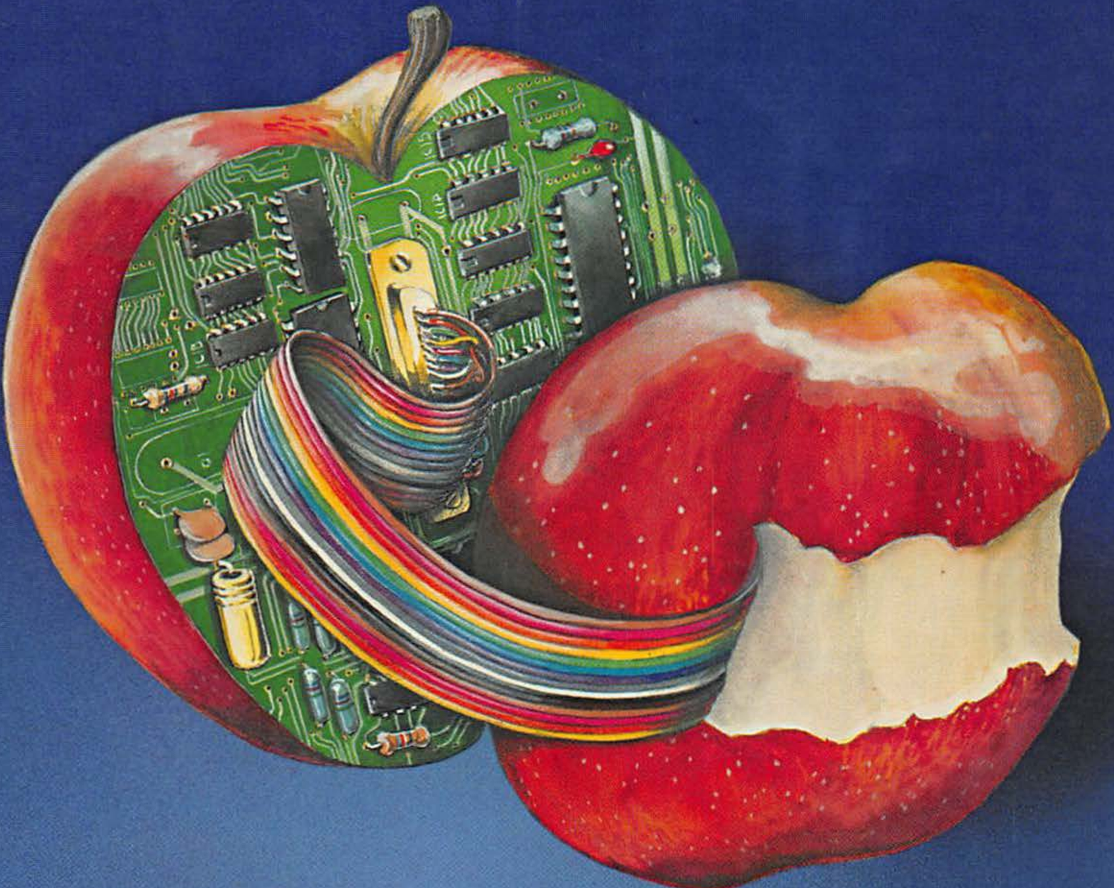
$
12.95

The Apple® Connection



James W. Coffron

SYBEX

John
Dana
949-3651

The Apple® Connection

An Introduction to the Techniques and Principles of Apple Computer Interfacing

The Apple® Connection

An Introduction to the Techniques and Principles of Apple Computer Interfacing

James W. Coffron

SYBEX

Berkeley • Paris • Düsseldorf

Cover Design by Daniel Le Noury
Book layout and design by Marlyn Amann

Library of Congress Card Number: 82-50620
ISBN 0-89588-085-7
First Edition 1982
Printed in the United States of America
10 9 8 7 6 5 4 3 2

This book is for my family—
Carol, Jeffrey, Kelly and Rocky

Contents

Acknowledgements:

I would like to thank the staff at Sybex for their help in getting this book produced. Special thanks to Jim Compton for his assistance with the numerous corrections and revisions to the original text, and to Jim Horn for his careful and thorough technical review of the manuscript.

Preface

If you have recently purchased an Apple® computer, or are thinking of purchasing one, many questions about the system have probably arisen in your mind. Many new computer owners are curious about the potential extent of their computer's overall usefulness. While it is true that a specific use for the system is probably the reason you bought it in the first place, you may wonder what else it can do.

It is safe to say that there are more potential future uses of the system than any purchaser can dream of at present. As you sit in front of the computer, you can look forward to many hours of enjoyment using the numerous available application programs, and the various "off-the-shelf" games that can be played on the system. If you are a beginner in home computers, or computer languages, these games and programs may seem difficult to master at first, but this difficulty will soon pass, so do not fear.

At first you may be hesitant to use the system. A "fear of the unknown" surfaces as you test the machine's reactions to your nimble (or not-so-nimble) touches on the keys. Then, your boldness and confidence improves as you discover it is OK to make mistakes. Nothing drastic happens when you press the wrong key. The Apple computer is a very forgiving instrument.

Before long, you are deftly running, modifying, and writing programs. The once-formidable task of using a home computer has diminished. Soon you find yourself looking for new challenges and new applications for the computer. At this point you may start to wonder, "Can I use the computer around the home? Is it possible to control my appliances, heating, or security systems with my computer?"

