACALL DELAY         ; Call delay program
      MOV A,#06H          ; Move step sequence 06 to A
      MOVX @DPTR,A        ; Move contents of A to port A
      ACALL DELAY         ; Call delay program
      AJMP LOOP1
```

```
DELAY:MOV R5,#0FFH        ; Delay program
S2:   MOV R7,#0FFH
S1:   DJNZ R7,S1
      DJNZ R5,S2
      RET
```

9.5.3 INTERFACING DAC 08 USING 8255A

EXAMPLE 9.8

Interface DAC o8 using port B and write a program to generate square wave and ramp.

SOLUTION

Assume that the 8255A is interfaced to the 8051 as shown in Fig. 9.14. Then, Port A address is C0H, Port B address is C2H, Port C address is C4H and control register address is C6H. The circuit to interface DAC 08 is as shown in Fig. 9.17. Functions of DAC 08 have been discussed in Section 6.7. DAC 08 is connected to port B as shown in Fig. 9.17. Port B must be defined as output port. The control word is

100XX00X ------- 80H

Program to generate square wave

```
        MOV A,#80H       ; Move control word to A
        MOV R1,C6H       ; Load control register address to R1
        MOVX @R1,A       ; Move contents of A to control register
        MOV R1,C2H       ; Load port B address to R1
        CLR A            ; Clear A
START:  MOVX @R1,A       ; Move contents of A to port B
        LCALL DELAY      ; Delay decides the period of square wave
        CPL A            ; Complement contents of A
        SJMP START       ; Branch to start
DELAY:  MOV R1,#30H      ; Delay program
S2:     MOV R2,#0FFH
S1:     DJNZ R2,S1
        DJNZ R1,S2
        RET
        END
```

Program to generate ramp

```
        MOV A,#80H        ; Move control word to A
        MOV R1,C6H        ; Load control register address to R1
```

```
        MOVX @R1,A        ; Move contents of A to control register
        MOV R1,C2H        ; Load port B address to R1
        CLR A             ; Clear A
START:  MOVX @R1,A        ; Move contents of A to port B
        INC A             ; Increment A
        LCALL DELAY       ; DAC conversion time
        SJMP START        ; Branch to START
DELAY:  MOV R1,#0FH       ; Delay program
NEXT:   DJNZ R1,NEXT
        RET
        END
```

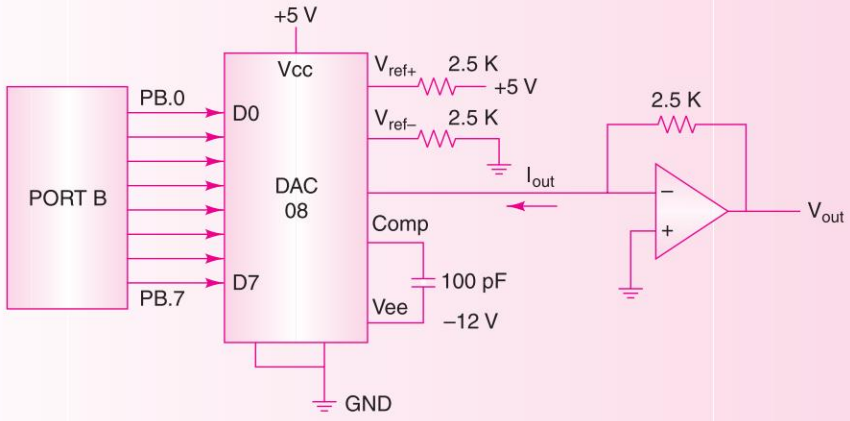


Figure 9.17 *8051 connection to DAC 08*

9.5.4 INTERFACING PRINTER USING 8255A

EXAMPLE 9.9

Interface a printer using port B and write a program to print ABCD.

SOLUTION

In this interfacing, printer interface allows the transfer of data under the control of two-handshake signals—strobe and busy. The printer is connected to the 8051 microcontroller using the 8255A as shown in Fig. 9.18. Data lines are connected to port B, the busy line is connected to PC1 and the strobe line is connected to PC4.

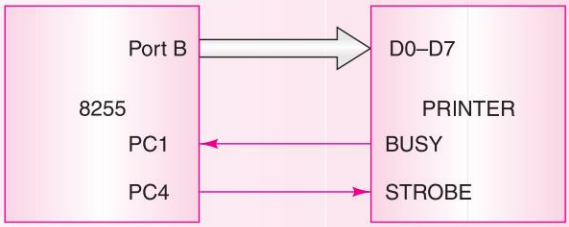


Figure 9.18 *Interfacing printer using 8255*

Printer timing is as shown in Fig. 9.19. First, the strobe line is kept high and then, the busy line is checked for low—if busy line is low, then data is placed on the data line and the strobe line is made low. This is repeated for the entire data. The flowchart is as shown in Fig. 9.20. Port B is defined as output port, lower port C is defined as input and the upper port C is defined as output port. The control word is

100X0001-----81H

Figure 9.19 *Timing diagram printer signals*

Assume that the printer operates in normal mode. The ASCII code for normal mode, carriage return and line feed are 00H, 0DH and 0AH respectively. Assume that FFH indicates end of the data, and FFH is stored in internal data memory location 40H. Assume that printer data is stored in internal RAM, from location 30H as shown in Table 9.7. Assume that 8255A is interfaced to the 8051 as shown in Fig. 9.11. Then port A address is C000H, port B address is C002H, port C address is C004H and control register address is C006H.

```
        ORG 0000H
        TEMP EQU 40H          ; Init. TEMP to 40H
        MOV R1,#30H           ; Init. R1 to 30H
        MOV DPTR,#0C006H      ; Load control register address to DPTR
        MOV A,#81H            ; Port B and PC4 as output, PC1 as input
```

```
        MOVX @DPTR,A
        MOV DPTR,#0C004H    ; Load port C address to DPTR
BACK:   MOV A,#10H          ; To make (PC4) = 1
        MOVX @DPTR,A        ; Strobe = 1
        CALL DELAY          ; Call delay program
LOOP1:  MOVX A,@DPTR        ; Check busy line (PC1)
        ANL A,#02H          ; Mask other bits
        JNZ LOOP1           ; Busy = 1, branch to loop1
        MOV A,@R1           ; Copy contents of internal memory to A
        CJNE A,TEMP,LOOP2   ; (A) ≠ (TEMP) branch to loop2
        SJMP LOOP3          ; (A) = (TEMP) branch to loop3
LOOP2:  MOV DPTR,#0C002H    ; Load port B address to DPTR
        MOVX @DPTR,A        ; Send data to printer
        MOV A,#00H          ; To make (PC4) = 0
        MOV DPTR,#0C004H    ; Load port C address to DPTR
        MOV @DPTR,A         ; Strobe = 0
        INC R1              ; INC R1
        JMP BACK            ; Branch to BACK
DELAY:  MOV R0,#0FH         ; Delay program
NEXT:   DJNZ R0,NEXT
        RET
LOOP3:  NOP
        END
```

TABLE 9.7 *Data stored in internal memory to print ABCD*

Address of internal memory	Data stored
40	FF
30	00
31	41
32	42
33	43
34	44
35	0D
36	0A
37	FF

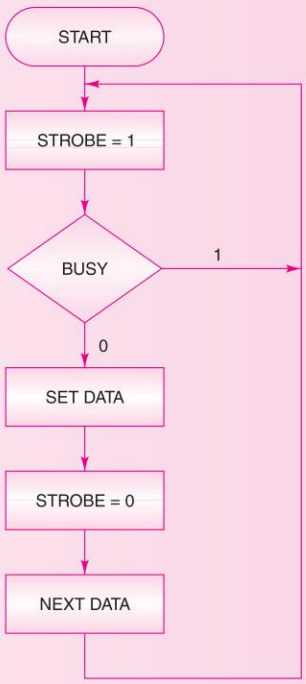


Figure 9.20 *Flow chart of printer*

9.6 SEMICONDUCTOR SENSORS AND SIGNAL CONDITIONING CIRCUITS

Semiconductor sensors are transducers that convert non-electrical signals into electrical signals. For example, temperature sensor is used to convert temperature into voltage. Semiconductor sensors are widely used for measurement and control of physical variables. Important specifications to keep in mind when selecting a sensor are

- Accuracy
- Size
- Packaging
- Long-term stability
- Repeatability

Temperature, pressure, level, flow, and humidity are the most important measurements in the industry. Temperature sensors are used in automotive systems. Pressure sensors are used in pneumatic and tactile detection systems. Humidity sensors are used to measure the water vapour content in air or other gases.

Semiconductor sensors require additional signal conditioning circuits to amplify and shift the sensor output signal to match the range of A/D converter. In this section, we will discuss signal conditioning circuits and an algorithm to interface temperature, pressure, and humidity sensors with the 8051 microcontroller using 8255A programmable peripheral chip. The overall process is as shown in Fig. 9.21. The operating parameters of temperature, pressure and humidity sensors are listed in Table 9.8.

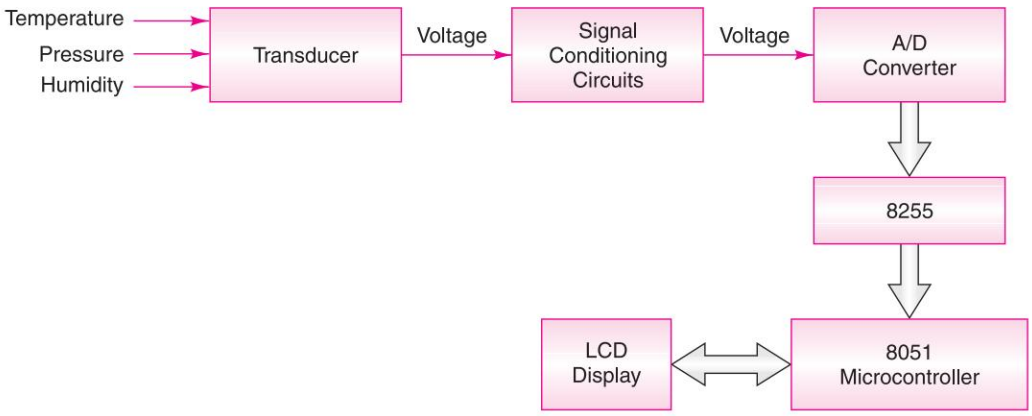


Figure 9.21 *Block diagram*

TABLE 9.8 *Operating parameters of temperature, pressure and humidity sensors*

Parameters	Pressure Sensor BPT	Temperature Sensor LM34	Humidity Sensor IH-3605
Operating range	800–1100 mbar	–50 to 300° F	0–100 % RH
Supply voltage	7–24 V dc	5 V to 30 V	+5 V
Voltage output	4.5 V to 5.5 V	Linear to +10 mV/°F	0.8 to 3.9 V

Signal Conditioning Circuits

Table 9.8 shows the transducer output voltages for pressure, temperature and humidity sensors. Signal conditioning circuits can be used to match the full range of the A/D converter. The signal conditioning circuit (shown in Fig. 9.22) is to amplify the sensor's span to full range of A/D converters. The signal conditioning circuit consists of

- Voltage follower
- Voltage shifter

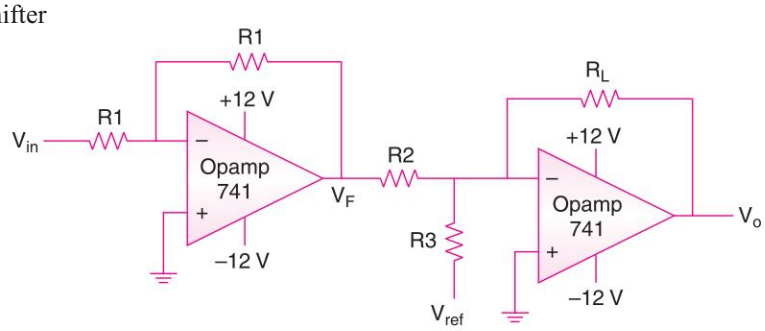


Figure 9.22 *Signal conditioning circuit*

Voltage Follower

Voltage follower circuits are useful as current amplifier or impedance converter for isolating the signals from high impedance sources. The output voltage of the voltage follower is given by

$$V_F = -V_{IN} \tag{9.1}$$

Voltage shifting circuits are used to match the range of A/D converter. The V_{out} of the voltage shifting circuits are given by

$$V_O = -\left[\frac{R_L}{R_2}(-V_{IN}) + \frac{R_L}{R_3}(V_{ref})\right] \tag{9.2}$$

$$= \frac{R_L}{R_2}V_{IN} - \frac{R_L}{R_3}V_{ref} \tag{9.3}$$

By selecting appropriate values of resistors and adjusting voltage V_{ref}, the signal conditioning circuits can be designed to match the range of A/D converter. The algorithm for measuring and displaying the temperature or pressure is as follows

- Initialise the 8255A port as input port.
- Take one sample from the transducer and convert to digital value.
- Read the value of the port.
- Using look up table, get the corresponding temperature or pressure or humidity value.
- Display the value.

The results can be displayed using seven-segment display or LCD. The A/D conversion program and display program have been discussed in Section 6.7. Hence, this program has been left to the students as an exercise.

SECTION REVIEW

1. Transducers are used to convert non-electrical to electrical signals. True/False?
2. List the specifications for selecting a sensor/transducer.
3. _____ sensor is used to convert water vapour content in air to voltage.
4. The operation range of LM324 sensor is –50º to 300ºF. True/False?
5. _____ is used for measurement of pressure.

9.7 DESIGN OF MINIMUM EMBEDDED SYSTEM

Figure 9.23 shows the circuit diagram of a simple 8051-based embedded system. An embedded system is designed by using 8KX8 bit program ROM, 8255A programmable peripheral interface, 7 push button switches and liquid crystal display. The address and data bus are shown in Fig. 9.23. The 8051 sends out the lower 8 bit of the address on the data bus. External latch 74LS373 is used to demultiplex address and data bus using ALE signal as discussed in Section 2.4. 12 MHz crystal is connected to pins XTAL1 and XTAL2. The control bus consists of $\overline{\text{PSEN}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ that help to select program ROM and to perform read and write operations.

Figure 9.23 shows interfacing of 8 Kbytes program ROM. Program ROM requires 13 address lines (A12–A0) to decode 8192 8 registers. The remaining address lines—A15, A14, A13 and $\overline{\text{PSEN}}$ are used as chip select pin for program ROM. Table 9.9 gives the memory address. Rom is selected for address 0000H –1FFFH.

