CHAPTER 1

Microcontroller Systems

1.1 Introduction

The term *microcontroller* or *microcomputer* is used to describe a system that includes a minimum of a microprocessor, program memory, data memory, and input–output (I/O). Some microcontroller systems include additional components, such as timers, counters, analog-to-digital (A/D) converters, and so on. Thus, a microcontroller system can be anything from a large computer having hard disks, floppy disks, and printers to a single-chip embedded controller.

In this book, we are going to consider only the type of microcontrollers that consist of a single silicon chip. Such microcontroller systems are also known as *embedded controllers*, and they are used in office equipment like PCs, printers, scanners, copy machines, digital telephones, fax machines, and sound recorders. Microcontrollers are also used in household goods, such as microwave ovens, TV remote control units, cookers, hi-fi equipment, CD players, personal computers, and fridges. Many microcontrollers are available in the market. In this book, we shall look at programming and system design using the programmable interface controller (PIC) series of microcontrollers manufactured by Microchip Technology Inc.

1.2 Microcontroller Systems

A microcontroller is a single-chip computer. *Micro* suggests that the device is small and *controller* suggests that the device can be used in control applications. Another term used for microcontrollers is *embedded controller*, because most of the microcontrollers are built into (or embedded in) the devices they control. For example, microcontrollers with dedicated programs are used in washing machines to control the washing cycles.

A microprocessor differs from a microcontroller in many ways. The main difference is that a microprocessor requires several other external components for its operation, such as program memory and data memory, I/O devices, and an external clock circuit. In general, a microprocessor-based system usually consists of several supporting chips interconnected and operating together. The power consumption and the cost of a microprocessor-based system are, thus, usually high. A microcontroller on the other hand has all the support chips incorporated inside the same chip. All microcontrollers operate on a set of instructions (or the user

D.O.I.: 10.1016/B978-1-85617-719-1.00005-1

program) stored in their memory. A microcontroller fetches the instructions from its program memory one by one, decodes these instructions, and then carries out the required operations.

Microcontrollers have traditionally been programmed using the assembly language of the target device. Although assembly language is fast, it has several disadvantages. An assembly program consists of mnemonics, and it is difficult to learn and maintain a program written using assembly language. Also, microcontrollers manufactured by different firms have different assembly languages, and the user is required to learn a new language every time a new microcontroller is to be used. Microcontrollers can also be programmed using one of the traditional high-level languages, such as Basic, Pascal, or C. The advantage of high-level language is that it is much easier to learn than an assembler. Also, very large and complex programs can easily be developed using a high-level language. For example, it is rather a complex task to multiply two floating point numbers using assembly language. The similar operation, however, is much easier and consists of a single statement in a high-level language. In this book, we shall be learning the programming of PIC microcontrollers using the popular C18 high-level C programming language developed by Microchip Inc.

In general, a single chip is all that is required to have a running microcontroller system. In practical applications, additional components may be required to allow a microcomputer to interface to its environment. With the advent of the PIC family of microcontrollers, the development time of a complex electronic project has been reduced from many days to several hours.

Basically, a microcomputer executes a user program that is loaded in its program memory. Under the control of this program, data is received from external devices (inputs), manipulated, and then sent to external devices (outputs). For example, in a simple microcontroller-based temperature data logging system, the temperature is read by the microcomputer using a temperature sensor. The microcomputer then saves the temperature data on an SD card at predefined intervals. Figure 1.1 shows the block diagram of our simple temperature data logging system.

The system shown in Figure 1.1 is a very simplified temperature data logger system. In a more sophisticated system, we may have a keypad to set the logging interval and an

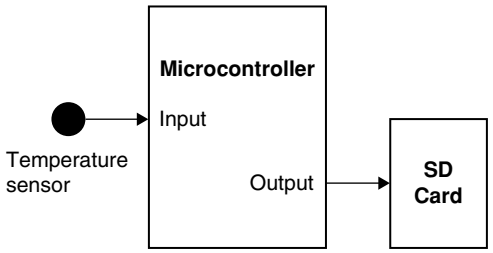


Figure 1.1: Microcontroller-Based Temperature Data Logger System

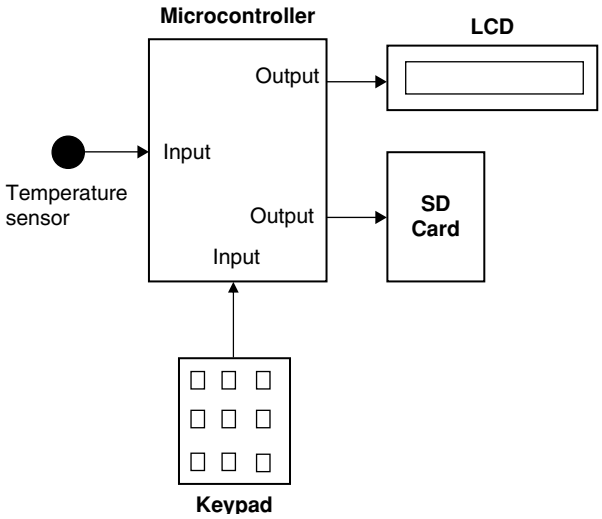


Figure 1.2: Temperature Data Logger System with a Keypad and LCD

LCD to display the current temperature. Figure 1.2 shows the block diagram of this more sophisticated temperature data logger system.

We can make our design even more sophisticated (see Figure 1.3) by adding a real-time clock chip (RTC) to provide the absolute date and time information so that the data can be saved with date and time stamping. Also, the temperature readings can be sent to a PC every second for archiving and further processing. For example, a graph of the temperature change can be

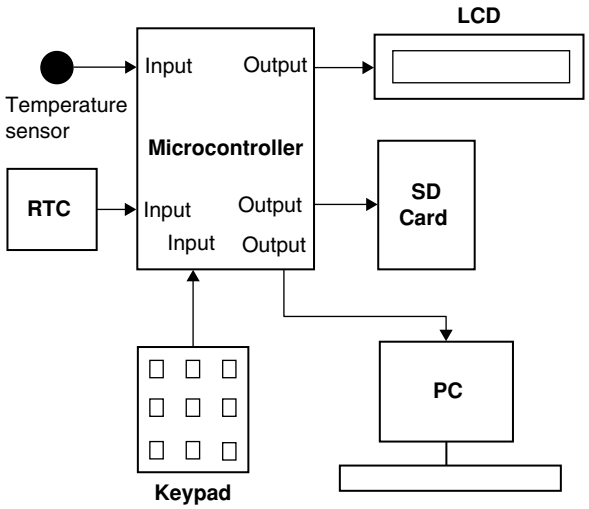


Figure 1.3: More Sophisticated Temperature Data Logger

plotted on the PC. As you can see, because the microcontrollers are programmable, it is very easy to make the final system as simple or as complicated as we like.