You know these things are possibilities, because you have read about them. However, you may feel it is far beyond your ability to accomplish them. If you think that, you will soon see that you are wrong. The realization of these controls with your Apple computer is not beyond your capabilities. The information required may be different from that which you use every day, but making *the Apple Connection* is a straightforward process, and not very complex.

The designers of the Apple computer have used valuable foresight in anticipating that system users who do not know or care much about hardware may wish to create new interfaces between their system and the outside world. With that in mind, the Apple computer architecture was designed to make the interfacing job an easy one. You do not have to be a computer expert to construct the hardware for controlling external devices or to write the software for control. This claim will be borne out as you progress through the examples outlined in the text.

So if you are ready to come into the world of computer control, this book is the first step. It will open the door and provide you with the essential information needed to connect your computer to a variety of peripheral devices.

Without any further hesitation, turn the pages and learn to make *the Apple Connection.*

Introduction

This book is written for everyone who wants to understand how the Apple, as well as other home computers, can be interfaced to the outside world. Specific examples are shown, using the Apple computer to illustrate the essential concepts of computer control and interfacing. However, the information and ideas presented here can be readily adapted to most home computer systems.

Interfacing and controlling external devices with an Apple computer will involve the use of both software and hardware. To that end, this text assumes the reader can write simple programs in BASIC. An extensive knowledge of BASIC is not required to get the maximum value from this text. The hardware concepts are presented with the understanding that many readers may not be familiar with digital electronics. You do not have to be a software or hardware expert to make good use of the information given in this text.

This book is organized so as to enable the reader to understand how all the pieces of the interfacing and control puzzle fit together. The path is straightforward and not complicated, but you will have to learn some new information and concepts. All the essential information for interfacing and controlling external devices with the Apple and other home computers is given in the pages that follow.

Chapter 1 starts off with an introduction to, and a definition of, the concept of computer control and presents some new vocabulary.

Chapter 2 discusses the software required to output information to an external device with the Apple computer. The programming language is BASIC.

Chapter 3 discusses the software required to input information into the Apple computer from an external device.

Chapter 4 introduces the basic hardware concepts necessary to input and output information to and from the Apple computer. This chapter is designed for readers who are unfamiliar with digital electronics, and want to learn only as much about it as they need for practical purposes.

Chapter 5 presents an application of computer control in the design of a home security system. It starts with a definition of the problem and works through the software and hardware concepts necessary to have a working home security system. After this chapter you will have a good general idea of how to use the computer in a home security application.

Chapter 6 shows how to interface the Apple computer to allow automatic control of home appliances that run from 120- or 220-volt AC sources, such as toasters, lamps, and coffee pots.

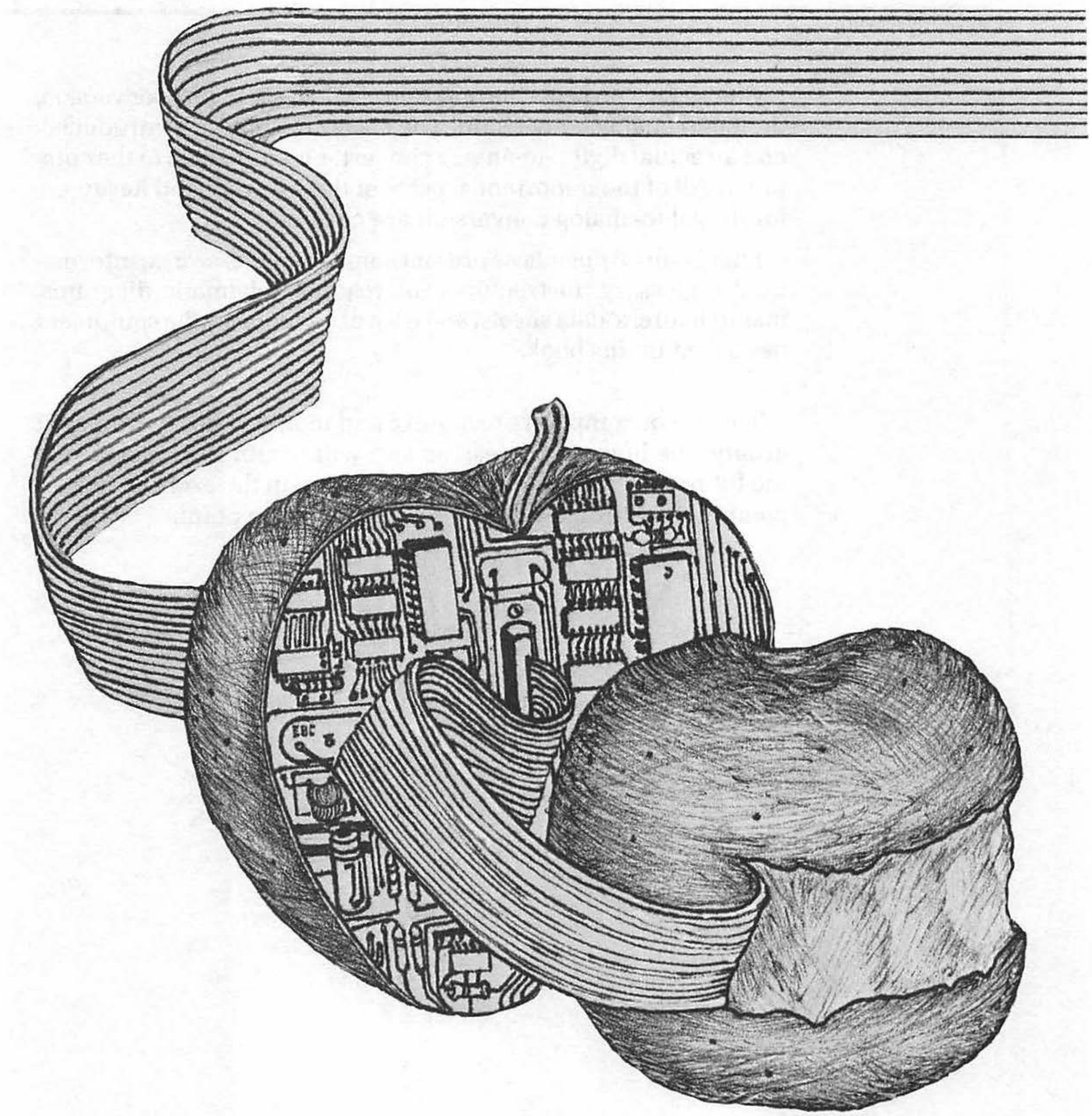
Chapter 7 discusses the difference between the terms *analog* and *digital*, using examples comparing how each would appear in an analog or digital environment. This chapter concludes with an explanation of the term *transducer*.