TABLE 9.9 *Memory address table*

Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Program ROM (8 K)	0	0	0	0	0	0	0	0	0 to 1	0	0	0	0	0	0	0
	0	0	0	1	1	1	1	1		1	1	1	1	1	1	1

The 8255A is interfaced with the 8051 microcontroller such that the data bus is connected to data lines of the 8255A. A1 and A0 lines of address bus are connected to A1 and A0 pins of the 8255A. The remaining address lines—A7, A6, A5, A4, A3 and A2 are connected to decoder 74LS138. Y0 of decoder is connected to $\overline{\text{CS}}$ of 8255. Seven push button switches are connected using P1.0 to P1.6 of the 8051 and LCD is selected for address 47H.

TABLE 9.10 *8255-address table*

A7	A6	A5	A4	A3	A2	A1	A0	Port
0	0	1	0	0	0	0	0	Port A
0	0	1	0	0	0	0	1	Port B
0	0	1	0	0	0	1	0	Port C
0	0	1	0	0	0	1	1	Control register

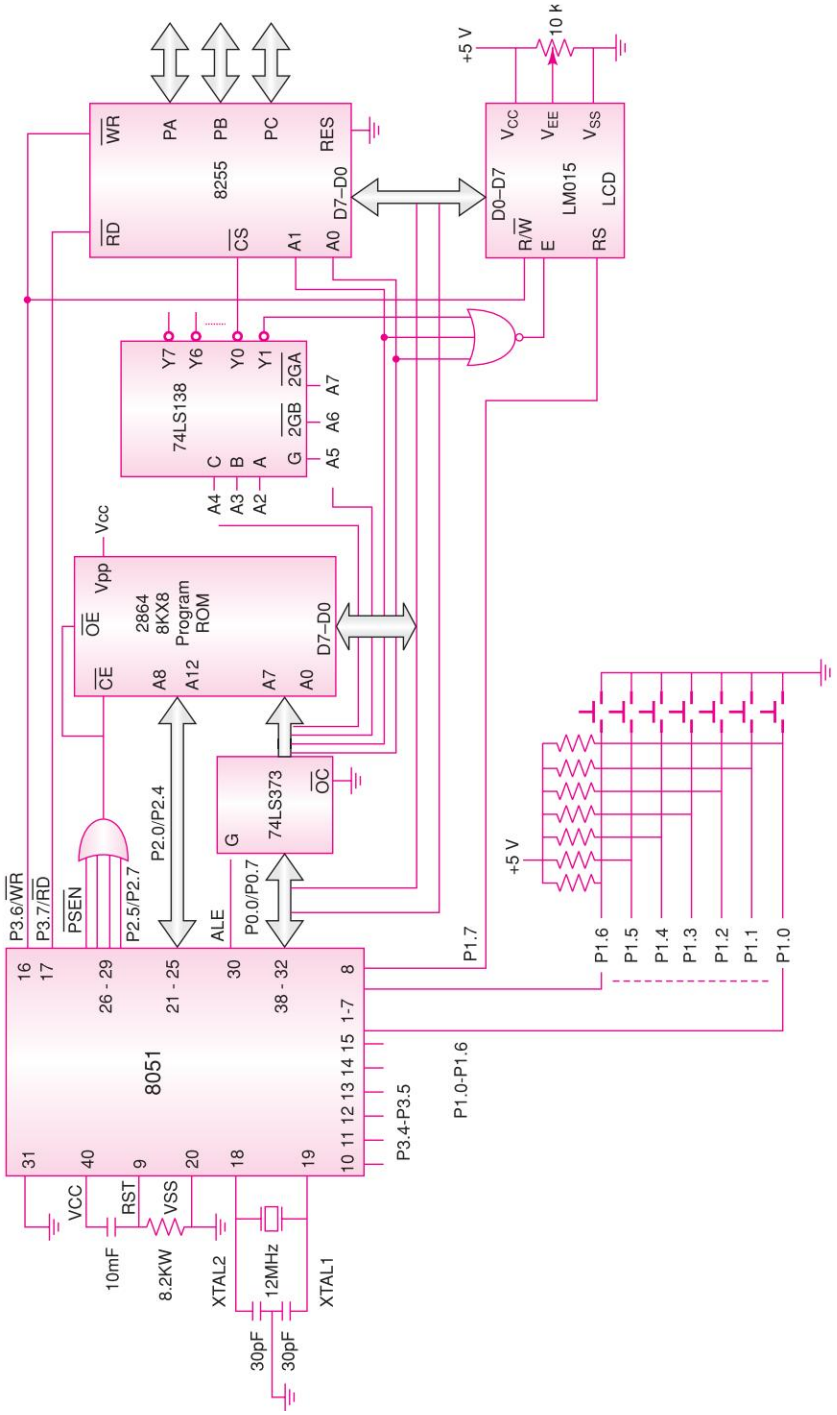


Figure 9.23 *Minimum embedded system-using 8051*

Table 9.10 gives the address of port A, B, C and control register. Port A is selected for address 40H, port B is select ed for address 41H, port C is selected for address 42H and control register is selected for address 43H.

9.8 8051 BASED PROJECTS

PROJECT 1

Title

Blue Tooth Based Security System for Process Industry

Objective Design and implement a security system using blue tooth, which sends a warning message to the administrator regarding any undesired condition in the process industry.

Description The system consists of temperature sensor, blue tooth transmitter and receiver, and also LDR, which is used as smoke and intruder detector. Sensors are interfaced to the 8051 microcontroller, which is programmed to send a message to the blue tooth transmitter. The blue tooth receiver is connected to the computer of the administrator. When the computer receives a warning message, it automatically puts off the power and turns on a buzzer.

PROJECT 2

Title

Automated Food Processing System

Objective Design an automated food processing machine, which provides a user friendly and efficient means for food processing.

Description In food processing, a constant temperature is to be maintained throughout the vessel. A heating coil heats the vessel uniformly. The temperature of the vessel is sensed by thermocouple. The 8051 microcontroller is programmed to switch the heating coil ON and OFF, when the vessel is heated to setpoint temperature. The LCD is connected to the microcontroller to display the desired temperature of the vessel. The microcontroller also drives a stepper motor, which rotates the stirrer in clockwise and anticlockwise direction alternatively. The rotation of the stirrer is done at equal intervals in both the directions to provide uniform mixing of the ingredients.

PROJECT 3

Title

Data Acquisition and Navigation Control of Robot

Objective Design a system to acquire data from remote location using navigational robot.

Description LM35 temperature sensor on the robot is connected to ADC. The digitised data is encoded and transmitted using RF transmitter. At the other end, the data is received, decoded and displayed on LCD.

The commands are issued using matrix keyboard interfaced to a microcontroller. The microcontroller sends the signals through encoder and RF transmitter. In the robot, the RF receiver decodes the signal and drives the DC motors of the robot. The path traced by the robot is detected using a camera on the chassis and monitored using a PC.

PROJECT 4

Title

Measurement of Wobbling of Vehicle Tyres

Objective Design a measurement system for wobbling of tyres using IR sensor.

Description The distance between Infrared sensor and tyre is measured using IR sensor. The sensor output is connected to ADC. The microcontroller is interfaced to ADC. The microcontroller program compares the measured values with ISO standard values. The micrcontroller is connected with the LCD, to display the message 'Tyre under test is OK' or 'not OK'.

PROJECT 5

Title

Wireless Controlled Rotorcraft

Objective Control the speed and direction of the rotorcraft using RF.

Description Rotorcraft is the hybridisation of both helicopter and aircraft. Varying the speed of left wing and right wing controls the engine speed and direction of the rotorcraft. Control signals from ground are sent through RF transmitter. In the rotorcraft onboard, RF receiver receives control signals. The receiver is interfaced to the microcontroller. The microcontroller controls speed of motors and direction of the rotorcraft.

PROJECT 6

Title

GSM Based Process Control System

Objective Control instruments or high end appliances in process plant through mobile phones.

Description Temperature sensor, pressure sensor and heater coil are connected to the micrcontroller using suitable signal conditioning circuits. A program is written to compare the set point values and measured values. If the measured value crosses the set point value, the 8051 microcontroller sends an alert SMS to the process engineer's mobile phone. The GSM module is used for transferring the alert message to the GSM network. An FBUS data cable is used to interface GSM module with the microcontroller.

PROJECT 7

Title

PIR Security System using GSM

Objective Design a security system to protect homes and offices using passive infrared (PIR) sensor and GSM.

Description In this system, PIR sensor, LCD, keyboard and GSM module are interfaced to the 8051 microcontroller. GSM modem is interfaced via serial port. PIR module senses human radiations. The microcontroller is programmed to send an alert SMS message to the authorised person through mobile, and the message is displayed using LCD.

PROJECT 8

Title

Measurement of GAIT Parameters using Ultrasonic Transducers

Objective Design a system to measure GAIT parameters step length and cadence using ultrasonic transducers and foot switches.

Description In this system, ultrasonic transducers and foot switches are interfaced to the microcontroller. GAIT analysis is the process of quantification and interpretation of human locomotion, which reflects pathologies. Cadence is the measurement of speed in steps taken per minute. Foot switches are placed on the heel. When a person walks, the heel is pressed which in turn presses the foot switch. At that instant, the timer in microcontroller is turned ON. When the other heel touches the ground, the timer is turned OFF. In this manner, the time taken for a single step is computed. Ultrasonic transmitter is placed at the tip of one foot, and receiver is placed at the heel of the other foot. Ultrasonic pulses are sent, when the foot switch is turned ON and the timer is also turned ON at the same instant. When the pulse of the other foot is received, the timer is turned OFF. Timer indicates the time taken for ultrasonic waves to traverse from the transmitter to receiver—thus the step length is obtained. The output is displayed on an LCD interfaced to the microcontroller.

PROJECT 9

Title

Measurement of Heart Rate and Body Temperature

Objective Design a module to measure and display heart rate and body temperature.

Description The temperature is sensed using LM35. Heart beat stethoscope and microphone combination are used as sensor. These sensors are interfaced to the microcontroller via signal conditioning circuit, filters and ADC. The microcontroller program is written to display body temperature and heart rate using an LCD.

PROJECT 10

Title

Muscle Stimulator

Objective Design a muscle stimulator that produces a series of electrical shocks to stimulate the muscles.

Description Using a microcontroller, pulses of short time duration are generated. These pulses are connected to surface electrode through power amplifier and step up transformer. When the electrodes are connected to human body, these pulses stimulate the muscle. The microcontroller is interfaced to keyboard and LCD. Here, the user can adjust the stimulation time and magnitude. Entire setup is battery operated to avoid electrical hazards.

PROJECT 11

Title

Voice Operated Home Appliances for the Physically Challenged

Objective Design a voice recognition system to aid physically challenged to perform their day-to-day activities.

Description AP7003-02 voice recognition IC is used. It consists of built-in microphone amplifier, ADC, and speech processor. After pre-recording, AP70003 can recognise 12 sentences of 1.5 seconds length. Speech processor is interfaced to the microcontroller. Microcontroller output is transmitted via RF transmitter. Receiver is also interfaced to another microcontroller. Microcontroller is interfaced to various home appliances. Physically challenged person can give a voice command, which is recognised, encoded and transmitted to the receiver. The microcontroller at the receiving end controls the appliances based on the command.

PROJECT 12

Title

ECG Analysis and Telemetry using GSM

Objective Design a portable system for remote monitoring of cardiac activity.

Description ECG signals are acquired, signal conditioned and digitised using ADC. In this system, ADC, LCD, keyboard and GSM modem are interfaced to the 8051 microcontroller. The microcontroller is programmed to calculate heart rate and send SMS using GSM, when arrhythmia conditions arise. Heart rate is displayed using an LCD.

PROJECT 13

Title

Obstacle Detection for Vehicles using Ultra Sound Signals

Objective Design a collision prevention system for vehicles using ultrasound transmitter and receiver.

Description Ultra sound transmitter and receiver are mounted on the vehicle. The ultrasound signals received are converted to audio frequency signals and amplified. Amplified signals are connected to a loudspeaker. Transmitter and receiver are also interfaced to the 8051 microcontroller. The 8051 microcontroller is programmed to calculate the distance between the obstacle and the vehicle. The distance is displayed on an LCD.

PROJECT 14

Title

Automated Parking Lot Billing System

Objective Design and implement a secure and automated parking lot billing system.

Description At the entry end of a parking lot, data is entered using a keyboard. The data includes registration number, and other description like two-wheeler, four wheeler, time, etc. The keyboard and LCD are interfaced to the 8051 microcontroller. The data entered for the vehicle is also displayed on the LCD. This data is sent to the PC via RS232 interface and the data is stored in the database. The PC acknowledges the valid entry of data. The printer interfaced to the microcontroller prints the details. The printed slip is retained by the owner of the vehicle, which is later used at the exit end. At the exit end, a microcontroller system is used to enter the data and transmit the same to the PC. When the details are entered at the exit end, the program residing in the PC calculates the time of parking and prepares the bill. The bill with details is printed and displayed on LCD.

PROJECT 15

Title

Electric Guitar Tuner

Objective Design a simple electric guitar tuner using the 8051 microcontroller.

Description A magnetic pick up connected guitar is used as transducer. The signal from magnetic pickup is in the form of damped sinusoidal voltage. After signal conditioning, the signal is converted to square wave. The frequency of the wave corresponds to the frequency of the string of the guitar. This frequency is used to trigger a counter of the 8051 microcontroller. The microcontroller program compares the frequency with standard frequencies and indicates the message to increase or decrease the tension of the string.

PROJECT 16

Title

RFID based Office Automation

Objective Design RFID based automation system to calculate stay hours and attendance of employees in an office.

Description RFID card reader is interfaced to the 8051 microcontroller. When card is read, the employee code is recorded in the microcontroller. The same data is transmitted to PC via RS232 bus. The microcontroller is also interfaced to LCD and buzzer. When the card is read, the employee number is displayed on the LCD.

CHAPTER SUMMARY

In this chapter extension of I/O ports using the 8255A has been discussed. Architecture and interfacing of the 8255A with the 8051 microcontroller has been elucidated with examples. Different types of sensors and design of signal conditioning circuits have been covered. At the end of the chapter, the design of minimum embedded system is shown with the help of a circuit diagram.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. In the 8255A, BSR or I/O mode is selected by _____ .
   (a) D7 bit of control register (b) D0 bit of control register
   (c) D6 bit of control register (d) D1 bit of control register
2. In the 8255A, if control word is 00H, then _____ .
   (a) PC0 bit is reset (b) PC1 bit is reset (c) PC2 bit is reset (d) PC3 bit is reset
3. In the 8255A, if the control word is E0H, then port A operates _____ .
   (a) As input or output port without handshake signals
   (b) As input or output port with handshake signals
   (c) Bi-directional port with handshake signals
   (d) Bi-directional port without handshake signals
4. _____ control word sets all the ports as output ports.