A microcontroller is a very powerful electronic device that allows a designer to create sophisticated I/O data manipulation under program control. Microcontrollers are classified by the number of bits they process. Eight-bit microcontrollers are the most popular ones and are used in most microcontroller-based monitoring and control applications. Microcontrollers of 16 and 32 bits are much more powerful but usually more expensive and not required in many small-to-medium-size, general-purpose applications where microcontrollers are generally used.

The simplest microcontroller architecture consists of a microprocessor, program and data memory, and I/O circuitry. The microprocessor itself consists of a central processing unit (CPU) and the control unit (CU). The CPU is the brain of the microprocessor, where all the arithmetic and logic operations are performed. The CU controls the internal operations of the microprocessor and sends out control signals to other parts of the microprocessor to carry out the required instructions.

Memory is an important part of a microcontroller system. Depending upon the type used, we can classify memory into two groups: program memory and data memory. Program memory stores the application program written by the programmer and is usually nonvolatile; i.e., data is not lost after the removal of power. Data memory is where the temporary data used in a program is stored and is usually volatile; i.e., data is lost after the removal of power.

There are basically six types of memory, as summarized below.

1.2.1 Random Access Memory

Random access memory (RAM) is a general-purpose memory that usually stores the user data in a program. RAM is volatile in the sense that it cannot retain data in the absence of power; i.e., data is lost after the removal of power. The RAM in a system is either static RAM (SRAM) or dynamic RAM (DRAM). The SRAMs are fast, with access time in the range of a few nanoseconds, which makes them ideal memory chips in computer applications. DRAMs are slower and because they are capacitor based they require refreshing every several milliseconds. DRAMs have the advantage that their power consumption is less than that of SRAMs. Most microcontrollers have some amount of internal RAM, commonly 256 bytes, although some microcontrollers have more and some have less. For example, the PIC18F452 microcontroller has 1536 bytes of RAM, which should be enough for most microcontroller-based applications. In most microcontroller systems, it is possible to extend the amount of RAM by adding external memory chips if desired.

1.2.2 Read Only Memory

Read only memory (ROM) is a type of memory that usually holds the application program or fixed user data. ROM is nonvolatile. If power is removed from ROM and then reapplied, the original data will still be there. ROMs are programmed at the factory during the manufacturing process and their content cannot be changed by the user. ROMs are only useful if you have developed a microcontroller-based application and wish to order several thousand microcontroller chips preprogrammed with this program.

1.2.3 Programmable Read Only Memory

Programmable read only memory (PROM) is a type of ROM that can be programmed in the field, often by the end user, using a device called a PROM programmer. PROM is used to store an application program or constant data. Once a PROM has been programmed, its contents cannot be changed again. PROMs are usually used in low production applications where only several such memories are required.

1.2.4 Erasable Programmable Read Only Memory

Erasable programmable read only memory (EPROM) is similar to ROM, but the EPROM can be programmed using a suitable programming device. EPROMs have a small clear glass window on top of the chip where the data can be erased under strong ultraviolet light. Once the memory is programmed, the window should be covered with dark tape to prevent accidental erasure of the data. An EPROM must be erased before it can be reprogrammed. Many development versions of microcontrollers are manufactured with EPROMs where the user program can be stored. These memories are erased and reprogrammed until the user is satisfied with the program. Some versions of EPROMs, known as one time programmable (OTP) EPROMs, can be programmed using a suitable programmer device, but these memories cannot be erased. OTP memories cost much less than EPROMs. OTP is useful after a project has been developed completely, and it is required to make many copies of the final program memory.

1.2.5 Electrically Erasable Programmable Read Only Memory

Electrically erasable programmable read only memory (EEPROM) is a nonvolatile memory. These memories can be erased and can also be reprogrammed using suitable programming devices. EEPROMs are used to save constant data, such as configuration information, maximum and minimum values of a measurement, and identification data. Some microcontrollers have built-in EEPROMs. For example, PIC18F452 contains a 256-byte EEPROM where each byte can be programmed and erased directly by applications software. EEPROMs are usually very slow. The cost of an EEPROM chip is much higher than that of an EPROM chip.

1.2.6 Flash EEPROM

Flash EEPROM is another version of EEPROM type memory. This memory has become popular in microcontroller applications and is used to store the user program. Flash EEPROM is nonvolatile and is usually very fast. The data can be erased and then reprogrammed using a suitable programming device. Some microcontrollers have only 1K of flash EEPROM, while some others have 32 K or more. The PIC18F452 microcontroller has 32 KB of flash memory.

1.3 Microcontroller Features

Microcontrollers from different manufacturers have different architectures and different capabilities. Some may suit a particular application while others may be totally unsuitable for the same application. Some of the hardware features of microcontrollers in general are described in this section.

1.3.1 Buses

The connections between various blocks of a computer system are called buses. A bus is a common set of wires that carry a specific type of information. In general, every computer system has three buses: address bus, data bus, and control bus.

An address bus carries the address information in a computer system. It is a unidirectional bus having 16 bits in small computer systems and 32 or more bits in larger systems. An address bus usually carries the memory addresses from the CPU to the memory chips. This bus is also used to carry the I/O addresses in many computer systems.

A data bus carries the data in a computer system. It is a bidirectional bus having 8 bits in small systems and 14, 16, 32, or even more bits in larger systems. A data bus carries the memory data from the CPU to the memory chips. In addition, data is carried to other parts of a computer via the data bus.

The control bus is usually a smaller bus and is used to provide control signals to most parts of a computer system. For example, memory read and write control signals are carried by the control bus.

1.3.2 Supply Voltage

Most microcontrollers operate with the standard logic voltage of +5 V. Some microcontrollers can operate at as low as +2.7 V and some will tolerate +6 V without any problems. You should check the manufacturers' data sheets about the allowed limits of the power supply voltage. For example, PIC18F452 microcontrollers can operate with a power supply +2 to +5.5 V.

A voltage regulator circuit is usually used to obtain the required power supply voltage when the device is to be operated from a mains adaptor or batteries. For example, a 5-V regulator is required if the microcontroller is to be operated using a 9-V battery.

1.3.3 The Clock

All microcontrollers require a clock (or an oscillator) to operate. The clock is usually provided by connecting external timing devices to the microcontroller. Most microcontrollers will generate clock signals when a crystal and two small capacitors are connected. Some will operate with resonators or external resistor-capacitor pair. Some microcontrollers have built-in timing circuits and they do not require any external timing components. If the application is not time sensitive, then external or internal (if available) resistor-capacitor timing components should be used to lower the costs.

An instruction is executed by fetching it from the memory and then decoding it. This usually takes several clock cycles and is known as the *instruction cycle*. In PIC microcontrollers, an instruction cycle takes four clock periods. Thus, the microcontroller is actually operated at a clock rate, which is a quarter of the actual oscillator frequency. For example, in a PIC microcontroller operating at 4-MHz clock, the instruction cycle time is only 1 μs (frequency of 1 MHz). The PIC18F series of microcontrollers can operate with clock frequencies up to 40 MHz.

