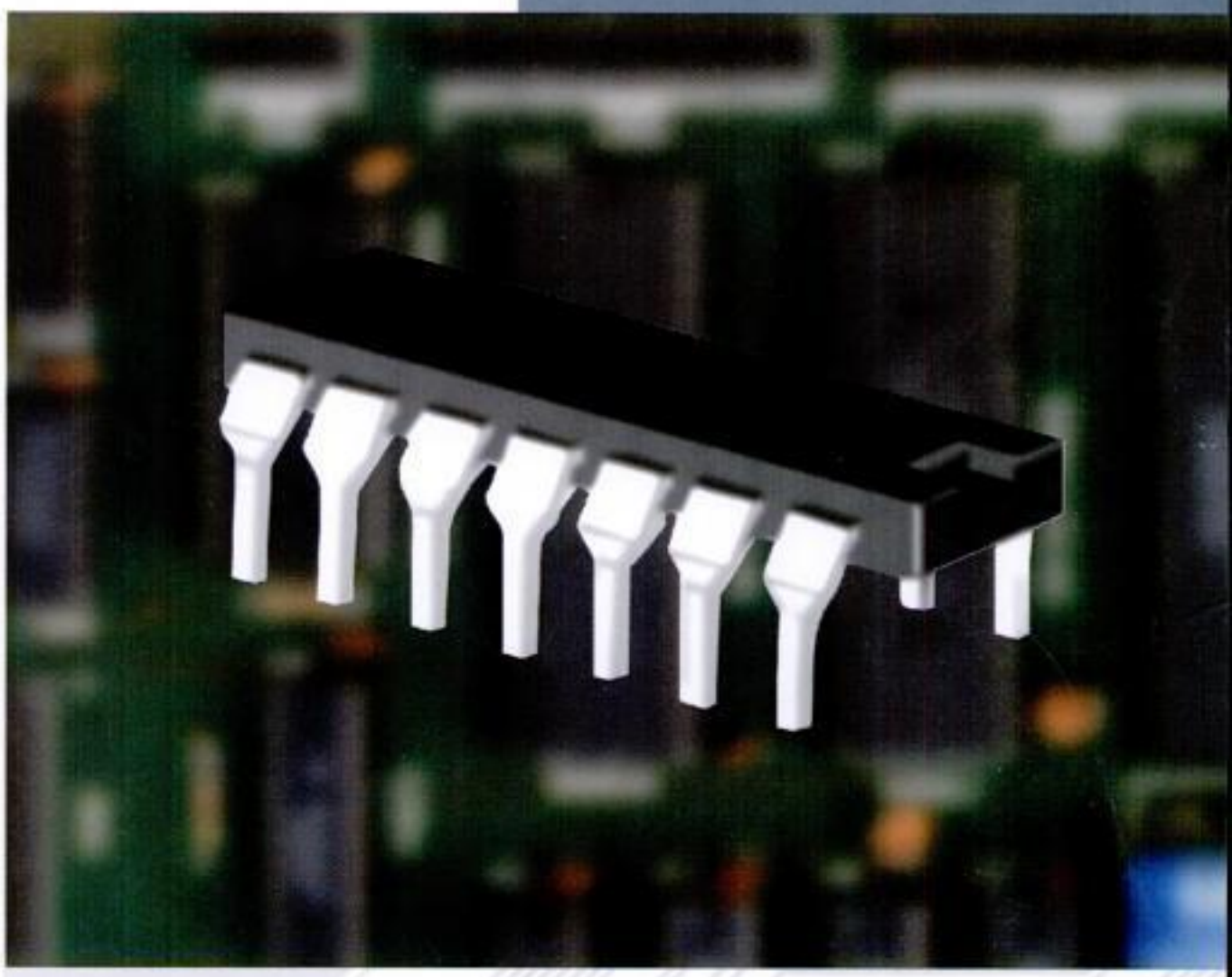
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The 8051 Microcontrollers

Architecture, Programming and Applications



K. Uma Rao
Andhe Pallavi

ALWAYS LEARNING

PEARSON

THE 8051 MICROCONTROLLER

Architecture, Programming & Applications

Dr. K. Uma Rao

Dean, Faculty of Engineering,
Professor and Head, Department of Electronics and Communication Engineering,
RN Shetty Institute of Technology,
Channasandra, Bangalore

Dr. Andhe Pallavi

Professor and Head, Department of Instrumentation Technology,
RN Shetty Institute of Technology,
Channasandra, Bangalore

PEARSON

ISBN: 978-81-317-3252-6

First Impression, 2010
Second Impression, 2011

Published by Dorling Kindersley (India) Pvt. Ltd, licencees of Pearson Education in South Asia.

Head Office: 7th Floor, Knowledge Boulevard, A-8 (A), Sector-62, Noida 201309, UP, India.

Registered Office: 11 Community Centre, Panchsheel Park, New Delhi 110 017, India.

Printed in India by Chennai Micro Print.

Contents

1 Computers, Microprocessors and Microcontrollers—An Introduction **1**

1.1 Introduction 2

1.2 Common Terminology Associated with Computing Systems 3

1.2.1 Hardware 3

1.2.2 Software 3

1.2.3 Firmware 3

1.2.4 Binary Digits 3

1.2.5 Memory 4

1.2.6 Input/Output or I/O 4

1.2.7 Central Processing Unit 4

1.2.8 Bus 4

1.2.9 Address Bus 4

1.2.10 Data Bus 5

1.2.11 Control Bus 5

1.2.12 Ports 5

1.2.13 Register Section 5

1.2.14 ASCII 6

1.2.15 Operating System 6

1.2.16 Time Sharing 6

1.2.17 Multi-Tasking 7

1.3 Microprocessors and Microcontrollers 7

1.4 CISC and RISC Systems 10
1.5 Computing Languages 11
1.6 Memory 12
1.6.1 Random Access Memory (RAM) 12
1.6.2 Read Only Memory (ROM) 14
1.6.3 Cache Memory 15
1.6.4 Memory Latency 15
1.7 Computer Architecture: Harvard and Von Neumann 16
1.8 Evolution of Microcontrollers—4-bit to 32-bit 18
1.8.1 Selection of a Microcontroller 18
1.8.2 4-bit Microcontrollers 19
1.8.3 8-bit Microcontrollers 19
1.9 Summary 20
1.10 Questions 20

2 Data Representation 23
2.1 Introduction 24
2.2 Number System 24
2.2.1 Binary System 24
2.2.2 Octal System 25
2.2.3 Hexadecimal System 25
2.2.4 Conversion from Decimal to Radix r 26
2.3 Decimal Representation 28
2.4 Complements 29
2.4.1 $(r-1)$'s Complement 29
2.4.2 r's Complement 30
2.4.3 Subtraction of Unsigned Numbers using r's Complement 31
2.5 Fixed-Point Representation 32
2.5.1 Signed Integer Representation 33
2.5.2 Arithmetic Addition of Signed Numbers 34

2.5.3 Arithmetic subtraction of signed numbers 35
2.5.4 Decimal Fixed-Point Representation 36
2.6 Floating-Point Representation 38
2.7 Other Binary Codes 38
2.7.1 Gray Code 39
2.7.2 Other Decimal Codes 39
2.7.3 Alphanumeric Codes 40
2.8 Summary 44
2.9 Questions 44

3 8051 Architecture 47
3.1 Introduction 48
3.2 Block Diagram of 8051 48
3.3 Pin Diagram of 8051 51
3.4 Clock and Machine Cycle for 8051 51
3.5 Registers of 8051 53
3.5.1 Program Counter (PC) 54
3.5.2 Data Pointer (DPTR) 54
3.5.3 A and B Registers 54
3.5.4 Program Status Word (PSW) Register 54
3.5.5 Special Function Registers (SFR's) 57
3.6 The 8051 Internal Memory 58
3.6.1 Internal RAM 58
3.6.2 Internal ROM 60
3.6.3 External Memory 60
3.6.4 Memory Address Decoding 62
3.7 Stack and Stack Pointer 69
3.8 Timers and Counters 69
3.8.1 Basic Timer Registers 70
3.8.2 Timer Operation 70
3.8.3 Timer SFR's 71

3.8.4 TMOD Register 71
3.8.5 Timer Modes 75
3.8.6 TCON Register 77
3.9 I/O Ports 78
3.9.1 Port 0 (P0: Address 80H) 78
3.9.2 Port 1 (P1: Address 90H) 79
3.9.3 Port 2 (P2: Address A0H) 79
3.9.4 Port 3 80
3.9.5 Writing to a Port 81
3.9.6 Reading a Port 81
3.10 Serial Input/Output 82
3.10.1 Basic Concepts in Serial I/O (Synchronous vs. Asynchronous Transmission) 82
3.10.2 Serial Port Control (SCON) Register 84
3.10.3 Power Mode Control (PCON) SFR 85
3.10.4 SBUF Register 87
3.10.5 Data Transmission and Reception 87
3.10.6 Serial Data Transmission Modes 87
3.11 Interrupts 91
3.11.1 8051 Interrupts 91
3.11.2 Interrupt Destination 92
3.11.3 Interrupt Enable (IE) SFR 93
3.11.4 Polling Sequence 93
3.11.5 Interrupt Priority (IP) SFR 94
3.12 Supply Voltage 95
3.13 Status of SFR's on Reset 95
3.14 Machine Cycles 96
3.15 Detailed Pin Description 98
3.16 Summary 99
3.17 Questions 99

4 Assembly Language Programming I—Addressing Modes and Data Transfer **103**

4.1 Introduction 104

4.2 Assembly Language 104

4.2.1 Structure of Assembly Language 105

4.2.2 Steps to Create an ALP 106

4.2.3 Program Code in ROM 107

4.2.4 Execution of the Program by 8051 108

4.3 Flow Charts and Algorithm 108

4.4 8051 Data Types and Directives 110

4.4.1 Directives 110

4.5 Addressing Modes 112

4.5.1 Immediate Addressing Mode 113

4.5.2 Register Addressing Mode 114

4.5.3 Direct Addressing Mode 115

4.5.4 Indirect Addressing Mode 118

4.5.5 Indexed Addressing Mode for ROM Access 120

4.6 Data Transfer with Stack 121

4.6.1 PUSH Instruction 121

4.6.2 POP Instruction 122

4.7 Data Exchange 123

4.8 Complete set of Data Transfer Instructions 126

4.9 Summary 127

4.10 Questions 128

5 Assembly Language Programming II – Arithmetic and Logic Operator **131**

5.1 Introduction 132

5.2 Addition 132

5.2.1 Addition of Unsigned Numbers 132

5.2.2 Addition of 16-bit Numbers 135
5.2.3 Signed Addition 136
5.3 Incrementing and Decrementing 137
5.4 Subtraction 138
5.5 Multiplication 139
5.6 Division 140
5.7 Decimal Addition 141
5.8 Summary of Arithmetic Operations of 8051 148
5.9 Logical Operations: Byte Level 149
5.9.1 AND Operation 149
5.9.2 OR Operation 150
5.9.3 Exclusive OR Operation 150
5.10 CLEAR and COMPLEMENT Accumulator 154
5.10.1 Clear Accumulator 154
5.10.2 Complement Accumulator 154
5.11 Bit Level Logical Operations 155
5.11.1 Bit Level AND 155
5.11.2 Bit Level OR 155
5.11.3 Complement Bits 156
5.11.4 Clear Bits 157
5.11.5 Data Transfer into Bits 157
5.11.6 Set Addressed Bits 157
5.12 Rotate Operation 158
5.12.1 Rotate Accumulator Right 159
5.12.2 Rotate Through the Carry 160
5.12.3 Serialize Data 161
5.13 Swap Operation 162
5.14 Summary of Bit-Level Logical Operations 164
5.15 Summary 165
5.16 Questions 165

6 Assembly Language III—Jump and Call Instructions **167**

6.1 Introduction 168

6.2 Address Range of Jump and Call Instruction 168

6.2.1 Relative Range 168

6.2.2 Short Absolute Range 169

6.2.3 Long Absolute Range 170

6.3 Jump Instructions 170

6.3.1 Unconditional Jump 170

6.3.2 Bit Jump Instructions 171

6.3.3 Byte Jump Instructions 173

6.4 CALL Instruction 178

6.4.1 Subroutine 178

6.4.2 Sequence of Events in CALL Execution 179

6.5 Summary 181

6.6 Questions 181

7 Programming 8051 with C **183**

7.1 Introduction 184

7.1.1 Advantages of Programming in 'C' for Microcontrollers 184

7.1.2 Disadvantages of Programming in 'C' 184

7.2 Declaring Variables 184

7.2.1 Data Types in 8051 'C' 185

7.2.2 Arrays and Strings 185

7.2.3 Number Representation 187

7.3 Writing a Simple C Program 187

7.4 Delay Generation in C 193

7.5 Programming Ports of 8051 with C 198

7.6 Operators in 8051C 207

7.7 Serial Port Programming 213

7.8 Code Conversions in C 218

7.9 Code Space 226
7.10 Summary 227
7.11 Questions 228

8 Timers/Counters and Serial Port in 8051 231
8.1 Introduction 232
8.2 Time Delay Generation using Timers 232
8.2.1 Procedure for Time Delay Generation 232
8.2.2 Generation of Square Wave 236
8.2.3 Timer mode 0 Programming 238
8.2.4 Large Time Delays 243
8.3 Application of Timers in mode 2 246
8.4 Counter Application 251
8.5 Serial Data Transfer 264
8.5.1 8051 Connection to RS232 264
8.5.2 Special Function Registers of Serial Port 265
8.5.3 Review of Working of Serial Port 265
8.5.4 Procedure to Program the 8051 to Transfer Data Serially 266
8.5.5 Procedure for Programming the 8051 to Receive Data Serially 276
8.5.6 Procedure for Doubling the Baud Rate in the 8051 277
8.6 Second Serial Port in 8051 281
8.6.1 Features of Second Serial Port 282
8.7 Summary 289
8.8 Questions 290

9 Interrupts 293
9.1 Introduction 294
9.2 Review of Interrupts in 8051 294
9.2.1 Execution of an Interrupt by 8051 (On the Recept of Interrupt Request) 296

9.3 External Interrupts 310
9.3.1 Level Triggered Interrupt Mode 310
9.3.2 Edge Triggered Interrupts Mode 313
9.4 Serial Communication Interrupt 316
9.5 Priority Implementation for 8051 Interrupts 322
9.5.1 Changing Interrupt Priority 324
9.6 Summary 332
9.7 Questions 333

10 Interfacing the 8051 335
10.1 Introduction 336
10.2 Interfacing a LED and a 7-Segment Display to an 8051 336
10.2.1 Multiplexed 7-Segment Display 339
10.2.2 LCD Display 341
10.3 Interfacing a Single Key (Push Button) to the 8051 355
10.4 MATRIX KEYPAD or Interfacing Keyboard to the 8051 360
10.5 Stepper Motor Interfacing to 8051 365
10.6 Interfacing a DAC to an 8051 375
10.7 DC Motor Interfacing to 8051 385
10.7.1 Features of a DC Motor 385
10.7.2 Interfacing DC Motors Using Opto Isolators 390
10.7.3 PWM Control of DC Motor 391
10.8 ADC-Analog-To-Digital Converters 393
10.8.1 Parameters of ADC 394
10.8.2 Interfacing ADC 0804 to 8051 395
10.8.3 ADC 0808/0809 397
10.8.4 ADC 0848 399
10.8.5 Serial ADCs 401
10.9 Summary 406

11 Simulation of 8051 using Keil Software (Lab Manual) 407

11.1 Introduction 408

11.2 Features of the 8051 Version Used 408

11.2.1 Processor Features 408

11.3 Creating and Compiling a μVision2 Project 409

11.4 Programming in ALP 411

11.10 DAC (Digital to Analog Converter) Interfacing to 8051 438

Appendix A 457

Appendix B 461

References 469

Index 471

Chapter 1

Computers, Microprocessors and Microcontrollers—An Introduction

Living in a world heavily computer oriented, we are bombarded with a number of words and terms related to computers. As technology has invaded right into our homes, it is very difficult to find a product which is available without some digital control in it. Before we learn the basics of microcontrollers, we will familiarize ourselves with some of the terms and try to grasp an overview of computers and computing systems.

Learning Objectives

At the conclusion of this chapter, you will be able to:

- Define terms such as microcomputer, microprocessor, microcontroller, hardware, software, firmware, timesharing, multitasking, distributed processing and multiprocessing.
- Explain the difference between higher level languages and assembly language of a computer.
- Define ASCII code and explain its relationship to binary code and alphanumeric characters.
- Explain memory organization and memory map.
- Explain how memory address is assigned to a memory chip.
- List the different types of memory and their functions.
- Understand and describe how a microprocessor/microcontroller fetches and executes an instruction.

1.1 Introduction

The microprocessor can be viewed as a logic device which can be programmed, to enable it to be used to control processes or as a data processing unit or as the computing unit of a computer. To understand how the microprocessor came into existence, we must understand a brief history of the growth of two major technologies: digital computers and solid-state circuits. These two areas integrated to give birth to microprocessors and microcontrollers, subsequently.

The *digital computer* is a set of digital circuits controlled by a program that makes it do the job we want to be done. The program tells the computer how to process the data using arithmetic and logic operations, memory circuits and input/output devices. The way the arithmetic and logic circuits, memory circuits and input/output devices are put together, so that they function as one unit, is called the *architecture.*

The computer can be represented as shown in the block diagram of Fig. 1.1.

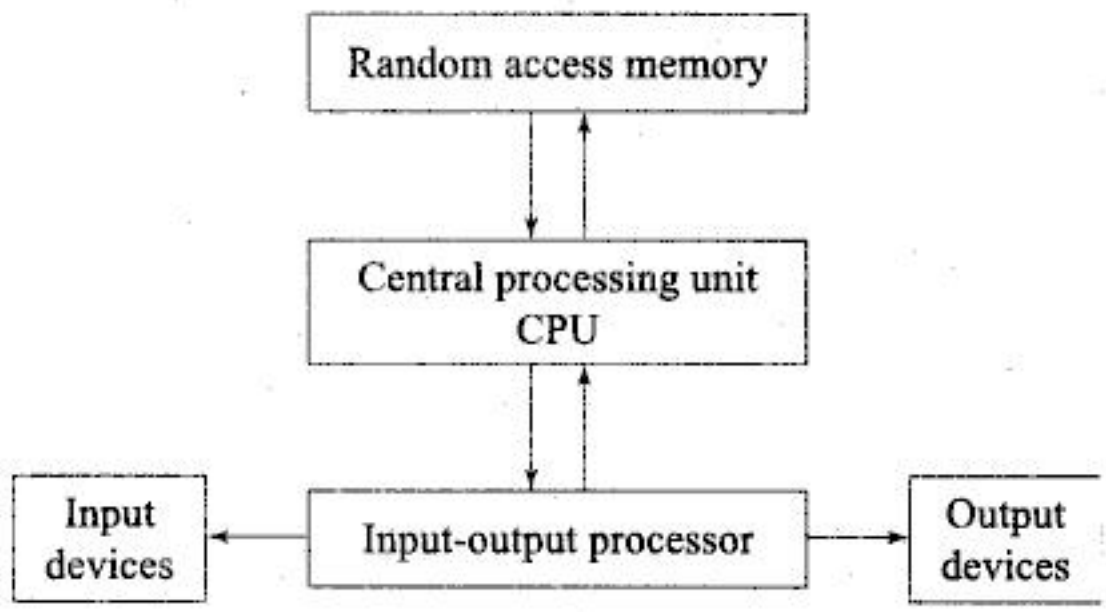


Figure 1.1. Block diagram of a digital computer.

Any digital computer has the following major components:

- The *Central Processing Unit* (CPU) which consists of the *Arithmetic and Logic Unit* (ALU) which is responsible for performing all arithmetic and logic operations and the *control unit* which controls various operations of the computer. The CPU is responsible for performing the specific task by communicating with the memory and input/output devices.
- The *memory unit* which is dealt with in detail later.
- The *input devices* like keyboard, mouse, joystick etc through which the user can enter data into the computer.
- The *output devices* like monitors, printers, CDs etc where the results can be displayed or stored.

The largest and most powerful computers are called the *mainframes*. They are designed to work at very high speeds with large data words and have massive amounts of memory. They are generally used for very large data bases like military, large business enterprises, creating graphics for movies etc. Examples are Cray Y-MP/832, IBM 4381 and Honeywell DPS8. The fastest and most powerful mainframes are called *supercomputers*.

A scaled down version of the mainframe is the *minicomputer*. They run more slowly, work with smaller data and do not have as much memory capacity as a mainframe. They are used in business data processing, process control, etc. Examples are DEC VAX 6360, DGMV/8000II etc.

Microcomputers are small computers which have a microprocessor as its CPU. The *microprocessor* is a programmable, synchronously operating (Clock-driven) register based device that reads binary instructions from memory, accepts binary data, processes it and provides the output. They are extensively used today in a wide range of applications from automobile control, toys, washing machines to computer-aided design systems. A microcomputer which has the microprocessor, memory and I/O device on a single chip becomes the *microcontroller*.

1.2 Common terminology associated with computing systems

1.2.1 Hardware

It refers to the physical components and circuits of the system.

1.2.2 Software

Software refers to the programs written in the form of commands/instructions, either to perform a task or to operate the computer.

1.2.3 Firmware

It refers to the programs stored permanently in the ROM or other devices, written for a specific application.

1.2.4 Binary digits

All computing systems operate in binary digits, 0 and 1 called bits. In the physical system, a bit refers to a voltage level. In logic systems which use positive logic a lower voltage represents 0 and a higher voltage represents a 1.

They are also called low and high respectively. Negative logic, uses the reverse wherein a lower voltage represents logic 1 and a higher voltage represents a logic 0. Microprocessors and microcontrollers process a group of bits called the *word*. For example an 8-bit processor, operates on data of 8 bits and so on. A word of 4-bit length is called a *nibble* and a word of 8 bit length is called a *byte*.

1.2.5 Memory

It is the storage element in the computing system. The first purpose is to store the codes for the sequence of instructions which the computer has to perform. The second purpose it to store the binary-coded data upon which the computer operates. Typically, the memory unit is made of 8-bit registers arranged in a sequence. These are arranged in groups of powers of 2. A semiconductor memory with $2^{10} = 1024$ registers is known as 1 K memory chip (in computer technology, 1 K refers to 1024 and not 1000). Similarly we have standard 4 K, 8 K etc. memory chips. Each register in the memory is identified by a unique address.

1.2.6 Input/Output or I/O

The I/O section allows the computer to take in data from the outside world or send data to the outside world. The peripherals such as keyboards, video terminals, printers and modems are connected to the I/O section. They allow the user to communicate with the computer.

1.2.7 Central Processing Unit

This controls the operation of the computer. In a microcomputer the CPU is a microprocessor. The CPU fetches the instruction to be carried out from the memory, decodes the instruction into a series of operations and then executes them.

1.2.8 Bus

The bus is a group of lines used to transfer bits between various components of the computer. For example, between CPU and memory; CPU and output device etc.

1.2.9 Address Bus

The address bus consists of 16, 20, 24 or 32 parallel signal lines. The CPU sends the address of the memory location that is to be accessed (written to or

read from) on these lines. The number of memory locations that the CPU can address is determined by the number of address lines. If it has N address lines it can address 2^N distinct memory locations.

1.2.10 Data Bus

The data bus consists of 8, 16 or 32 parallel lines. The data lines are bidirectional, meaning that data can flow in both directions, between the computer and memory.

1.2.11 Control Bus

The control bus consists of signal lines, over which the CPU sends signals to control the various operations of the computer, its communication with the outside world and peripheral devices. As an example, we can consider the memory read operation. The CPU sends the memory read signal over the control bus to the memory and enables the addressed memory to output the data word onto the data bus. Similarly other common control signals are memory write, I/O read, I/O write etc.

1.2.12 Ports

The actual physical device used to interface the computer buses to external systems are called ports. An input port allows data from a keyboard, A/D converter or some other source to be read into the CPU. Similarly, the output port is used to send the data to a peripheral device such as a video display, D/A converter or a printer. Physically, the ports are nothing but latches (like D flip-flops). If they are used as an *input port*, the D-inputs are connected to the external device and the output is connected to the data bus. Data will then be transferred through these latches when the control unit sends the control signal. If they are used as an *output port*, the D inputs of the latches are connected to the data bus and the Q outputs are connected to some external device. Data is then transferred to the external device when the latches are enabled by a control signal.

1.2.13 Register section

The register section in a microprocessor or a microcontroller consists of a set of registers, which are on the chip and are used to store data temporarily during execution of a program. They are accessible to the user through the

instructions. Since they are on chip, the CPU can access these faster than the external memory.

1.2.14 ASCII

In communication, we require standard formats for exchange of information. This is one such standard widely used. ASCII stands for American Standard Code for Information Interchange. This is a 7-bit alphanumeric binary code with 128 combinations ($2^7 = 128$). Each combination is assigned to a letter, decimal digit, a symbol or a machine command. The later versions have expanded this code to an 8-bit code with 256 combinations.

1.2.15 Operating System

Operating system is a set of programs that manages the interaction between hardware and software. Examples are MS-DOS, MS-Windows, UNIX, LINUX etc.

1.2.16 Time Sharing

A common method for providing computer access is the method of time-sharing. Here, several terminals (output devices) are connected to the computer. The terminal can be remotely located and connected to the main computer through direct wires or telephone wires. Normally the rate at which the user enters and interprets data is much slower than the rate at which the computers process data. Hence, the computer can serve many users by dividing its time amongst them. This is called *time-sharing* and allows several users to interact with the computer at the same time. Each user can get information or store information in the memory connected to the main computer. A typical example is an airline/railways reservation system which allows users, spread across wide geographical areas, to access information and make reservations. Similarly, in a factory, such a time-shared system will permit the computer to control a number of machines.

In distributed processing or multiprocessing, the terminals of the time-sharing systems are replaced by microcomputers, so that each user can do tasks locally without having to use the main computer at all. However, the connectivity to the main computer through a network, permits the user to access the computing power, memory and other resources of the main computer. The advantage of this system, is that in the event of the main computer failing, the microcomputers can continue working locally. Further, it also relieves the main computer from many tasks which can be performed by the microcomputers.

1.2.17 Multi-tasking

Multi-tasking refers to the ability of the computer to do a number of tasks when it controls machines or processes which are much slower than it. In such a case it can check and adjust a number of parameters, and then get back to the first one and repeat the process. Such a system is called a multi-tasking system, since it appears as if the computer is performing all the tasks at once.

1.3 Microprocessors and Microcontrollers

The basic idea behind both of them is the same. However, they have a number of differences which determines the choice of one over the other. Microcontroller was a by product of the microprocessor. The microprocessor is a multipurpose, programmable, device that can process data. The first microprocessor was manufactured by INTEL Corporation in 1971. The microprocessor is not complete by itself and has very limited memory on chip. Hence, for it to be of use, it must be interfaced with memory, I/O devices and other peripherals. It is programmable, which means that it can be instructed to perform given tasks within its capability. These tasks are communicated to the processor by means of instructions which are stored in the memory sequentially. The microprocessor starts from the first instruction to be executed. It fetches it from memory, then decodes it and executes the instruction. It continues doing so, until it comes across an instruction to stop. During this operation the microprocessor uses the system buses to fetch the instruction and data from the memory. It uses the register section to store data temporarily and then performs the necessary computation using the ALU section. It finally sends the data to the output device through the system bus. The main idea behind a microprocessor is that it is a *general purpose device.* It is very flexible. The length of the data bits which majority of the instructions operate upon determines the size of the processor. Therefore we have 4-bit, 8-bit, 16-bit, 32-bit etc. processors. Starting from the 4-bit 4004 processor in 1971, Intel Corporation has come up with a number of microprocessors which include 8086 (1978), 80486 (1989) and the more recent Pentium (2003). Motorola is another leading manufacturer. Its common processors are 6800 (1974), 68030 (1987), 68040 (1989) and PowerPC604 (1994). Other popular processors are Athlon (from AMD), MBL8086 (Fujitsu), SRP1030 (Sun Microsystems), TC85R4000 (Tisguba), TMS390 (Texas Instruments) and Alpha (DEC).

Thus, in short the microprocessor is a *general purpose device* which reads data, performs extensive calculations on it, and stores the results in a mass storage device or it displays it on a video screen, to be viewed by the user.

The *microcontroller* on the other hand can be viewed as a computer on a single chip. This essentially means that the CPU, RAM, ROM and may be other devices like Timers, etc, are all located on the same chip and for many applications, no interface with other devices is needed. The contrast between the microprocessor and microcontroller is shown in Fig. 1.2.