   (a) 80H (b) C0H (c) 70H (d) F0H
5. If the 8255A is reset, then all the ports function as _____ .
   (a) Latches (b) Unidirectional buffers
   (c) Bi-directional buffers (d) None of the above
6. _____ control word is used to define port A as input, port B as output and port C as input.
   (a) 99H (b) 80H (c) 88H (d) 85H
7. If A1 and A0 pins of the 8255A are 10, then _____ is selected.
   (a) Port A (b) Port B (c) Port C (d) Control register
8. If D6 and D5 bits of control word are 11, then the 8255A operates in _____ .
   (a) Mode 0 (b) Mode 1 (c) Mode 2 (d) Mode 3
9. If $\overline{CS}$ pin of the 8255A is 1, then all the ports of the 8255A _____ .
   (a) are driven to high impedance state (b) are driven to logic 1 state
   (c) are driven to logic 0 state (d) None of the above
10. Operating range of temperature sensor LM34 is _____ .
    (a) –50° to 300° F (b) 0° to 100° F (c) 0° to 100° C (d) 50° to 200° C

11. _____ in the 8255A is bit addressable.
    (a) Port A (b) Port B (c) Port C (d) None of the above
12. Transducer converts _____ .
    (a) Electrical signal to electrical signal (b) Physical signal to electrical signal
    (c) Electrical signal to physical signal (d) Voltage to current
13. Voltage output of BPT pressure sensor is _____ .
    (a) 0 to 5 V (b) 0 to 5 mV (c) 4.5 to 5.5 V (d) 4.5 mV to 5.5 mV
14. Voltage output of humidity sensor IH 3605 is _____ .
    (a) 0.8 to 3.9 V (b) 0.3 to 3 V (c) 0 to 5 mV (d) 0.8 to 3.9 mV
15. Unit for measurement of humidity is _____ .
    (a) mbar (b) RH (c) F (d) Webbers

REVIEW QUESTIONS

9.1 What are handshake signals? Explain the functions of handshake signals in mode 1 operation of the 8255A.

9.2 With the necessary block and timing diagram, explain mode 1 operation of the 8255A.

9.3 Explain with the necessary timing diagram, mode 2 operation of the 8255A.

9.4 List and explain various registers in the 8255A.

9.5 With the necessary control word, explain the operation of the 8255 in BSR mode and mode 0.

9.6 Explain with the necessary diagram, 8 bit A/D converter and seven-segment display interface using the 8255A. Write 8051 ALP to display A/D converter value.

9.7 Explain with the necessary diagram, the application of the 8255A to interface keyboard and display system in mode 1.

9.8 Interface a printer using the 8255A and write a program to print the message 'HELLO'.

9.9 Interface a stepper motor to the 8051 using the 8255 and write an ALP to rotate the stepper motor one and half revolution clockwise.

9.10 Show the parallel printer interface to the 8051 using the 8255A and write timing waveform for transfer of data using handshake signals.

9.11 Design the necessary signal conditioning circuit for interfacing temperature sensor LM34.

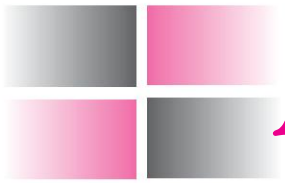
9.12 Design the necessary signal conditioning circuit for interfacing humidity sensor IH-3605.

9.13 Design the necessary signal conditioning circuit for interfacing pressure sensor BPT.

9.14 Interface the 8255A with the 8051 microcontroller such that port A, B, C, and control register are selected for address E000H, E004H, E008H and E00CH respectively.

9.15 Interface the 8255A with the 8051 microcontroller such that port A, B, C, and control register are selected for address E0H, E2H, E4H and E6H respectively.

The McGraw-Hill Companies



Appendix A

P89C60X2/61X2 FLASH MICROCONTROLLER-DATA SHEETS

FEATURES

- 80C51 Central Processing Unit
  - 64 Kbytes Flash
  - 512 bytes RAM (P89C60 2)
  - 1024 bytes RAM (P89C61 2)
  - Boolean processor
  - Fully static operation
- In-System Programmable (ISP) Flash memory
- 12-clock operation with selectable 6-clock operation (via software or via parallel programmer)
- Memory addressing capability
  - Up to 64 Kbytes ROM and 64 Kbytes RAM
- Power control modes
  - Clock can be stopped and resumed
  - Idle mode
  - Power-down mode
- Two speed ranges
  - 0 to 20 MHz with 6-clock operation
  - 0 to 33 MHz with 12-clock operation
- LQFP, PLCC, and DIP packages
- Dual Data Pointers
- Three security bits
- Four interrupt priority levels
- Six interrupt sources
- Four 8 bit I/O ports
- Full-duplex enhanced UART
  - Framing error detection
  - Automatic address recognition
- Three 16 bit timers/counters T0, T1 (standard 80C51) and additional T2 (capture and

compare)
- Programmable clock-out pin
- Watchdog timer
- Asynchronous port reset
- Low EMI (inhibit ALE, 6-clock mode)
- Wake-up from Power Down by an external interrupt

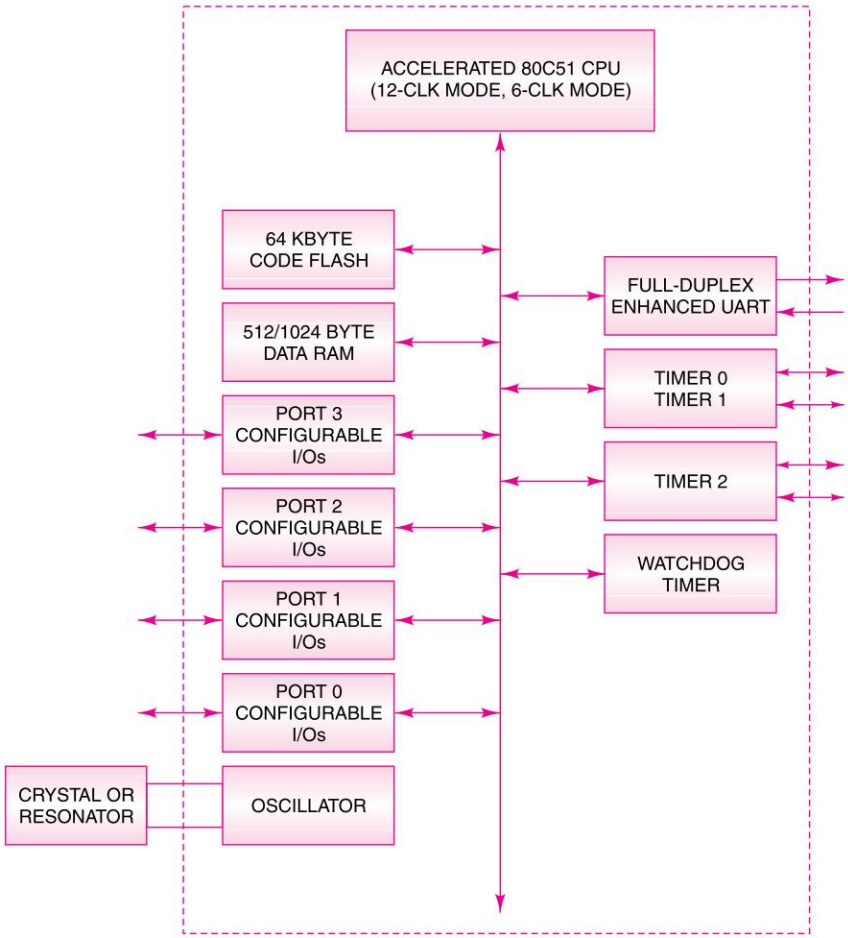


Figure 1 *Block diagram of Flash Microcontroller*

FLASH EPROM MEMORY

GENERAL DESCRIPTION

The P89C60×2/61×2 Flash memory augments EPROM functionality with in-circuit electrical erasure and programming. The Flash can be read and written as bytes. The Chip Erase operation will erase the entire progaram memory. The Block Erase function can erase any Flash block.

In-system programming (ISP) and standard parallel programming are both available. On-chip erase and write timing generation contribute to a user friendly programming interface.

The P89C60×2/61×2 Flash reliably stores memory contents even after 10,000 erase and program cycles. The cell in designed to optimise the erase and programming mechanisms. In addition, the combination of advanced tunnel oxide processing and low internal electric fields for erase and programming operations produced reliable cycling. The P89C60X2/61X2 uses a +5 V Vpp supply to perform the Program/Erase algorithms (12 V tolerant).

FEATURES

- Flash EPROM internal program memory with Block Erase.
- Internal 1 Kbyte fixed BootROM, containing low-level in-system programming routines and a default serial loader.
- Loader in BootROM allows in-system programming via the serial port.
- Up to 64 Kbytes external program memory, if the internal program memory is disabled ($\overline{EA} = 0$)
- Programming and erase voltage + 5 V (+12 V tolerant).
- Read/Programming/Erase using ISP:
  - Byte Programming (8 µs).
  - Typical erase times:
    Block Erase (4 Kbytes) in 3 seconds.
    Full-chip erase in 15 seconds.
- Parallel programming with 87C51 compatible hardware interface to programmer.
- Programmable security for the code in the Flash.
- 10,000 minimum erase/program cycles for each byte.
- 10-year minimum data retention.

FLASH PROGRAMMING AND ERASURE

There and two methods of erasing or programming of the Flash memory that may be used. First, the on-chip ISP boot loader may be invoked. Second, the Flash may be programmed or erased using parallel method by using a commercially available EPROM programmer. The parallel programming method used by these devices is similar to that used by EPROM 87C51, but it is not identical, and the commercially available programmer will need to have support for these devices.

FLASH MEMORY CHARACTERISTICS

Flash User Code Memory Organisation

The P89C60X2/61X2 contains 64 Kbytes Flash user code program memory organised into 4 Kbyte blocks (see Figure 1).

Boot ROM

When the microcontroller programs its Flash memory during ISP, all of the low level details are handled by code that is contained in a 1 Kbyte BootROM. BootROM operations include: erase block, program byte, verify byte, program security bit, etc.

Clock Mode

The clock mode feature sets operating frequency to be 1/12 or 1/6 of the oscillator frequency. The clock mode configuration bit, FX2, is located in the Security Block (See Table 1). FX2, when programmed, will override the SFR clock mode bit

(X2) in the CKCON register. If FX2 is erased, then the SFR bit (X2) may be used to select between 6-clock and 12-clock mode.

In-System Programming (ISP)

The In-System Programming (ISP) is performed without removing the microcontroller from the

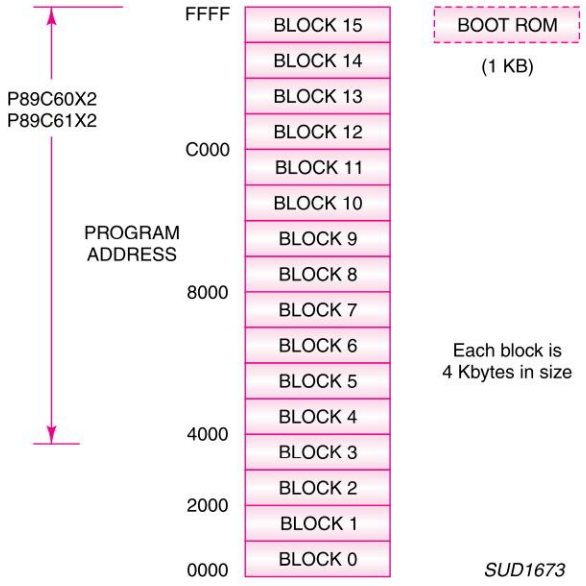


Figure 1 *Flash Memory Configuration*

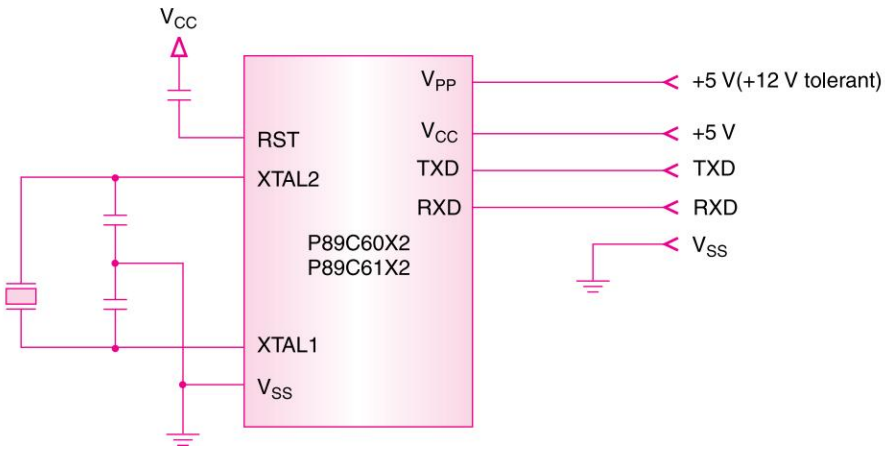


Figure 2 *In-System programming with a minimum of pins*

system. The In-System Programming (ISP) facility consists of a series of internal hardware resources coupled with internal firmware to facilitate remote programming of the P89C60X2/61X2 through the serial port. This firmware is provided by Philips and embedded within each P89C60X2/61X2 device.

The Philips In-System Programming (ISP) facility has made in-circuit programming in an embedded appplication possible with a minimum of additional expencse in components and circuit board area.

The ISP function uses five pins: T D, R D, V_{SS}, V_{CC}, and V_{PP} (see Figure 2). Only a small connector needs to be available to interface your application to an external circuit in order to use this feature. The V_{PP} supply should be adequately decoupled and V_{PP} not allowed to exceed datasheet limits.

Free ISP software is available from the Embedded Systems Academy. "FlashMagic"

1. Direct your browser to the following page: http://www.esacademy.com/software/flashmagic/
2. Download Flashmagic
3. Execute "flashmagic.exe" to install the software

Using the In-System Programming (ISP)

The ISP feature allows for a wide range of baud rates to be used in your application, independent of the oscillator frequency. It is also adaptable to a wide range of oscillator frequencies. This is accomplished by measuring the bit-time of a single bit in a received character. This information is then used to program the baud rate in terms of timer counts based on the oscillator frequency. The ISP feature requires that an initial character (an uppercase U) be sent to the P89C60X2/61X2 to establish the baud rate. The ISP firmware provides auto-echo of received characters.