1.3.4 Timers

Timers are important parts of any microcontroller. A timer is basically a counter, which is driven either by an external clock pulse or by the internal oscillator of the microcontroller. A timer can be 8 or 16 bits wide. Data can be loaded into a timer under program control and the timer can be stopped or started by program control. Most timers can be configured to generate an interrupt when they reach a certain count (usually when they overflow). The interrupt can be used by the user program to carry out accurate timing-related operations inside the microcontroller. The PIC18F series of microcontrollers have at least three timers. For example, the PIC18F452 microcontroller has three built-in timers.

Some microcontrollers offer capture and compare facilities where a timer value can be read when an external event occurs or the timer value can be compared to a preset value and an interrupt generated when this value is reached. Most PIC18F microcontrollers have at least two capture and compare modules.

1.3.5 Watchdog

Most microcontrollers have at least one watchdog facility. The watchdog is basically a timer that is normally refreshed by the user program, and a reset occurs if the program fails to refresh the watchdog. The watchdog timer is used to detect serious problems in programs,

such as the program being in an endless loop. A watchdog is a safety feature that prevents runaway software and stops the microcontroller from executing meaningless and unwanted code. Watchdog facilities are commonly used in real-time systems where it is required to regularly check the successful termination of one or more activities.

1.3.6 Reset Input

A reset input is used to reset a microcontroller externally. Resetting puts the microcontroller into a known state such that the program execution starts usually from address 0 of the program memory. An external reset action is usually achieved by connecting a push-button switch to the reset input such that the microcontroller can be reset when the switch is pressed.

1.3.7 Interrupts

Interrupts are very important concepts in microcontrollers. An interrupt causes the microcontroller to respond to external and internal (e.g., a timer) events very quickly. When an interrupt occurs, the microcontroller leaves its normal flow of program execution and jumps to a special part of the program known as the interrupt service routine (ISR). The program code inside the ISR is executed and upon return from the ISR the program resumes its normal flow of execution.

The ISR starts from a fixed address of the program memory. This address is known as the *interrupt vector address*. Some microcontrollers with multi-interrupt features have just one interrupt vector address, while some others have unique interrupt vector addresses, one for each interrupt source. Interrupts can be nested such that a new interrupt can suspend the execution of another interrupt. Another important feature of a microcontroller with multi-interrupt capability is that different interrupt sources can be given different levels of priority. For example, the PIC18F series of microcontrollers have low-priority and high-priority interrupt levels.

1.3.8 Brown-Out Detector

Brown-out detectors are also common in many microcontrollers, and they reset a microcontroller if the supply voltage falls below a nominal value. Brown-out detectors are safety features, and they can be employed to prevent unpredictable operation at low voltages, especially to protect the contents of EEPROM type memories if the supply voltage falls.

1.3.9 A/D Converter

An A/D converter is used to convert an analog signal like voltage to digital form so that it can be read and processed by a microcontroller. Some microcontrollers have built-in A/D converters. It is also possible to connect an external A/D converter to any type of

microcontroller. A/D converters are usually 8–10 bits having 256–1024 quantization levels. Most PIC microcontrollers with A/D features have multiplexed A/D converters where more than one analog input channel is provided. For example, the PIC18F452 microcontroller has 10-bit, 8-channel A/D converters.

The A/D conversion process must be started by the user program and it may take several hundreds of microseconds for a conversion to complete. A/D converters usually generate interrupts when a conversion is complete so that the user program can read the converted data quickly.

A/D converters are very useful in control and monitoring applications because most sensors (e.g., temperature sensor, pressure sensor, and force sensor) produce analog output voltages that cannot be read by a microcontroller without an A/D converter.

1.3.10 Serial I/O

Serial communication (also called RS232 communication) enables a microcontroller to communicate with other devices using the serial RS232 communication protocol. For example, a microcontroller can be connected to another microcontroller or to a PC and exchange data using the serial communication protocol. Some microcontrollers have built-in hardware called universal synchronous-asynchronous receiver-transmitter (USART) to implement a serial communication interface. The baud rate and the data format can usually be selected by the user program. If serial I/O hardware is not provided, it is easy to develop software to implement the serial data communication using any I/O pin of a microcontroller. The PIC18F series of microcontrollers have built-in USART modules.

Some microcontrollers (e.g., PIC18F series) incorporate a serial peripheral interface (SPI) or an integrated interconnect (I^2C) hardware bus interface. These enable a microcontroller to interface to other compatible devices easily.

1.3.11 EEPROM Data Memory

EEPROM type data memory is also very common in many microcontrollers. The advantage of an EEPROM is that the programmer can store nonvolatile data in such a memory and can also change this data whenever required. For example, in a temperature monitoring application, the maximum and the minimum temperature readings can be stored in an EEPROM. Then, if the power supply is removed for whatever reason, the values of the latest readings will still be available in the EEPROM. The PIC18F452 microcontroller has 256 bytes of EEPROM. Some other members of the family have more (e.g., PIC18F6680 has 1024 bytes) EEPROMs.

1.3.12 LCD Drivers

LCD drivers enable a microcontroller to be connected to an external LCD display directly. These drivers are not common because most of the functions they provide can be implemented in the software. For example, the PIC18F6490 microcontroller has a built-in LCD driver module.

1.3.13 Analog Comparator

Analog comparators are used where it is required to compare two analog voltages. Although these circuits are implemented in most high-end PIC microcontrollers, they are not common in other microcontrollers. The PIC18F series of microcontrollers have built-in analog comparator modules.

1.3.14 Real-Time Clock

Real-time clock (RTC) enables a microcontroller to have absolute date and time information continuously. Built-in real-time clocks are not common in most microcontrollers because they can easily be implemented by either using a dedicated RTC or by writing a program.

1.3.15 Sleep Mode

Some microcontrollers (e.g., PIC) offer built-in sleep modes where executing this instruction puts the microcontroller into a mode where the internal oscillator is stopped and the power consumption is reduced to an extremely low level. The main reason for using the sleep mode is to conserve the battery power when the microcontroller is not doing anything useful. The microcontroller usually wakes up from the sleep mode by external reset or by a watchdog time-out.

1.3.16 Power-on Reset

Some microcontrollers (e.g., PIC) have built-in power-on reset circuits, which keep the microcontroller in reset state until all the internal circuitry has been initialized. This feature is very useful as it starts the microcontroller from a known state on power-up. An external reset can also be provided where the microcontroller can be reset when an external button is pressed.

1.3.17 Low-Power Operation

Low-power operation is especially important in portable applications where the microcontroller-based equipment is operated from batteries. Some microcontrollers (e.g., PIC) can operate with less than 2 mA at a 5-V supply and approximately 15 μA at a 3-V supply. Some other microcontrollers, especially microprocessor-based systems where there could be several chips, may consume several hundred milliamperes or even more.

1.3.18 Current Sink/Source Capability

This is important if the microcontroller is to be connected to an external device that may draw large current for its operation. PIC microcontrollers can source and sink 25 mA of current from each output port pin. This current is usually sufficient to drive light-emitting diodes (LEDs), small lamps, buzzers, small relays, etc. The current capability can be increased by connecting external transistor switching circuits or relays to the output port pins.