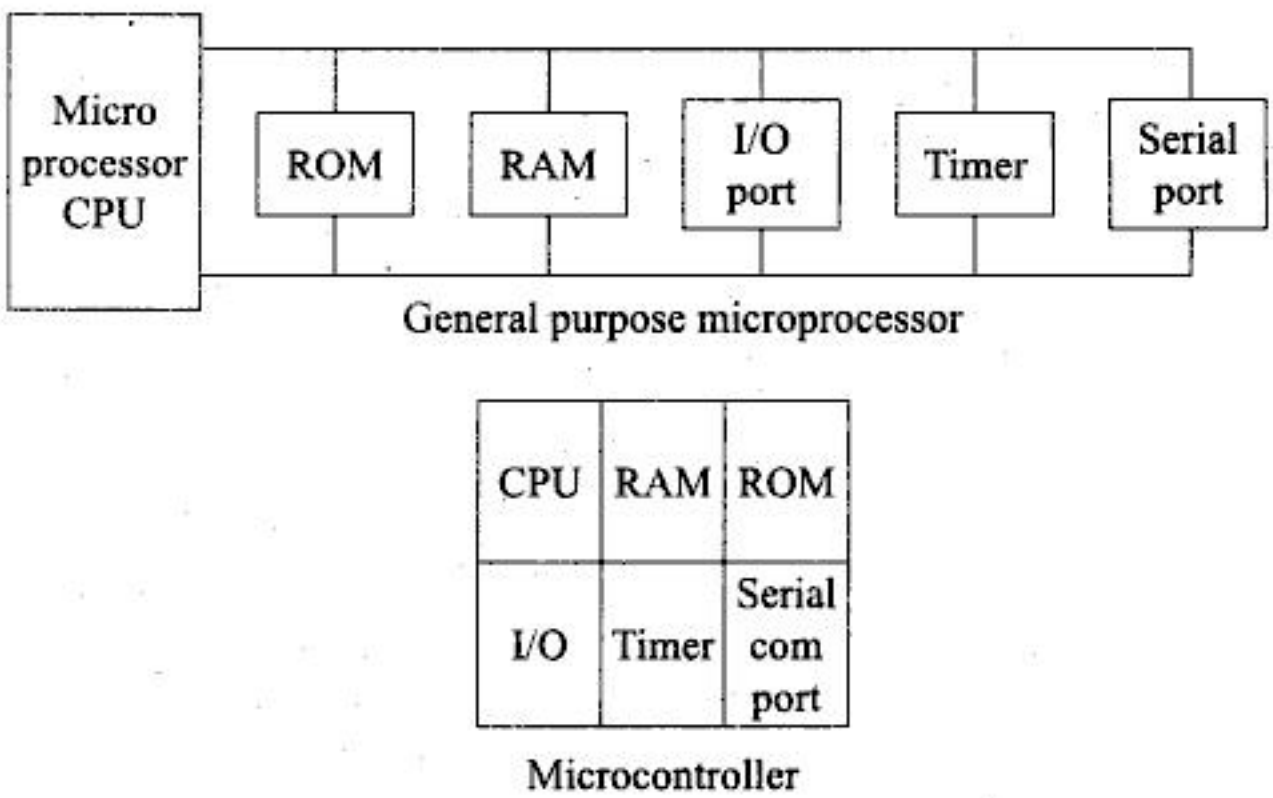


Figure 1.2. Microprocessor vs. Microcontroller.

Microcontrollers range from very simple to extremely complex designs. The microcontroller is meant to read data, perform limited calculations on it and *control the environment* depending on the calculations. Generally, the control of an operation is stored as a program, permanently in the microcontroller and does not change with the lifetime of the system. In general, the microcontroller uses a very limited number of instructions. Unlike microprocessors, in microcontroller, many instructions are coupled with pins on the IC chip. This means that several pins have multiple functions, which can be chosen by the programmer. In other words, the pins are programmable. (More of this in detail, when we discuss architecture).

Another major difference between the microprocessor and microcontroller is the way in which data can be moved between the memory and the CPU. In a microprocessor there are several instructions to achieve this in different ways, whereas in a microcontroller there are very few instructions to achieve this.

Microcontrollers also have several bit-handling instructions. This means that individual bits can be operated upon or changed in some registers. This makes them extremely suitable for control applications. Normally, in microprocessors most of the instructions are byte-handling, which means that the complete byte can be operated upon/changed in the registers. They are equipped with very few bit-handling instructions. Common microcontrollers are Intel 8051,

DS89C440, DS87520, AT89LV52, etc. In brief the comparative features of microprocessors and microcontrollers are shown in Table 1.1.

Table 1.1 Comparison of microprocessors and microcontroller.

	Microprocessor	Microcontroller
1.	CPU is stand-alone. RAM, ROM, I/O, timer are separate and interfaced with CPU	CPU, RAM, ROM, I/O and timer are all located on a single chip
2.	Designer can decide on the amount of ROM, RAM etc. to be connected	Fixed amount of ROM, RAM, I/O ports on chip
3.	Expansive applications	Applications in which cost, space and power are critical
4.	Versatile and general purpose	Not very versatile
5.	Large number of instructions with flexible addressing modes	Limited number of instructions with few addressing modes
6.	Very few instructions which have bit-handling capability	Many instructions with bit-handling capability

An *embedded system* is a system with the processor/controller embedded into that application. An embedded product uses a microprocessor or a microcontroller to do one task only. In an embedded system generally, there is only one application software that is burned into the ROM. Examples are video game players, mp3, printer, electronic lock etc. A critical need of an embedded system is to decrease power consumption. This is done by integrating more functions into the chip. At times the terms embedded system and microcontroller are used interchangeably. Embedded products using microcontrollers are used in a variety of applications:

- **Home Appliances**—Intercom, VCR, Cellular phones, Video games, Camera, Fax machines, Security systems, TVs, Musical instruments, door openers, sports gear etc.
- **OfŒe**—Telephones, Computers, Security systems, Laser printer, Computers, Air conditioners and heaters etc.
- **Others**—Instrumentation, space craft, Smart cards, Entertainment, etc.

The processors are broadly divided into CISC and RISC systems as discussed next.

1.4 CISC and RISC Systems

Based on the instruction set, we broadly classify computers/microprocessors/microcontrollers into CISC (Complex Instruction Set Computers) and RISC (Reduced Instruction Set Computers) devices. In CISC devices, there are a large number of instructions, each of which has a different permutation of the same operation (like data access, data transfer etc.). This gives the programmer flexibility in writing the programs. The major characteristics of CISC systems are

- Typically large number of instructions; around 100–250 instructions.
- Some instructions perform specialized tasks and are used infrequently.
- A large number of addressing modes. (Addressing mode is the manner in which data is obtained).
- Variable length instruction formats.
- Execution time for an instruction may take several clock cycles.
- Execution time for each instruction may be different.
- Efficient use of memory.
- Robustness of the instruction set is given precedence over speed.
- Instructions are available to manipulate operands in memory.

RISC processors on the other hand have a very limited number of instructions. The major characteristics of RISC systems are

- Relatively few instructions.
- Instructions are executed in small clock periods. Hence they are faster than CISC.
- Very few addressing modes.
- Limited memory access made available to certain instructions.
- All operations are performed within the registers of the CPU.
- Instructions are of fixed length.
- Control is hardwired in the system.

Typical CISC microprocessors are 8085, 8086, Pentium (all from Intel), M6800 (Motorola), Z-80 (Zilog) and microcontrollers are 8051 series (Intel). A few examples of RISC microcontrollers are the PIC microcontroller series from Microchip (only 33/35 instructions).

We now go to the different languages available to program computing systems.

1.5 Computing Languages

Digital devices recognize, understand and operate binary numbers. A number of bits are combined into a single unit, during data processing. A *word* is defined as the number of bits a processor/controller recognizes and processes at a time. The word length ranges from four bits in small systems to 64-bits in high speed processors. A group of 8 bits is called a *byte* and a group of 4 bits is called a *nibble*.

Each microprocessor/microcontroller has its own set of instructions, communicated to it in binary language called the *Machine Language*. An instruction in the machine language is simply, a combination of bits to give a specific meaning to the logic circuits using them. It would be extremely difficult for people to read and write machine language instructions. For example, '00111100' represents the instruction to increment the contents of a register called the accumulator in 8085 microprocessor. This string of bits hardly makes any sense to a person reading it, nor does it appear meaningful, since it bears no direct relevance to its task. However, this is the only language the CPU understands!

To help in easier programming, we use English like words, to program in *Assembly Language*. For example the instruction to increment the accumulator is written as 'INR A', which makes a lot more sense than '00111100'! These instructions are called *mnemonics*. The mnemonics are converted to machine language using an Assembler. The assembly language is specific to the microprocessor/microcontroller and hence the programs are not transferable from one system to another. In other words the programs are said to lack 'portability'. To overcome this limitation, general purpose languages have been developed which are independent of the processor. They are called *High-Level Languages*. Examples are Basic, Fortran, C, C^{++} etc. The instructions in high-level languages are called *Statements*.

$$C = A + B$$

is a statement in Fortran which assigns the variable C to the sum of A and B. Programs written in high-level languages are independent of the processor and hence are easily portable form one machine to another.

These statements are converted into machine language instructions using a Compiler or an Interpreter. A *compiler* translates the entire high-level language program called the *source code*, into machine language code, called the *object code*. *Interpreters* on the other hand, read one instruction at a time and convert it into object code.

Compilers and interpreters require large memory space because higher-level languages require several machine codes to translate into binary. On the

other hand, assembly level languages have a one is to one correspondence between the mnemonics and the machine code. Therefore, assembly language programs are compact and require less memory space. They are suitable for programs which are small and compact. They are also preferred for real-time applications where the program efficiency is critical.

1.6 Memory

Memory is used to store instructions and data. What exactly is memory? It is a device which can store binary digits—a 0 or a 1—in the form of two discrete voltage levels or as charge in a capacitor. A flip-flop or a latch is a basic element of memory, where the bits are stored as voltage levels. Its called a memory cell. A group of flip-flops are used to store a *word*. This group is identified as a single unit, by a memory address. The number of bits in this group determines the *word length* of the memory chip. The number of bits a semiconductor memory can store is called a chip *capacity*. This can be in Kbits (Kilo bits) or Mbits (Mega bits). It should be noted that the storage capacity of memory chips is given in bits, whereas the storage capacity of computing systems is in bytes.

An important characteristic of the memory is the speed at which the data in the memory can be accessed. To access the data, the address is placed on the address pins, the READ pin is activated, and after a certain amount of time has elapsed, the data is loaded on to data pins. The speed of the memory chip is called the *access time*.

Basically memories are classified into *prime memory* which is used in the system for storing instructions and data and *storage memory*, used to store information. Prime memory are of different types. They are briefly discussed below.

1.6.1 Random Access Memory (RAM)

These memory locations can be randomly accessed for information transfer. A simple block diagram of a RAM is shown in Fig. 1.3.

The k address lines can address a maximum of 2^k memory locations. The size of a RAM with k address lines and n data lines is 2^k words, or $2^k \times n$ bits. The two operations RAM can perform are read and write operations. The Write operation, transfers a word into the memory (writes to the memory) and a Read operation transfers a word out of memory (reads from memory). This memory is *volatile*, which means that when the power is turned off, the contents are destroyed. Static RAM (SRAM) is made of flip-flops. Each flip-flop normally

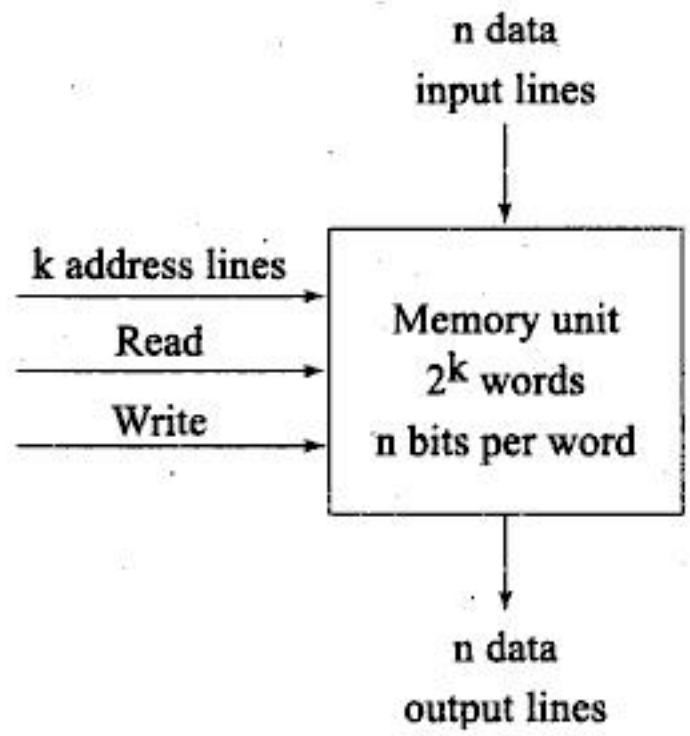


Figure 1.3. Random Access Memory.

requires 6 transistors (or MOSFETs) to hold a single bit of data. It is expensive and significantly faster than Dynamic RAM (DRAM). It is therefore used where either speed or low power, or both, are principal considerations. SRAM is also easier to control (interface to). SRAM is less dense than DRAM and is therefore not used for high-capacity, low-cost applications such as the main memory in personal computers. The power consumption of SRAM varies widely depending on how frequently it is accessed; it can consume heavy power when used at high frequencies. On the other hand, static RAM used at a slower pace, such as in applications with moderately clocked microprocessors, draw very little power and can have a nearly negligible power consumption when sitting idle—in the region of a few microwatts.

Dynamic RAM is the most common type of memory in use today. Inside a dynamic RAM chip, each memory cell holds one bit of information and is made up of two parts: a transistor and a capacitor. The capacitor holds the bit of information—a 0 or a 1, as charge. The transistor acts as a switch that lets the control circuitry on the memory chip read the capacitor or change its state. The main advantages of DRAM are high density, cheaper cost per bit, and lower power consumption per bit. The problem with the capacitor is that is has a leak. In a matter of a few milliseconds it gets discharged. Therefore, for dynamic memory to work, either the CPU or the *memory controller* has to recharge all of the capacitors holding a 1, before they discharge. To do this, the memory controller reads the memory and then writes it back. This *refresh* operation happens automatically thousands of times per second. The dynamic RAM gets its name from this refresh operation. This refreshing takes time and slows down the memory. DRAMs are available today in the range of 2G-bit capacity (G-Giga = 10^9).

Some standard RAM chips are shown in Table 1.2.

Table 1.2 RAM chips.

Chip	Type	Capacity	Organization	Speed
6116P-1	SRAM	16 K	2K × 8	100 ns
6116LP-3	SRAM	16 K	2K × 8	150 ns
6264LP-12	SRAM	64 K	2K × 8	120 ns
62256LP-12	SRAM	256 K	32K × 8	120 ns
4164-15	DRAM	64 K	64K × 1	150 ns
41464-8	DRAM	256 K	256K × 4	80 ns
41256-6	DRAM	256 K	256K × 1	60 ns
511000P-8	DRAM	1 M	1M × 1	80 ns
514100-7	DRAM	4 M	4M × 1	70 ns

1.6.2 Read Only Memory (ROM)

It's a memory unit that performs a read operation only. It does not have a write capability. This means that binary information stored once in ROM is permanent and cannot be altered. An $m \times n$ ROM is an array of memory cells organized into m words of n bits each. In a computer system, ROM is used to store fixed programs which are not to be altered. In *Mask ROM* the desired contents are permanently programmed in it, by the IC manufacturer, during fabrication. It is not user-programmable. It is expensive and used when the needed volume of chips is high and it is sure that the contents will not change. It must be noted that all ROM memories have 8 data pins. A typical ROM is shown in Fig. 1.4.

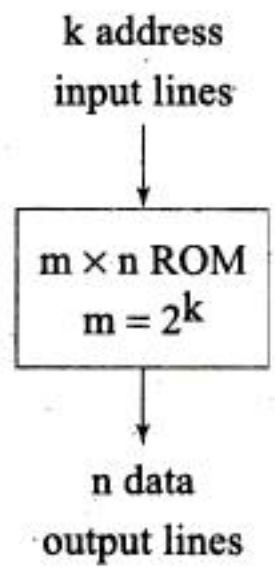


Figure 1.4. Read Only Memory.

Programmable Read-Only Memory (PROM), or ***one-time programmable ROM*** (OTP), can be written to or programmed via a special device called a PROM programmer. Typically, this device uses high voltages to permanently destroy or create internal links within the chip. Consequently, a PROM can only be programmed once. Erasable PROM (EPROM) was invented to

make changes in the contents of PROM after it is burned. It can be erased by exposure to strong ultraviolet light (typically for 10 minutes or longer), then rewritten with a process that again requires application of higher than usual voltage. It can be erased typically thousands of times. Repeated exposure to UV light will eventually wear out an EPROM. Electrically Erasable PROM (EEPROM) is based on a similar semiconductor structure to EPROM, but allows its entire contents (or selected banks) to be electrically erased, then rewritten electrically. In addition, in EEPROM we can select which byte is to be erased, unlike an EPROM where the entire contents of the ROM are erased. The advantage of the EEPROM is in the fact that we can erase and program its contents while it is still in the system board and it does not require physical removal of the chip. Neither does it require a special device for erasure or programming.

Flash memory (or simply *flash*) is a modern type of EEPROM invented in 1984. Flash memory can be erased and rewritten faster than ordinary EEPROM, and newer designs have very high endurance (exceeding 1,000,000 cycles). Here the entire memory contents are erased, unlike EEPROM where partial bytes can be erased. Some standard ROM chips are shown in Table 1.3.

Table 1.3 ROM chips.

Chip	Type	Capacity	Organization	Speed
2716	UV-EPROM	16 K	2 K × 8	450 ns
27C32-1	UV-EPROM	32 K	4 K × 8	450 ns
27128-25	UV-EPROM	128 K	16 K × 8	250 ns
27C040-15	UV-EPROM	4096 K	512 K × 8	150 ns
2864A	EEPROM	64 K	8 K × 8	250 ns
28C256-25	EEPROM	256 K	32 K × 8	250 ns
28F256-20	Flash	256 K	22 K × 8	200 ns
28F020-15	Flash	2048 M	256 K × 8	150 ns

1.6.3 Cache memory

It is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. SRAMs are used as cache memory, since they are faster than DRAMs.

1.6.4 Memory Latency

It is the time between initiating a request for a byte or word in memory, until it is retrieved. If the data are not in the processor's cache, it takes longer to obtain

them, as the processor will have to communicate with the external memory cells. Latency is therefore a fundamental measure of the speed of memory: the less the latency, the faster the reading operation.

1.7 Computer Architecture: Von Neumann and Harvard

Every computer needs to store the instructions or code and also the data. Depending on how these are stored in the memory and how the memory is accessed we have two broad classifications for the architecture, namely, Harvard and Von Neumann.

Von Neumann Architecture: (Princeton architecture)

The block diagram of the Von Neumann architecture is shown in Fig. 1.5.

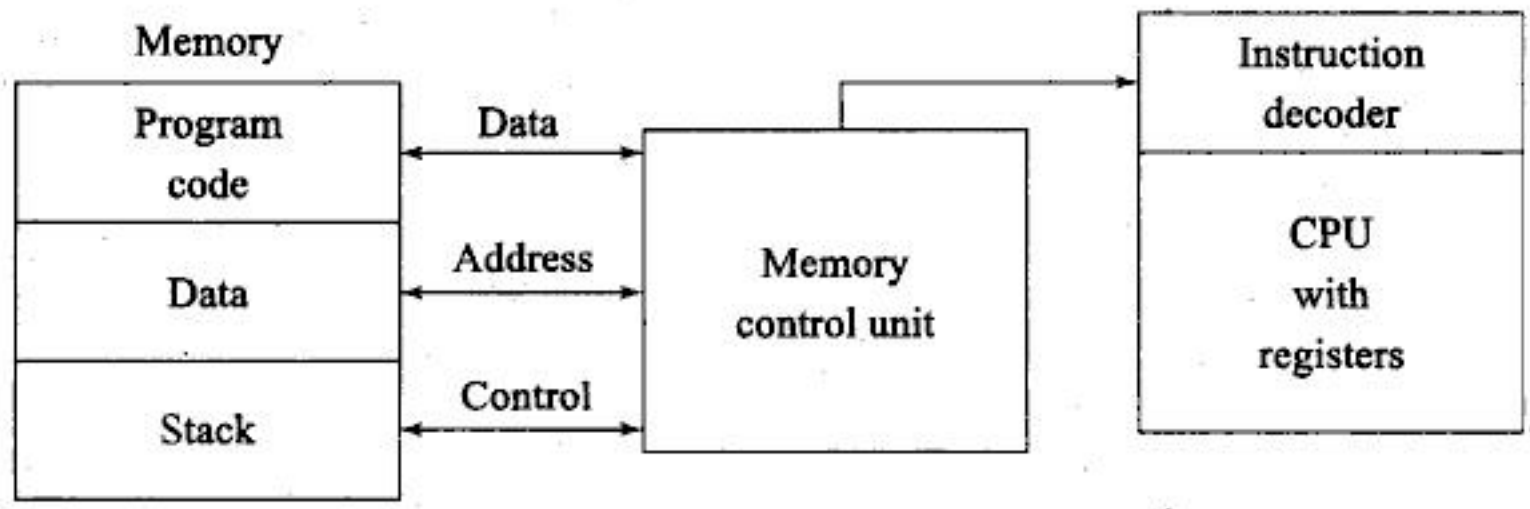


Figure 1.5. Block diagram of Von Neumann Architecture.

The main features of this architecture are as follows:

- It uses a single memory space for both instructions and data. It is also called a 'stored-program computer'.
- It has limited data transfer rate—called throughput—between the CPU and memory, compared to the amount of memory. In modern machines, throughput is much smaller than the rate at which the CPU can work. This seriously limits the effective processing speed when the CPU is required to perform minimal processing on large amounts of data, as the CPU is continuously forced to wait for vital data to be transferred to or from memory.
- As CPU speed and memory size have increased much faster than the throughput between them, this bottleneck has become more of a problem.
- Program modifications can be quite harmful, either by accident or design. In some simple stored-program computer designs, a malfunctioning program

can damage itself, other programs, or the Operating System possibly leading to a crash. This ability for programs to create and modify other programs is also frequently exploited by Malware (malicious software).

Most processors like 8085, 8086, M6800 etc. use this architecture.

Harvard architecture

The block diagram of a Harvard architecture system is shown in Fig. 1.6.

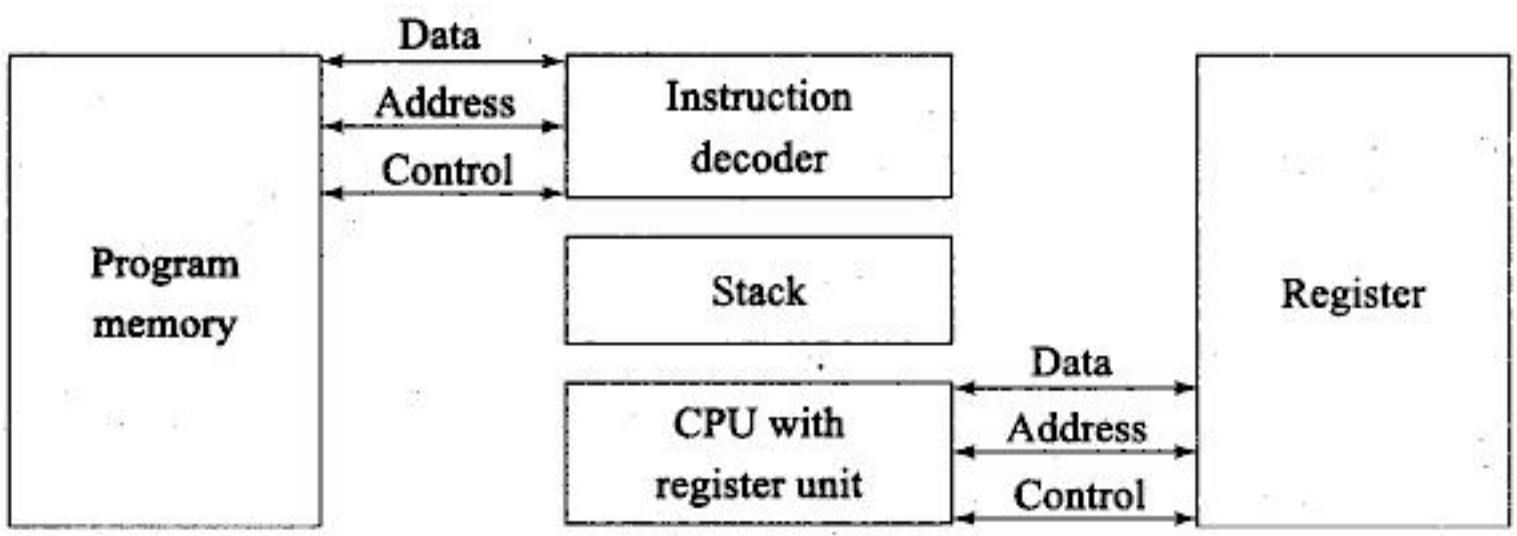


Figure 1.6. Block diagram of Harvard Architecture.

The main characteristics of this architecture are

- Physically separate storage and signal pathways for instructions and data. This implies that there is a separate 'Program memory' and 'Data memory'.
- The characteristics of the two memories like the word width, timing implementation technology, and memory address structure and size can be different.
- In some systems, instructions can be stored in ROM while data memory is generally RAM. In some systems, there is much more instruction memory than data memory, so the size of instruction addresses are much larger than data addresses. (For example we can have 10 K bytes allotted for instructions and 2 K for data).
- The CPU can both read an instruction and perform a data memory access at the same time, even without a cache memory. A Harvard architecture computer can thus be faster for a given circuit complexity because instruction fetches and data access do not use a single memory pathway.

Harvard architectures are also frequently used in:

- Specialized Digital Signal Processors DSPs, commonly used in audio or video processing products.

- Most general purpose small microcontrollers used in many electronics applications, such as the PIC microcontrollers by Microchip Technology, Inc., and AVR by Atmel Corp. These processors are characterized by having small amounts of program and data memory, and take advantage of the Harvard architecture and RISC so that most instructions are executed within only one machine cycle.

1.8 Evolution of microcontrollers—4-bit to 32-bit

Different applications demand microcontrollers that offer the right amount of functionality at minimum cost. A single microcontroller design is not possible to meet all the demands economically. Additional functions have been incorporated on-chip such as

- A/D converters which can convert an analog signal to a digital signal.
- Serial data communication—Synchronous and asynchronous.
- Watch dog timers which reset the controller if the program hangs.
- Pulse width modulation.
- Phase locked loops, used for synchronous communication.
- External bus controllers.

1.8.1 Selection of a microcontroller

There are a wide variety of microcontrollers available in the market. Programs written for one will not run on others. The choice of the microcontroller is determined by three parameters;

(i) It must perform the required task efficiently and effectively. Here we consider the following during selection:

- Speed
- The memory on chip—both ROM and RAM
- Packaging—the number of pins and the packaging format. This determines the required space and assembly layout.
- Power consumption.
- The number of I/O ports available.
- Ease of upgrading.
- Cost per unit.

(ii) Ease of product development. This includes the availability of software development tools like assemblers, compilers, debuggers, emulators, simulators etc.

(iii) Availability and reliable source for the microcontroller.

1.8.2 4-bit microcontrollers

These microcontrollers are widely used. The number of pins of a chip depends on the data size, commonly handled by the microcontroller. Hence, 4-bit microcontrollers are compact. They are cheapest smart-chips available in the market and used in LED/LCD display drivers, portable battery chargers, etc. Examples of 4 bit microcontrollers are 2902 Slice (Altera); M34501 (Renasas); ATAM862-3, ATAM862-4, ATAM862-8 (ATMEL).

1.8.3 8-bit microcontrollers

They are the most popular and widely used microcontrollers in the market today with a number of companies manufacturing them. Eight bit data word has been found adequate for a number of control applications. 8 bit controllers can have 256 decimal values. Further, the ASCII code is 8 bits long, making this size effective for serial data communication. Another incentive is the fact that most low-cost memory chips store one byte per memory location and hence can be easily interfaced with the microcontroller.