Chapter 8 shows how to perform analog-to-digital conversion with the computer. General concepts applicable to most home computers are discussed. An actual analog-to-digital converter is connected to the Apple computer, with all important software and hardware details shown. The chapter concludes with a complete hardware and software system for measuring temperature with the Apple computer.

Chapter 9 presents the opposite of analog-to-digital conversion, digital-to-analog conversion. The basic concepts are introduced, and an actual digital-to-analog converter is connected to the computer. All of the important aspects of the software and hardware for digital-to-analog conversion are covered.

Finally, the *Appendices* present some useful reference information: a glossary, instructions for reading schematic diagrams, manufacturers' data sheets, and a list of vendors for the equipment described in this book.

The use of computers to control and monitor the environment around the home is increasing and will continue to increase in the future. If you want to become involved in the exciting field of computer control, this book is a good starting point.



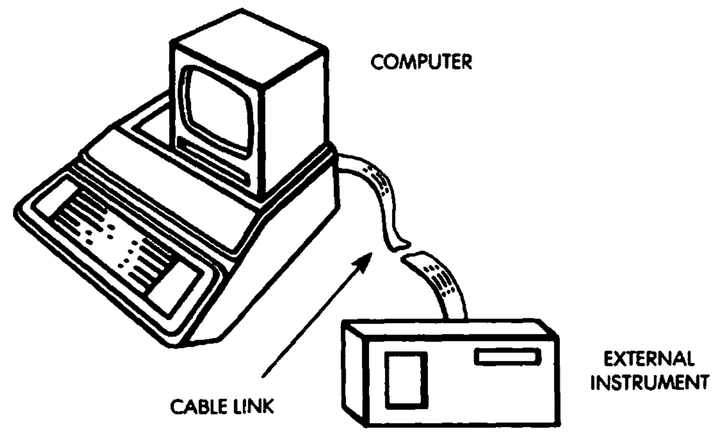
Chapter 1

Introduction to Computer Control

BEFORE WE LAUNCH into the discussions of actual computer control that start in Chapter 2, let us take some time to explore the meaning of the term "computer control." To some, this term may call up images of futuristic robots, huge, automated factories and complex spacecraft. To others, computer control may seem like something which only scientists use—inevitable, but too complicated for them to understand. In fact, scientists and industrial designers *do* use computer control in spacecraft and automated factories, and these applications are quite complex, but the term can also be applied to much simpler home applications, such as those described in this book.

1.1: WHAT IS COMPUTER CONTROL?

The overall objective of this book is to enable you to understand computer control. With this understanding will come new insight, allowing persons not directly involved with scientific applications of the computer to see many ways the computer can be applied in the home. As these home applications of the computer are developed, you will lose whatever fear you may have of automation and gain respect for what computer control can do and how



The main concept of computer control is that the computer will direct the action of an external piece of hardware. The link shown here is made by a cable. However, the link may also be made by transmission at radio frequencies, without a physical connection.

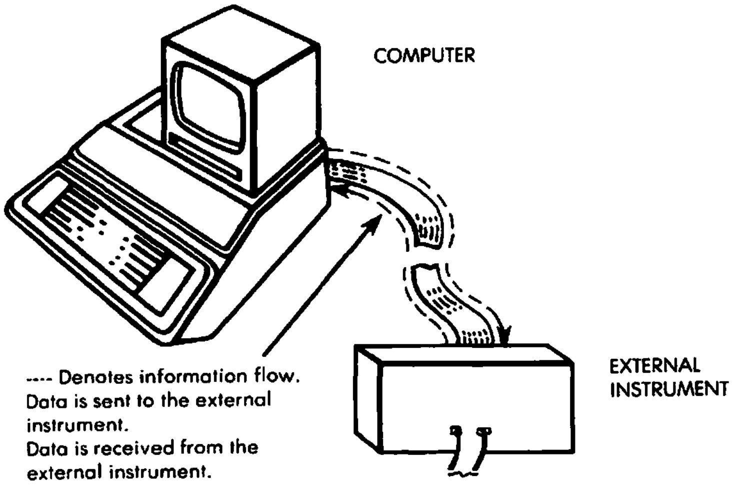
Figure 1.1

it can help you. Another major objective of this book is to show that computer control does not have to be complicated.