1.3.19 USB Interface

USB is currently a very popular computer interface specification used to connect various peripheral devices to computers and microcontrollers. Some PIC microcontrollers provide built-in USB modules. For example, PIC18F2X50 has built-in USB interface capabilities.

1.3.20 Motor Control Interface

Some PIC microcontrollers (e.g., PIC18F2X31) provide motor control interface.

1.3.21 Controller Area Network Interface

Controller area network (CAN) bus is a very popular bus system used mainly in automation applications. Some PIC18F series of microcontrollers (e.g., PIC18F4680) provide CAN interface capabilities.

1.3.22 Ethernet Interface

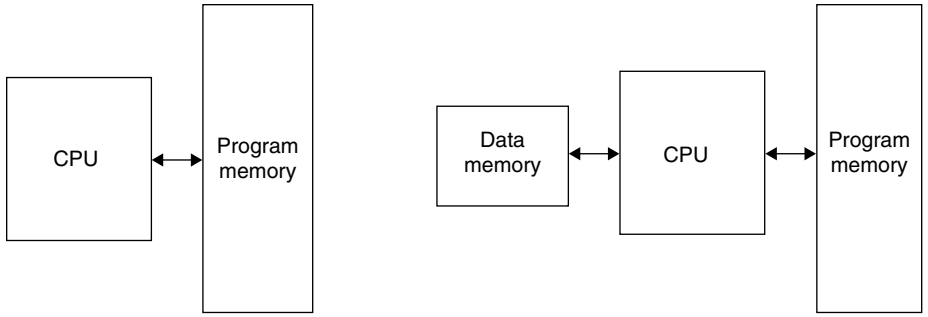
Some PIC microcontrollers (e.g., PIC18F97J60) provide Ethernet interface capabilities. Such microcontrollers can easily be used in network-based applications.

1.3.23 ZigBee Interface

ZigBee is an interface similar to Bluetooth and is used in low-cost wireless home automation applications. Some PIC18F series of microcontrollers provide ZigBee interface capabilities, making the design of such wireless systems very easy.

1.4 Microcontroller Architectures

Usually two types of architecture are used in microcontrollers (see Figure 1.4): *Von Neumann architecture* and *Harvard architecture*. Von Neumann architecture is used by a large percentage of microcontrollers, where all memory space is on the same bus and instruction and data use the same bus. In the Harvard architecture (used by the PIC microcontrollers),



(a) Von Neumann architecture (b) Harvard architecture

Figure 1.4: Von Neumann and Harvard Architectures

code and data are on separate buses, and this allows the code and data to be fetched simultaneously, resulting in an improved performance.

1.4.1 Reduced Instruction Set Computer and Complex Instruction Set Computer

Reduced instruction set computer (RISC) and complex instruction set computer (CISC) refer to the instruction set of a microcontroller. In an 8-bit RISC microcontroller, data is 8 bits wide but the instruction words are more than 8 bits wide (usually 12, 14, or 16 bits), and the instructions occupy one word in the program memory. Thus, the instructions are fetched and executed in one cycle, resulting in an improved performance.

In a CISC microcontroller, both data and instructions are 8 bits wide. CISC microcontrollers usually have over 200 instructions. Data and code are on the same bus and cannot be fetched simultaneously.

1.5 Choosing a PIC Microcontroller

Choosing a microcontroller for an application requires taking into account the following factors:

- Microcontroller speed
- The number of I/O pins required
- The peripheral devices required (e.g., USART and A/D converter)
- The memory size (RAM, flash, EEPROM, etc.)
- Power consumption
- Physical size

1.6 Number Systems

The efficient use of a microprocessor or a microcontroller requires a working knowledge of binary, decimal, and hexadecimal numbering systems. This section provides a background for those who are unfamiliar with these numbering systems and who do not know how to convert from one number system to another one.

Number systems are classified according to their bases. The numbering system used in everyday life is base 10 or the decimal number system. The most commonly used numbering system in microprocessor and microcontroller applications is base 16 or hexadecimal. In addition, base 2 (binary) or base 8 (octal) number systems are also used.

1.6.1 Decimal Number System

As you all know, the numbers in this system are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. We can use the subscript 10 to indicate that a number is in decimal format. For example, we can show the decimal number 235 as 235_{10}.

In general, a decimal number is represented as follows:

$$a_n \times 10^n + a_{n-1} \times 10^{n-1} + a_{n-2} \times 10^{n-2} + \cdots + a_0 \times 10^0$$

For example, decimal number 825_{10} can be shown as follows:

$$825_{10} = 8 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

Similarly, decimal number 26_{10} can be shown as follows:

$$26_{10} = 2 \times 10^1 + 6 \times 10^0$$

or

$$3359_{10} = 3 \times 10^3 + 3 \times 10^2 + 5 \times 10^1 + 9 \times 10^0$$

1.6.2 Binary Number System

In the binary number system, there are two numbers: 0 and 1. We can use the subscript 2 to indicate that a number is in binary format. For example, we can show binary number 1011 as 1011_2.

In general, a binary number is represented as follows:

$$a_n \times 2^n + a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + \cdots + a_0 \times 2^0$$

For example, binary number 1110_2 can be shown as follows:

$$1110_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

Similarly, binary number 10001110_2 can be shown as follows:

$$10001110_2 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

1.6.3 Octal Number System

In the octal number system, the valid numbers are 0, 1, 2, 3, 4, 5, 6, and 7. We can use the subscript 8 to indicate that a number is in octal format. For example, we can show octal number 23 as 23_8.

In general, an octal number is represented as follows:

$$a_n \times 8^n + a_{n-1} \times 8^{n-1} + a_{n-2} \times 8^{n-2} + \cdots + a_0 \times 8^0$$

For example, octal number 237_8 can be shown as follows:

$$237_8 = 2 \times 8^2 + 3 \times 8^1 + 7 \times 8^0$$

Similarly, octal number 1777_8 can be shown as follows:

$$1777_8 = 1 \times 8^3 + 7 \times 8^2 + 7 \times 8^1 + 7 \times 8^0$$

1.6.4 Hexadecimal Number System

In the hexadecimal number system, the valid numbers are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. We can use the subscript 16 or H to indicate that a number is in hexadecimal format. For example, we can show hexadecimal number 1F as $1F_{16}$ or as $1F_H$.

In general, a hexadecimal number is represented as follows:

$$a_n \times 16^n + a_{n-1} \times 16^{n-1} + a_{n-2} \times 16^{n-2} + \cdots + a_0 \times 16^0$$

For example, hexadecimal number $2AC_{16}$ can be shown as follows:

$$2AC_{16} = 2 \times 16^2 + 10 \times 16^1 + 12 \times 16^0$$

Similarly, hexadecimal number $3FFE_{16}$ can be shown as follows:

$$3FFE_{16} = 3 \times 16^3 + 15 \times 16^2 + 15 \times 16^1 + 14 \times 16^0$$

1.7 Converting Binary Numbers into Decimal

To convert a binary number into decimal, write the number as the sum of the powers of 2.