The most popular amongst the 8-bit microcontrollers is the 8051 series. It was developed by Intel in 1980 for use in embedded systems. Today it has been superseded by a vast range of faster and functionally enhanced 8051—compatible devices manufactured by many companies including Atmel, Maxim integrated products, Philips semiconductor, Nuvoton, Silicon laboratories, Texas instruments and Cypress semiconductor. Intel refers to its 8051 series as MCS-51. Two other members of 8051 family are 8031 and 8052. 8031 is a cut down version of the original Intel 8051, without internal ROM memory. The 8052 is an enhanced version of the original 8051, with increased internal ROM and RAM. A comparison of the features of 8031, 8051 and 8052 is given in Table 1.4.

Variations include providing A/D and D/A converters on-chip, Flash and EEPROM on-chip and differences in ROM sizes. Dallas semiconductors has a range of 8051/8052 versions of microcontrollers differing mainly in the on-chip ROM and RAM size like DS89C420/30 which has 16 K flash memory, 256 bytes of RAM and DS80C320 which has no ROM and only 256 bytes RAM. Similarly Atmel corporation has many versions like AT89C51, etc.

Table 1.4 .

Feature	8031	8051	8052
On chip ROM (bytes)	0	4 K	8 K
On chip RAM (bytes)	128	128	256
Timers	2	2	3
I/O Pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	6	8

Some of the current trends in design of processors and controllers are to include pipelining, superscaling and out of order execution. In pipe lining, an instruction is executed in a number of stages at the same time, to increase the execution speed. Data path is split into a number of different functional units and multiple instructions can use the data path at the same time. Superscaling is used with parallel processing of instructions which permits more than one instruction per clock cycle. Instructions can also be fed into the processor, not necessarily in the same sequence as which they are executed. This is called out of order execution. This optimizes hardware-software utilization. Motorola uses this.

1.9 Summary

In this chapter you have been introduced to the terms commonly used in computing systems. A brief history of microprocessors and microcontrollers has been given, to familiarize the reader with the evolution of computing systems. RISC and CISC processors provide the user with a choice to give priority to speed or simplicity. The instructions and data are all stored in memory. They can be stored in the same memory like in Von Neumann architecture or in separate memory like in Harvard architecture. The reader is now ready for a detailed discussion of 8051, in the coming chapters!

1.10 Questions

1. List the differences between microcontrollers and microprocessors.
2. What are the basic units in a digital computers?
3. List some common applications of microcontrollers.
4. Elaborate on Princeton and Harvard architecture used in processors.
5. Discuss the differences between RISC and CISC computers.

6. Describe the basic unit of a ROM and RAM.
7. What are the different types of RAM and ROM available?
8. Define memory latency.
9. Distinguish between assembly language and high level language.
10. What are the features which dictate the choice of a microcontroller for an application?

Chapter 2

Data Representation

Information stored and processed in memory or processor registers is always in binary form. The binary digits can be manipulated in different ways. This chapter deals with the representation of data in binary form and all operations associated with it.

Learning objectives

At the end of the chapter you will be able to:

- Represent fixed point and floating point numbers in binary
- Find the complement's of numbers
- Perform binary arithmetic
- Know binary codes
- Perform BCD operations
- Understand error detecting codes.

2.1 Introduction

Binary information in digital computers is stored in memory and the processor registers. Control information in the form of a bit or group of bits, is used to specify the control signals needed for data manipulation in other registers.

The data types may be classified into one of the following (i) numbers used in arithmetic computations (ii) alphabets used in data processing (iii) other symbols used for specific purposes. All types of data are represented in computer registers in binary coded form. Binary number system is most natural for digital systems. However, other number systems are used, especially decimal number system, since it is familiar to people.

2.2 Number system

A number system of radix r (also called base) is a system that uses distinct symbols for r digits. Numbers are represented by a string of digits. To determine the value of the number, each digit has to be multiplied by an integer power of r and then the sum of all weighted digits taken. For example in decimal system, $r = 10$ representing digits from 0 to 9. A number say 324.8 is understood to represent the value,

$$3 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 8 \times 10^{-1},$$

3 hundreds + 2 tens + 4 units + 8 tenths. Similarly, any decimal number can be interpreted to find the value it represents. In general a *Number system* with *radix r* is given weights as follows:

$$\text{Number} : \cdots a_4\, a_3\, a_2\, a_1\, a_0 \cdot a_{-1}\, a_{-2}\, a_{-3} \cdots$$

$$\text{Value} : \cdots + a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \cdots$$

2.2.1 Binary system

The binary system uses a radix 2. The two digit symbols used are 0 and 1. Generally a number is written within parentheses and the radix written as a subscript. For example

$$(01100101)_2; \quad (823.6)_{10} \quad \text{and so on.}$$

A binary number

$$(01100101)_2 = 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^6$$

$$= 0 + 64 + 32 + 0 + 0 + 4 + 0 + 1$$
$$= (101)_{10}$$

2.2.2 Octal system

The octal system uses a radix 8, to represent digits from 0 to 7. Its weights are powers of 8. For example

$$(432.6)_8 = 4 \times 8^2 + 3 \times 8^1 + 2 \times 8^0 + 6 \times 8^{-1}$$
$$= 4 \times 64 + 3 \times 8 + 2 \times 1 + \frac{6}{8} = (282.75)_{10}.$$

Since $2^3 = 8$, each octal digit corresponds to three binary digits. We convert from octal to binary and vice versa in the example next.

Example 2.1.

(i) Convert $(324.7)_8$ to binary

(ii) Convert $(10100101)_2$ to octal and decimal

Solution

(i) $(324.7)_8 = (\underbrace{011}_{3}\ \underbrace{010}_{2}\ \underbrace{100}_{4} \cdot \underbrace{111}_{7})_2$

(ii) $(\underbrace{10}_{2}\ \underbrace{100}_{4}\ \underbrace{101}_{5})_2 = (245)_8 = (165)_{10}$

Decimal value can be obtained as follows

$$1 \times 2^8 + 0 + 1 \times 2^6 + 0 + 0 + 1 \times 2^2 + 0 + 2^1 = (165)_{10}$$

or

$$2 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 = (165)_{10}.$$

2.2.3 Hexadecimal system

The hexadecimal system uses the radix 16. The 16 symbols are the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and the alphabets A, B, C, D, E, F to represent decimal 10, 11, 12, 13, 14 and 15 respectively. Consider,

$$(3A2)_{16} = 3 \times 16^2 + A \times 16^1 + 2 \times 16^0$$
$$= (930)_{10}$$

Since $2^4 = 16$, each Hex digit is represented by a 4-bit binary word.

Example 2.2.

(i) Convert $(26B)_{16}$ into binary and octal.

(ii) Convert $(10011100)_2$ into hex and octal.

Solution

(i) $(26B)_{16} = (\underbrace{0010}_{2}\ \underbrace{0110}_{6}\ \underbrace{1011}_{B})_2 = 2\times16^2+6\times16^1+11\times16^0 = (619)_{10}$

$\underbrace{001}_{1}\ \underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{011}_{3} = (1153)_8 = 1\times8^3+1\times8^2+5\times8^1+3\times8^0$

$= (619)_{10}$

(ii) $(\underbrace{1001}_{9}\ \underbrace{1100}_{C})_2 = (9C)_{16} = (156)_{10}$

$(\underbrace{10}_{2}\ \underbrace{011}_{3}\ \underbrace{100}_{4})_2 = (234)_8 = (156)_{10}$

(You can always pad with leading zeros).

Table 2.1 gives the binary coded octal numbers and Table 2.2 gives the hexadecimal number coded in binary.

2.2.4 Conversion from decimal to radix *r*

Conversion from decimal to its equivalent representation in the radix *r* system is carried out by separating the number into its integer and fraction part

Table 2.1 Binary coded octal numbers..

Octal number	Binary-coded octal	Decimal equivalent
0	0 0 0	0
1	0 0 1	1
2	0 1 0	2
3	0 1 1	3
4	1 0 0	4
5	1 0 1	5
6	1 1 0	6
7	1 1 1	7
1 0	0 0 1 0 0 0	8
1 1	0 0 1 0 0 1	9
1 1	0 0 1 0 1 0	10
6 2	1 1 0 0 1 0	50
1 2 3	0 0 1 0 1 0 0 1 1	83
6 5 7	1 1 0 1 0 1 1 1 1	431

Table 2.2 Binary coded Hex numbers.

Hexadecimal number	Binary-coded hexadecimal	Decimal equivalent
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
A	1 0 1 0	10
B	1 0 1 1	11
C	1 1 0 0	12
D	1 1 0 1	13
E	1 1 1 0	14
F	1 1 1 1	15
2 3	0 0 1 0 0 0 1 1	35
4 9	0 1 0 0 1 0 0 0	73
E 7	1 1 1 0 0 1 1 1	231

and converting each one of them separately. The integer part of the decimal number is converted to base r, by successive divisions by r and accumulation of the remainders. The last remainder forms the most significant digit. The conversion of the decimal fraction to radix r is accomplished by successive multiplications by r and accumulation of the integer digits.

Example 2.3. Convert $(32.5)_{10}$ into (i) binary, (ii) octal and (iii) hexadecimal equivalent numbers.

Solution

(i)

Binary Integer

2	32		
2	16	——	0
2	8	——	0
2	4	——	0
2	2	——	0
	1	——	0

Fraction

0.5×2

1.0

$(32.5)_{10} = (100000.1)_2$

(ii) Octal

$8 \mid 32$

$4 \quad —— \quad 0$

0.5×8

4.0

$(32.5)_{10} = (40.4)_8$

(iii) $16 \mid 32$

$2 \quad —— \quad 0$

0.5×16

8.0

$(32.5)_{10} = (20.8)_{16}$

The registers in processors contain a string of binary numbers. Specifying the contents of the registers in octal reduces the digits by one-third and representing by hexadecimal reduces the digits by one fourth. Consider a 16-bit register contents, "1001000111010101", It can be easily specified as 91D5 in hexadecimal or 110725 in octal. Manuals generally choose the hexadecimal system to specify contents of register.

2.3 Decimal representation

Decimal system is most commonly used by human beings, but cannot be understood by the computer. To overcome this problem, all decimal numbers are converted into binary numbers; operations performed in binary, and result converted back to decimal numbers.

A common representation of decimal numbers is the Binary-Coded Decimal (BCD). A binary code is a group of n bits which can be combined in 2^n combinations. The decimal system has 10 distinct digits, 0–9 and hence we need 10 distinct combinations to symbolize each one of the digits. 3-bits are insufficient since $2^3 = 8$, with four bits we get $2^4 = 16$, combinations. Therefore six combinations will be redundant and go unused. We can choose 10 combinations out of 16, in a number of ways giving rise to numerous different codes. The assignment most commonly used is the direct binary assignment as listed in Table 2.3.

Note that the combinations from 1010 to 1111 are unassigned. Each code can be thought of as a symbol to represent the corresponding decimal number. To get the BCD of a number, say 847, we write the codes for 8, 4 and 7 as

$$847 \rightarrow 1000 \quad 0100 \quad 0111$$

Table 2.3 Binary-coded decimal numbers.

Decimal number	BCD number
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
45	0100 0101
92	1001 0010

It is important to understand the difference between converting a decimal number to binary and binary coding of the number. For example,

$$(16)_{10} = (10000)_2 = 1 \times 2^4 + 0 + 0 + 0 + 0$$

$$(16)_{10} = \underbrace{0001}_{1} \ \underbrace{0110}_{6} \text{ in BCD.}$$

2.4 Complements

Complements are used in digital computers, normally to simplify the subtraction operation. There are two types of complements.

2.4.1 $(r-1)$'s complement

Given a number N in radix r having n digits, $(r-1)$'s complement of N is defined as $(r^n - 1) - N$. In decimal system, $r = 10, r - 1 = 9$. Therefore 9's complement is $(10^n - 1) - N$. Now 10^n is 1 followed by n zeroes. $10^n - 1$ will be n 9's. For example if $n = 3$, $10^3 - 1 = 1000 - 1 = 999$ (3 9's). Therefore the 9's complement of a number N is obtained by subtracting each digit from 9.

Example 2.4. What is the 9's complement of (i) 834 (ii) 62937 (iii) 900.

Solution

(i) 9's complement of 834 is $999 - 834 = 165$

(ii) 9's complement of 62937 = 37062

(iii) 9's complement of 900 = 099.

For binary numbers, $r = 2$ and $r - 1 = 1$. The 1's complement of a number N with n bits is $(2^n - 1) - N$. Again in binary, 2^n is represented by a 1 followed by n zeroes. For example if $n = 3$, $2^3 = 1000$. So $2^n - 1$ is a binary number represented by n 1's. For example, if $n = 3$, $2^n - 1 = 1000 - 1 = 111$. So the 1's complement of a binary number is obtained by subtracting each digit from 1. We know $1 - 1 = 0$ and $1 - 0 = 1$. Therefore, the 1's complement is simply formed by changing the 1's to zero and 0's to 1.

Example 2.5. Determine the 1's complement of (i) 111 (ii) 100010 (iii) 001101.

Solution

(i) 1's complement of 111 is 000

(ii) 1's complement of 100010 is 011101

(iii) 1's complement of 001101 is 110010.

2.4.2 *r*'s complement

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $n \neq 0$ and 0 for $N = 0$. It is obtained by adding 1 to the $(r - 1')$s complement.

Example 2.6.

(i) Find the 10's complement of 834

(ii) 2's complement of 111

Solution

(i) 9's complement of 834 = 165

10's complement of 834 = 166

(ii) 1's complement of 111 is 000

2's complement of 111 is 001.

* The 10's complement of N can be formed by leaving all least significant 0's unchanged, subtracting the first non zero least significant digit from 10, and then subtracting all higher digits from 9.
* 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged, and then replacing 0's by 1's and 1's by 0's.

2.4.3 Subtraction of unsigned numbers using r's complement

The subtraction of two n-digit unsigned numbers $M - N$ $(N \neq 0)$ in base r is done as follows:

1. Add the minuend M to the r's complement of the subtrahend. We get $M + (r^N - N) = M - N + r^n$.
2. If $M \geq N$, the sum produces an end carry r^n which when discarded gives $M - N$.
3. If $M < N$, the sum does not produce carry and we get $r^n - (N - M)$ which is the r's complement of $(N - M)$. So to get the answer, take r's complement of sum and place negative sign.

Example 2.7. Perform the following

(i) 34672 – 18024

(ii) 18024 – 34672

Solution

(i)

$$\begin{aligned} \text{M} &= 34672 \\ \text{10's complement of N} &= 81976 \\ \hline & 1,16648 \\ & \downarrow \\ & \text{discard} \end{aligned}$$

(ii)

$$\begin{aligned} \text{M} &= 18024 \\ \text{10's complement of N} &= 65328 \\ \hline & 83352 \end{aligned}$$

There is no end carry. Take 10's complement of 83352 i.e. 16648. So answer is −16648.

Example 2.8. Perform the following

(i) 10110110 − 10001001

(ii) 10001001 − 10110110

Solution

(i)

$$\begin{aligned} \text{M} &= 10110110 \\ \text{2's complement of N} &= 01110111 \\ \hline & 1,00101101 \\ & \downarrow \\ & \text{discard} \end{aligned}$$

(ii)

$$\begin{aligned} \text{M} &= 10001001 \\ \text{2's complement of N} &= 01001010 \\ \hline \text{No carry} \quad & 11010011 \end{aligned}$$

Take 2's complement of result i.e. 00101101. Answer is −00101101.

In the above examples we were dealing with unsigned numbers. We need a method to represent negative numbers.

2.5 Fixed-Point representation

Positive integers including zero can be represented as unsigned numbers. We need a notation to represent negative integers. In our arithmetic we indicate a negative number by a minus sign. Now we however need a notation in terms of 1's and 0's. As a convention, it is customary to represent the sign of a number with a bit placed in the leftmost position of the number. The sign bit is equal to 0 for positive numbers and 1 for negative numbers.

We also have to position the point to separate the integer and fraction part. The binary point cannot be stored in a register. The point can be specified by giving it a fixed position, by assuming it is always fixed in one position. The point can be in the extreme left of a register, in which case the stored number is

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

2.5.2 Arithmetic addition of signed numbers

The negative numbers are represented in the 2's complement form. The rule is as follows: Add the two numbers, including their sign bits, and discard any carry out from most significant bit. If the sum is negative, it is in two's complement. Lets consider the following examples

(a) (+6) + (+13)

$$\begin{array}{rr} +6 & 0,0000110 \\ +13 & 0,0001101 \\ \hline +19 & 0,0010011 \end{array}$$

(b) (+6) + (−13)

$$\begin{array}{rr} +6 & 0,0000110 \\ -13 & 1,1110011 \\ \hline -7 & 1,1111001 \\ & \downarrow \\ & \text{2's complement of } +7 \end{array} \qquad \begin{array}{rr} +13 & 0,0001101 \\ -13 & 1,1110011 \end{array}$$

(c) (−6) + (+13)

$$\begin{array}{rr} -6 & 1,1111010 \\ +13 & 0,0001101 \\ \hline +7 & 10,0000111 \\ & \downarrow \\ & \text{Discard} \end{array}$$

(d) (−6) − (−13)

$$\begin{array}{rr} -6 & 1,1111010 \\ -13 & 1,1110011 \\ \hline -19 & 11,1101101 \\ & \downarrow \\ & \text{Discard} \end{array}$$

Now lets consider another example

(+70) + (60)

```
          0   1
+70     0, 1000110
+60     0, 0111100
        ----------
        1, 0000010
        ----------
```

Here we see that the result which should have a positive number, is a negative number as the sign bit is 1. If we consider the result to be 9-bit including carry out as 010000010, it is +130, which is correct. But the 8-bit register cannot hold this! We say an **overflow** has occurred. The reason is the largest 8-bit positive number which can be represented is 0,1111111 which is +127. The result +130 is greater than this. The overflow is detected by EXOR operation of carry into sign bit and carry out of sign bit. Overflow occurs only if we add two positive numbers or add two negative numbers.

2.5.3 Arithmetic subtraction of signed numbers

The rule for subtraction is as follows:

- Take the 2's complement of subtrahend including the sign bit
- Add it to the minuend, including the sign bit
- Discard if there is any carry.

Consider 23 – 12

```
 23        0,  0010111   | +12  0, 0001100
-12        1,  1110100   | -12  1, 1110011 +1 = 11110100
           ---------     |
+11    1   0,  0001011   |
       ↓   ---------     |
    Discard              |
```

Now consider 70 – (–60). We now follow the same rule.

```
          0   1
70      0,1000110
60      0,0111100   * Take 2's complement of
        ----------    –60, which is +60.
        1,0000010
        ----------
```

Again overflow has occurred. It is again detected by EXOR of carry into sign bit and carry out of sign bit.

2.5.4 Decimal fixed-point representation

If we use the BCD code, each decimal digit would require 4-bits. For example 1348 would be stored as 0001 0011 0100 1000. Decimal representation needs more storage space since number of bits is greater than that needed for its binary representation. The hardware required to perform decimal arithmetic is also more complex. The signed representation uses 4-bits to conform with BCD code. A plus is normally designated with four 0's and a minus with the BCD equivalent of 9, 1001. Negative numbers are represented by their 10's complement.

BCD addition

The addition of two BCD numbers may not represent an appropriate BCD value. For example consider adding $(34)_{BCD}$ and $(27)_{BCD}$.

$(34)_{10}$	0011	0100	BCD
$(27)_{10}$	0010	0111	BCD
$(61)_{10}$	0101	1011	5BH

The result is 5BH. The reason is that the CPU can perform only binary addition. In BCD, any number larger than 9 is invalid and needs to be adjusted by adding 6 in binary. In the previous problem we add 5B + 06,

5B	0101	1011
06	0000	0110
	0110	0001 = 61_{BCD}.

We need to adjust to get BCD sum using the following rules for 2 digit BCD numbers:

(i) Add 06 if the sum of the first two BCD digits (lower nibbles)is greater than nine or if there is carry out from bit position three to bit position 4.

For example consider $49_{10} + 21_{10}$

49_{10}	=	0100	1001	BCD
21_{10}	=	0010	0001	BCD
				Greater than 9
70_{10}		0110	1010	6AH
	+06	0000	0110	← Correction
		0111	0000	70_{BCD}

Consider $49_{10} + 28_{10}$

49_{10}	=	0100 $\overset{1}{\frown}$ 1001	BCD	
28_{10}	=	0010 1000	BCD	
77_{10}		0111 0001	71H	← Carry from lower nibble to upper nibble
	+06	0000 0110		
		0111 0111	77_{BCD}	

(ii) Add 60 if neither of the conditions in (i) occurs and the sum of the upper nibbles is greater than 9 or there is a carry out of upper nibble. Lets consider two examples $94_{10} + 12_{10}$ and $94_{10} + 82_{10}$ to illustrate this.

94_{10}	10010100	BCD	
$+12_{10}$	00010010	BCD	
106_{10}	10100110	A6H	
+60	01100000		
	1,00000110	06 BCD with carry	

94_{10}	10010100	BCD	
82_{10}	10000010	BCD	
$1,76_{10}$	1,00010110	06H	
+60	01100000		
	1,01110110	76 BCD	

(iii) 66 is added whenever the conditions in (i) and (ii) are satisfied. Consider $99_{10} + 71_{10}$.

99_{10}	1001 1001	BCD	
71_{10}	0111 0001	BCD	
$1,70_{10}$	1,0000 1010	0AH	
+66	0110 0110		
	1,0111 0000	70 BCD	

We add 66 because the sum of lower nibbles is greater than 9 and there is carry from the upper nibble.

2.6 Floating-point representation

The floating-point representation has two parts. The first part represents a signed, fixed-point number called the mantissa. The second part designates the position of the point (decimal or binary) and is called the exponent. The fixed-point mantissa may be a fraction or an integer. Consider the decimal number 823.51 which can be written as follows

Fraction	Exponent
+0.82351	+03

The value of the exponent indicates that the actual decimal point is three positions to the right. Floating-point is always interpreted as

$$m \times r^e.$$

Only the mantissa m and the exponent e are physically represented in the registers (including their signs). The radix r and position of the point are always assumed.

A floating-point binary number is similarly represented. Consider a number $+1101{\cdot}11$, to be represented by a 8-bit fraction and a 6-bit exponent. We represent it as follows:

Fraction	Exponent
01101110	000100
↓	↓
Sign bit of fraction	Sign bit of exponent

The floating point number is equivalent to

$$m \times 2^e = +(\cdot 1101110)_2 \times 2^{+4}$$

Arithmetic operations with floating-point number are more complicated than arithmetic operations with fixed-point number. Their execution takes longer time and requires more complex hardware. Most computers have the capability to perform floating point arithmetic.

2.7 Other binary codes

Though the BCD code for decimal numbers discussed in the last section is very popular, there are other codes available, which have other applications. We shall discuss few of the popular ones.

2.7.1 Gray code

Gray code belongs to the class of codes referred to as *unit-distance codes*. The basic property of a unit-distance code is that only one bit changes between two successive integers which are being coded. The code is extremely useful in analog-to-digital conversion. It is also used to provide the timing sequences that control the operations in a digital system. The Gray code is a counter whose flip-flops go through a sequence of states, specified in Table 2.4.

Table 2.4 Gray code.

Decimal number	Gray code	Decimal number	Gray code
0	0000	8	1100
1	0001	9	1101
2	0011	10	1111
3	0010	11	1110
4	0110	12	1010
5	0111	13	1011
6	0101	14	1001
7	0100	15	1000

2.7.2 Other decimal codes

Binary codes for decimal digits require a minimum of four bits. Numerous codes can be formed choosing 10 of the 16 combinations of four bits. Some popular codes are listed in Table 2.5.

2421 and excess 3 codes are examples of self complementing codes. In them, the 9's complement of a decimal digit is obtained by replacing the 1's with 0's and 0's with 1's in the binary code.

8421 and 2421 are weighted codes. The bits are multiplied by the weight of the position and the sum of these taken to obtain the decimal digit. For example consider the following:

- In 8421, $1001 = 8 \times 1 + 0 + 0 + 1 \times 1 = 9$
- In 2421, $1011 = 2 \times 1 + 0 + 2 \times 1 + 1 \times 1 = 5$

The excess three code is an example of an unweighted code. Its binary code is obtained from the corresponding BCD binary number after addition of binary 3 (0011).

Table 2.5 Binary codes for decimal digits.

Decimal digit	BCD 8421	2421	Excess · 3	Excess · 3 Gray
0	0000	0000	0011	0010
1	0001	0001	0100	0110
2	0010	0010	0101	0111
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1100
6	0110	1100	1001	1101
7	0111	1101	1010	1111
8	1000	1110	1011	1110
9	1001	1111	1100	1010
Unused	1010	0101	0000	0000
combination	1011	0110	0001	0001
	1100	0111	0010	0011
	1101	1000	1101	1000
	1110	1001	1110	1001
	1111	1010	1111	1011

2.7.3 Alphanumeric codes

Many applications of digital computers require the handling of data, consisting not only of numbers, but also letters of the alphabets and certain special characters, such as $, +, / etc. The standard alphanumeric code is the ASCII (American Standard Code for Information Interchange) code. The earlier list, had a seven bit code. The recent ASCII is an 8-bit code, extended to include newer characters used with graphics. ASCII codes are shown in Table 2.6, though the list is not exhaustive.

Another alphanumeric code used in IBM is the EBCDIC (Extended BCD Interchange Code). Alphanumeric codes are used internally in a computer for data-processing or externally for data transmission.

Lets consider a few examples to strengthen the concepts visited in the chapter.

Example 2.9. Convert into decimal: 101110; 1110101 and 110110100.

Solution

(i) $$\begin{aligned} 101110 &= 1 \times 2^5 + 0 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \\ &= 32 + 0 + 8 + 4 + 2 + 0 \\ &= (46)_{10} \end{aligned}$$

Table 2.6 ASCII code.

Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011	0	011 0000
L	100 1100		011 1110
M	100 1101	Space	010 1000
N	100 1110	.	010 1001
O	100 1111	(	010 1011
P	101 0000	)	010 1101
Q	101 0001	+	010 0100
R	101 0010	-	010 1010
S	101 0011	$	010 1111
T	101 0100	*	010 1100
U	101 0101	/	011 1101
V	101 0110	,	001 0000
W	101 0111	=	001 0000
X	101 1000		
Y	101 1001		
Z	101 1010		

(ii)

$$\begin{aligned} 1110101 &= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 + 1 \times 2^2 + 0 + 1 \\ &= 64 + 32 + 16 + 0 + 4 + 0 + 1 \\ &= (117)_{10} \end{aligned}$$

(iii)

$$\begin{aligned} 110110100 &= 1 \times 2^8 + 1 \times 2^7 + 0 + 1 \times 2^5 + 1 \times 2^4 + 0 + 1 \times 2^2 + 0 + 0 \\ &= 256 + 128 + 32 + 16 + 4 \\ &= 436. \end{aligned}$$

Example 2.10. Convert the numbers indicated to decimal (i) $(12121)_3$ (ii) $(4310)_5$ (iii) $(50)_7$ (iv) $(198)_{12}$.

Solution

(i)

$$(12121)_3 = 1 \times 3^4 + 2 \times 3^3 + 1 \times 3^2 + 2 \times 3^1 + 1 \times 3^0$$
$$= 81 + 54 + 9 + 6 + 1 = (151)_{10}$$

(ii)

$$(4310)_5 = 4 \times 5^3 + 3 \times 5^2 + 1 \times 5^1 + 0$$
$$= 500 + 75 + 5 = (580)_{10}$$

(iii)

$$(50)_7 = 5 \times 7^1 + 0 = (35)_{10}$$

(iv)

$$(198)_{12} = 1 \times 12^2 + 9 \times 12^1 + 8 \times 12^0$$
$$= 144 + 108 + 8 = (260)_{10}$$

Example 2.11. Convert the following decimal numbers to the specified bases (i) 7563 to octal (ii) 1928 to Hex (iii) 175 to binary.

Solution

(i)

8	7563	$= (16613)_8$
8	945 —	3
8	118 —	1
8	14 —	6
	1 —	6

(ii)

16	1928	$= (788)_{16}$
16	120 —	8
	7 —	8

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

(5) Add $(835)_{BCD}$ to $(695)_{BCD}$. Let the result be in BCD.

(6) Represent −82 in three different ways.

(7) Define with example the condition of overflow.

(8) Specify some commonly used binary codes for decimal numbers.