We use the Apple computer in this text as the means of control. However, the concept of computer control is applicable to almost any home computer. Further, if you have an Apple computer at home you have already used computer control and may not have been aware of it.

To answer the question of what is meant by computer control we will show several examples of how a home computer is used. The concept of computer control is quite simple, and is graphically illustrated in Figure 1.1. In this diagram the computer is connected in some way to direct the action of another piece of hardware. This, in essence, is what computer control is all about. The computer is directing the physical or electrical action of an external hardware device.

In almost any computer control application, the computer must have a way of understanding how the external hardware is


COMPUTER
EXTERNAL INSTRUMENT
---- Denotes information flow.
Data is sent to the external instrument.
Data is received from the external instrument.

The computer will send information to the external device to control the device's actions. Information can also be sent from the external device to be read by the computer.

Figure 1.2

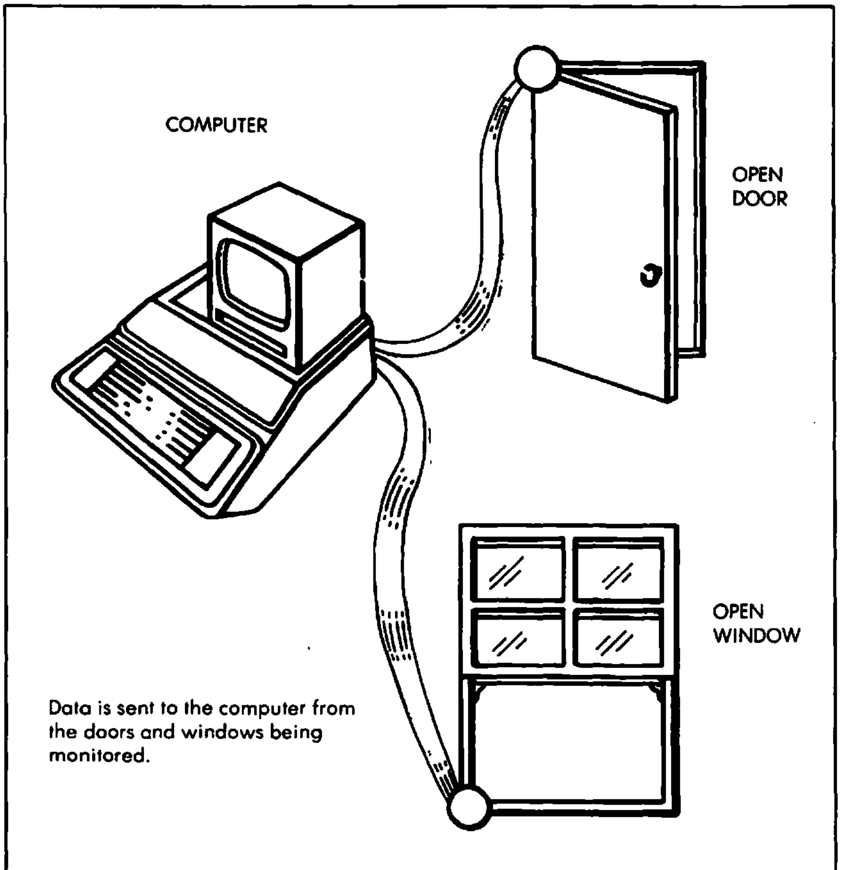
responding to its control. Therefore, the computer must not only direct the action of the external hardware, but monitor it also. In Figure 1.2 we see that the computer will receive information from the external hardware. Based on that information, the computer can modify the directions it gives to the external device.

This simple example illustrates the basic elements of computer control. The two processes—sending directions to the external hardware from the computer, and receiving information from the external device—are the essential concepts of computer control. At this point in our discussion, it may be valuable to list these two important concepts:

1. The computer sends directions to the external hardware.

2. The computer receives information from the external hardware.

These two ideas are the basis for computer control. The purpose of this book is to explain *how* these two tasks are performed.


COMPUTER
OPEN DOOR
OPEN WINDOW
Data is sent to the computer from the doors and windows being monitored.

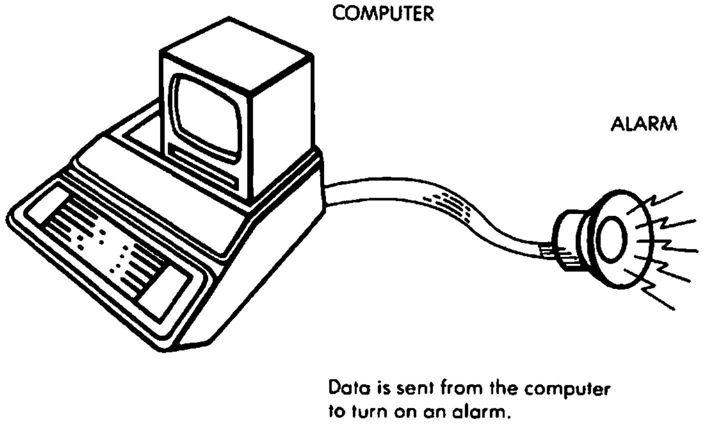
Doors and windows send information to the computer to report their status, either "open" or "closed."