Once baud rate initialisation has been performed, the ISP firmware will only accept intel Hex-type records. Intel Hex records consist of ASCII characters used to represent hexadecimal values and are summarised below:

:NNAAAARRDD..DDCC<crif>

In the Intel Hex record, the "NN" represents the number of data bytes in the record. The P89C60X2/61X2 will accept up to 16 (10H) data bytes. The "AAAA" string represents the address of the first byte in the record. If there are zero bytes in the record, this field is often set to 0000. The "RR" string indicates the record type. A record type of "00" is a data record type. A record type of "01" indicates the end-of-file mark. In this application, additional record types will be added to indicate either commands or data for the ISP facility. The maximum number of data bytes in a record is limited to 16 (decimal). ISP commands are

TABLE 1

CLOCK MODE CONFIG BIT (FX2)	X2 bit in CKCON	DESCRIPTION
erased	0	12-clock mode (default)
erased	1	6-clock mode
programmed	x	6-clock mode

Note

1. Default clock mode after ChipErase is set to 12-clock.

TABLE 2 *Intel-Hex records used by In-System programming*

RECORD TYPE	COMMAND/DATA FUNCTION
00	**Program Data** :nnaaaa00dd....ddcc **Where:** nn = number of bytes (hex) in record aaaa = memory address of first byte in record dd....dd = date bytes cc = checksum **Example:** : 10008000AF5F67F0602703E0322CFA92007780C3FD
01	**End of File (EOF), no operation** : xxxxxx01cc **Where:** xxxxxx = required field, but value is a "don't care" cc = checksum **Example:** : 00000001FF
03	**Miscellaneous Write Functions** :nnxxxx03ffssddcc **Where:** nn = number of bytes (hex) in record xxxx = required field, but value is a "don't care" 03 = Write Function ff = subfunction code ss = selection code dd = data input (as needed) cc = checksum **Subfunction Code = 04 (Set Status Byte to 00h)** ff = 04 ss = don't care **Example:** : 020000030400F7 set status byte to 00h (device executes user code after Reset) **Subfunction Code = 05 (Program Security Bits)** ff = 05 ss = 00 program security bit 1 (inhibit wiriting to Flash) 01 program security bit 2 (inhibit Flash verify) 02 program security bit 3 (disable external memory) **Example:** : 020000030501F5 program security bit 2 **Subfunction Code = 06 (Program Flash X2 bit)** ff = 06 ss = 02 program FX2 bit (dd = 80) 6-clk. mode enabled dd = data **Example 1:** : 0300000306028072 program FX2 bit (enable 6-clk. mode)

TABLE 3 *The MCS51 family of microcontrollers*

DEVICE	ROM/EPROM (bytes)	Register RAM (bytes)	Speed (MHz)	I/O Pins	Timer/ Counters	UART	Interrupt Sources	PCA Channels	A/D Channels	SEP	GSC	DMA Channels	Lock Bits	Power Down & Idle Modes
8051 Product Line														
8031AH	ROMLESS	128	12	32	2	1	5	0	0	0	0	0	-	-
8051AH	4K ROM	128	12	32	2	1	5	0	0	0	0	0	0	-
8051AHP	4K ROM	128	12	32	2	1	5	0	0	0	0	0	p	-
8751H	4K EPROM	128	12	32	2	1	5	0	0	0	0	0	1	-
8751BH	4K EPROM	128	12	32	2	1	5	0	0	0	0	0	2	-
8052 Product Line														
8032AH	ROMLESS	256	12	32	3	1	6	0	0	0	0	0	-	-
8052AH	K ROM	256	12	32	3	1	6	0	0	0	0	0	0	-
8752BH	K EPROM	256	12	32	3	1	6	0	0	0	0	0	2	-
80C51 Product Line														
80C31BH	ROMLESS	128	12,16	32	2	1	5	0	0	0	0	0	-	Yes
80C51BH	4K ROM	128	12,16	32	2	1	5	0	0	0	0	0	0	Yes
80C51BHP	4K ROM	128	12,16	32	2	1	5	0	0	0	0	0	p	Yes
87C51	4K EPROM	128	12,16,20,24 I	32	2	1	5	0	0	0	0	0	3	Yes
8XC52/54/58 Product Line														
80C32	ROMLESS	256	12,16,20,24 I	32	3	1	6	0	0	0	0	0	-	Yes
80C52	8K ROM	256	12,16,20,24 I	32	3	1	6	0	0	0	0	0	1*	Yes
87C52	8K EPROM	256	12,16,20,24 I	32	3	1	6	0	0	0	0	0	3	Yes
80C54	16K ROM	256	12,16,20,24 I	32	3	1	6	0	0	0	0	0	1	Yes
87C54	16K EPROM	256	12,16,20,24 I	32	3	1	6	0	0	0	0	0	3	Yes
80C58	32K ROM	256	12,16,20,24 I	32	3	1	6	0	0	0	0	0	1	Yes
87C58	32K EPROM	256	12,16,20,24	32	3	1	6	0	0	0	0	0	3	Yes
8XC52/54/58 Product Line														
80L52	8K ROM	256	12,16,20*	32	3	1	6	0	0	0	0	0	1	Yes
87L52	8K OTP ROM	256	12,16,20*	32	3	1	6	0	0	0	0	0	3	Yes
80L54	18K ROM	256	12,16,20*	32	3	1	6	0	0	0	0	0	1	Yes
87L54	16K OTP ROM	256	12,16,20*	32	3	1	6	0	0	0	0	0	3	Yes
80L58	32K ROM	256	12,16,20*	32	3	1	6	0	0	0	0	0	1	Yes
87L58	32K OTP ROM	256	12,16,20*	32	3	1	6	0	0	0	0	0	3	Yes

(Contd)

DEVICE	ROM/EPROM (bytes)	Register RAM (bytes)	Speed (MHz)	I/O Pins	Timer/ Counters	UART	Interrupt Sources	PCA Channels	A/D Channels	SEP	GSC	DMA Channels	Lock Bits	Power Down & Idle Modes
8XC51FA/FB/FC Product Line														
80C51FA	ROMLESS	256	12,16	32	3	1	7	5	0	0	0	0	-	Yes
83C51FA	8K ROM	256	12,16	32	3	1	7	5	0	0	0	0	0	Yes
87C51FA	8K EPROM	256	12,16,20,24 I	32	3	1	7	5	0	0	0	0	3	Yes
83C51FB	16K ROM	256	12,16,20,24 I	32	3	1	7	5	0	0	0	0	1	Yes
87C51FB	16K EPROM	256	12,16,20,24 I	32	3	1	7	5	0	0	0	0	3	Yes
83C51FC	32K ROM	256	12,16,20,24 I	32	3	1	7	5	0	0	0	0	1	Yes
87C51FC	32K EPROM	256	12,16,20,24 I	32	3	1	7	5	0	0	0	0	3	Yes
8XL51FA/FB/FC Product Line														
80L51FA	ROMLESS	256	12,16,20*	32	3	1	7	5	0	0	0	0	-	Yes
83L51FA	8K ROM	256	12,16,20*	32	3	1	7	5	0	0	0	0	1	Yes
87L51FA	8K OTP ROM	256	12,16,20*	32	3	1	7	5	0	0	0	0	3	Yes
83L51FB	16K ROM	256	12,16,20*	32	3	1	7	5	0	0	0	0	1	Yes
87L51FB	16K OTP ROM	256	12,16,20*	32	3	1	7	5	0	0	0	0	3	Yes
83L51FC	32K ROM	256	12,16,20*	32	3	1	7	5	0	0	0	0	1	Yes
87L51FC	32K OTP ROM	256	12,16,20*	32	3	1	7	5	0	0	0	0	3	Yes
8XC51GX Product Line														
80C51GB	ROMLESS	256	12,16	48	3	1	15	10	8	1	0	0	-	Yes
83C51GB	8K ROM	256	12,16	48	3	1	15	10	8	1	0	0	1	Yes
87C51GB	8K EPROM	256	12,16	48	3	1	15	10	8	1	0	0	3	Yes
8XC1522 Product Line*														
80C152JA	ROMLESS	256	16,5	40	2	1	11	0	0	1	1	2	-	Yes
80C152JB	ROMLESS	256	16,5	58	2	1	11	0	0	1	1	2	-	Yes
83C152JA	8K ROM	256	16,5	40	2	1	11	0	0	1	1	2	0	Yes
8XC51SL Product Line														
80C51SL-BG	ROMLESS	256	16	24	2	1	10	0	4	0	1	0	-	Yes
81C51SL-BG	8K *ROM	256	16	24	2	1	10	0	4	0	1	0	0	Yes
83C51SL-BG	8K ROM	256	16	24	2	1	10	0	4	0	1	0	0	Yes
80C51SLAH	ROMLESS	256	16	24	2	1	10	0	4	0	1	0	-	Yes
81C51SLAH	16K *ROM	256	16	24	2	1	10	0	4	0	1	0	0	Yes
83C51SLAH	16K ROM	256	16	24	2	1	10	0	4	0	1	0	0	Yes
87C51SLAH	16K EPROM	256	16	24	2	1	10	0	4	0	1	0	0	Yes
80C51SLAL	ROMLESS	256	16	24	2	1	10	0	4	0	1	0	-	Yes
81C51SLAL	16K *ROM	256	16	24	2	1	10	0	4	0	1	0	0	Yes
83C51SLAL	16K ROM	256	16	24	2	1	10	0	4	0	1	0	0	Yes
87C51SLAL	16K EPROM	256	16	24	2	1	10	0	4	0	1	0	0	Yes

ROM/OTP ROM/EPROM (bytes):
Speed (MHz);

Lock Bits:

8XC152 Product Line*
8XC51SL Product Line*

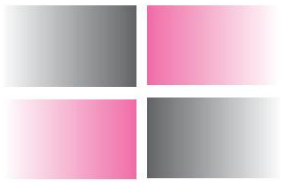
*ROM = SystemSoft Standard BIOS
24 = 24 MHz internal-only operation
20* = 20 MHz Available for Commerical Temperature Range Only
1* = 1 Lock Bit for 20 MHz & 24 MHz parts, no Lock Bit for 12 & 16 MHz parts
P = Program verification disabled, external memory access limited to 4K
= Communication Controller
= Keyboard Controller

summarised in Table 2.

As a record is received by the P89C60X2/61X2, the information in the record is stored internally and a checksum calculation is performed. The operation indicated by the record type is not perfomed untill the entire record has been received. Should an error occur in the checksum, the P89C60X2/61X2 will send an "X" out the serial port indicating a checksum error. If the checksum calculation is found to match the checksum in the record, then the command will be executed. In most cases, successful reception of the record will be indicated by transmitting a "." character out the serial port (displaying the contents of the internal program memory is an exception).

In the case of a Data Record (record type 00), an additional check is made. A"." character will NOT be sent unless the record checksum matched the calculated checksum and all of the bytes in the record were successfully programmed. For a data record, an "X" indicates that the checksum failed to match, and an "R" character indicates that one of the bytes did not properly program. It is necessary to send a type 02 record (specify oscillator frequency) to the P89C60X2/61X2 before programming data.

The ISP facility was designed to that specific crystal frequencies were not required in order to generate baud rates or time the programming pulses.



Appendix B

ASCII CODE VALUES

ASCII (DECIMAL)	ASCII (HEXADECIMAL)	CHARACTER
000	0	(NUL)
001	1	A
002	2	B
003	3	ª
004	4	©
005	5	¨
006	6	«
007	7	(BEEP)
008	8	
009	9	(TAB)
010	A	(LF)
011	B	(HOME)
012	C	(FF)
013	D	(CR)
014	E	
015	F	-
016	10	I
017	11	J
018	12	K
019	13	L
020	14	M
021	15	N
022	16	O
023	17	P
024	18	Q
025	19	R
026	1A	S
027	1B	(ESC)
028	1C	(RIGHT)
029	1D	(LEFT)
030	1E	(UP)
031	1F	(DOWN)
032	20	(SPACE)
033	21	!
034	22	“
035	23	#
036	24	$
037	25	%
038	26	&
039	27	‘
040	28	(
041	29	)
042	2A	*
043	2B	+
044	2C	,
045	2D	-
046	2E	.
047	2F	/
048	30	0
049	31	1
050	32	2
051	33	3
052	34	4
053	35	5
054	36	6
055	37	7
056	38	8
057	39	9
058	3A	:
059	3B	;
060	3C	<
061	3D	=
062	3E	>
063	3F	?
064	40	@
065	41	A
066	42	B
067	43	C
068	44	D
069	45	E
070	46	F
071	47	G
072	48	H
073	49	I
074	4A	J
075	4B	K
076	4C	L
077	4D	M
078	4E	N
079	4F	O
080	50	P
081	51	Q
082	52	R
083	53	S
084	54	T
085	55	U
086	56	V
087	57	W
088	58	X
089	59	Y
090	5A	Z
091	5B	[
092	5C	\

093	5D	]
094	5E	^
095	5F	_
096	60	‘
097	61	A
098	62	B
099	63	C
100	64	D
101	65	E
102	66	F
103	67	G
104	68	H

105	69	I
106	6A	J
107	6B	K
108	6C	L
109	6D	M
110	6E	N
111	6F	O
112	70	P
113	71	Q
114	72	R
115	73	S
116	74	T

117	75	U
118	76	V
119	77	W
120	78	X
121	79	Y
122	7A	Z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	T

Appendix C

74LS373, ADC 0808, DAC 0808-DATA SHEETS

The user thus needs to provide the P89C60X2/61X2 with information required to generate the proper timing. Record type 02 is provided for this purpose.

OCTAL TRANSPARENT LATCH WITH 3-STATE OUTPUTS; OCTAL D-TYPE FLIP-FLOP WITH 3-STATE OUTPUT

The SN54/74LS373 consists of eight latches with 3-state outputs for bus organised system applications. The flip-flops appear transparent to the data (data changes asynchronously) when Latch Enable (LE) is HIGH. When LE is LOW, the data that meets the setup times is latched. Data appears on the bus when the Output Enable ($\overline{OE}$) is LOW. When $\overline{OE}$ is HIGH, the bus output is in the high impedance state.

The SN54/74LS374 is a high-speed, low-power Octal D-type Flip-Flop featuring separate D-type inputs for each flip-flop and 3-state outputs for bus oriented applications. A buffered Clock (CP) and Output Enable (OE) is common to all flip-flops. The SN54/74LS374 is manufactured using advanced LOW Power Schottky technology and is compatible with all Motorola TTL families.