(9) What does the number 9AH represent in unsigned representation, signed—magnitude, signed 1's complement and signed 2's complement.

(10) Convert BCD 54H to ASCII.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3.3 Pin diagram of 8051

The pin diagram of 8051 and the assignment (function) of each pin is shown in Fig. 3.3. Some pins have dual assignments. The chip has 40 pins, and 64 functions. Therefore, 24 pins have dual functions.

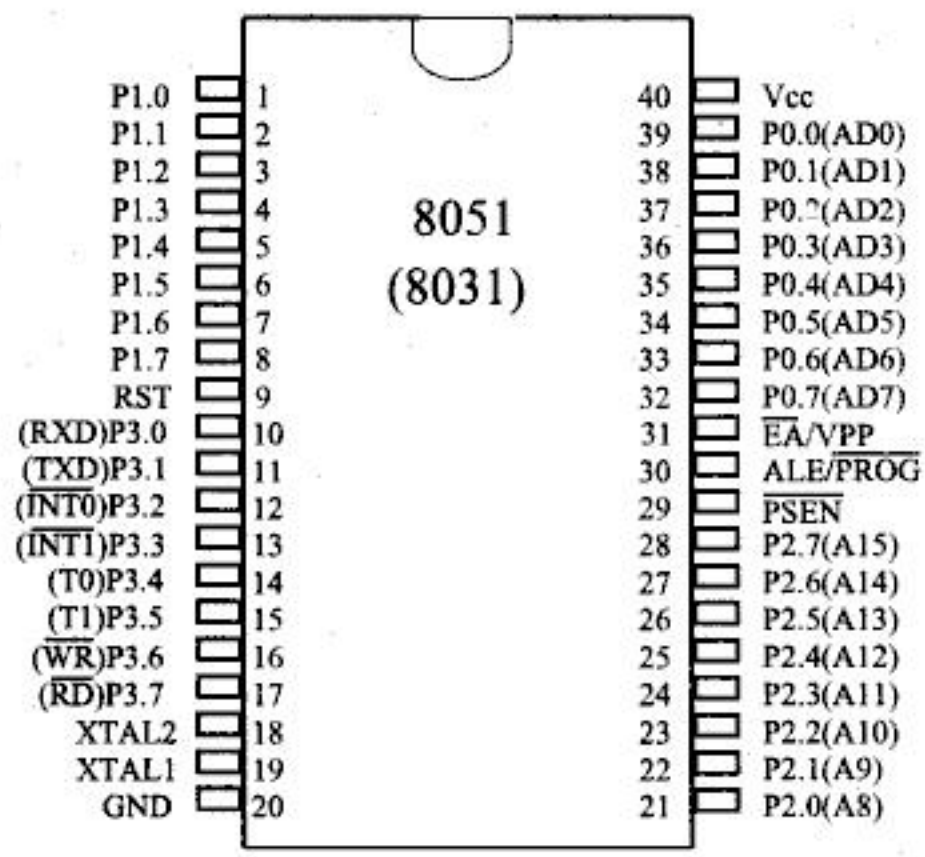


Figure 3.3. Pin diagram of 8051.

The program instructions or the physical connection at the pins determine their assignment and how they are used. It is to be noted that in 8051, the address bus is made of 16 lines. The lower-order byte of the address and the data bus share the same lines, designated $AD_0 - AD_7$. The higher order byte is carried on $A_8 - A_{15}$. The Address Latch Enable (ALE) signal determines if the lines $AD_0 - AD_7$ are used for the address or data. If ALE is high then the lines are used to transfer the lower order byte of address. If ALE is low, then the lines are used for data transfer.

A detailed discussion of all the pins, will be done at the end of the chapter, after the reader gets familiar with the hardware.

3.4 Clock and Machine cycle for 8051

All internal operations of the 8051 are synchronized by the clock pulses. In most microcontrollers there is built-in circuitry which allows a simple connection of a crystal or ceramic resonators or an external clock source. Some microcontrollers have an internal ring oscillator, which permits them to run without any external clock source. The typical quartz crystal with capacitors is shown in Fig. 3.4.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3.5.1 Program Counter (PC)

The program counter is a 16-bit register. The PC points to the address of the next instruction to be executed. The instructions (opcodes) are fetched from the memory location addressed by PC. After the CPU fetches the opcode, the PC is automatically incremented to point to the next instruction. In 8051, on chip ROM is available for addresses from 0000H to 0FFFH. If the memory exceeds 0FFFH, then the memory chip has to be externally interfaced with 8051. Since PC is 16-bits wide, 8051 can access programs from address 0000H to FFFFH, a total of 64 K bytes of code. The contents of the PC may be altered by certain instructions like CALL, RET etc. (more about this when we deal with instructions). The PC is the only register which does not have an internal address. By default the PC is set to 0000H on reset of the microcontroller. This means that it expects the opcode of the first instruction to be stored at ROM address 0000H. For this reason in the 8051 system, the first opcode must be burned into memory location 0000H of program ROM.

3.5.2 Data Pointer (DPTR)

The data pointer is a 16-bit register, made of two 8-bit registers called DPH (High) and DPL (Low). It is an index register that provides access to external memory. DPH and DPL hold the higher order byte and lower order byte of the address. In the instructions, DPTR can be specified as a 16-bit register, or it can be specified individually as an 8-bit register (DPL or DPH). The DPTR does not have a single internal address. DPH and DPL are assigned internal addresses separately (DPH-83; DPL-82).

3.5.3 A and B registers

These two 8-bit registers hold the result of many arithmetic and logical operations of the CPU. The Accumulator (A) is used in many operations, including addition, subtraction, integer multiplication, and division, and Boolean bit manipulations. It is a bit-addressable register, meaning that each bit of the accumulator can be accessed for reading or for altering. The A register is also used for all data transfers between 8051 and any external memory. The register B may be used as a location where data may be stored. It is also used with register A for multiplication and division operations and has no other function.

3.5.4 Program Status Word (PSW) register

The program status word register is an 8-bit register. It is also referred to as the *flag register*. It indicates certain conditions like carry, parity, sign etc.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

In instructions, the SFR's can be addressed by their functional names or by their addresses. The other SFR's like PCON, TCON, TMOD etc, will be dealt with in the relevant sections.

3.6 The 8051 internal memory

A computer/microprocessor/microcontroller must have memory to store program codes and also data. The 8051 has internal ROM and RAM memory. It has a Harvard architecture which uses the same address in different memory locations for code and data. It can be interfaced with additional external memory also. The internal circuitry accesses the correct memory segment, based on the instruction being executed.

3.6.1 Internal RAM

The 8051 has 128 bytes of internal RAM. This is shown in Fig. 3.7.

The RAM is divided into three segments:

- **Register banks:** There are four *register banks*, each with eight registers, which make up 32 working registers, with address from 00H to 1FH. The register banks are numbered 0 to 3. The selection of the register bank is done by the RS1 (PSW.4) and RS0 (PSW.3) bits. The eight registers are named R0 to R7. After selection of the bank, the particular register in the bank can be addressed by its name (R0 to R7) or by its address. Register banks which are not selected can be used as general purpose RAM. Register bank 0 is selected by default on reset.
- **Bit/byte addressable RAM:** The 16 bytes of RAM from address 20H to 2FH are also bit addressable. This means that each of their bits can be addressed individually. This forms a total of 128 ($16 \times 8 = 128$) addressable bits. Each bit has a unique address from 00H to 7FH. It is important to know the difference between the bit address and the byte address. A few examples are listed below (refer Fig. 3.7).
  - 17H is the bit address of the bit 7 in byte address 22H.
  - 30H is the bit address of the bit 0 in byte address 26H.
  - 69H is the bit address of the bit 1 in byte address 20H.

  Addressable bits are very useful in control of binary events (like switching on or off a switch). They save on precious RAM memory since using a byte instead of a bit, would be inefficient. The instructions specify whether to access the byte address or bit address.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

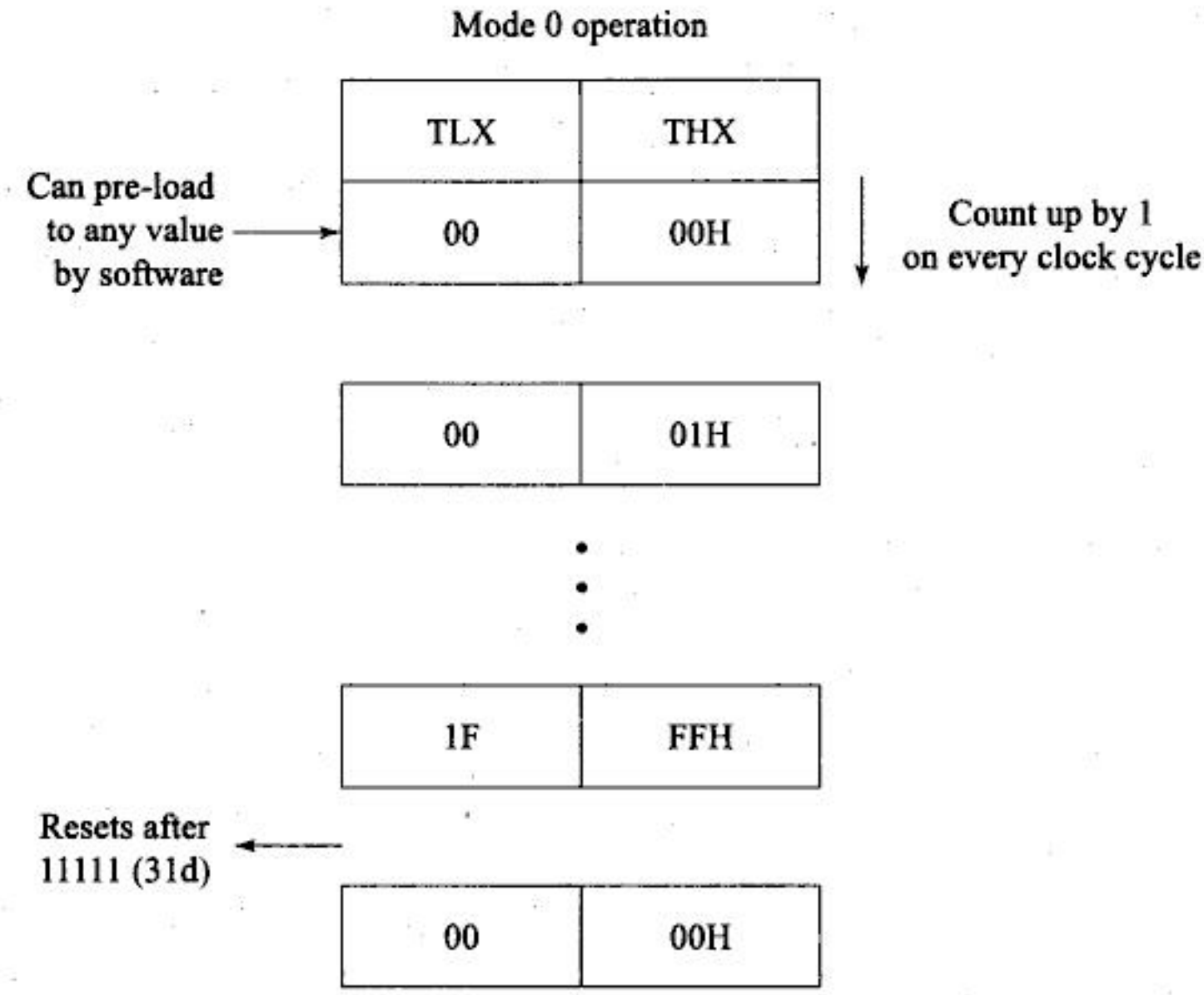


Figure 3.16. Mode 0 operation.

Example 3.18. What is value of TMOD to run Timer 0 in mode 1 with external control and Timer 1 in mode 2 as a counter?

Solution First set TR0 = 1; TR1 = 1. This initializes both timers to ON state.

$$\text{TMOD} = 09\text{H} + 60\text{H} = 69\text{H}.$$

3.8.5 Timer modes

Mode 0: In mode 0, the timers are used as 13-bit timers. In recent development, 8051 is not used in this mode. In 13-bit mode TLX (X = 0 or 1 representing Timer 0 or Timer 1 respectively), is incremented from 00000000 (0d) to 00011111 (31d). When incremented from 31d, it will reset to "0". Thus, only 13-bits of the 16-bits are used—bits 0–4 of TLX and bits 0–7 of THX. This means that the timer can count $2^{13} = 8192$ values. If we initialize it to 0, then it will reset after 8192 machine cycles. This is shown in Fig. 3.16. In all modes of operation, when the Timers reset, a flag is set in the TCON register (to be discussed in next section)

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

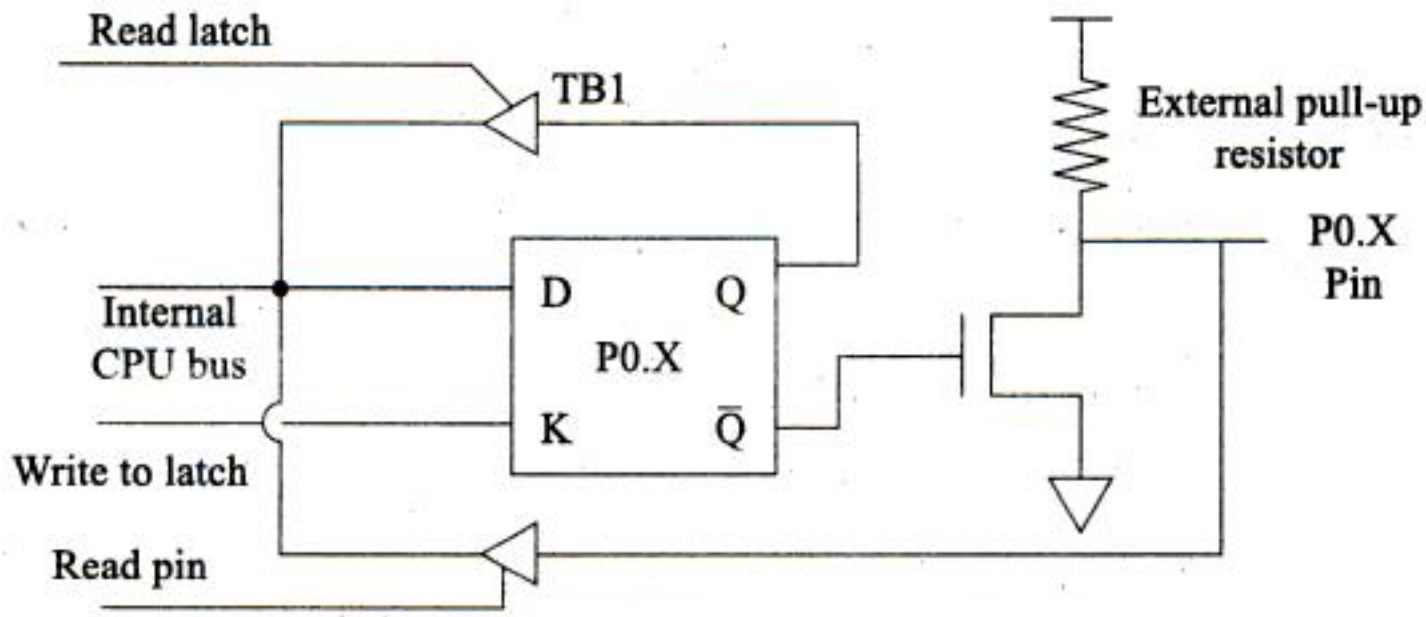


Figure 3.18. Port 0 connection.

any signal connected to the input pin and the input signal is therefore directed to the tri-state buffer TB1. So when we read the port, we are in essence reading data present at the pin.

The pull-up resistors will supply a logic high when P0 is used as an output port. Each I/O line can be independently used as an input or an output. It cannot be used as a general purpose I/O when being used as address/data bus.

3.9.2 Port 1 (P1: Address 90H)

This is an 8-bit input/output port. It is also bit-addressable. The addresses are shown in Table 3.8.

Table 3.8 Port 1 bits.

Hex byte address	Bit address								Symbol
90H	97	96	95	94	93	92	91	90	P1
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1.X

This port does not need any external pull-up resistors since they are available internally. Like P0, to configure P1 as an input port, a 1 must be written to all the bits required to act as inputs. Figure 3.19 shows port 1 connections.

3.9.3 Port 2 (P2; Address A0H)

This is an input/output port. It is also bit addressable. It has built-in pull up resistors.. Like in Port 1, it must first be programmed by writing 1 to all bits required to be an input. The bit addresses are given in Table 3.9.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

4.2.2 Steps to create an ALP

The steps in creating an ALP are as shown in Fig. 4.1.

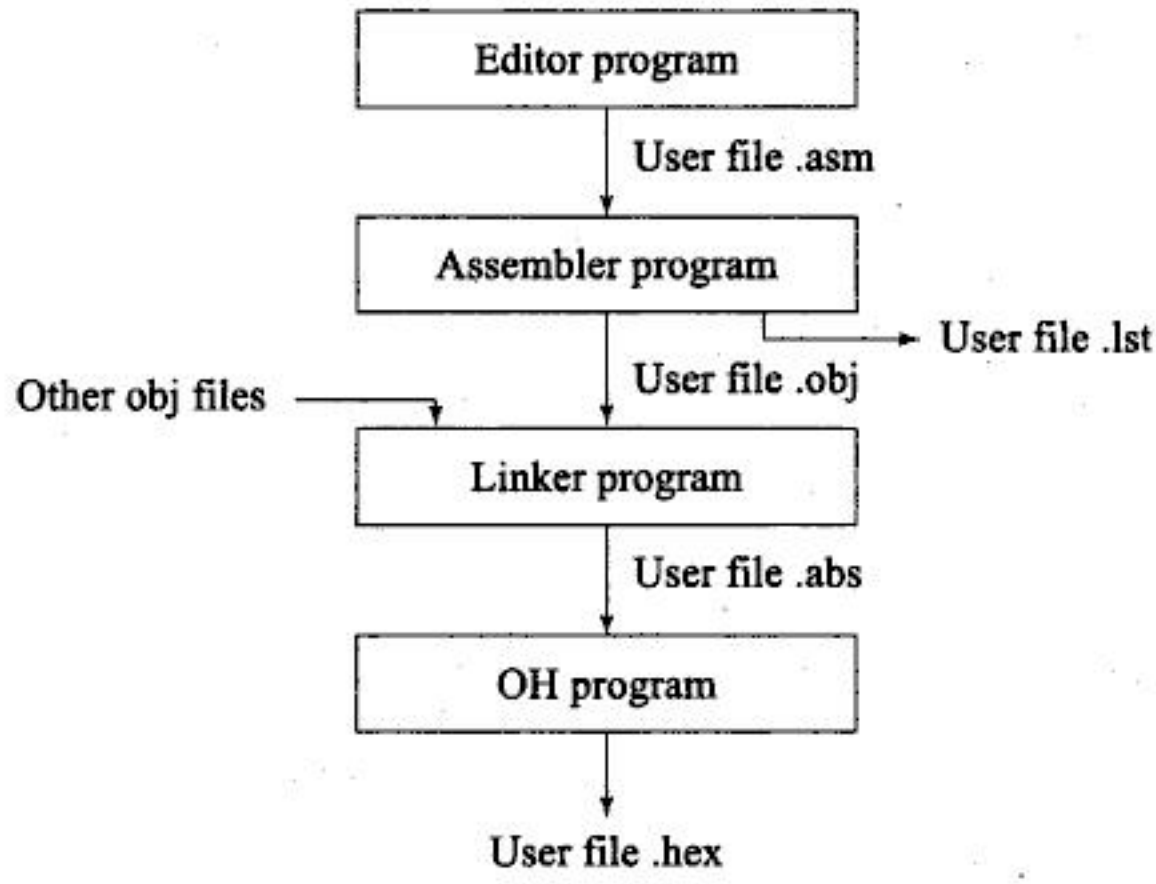


Figure 4.1. Steps to create an ALP.

Step 1: We first use a text editor to type the ALP. Here, we assume that the programming is done via a personal computer. Common text editors are MS-DOS EDIT, NOTEPAD, MS-WORD etc, which are used to create and edit files. The editor, must be able to produce an ASCII file. Standard assemblers use an extension of '.asm' or '.src', for the source file. For example we can have a source file "program.asm" or "program.src".

Step 2: The ".asm" file created in step 1, which is the program code, is fed to the 8051 assembler. As mentioned earlier, the assembler converts the instructions into machine code. The assembler produces a file, called the object file and another file called the list file. These files have an extension of ".obj" (program.obj) and ".lst" (program.lst) respectively.

The list (.lst) file, is useful to the programmer (it is optional). It lists all the opcodes and addresses, as well as errors that are detected by the assembler. This file can be accessed by the text editor, displayed on the monitor or printed to obtain a hard copy. They are useful in finding the syntax errors. After these errors are corrected, the ".obj" file is sent as input to the linker program.

Step 3: The link program takes one or more object files and produces an absolute object file with an extension ".abs".

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

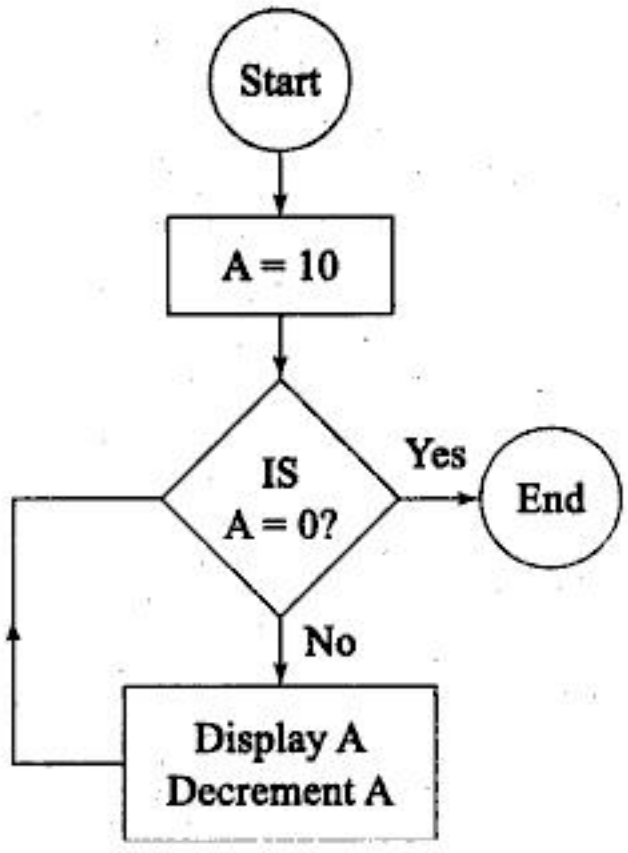


Figure 4.2. Flow chart.

3. If $A = 0$, end the program
   Else display A
4. Decrement A
5. Go to step 2.

It can be seen that both flow charts and algorithms, help in understanding the sequential steps of the problem, the decisions involved, the flow of data etc. Since, they are independent of the language (the same flow chart can be used to code in any assembly language or high-level language), they are portable, convey the logic of the problem and act as a guide to code the problem in any suitable language.

4.4 8051 data types and directives

The 8051 microcontroller has only one data type: an 8-bit binary data. The size of each register is 8-bits. If data of larger sizes are to be handled, they have to be broken into 8-bit data which can be processed by the CPU. Signed or unsigned arithmetic can be used.

4.4.1 Directives

An assembler is a program, to translate the ALP to machine language. Like any other program, it has its own programming language and syntax, which

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
6.  MOV   DPH, #23H      ;
7.  MOV   P1, #55H       ;
8.  MOV   TMOD, #10H     ;
```

Now consider

```
MOV   DPTR, #2FFF0 H
```

This would produce an error because the value is larger than 16-bits. We can use EQU directive to access data immediately as follows:

```
TIME  EQU 40
:
MOV   TL1, #TIME;TL1 = 28H (hex of 40)
```

4.5.2 Register addressing mode

This involves the use of registers to hold the data to be manipulated. Both the source and destination are registers and their sizes should match. The general format is

```
MOV   Ra, A
MOV   A, Rb
```

where Ra and Rb are restricted to A, DPTR (DPL and DPH) and R0–R7 (of the selected bank). These registers are accessible by name. Other registers in 8051 may be addressed in the direct addressing mode. Here are some examples:

```
1.  MOV   A, R1      ;Move contents of register R1 into A
2.  MOV   R3, A      ;Move contents of A into R3
3.  MOV   R7, DPH    ;Move contents of R7 to DPH
4.  MOV   DPL, R2    ;Move contents of R2 to DPL
```

- We can move data between accumulator and Rn ($n = 0$ to 7) but movement between Rn registers is not allowed.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

(ii) Lets use register addressing with accumulator.

```
MOV   A, #22H  ;
MOV   R0, A    ;
MOV   R1, A    ;
MOV   R2, A    ;
```

The first instruction is two bytes and the remaining ones are one byte. We have 5 bytes, 4 lines.

(iii) Lets use direct addressing.

```
MOV   R0, #22H   ;
MOV   01H, 00H   ; } Move contents of location 00H
MOV   02H, 00H   ; } (address of R0) to 01H (R1)
                     and 02H (R2)
```

The first instruction is two bytes. The direct address instruction is three bytes

```
(opcode + first address + second address).
```

We therefore have 8 bytes, 3 lines.

4.5.4 Indirect addressing mode

The indirect addressing mode uses a register to hold the actual address which contains the required data. In other words, the contents of the register is a pointer to the data address. If the data is inside the CPU then, only registers R0 and R1 are used as pointers. R2–R7, cannot be used. When R0 and R1 are used as pointers, they must be prefixed with '@' symbol. Consider

```
MOV   A, @R0  ;Move the contents of RAM location whose address
              ;is in R0 to A
```

Example 4.5. What does the following code segment do?

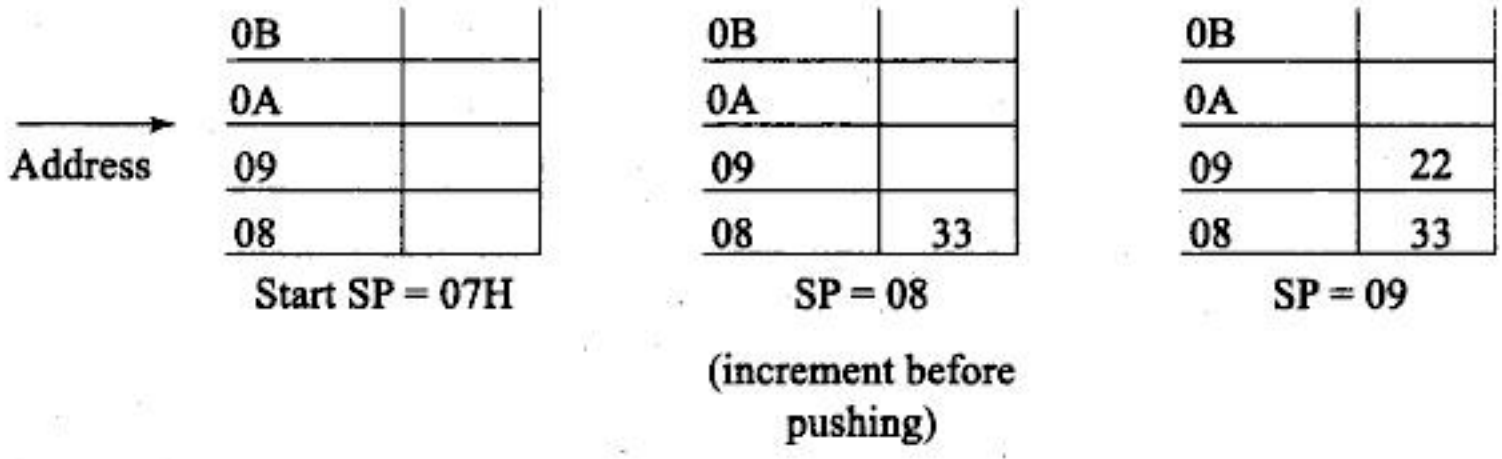
```
MOV   40H, #55H ;
MOV   R1, #40H  ;
MOV   A, @R1    ;
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Solution SP is initialized at 07H. First two instructions load 33H and 22H into R3 and R2 respectively. Stack is changed as below.