Example 1.1

Convert binary number 1011_2 into decimal.

Solution

Write the number as the sum of the powers of 2:

$$1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$
$$= 8 + 0 + 2 + 1$$
$$= 11$$

or $1011_2 = 11_{10}$.

■

Example 1.2

Convert binary number 11001110_2 into decimal.

Solution

Write the number as the sum of the powers of 2:

$$11001110_2 = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$
$$= 128 + 64 + 0 + 0 + 8 + 4 + 2 + 0$$
$$= 206$$

or $11001110_2 = 206_{10}$.

Table 1.1 shows the binary equivalent of decimal numbers from 0 to 31.

Table 1.1: Binary Equivalent of Decimal Numbers

Decimal	Binary	Decimal	Binary
0	00000000	16	00010000
1	00000001	17	00010001
2	00000010	18	00010010
3	00000011	19	00010011
4	00000100	20	00010100

—cont'd

Table 1.1: Binary Equivalent of Decimal Numbers —cont'd

Decimal	Binary	Decimal	Binary
5	00000101	21	00010101
6	00000110	22	00010110
7	00000111	23	00010111
8	00001000	24	00011000
9	00001001	25	00011001
10	00001010	26	00011010
11	00001011	27	00011011
12	00001100	28	00011100
13	00001101	29	00011101
14	00001110	30	00011110
15	00001111	31	00011111

■

1.8 Converting Decimal Numbers into Binary

To convert a decimal number into binary, divide the number repeatedly by 2 and take the remainders. The first remainder is the least significant digit (LSD) and the last remainder is the most significant digit (MSD).

■ Example 1.3

Convert decimal number 28_{10} into binary.

Solution

Divide the number by 2 repeatedly and take the remainders:

$28/2 \rightarrow 14$ Remainder 0 (LSD)

$14/2 \rightarrow 7$ Remainder 0

$7/2 \rightarrow 3$ Remainder 1

$3/2 \rightarrow 1$ Remainder 1

$1/2 \rightarrow 0$ Remeinder 1 (MSD)

The required binary number is 11100_2.

■

Example 1.4

Convert decimal number 65_{10} into binary.

Solution

Divide the number by 2 repeatedly and take the remainders:

$65/2 \rightarrow 32$ Remainder 1 (LSD)
$32/2 \rightarrow 16$ Remainder 0
$16/2 \rightarrow 8$ Remainder 0
$8/2 \rightarrow 4$ Remainder 0
$4/2 \rightarrow 2$ Remainder 0
$2/2 \rightarrow 1$ Remainder 0
$1/2 \rightarrow 0$ Remainder 1 (MSD)

The required binary number is 1000001_2.

■

Example 1.5

Convert decimal number 122_{10} into binary.

Solution

Divide the number by 2 repeatedly and take the remainders:

$122/2 \rightarrow 61$ Remainder 0 (LSD)
$61/2 \rightarrow 30$ Remainder 1
$30/2 \rightarrow 15$ Remainder 0
$15/2 \rightarrow 7$ Remainder 1
$7/2 \rightarrow 3$ Remainder 1
$3/2 \rightarrow 1$ Remainder 1
$1/2 \rightarrow 0$ Remainder 1 (MSD)

The required binary number is 1111010_2.

■

1.9 Converting Binary Numbers into Hexadecimal

To convert a binary number into hexadecimal, arrange the number in groups of four and find the hexadecimal equivalent of each group. If the number cannot be divided exactly into groups of four, insert zeroes to the left-hand side of the number.

Example 1.6

Convert binary number 10011111_2 into hexadecimal.

Solution

First, divide the number into groups of four and then find the hexadecimal equivalent of each group:

$$\underset{}{10011111} = \underset{9}{1001}\ \underset{F}{1111}$$

The required hexadecimal number is $9F_{16}$. ■

Example 1.7

Convert binary number 1110111100001110_2 into hexadecimal.

Solution

First, divide the number into groups of four and then find the equivalent of each group:

$$\underset{}{1110111100001110} = \underset{E}{1110}\ \underset{F}{1111}\ \underset{0}{0000}\ \underset{E}{1110}$$

The required hexadecimal number is $EF0E_{16}$. ■

Example 1.8

Convert binary number 111110_2 into hexadecimal.

Solution

Because the number cannot be divided exactly into groups of four, we have to insert zeroes to the left of the number:

$$\underset{}{111110} = \underset{3}{0011}\ \underset{E}{1110}$$

The required hexadecimal number is $3E_{16}$.

Table 1.2 shows the hexadecimal equivalent of decimal numbers 0 to 31.

Table 1.2: Hexadecimal Equivalent of Decimal Numbers

Decimal	Hexadecimal	Decimal	Hexadecimal
0	0	16	10
1	1	17	11
2	2	18	12
3	3	19	13
4	4	20	14
5	5	21	15
6	6	22	16
7	7	23	17
8	8	24	18
9	9	25	19
10	A	26	1A
11	B	27	1B
12	C	28	1C
13	D	29	1D
14	E	30	1E
15	F	31	1F

1.10 Converting Hexadecimal Numbers into Binary

To convert a hexadecimal number into binary, write the 4-bit binary equivalent of each hexadecimal digit.

Example 1.9

Convert hexadecimal number $A9_{16}$ into binary.

Solution

Writing the binary equivalent of each hexadecimal digit

$$A = 1010_2 \quad 9 = 1001_2$$

The required binary number is 10101001_2.

Example 1.10

Convert hexadecimal number $FE3C_{16}$ into binary.

Solution

Writing the binary equivalent of each hexadecimal digit

$$F = 1111_2 \quad E = 1110_2 \quad 3 = 0011_2 \quad C = 1100_2$$

The required binary number is 1111111000111100_2.

■

1.11 Converting Hexadecimal Numbers into Decimal

To convert a hexadecimal number into decimal, we have to calculate the sum of the powers of 16 of the number.

Example 1.11

Convert hexadecimal number $2AC_{16}$ into decimal.

Solution

Calculating the sum of the powers of 16 of the number:

$$\begin{aligned} 2AC_{16} &= 2 \times 16^2 + 10 \times 16^1 + 12 \times 16^0 \\ &= 512 + 160 + 12 \\ &= 684 \end{aligned}$$

The required decimal number is 684_{10}.

■

Example 1.12

Convert hexadecimal number EE_{16} into decimal.

Solution

Calculating the sum of the powers of 16 of the number

$$\begin{aligned} EE_{16} &= 14 \times 16^1 + 14 \times 16^0 \\ &= 224 + 14 \\ &= 238 \end{aligned}$$

The required decimal number is 238_{10}.

■

1.12 Converting Decimal Numbers into Hexadecimal

To convert a decimal number into hexadecimal, divide the number repeatedly by 16 and take the remainders. The first remainder is the LSD and the last remainder is the MSD.

Example 1.13

Convert decimal number 238_{10} into hexadecimal.