Figure 1.3

1.2: A PRACTICAL EXAMPLE OF THE TWO BASIC CONCEPTS

An example of computer control that most of us can imagine ourselves using is a home security system. With the help of diagrams, we will see that this goal can be achieved using only the two concepts we have outlined. We will return to this example in Chapter 5 and develop such a system in detail.

Let us start by defining what it is we want the security system to do. In short, we want our home safe from intruders. Unfortunately,



The computer will send information to the alarm to inform the user that a window or door has been opened.

Figure 1.4

this type of statement is useless for our purpose, because the word "safe" conjures up entirely different meanings to different people. For this example, let us define the function of our security system a little more precisely: the system will detect any window or door being opened. When this occurs, the system will indicate which door or window it is. Finally, the system will sound an alarm if any door or window is open. This is the definition of exactly what we wish the system to do. In a later chapter of this book, we will expand on this system to show many new ideas for a home security system that is computer controlled.

Using this definition will require that the doors and windows be capable of sending information to the computer. This is shown in Figure 1.3. This will involve the second concept that was given in Section 1.1. That is, the computer receives information from an external device.

If the computer detects that one of the windows or doors has been opened, an alarm must be set off. This is shown in Figure 1.4. In order for the computer to set off the alarm, information from the computer must be sent to the alarm to direct the external hardware to emit the noise. This type of computer action was shown as concept number 1 in Section 1.1, where the computer will send information to an external device.

Looking at Figures 1.3 and 1.4, we can see that all of the functions we require our system to perform can be done by the computer using only the two concepts outlined. It is true that we have, so to speak, "waved our hands" over some important points. For example, *how* does a door or window send electrical information to a computer? *How* does a computer sound an alarm? These points were deliberately ignored. It is possible for a door or window to be made to send electrical information to the computer and for the computer to sound an alarm. How that is done is what this book is all about. We will show exactly how to do these things in later chapters. It is too early in the discussion to approach these details yet. We must first understand where we are heading. From our first example, we can see that the entire subject of computer control can be reduced to the repeated application of the two basic concepts. These two concepts are what should be understood at this time. You may be assured that this text will cover in *detail* how to achieve the points glossed over in this first example.

1.3: SOME NEW VOCABULARY

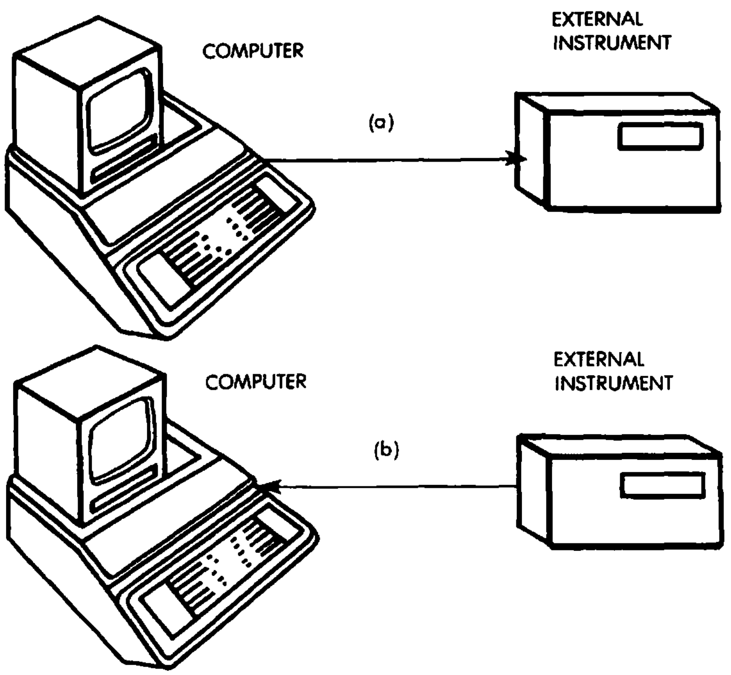
When we enter a new area of study, a major obstacle we face is learning the vocabulary essential to the area. Interfacing a computer to control an external device is no exception. This section will discuss some of the new words that you are likely to encounter when reading or talking about the topic. If you are reading this book, and own or have access to an Apple, you are probably also beginning to read some of the magazines and books now available on the subject. You may also have friends or associates who talk "computerese." All the new terms you may encounter cannot be defined here, but as the topics in the text warrant it, new vocabulary will be introduced. The new words given in this section are meant to help the beginner to understand the literature (including the Apple documentation) as quickly as possible.

The two major concepts we have introduced, sending information from a computer to an external device, and receiving information from the external device to be used by the computer, are referred to as *output* and *input*, respectively. These terms are applied both to the act or event of communicating information between a computer and an external device, and the information itself. Thus, output is both the transfer of information from the computer to the external device and the information sent, and input is the transfer of information from an external source into the computer, and the information entered.