4.6.2 POP instruction

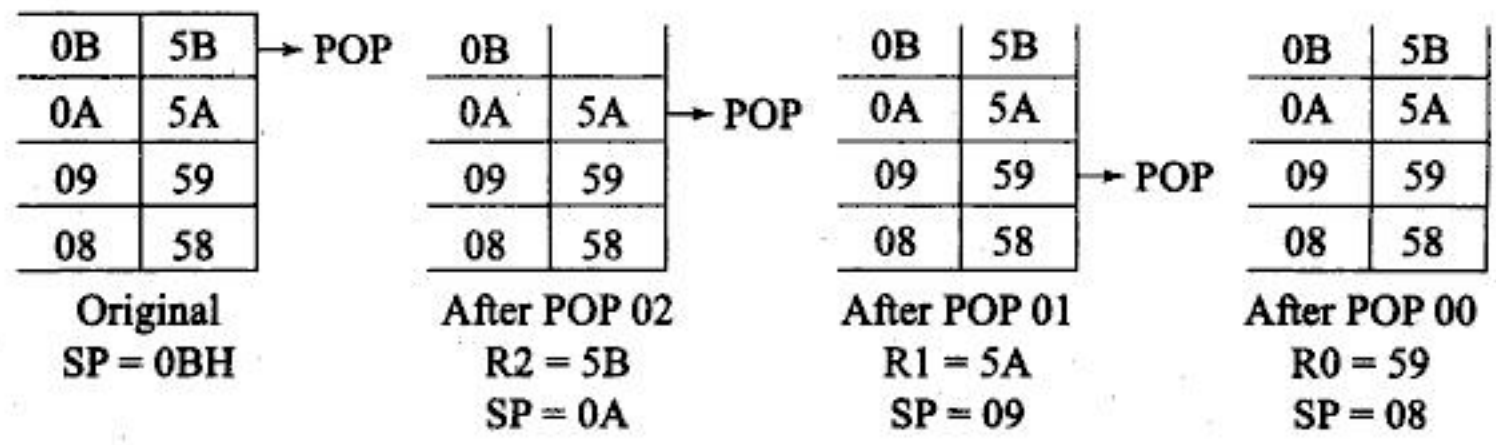
A POP operation copies data from the stack to the destination address. The data is copied and then SP is decremented.

- POP data to destination
- Decrement SP.

Example 4.8. Examine contents of stack and show how data is popped and the contents of SP.

```
POP  02H
POP  01H
POP  00H
```

Solution



Note

- Stack starts at location 07H by default.
- We can use 08H to 1FH (24 bytes) in RAM for stack. Locations 20–2FH should not be used since they are bit-addressable.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Note the following:

- The status of carry flag indicates a result greater than 8 bits. This status is important when we are using unsigned numbers.
- The overflow flag indicates that the range of signed number representation has been exceeded. (Even though result may be 8-bits.) The status of this flag is important when dealing with signed numbers.

5.3 Incrementing and Decrementing

Two instructions are available for incrementing and decrementing the contents of a register or a memory location by 1.

The complete list of Increment and Decrement operations are given below.

Mnemonic		Operation
INC	A	;Add 1 to accumulator
INC	Rn	;Add 1 to Register Rn
INC	@Rp	;Add 1 to location whose address is in Rp
INC	*addr*	;Add 1 to contents of direct address '*addr*'
INC	DPTR	;Add 1 to the 16-bit contents of DPTR
DEC	A	;Subtract 1 from A
DEC	Rn	;Subtract 1 from Register Rn
DEC	@Rp	;Subtract 1 from location whose address is the Rp
DEC	*addr*	;Subtract 1 frp, contents of direct address '*addr*'
DEC	DPTR	;X does not exist

Note

- INC and DEC instructions do not affect Math flags.
- If the contents are FFH and it is incremented it overflows to 00H.
- If the contents are 00H and it is decremented it underflows to FFH.
- DPTR overflows flow 0FFFFH to 0000H.
- If the direct address '*addr*' is a port address the latch of the port is altered.

Example 5.4. Consider a series of operations listed below and what they do.

```
1   MOV   A, #23H          ;A = 23 H
2   DEC   A                ;A = 22 H
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

5.9.2 OR operation

Here two words are ORed. The corresponding bits of the two words are ORed. The instructions are as follows:

Mnemonic	Operation
ORL A, #n	OR contents of A with immediate byte #n. Store result in A.
ORL A, @Rp	OR contents of A with contents of direct address stored in Rp. Store result in A.
ORL A, *addr*	OR contents of A with contents of direct address *addr*. Store result in A.
ORL A, Rn	OR contents of A with contents of register Rn. Store result in A.
ORL *addr*, A	OR contents of A and contents of direct address *addr*. Store result in *addr*.
ORL *addr*, #n	OR contents of direct address *addr* and byte #n. Store result in *addr*.

5.9.3 Exclusive OR operation

Here two words are exclusively ORed which means the corresponding bit positions of the two words are exclusively ORed. The instructions are:

Mnemonic	Operation
XRL A, #n	Exclusive OR contents of A with immediate byte #n. Store result in A.
XRL A, @Rp	EXOR contents of A and contents of direct address in Rp. Store result in A.
XRL A, *addr*	EXOR contents of A and contents of direct address *addr*. Store result in A.
XRL A, Rn	EXOR contents of A and contents of Rn. Store result in A.
XRL *addr*, A	EXOR contents of A and contents of direct address *addr*. Store result in *addr*
XRL *addr*, #n	EXOR contents of direct address *addr* with immediate byte #n. Store result in *addr*.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Solution

(i) The first instruction ANDs contents of *port P0 latch* with 0FH, since *P0 is* here the *destination*. Therefore we have P0 AND 0F, which turns off the upper nibble of P0, turning those transistors off. The result is stored in latch of P0.

(ii) Here P0 is the *source*. Hence the *pin data* is read. Since they are all at ground level, the pin data will be 00H. Hence

```
ANL  A, P0
```

ANDs accumulator with 00H. Hence A = 00H.

5.10 CLEAR and COMPLEMENT accumulator

Two instructions are exclusively used with the accumulator.

5.10.1 Clear accumulator

The contents of the accumulator are cleared. Each bit is made 0.

Mnemonic	Operation
CLR A	Clear each bit of A to 0.

This is equivalent to MOV A,#00H. This is a two byte instruction whereas CLR A is a one byte instruction.

5.10.2 Complement accumulator

Every bit of accumulator is complemented. i.e. a 0 is made 1 and a 1 is made 0.

Mnemonic	Operation
CPL A	Complement every bit of accumulator.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

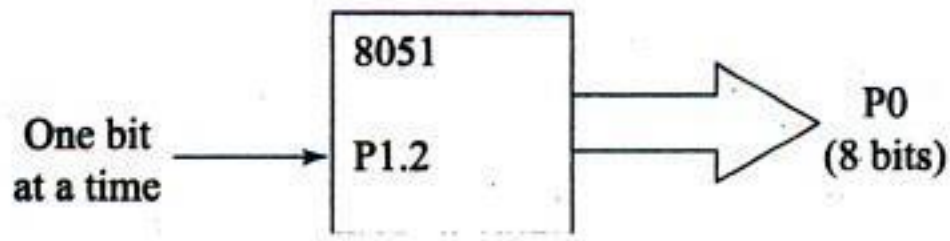
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Solution The figure below explains the task on hand.



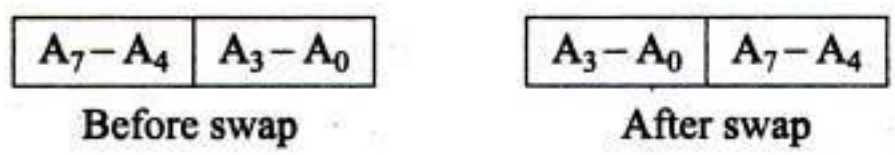
The program is explained below.

Mnemonic			Operation
	MOV	R0, #08	;Move the count to R0.
	SETB	P1.2	;Recollect that a port pin must be made 1 to act as input pin. P1·2 is made to act as input pin.
Here:	MOV	C, P1.2	;Data is moved from P1·2 to CY
	RRC	A	;CY goes into A
	DJNZ	R0, Here	;Repeat the process till R0 is zero (8 times). At the end the entire byte is in A.
	MOV	P0, A	
	End		

5.13 Swap operation

The swap instruction works only on the accumulator.

Mnemonic	Operation
SWAP A	;It swaps the lower nibble and upper nibble of A.



Example 5.29. What are the contents of the accumulator on execution of the following code segment?

```
MOV   A, #3BH
SWAP  A
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

- No flags are affected unless the bit b in the instruction JBC is a flag bit in PSW.
- In JBC if the bit addressable bit b is a port bit, the port latch is read, tested and altered (Remember, read, modify and write?)

Example 6.1. Consider the following code segment:

```
        MOV   A,#20H    ;
        MOV   R0, A     ;
HERE:   ADD   A, R0     ;
        JNC   HERE      ;
        MOV   A,#FFH    ;
END
```

How is the program executed? After how many loops does the program quit the loop HERE?

Solution

- We first move 20H to A
- Then 20H is moved to R0

HERE → • 20H + 20H = 40H; CY = 0

So JNC is true (since C = 0). Therefore problems loops to HERE. Now it proceeds as follows

(i) 40H + 20H = 60H; CY = 0 so loop

(ii) 60H + 20H = 80H; CY = 0 so loop

(iii) 80H + 20H = A0H; CY = 0 so loop

(iv) A0H + 20H = C0H; CY = 0 so loop

(v) C0H + 20H = E0H; CY = 0 so loop

(vi) E0H + 20H = 00H; CY = 1 quit loop

So after looping 6 times, the carry flag is set. Therefore the condition "JNC" is false and the program quits the loop. Then the next instruction MOV A,#FFH is executed.

Example 6.2. Can the program of example 6.1 be executed in a different way?

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Solution

```
        MOV     P0, #0FFH       ;Configure P0 as input port
        MOV     A, P0           ;Move contents to A
        CJNE    A, #60H, NEXT   ;If A ≠ 60H, jump
        SJMP    EXIT            ;
NEXT:   JNC     OVER            ;If A > 60H, CY = 0
        MOV     R1, A           ;If A < 60H, move to R1
        SJMP    EXIT            ;
OVER:   MOV     R0, A           ;If A > 60H, move to R0
EXIT:   END
```

Example 6.6. RAM locations 50H–59H contain 10 numbers, one of which is 35H. Load the address of the number 35H into R3. All marks are less than or equal to 25.

Solution

```
        MOV     R0, #50H        ;Load first address to R0
        MOV     R2, #0AH        ;Load count into R2
        MOV     A, #35H         ;Load A = 35H, the number to be
                                ;searched for
HERE:   CJNE    A, @R0, NEXT    ;Compare RAM data with 35H
        MOV     R3, R0          ;If they are equal copy address
                                ;which is in R0 to R3
        SJMP    EXIT            ;
NEXT:   INC     R0              ;
        DJNZ    R2, HERE        ;
EXIT:   ......
```

Example 6.7. Read the contents of P1 and check if it is equal to 50H. If it is, send FFH to P2; otherwise P2 is cleared.

Solution

```
        MOV     P2, #00H    ;Clear P2
        MOV     P1, #0FFH   ;Make P1 an input port
        MOV     R2, #50H    ;R2 = 50H
        MOV     A, P1       ;Read P1
        XRL     A, R2       ;EXOR A with 50H. If they are equal,
                            ;A = 00H after EXOR
        JNZ     EXIT        ;If A ≠ 00H, EXIT
        MOV     P2, 0FFH    ;If A = 00H, then P2 = FFH
EXIT:   ......
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

5. The RET instruction at the end of the subroutine, pops the return address to the PC.

When a hardware interrupt takes place, the "Interrupt Disable flip-flops" are set, to prevent another interrupt with same priority level from taking place, until an interrupt return instruction has been executed in the interrupt subroutine. The mnemonic is "RETI". There is not much difference between RET and RETI, except for the enabling of the interrupt logic. RET is used at the end of subroutines called by the LCALL or ACALL instructions. RETI is used by subroutines called by an interrupt. Use of RETI at the end of a software called subroutine may enable the interrupt logic erroneously.

A simple program is given below to illustrate concept of subroutine.

Example 6.11. Write a program to toggle all the bits of port 1, with a time delay between toggling.

Solution Lets consider the numbers 55H and AAH

$$\left.\begin{array}{l} 55H \rightarrow 01010101 \\ AAH \rightarrow 10101010 \end{array}\right\} \text{bits are toggled.}$$

The delay is written as a subroutine.

```
        ORG    00          ;
        MOV    A, #55H     ;
HERE:   MOV    P1, A       ;
        ACALL  DELAY       ;
        CPL    A           ;
        SJMP   HERE        ;
```

Delay subroutine

```
DELAY:  MOV    R5, #0FFH   ;
LOOP:   DJNZ   R5, LOOP    ;
        RET
        END
```

Many examples are presented in the next few chapters where subroutines are used.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

7.1 Introduction

Chapters 4 to 6 presented the Assembly language instructions of 8051. There are many advantages of programming in assembly language. However, programming with C has become popular because of the ease of programming as compared to ALP. In this chapter we will see the advantages & disadvantages of using C. The data types and programming concepts of 8051C are presented in detail with a number of examples.

7.1.1 Advantages of programming in 'C' for microcontrollers

1. Programming in assembly language is tedious and time consuming, while 'C' programming is less time consuming and easier to write.
2. 'C' code is portable to other microcontrollers; that is the same 'C' program (with little or no modifications) using the corresponding C cross-compiler can be loaded into a different microcontroller.
3. C programs are easier to modify and update
4. Code available in function libraries (such as sine, sqrt, etc) can be used while writing the 'C' programs.

7.1.2 Disadvantages of Programming in 'C'

1. Assembly language programs have fixed size for the hex files produced whereas for the same 'C' programs, different 'C' compilers produce different hex code sizes. Hence when we want to calculate exact time delay for 'C' programs, it is difficult.
2. Microcontrollers have limited on-chip ROM and the code space (to store program codes) is also limited (64K bytes in 8051). A misuse of data types by the programmer in writing 'C' programs, for example using 'int' data type instead of 'unsigned char' can lead to a large size hex file, (which some times may not fit in the limited code space). The same problem does not arise in assembly language program.
3. The general purpose registers of the 8051, such as R0–R7, A and B are under the control of the C compiler and are not accessed by C statements.

7.2 Declaring variables

In C it is necessary to inform the compiler about all the variables which are available before using them. This enables the compiler to know what type of

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The 'C' program generally has a list of global variable declarations after the #include <> line, followed by a list of subroutines used in the program. Then we have the main() function with local variable declarations and the main body of the program within the flower brackets.

Generally microcontrollers once powered up (reset) keep executing one machine code (instruction) after the other starting from the location 00H in ROM space (*code space*). Towards this end, most of the 8051 programs are designed to be executing indefinitely. If the task is finite, for example finding the largest element in an array, etc, then after the entire program is coded, the last line in assembly language program is coded as

```
HERE:   SJMP    HERE
```

This statement will make the 8051 to keep jumping to the same location and not execute any further codes in the code space. Suppose this statement is not coded, then the 8051 executes the next opcode (machine code) pointed by program counter (which gets automatically incremented for every instruction). The assembler would have coded only the required hex code, but after the finite task is over, and as the above SJMP statement is not there and the ROM space may contain unspecified codes, the 8051's execution is indeterminate. This situation has to be avoided.

In 'C' program this is implemented by using the statement

```
while(1);
```

Since '1' is true, the while(1) loop will never terminate and is same as HERE:SJMP HERE in assembly language.

If the algorithm consisting of say, do task 1, task 2, task 3, etc and repeat from task 1, then in assembly language program (ALP) we have a SJMP to task 1. In 'C' program, the task 1, task 2, task 3, etc which have to be repeated are enclosed in the while(1) loop. (*NOTE: no semicolon after while(1)*, instead open a flower bracket and write the repeated tasks to be performed.)

```
main()
{... declaration, initialization parts,
    ... non repeated tasks, etc
    while(1)                    //write tasks to be repeated in the while loop
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
    while(1);                    //wait here
}
```

RESULT: 30H, 31H, 41H & 42H (ASCII values of 0, 1, A & B respectively) are displayed at port P1.

Example 7.4. Write a program to toggle the pin P2.4, 300 times.

Solution Since the count 300 is greater than 256(FFH), the variable used for counter say 'n' is defined as unsigned int n; [instead of unsigned char]. Also we may use a variable name for the pin P2.4. Since P2 is a special function register (sfr), its bits are defined using the 'sbit' data type. The 'C' program for example 7.4 is given below

```
#include <reg51.h>
sbit OPIN=P2^4;                  //P2^4 implies D4 bit of port P2
void main()
{unsigned int n;
    for (n=0; n<300; n++)        //repeat for 300 times
    {OPIN=0;
    OPIN=1;
    }
    while(1);                    //wait here
}
```

NOTE: The above example illustrates the access of I/O pins in 8051. It is an example of bit addressable I/O programming.
The next program is an example of signed numbers.

Example 7.5. Write a 8051 C program to send values of −3, 2, −1, 4, −2 to port P1.

Solution As shown in Example 7.3, concatenate all the required values into an array and use for loop to send to port P1. The 'C' program is shown below

```
sfr P1=0×81;                     //define P1 since #include <reg51.h>
                                 //is not used
void main()
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Solution Let the initial value on P0 & P1 be 043H and 55H respectively. The toggled values are BCH and AAH respectively (refer example 7.2).

Algorithm:

1. Send initial values on ports P0 and P1
2. Generate a time delay of 1ms
3. Send the toggled values to the ports
4. Generate the time delay of 1ms
5. Repeat from step 1.

C Program

```
#include <reg51.h>
void main()
{unsigned int i;                    //i is a count >255, hence integer is used
    while(1)                        //repeat continuously
{P0=0×43;                           //send initial values to the port
    P1=0×55;
    for (i=0; i<1275; i++);         //delay of 1ms
    P0=0×BC;                        //toggled values
    P1=0×AA;
    for (i=0; i<1275; i++);         //delay of 1ms
}                                   //end of while
}                                   //end of main.
```

Example 7.8. Write a program to generate a square wave of 250ms on-time, 50% duty cycle on pin P1.4.

Solution Making a pin low and high continuously generates a square wave 50% duty cycle implies the on-time = off time = 250ms. The square wave generated is shown in Fig. 7.1.

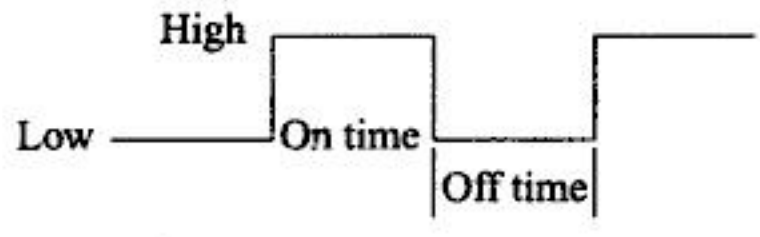


Figure 7.1. Square wave with 50% duty cycle.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
    }                                    //end of main
void delay (unsigned int count)
{unsigned int i,j;
        for (i=0;i<count;i++)
        for (j=0;j<1275;j++);  //1ms delay from inner loop
}
```

Example 7.16. Write an 8051 'C' program to send the message "THE EARTH IS BEAUTIFUL" to an LCD connected to the 8051. The data pins of the LCD are connected to P1 and the data at these pins is latched into the LCD when its enable pin (connected at P2.0) goes from high to low (i.e., a negative edge at P2.0).

Solution

(1) Declare an array or string to hold the message. Say use unsigned char message[] = "THE EARTH IS BEAUTIFUL".

NOTE: Now the element message [0] contains 54H, the ASCII value of 'T'. Similarly message [1] contains 48H, the ASCII value of "H". The last element in the array message [22] contains 4CH ("L"). The ASCII value of spaces (20H) are also contained in the message [] array.

2. The characters (ASCII values) in message [] array are sent one after other to P1 port.
3. Each time a character is sent to P1, the enable pin P2.0 is made high and low.
4. Steps 2 and 3 are repeated till the end of the array. (Here 22 characters are sent.)

C Program for example 7.16

```
#include <reg51.h>
void main()
{unsigned char message[] = "THE EARTH IS BEAUTIFUL";
    unsigned char i;                    //only 22 characters to send
    for(i=0;i<22;i++)                   //note: 22 (not 0×22) is used
        {P1 = message [i];              //send character to port P1
        P2^0=1;                         //make enable pin high to low to
                                        //latch data
        P2^0=0;
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Table 7.3 Bit logic operators.

Symbol	Bit-wise operator	Example	Function	Uses
&	AND	A&B	Performs bit wise AND function	To mask certain bits
|	OR	A|B	Performs bit wise OR function	To set certain bits high
^	EXOR	A^B	Performs bit wise EXOR function	To clear a value, to toggle
~	Inverter	Y=~B	Performs bit wise NOT function	To toggle the pin status

Apart from the bit wise logic operators, there are two bit wise shift operators in C which are (i) shift right (≫) and (ii) shift left (≪).

Their format is A ≫ B; where A is the data (which has to be shifted) and B is the number of bits to be shifted right.

Similarly the format for left shift is A ≪ B where A is the data and B is the number of bits to be shifted left.

NOTE: SHIFT right by 1-bit is equal to division by 2 & left SHIFT by 1-bit is equal to multiplication by 2.

Example 7.21. Explain the result of the following 'C' statements.

(a) P0 = 0 × 35 & 0 × 0F;

Solution:

It is a bit wise AND:

$$\begin{array}{rcl} 0\times35 & = & 0011\ 0101\ \text{B} \\ \underline{0\times0F} & = & \underline{0000\ 1111\ \text{B}} \\ 0\times05 & & 0000\ 0101\ \text{B} \end{array}$$ } AND operation

The contents of P0 are

$\boxed{\text{P0} = 0\times05}$

(b) P1 = 0 × 04 | 0 × 68;

Solution:

It is a bit wise OR:

$$\begin{array}{rcl} 0\times04 & = & 0000\ 0100 \\ \underline{0\times68} & = & \underline{0110\ 1000} \\ 0\times6C & & 0110\ 1100 \end{array}$$ } OR operation

and $\boxed{\text{P1} = 0\times6C}$ or $\boxed{\text{P1} = 6\text{CH}}$

(c) P2 = 0 × 54 ^ 0 × 78;

Solution:

It is a bit wise EXOR:

$$\begin{array}{rcl} 0\times54 & = & 0101\ 0100 \\ \underline{0\times78} & = & \underline{0111\ 1000} \\ 0\times2C & & 0010\ 1100 \end{array}$$ } Bit-wise EXOR

and $\boxed{\text{P2} = 2\text{CH}}$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
        temp2=temp2 & 0×D7;     //mask D3 & D5 bits of temp2
        while(1);
    }
```

Run the above program in Keil simulator and observe the values of P0, temp1 & temp2 in the watch window.

Example 7.25. Write an 8051 C program to read the P1.0 & P1.1 bits and send the ASCII characters of '0', '1', '2' and '3' to P0 for the combinations 00, 01, 10 and 11 of P1.1 & P1.0 bits.

Solution Since only D_0 & D_1 bits of P1 are needed, mask the other bits D_2 to D_7. Hence the mask value is 0000 0011 = 03H.

Algorithm:

1. Make P1 as input port (send FFH to P1)
2. Read P1 value
3. Mask all bits except D_0 & D_1 of P1 and put the masked value in x
4. If $x = 0$; send '0' to P0, else if $x = 1$; send '1' to P0; elseif $x = 2$; send '2' to P0; else send '3' to P0
5. Repeat from Step 2.

C program

```
#include <reg51.h>
void main()
{unsigned char i;
    P1=0×FF;                  //make P1 as input port
    while(1)                  //repeat forever
    {i=P1 & 0×3;              //mask all port pins except P1.1 & P1.0
        if(i==0) P0='0';      //0 in single quotes sends ASCII value to P0
    elseif (i=1) P0='1';
    elseif (i=2) P0='2';
    else P0='3';
    }                         //end of while
}                             //end of main.
```

NOTE: This program can also be written with a switch & case statements. As seen from examples 7.23, 24 & 25, all the logic bit operators have special uses (such as masking, toggling, etc). Similarly the shift operators shift left and shift right are extensively used in data serialization (that is transferring a

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
    SW=1;                   //make P2.0 as an input pin
    ACC=0×44;               //load accumulator with data to be shifted
                            //i.e., 44H
    if(SW==0)               //LSB first
        {for(x=0;x<8;x++)
        {P1^0=ACC^0;        //LSB of accumulator on port pin for
                            //serial transfer
        ACC=ACC≫1;          //shift right by 1-bit position
        }                   //end of for
        }                   //end of if
    else                    //this point is reached when SW=1
        {for(x=0;x<8;x++)
            {P1^0=ACC^7;    //MSB bit sent first
            ACC=ACC≪1;      //shift left once for next bit
            }               //end of for
        }                   //end of else
    while(1);               //wait here
}                           //end of main.
```

Example 7.29. Write a 'C' program to bring in a byte of data serially one bit at a time via P1.0. The LSB should come in first.

Solution The serial data being received has LSB first, so feed it at D_7 of accumulator. So when the next bit is received, shift the LSB received in accumulator by one right shift & enter the new bit again at D_7 position of accumulator. Repeat this for all 8 data bits received. This is illustrated below.

No. of right shifts of accumulator	Received data bit on P1.0	Accumulator contents (ACC.7 = P1.0 after shifting)
0	D_0 (LSB first)	D_0 × × × × × × × → Initially D_0 can be filled in at D_7
1	D_1	D_1 D_0 × × × × × ×
2	D_2	D_2 D_1 D_0 × × × × ×
3	D_3	D_3 D_2 D_1 D_0 × × × ×
4	D_4	D_4 D_3 D_2 D_1 D_0 × × ×
5	D_5	D_5 D_4 D_3 D_2 D_1 D_0 × ×
6	D_6	D_6 D_5 D_4 D_3 D_2 D_1 D_0 ×
7	D_7	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

NOTE: Shift contents of accumulator first and then load the received data bit at D7 of ACC

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
void main()
    {ACC=ASCII;              //get data into accumulator
    ACC=ACC & 0×0F;          //mask with 0FH
    P0=ACC;                  //display unpacked BCD
    while(1);
    }
```

ALP

```
START:   MOV    A, 32H     ;ASCII data in 32H location is
                           ;moved to A
         ANL    A, 0FH     ;mask upper nibble
         MOV    P0, A      ;display unpacked BCD at P0
HERE:    SJMP   HERE
```

Example 7.33. Write an 8051 'C' and assembly language program (ALP) to convert packed BCD number to ASCII and display the bytes on P1 and P2.

Solution The packed BCD 29 is converted to unpacked BCD digits 02 and 09 and each unpacked BCD number is converted to ASCII by adding 30H.

Algorithm:

1. Obtain the unit digit of packed BCD number by masking with 0FH
2. OR with 30H to get the ASCII value and display on P1
3. Obtain the tens digit of packed BCD number by masking with F0H and shift the masked value by 4-bit positions to right (to bring it to units place & hence the unpacked BCD form, 02 for the above example)
4. OR the shifted value with 30H to get ASCII representation of the tens digit and display at P2.

Result: P2 = 32H & P1 = 39H, (which are ASCII values of 2 & 9).

C Program

```
#include <reg51.h>
void main()
{unsigned char x;
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
        MOV    P1, A
        MOV    A, R4      ;get units digit
        MOV    P0, A
HERE:   SJMP   HERE
```

Example 7.36. Write a 8051 C program and ALP to convert a hexadecimal number FDH to ASCII numbers after converting it to BCD.

Solution The conversion of hexadecimal number to decimal is the same as above after the conversion to decimal is over, to convert to ASCII just OR it with 30H.

C Program

```
#include <reg51.h>
void main()
{unsigned char temp1, units, tens, hundreds;
    unsigned char hexnum=0×FD;
    temp1=hexnum/10;
    units=hexnum %10;              //remainder
    hundreds=temp1/10;             //quotient
    tens=temp1 %10;                //remainder
    P0=units|0×30;                 //ASCII value of units
    P1=tens|0×30;                  //ASCII value of tens
    P2=hundreds|0×30;              //ASCII value of hundreds
while(1);
}                                  //end of main
```

ALP:

```
START:  MOV    B, #10
        MOV    R0, #FDH
        MOV    A, R0
        DIV    AB
        MOV    R4, B      ;remainder is units
        MOV    B, #10
        DIV    AB
        ORL    A, #30H    ;ASCII of hundreds
        MOV    P2, A
        MOV    A, B
        ORL    A, #30H    ;ASCII of tens
        MOV    P1, A
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

chapter also presented programs for code conversions. The memory allocation in 8051 C is compared with that of ALP.