Solution

Dividing the number repeatedly by 16

238/16 → 14 Remainder 14 (E) (LSD)

14/16 → 0 Remainder 14 (E) (MSD)

The required hexadecimal number is EE_{16}.

■

Example 1.14

Convert decimal number 684_{10} into hexadecimal.

Solution

Dividing the number repeatedly by 16

684/16 → 42 Remainder 12 (C) (LSD)

42/16 → 2 Remainder 10 (A)

2/16 → 0 Remainder 2 (MSD)

The required hexadecimal number is $2AC_{16}$.

■

1.13 Converting Octal Numbers into Decimal

To convert an octal number into decimal, calculate the sum of the powers of 8 of the number.

Example 1.15

Convert octal number 15_8 into decimal.

Solution

Calculating the sum of the powers of 8 of the number

$$15_8 = 1 \times 8^1 + 5 \times 8^0$$
$$= 8 + 5$$
$$= 13$$

The required decimal number is 13_{10}. ■

Example 1.16

Convert octal number 237_8 into decimal.

Solution

Calculating the sum of the powers of 8 of the number

$$237_8 = 2 \times 8^2 + 3 \times 8^1 + 7 \times 8^0$$
$$= 128 + 24 + 7$$
$$= 159$$

The required decimal number is 159_{10}. ■

1.14 Converting Decimal Numbers into Octal

To convert a decimal number into octal, divide the number repeatedly by 8 and take the remainders. The first remainder is the LSD and the last remainder is the MSD.

Example 1.17

Convert decimal number 159_{10} into octal.

Solution

Dividing the number repeatedly by 8

$$159/8 \rightarrow 19 \quad \text{Remainder } 7 \quad \text{(LSD)}$$
$$19/8 \rightarrow 2 \quad \text{Remainder } 3$$
$$2/8 \rightarrow 0 \quad \text{Remainder } 2 \quad \text{(MSD)}$$

The required octal number is 237_8.

■

■ Example 1.18

Convert decimal number 460_{10} into octal.

Solution

Dividing the number repeatedly by 8

$$460/8 \rightarrow 57 \quad \text{Remainder } 4 \quad \text{(LSD)}$$
$$57/8 \rightarrow 7 \quad \text{Remainder } 1$$
$$7/8 \rightarrow 0 \quad \text{Remainder } 7 \quad \text{(MSD)}$$

The required octal number is 714_8.

Table 1.3 shows the octal equivalent of decimal numbers 0–31.

Table 1.3: Octal Equivalent of Decimal Numbers

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

■

1.15 Converting Octal Numbers into Binary

To convert an octal number into binary, write the 3-bit binary equivalent of each octal digit.

Example 1.19

Convert octal number 177_8 into binary.

Solution

Write the binary equivalent of each octal digit:

$$1 = 001_2 \quad 7 = 111_2 \quad 7 = 111_2$$

The required binary number is 001111111_2. ■

Example 1.20

Convert octal number 75_8 into binary.

Solution

Write the binary equivalent of each octal digit:

$$7 = 111_2 \quad 5 = 101_2$$

The required binary number is 111101_2. ■

1.16 Converting Binary Numbers into Octal

To convert a binary number into octal, arrange the number in groups of three and write the octal equivalent of each digit.

Example 1.21

Convert binary number 110111001_2 into octal.

Solution

Arranging in groups of three

$$110111001 = \underset{6}{110}\ \underset{7}{111}\ \underset{1}{001}$$

The required octal number is 671_8. ■

1.17 Negative Numbers

The most significant bit of a binary number is usually used as the sign bit. By convention, for positive numbers this bit is 0 and for negative numbers this bit is 1. Table 1.4 shows the 4-bit positive and negative numbers. The largest positive and negative numbers are +7 and −8, respectively.

To convert a positive number into negative, take the complement of the number and add 1. This process is also called the 2's complement of the number.

Example 1.22

Write decimal number −6 as a 4-bit number.

Solution

First, write the number as a positive number, then find the complement and add 1:

```
0110   +6
1001   complement
   1   add 1
------
1010   which is −6
```

■

Table 1.4: Four-Bit Positive and Negative Numbers

Binary Numbers	Decimal Equivalent
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Example 1.23

Write decimal number –25 as an 8-bit number.

Solution

First, write the number as a positive number, then find the complement and add 1:

```
00011001   +25
11100110   complement
        1  add 1
--------
11100111   which is –25
```

■

1.18 Adding Binary Numbers

The addition of binary numbers is similar to the addition of decimal numbers. Numbers in each column are added together with a possible carry from a previous column. The primitive addition operations are as follows:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$ generate a carry bit

$1 + 1 + 1 = 11$ generate a carry bit

Some examples are given below.

Example 1.24

Find the sum of binary numbers 011 and 110.

Solution

We can add these numbers as in the addition of decimal numbers:

```
  011   First column:    1 + 0 = 1
+ 110   Second column:   1 + 1 = 0  and a carry bit
------
 1001   Third column:    1 + 1 = 10
```

■

Example 1.25

Find the sum of binary numbers 01000011 and 00100010.

Solution

We can add these numbers as in the addition of decimal numbers:

```
  01000011     First column:     1 + 0 = 1
+ 00100010     Second column:    1 + 1 = 10
----------
  01100101     Third column:     0 + carry = 1
               Fourth column:    0 + 0 = 0
               Fifth column:     0 + 0 = 0
               Sixth column:     0 + 1 = 1
               Seventh column:   1 + 0 = 1
               Eighth column:    0 + 0 = 0
```

■

1.19 Subtracting Binary Numbers

To subtract two binary numbers, convert the number to be subtracted into negative and then add the two numbers.

Example 1.26

Subtract binary number 0010 from 0110.

Solution

First, let's convert the number to be subtracted into negative:

```
  0010   number to be subtracted
  1101   complement
     1   add 1
-------
  1110
```

Now, add the two numbers:

$$\begin{array}{r} 0110 \\ +\ 1110 \\ \hline 0100 \end{array}$$

Because we are using 4 bits only, we cannot show the carry bit.

■

1.20 Multiplication of Binary Numbers

Multiplication of two binary numbers is same as the multiplication of decimal numbers. The four possibilities are as follows:

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

Some examples are given below.

■ Example 1.27

Multiply the two binary numbers 0110 and 0010.

Solution

Multiplying the numbers

$$\begin{array}{r} 0110 \\ 0010 \\ \hline 0000 \\ 0110 \\ 0000 \\ 0000 \\ \hline 001100 \text{ or } 1100 \end{array}$$

In this example, 4 bits are needed to show the final result.

■

Example 1.28

Multiply binary numbers 1001 and 1010.

Solution

Multiplying the numbers

$$\begin{array}{r} 1001 \\ 1010 \\ \hline 0000 \\ 1001 \\ 0000 \\ 1001 \\ \hline 1011010 \end{array}$$

In this example, 7 bits are required to show the final result. ■

1.21 Division of Binary Numbers

The division of binary numbers is similar to the division of decimal numbers. An example is given below.

Example 1.29

Divide binary number 1110 by binary number 10.