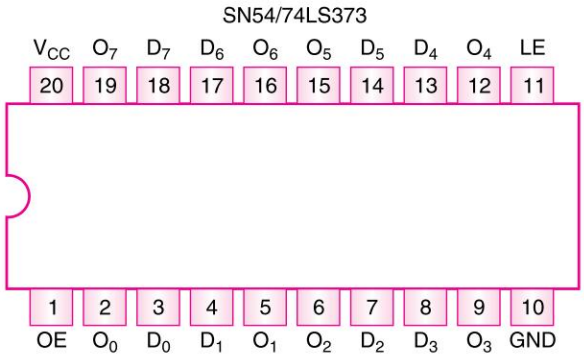
- ∞ Eight Latches in a Single Package
- ∞ 3-State Outputs for Bus interfacing
- ∞ Hysteresis on Latch Enable
- ∞ Edge-Triggered D-Type Inputs
- ∞ Buffered Positive Edge-Triggered Clock
- ∞ Hysteresis on Clock Input to Improve Noise Margin
- ∞ Input Clamp Diodes Limit High Speed Termination Effects

PIN NAMES

$D_0 - D_7$	Data Inputs
LE	Latch Enable (Active HIGH) Input
CP	Clock (Active HIGH going edge) Input
$\overline{OE}$	Output Enable (Active LOW) Input
$O_0 - O_7$	Outputs (Note b)

LOADING (Note a)

HIGH	LOW
0.5 U.L.	0.25 U.L.
0.5 U.L.	0.25 U.L.
0.5 U.L.	0.25 U.L
0.5 U.L.	0.25 U.L.
65 (25) U.L.	15 (7.5) U.L.



LS373

D_n	LE	OE	O_n
H	H	L	H
L	H	L	L
X	L	L	Q_0
X	X	H	Z*

H = High Voltage Level
L = Low Voltage Level
X = Immaterial
Z = High Impedance

Pin diagram of SN54/74LS373

ADC0808/ADC0809 8-BIT µP COMPATIBLE A/D CONVERTERS WITH 8-CHANNEL MULTIPLEXER

GENERAL DESCRIPTION

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8 bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8 bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilised comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals.

The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE* outputs.

The design of the ADC0808, ADC0809 has been optimised by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications.

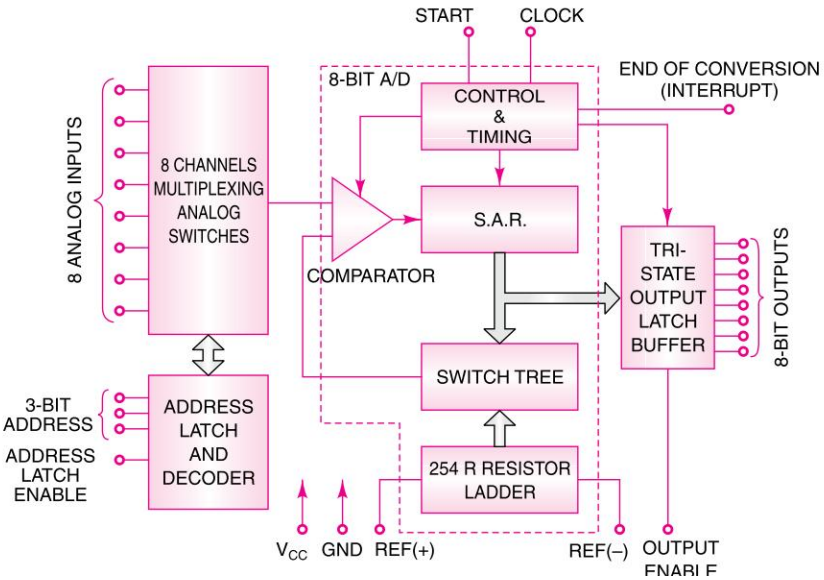
FEATURES

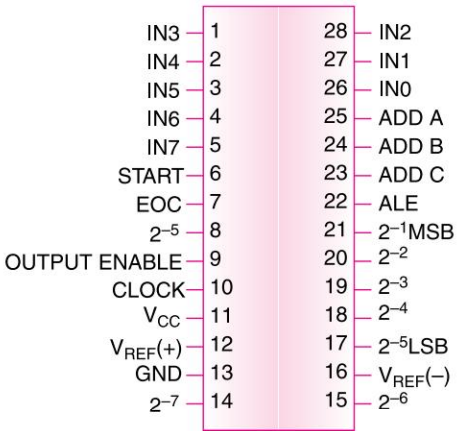
- Easy interface to all microprocessors
- Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
- No zero or full-scale adjust required
- 8-channel multiplexer with address logic
- 0 V to 5 V input range with single 5 V power supply
- Outputs meet TTL voltage level specifications
- Standard hermetic or molded 28-pin DIP package
- 28-pin molded chip carrier package
- ADC0808 equivalent to MM74C949
- ADC0809 equivalent to MM74C949-1

KEY SPECIFICATIONS

- Resolution — 8 bit
- Total Unadjusted Error — ±½ LSB and ±1 LSB
- Single Supply — 5 V_{DC}
- Low Power — 15 mW
- Conversion Time — 100 µs

BLOCK DIAGRAM





Pin diagram

FUNCTIONAL DESCRIPTION

Multiplexer

The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. Table 1 shows the input states for the address lines to select any channel. The address is latched into the decoder on the low-to-high transition of the address latch enable signal.

Table 1

Selected Analog Channel	Address line		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

CONVERTER CHARACTERISTICS

The Converter

The heart of this single chip data acquisition system is its 8 bit analog-to-digital converter. The converter is designed to give fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into 3 major sections: the 256R ladder network, the successive approximation register, and the comparator. The converter's digital outputs are positive true.

The 256R ladder network approach (Figure 1) was chosen over the conventional R/2R ladder because of its inherent monotonicity, which guarantees no missing digital codes. Monotonicity is particularly important in closed loop feedback control systems. A non-monotonic relationship can cause oscillations that will be catastrophic for the system. Additionally, the 256R network does not cause load variations on the reference voltage.

The bottom resistor and the top resistor of the ladder network in Figure 1 are not the same value as the remainder of the network. The difference in these resistors causes the output characteristic to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has reached +½ LSB and succeeding output transitions occur every 1 LSB later up to full-scale.

The successive approximation register (SAR) performs 8 iterations to approximate the input voltage. For any SAR type converter, n-iterations are required for an n-bit converter. Figure 2 shows a typical example of a 3 bit converter. In the ADC0808, ADC0809, the approximation technique is extended to 8 bit using the 256R network.

The A/D converter's successive approximation register (SAR) is reset on the positive edge of the start conversion (SC) pulse. The conversion is begun on the falling edge of the start conversion pulse. A conversion in process will be interrupted by receipt of a new start conversion pulse. Continuous conversion

may be accomplished by tying the end-of-conversion (EOC) output to the SC input. If used in this mode, an external start conversion pulse should be applied after power up. End-of-conversion will go low between 0 and 8 clock pulses after the rising edge of start conversion.

The most important section of the A/D converter is the comparator. It is this section which is responsible for the ultimate accuracy of the entire converter. It is also the comparator drift which has the greatest influence on the repeatability of the device. A chopper-stabilised comparator provides the most effective method of satisfying all the converter requirements.

The chopper-stabilised comparator converts the DC input signal into an AC signal. This signal is then fed through a high gain AC amplifier and has the DC level restored. This technique limits the drift component of the amplifier since the drift is a DC component which is not passed by the AC amplifier. This makes the entire A/D converter extremely insensitive to temperature, long term drift and input offset errors. Figure 4 shows a typical error curve for the ADC0808 as measured using the procedures outlined in AN-179.

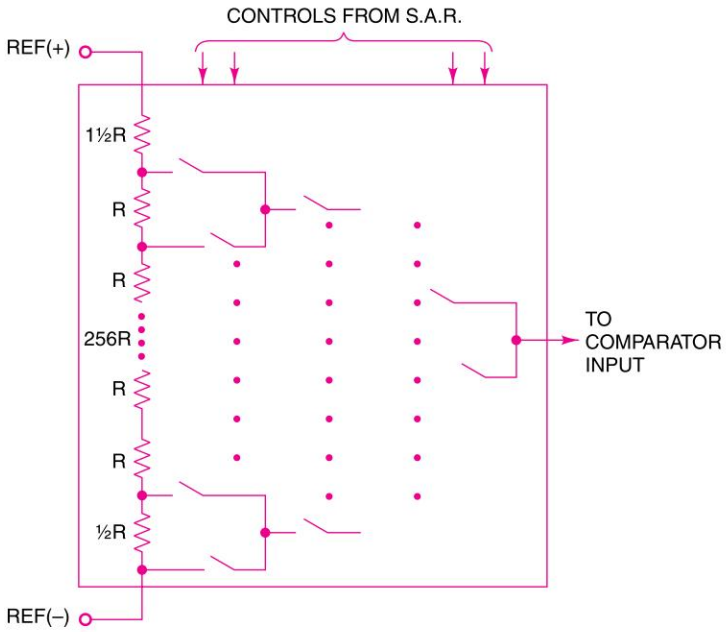


Figure 1 *Resistor Ladder and Switch Tree*

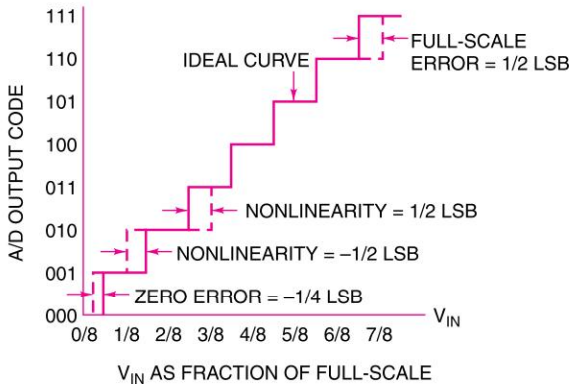


Figure 2 *3 bit A/D Transfer curve*

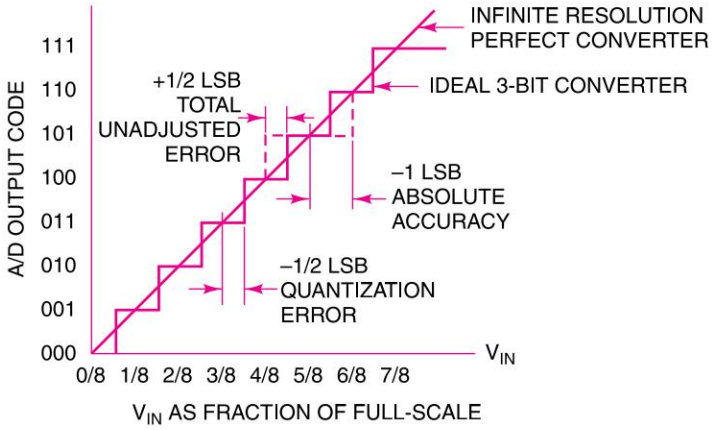


Figure 3 *3 bit A/D absolute accuracy curve*

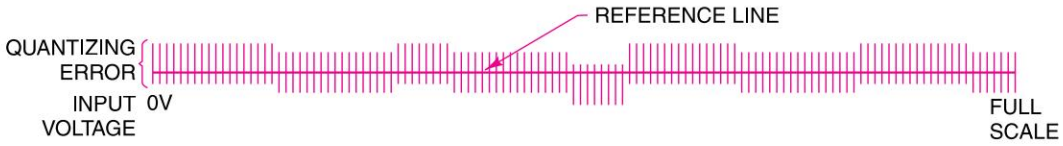


Figure 4 *Typical error curve*

DAC0808
8-BIT D/A CONVERTER

GENERAL DESCRIPTION

The DAC0808 is an 8 bit monolithic digital-to-analog converter (DAC) featuring a full scale output current setting time of 150 ns while dissipating only 33 mW with ±5 V supplies. NO reference current (I_{REF}) trimming is required for most applications since the full scale output current is typically ±1 LSB of 255 $I_{REF}/256$. Relative accuracies of better than ±0.19% assure 8-bit monotonicity and linearity while zero level output current of less than 4 µA provides 8 bit zero accuracy for $I_{REF} \geq 2$ mA. The power supply currents of the DAC0808 is independent of bit codes, and exhibits essentially constant device characteristics over the entire supply voltage range.

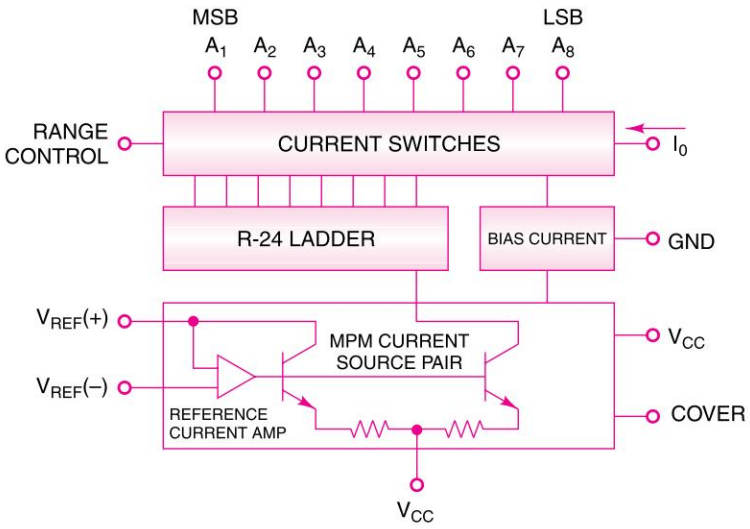
The DAC0808 will interface directly with popular TTL, DTL or CMOS logic levels, and is a direct replacement for the MC1508/MC1408. For higher speed applications, see DAC0800 data sheet.