7.11 Questions

1. Indicate the data type you would use for
   (a) temperature
   (b) age of person
   (c) count vehicles crossing a junction
   (d) name of a person
   (e) memory address
   (f) a message to thank people
2. Give the hex value sent to the port in each of the following C statements.
   (a) P2 = 14
   (b) P2 = 0 × 14
   (c) P2 = 'A'
   (d) P2 = '7'
   (e) P2 = 45
   (f) P2 = 0 × 0F; P2 = 'X'.
3. What are the advantages and disadvantages of programming in C?
4. What are the factors which affect the delay program?
5. Differentiate between the sbit and bit data type.
6. Write a C program to toggle P2.2 every 100ms.
7. Write a C program to count up P0 from 0–99 continuously.
8. Explain the usage of

   while(1)

   statement in a C program.
9. Write a time delay function for 50ms.
10. Indicate the port output in each case.
    (a) P0 × F0 & 0 × 25;
    (b) P0 = 0 × 25 & 0 × 69;
    (c) P1 = 0 × F1^0 × 80;
    (d) P2 = 0 × 80^0 × EA;

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
    while (1)
    {TL0=0×1B;                    //initial value for 4ms in mode 0
        TH0=0×1A;
        outpin=1;                 //make port pin High
        delay ();
        TL0=0×04;                 //initial value for 3ms in mode 0
        TH0=0×54;
        outpin=0;                 //make port pin Low
        delay();
    }                             //end of while loop
}                                 //end of main
void delay (void)
{TR0=1;                           //start timer
        while (TF0==1);           //wait till TF is set
        TR0=0;                    //stop timer
        TF0=0;                    //clear TF
}
```

The initial values are loaded in the main program in the above program. An alternate method is to load the initial values in the delay subroutine. In this case two subroutines have to be written for the two different delays say ondelay and offdelay as shown below in the C program. For each delay (on/off) different timers can be used. In the below program ondelay is generated by timer 0 in mode 0 & offdelay by timer 1 in mode 0. By this method flexibility is obtained in the use of timers and also the modes used.

Alternate C program

```
#include <reg51.h>
void delayon (void);
void delayoff (void);
sbit outpin=P3^4;
void main()
    {TMOD=0×00; ,                 //Timer 0 & Timer 1 in mode 0
        while (1)
        {outpin=1;                //make port pin High
        delayon ()
        outpin = 0;               //make port pin low
        delayoff ();
        }                         //end of while
    }                             //end of main.
void delayon (void)
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

the initial value). Hence timer in mode 2 has autoreload facility. This mode has many applications, the common being the setting of the baud rate in serial communication.

Example 8.7. Write a program to generate a square wave of frequency 10KHz on pin 1.4. Use timer 0 in mode 2 with a crystal frequency of 22MHz.

Solution Mode 2 of timer is a 8-bit timer with a maximum value of FFH. To generate a square wave of 10 KHz, the time delay in T_{ON} & T_{OFF} of the square wave is the same (50% duty cycle) and is equal to $\frac{T}{2}$ where $T = \frac{1}{f} = \frac{1}{10k} = 0.1$ms. Hence $T_{ON} = \frac{0.1\,ms}{2} = 0.05$ms.
Hence required delay = 0.05ms

$$\begin{aligned}
\text{(Initial value - 1)} &= \text{(maximum value of mode 2)} \\
&\quad - \text{Required delay} \times \frac{\text{Crystal frequency}}{12} \\
&= (\text{FF}) - 0.05 \times 10^{-3} \times \frac{22 \times 10^6}{12} \\
&= 255 - 91.6 \\
&= 163 \\
&= \text{A3H}
\end{aligned}$$

Initial value = A4H; TH0 = A4 (Timer 0 is used)

Algorithm

1 Initialize TMOD for timer 0 in mode 2
2 Load the initial value in TH0
3 Start the timer (TR0 = 1)
4 Wait until timer overflows (TF0 = 1)
5 Stop timer (TR0 = 0)
6 Clear timer flag (TF0 = 0)
7 Toggle port pin (for square wave)
8 Repeat from step 3 (no need to reload the initial value)

NOTE: Step 5 is optional. If step 5 is deleted, repeat from step 4 (no need to start again).

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

In many applications when the number of counts the timer in mode 2 has to count is known, say N, then instead of entering the initial value in TH as (FFH – N +1), we can enter –N directly in the assembly language program.

Example 8.9. Calculate the value (in Hex) loaded into TH register for each of the following cases

(i) MOV TH1, #-48

(ii) MOV TH1, #-48H

(iii) MOV TH0, #-12

(iv) MOV TH0, #12H.

Solution

(i) TH1 = D0H

(ii) TH1 = B8H

(iii) TH0 = F4H

(iv) TH0 = 12.

8.4 Counter application

In all the above programs we have used the timers in timer mode; $C/\overline{T} = 0$ in TMOD register. In the timer mode, the timers count the clock pulses from 8051's crystal oscillator, where the clock pulse frequency is $\frac{\text{crystal frequency}}{12}$. In the counter mode of operation; $C/\overline{T} = 1$ in TMOD register and the timers count the clock pulses from an external source (outside 8051).

Timer 0 in counter mode will count the clock pulses given at P3.4 (port 3) and timer 1 will count the clock pulses given at P3.5. The Fig. 8.3 illustrates the use of timer 0 as a counter

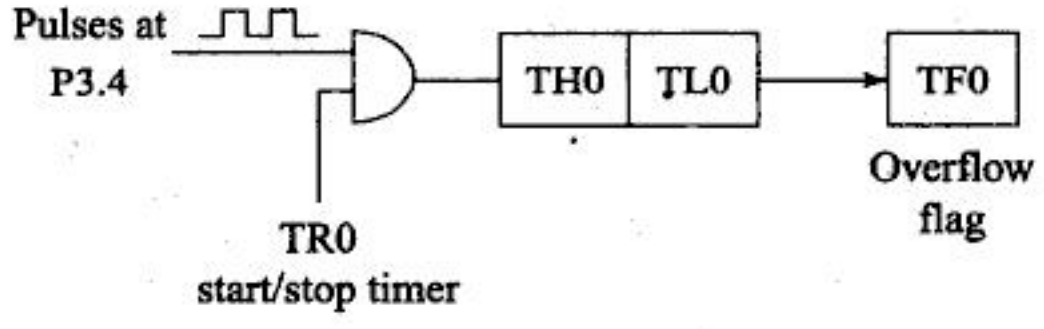


Figure 8.3. Timer 0 acting as a counter.

NOTE: The mode 0,1,2 operation in timer mode & counter mode of operation remain the same.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Example 8.12. Assume that a 1 Hz external clock is being fed into pin T1 (P3.5). Implement a 8-bit upcounter on port P1 to count from an initial value to FFh.

Solution Use counter 1 (because of T1) in mode 2 (for auto reload from initial value).

Assembly language program

```
        MOV    TMOD #60H    ;counter 1, mode 2
        SETB   T0           ;make port pin input
        MOV    TH0, #20H    ;initial value = 20H
                            ;(can be any value)
        SETB   TR0          ;start timer
AGAIN:  MOV    A, TL0       ;get count in A
        MOV    P1, A        ;and display in P1
        SJMP   AGAIN
```

C Language program (Counter 1)

```
#include <reg51.h>
void main()
{T1=1;
    TMOD=0×60;
    TH1=0×20;                    //initial value
    TR1=1;                       //start timer
    while(1)
        {P1=TL1;                 //display count
        }
}
```

C program (considering TF1)

```
#include <reg51.h>
void main()
{T1=1; TMOD=0×60;
    TH1=0×20;
    while(1)
    {do
        {TR1=1;
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
        SETB  TR1
RPT:    MOV   A, P1          ;read from P1 port
        MOV   SBUF, A        ;send to SBUF
WAIT:   JNB   T1, WAIT       ;wait till transmission is complete
        CLR   TI
        SJMP  RPT            ;clear TI for next transmission
```

C program

```
#include <reg51.h>
void main()
{unsigned char x;
    P1=0×FF;
    TMOD=0×20; TH1=0×FA;
    SCON=0×50; TR1=1;
    while(1)
        {x=P1;                    //read P1 port
        SBUF=x;                   //place in SBUF
        while (TI==0);            //wait till serial transmission is over
        TI=0;                     //clear for next transmission
        }
}
```

In the next few programs the serial transmission program is converted into a subroutine which can be used in any main program.

Example 8.18. Consider that a switch SW is connected to pin P2.3. Monitor the SW status and if SW = 0: send 'Hello' and if SW = 1: send 'world' serially. Assume XTAL = 11.0592 MHz, baud rate of 9600, 8 bitdata, 1 stop bit.

Algorithm:

1. Initialize for serial transmission (TMOD, SCON, TH1, TR1).
2. Make P2.3 as input bit.
3. Read P2.3 data pin.
4. If P2.3 pin is high, send 'world' & if low, send 'hello'.
5. Repeat from step 3.

NOTE: In the assembly language program, while storing the strings "Hello" at locations label PLOW and "world" at PHIGH, the last character after the

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Solution Use Timer 1, mode 2 for serial transmission. Now the remaining timer 0, mode 2 can be used to generate the square wave on pin P1.3.

To transmit the information of square wave, check the status of the pin P1.3. When this pin P1.3 is High, transmit FF serially, so that at the receiver it is decoded as 1. Similarly when the pin P1.3 is low, transmit 00 serially. Hence at the receiver side a square wave can be regenerated.

Algorithm:

1. Initialize Timer 1 & 0 in mode 2
2. Load initial value in TH1 for the required baud rate
3. Load initial value in TH0 for square wave frequency (let TH0 = 00)
4. Start Timer 1
5. Clear P1.3 and make accumulator = 00H
6. Start Timer 0 and wait for it to overflow.
7. Stop Timer 0, clear TF0
8. Complement the pin and accumulator value (00→FF→00 ...)
9. Output accumulator value serially
10. Repeat from step 6.

Assembly Language program

```
        ORG     00
        MOV     TMOD, #22H   ;Timer 0, 1 in mode 2
        MOV     SCON, #50H   ;
        MOV     TH1, #-3     ;9600 baud
        MOV     TH0, #00H    ;square wave freq (minimum)
                             ;(delay is maximum)
        SETB    TR1          ;start timer1 for serial port
        MOV     A, #00       ;Initialize accumulator
        CLR     P1.3         ;and pin level
AGAIN:  SETB    TR0          ;start timer for square wave
WAIT:   JNB     TF0, WAIT
        CLR     TR0          ;stop timer 0
        CLR     TF0
        CPL     P1.3         ;toggle pin
        CPL     A            ;complement accumulator
        MOV     SBUF, A      ;transmit data
HERE:   JNB     TI, HERE     ;wait till transmission completes
        CLR     TI
        SJMP    AGAIN
        END
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

monitor TI. While sending message"BEAUTIFUL", write into SBUF1 and monitor TF1.

Assembly language program

```
        SBUF1  EQU 0C1H;
        SCON1  EQU 0C0H
        TF1    BIT 0C1H
        ORG    OOH
        MOV    TMOD, #20H
        MOV    TH1, #-3        ;9600 baud
        MOV    SCON, #50H      ;SCON for both serial ports
        MOV    SCON1, #50H
        SETB   P2.2            ;configure for input pin (SW)
        SETB   TR1             ;start timer
Again:  JB     P2.2, high      ;go to display 'beautiful' if P2.2=1
        MOV    DPTR, #DISP1    ;SW=0, hence display fine at serial
                               ;port #0
next:   CLR    A
        MOVC   A, @A+DPTR      ;get value from ROM
        JZ     Again           ;if character is zero, break
                               ;the loop (end of string)
        MOV    SBUF, A         ;send to serial port #0 for
                               ;transmission
wait:   JNB    TI, wait        ;wait if TI=0, i.e., until end of
                               ;transmission
        CLR    TI              ;reset flag for next character
        INC    DPTR            ;increment DPTR for next consecutive
                               ;address in string
        SJMP   next
high:   MOV    DPTR, #DISP2
next2:  CLR    A
        MOVC   A, @A+DPTR      ;get character from string 2
        JZ     Again           ;end of string 2, break
        MOV    SBUF1, A        ;transmit using serial port #1
wait1:  JNB    TI1, wait1      ;wait till transmission is complete
        CLR    TI1
        INC    DPTR
        SJMP   next2
DISP1:  DB     ''FINE'', 0
DISP2:  DB     ''BEAUTIFUL'', 0
        END
```

The corresponding C program is given below

```
#include <reg51.h>
sbit SW=P2^2;                    //declare the port pin as a variable SW
sfr SBUF1=0×C1;                  //define the registers of serial port #1
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Chapter 9

Interrupts

Interrupts, abort the sequential execution of instructions. Interrupts are generated by the external environment. The 8051, has five interrupts (excluding reset). These interrupts can be enabled or disabled. Their priorities can also be set. Two SFR's namely, Interrupt Enable and Interrupt priority are used to control the interrupts.

Learning objectives

- Enable and disable interrupts.
- Set the priority of interrupts.
- Explain interrupt vector table.
- Program timers using interrupts.
- Program interrupt-based serial communication.
- Service more than one interrupt.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

NOTE: Once 8051 jumps (branches) to the ISR address (in the interrupt vector table), the interrupt flag TF0 (or TF1) is automatically cleared by the on-chip hardware. The user neednot write a separate CLR TF0 instruction.

In Example 9.2, the ISR is very small (less than 8 bytes), and hence it can be written in the interrupt vector table itself (note, the interrupt vector table is from 00H to 002FH and each interrupt is allotted 8 bytes). In example 9.3, the ISR occupies more than 8 bytes, hence a jump to a location in the program space away from the interrupt vector table is provided at the ISR address in the interrupt vector table.

Example 9.2. Write a program to generate a square wave of 10 kHz with timer 0 in mode 2 at port pin 1.3 using interrupt mode. Also display a value of 'A' at port 2 and 'B' at port 0. XTAL = 22MHz.

Solution The black schematic of the above problem statement is shown in Fig. 9.1.

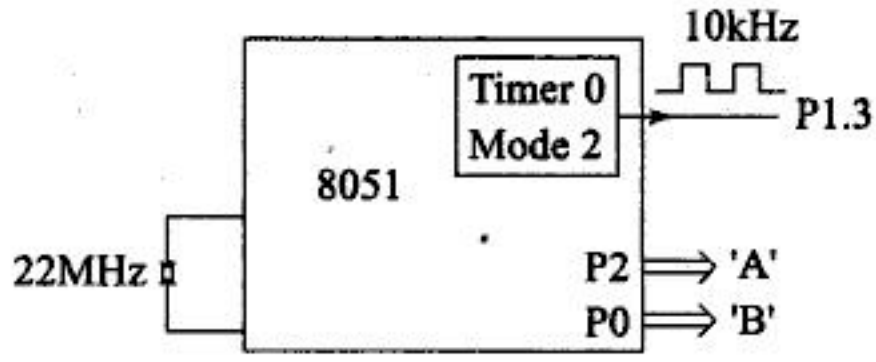


Figure 9.1. Example 9.2—Schematic.

The main program will initialize timer and keeps on displaying 'A' & 'B' at the ports. In the interrupt service routine (ISR), the port pin 1.3 is toggled to generate the square wave. The ISR is branched to 000BH (address of timer 0 interrupt) when TF0 is set (i.e., timer 0 overflows from FFH to 00H or initial value). Since mode 2 is used, no need to load the initial value as this mode has autoreload facility. Also no need to reset TF0 flag, as the on-chip hardware circuitry will reset TF0 when 8051 branches to the interrupt vector table.

Algorithm

Main program

1 Initialize TMOD for timer 0, mode 2

2 Load initial count in TH0 for 10 kHz

3 Enable interrupts (EA = 1) and timer 0 interrupt (ET0 = 1) in IE register

4 Start timer 0

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The corresponding C program is given below.

```
#include <reg51.h>
void timer1(void) interrupt 3     //refer table 9.2
{
  TR1=0 ;                         //stop timer 1
  if (P2^4=0)
    {TH1=0×FC;                    //on-time initial value to get 1ms
    TL1=0×67;
      P2^4=1;                     //make pin high
      TR1=1;                      //start timer
    }else
    {TH1=0×F8;                    //Off-time initial value
      TL1=0×CE;
      P2^4=0;                     //make pin low
      TR1=1;
    }
  }                               //end of subroutine
void main ()
  {unsigned char val;
    TMOD=0×10;                    //timer 1, mode 1
    IE=0×88;                      //enable timer 1 interrupt, can written as
                                  //EA=1; ET1=1;
      TH1=0×FC;                   //on-time initial value
      TL1=0×67;
      P2^4=1;                     //make pin high
      TR1=1;                      //start timer
      while (1)                   //repeat continuously
      {val=P0;                    //read value from port P0
      P1=val;                     //and display at P1
      }                           //end of while
}                                 //end of main
```

The below program is an example of enabling two timer interrupts simultaneously in the same program.

Example 9.4. Write a program to generate two square waves of 5 kHz and 25 kHz at pins P1.3 and P2.3 respectively, with crystal frequency of 22 MHz and in interrupt mode.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

9.3 External interrupts

In the following sections the features of external hardware interrupts are elaborated. There are two external hardware interrupts in 8051. They are INT0 and INT1, located on pins P3.2 and P3.3, with interrupt vectors (addresses) at 0003H and 0013H respectively. They are enabled/disabled using the EX0 and EX1 bits in the IE register.

Each external hardware interrupt can be programmed for two types of activation (1) Level triggered and (2) Edge triggered; selected by the reset/set of IT0 and IT1 bits in TCON register.

9.3.1 Level Triggered interrupt mode

The interrupting device should place a 'low level' on the external interrupt pin (INT0/INT1) for at least 4 machine cycles to be recognized as an interrupt request by 8051. In this mode the external interrupt pin is made high if no ISR is needed to be executed. The features and working of the level triggered interrupt mode is explained below.

1 Level triggered interrupt mode is selected when the corresponding IT0/IT1 bit in the TCON register is low. (Generally on reset TCON register is cleared and hence this is the default mode)
2 to trigger an interrupt (i.e., for the interrupt to be recognized by the 8051), the low level should be held for at least 4 machine cycles.
3 Once the interrupt is recognized, the microcontroller finishes the current execution, saves the next address on the stack (PC pushed on to stack), and jumps to the interrupt vector table (0003H/0013H address in ROM) to service the interrupt.
4 The microcontroller executes the ISR till the RETI instruction
5 The low-level signal at the interrupt pin INT0/INT1 should be removed before the execution of RETI instruction. If it is still low during the RETI instruction, then the 8051 recognizes it as a new interrupt and will branch again to the ISR.
6 The interrupt flags IE0 and IE1 (similar to TF0/TF1 for timers) are not used in the low-level triggered interrupt mode.

Disadvantages of level-triggered interrupt mode

1 The minimum duration of the low-level on the external pin (INT0/INT1) is 4 machine cycles. If it is less than this, the interrupt maynot be recognized by 8051.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
        CPL    P1.3          ;toggle pin
        RETI
        ORG    0030H
START:  SETB   IE0
        SETB   TCON.0
        MOV    IE, #81H      ;enable INT0 interrupt
HERE:   SJMP   HERE
        END
```

C language Program

```
#include <reg51.h>
void interrupt0(void) interrupt 0      //ISR for INT0
  {P1^3=~P1^3;
  }
  void main ()
    {IE0=1;                            //INT0 – edge triggered mode
        IE=0×81;                       //enable INT0 interrupt
        while (1);                     //wait indefinitely
    }
```

Example 9.7. (b) Write a 8051 C program to count a 1 Hz pulse connected to INT1 pin and display it on P0.

Solution INT1 interrupt should be configured as edge triggered interrupt to count the number of pulses (either +ve edge or −ve edge is sufficient).

Algorithm

1 Enable edge triggered interrupt mode by enabling IE1 = 1 in TCON

2 Enable external interrupt INT1 by making EX1 = 1 & EA = 1 in IE register (=84H)

3 Display the count in a globally defined variable 'counter' on P0 & jump to step 3.

ISR at 0013H

1 Increment "counter"

2 Return from interrupt

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Serial port ISR at 0023H

1 Jump to SERIAL subroutine

SERIAL subroutine (written after main program or use ORG 00A0H, etc.)

1 If TI is not set, go to step 3 for receiving data serially

2 If TI is set, (a) clear TI, (b) move data from next location in RAM (from 45H) into SBUF for transmission, (c) Decrement counter (d) Return from ISR

3 If RI is set, just read SBUF value and clear RI and return from ISR

(**NOTE:** The value read from SBUF is just discarded, as in this program we are not doing anything with the reception of serial data, but by chance any serial data is received, SBUF is read & RI is cleared).

For 'C' program, the above example is modified as-transmit 10 bytes of data stored in the string mydata. The storing of data from mydata in RAM location is done using the instruction:

Unsigned char mydata[]=0×30, 40H, 'A', "SPEED";

In the above initialization of mydata, the serial data transmitted is 30H, 40H, 41H, (ASCII value of A), 53H, 50H, 45H, 45H, 44H.

ASCII values of "SPEED"

To store the same string in *ROM location*, the codeword 'code' is used

code unsigned char mydata[]=30H, 40H, "A", "SPEED";

Assembly language Program

```
        ORG    00
        SJMP   START
        ORG    023H
        LJMP   SERIAL
        ORG    0030H
START:  MOV    TMOD, #20H
        MOV    TH1, #-3
        MOV    SCON, #50H
        MOV    IE, #90H
        SETB   TR1
        MOV    R0, #10          ;counter for 10 data bytes
        MOV    R1, #45H         ;starting address of data in RAM
        MOV    A, @R1           ;move data from RAM to A
        MOV    SBUF, A          ;serially transmit data
                                ;(one serial data is transmitted in
                                ;main program for TI to be set;
                                ;rest in subroutine);
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

(6) Start timer 0 and timer 1

(7) Transmit a letter 'H' once serially (for TI interrupt to be set)

(8) Read from P1.7 and send to P1.3

(9) Repeat step 8 continuously.

Timer 0 ISR at 000BH

(1) Clear TF0

(2) Reload initial values into TH0: TL0 = F8CE H for square wave

(3) Complement bit P2.3 for square wave

(4) Return from interrupt.

Serial ISR at 0023H

SJMP to serial routine.