Solution

Dividing the numbers

$$\begin{array}{r} 111 \\ 10\overline{)1110} \\ 10 \\ \hline 11 \\ 10 \\ \hline 10 \\ 10 \\ \hline 00 \end{array}$$

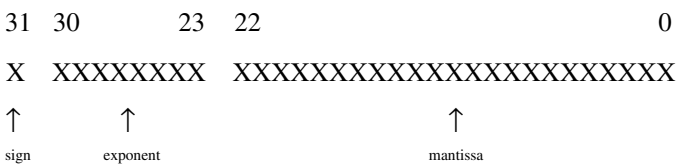
gives the result 111_2. ■

1.22 Floating Point Numbers

Floating point numbers are used to represent noninteger fractional numbers and are used in most engineering and technical calculations, for example, 3.256, 2.1, and 0.0036. The most commonly used floating point standard is the IEEE standard. According to this standard, floating point numbers are represented with 32 bits (single precision) or 64 bits (double precision).

In this section, we will look at the format of 32-bit floating point numbers only and see how mathematical operations can be performed with such numbers.

According to the IEEE standard, 32-bit floating point numbers are represented as follows:



The most significant bit indicates sign of the number, where 0 indicates positive and 1 indicates negative.

The 8-bit exponent shows the power of the number. To make the calculations easy, the sign of the exponent is not shown, but instead excess 128 numbering system is used. Thus, to find the real exponent, we have to subtract 127 from the given exponent. For example, if the mantissa is "10000000," the real value of the mantissa is 128 − 127 = 1.

The mantissa is 23 bits wide and represents the increasing negative powers of 2. For example, if we assume that the mantissa is "11100000000000000000," the value of this mantissa is calculated as follows: $2^{-1} + 2^{-2} + 2^{-3} = 7/8$.

The decimal equivalent of a floating point number can be calculated using the following formula:

$$\text{Number} = (-1)^s \, 2^{e-127} \, 1 \cdot f,$$

where $s = 0$ for positive numbers, 1 for negative numbers,
e = exponent (between 0 and 255), and
f = mantissa.

As shown in the above formula, there is a hidden "1" before the mantissa; i.e., mantissa is shown as "$1 \cdot f$."

The largest and the smallest numbers in 32-bit floating point format are as follows:

The largest number

0 11111110 11111111111111111111111

This number is $(2 - 2^{-23})\ 2^{127}$ or decimal 3.403×10^{38}. The numbers keep their precision up to six digits after the decimal point.

The smallest number

0 00000001 00000000000000000000000

This number is 2^{-126} or decimal 1.175×10^{-38}.

1.23 Converting a Floating Point Number into Decimal

To convert a given floating point number into decimal, we have to find the mantissa and the exponent of the number and then convert into decimal as shown above.

Some examples are given here.

Example 1.30

Find the decimal equivalent of the floating point number given below:

0 10000001 10000000000000000000000

Solution

Here,

sign = positive

exponent = 129 − 127 = 2

mantissa = $2^{-1} = 0.5$

The decimal equivalent of this number is $+1.5 \times 2^2 = +6.0$.

■

Example 1.31

Find the decimal equivalent of the floating point number given below:

0 10000010 11000000000000000000

Solution

In this example,

$$\text{sign} = \text{positive}$$

$$\text{exponent} = 130 - 127 = 3$$

$$\text{mantissa} = 2^{-1} + 2^{-2} = 0.75$$

The decimal equivalent of the number is $+1.75 \times 2^3 = 14.0$. ■

1.23.1 Normalizing the Floating Point Numbers

Floating point numbers are usually shown in normalized form. A normalized number has only one digit before the decimal point (a hidden number 1 is assumed before the decimal point).

To normalize a given floating point number, we have to move the decimal point repetitively one digit to the left and then increase the exponent after each move.

Some examples are given below.

Example 1.32

Normalize the floating point number 123.56.

Solution

If we write the number with a single digit before the decimal point, we get

$$1.2356 \times 10^2$$ ■

Example 1.33

Normalize the binary number 1011.1_2.

Solution

If we write the number with a single digit before the decimal point, we get

$$1.0111 \times 2^3$$ ■

1.23.2 Converting a Decimal Number into Floating Point

To convert a given decimal number into floating point, we have to carry out the following steps:

- Write the number in binary
- Normalize the number
- Find the mantissa and the exponent
- Write the number as a floating point number

Some examples are given below.

Example 1.34

Convert decimal number 2.25_{10} into floating point.

Solution

Writing the number in binary

$$2.25_{10} = 10.01_2$$

Normalizing the number,

$$10.01_2 = 1.001_2 \times 2^1$$

Here, $s = 0$, $e - 127 = 1$ or $e = 128$, and $f = 00100000000000000000000$.

(Remember that a number 1 is assumed on the left-hand side, even though it is not shown in the calculation.) We can now write the required floating point number as follows:

s	e	f
0	10000000	(1)001 0000 0000 0000 0000 0000

or the required 32-bit floating point number is

$$01000000000100000000000000000000$$

■

Example 1.35

Convert the decimal number 134.0625_{10} into floating point.

Solution

Writing the number in binary

$$134.0625_{10} = 10000110.0001$$

Normalizing the number

$$10000110.0001 = 1.00001100001 \times 2^7$$

Here, $s = 0$, $e - 127 = 7$ or $e = 134$, and $f = 00001100001000000000000$

We can now write the required floating point number as follows:

s	e	f
0	10000110	(1)00001100001000000000000

or the required 32-bit floating point number is

$$01000011000001100001000000000000$$

■

1.23.3 Multiplication and Division of Floating Point Numbers

The multiplication and division of floating point numbers is rather easy and the steps are given below:

- Add (or subtract) the exponents of the numbers
- Multiply (or divide) the mantissa of the numbers
- Correct the exponent
- Normalize the number

The sign of the result is the EXOR of the signs of the two numbers.

Because the exponent is processed twice in the calculations, we have to subtract 127 from the exponent.

An example is given below to show the multiplication of two floating point numbers.

■ Example 1.36

Show the decimal numbers 0.5_{10} and 0.75_{10} in floating point and then calculate the multiplication of these numbers.

Solution

We can convert the numbers into floating point as follows:

$$0.5_{10} = 1.0000 \times 2^{-1}$$

Here, $s = 0$, $e - 127 = -1$ or $e = 126$ and $f = 0000$

or

$$0.5_{10} = 0 \;\; 01110110 \;\; (1)000 \;\; 0000 \;\; 0000 \;\; 0000 \;\; 0000 \;\; 0000$$

Similarly,

$$0.75_{10} = 1.1000 \times 2^{-1}$$

Here, $s = 0$, e = 126, and $f = 1000$

or

$$0.75_{10} = 0\ \ 01110110\ \ (1)100\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000$$

Multiplying the mantissas, we get "(1)100 0000 0000 0000 0000 0000." The sum of the exponents is 126 + 126 = 252. Subtracting 127 from the mantissa, we obtain 252 − 127 = 125. The EXOR of the signs of the numbers is 0. Thus, the result can be shown in floating point as follows:

$$0\quad 01111101\qquad (1)100\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000$$

The above number is equivalent to decimal 0.375 ($0.5 \times 0.75 = 0.375$), which is the correct result.