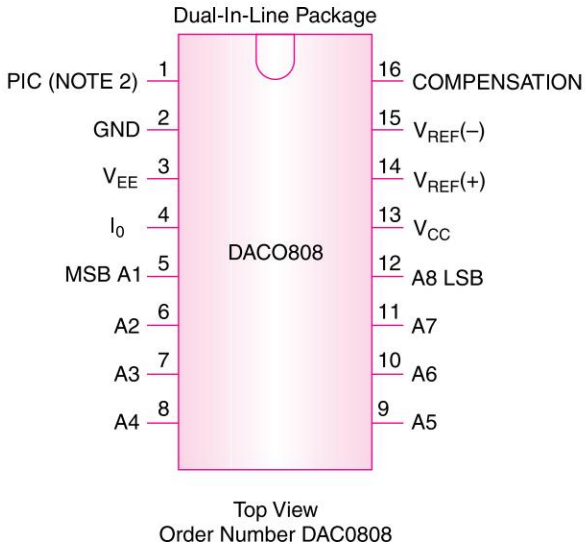
FEATURES

- Relative accuracy: ±0.19% error maximum
- Full scale current match: ±1 LSB typ
- Fast settling time: 150 ns typ
- Noninverting digital inputs are TTL and CMOS compatible

- High speed multiplying input slew rate: 8 mA/μs
- Power supply voltage range: ±4.5 V to ±18 V
- Low power consumption: 33 mW @ ±5 V

BLOCK AND CONNECTION DIAGRAMS





Top View
Order Number DAC0808

Appendix D

8052 MICRO-CONTROLLER-DATA SHEETS

MEMORY ORGANISATION OF 8052

8052 Program memory and Data memory is as shown in Fig. 1 and 2.

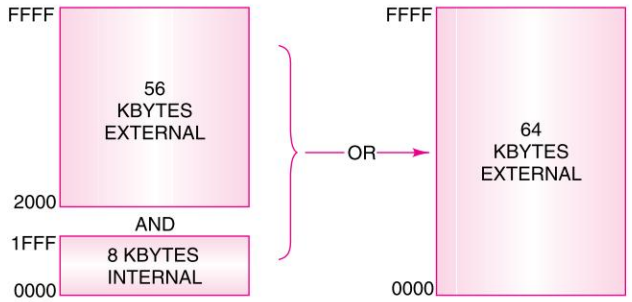


Figure 1 *8052 Program memory*

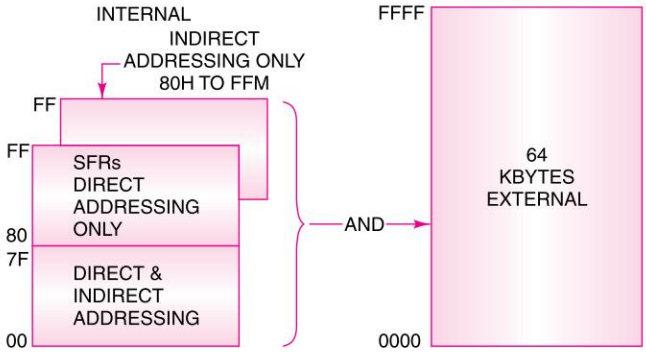


Figure 2 *8052 Program memory*

T2CON: TIMER/COUNTER 2 CONTROL REGISTER, BIT ADDRESSABLE

8052 ONLY

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

TF2 T2CON. 7 Timer 2 overflow flag set by hardware and cleared by software. TF2 cannot be set when either RCLK = 1 or CLK = 1

EXF2 T2CON. 6 Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX, and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.

RCLK T2CON. 5 Receive clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its receive clock in modes 1 & 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.

TLCK T2CON. 4 Transmit clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its transmit clock in modes 1 & 3. TCLK = 0 causes Timer 1 overlows to be used or the transmit clock.

EXEN2 T2CON. 3 Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of negative transition on T2EX if Timer 2 is not being used to clock the Serial Port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.

TR2 T2CON. 2 Software START/STOP control for Timer 2. A logic 1 starts the Timer.

C/$\overline{T2}$ T2CON. 1 Timer or Counter select.
0 = Internal Timer. 1 = External Event Counter (falling edge triggered).

CP/$\overline{RL2}$ T2CON. 0 Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, Auto-Reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the Timer is forced to Auto-Reload on Timer 2 overflow.

TIMER/COUNTER 2 SET-UP

Except for the baud rate generator mode, the values given for T2CON do not include the setting of the TR2 bit. Therefore, bit TR2 must be set, separately, to turn the Timer on.

AS A TIMER

TABLE 7

Mode	T2 CON	
	Internal Control (Note 1)	External Control (Note 2)
16 bit Auto-Reload	00H	08H
16 bit Capture	01H	09H
BAUD rate generator receive & transmit same baud rate	34H	36H
receive only	24H	26H
transmit only	14H	16H

AS A COUNTER

TABLE 8

Mode	TMOD	
	Internal Control (Note 1)	External Control (Note 2)
16 bit Auto-Reload	02H	0AH
16 bit Capture	03H	0BH

Notes

1. *Capture/Reload occurs only on Timer/Counter overflow.*
2. *Capture/Reload occurs on Timer/Counter overflow and a 1 to 0 transition on T2EX (P1.1) pin except when Timer 2 is used in the baud rate generating mode.*

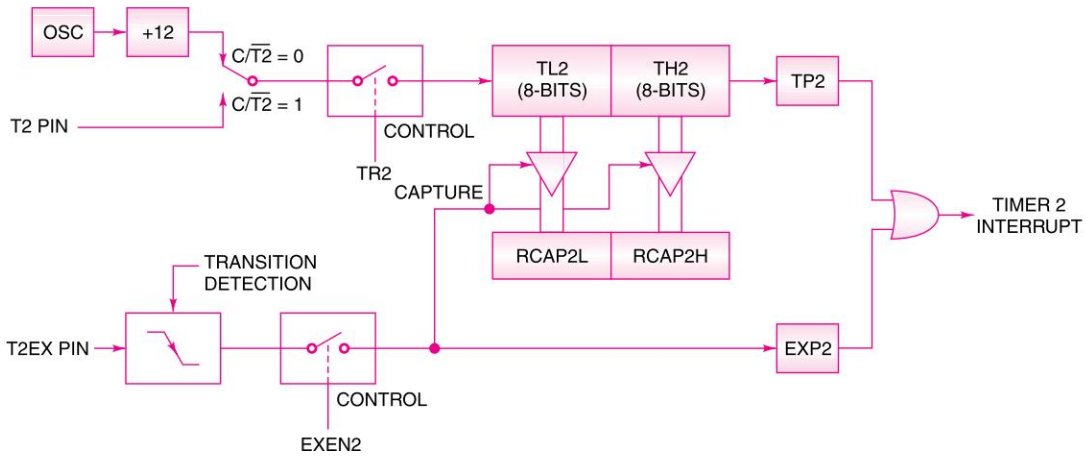


Figure 3 *Timer 2 in capture mode*

The serial port can operate in 4 modes:

Mode 0 Serial data enters and exits through RXD.TXD outputs the shift clock. 8 bit are transmitted/ received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Mode 1 10 bit are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

Mode 2 11 bit are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On Transimit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8. ON

receive, the 9th data bit goes into RB8 in Special Function Register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency.

Mode 3 11 bit are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit of REN = 1.

MULTIPROCESSOR COMMUNICATIONS

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignorng the coming data bytes.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

SERIAL PORT CONTROL REGISTER

The serial port control and status register is the Specail Function Register SCON, shown in Figure 5. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8), and the serial port interrupt bits (TI and RI).

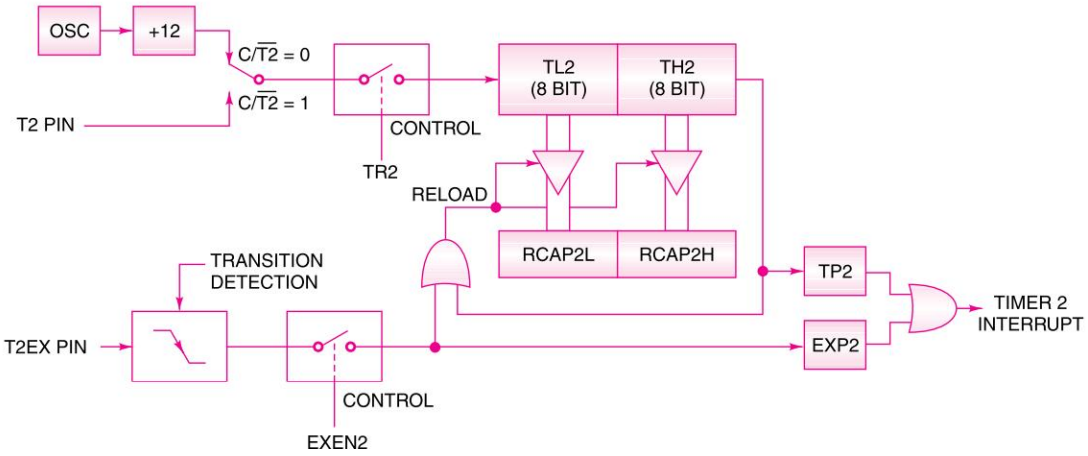


Figure 4 *Timer 2 in Auto-Reload mode*

(MSB) (LSB)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Where SM0, SM1 specify the serial port mode, as follows:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	fosc./12
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	fosc./64 or fosc./32
1	1	3	9-bit UART	variable

- SM2 enables the multiprocessor communication feature in Modes 2 and 3. In Mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In Mode 1, if SM2 – 1 than RI will not be activated if a valid stop bit was not received. In Mode D, SM2 should be 0.
- REN enables serial reception. Set by software to enable reception. Clear by software to disable reception.
- TB8 is the 9th data bit that will be transmitted in Modes 2 and 3. Set or clear by software as desired.
- FB8 in Modes 2 and 3, is the 9th data bit that was received, in Mode1, if SM2 – 0. RB8 is the stop bit that was received, in Mode 0, RB8 is not used.
- TI is transmit interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must to cleared by software.
- RI is receive interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.

Figure 5 *SCON: Serial Port Control Register*

In the Capture Mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 is a 16 bit timer or counter which upon overflowing sets bit TF2, the Timer 2 overflow bit, which can be used to generate an interrupt. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. (RCAP2L and RCAP2H are new Special Function Registers in the 8052.) In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2, like TF2, can generate an interrupt.

The Capture Mode is illustrated in Figure 3.

In the auto-reload mode there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over it not only sets TF2 but also causes the Timer 2 registers to be reloaded with the 16 bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that 1-to-0 transition at external input T2EX will also trigger the 16 bit reload and set EXF2.

The auto-reload mode is illustrated in Figure 4.

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1. It will be described in conjunction with the serial port.

SERIAL INTERFACE

The serail port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at Specail Function Register SBUF. Writing to SBUF loads the transmit register, and reading SBUF accesses a physically separate receive register.

Appendix E

80196 MICRO-CONTROLLER-DATA SHEETS

80196-ARCHITECTURAL OVERVIEW

The 16 bit 8XC196K*x*, 8XC196J*x*, and 87C196CA CHMOS microcontrollers are designed to handle high-speed calculations and fast input/output (I/O) operations. They share a common architecture and instruction set with other members of the MCS® 96 microcontroller family. Figure 1 shows pin details of 8XC196K*x* and Table 1 signal classification according to functions.

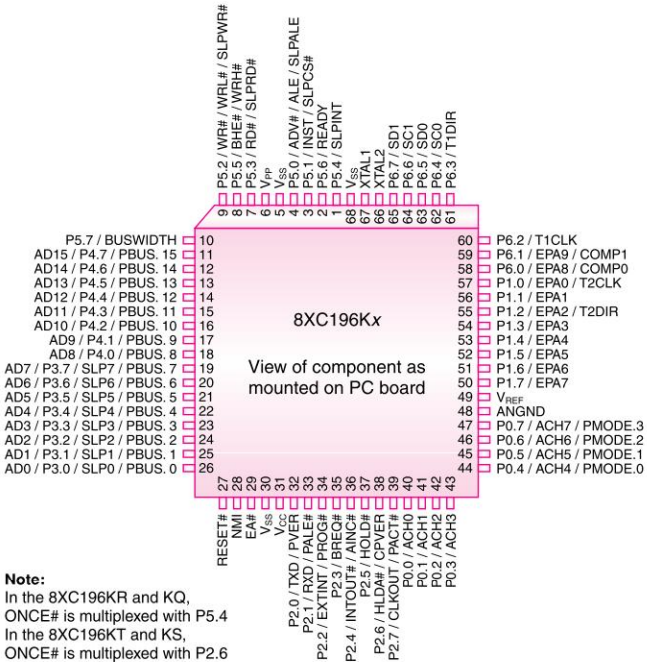


Figure 1 *8XC196Kx 68-lead PLCC package*

TABLE 1 *8xC196Kx signals arranged by functional categories*

Input/Output	Input/Output (Cont'd)	Programming Control	Bus Control & Status
P0.7:0/ACH7:0	P6.5/SD0	AINC#	ALE/ADV#
P1.0/EPA0/T2CLK	P6.6/SC1	CPVER	BHE#WRH#
P1.1/EPA1	P6.7/SD1	PACT#	BREQ#
P1.2/EPA2/T2DIR		PALE#	BUSWIDTH
P1.7:3/EPA7:3	**Processor Control**	PBUS.15:0	CLKOUT
P2.0/TXD	EA#	MPODE.3:0	HOLD#
P2.1/RXD	EXTINT	PROG#	HLDA#
P2.7:2	NMI	PVER	INST
P3.7:0	ONCE#		INTOUT#
P4.7:0	RESET#	**Power & Ground**	READY
P5.7:0	SLPINT	ANGND	RD#
P6.0/EPA8/COMP0	XTAL1	V_{CC}	SLPALE†
P6.1/EPA9/COMP1	XTAL2	V_{PP}	SLPCS#†
P6.2/T1CLK		V_{REF}	SLPWR#†
P6.3/T1DIR	**Address & Data**	V_{SS}	SLPRD#†
P6.4/SCO	AD15:0		WR#/WRL#
	SLP7:0†		

†Slave port signal

TYPICAL APPLICATIONS

MCS 96 microcontrollers are typically used for high-speed event control systems. Commercial applications include modems, motor-control systems, printers, photocopiers, air conditioner control systems, disk drives, and medical instruments. Automotive customers use MCS 96 microcontrollers in engine-control systems, airbags, suspension systems, and antilock braking systems (ABS).

DEVICE FEATURES

Table-1 lists the features of each member of the 8XC196K*x* family.