Serial routine

(1) If TI = 1, then clear TI, send 'H' to SBUF & RETI

(2) Else clear RI and return from subroutine

Assembly language Program

```
ORG    00           ;a reset vector
LJMP   START
ORG    00BH         ;Timer 0 vector since ISR is small,
                    ;code here itself
CLR    TR0          ;Stop timer 0
CLR    TF0          ;clear the interrupt flag-optional
                    ;as RETI will clear it
CPL    P2.3         ;complement bit for square wave
                    ;generation
MOV    TH0, #F8H    ;Reload initial value (remember mode
NOV    TL0, #0CEH   ;1 has no auto-reload capacity)
SETB   TR0          ;start timer again
RETI

ORG    0023H        ;serial interrupt vector
LJMP   SERIAL

ORG    0030H        ;main program
MOV    TMOD #21H    ;Timer 1 in mode 2 and Timer 0
                    ;in mode 1
MOV    TH1, #-3H    ;for baud rate of 9600
MOV    TH0, #0F8H   ;initial value for square wave
MOV    TH1, #0CEH
MOV    SCON, #50H   ;serial mode 1; 8-bit data
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service. Every interrupt has a program called the interrupt service routine, stored in a predetermined RAM location. The six interrupts are reset, two to indicate timer overflows, two external interrupts and a serial communication interrupt.

The 8051 can be programmed to enable or disable an interrupt. It can also be programmed to set the interrupt priority. This chapter has presented several programs in both assembly language and C to illustrate the functioning of interrupts.

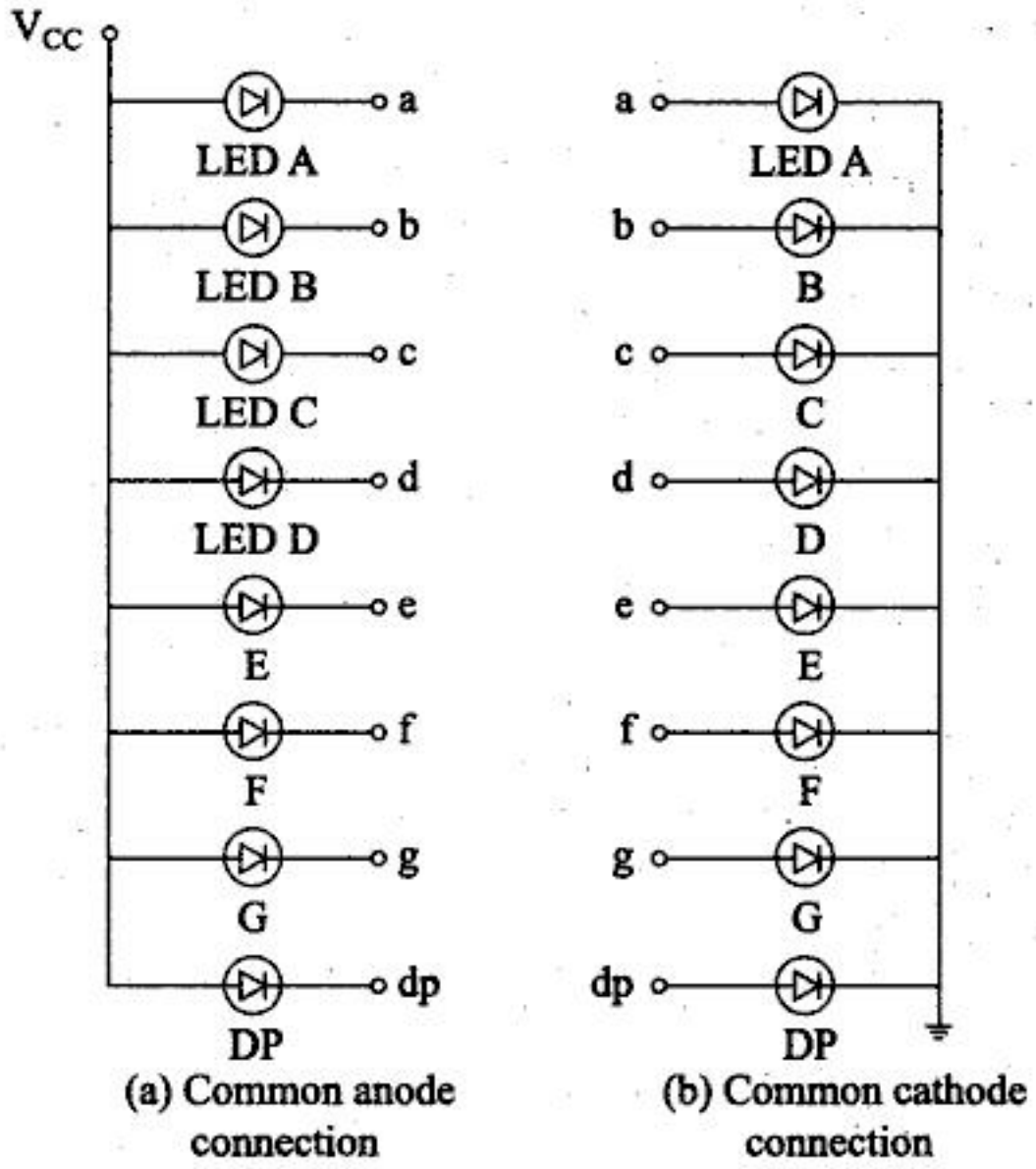
9.7 Questions

1. What is the advantage of using interrupts?
2. What is the meaning of vectored interrupt?
3. In 8051 clearly explain how different interrupts are enabled/disabled.
4. Why is the SJMP/LJMP instruction on address 0?
5. Clearly explain the sequence of events after an interrupt occurs?
6. Explain how two square waves of different frequencies can be generated by the 8051.
7. If Timer 1 is programmed for mode 1, TH0 = FFH; TL1 = F8H and IE bit for T1 is enabled, explain how interrupt is activated.
8. Write a program in which every 2 seconds the LED connected to P2.1 is turned on and off four times.
9. How many hardware interrupts are available in 8051? How are they activated?
10. Explain the difference between level triggered and edge triggered interrupts.
11. How do we take care to see that a single interrupt is not activated multiple times?
12. Write a program to get data from P0 and send it to P1 while T0 is used to generate a 2KHz square wave on P2.2.
13. Explain the serial communication interrupt and how it is activated.
14. Explain the difference between RET and RETI.
15. Write an ALP and a C program to get data serially and send it to P1, while T0 (timer 0) turns an LED on and off every second at P0.2.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



(a) Common anode connection

(b) Common cathode connection

(c) Common anode 7-segment display

Figure 10.2.

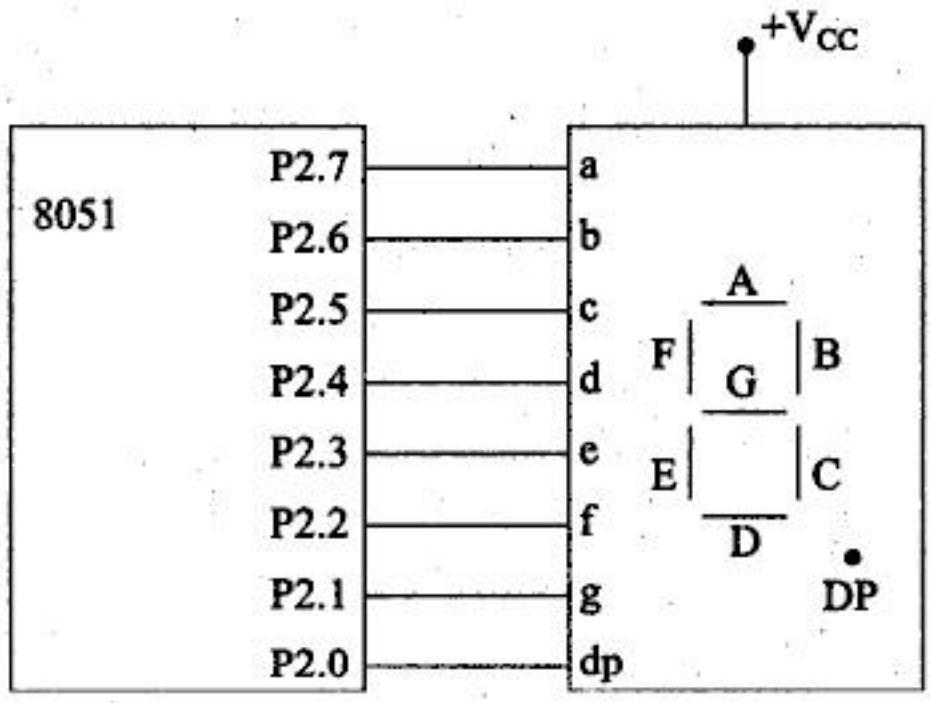


Figure 10.3. 8051 connected to a 7-segment display.

c, d, e & 'f' and a low on pins '8' and 'dp' should be sent to the 8051 port (here P2) where the 7-segment display is connected.

Example 10.1. Display numbers 0-9 continuously on a common anode 7-segment display connected to 8051 at port P2.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

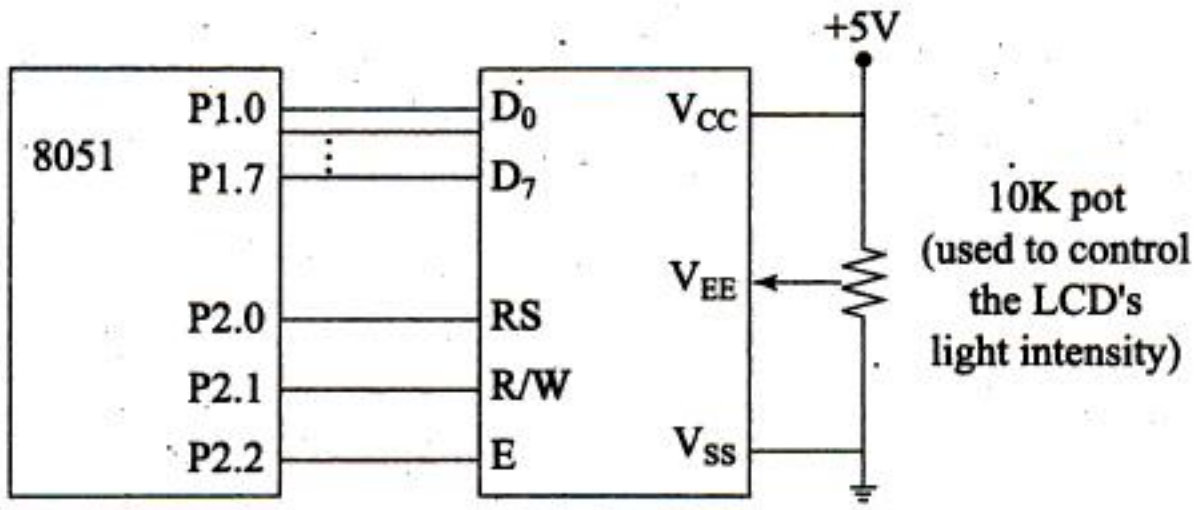


Figure 10.6. Interfacing of LCD to 8051.

Table 10.2 LCD pin functions.

Pins	V_{SS}, V_{CC}, V_{EE}	Function Ground, +5V, & contrast control pins
CONTROL PINS	RS	RS = 0 – select command register RS = 1 – select data register
	R/W	R/W = 0 – write to LCD R/W = 1 – Read from LCD
	E	Enable – high-to-low pulse (450 ns wide) to latch data D_0–D_7 into LCD
DATA PINS	D_0–D_7 (bidirectional)	8 data pins, D_0–D_7, used to send data/command byte to LCD or read from LCD

2. The ASCII value of the character to be displayed (say 41H to display 'A') is sent on the D_0–D_7 data lines with RS = 1 (data register), R/W = 0 and a high-to-low pulse on the 'E' pin.

Command words used in LCD are many, due to the various features available in LCD. The LCD display can display 16/20 characters per line and the number of lines in the display can vary from 1, 2 or 4 lines depending on the model. In the LCD, the data can be displayed at any location, say 3rd character position in line 2, etc. The command word 80H sent to the LCD, positions the cursor at position 0 on line 1. Similarly 86H, command word positions the cursor at position 6 on line 1. On line 2, the cursor position command word starts from COH and the 19th position (for 20 × 2 LCD i.e., 20 characters per line & 2 lines) on line 2 is D3H.

The data byte sent to the LCD after the command word COH is sent, is displayed at position 0, line 2. Before the next data byte is written the cursor should be shifted right automatically. For this during the initialization of the

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

;Delay1

```
DELAY1:  MOV     R3, #10     ;small delay
LOOP3:   MOV     R4, #255
LOOP4:   DJNZ    R4, LOOP4
         DJNZ    R3, LOOP3
         RET
         END
```

NOTE: DATAWRT and CMDWRT subroutines are almost same except for RS, which is low (RS = 0) for command & high (RS = 1) for data.

Generally on a LCD when a string, such as 'HELLO' in the above example, has to be displayed, the programming technique is to store the string (array) in the ROM (say at location 200H after all the programs-main and subroutines, have been entered). This string is accessed using 'MOVC' command as shown below. The DATAWRT, CMDWRT, INITLCD, DELAY, etc subroutines remain the same.

```
         ORG     0H
         SJMP    START
         ORG     30H
START:   ACALL   INITLCD           ;initialize LCD
         MOV     DPTR, #DISPDATA   ;address of data string
                                   ;to be displayed
NEXT:    CLR     A
         MOVC    A, @A+DPTR        ;get data pointed by DPTR
                                   ;into Accumulator
         JZ      HERE              ;if character is zero,
                                   ;indicates
                                   ;end of string, so end
         ACALL   DATAWRT           ;display data on LCD
         INC     DPTR              ;point to next character
                                   ;in the string
         SJMP    NEXT
HERE:    SJMP    HERE
                                   ;DATAWRT subroutine
                                   ;CMDWRT subroutine
                                   ;INITLCD subroutine
                                   ;DELAY subroutine
         ORG     200H              ;ROM location from
                                   ;which string
                                   ;to be displayed is stored
DISPDATA: DB     "HELLO", 0        ;data and NULL
         END
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

;INITLCD Subroutine

```
INITLCD:  MOV     A, #38H     ;2 lines, 5×7
          ACALL   CMDWRT
          MOV     A, #0EH     ;cursor ON
          ACALL   CMDWRT
          MOV     A, #01      ;clear LCD display
          ACALL   CMDWRT
          MOV     A, #06      ;shift right
          ACALL   CMDWRT
          MOV     A, #80H     ;1st line display
          ACALL   CMDWRT
          RET
```

;CMDWRT Subroutine

```
CMDWRT:   ACALL   READY       ;wait till LCD is ready
          MOV     P1, A       ;send command to P1
          CLR     P2.0        ;RS=0 for command
          CLR     P2.1        ;R/W=0 for write
          SETB    P2.2        ;E=1 FOR H--L pulse
          ACALL   DELAY1      ;small delay
          CLR     P2.2        ;E=0
          RET
```

;READY Subroutine

```
READY:    SETB    P1.7        ;make P1.7, i.e., D7 as input pin
          CLR     P2.0        ;RS=0 for command register
                              ;(Busy flag is D7 of command
                              ;register)
          SETB    P2.1        ;R/W=1 for read operation
                              ;read command register by giving
                              ;a low-high pulse on 'E' pin
BACK:     CLR     P2.2        ;E=0 for L-H pulse
          ACALL   DELAY1      ;small delay
          SETB    P2.2        ;E=1 (now the contents of command
                              ;register are put on D0--D7)
          JB      P1.7, BACK  ;if P1.7=D7 of command
                              ;register=BUSY FLAG is 1, wait
                              ;by reading command register
                              ;again & again
          RET
                              ;DELAY1 subroutine
DELAY1:   MOV     R3, #10
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
            ORG     030H        ;main program
MAIN1:      SETB    TCON.0      ;make INT0 as edge triggered
            MOV     IE, #81H    ;EA=1 & EX0=1; enable INT0
                                ;interrupt
            MOV     TMOD, #20H  ;Timer 1, mode 2
                                ;(for serial clock)
            MOV     TH1, #-3    ;9600 baud rate
                                ;(refer timer/serial chapter)
            MOV     SCON, #50H  ;enable serial transmission
            SETB    TR1         ;start timer 1
HERE:       SJMP    HERE        ;wait here
            ORG     0080H       ;transmit subroutine
TRANSMIT:   MOV     SBUF, #30H  ;ASCII code for '0'
WAIT:       JNB     TI, WAIT    ;wait till TI=1 indicating
                                ;transmission complete
            CLR     TI          ;clear TI flag for next
                                ;transmission
            RETI                ;return from interrupt
            END
```

Example 10.6. Two switches are connected to pins P1.0 and P1.1 as shown in Fig. 10.11. They are also vectored to interrupt location 003H, i.e., INT0. Write a program to display 01 on port 2 when SW1 is pressed, 02 when SW2 is pressed and 0FH when both are pressed.

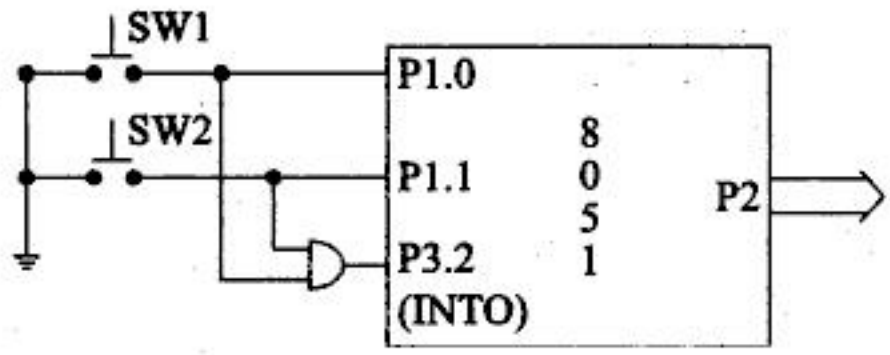


Figure 10.11. Schematic of Example 10.6.

Solution The port pins P1.0 & P1.1 are high when SW1 & SW2 are not pressed (because of internal pull-up resistors). When SW1 is pressed P1.0 goes low and even the output of the AND gate at P3.2 goes low. Since P3.2 is INT0, if EX0 is enabled, then the ISR at 003H location is executed. Pressing any one of the switches makes the output of the AND gate low & the INT0 ISR is executed.

Algorithm:

1. Enable P1.0 & P1.1 as input pins
2. Enable INT0 interrupt (EA=1 and EX0=1 in IE register, i.e., IE=81H)

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

This example requires the ASCII value of key pressed to be displayed on P0. If ASCII values are not required and only the key value is required then, instead of MOV DPTR, #ROW0, we can have MOV R2, #0 for row 0, MOV R2, #4 for row 1, MOV R2, #8 for row 2 & MOV R2, #C for row 3, and in the column part instead of INC DPTR, we have INC R2 and in the found part just found: MOV A, R2; MOV P0, A; LJMP AGAIN.

10.5 Stepper motor interfacing to 8051

A stepper motor translates electrical pulses into mechanical movement. A conventional motor (AC/DC motor) shaft runs freely, whereas the stepper motor shaft moves in a fixed increment & hence the shaft position can be controlled precisely, say move by 4° & stop. Stepper motors are used for position control applications such as dot matrix printers, disk drives, robotics, etc.

There are two types of stepper motors (SM)—permanent magnet SM & variable reluctance SM, depending on the rotor type (whether permanent magnet is used or not). The permanent magnet SM consists of a permanent magnet rotor (also called the shaft) surrounded by a stator as shown in Fig. 10.13(a). Generally the stator has 4 windings that are paired with a center-tapped common as shown in Fig. 10.14. The center tap allows a change of current direction in each of the two coils, hence changing the direction of polarity in the stator poles which return leads to a change in the direction of rotor rotation.

The rotation of the rotor in a SM along with the winding energization sequence is shown in Table 10.5.

Depending on the number of teeth on the stator & rotor, the stepper motor rotates a fired number of steps per revolution. The commonly available number of steps for one revolution are 500, 200, 180, 144, 72, 48, 24. The step angle, i.e., the movement of a single step of a stepper motor is calculated as $\frac{360°}{\text{no. of steps per revolution}}$. Say for 200 steps per revolution. The step angle is $\frac{360°}{200} = 1.8°$ per step.

Similarly step angle for 72 steps per revolution is $\frac{360}{72} = 5$.

For the 4-step switching sequence shown above, after four steps the same two windings will be 'ON', i.e., the sequence repeats after every 4 steps. After completing 4 steps, the rotor moves only one tooth pitch. Hence if the rotor has 50 teeth (each teeth is one pole), the number of steps for one complete revolution is 4 steps × 50 rotor teeth = 200 steps/revolution. Hence for smaller step angles (i.e., more steps/revolution), the rotor must have more teeth.

To double the number of steps/revolution, say 400 instead of 200, we follow the 8-step sequence shown in Table 10.6. Here, with this method, the

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
unsigned char i;
    while(1)                    //repeat continuously
    {for (i=0; i<8; i++)        //repeat for 8 steps with values from table
    {P1=table[i];
        delay(1000);
    }
    }                           //end of while
}                               //end of main
void delay (unsigned int initial)
{unsigned int i;
    for (i=0; i<initial; i++);
}
```

Example 10.11. Write a program to monitor the status of a switch SW connected to pin P2.7 and perform the following:

(a) If SW = 0, the stepper motor rotates clock wise

(b) If SW = 1, the stepper motor rotates counter clock wise (ACW)

Use the wave-drive 4-step sequence.

Solution

Algorithm:

1. Initialize the phase sequence as 88H for 4-step wave drive (NOTE: Apart from 88H, either 44H, 22H or 11H can be used)
2. Check SW i.e., P2.7 status, if high rotate the phase sequence left (ACW) else rotate the phase sequence right for clockwise direction
3. Output the phase sequence on port P1, call delay
4. Repeat from step 2 continuously.

ALP

```
        ORG     0H
START:  SETB    P2.7        ;make the pin as input pin
        MOV     A, #88H     ;initial phase value for 4-step
                            ;wave drive
next:   JNB     P2.7, CW    ;if switch = 0, go to clock wise
        RL      A           ;rotate left for anti-clockwise as
                            ;P2.7 is high
        MOV     P1, A       ;output on port P1
        ACALL   delay
        SJMP    next
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Now if D_0–D_7 = FFH, then $I_{out} = 2mA(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{256}) = 1.99mA$.

If D_0–D_7 = 80H, then $I_{out} = 2mA\ (\frac{1}{2} + \frac{0}{4} + \frac{0}{10} + \cdots + \frac{0}{256}) = 2mA(-\frac{1}{2}) =$ 1mA.

If D_7–D_0 = 30H, then $I_{out} = 2mA\ (\frac{0}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} + 0 + \cdots + 10) =$ $2mA \times \frac{3}{16} = 0.37mA$.

Using the I–V converter this I_{out} is converted to V_{out}.

The maximum V_{out}, for D_0–D_7 = FFH is calculated as $V_{out} = I_{out} \times 5k\Omega$ i.e., $V_{out} = 1.99mA \times 5K = 9.96V$ (where 5kΩ is the R_f in the I–V converter). For D_7–D_0 = 80H, the I_{out} = 1mA & the corresponding $V_{out} = 1mA \times 5K = 5V$. For D_7–D_0 = 00H, the $I_{out} = 0$ & $V_{out} = 0V$.

Another method to calculate the output analog voltage is to find the resolution of the DAC and multiply it by the digital word.

$$\text{Resolution of the DAC} = \frac{V_{max}}{2^n} = \frac{10V}{2^8} = 39.06mV \cong 39mV$$

Hence for D_7–D_0 = 80H, the output analog voltage is

$$V_{out} = \text{resolution of DAC} \times (D_7\text{–}D_0)_{BCD}$$
$$= 39mV \times 128 = 5V$$

DAC is commonly used in wave form generation as shown in the examples below.

Example 10.12. Interface a DAC 0808 to 8051 at port P1 and write an 8051 program to generate a sawtooth waveform.

Solution Consider that the interfacing is as shown in Fig. 10.17.

A sawtooth (ramp) wave form as shown in Fig. 10.18 has an amplitude which is continuously increasing to a maximum value.

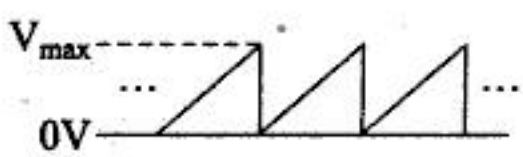


Figure 10.18. Sawtooth waveform.

NOTE: V_{max} is generated when D_7–D_0 = FFH.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Generally a sine wave starts at 0V with a positive half (0–180°) and a negative half (180°–260°) with +Vm as positive peak and −Vm as negative peak. The value of the sine wave at any point $v(t) = V_m \sin\theta$.

Here the DAC interfacing to 8051 setup produces an output voltage V_{out} in the range 0 to 10V, no negative values. Hence the range 0 to 10V is divided into 2 portions; 0–5V for negative half & 5V to 10V for positive half or simply the sine wave voltage $v(t) = 5 + 5\sin\theta$. The values for different sine angles 'θ' are given in Table 10.9, along with the digital byte to produce them. The digital bytes in the lookup table are calculated as voltage/resolution.

Table 10.9 Sine table.

Angle θ (in degree)	sin θ	V_{out} 5V + 5 sin θ	DAC i/p V_{out}/39.06
0°	0	5	128
30°	0.5	7.5	192
60°	0.866	9.33	238
90°	1	10	255
120°	0.866	9.33	238
150°	0.5	7.5	192
180°	0	5	128
210°	–0.5	2.5	64
240°	–0.866	0.669	17
270°	–1	0	0
300°	–0.866	0.669	17
330°	–0.5	2.5	64
360°	0	5	128

ALP for sine wave generation

```
AGAIN:  MOV     DPTR, #TABLE
        MOV     R2, #12     ;count value for 20 step b/w 0--360°
NEXT:   CLR     A
        MOVC    A, @A+DPTR  ;get value from look-up table &
                            ;o/p to DAC
        MOV     P1, A
        INC     DPTR        ;get next value
        DJNZ    R2, NEXT
        SJMP    AGAIN
        ORG     300H
TABLE:  DB 128, 192, 238, 255, 238, 192, 128, 64, 17, 0, 17, 64, 128
END
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

C Program

```
#include <reg51.h>
void main()
{P2^7=1;                          //make port pin as input pin
    while(1)                      //repeat continuously
    {if (P2^7==1)
        P1=0×09;                  //anticlock-wise direction
        else
        P1=0×06;                  //clock-wise direction
    }                             //end of while
}                                 //end of main
```

Apart from using a H-bridge, an IC such as L293 can be interfaced between 8051 and DC motor to control the direction of the DC motor as shown in Fig. 10.22. The optoisolator ILQ740 opto provides additional protection of the 8051 by isolating the 8051 from the DC motor circuit which is of higher power rating. The IC L293 has 3 input pins which are (1) enables pin-which enables the IC L293 when high (2) INPUT1 = 1, turns the motor in clock-wise direction (3) INPUT2 = 1 turns the motor in counter clock-wise by controlling current direction at OUTPUT1 & OUTPUT2 pins.

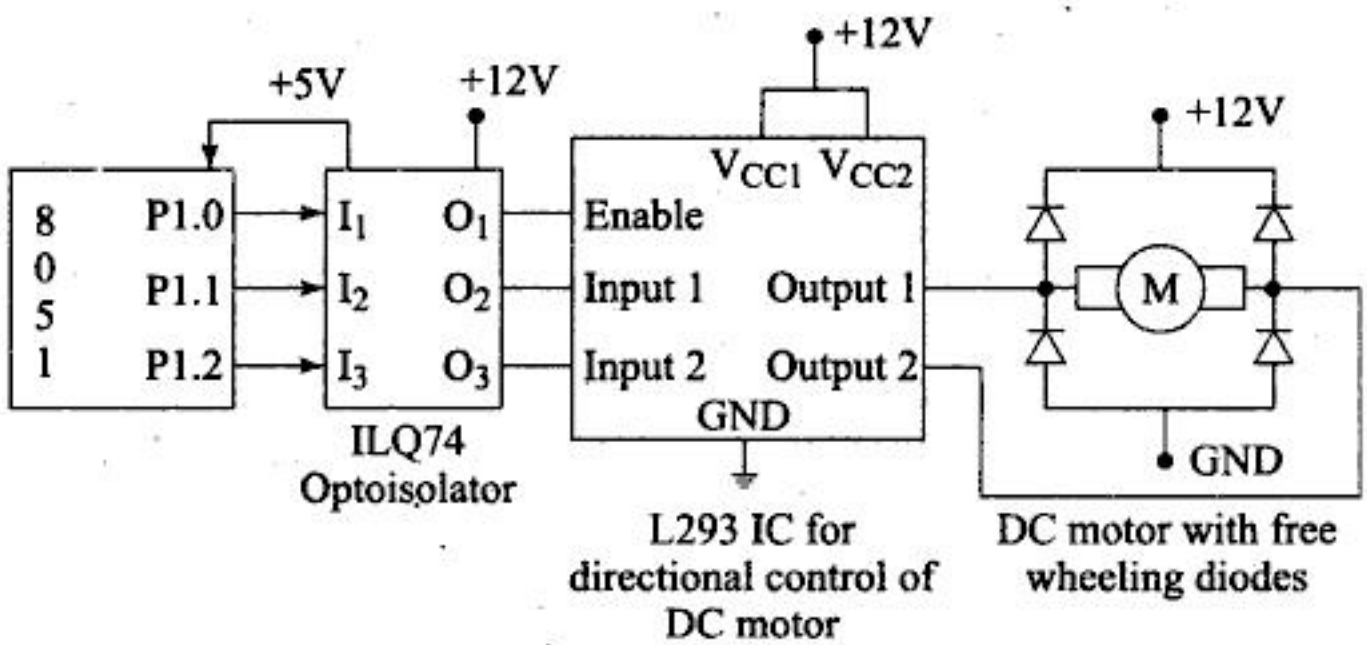


Figure 10.22. Bidirectional motor control using a L293 chip.

NOTE: Use a separate power supply (of higher rating) for the motor & L293 and another separate power supply for the 8051.

Example 10.20. Consider that a DC motor is interfaced to an 8051 through a L293. Write an 8051 program to monitor the status of a switch SW connected

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

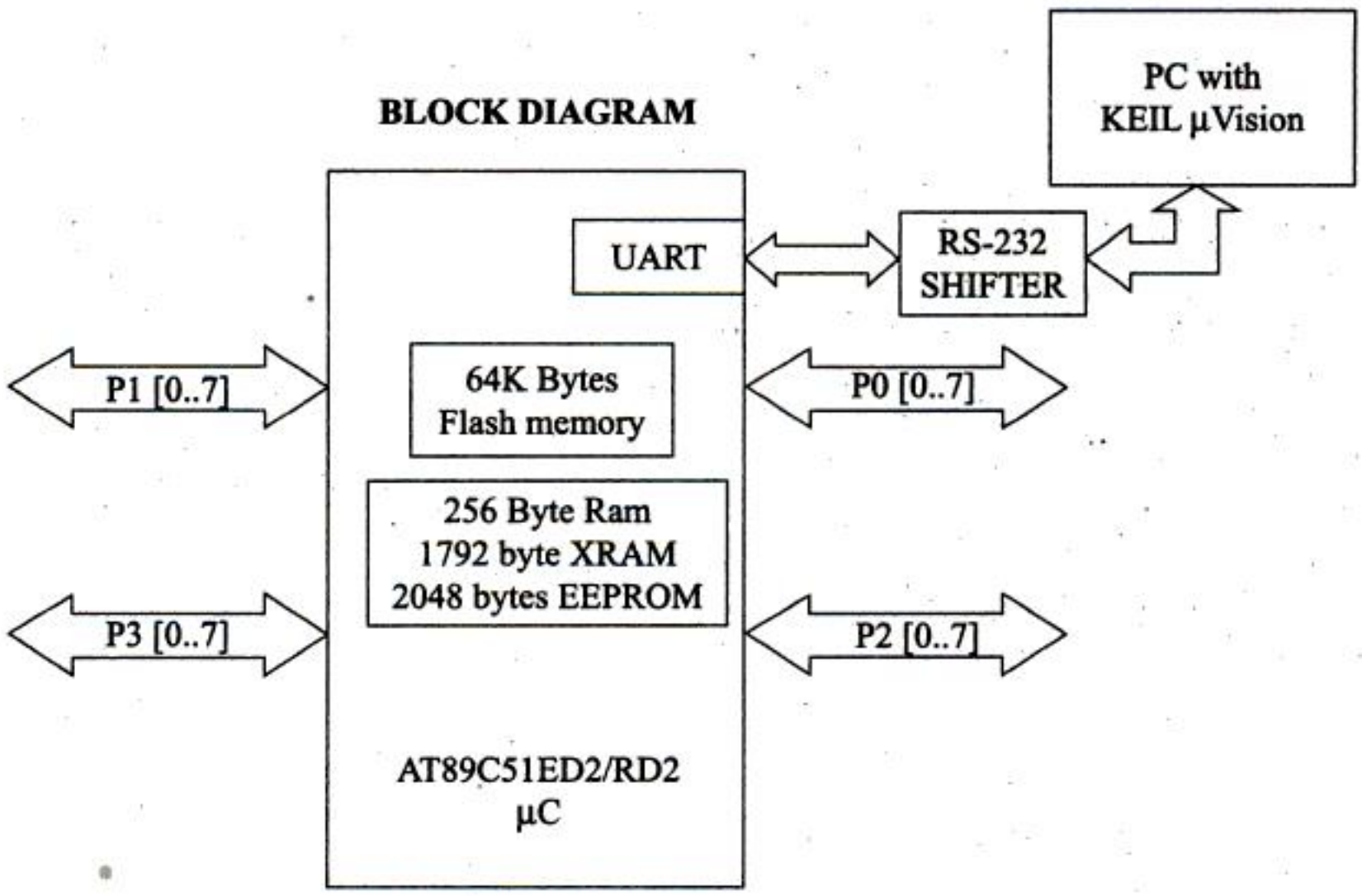
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

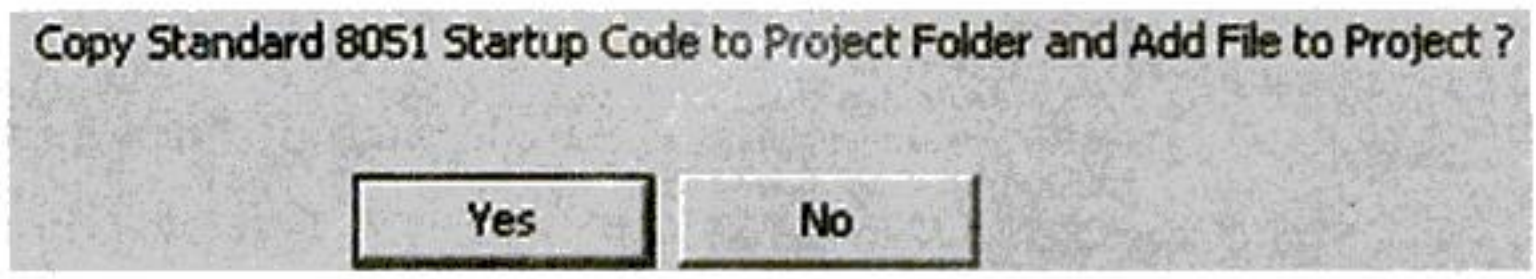


11.3 Creating and compiling a μVision2 project

Before assembling and simulating an 8051 program, the program has to be written in a *.asm file for assembly language programs and *.C file for C programs.The steps to be followed to simulate using the Keil Compiler are as follows:

1. Double Click on the μVision3 icon on the desktop.

2. Close any previous projects that were opened using – Project → Close.
3. Start **Project – New Project**, and select the CPU from the device database (Database-Atmel- AT89C51ED2). (Select AT89C51ED2 or AT89C51RD2 as per the board).On clicking 'OK', the following option is displayed. Choose Yes.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
L1:       MOV    dptr, #9000H //array stored from
                                //address 9000H
          MOV    A,R0         //initialize exchange counter
          MOV    R1,A
L2:       MOVX   a, @dptr     //GET NUMBER FROM ARRAY
          MOV    B, A         //& STORE IN B
          INC    dptr
          MOVX   a, @dptr     //next number in the array
          CLR    C            //reset borrow flag
          MOV    R2, A        //STORE IN R2
          SUBB   A, B         //2nd - 1st no.---no compare
                              //instruction in 8051
          JC     NOEXCHG      // JNC - FOR ASCENDING ORDER
          MOV    A,B          //EXHANGE THE 2 NOES IN
                              //THE ARRAY
          MOVX   @dptr,a
          DEC    DPL          //DEC dptr-INSTRUCTION
                              //NOT PRESENT
          MOV    a,R2
          MOVX   @dptr,a
          INC    DPTR
NOEXCHG:  DJNZ   R1,L2        //decrement compare counter
          DJNZ   R0,L1        //decrement pass counter
here:     SJMP   here
          END
```

Algorithm

1. Store the elements of the array from the address 9000H
2. Initialize a pass counter with array size-1 count (for number of passes).
3. Load compare counter with pass counter contents & initialize DPTR to point to the start address of the array (here 9000H).
4. Store the current and the next array elements pointed by DPTR in registers B and r2 respectively.
5. Subtract the next element from the current element.
6. If the carry flag is set (for ascending order) then exchange the 2 numbers in the array.
7. Decrement the compare counter and repeat through step 4 until the counter becomes 0.
8. Decrement the pass counter and repeat through step 3 until the counter becomes 0.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

4. The high part of the square result (SQH) is stored on the stack.
5. Multiply the low part of the square result (SQL) with x (partial cube result).
6. Store the low part of the above result at 9001H & the high part in R2.
7. Retrieve the high part of the square result (SQH) stored on the stack & multiply with x.
8. Add the low part of the above result (SQH*X) with R2 and store in 9002H.
9. Add the high part (SQH*X) with the resulting carry and store in 9003.