If you have ever used an Apple computer, you have made use of input and output. When you press a key on the keyboard, the computer is inputting the information you pressed. When the letter you pressed on the keyboard appears on the screen of your video monitor or TV, the computer is outputting information. We do not usually think of the keyboard or the video monitor in the Apple computer as devices for input and output, because they are integral parts of the computer system. In fact, although these components are part of the system, they are external to the computer's *central processing unit*, or *CPU*. Ordinarily, we think of input and output as being performed through external devices connected to the computer by, say, a cable. These devices might include the special applications we will be developing in this book, or commercially available *peripherals*.

Consider the case of a printer attached to a system. When the Apple computer gives a printout, the printer is operating under the control of the computer. During the time the printer is printing and causing the paper to scroll, the computer is outputting information to the printer. If you have a floppy disk drive or tape cassette attached to your Apple system, the computer is controlling these devices. When the computer writes your program onto the disk or the tape cassette, it is outputting information. When the computer reads a program from the floppy disk or the tape cassette, an input event is occurring.

Figure 1.5 illustrates the general concept of input and output. These two terms are sometimes combined and abbreviated to the single term *I/O*. Whenever the computer is controlling an external device or performing input and output events, the computer is said to be "performing I/O."



a) This is an example of output. *The computer is sending information to the external device.*

b) This is an example of input. *The computer is receiving information from the external device.*

Figure 1.5

The next term we will discuss is *transducer*. This word is used quite a lot in discussing I/O with a computer. To illustrate what a transducer is, let us return to our example of a home security system. In this case we constructed a means by which the computer would receive input information concerning the physical position of a door or a window. The information that would be input to the computer is electrical. The door or window gives out only physical movement information. That is, all the door or window can do is move. Therefore, we need some type of device that will change the physical movement of the door or window into electrical information that can be input to the computer. The device that allows this to occur is called a *transducer.*

This example is one type of transducer. As we will see in Chapter 5, the device used is a simple switch, open when the window is open and closed when it is closed. In general, a transducer is a device or piece of hardware which produces an electrical output when given a physical input, which translates one form of energy or information into another. In the example just given the transducer must be capable of converting movement into electrical information. Other examples of transducers are:

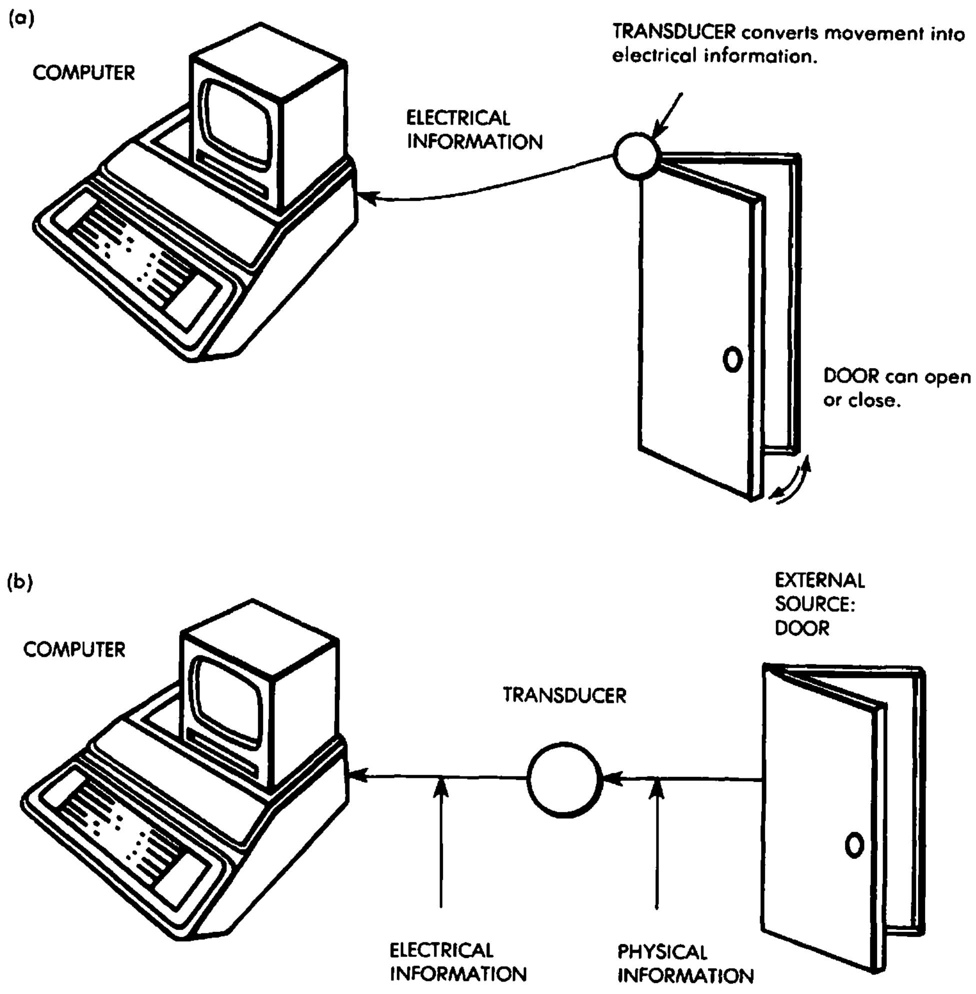
1. Temperature transducer
2. Pressure transducer
3. Revolutions per minute transducer

As you can see, transducers are classified by the type of information they translate into electricity. Figure 1.6 shows how a transducer would fit into the computer input path from an external source.