Device	Pins	OTPROM/ EPROM/ ROM (1)	Register RAM (2)	Code/ Data RAM	I/O Pins	EPA pins	SIO/ SSIO Ports	A/D Channels	External Inte- rrupt
8XC196JV (3)	52	48 K	1536	512	56	6	3	6	1
8XC196KT	68	32 K	1024	512	56	10	3	8	2
8XC196JT (3)	52	32 K	1024	512	41	6	3	6	1
87C196CA (4)	68	32 K	1024	256	51	6	3	6	2
8XC196KS (3)	68	24 K	1024	256	56	10	3	8	2
8XC196KR	68	16 K	512	256	56	10	3	8	2
8XC196JR	52	16 K	512	256	41	6	3	6	1
8XC196KQ	68	12 K	384	128	56	10	3	8	2
8XC196JQ	52	12 K	384	128	41	6	3	6	1

Notes

1. *Optional. The second character of the device name indicates the presence and type of nonvolatile memory. 80C196xx = none; 83C196xx = ROM; 87C196xx = OTPROM or EPROM.*
2. *Register RAM amounts include the 24 bytes allocated to core SFRs and the stack pointer.*
3. *The 8XC196JT, JV, and KS are offered in automotive temperature ranges only. The 87C196CA, 8XC196JQ, JR, KQ, KR, and KT are offered in both automotive and commercial temperature ranges.*
4. *The 87C196CA also has an on-chip networking peripheral that supports CAN specification 2.0.*

BLOCK DIAGRAM

Figure 2.1 shows the major blocks within the device. The core of the device (Figure 2.2) consists of the central processing unit (CPU) and memory controller. The CPU contains the register file and the register

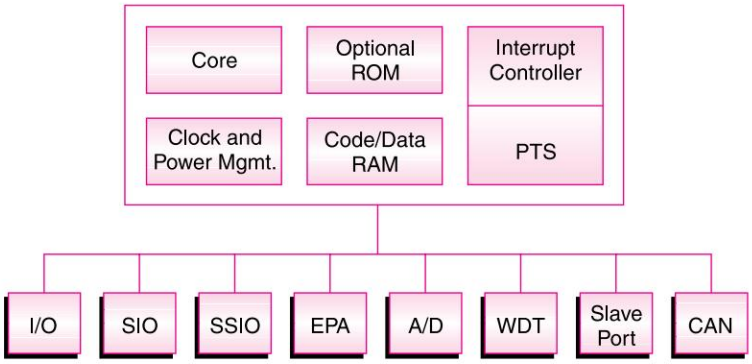


Figure 2.1 *8XC196Kx Block diagram*

arithmetic-logic unit (RALU). The CPU connects to both the memory controller and an interrupt controller via a 16 bit internal bus. An extension of this bus connects the CPU to the internal peripheral modules. In addition, an 8 bit internal bus transfers instruction bytes from the memory controller to the instruction register in the RALU.

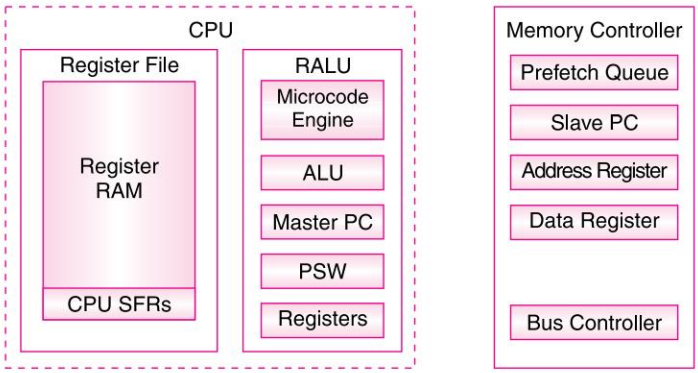


Figure 2.2 *Block diagram of the Core*

CPU CONTROL

The CPU is controlled by the microcode engine, which instructs the RALU to perform operations using bytes, words, or double words from either the 256 byte lower register file or through a *window* that directly accesses the upper register file. CPU instructions move from the 4 byte queue in the memory controller into the RALU's instruction register. The microcode engine decodes the instructions and then generates the sequence of events that cause desired functions to occur.

REGISTER FILE

The register file is divided into an upper and a lower file. In the lower register file, the lowest 24 bytes are allocated to the CPU's special-function registers (SFRs) and the stack pointer, while the remainder is available as general-purpose register RAM. The upper register file contains only general-purpose register RAM. The register RAM can be accessed as bytes, words, or doublewords.

The RALU accesses the upper and lower register files differently. The lower register file is always directly accessible with register-direct addressing. The upper register file is accessible with register-direct addressing only when *windowing* is enabled. Windowing is a technique that maps blocks of the upper register file into a *window* in the lower register file.

REGISTER ARITHMETIC-LOGIC UNIT (RALU)

The RALU contains the microcode engine, the 16 bit arithmetic logic unit (ALU), the master program counter (PC), the program status word (PSW), and several registers. The registers in the RALU are the instruction register, a constants register, a bit-select register, a loop counter, and three temporary registers (the upper-word, lower-word, and second-operand registers).

The PSW contains one bit (PSW.1) that globally enables or disables servicing of all maskable interrupts, one bit (PSW.2) that enables or disables the peripheral transaction server (PTS), and six Boolean flags that reflect the state of your program.

All registers, except the 3 bit bit-select register and the 6 bit loop counter, are either 16 or 17 bit (16 bit plus a sign extension). Some of these registers can reduce the ALU's workload by performing simple operations.

The RALU uses the upper- and lower-word registers together for the 32 bit instructions and as temporary registers for many instructions. These registers have their own shift logic and are used for operations that require logical shifts, including normalize, multiply, and divide operations. The six-bit loop counter counts repetitive shifts. The second-operand register stores the second operand for two-operand instructions, including the multiplier during multiply operations and the divisor during divide operations. During subtraction operations, the output of this register is complemented before it is moved into the ALU. The RALU speeds up calculations by storing constants (e.g., 0, 1, and 2) in the constants register so that they are readily available when complementing, incrementing, or decrementing bytes or words. In addition, the constants register generates single-bit masks, based on the bit-select register, for bit-test instructions.

Code Execution

The RALU performs most calculations for the device, but it does not use an *accumulator*. Instead it operates directly on the lower register file, which essentially provides 256 accumulators. Because data does not flow through a single accumulator, the device's code executes faster and more efficiently.

Instruction Format

MCS 96 microcontrollers combine a large set of general-purpose registers with a three-operand instruction format. This format allows a single instruction to specify two source registers and a separate destination register. For example, the following instruction multiplies two 16 bit variables and stores the 32 bit result in a third variable.

```
MUL RESULT, FACTOR_1, FACTOR_2    ; multiply FACTOR_1 and FACTOR_2
                                  ; and store answer in RESULT
                                  ;(RESULT)¬(FACTOR_1 × FACTOR_2)
```

INTERRUPT SERVICE

The device's flexible interrupt-handling system has two main components: the programmable interrupt controller and the peripheral transaction server (PTS). The programmable interrupt controller has a hardware priority scheme that can be modified by your software. Interrupts that go through the interrupt controller are serviced by interrupt service routines that you provide. The peripheral transaction server (PTS), a microcoded hardware interrupt processor, provides high speed, low-overhead interrupt handling. You can configure most interrupts (except NMI, trap, and unimplemented opcode) to be serviced by the PTS instead of the interrupt controller. The PTS can transfer bytes or words, either individually or in blocks, between any memory locations, manage multiple analog-to-digital (A/D) conversions, and generate pulse-width modulated (PWM) signals. PTS interrupts have a higher priority than standard interrupts and may temporarily suspend interrupt service routines.

INTERNAL PERIPHERALS

The internal peripheral modules provide special functions for a variety of applications.

I/O PORTS

The 8XC196K*x*, 8XC196J*x*, and 87C196CA have seven I/O ports, ports 0–6. Individual port pins are multiplexed to serve as standard I/O or to carry special-function signals associated with an on-chip peripheral or an off-chip component. If a particular special-function signal is not used in an application, the associated pin can be individually configured to serve as a standard I/O pin. Ports 3 and 4 are exceptions. Their pins must be configured either as all I/O or as all address/data. Port 0 is an input-only port that is also the analog input for the A/D converter. Ports 1, 2, and 6 are standard, bi-directional I/O ports. Port 1 provides pins for the EPA and timers. Port 2 provides pins for the serial I/O (SIO) port, interrupts, bus control signals, and clock generator. Port 6 provides pins for the event processor array (EPA) and synchronous serial I/O (SSIO) port. Ports 3, 4, and 5 are memory-mapped, bi-directional I/O ports. Ports 3 and 4 serve as the external address/data bus. Port 5 provides bus control signals; for the 8XC196K*x*, it can also provide pins for the slave port.

SERIAL I/O (SIO) PORT

The serial I/O (SIO) port is an asynchronous/synchronous port that includes a universal asynchronous receiver and transmitter (UART). The UART has one synchronous mode (mode 0) and three asynchronous modes (modes 1, 2, and 3) for both transmission and reception. The asynchronous modes are full duplex, meaning that they can transmit and receive data simultaneously. The receiver is buffered, so the reception of a second byte may begin before the first byte is read. The transmitter is also buffered, allowing continuous transmissions.

SYNCHRONOUS SERIAL I/O (SSIO) PORT

The synchronous serial I/O (SSIO) port provides for simultaneous, bidirectional communications between two 8XC196 family devices or between an 8XC196 device and another synchronous serial I/O device. The SSIO port consists of two identical transceiver channels with a dedicated baud-rate generator. The channels can be programmed to operate in several modes.

SLAVE PORT (8XC196KX ONLY)

The slave port offers an alternative for communication between two CPU devices. Traditionally, system designers have had three alternatives for achieving this communication a serial link, a parallel bus without a dual-port RAM (DPRAM), or a parallel bus with a DPRAM to hold shared data.

A serial link, the most common method, has several advantages: it uses only two pins from each device, it needs no hardware protocol, and it allows for error detection before data is stored. However, it is relatively slow and involves software overhead to differentiate data, addresses, and commands. A parallel bus increases communication speed, but requires more pins and a rather involved hardware and software protocol. Using a DPRAM offers software flexibility between master and slave devices, but the hardware interconnect uses a demultiplexed bus, which requires even more pins than a simple parallel connection does. The DPRAM is also costly, and error detection can be difficult. The SSIO offers a simple means for implementing a serial link. The multiplexed address/data bus can be used to implement a parallel link, with or without a DPRAM. The slave port offers a fourth alternative. The slave port offers the advantages of the traditional methods, without their drawbacks. It brings the DPRAM on-chip. With this configuration, an external processor (master) can simply read from and write to the on-chip memory of the 8XC196

(slave) device. The slave port requires more pins than a serial link does, but fewer than the number used for a parallel bus. It requires no hardware protocol, and it can interface with either a multiplexed or a demultiplexed bus. The master simply reads or writes as if there were a DPRAM device on the bus. Data error detection can be handled through the software.

EVENT PROCESSOR ARRAY (EPA) AND TIMER/COUNTERS

The event processor array (EPA) performs high-speed input and output functions associated with its timer/counters. In the input mode, the EPA monitors an input for signal transitions. When an event occurs, the EPA records the timer value associated with it. This is a *capture* event. In the output mode, the EPA monitors a timer until its value matches that of a stored time value. When a match occurs, the EPA triggers an output event, which can set, clear, or toggle an output pin. This is a *compare* event. Both capture and compare events can initiate interrupts, which can be serviced by either the interrupt controller or the PTS.

Timer 1 and timer 2 are both 16 bit up/down timer/counters that can be clocked internally or externally. Each timer/counter is called a *timer* if it is clocked internally and a *counter* if it is clocked externally.

ANALOG-TO-DIGITAL CONVERTER

The analog-to-digital (A/D) converter converts an analog input voltage to a digital equivalent. Resolution is either 8 or 10 bit; sample and convert times are programmable. Conversions can be performed on the analog ground and reference voltage, and the results can be used to calculate gain and zero-offset errors. The internal zero-offset compensation circuit enables automatic zero offset adjustment. The A/D also has a threshold-detection mode, which can be used to generate an interrupt when a programmable threshold voltage is crossed in either direction. The A/D scan mode of the PTS facilitates automated A/D conversions and result storage. The main components of the A/D converter are a sample-and-hold circuit and an 8 bit or 10 bit *successive* approximation analog-to-digital converter.

WATCHDOG TIMER

The watchdog timer is a 16 bit internal timer that resets the device if the software fails to operate properly.

CAN SERIAL COMMUNICATIONS CONTROLLER (87C196CA ONLY)

The 87C196CA device has a peripheral not found on 8XC196J*x* or 8XC196K*x* devices, the CAN (controller area network) peripheral. The CAN serial communications controller manages communications between multiple network nodes. This integrated peripheral is similar to Intel's standalone 82527 CAN serial communications controller, supporting both the standard and extended message frames specified by the CAN 2.0 protocol parts A and B.

SPECIAL OPERATING MODES

In addition to the normal execution mode, the device operates in several special-purpose modes. Idle and powerdown modes conserve power when the device is inactive. On-circuit emulation (ONCE) mode electrically isolates the microcontroller from the system, and several other modes provide programming options for nonvolatile memory.

REDUCING POWER CONSUMPTION

In idle mode, the CPU stops executing instructions, but the peripheral clocks remain active. Power consumption drops to about 40% of normal execution mode consumption. Either a hardware reset or any enabled interrupt source will bring the device out of idle mode. In powerdown mode, all internal clocks are frozen at logic state zero and the oscillator is shut off. The register file, internal code and data RAM, and most peripherals retain their data if VCC is maintained. Power consumption drops into the µW range.

TESTING THE PRINTED CIRCUIT BOARD

The on-circuit emulation (ONCE) mode electrically isolates the 8XC196 device from the system. By invoking ONCE mode, you can test the printed circuit board while the device is soldered onto the board.

PROGRAMMING THE NONVOLATILE MEMORY

MCS 96 microcontrollers that have internal OTPROM or EPROM provide several programming options: • Slave programming allows a master EPROM programmer to program and verify one or more slave MCS 96 microcontrollers. Programming vendors and Intel distributors typically use this mode to program a large number of microcontrollers with a customer's code and data. • Auto programming allows an MCS 96 microcontroller to program itself with code and data located in an external memory device. Customers typically use this low-cost method to program a small number of microcontrollers after development and testing are complete.• Serial port programming allows you to download code and data (usually from a personal computer or workstation) to an MCS 96 microcontroller asynchronously through the serial I/O port's RXD and TXD pins. Customers typically use this mode to download large sections of code to the microcontroller during software development and testing. • Run-time programming allows you to program individual nonvolatile memory locations during normal code execution, under complete software control. Customers typically use this mode to download a small amount of information to the microcontroller after the rest of the array has been programmed. For example, you might use run-time programming to download a unique identification number to a security device. • ROM dump mode allows you to dump the contents of the device's nonvolatile memory to a tester or to a memory device (such as flash memory or RAM).

CAN SERIAL COMMUNICATIONS CONTROLLER

The 87C196CA has a peripheral not found in the 8XC196K*x* and 8XC196J*x* controllers the CAN (controller area network) peripheral. The CAN serial communications controller manages communications between multiple network nodes. This integrated peripheral is similar to Intel's standalone 82527 CAN serial communications controller. It supports both the standard and the extended message frames specified by CAN 2.0 protocol parts A and B developed by Robert Bosch, GmbH.

CAN FUNCTIONAL OVERVIEW

The integrated CAN controller transfers messages between network nodes according to the CAN protocol. The CAN protocol uses a multiple-master, contention-based bus configuration, which is also called CSMA/CR (carrier sense, multiple access, with collision resolution). Each CAN controller's input and output pins are connected to a two-line CAN bus through which all communication takes place.