■

1.23.4 Addition and Subtraction of Floating Point Numbers

The exponents of floating point numbers must be the same before they can be added or subtracted. The steps to add or subtract floating point numbers is as follows:

- Shift the smaller number to the right until the exponents of both numbers are the same. Increment the exponent of the smaller number after each shift.
- Add (or subtract) the mantissa of each number as an integer calculation, without considering the decimal points.
- Normalize the obtained result.

An example is given below.

■ Example 1.37

Show decimal numbers 0.5_{10} and 0.75_{10} in floating point and then calculate the sum of these numbers.

Solution

As shown in Example 1.36, we can convert the numbers into floating point as follows:

$$0.5_{10} = 0\ \ 01110110\quad (1)000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000$$

Similarly,

$$0.75_{10} = 0\ \ 01110110\ \ (1)100\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000$$

Because the exponents of both numbers are the same, there is no need to shift the smaller number. If we add the mantissa of the numbers without considering the decimal points, we get

$$\begin{array}{l} (1)000\ 0000\ 0000\ 0000\ 0000\ 0000 \\ (1)100\ 0000\ 0000\ 0000\ 0000\ 0000 \\ \hline (10)100\ 0000\ 0000\ 0000\ 0000\ 0000 \end{array} +$$

To normalize the number, we can shift it right by one digit and then increment its exponent. The resulting number is

$$0\ 01111111\ (1)010\ 0000\ 0000\ 0000\ 0000\ 0000$$

The above floating point number is equal to decimal number 1.25, which is the sum of decimal numbers 0.5 and 0.75.

To convert floating point numbers into decimal and decimal numbers into floating point, the freely available program given in the following Web site can be used:

http://www.babbage.cs.qc.edu/IEEE-754/Decimal.html

■

1.24 Binary-Coded Decimal Numbers

Binary-coded decimal (BCD) numbers are usually used in display systems like LCDs and seven-segment displays to show numeric values. BCD data is stored in either packed or unpacked forms. Packed BCD data is stored as two digits per byte and unpacked BCD data is stored as one digit per byte. Unpacked BCD data is usually returned from a keypad or a keyboard. The packed BCD is more frequently used, and this is the format described in the remainder of this section.

In packed BCD, each digit is a 4-bit number from 0 to 9. As an example, Table 1.5 shows the packed BCD numbers between 0 and 20.

Table 1.5: Packed BCD Numbers Between 0 and 20

Decimal	BCD	Binary
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101

Table 1.5: Packed BCD Numbers Between 0 and 20 —cont'd

Decimal	BCD	Binary
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	0001 0000	1010
11	0001 0001	1011
12	0001 0010	100
13	0001 0011	1101
14	0001 0100	1110
15	0001 0101	1111
16	0001 0110	1 0000
17	0001 0111	1 0001
18	0001 1000	1 0010
19	0001 1001	1 0011
20	0010 0000	1 0100

Example 1.38

Write the decimal number 295 as a packed BCD number.

Solution

Writing the 4-bit binary equivalent of each digit

$$2 = 0010_2 \qquad 9 = 1001_2 \qquad 5 = 0101_2$$

The required packed BCD number is $0010\ 1001\ 0101_2$. ■

Example 1.39

Write the decimal equivalent of packed BCD number $1001\ 1001\ 0110\ 0001_2$.

Solution

Writing the decimal equivalent of each 4-bit group, we get the decimal number 9961. ■

1.25 Summary

This chapter has given an introduction to the microprocessor and microcontroller systems. The basic building blocks of microcontrollers have been described briefly. The chapter has also provided an introduction to various number systems, and has described how to convert a given number from one base into another base. The important topics of floating point numbers and floating point arithmetic have also been described with examples.

1.26 Exercises

1. What is a microcontroller? What is a microprocessor? Explain the main differences between a microprocessor and a microcontroller.
2. Give some example applications of microcontrollers around you.
3. Where would you use an EPROM?
4. Where would you use a RAM?
5. Explain what types of memory are usually used in microcontrollers.
6. What is an I/O port?
7. What is an A/D converter? Give an example of use of this converter.
8. Explain why a watchdog timer could be useful in a real-time system.
9. What is serial I/O? Where would you use serial communication?
10. Why is the current sinking/sourcing important in the specification of an output port pin?
11. What is an interrupt? Explain what happens when an interrupt is recognized by a microcontroller.
12. Why is brown-out detection important in real-time systems?
13. Explain the differences between a RISC-based microcontroller and a CISC-based microcontroller. What type of microcontroller is PIC?
14. Convert the following decimal numbers into binary:
    a) 23 b) 128 c) 255 d) 1023
    e) 120 f) 32000 g) 160 h) 250
15. Convert the following binary numbers into decimal:
    a) 1111 b) 0110 c) 11110000
    d) 00001111 e) 10101010 f) 10000000

16. Convert the following octal numbers into decimal:
    a) 177 b) 762 c) 777 d) 123
    e) 1777 f) 655 g) 177777 h) 207

17. Convert the following decimal numbers into octal:
    a) 255 b) 1024 c) 129 d) 2450
    e) 4096 f) 256 g) 180 h) 4096

18. Convert the following hexadecimal numbers into decimal:
    a) AA b) EF c) 1FF d) FFFF
    e) 1AA f) FEF g) F0 h) CC

19. Convert the following binary numbers into hexadecimal:
    a) 0101 b) 11111111 c) 1111 d) 1010
    e) 1110 f) 10011111 g) 1001 h) 1100

20. Convert the following binary numbers into octal:
    a) 111000 b) 000111 c) 1111111 d) 010111
    e) 110001 f) 11111111 g) 1000001 h) 110000

21. Convert the following octal numbers into binary:
    a) 177 b) 7777 c) 555 d) 111
    e) 1777777 f) 55571 g) 171 h) 1777

22. Convert the following hexadecimal numbers into octal:
    a) AA b) FF c) FFFF d) 1AC
    e) CC f) EE g) EEFF h) AB

23. Convert the following octal numbers into hexadecimal:
    a) 177 b) 777 c) 123 d) 23
    e) 1111 f) 17777777 g) 349 h) 17

24. Convert the following decimal numbers into floating point:
    a) 23.45 b) 1.25 c) 45.86 d) 0.56

25. Convert the following decimal numbers into floating point and then calculate their sum:
    0.255 and 1.75

26. Convert the following decimal numbers into floating point and then calculate their product:
    2.125 and 3.75

27. Convert the following decimal numbers into packed BCD:
    a) 128 b) 970 c) 900 d) 125

28. Convert the following decimal numbers into unpacked BCD:
    a) 128 b) 970 c) 900 d) 125

29. Convert the following packed BCD numbers into decimal:
    a) 0110 0011 b) 0111 0100 c) 0001 0111