An example of a transducer that you have probably used is the keyboard on the Apple computer. The keyboard must convert the keypush into electrical information that the computer can input.

At this time you may wonder if a transducer is used when the computer is outputting information. In the example of the home security system, the computer would sound an alarm if an open window or door were detected. The alarm would be a loud bell or siren. This type of device receives electrical signals from the computer and produces a physical effect, the motion of air, or sound. Broadly, any device that translates one form of energy into another (including electricity into sound) can be called a transducer. In the context of computer control, however, these types of devices are not generally put into the class of transducers. Instead, they are grouped together generally as *output devices,* or simply called by their specific names: bell, motor, relay, fan, heater, and lights. Figure 1.7 shows how a relay is used to activate these devices to output information from the computer.

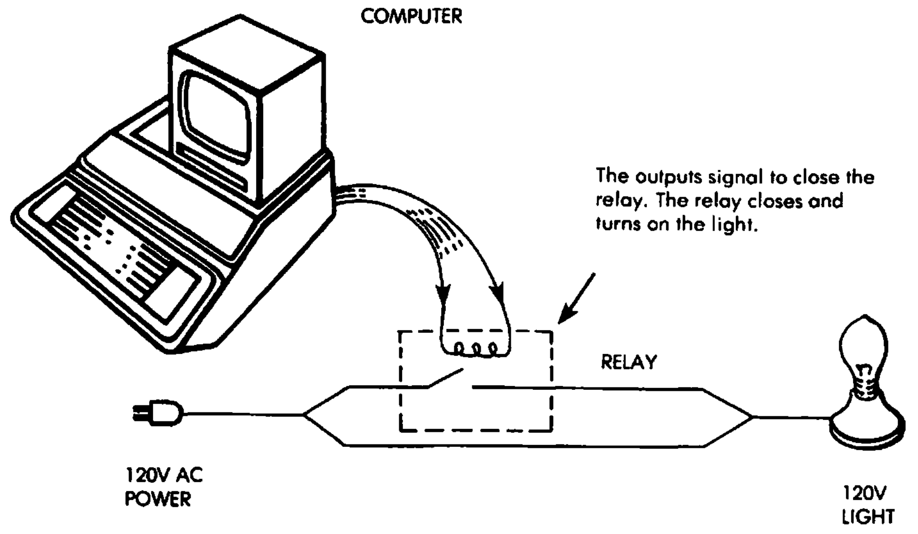
The next term to discuss is *digital.* Many people are already familiar with this particular word. In a general sense the term means that there are discrete values that a particular event can



a) A transducer will transform the physical movement of the door into electrical information which can then be input to the computer.

b) This block diagram shows where a transducer fits in the computer input path. The transducer will have a physical event input to it; that is, it will record a physical event. The output of the transducer will be an electrical signal related to the physical event being input to it.

Figure 1.6


COMPUTER
The outputs signal to close the relay. The relay closes and turns on the light.
RELAY
120V AC POWER
120V LIGHT

A physical device that will transform the electrical output of the computer into a physical event. This device, a relay, will transform the electrical output of the computer into movement of the relay contacts.

Figure 1.7

equal. For example, the television channels are assigned certain frequencies, out of an infinite number of possible frequencies. We can say that selecting a TV station is a digital process, because the channel values can only be those specified and nothing in between. We do not have channel 2.5, for example, we have channel 2 or 3.

The term *digital* is usually contrasted with the term *analog.* Digital systems *count* discrete units; analog systems *measure* over a continuous range. This topic is discussed at greater length in Chapter 7.

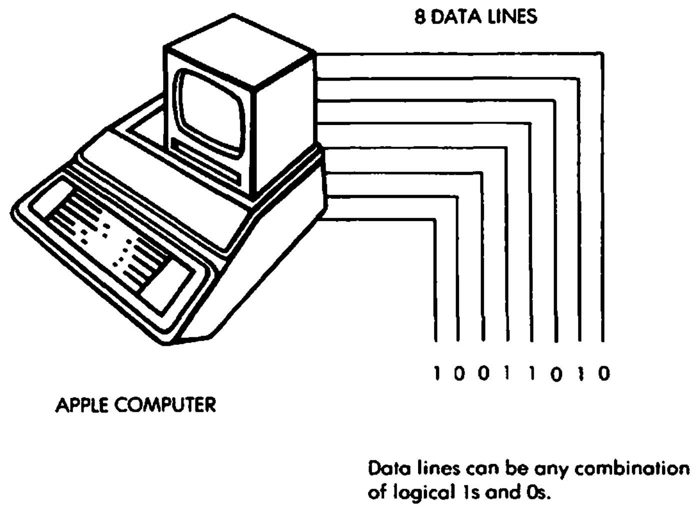
When applied to a computer, the term *digital* means there are discrete voltage levels present in the information. All information sent out by the computer (output) or received by the computer (input) must be made up of two voltage levels. All information that is used by the Apple computer consists of these two voltages. This is the reason the Apple computer is called a *digital* computer. There are discrete values for the information; further, there are only two discrete values.