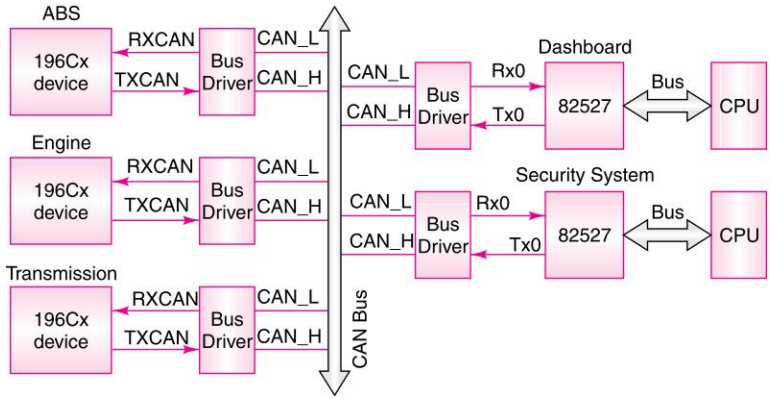
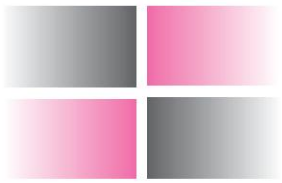


Figure 3 *A system using CAN controller*

The McGraw-Hill Companies



Appendix F

PIC16F87X

28/40-PIN 8-BIT CMOS FLASH MICROCONTROLLERS

DEVICES INCLUDED IN THIS DATA SHEET

- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

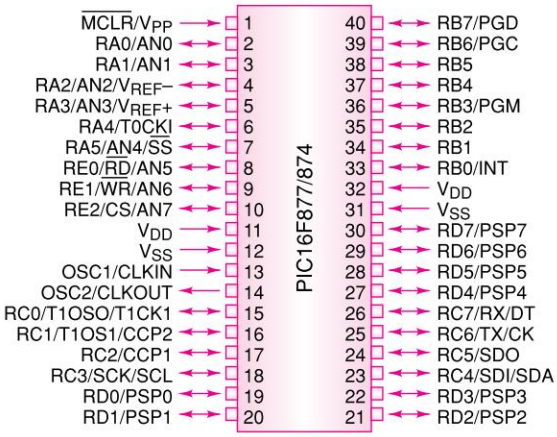
MICROCONTROLLER CORE FEATURES

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K 14 words of FLASH Program Memory,
  Up to 368 8 bytes of Data Memory (RAM)
  Up to 256 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology

- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5 V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0 V to 5.5 V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3 V, 4 MHz
  - 20 µA typical @3 V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram

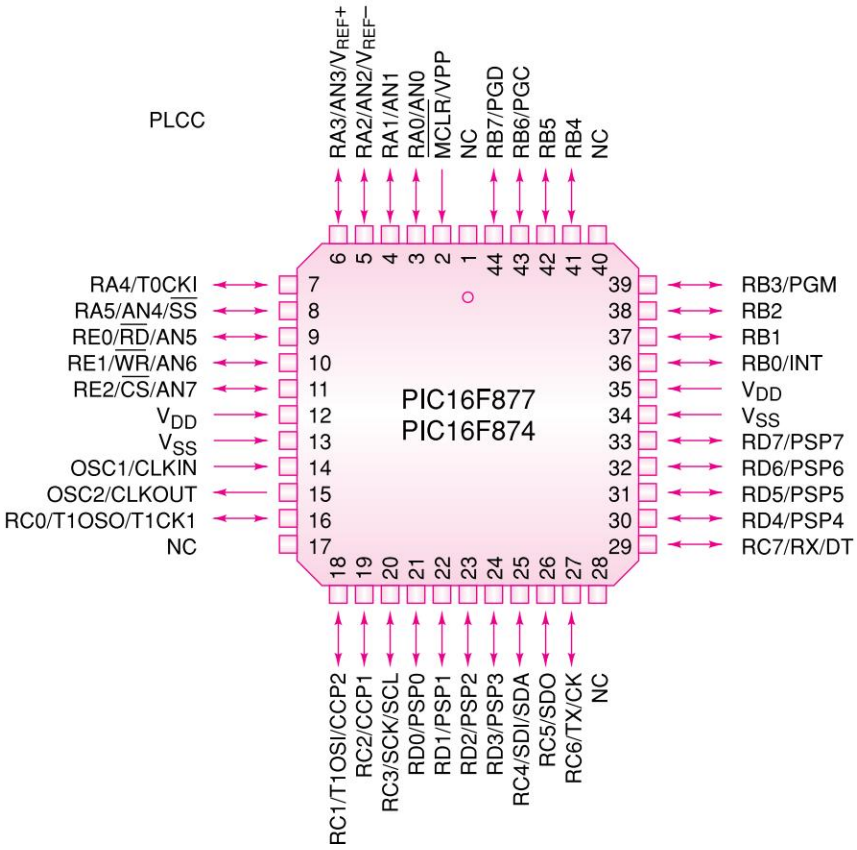
PDIP



PERIPHERAL FEATURES

- Timer 0: 8 bit timer/counter with 8 bit prescaler
- Timer 1: 16 bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer 2: 8 bit timer/counter with 8 bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16 bit, max. resolution is 12.5 ns
  - Compare is 16 bit, max. resolution is 200 ns
  - PWM max. resolution is 10 bit
- 10 bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I²C™ (Master/Slave)

- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9 bit address detection
- Parallel Slave Port (PSP) 8 bit wide, with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)



Pin diagram of PIC 16F877/16F874

DEVICE OVERVIEW

There are four devices (PIC16F873, PIC16F874, PIC16F876 and PIC16F877) covered by this data sheet. The PIC16F876/873 devices come in 28-pin packages and the PIC16F877/874 devices come in 40-pin packages. The Parallel Slave Port is not implemented on the 28-pin devices.

MEMORY ORGANISATION

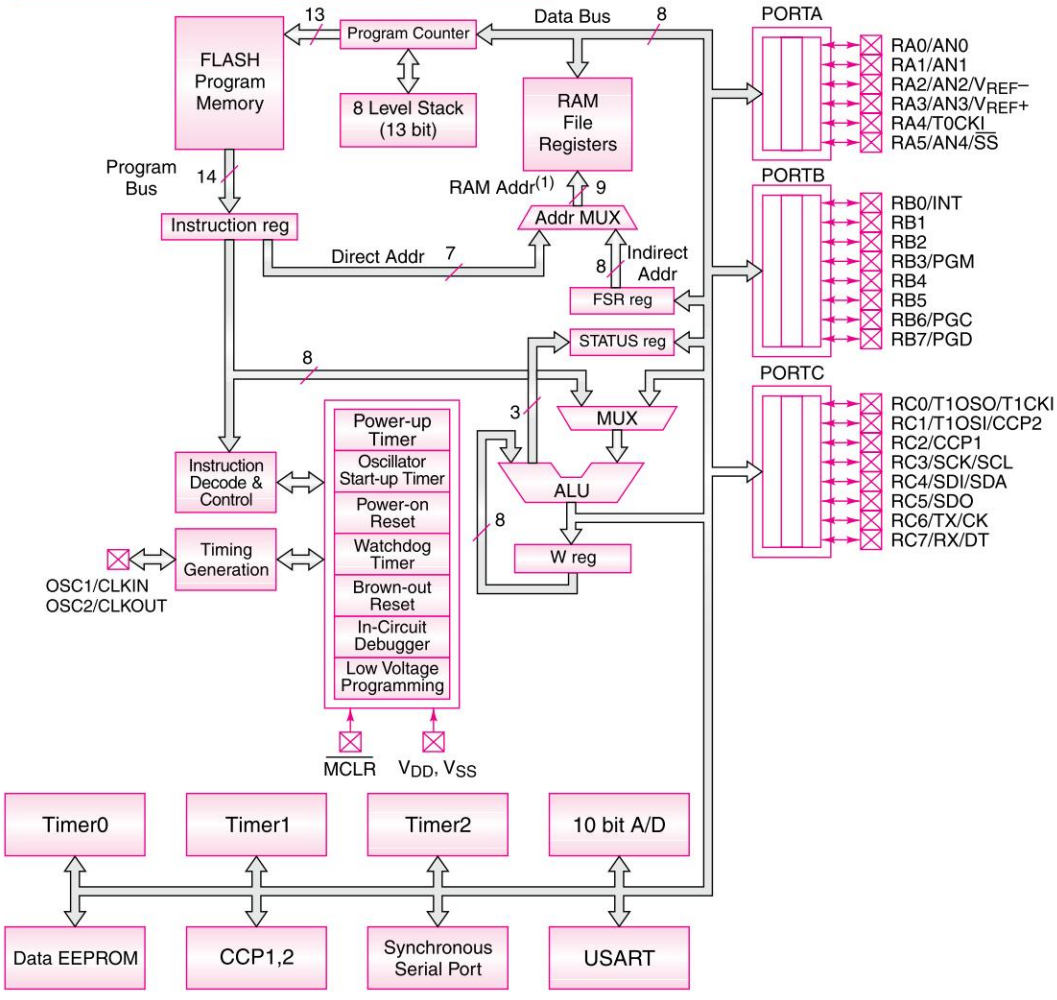
There are three memory blocks in each of the PIC16F87X MCUs. The Program Memory and Data Memory have separate buses so that concurrent access can occur and is detailed in this section.

Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

DATA MEMORY ORGANISATION

The data memory is partitioned into multiple blanks which contain the General Purpose Registers and the Special Function Registers. Bits RP1 (STATUS<6>) and RP0 (STATUS<5>) are the bank select bits.

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F873	4K	192 Bytes	128 Bytes
PIC16F876	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

Figure 1.1 *PIC16F873 and PIC16F876 block diagram*

RP1:RP0	Bank
00	0
01	1
10	2
11	3

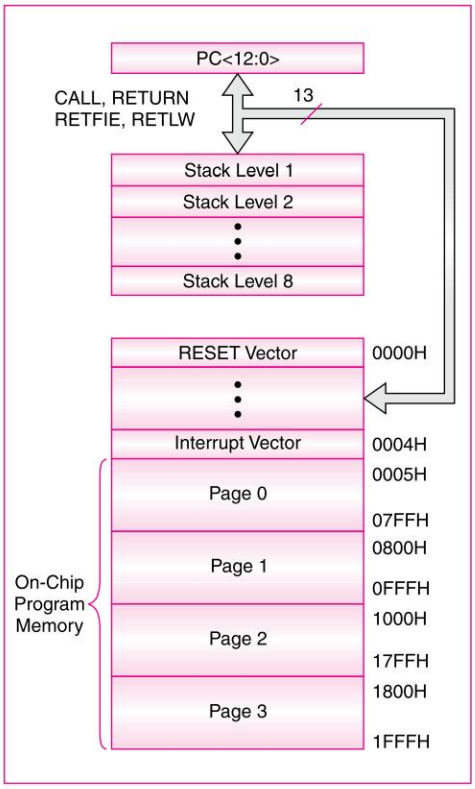


Figure 2.1 *PIC16F877/876 Program memory map and stack*

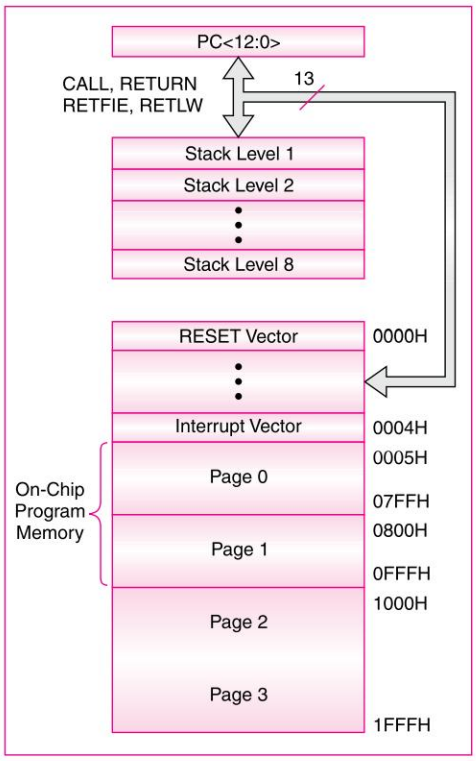


Figure 2.2 *PIC16F874/873 Program memory map and stack*

PROGRAM MEMORY ORGANISATION

The PIC16F87X devices have a 13 bit program counter capable of addressing an 8K 14 program memory space. The PIC18F877/876 devices have 8K 14 words of FLASH program memory, and the PIC16F873/874 devices have 4K 14. Accessing a location above the physically implemented address will cause a wraparound.

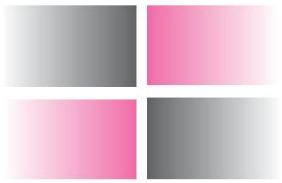
The RESET vector is at 0000H and the interrupt vector is at 0004H.

Each bank extends up to 7FH (128 Bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static Ram. All implemented banks contain Special Function Registers. Some frequently used Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

BIBLIOGRAPHY

1. Ramesh S. Goankar, 'Microprocessor Architecture Programming and Applications with the 8085', 3rd edition, Penram International, 1997.
2. David Calcutt, Fred Cowan, Hassan Parchizadeh, '8051 Microcontrollers: an Application Based Introduction', 1st edition, Elsevier, 2006
3. Han-Way Huang, 'Using the MCS-51 Microcontroller', Oxford University Press, 2000.
4. Kenneth J. Ayala, 'The 8051 Microcontroller: Architecture, Programming and Applications', 2nd edition, Thomson Delmar Learning, 1997.
5. Muhammed Ali Mazidi, Janice Gillispie Mazidi, 'The 8051 Microcontroller and Embedded Systems', 7th edition, Pearson Education, 2004.
6. Data Sheets, P89C60X2/61X2, 80C51 8-bit Flash microcontroller family, Philips, 2003.
7. Data sheets, SN54/74LS373/374, Octal Transparent Latch with 3-State Outputs; Octal D-Type Flip-Flop with 3-State Output, Motorola.
8. Data sheets, DAC0808 8-Bit D/A Converter, ADC0808/ADC0809 8-Bit µP Compatible A/D Converters with 8-Channel Multiplexer, National Semiconductors, 1999.
9. Data Sheets, PIC16F87X 28/40-Pin 8-Bit CMOS FLASH Microcontrollers, Microchip Technology Inc., 2001.
10. MCS-51 Microcontroller Family User's Manual, Intel, 1994.

The McGraw·Hill Companies



INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V