Program 11.3: Program Illustrating Bit Manipulations

7) Two eight bit numbers NUM1 & NUM2 are stored in external memory locations 8000H & 80001H respectively. Write an ALP to compare the 2 nos.

Reflect your result as: if NUM1<NUM2, SET LSB of data RAM 2F (bit address 78H)

IF NUM1>NUM2, SET MSB OF 2F(7FH). if NUM1 = NUM2-Clear both LSB & MSB of bit addressable memory location 2Fh

```
        ORG     0000H          //reset vector
        SJMP    30H
        ORG     30H
        MOV     DPTR,#8000H
        MOVX    A,@DPTR        //get number from 8000H
        MOV     R0,A           //R0=NUM1
        INC     DPTR
        MOVX    A,@DPTR        //A=NUM2
        CLR     C              //clear 'C' for SUBB
        SUBB    A,R0
        JZ      EQUAL
        JNC     BIG
        SETB    78H            //set bit at 78H bit address
        SJMP    END1
BIG:    SETB    7FH
        SJMP    END1
EQUAL:  CLR     77H
        CLR     7FH
END1:   SJMP    END1           //wait here
        END
```

Algorithm:

1. Store the elements of the array from the address 4000H
2. Move the first number in r0 and the second number in register A respectively

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

clear the TI flag and continue with transmission of the next byte by writing into the SBUF register. (The program can also be written in interrupt mode). The speed of the serial transmission is set by the baud rate which is done with the help of timer 1. (Refer Chapter 8). Timer1 must be programmed in mode 2 (that is, 8-bit, auto reload).

Baud rate Calculation: Crystal freq/ (12*32) = (11.0592MHz)/(12*32) = 28800.

Serial communication circuitry divides the machine cycle frequency (11.0592 MHz)/(12) by 32 before it is being used by the timer to set the baud rate.

To get 9600, 28800/3 is obtained by loading timer1 with -3 (i.e., FF – 3 = FD) for further clock division. For 2400 baud rate, 28800/12 => -12 = F4 in TH1.

Algorithm:

1. Initialize timer 1 to operate in mode 2 by loading TMOD register.
2. load TH1 with -3 to obtain 9600 baud.
3. Initialize the asynchronous serial communication transmission (SCON) register.
4. Start timer1 to generate the baud rate clock.
5. Transmit the characters "y" & "E" by writing into the SBUF register and waiting for the TI flag.

Program 11.7: Timer Delay Program

Program illustrating timer delay

13) Generate a 1second delay continuously using the on chip timer in interrupt mode.

```
ORG     0H          //Reset Vector
SJMP    30H
ORG     0BH         //TF0 vector
SJMP    ISR
ORG     30H         //main program
MOV     a, #00
MOV     R0, #0
MOV     R1, #0
MOV     TMOD, #02H  //00000010-Run timer0 in mode 2
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

2. Get the lower nibble & call ASCII routine
3. Store the converted ASCII value
4. Get the higher nibble & call ASCII routine
5. Store the converted ASCII value

ASCII subroutine

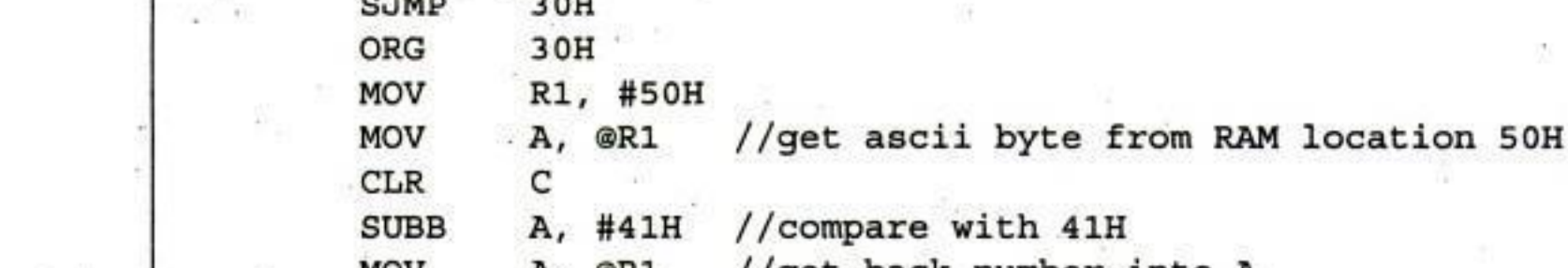
1. If digit greater than 09,(for A-F) add 07H & 30H
2. Else (i.e., for 0-9) add only 30H
3. return

18) Write an ALP to implement ASCII to hexadecimal conversion

```
        ORG     0000H
        SJMP    30H
        ORG     30H
        MOV     R1, #50H
        MOV     A, @R1    //get ascii byte from RAM location 50H
        CLR     C
        SUBB    A, #41H   //compare with 41H
        MOV     A, @R1    //get back number into A
        JC      SKIP      //if number <41, subtract only 30H
        CLR     C
        SUBB    A, #07H   //subtract 07 & 30H if number >41
SKIP:   CLR     C
        SUBB    A, #30H
        INC     R1
        MOV     @R1, A    //Store the hex code
here:   sjmp    here
        END
```

Result:

The ASCII code 45 at D:0050H is converted to hexadecimal -0E at 51H



NOTE:

For this program the input data should be only in the range **30H-39H & 41H to 46H**.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
  }
void main()
  {unsigned char freq = 500; //mid-delay in the range 100–1000
   P3 = 0 × FF;              //configure port 3 as input port
                             //for switches
   while(1)
   {if(!MORE);               //if MORE key is pressed
                             //(decreased frequency)
     {while(!MORE);          //wait till key is released
     if (freq < 1000)        //if freq is less than maximum,
                             //increase it
     freq = freq + 50;
     }                       //end of if
   if (!LESS)                //if less key is pressed
                             //(increase frequency)
     {while(!LESS);          //wait till key is released
     if (freq > 100)         //if freq is more than minimum,
                             //then decrease it
     freq = freq – 50;
     }                       //end of if
   P0 = 0 × ff;              //output high on port
   delay(freq);              //call delay
   P0 = 0 × 00;              //output low on port
   delay(freq);              //call delay
   }                         //end of while
  }                          //end of main
```

Program 11.11

External ADC and temperature control interface to 8051

Temperature control is a common application found every where, from the refrigerator, AC, oven to nuclear reactor. We discuss here a simple on-off temperature control, wherein the relay is turned on/off depending on whether the current temperature is greater/lesser than the set-point. The block diagram of the set-up is shown in Fig. 11.3.

In the above setup the temperature of water (or the temperature of the tip of a soldering iron) is controlled. The water is heated in the electric kettle. The power supply to the kettle is controlled by a relay. If the temperature is greater than the desired (set point) the power supply is cut-off by the relay and vice versa. The temperature is measured by a RTD, which is a transducer. The

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
   unsigned char i=0×80;  //i has the initial speed value=half speed=80H
   P3=0×ff;               //configure P3 to accept switches
while(1)
   {if(!inr)              //if increment is pressed
      {while(!inr);       //wait till key is released
if(i>10)                  //if speed is more than minimum
   i=i-10;                //increase the DC motor speed, by decreasing
                          //the count
   }                      //end of if
   if(!dcr)               //if decrement is pressed
   {
   while(!dcr);           //wait till key is released
   if(i<0xf0)             //decrease the DC motor speed, by increasing
                          //the count
   }                      //end of if
   P0=i;                  //output the value to port P0 for speed control
   }  }                   //end of while and main
```

Program 11.13 Elevator interface to 8051

The elevator interface card explained in the following section simulates the up/down movement of an elevator using LEDs, on giving a floor request by pressing the corresponding floor key. The elevator interface to 8051 is shown in Fig. 11.5.

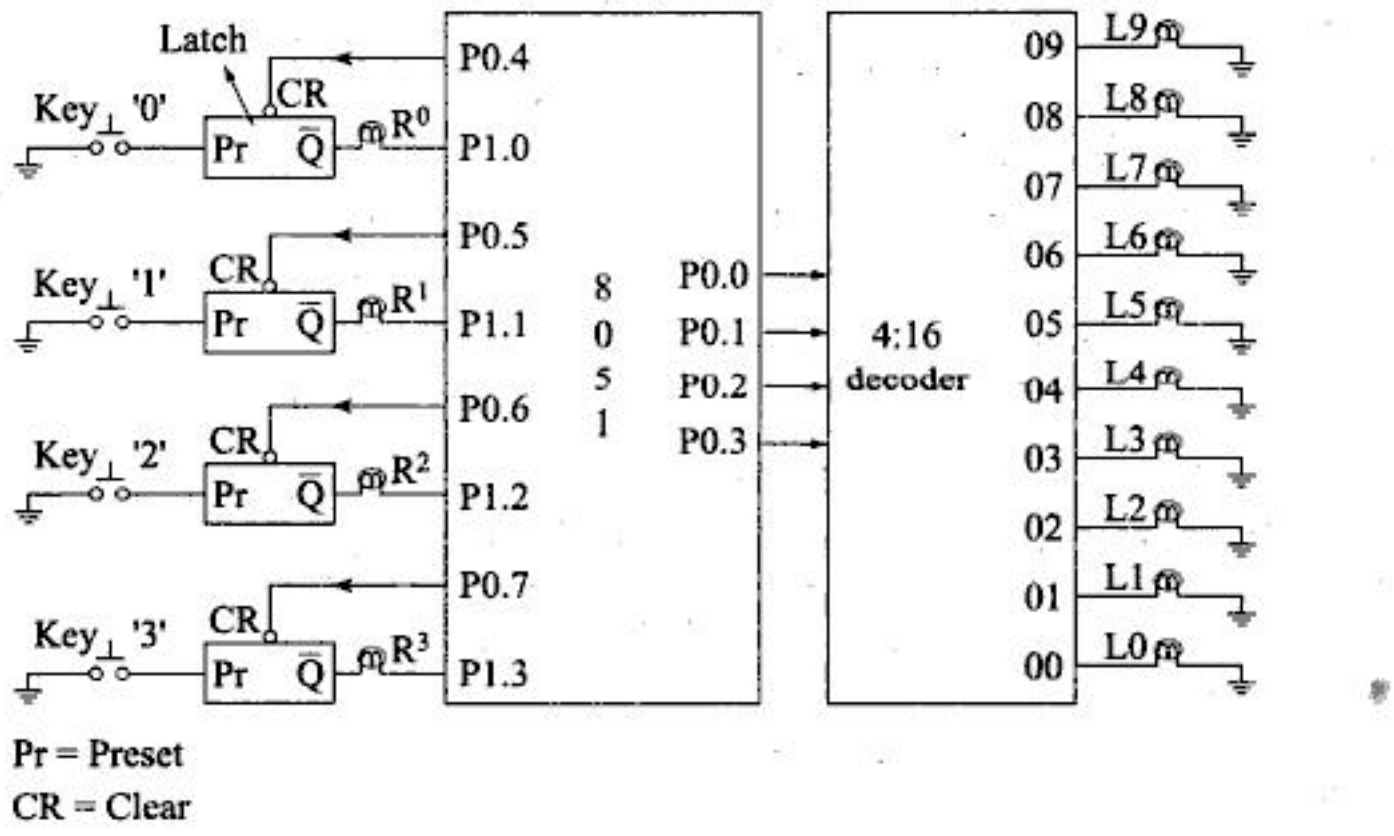


Figure 11.5. Elevator interface to 8051.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
   goto again;                    //start from beginning
   else
   {goto next; }
   }
}
void main()
{P0 = 0xff;                       //make as input port
 P1 = 0x00;
 InitLcd();                       //initialize LCD
 WriteString("KEY PRESSED=")
 while(1)
 {KeyScan();                      //call keypad subroutine
     WriteString("KEY PRESSED=")
Display();                        //display the value
 }                                }
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The CTS and RTS referred to as hardware control flow signals are used to control the flow of the data. When the PC wants to send the data it asserts RTS, and in response if the modem is ready to accept the data it sends CTS. If modem does not activate CTS due to any reason (say lack of room, no connection, etc), the PC de-asserts the DTR and tries again. DTR and DSR are used by the PC and modem to indicate that they are ON.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

B.1. 8051 Instruction set summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
DATA TRANSFER (Continued)				
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

Mnemonic		Description	Byte	Oscillator Period
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24

B.2. Instructions Opcodes in hexadecimal order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A + DPTR
74	2	MOV	A, # data
75	3	MOV	data addr, # data
76	2	MOV	@R0, # data
77	2	MOV	@R1, # data
78	2	MOV	R0, # data
79	2	MOV	R1, # data
7A	2	MOV	R2, # data
7B	2	MOV	R3, # data
7C	2	MOV	R4, # data
7D	2	MOV	R5, # data
7E	2	MOV	R6, # data
7F	2	MOV	R7, # data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A + PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR, # data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A + DPTR
94	2	SUBB	A, # data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A, # data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0, # data,code addr
B7	3	CJNE	@R1, # data,code addr
B8	3	CJNE	R0, # data,code addr
B9	3	CJNE	R1, # data,code addr
BA	3	CJNE	R2, # data,code addr
BB	3	CJNE	R3, # data,code addr
BC	3	CJNE	R4, # data,code addr
BD	3	CJNE	R5, # data,code addr
BE	3	CJNE	R6, # data,code addr
BF	3	CJNE	R7, # data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

References

1. Intel 8051 manual at "http://www.intel.com/design/mcs51/manuals/272383.htm"
2. Ramesh S Gaonkar, "Microprocessor Architecture, Programming and Applications with the 8085", Fourth Edition, Penram International.
3. Kenneth J. Ayala ; "The 8051 Microcontroller Architecture, Programming & applications" 2e, Penram International, 1996/Thomson Learning 2005
4. Muhammad Ali Mazidi and Janice Gillespie Mazidi and Rollin D. McKinlay; "The 8051 Microcontroller and Embedded Systems—using assembly and C "- PHI, 2006/Pearson, 2006
5. Predko ; "Programming and Customizing the 8051 Microcontroller" –, TMH
6. Raj Kamal, "Microcontrollers: Architecture, Programming, Interfacing and System Design", Pearson Education, 2005
7. Ajay V.Deshmukh; "Microcontrollers- Theory and Applications",TMH,2005
8. ESA (Electro Systems Associates) Microprocessor / Microcontroller Trainers and Interface Modules manual.
   http://esaindia.com/prdrng_microprocessor_electro.html#ESAMCB51
9. Dr. Ramani Kalpathi and Ganesh Raja, "Microcontrollers and Applications", 1 rev edn, Sanguine Technical Publishers, 2008

Index

A

ADC 0808/0809, 397
ADC 0848, 399
ADC-analog-to-digital converters, 393
  interfacing ADC 0804 to 8051, 395
  parameters of ADC, 394
Addition, 132
Address bus, 4
Addressing modes, 112
Algorithm, 108
Alphanumeric codes, 40
AND operation, 149
Architecture of 8051, 47
  block diagram of 8051, 48
  clock, 51
  data transmission and reception, 87
  external memory, 60
  I/O Ports, 78
  internal memory, 58
  internal RAM, 58
  internal ROM, 60
  machine cycle, 51, 96
  memory address decoding, 62
  pin diagram, 51
  reading a port, 81
  register banks, 58
  registers of, 53
  registers of 8051, 50
  SBUF register, 87
  serial data transmission modes, 87
  serial input/output, 82
  SFR's, 71
  stack, 69
  TCON register, 77
  timer modes, 75
  TMOD Register, 71
  writing to a Port, 81
Arrays, 185
ASCII, 6
Assembly language, 131, 167
  addition, 132
  CALL instruction, 178
  decrementing, 137
  incrementing, 137
  jump and call instruction, 168
  logical operations, 149
  multiplication, 139
  subtraction, 138
Assembly language, 104
  addressing modes, 103, 112
  data transfer, 103
  execution of the program by 8051, 108
  flow charts, 108
  immediate addressing mode, 113
  indirect addressing mode, 118
  program code in ROM, 107
  programming, 103
  steps to create, 106
  structure of, 105

B

Baud rate, 83
BCD addition, 36
Binary digits, 3
Binary system, 24
Bit jump instructions, 171
Bit level logical operations, 155
Bus, 4
Byte jump instructions, 173

C

Cache memory, 15
CALL instruction, 178
  subroutine, 178
Central processing unit, 2, 4
CISC (complex instruction set computers), 10
Clear accumulator, 154
Code conversions, 218
Complement accumulator, 154
Computer architecture, 16
  Harvard architecture, 17

Von Neumann architecture, 16
Computing languages, 11
Control bus, 5
Counters, 69, 231
application, 251

D

Data bus, 5
Data exchange, 123
Data pointer (DPTR), 54
Data representation, 23
alphanumeric codes, 40
binary system, 24
complements, 29
conversion, 26
decimal fixed-point representation, 36
decimal system, 28
fixed-point representation, 32
floating-point representation, 38
gray code, 39
hexadecimal system, 25
octal system, 25
signed 1's complement representation, 33
signed 2's complement representation, 33
signed integer representation, 33
signed-magnitude representation, 33
Data transfer, 121
Data transmission and reception, 87
Data types in 8051 'C', 185
DC motor interfacing, 385
Decimal addition, 141
Decimal fixed-point representation, 36
Decimal representation, 28
Declaring variables, 184
Decrementing, 137
Delay generation in C, 193
Digital computer, 2
architecture, 2
terminology associated, 3
Direct addressing mode, 115
Division, 140
Dynamic RAM, 13

E

Edge triggered interrupts mode, 313
Elevator interface, 449
EPROM, 15
Exclusive OR operation, 150
External interrupts, 310
edge triggered interrupts mode, 313
level Triggered interrupt mode, 310
serial communication interrupt, 316
8051 Architecture, 47
8051 Microcontroller, 48
block diagram of 8051, 48
clock, 51
data transmission and reception, 87
external memory, 60
I/O Ports, 78
internal memory, 58
internal RAM, 58
internal ROM, 60
machine cycle, 51, 96
memory address decoding, 62
pin diagram, 51
reading a port, 81
register banks, 58
registers of, 53
registers of 8051, 50
SBUF register, 87
serial data transmission modes, 87
serial input/output, 82
SFR's, 71
stack, 69
TCON register, 77
timer modes, 75
TMOD register, 71
writing to a Port, 81

F

Firmware, 3
Fixed-point representation, 32
Flag register, 54
Floating-point representation, 38
Flow charts, 108

G

Generation of square wave, 236
Gray code, 39

H

Hardware, 3
Harvard architecture, 17
Hexadecimal system, 25
High-level languages, 104

I

Immediate addressing mode, 113
Incrementing, 137
Indexed addressing mode, 120
Indirect addressing mode, 118
Interfacing, 335
DC motor interfacing, 385
interfacing a DAC, 375

interfacing a LED and a 7-segment display, 336
interfacing a single key, 355
interfacing ADC 0804 to 8051, 395
interfacing keyboard, 360
stepper motor interfacing, 365
Interfacing a DAC, 375
Interfacing ADC 0804 to 8051, 395
Interrupt destination, 92
Interrupt enable (IE) SFR, 93
Interrupt priority (IP) SFR, 94
Interrupts, 91, 293
changing interrupt priority, 324
execution of an interrupt, 296
external interrupts, 310
serial communication interrupt, 316

J

Jump and call instruction, 168
long absolute Range, 170
relative range, 168
short absolute range, 169
Jump Instructions, 170
bit jump instructions, 171
byte jump instructions, 173
unconditional Jump, 170

L

LCD display, 341
Level triggered interrupt mode, 310
Logical operations, 149
AND operation, 149
exclusive OR operation, 150
OR operation, 150
Long absolute range, 170

M

Machine cycles, 96
Mainframes, 3
Matrix keypad, 360
Memory, 4, 12
Memory address decoding, 62
Memory Latency, 15
Memory unit, 2
Microcontroller
algorithm for DC motor interface, 448
application in mode 2, 246
assembly language, 167
CALL instruction, 178
changing interrupt priority, 324
code space, 226
creating and compiling a μVision2 project, 409
DC motor interfacing, 385
elevator interface, 449
execution of an interrupt, 296
external interrupts, 310
interfacing a DAC, 375
interfacing a LED and a 7-segment display, 336
interfacing a single key, 355
interfacing ADC 0804 to 8051, 395
interfacing keyboard, 360
interrupts, 294
jump and call instruction, 168
large time delays, 243
operators in 8051C, 207
procedure for doubling the baud rate, 277
programming in 'C', 184
programming ports, 198
second serial port, 281
serial communication interrupt, 316
serial data transfer, 264
serial port, 231
serial port programming, 213
stepper motor interfacing, 365
stepper motor interfacing to 8051, 437
timer mode 0 programming, 238
Microcontroller, 47, 131
addition, 132
block diagram of 8051, 48
decrementing, 137
incrementing, 137
logical operations, 149
multiplication, 139
programming, 131
registers of 8051, 50
rotate operation, 158
subtraction, 138
swap operation, 162
Microcontrollers, 7
4-bit microcontrollers, 19
8-bit microcontrollers, 19
applications, 9
evolution of, 18
selection of, 18
Microprocessor, 2, 7
Minicomputer, 3
Multi-tasking, 7
Multiplexed 7-segment display, 339

N

Number system, 24
base, 24

binary system, 24
complements, 29
conversion, 26
decimal fixed-point representation, 36
decimal system, 28
floating-point representation, 38
gray code, 39
hexadecimal system, 25
octal system, 25
radix, 24
signed 1's complement representation, 33
signed 2's complement representation, 33
signed integer representation, 33
signed-magnitude representation, 33

O
Octal system, 25
Operating system, 6
Operators in 8051C, 207
OR operation, 150
Overflow, 35

P
Polling sequence, 93
POP instruction, 122
Ports, 5
Power mode control (PCON) SFR, 85
Princeton architecture, 16
Program counter, 54
Programmable read-only memory, 14
Programming 8051 with C, 183
  arrays, 185
  code space, 226
  data types in 8051 'C', 185
  declaring variables, 184
  delay generation in C, 193
  number representation, 187
  operators in 8051C, 207
  programming ports, 198
  serial port programming, 213
  strings, 185
  writing a simple C program, 187
PUSH instruction, 121
PWM control of DC motor, 391

R
$(r-1)$'s complement, 29
r's complement, 30
  r's complement, 30
  subtraction of unsigned numbers, 31
Random access memory, 12
Rate of transmission, 83
Read only memory, 14
Register addressing mode, 114
Register section, 5
Registers of 8051, 53
  A and B registers, 54
  data pointer, 54
  program counter, 54
  program status word, 54
  special function registers, 57
Relative range, 168
RISC (reduced instruction set computers), 10
Rotate operation, 158
  rotate accumulator right, 159
  rotate through the carry, 160

S
SBUF register, 87
Serial ADCs, 401
Serial communication interrupt, 316
Serial data transfer, 264
Serial data transmission modes, 87
Serial input/output, 82
Serial port control (SCON) register, 84
Serial port programming, 213
Serialize data, 161
Short absolute range, 169
Signed addition, 136
Signed integer representation, 33
Signed numbers, 34
  arithmetic addition, 34
  arithmetic subtraction, 35
Simulation of 8051 using Keil Software, 407
  algorithm for DC motor interface, 448
  arithmetic instructions, 417
  conversion programs, 432
  counters, 426
  creating and compiling a μVision2 project, 409
  data transfer instructions, 411
  elevator interface, 449
  interrupts, 408
  logical instructions, 422
  program illustrating bit manipulations, 421
  programming in ALP, 411
  serial data transmission, 428
  stepper motor interfacing to 8051, 437
  timer delay program, 429
Software, 3
Stack, 69
Stack pointer, 69
Standards in serial I/O, 84
Stepper motor interfacing, 365
Stepper motor interfacing to 8051, 437
Strings, 185

Subroutine, 178
Supercomputers, 3
Swap operation, 162

T
TCON register, 77
Time sharing, 6
Timer mode 0 programming, 238
Timer modes, 75
Timers, 69, 231
  application in mode 2, 246
  generation of square wave, 236
  large time delays, 243
  time delay generation, 232
TMOD register, 71

U
Unconditional jump, 170
Unit-distance codes, 39

V
Von Neumann architecture, 16

The 8051 Microcontrollers

Architecture, Programming and Applications

K. Uma Rao
Andhe Pallavi

The 8051 is a widely used microcontroller due to its simple architecture and ease of programming. This book presents in detail the architecture and programming of 8051 in both assembly language and in C, and discussed some typical applications of 8051 in detail. The lucid explanation of the subject is supported by a number of programming examples in assembly language and in C to strengthen the reader's comprehension. The fundamental concepts presented in the text will enable the reader to understand any other microcontroller available in the market with ease. Though designed for a undergraduate semester course on 8051, anyone interested in the subject will find this book an invaluable guide to the subject.

K. Uma Rao is the head of the Department of Electronics and Communication at R.N. Shetty Institute of Technology, Bengaluru.

Andhe Pallavi is the head of the Department of Instrumentation Technology at R.N. Shetty Institute of Technology, Bengaluru.

SANGUINE™
www.sanguineindia.com

ISBN 978-81-317-3252-6

www.pearsoned.co.in

ALWAYS LEARNING

PEARSON