The two voltage levels for the information used in the Apple computer are given the labels 0 and 1. These are the only choices each digit may take in base 2, the binary notation used in almost all computers. All information processed by a computer is expressed in combinations of these two digits. Also, a logic in which every statement is either "true" or "false" is a *binary* logic. In computer logic, the digits 0 and 1 are assigned to these two values, and are called "logical 0" and "logical 1." For these reasons, the two terms "binary" and "digital" are often used almost interchangeably in the computer field. In digital electronics the actual voltage value of a 0 or a 1 may vary among different types of digital equipment. In some types of digital systems, a 0 may be equal to −1.9 volts and a 1 may be equal to −1.5 volts. In other digital systems a 0 may be equal to 0.0 volts and a 1 equal to 9.0 volts. The voltage values used in the Apple computer are called TTL (Transistor-Transistor Logic) voltage levels. These voltage levels are approximately 0.0 to 0.8 volts for a logical 0, and 2.4 to 5.0 volts for a logical 1. We will discuss exactly what is meant by these voltage levels in Chapter 6. For now it is important to understand that there are only *two* distinct voltage levels present in the Apple computer when information is output or input.

The next term to discuss is *data*. This term is used in many different ways when discussing digital computers. For our purposes the term *data* will refer to the digital information that will be input to, or output from, the Apple computer. Physically, data in a computer can be thought of as a series of electrical pulses occurring at specified times.

Since the information processed by the Apple computer is digital, it can be referred to as digital data. That is, the Apple computer will output digital data, and input digital data only. No other type of data is acceptable. If you are a beginner in computers, other types of data may not occur to you at this point. However, as we proceed in this text we will see other meanings for the term *data*.

Let us now discuss the term *bit*. To define this term, we must first look at how digital data will appear in the Apple computer. Figure 1.8 shows what one typical piece of data would look like to the Apple computer. We see in Figure 1.8 that the data is composed



Data output lines from the computer can be set to any combination of logical 1s or 0s.

Figure 1.8

of eight single 1s, 0s or any combination of the two. Each unit of digital data is called a bit. The word is short for "binary digit." There are eight bits in the data shown in Figure 1.8.

The next new term is *byte*. A byte consists of eight bits that are communicated at the same time. The bits are said to be in *parallel*. We can, therefore, describe the digital data output and input by the Apple computer as data bytes. The Apple computer will either output or input digital data in units of eight bits. This information will become important when we discuss how to select exactly how many 1s and 0s will be present in the data byte. Figure 1.9 shows some different data bytes.

As we will see, the *position* of bits in a byte is significant, because a byte represents a number in base 2. Just as the position of numerals in a base 10 number determines their values as powers of 10, the position of numerals in a base 2 number (or bits in a byte) determines their values as powers of 2. For example, in the base 10

0	0	0	0	0	0	0	0	BYTE 1
1	1	0	1	0	1	1	0	BYTE 2
0	0	1	1	1	1	0	0	BYTE 3
1	0	0	0	1	1	1	1	BYTE 4
1	1	1	1	1	1	1	1	BYTE 5

Eight bits per unit of output or input

Examples of different data bytes. Each byte consists of a combination of eight digits, either logical 1 or logical 0, called bits.

Figure 1.9

number 357, the numeral 3 represents 3 × 10^2. In the number 35, the same numeral represents 3 × 10^1. Likewise, in the base 2 number 100, the numeral 1 represents 1 × 2^2, and in the number 10, it represents 1 × 2^1. In both number systems, the rightmost numerals represent powers of zero. As data bytes, all base 2 numbers are filled out to eight places by adding zeroes. Our examples above would thus be represented as 00000100 and 00000010. Of these eight bits, the leftmost bit represents the highest power of 2 (or *weight*), 2^7, and is called the *most significant bit* (*MSB*). The rightmost bit, 2^0, is called the *least significant bit* (*LSB*).

Sometimes, when discussing data input and output, people will refer to the data as "data words." A data word is the number of bits the computer treats as a unit and processes simultaneously. For the Apple computer the data word will be eight bits, and so data *byte* and data *word* are synonymous (for eight-bit computers). For other types of computers, the data word may be longer than eight bits.

Another term that is used often in computer interfacing and computer control is *nibble*. A nibble consists of four bits of digital data. Figure 1.10 shows some examples of data nibbles. In the Apple computer the data byte will consist of two parallel (simultaneous) nibbles. This is shown in Figure 1.11.

$$\underbrace{\begin{array}{cccc}0&0&0&0\\1&0&1&0\\0&1&0&1\\1&1&0&0\\0&0&1&1\end{array}}_{\text{4 BITS = NIBBLE}}\qquad\begin{array}{l}\text{NIBBLE 1}\\\text{NIBBLE 2}\\\text{NIBBLE 3}\\\text{NIBBLE 4}\\\text{NIBBLE 5}\end{array}$$

Examples of different data nibbles. Each nibble consists of four bits.

Figure 1.10

$$\underbrace{\underbrace{0\quad 0\quad 1\quad 1}_{\text{NIBBLE}}\qquad\underbrace{1\quad 1\quad 0\quad 0}_{\text{NIBBLE}}}_{\text{BYTE = 2 NIBBLES = 8 BITS}}$$

A data byte consists of two parallel nibbles.

Figure 1.11

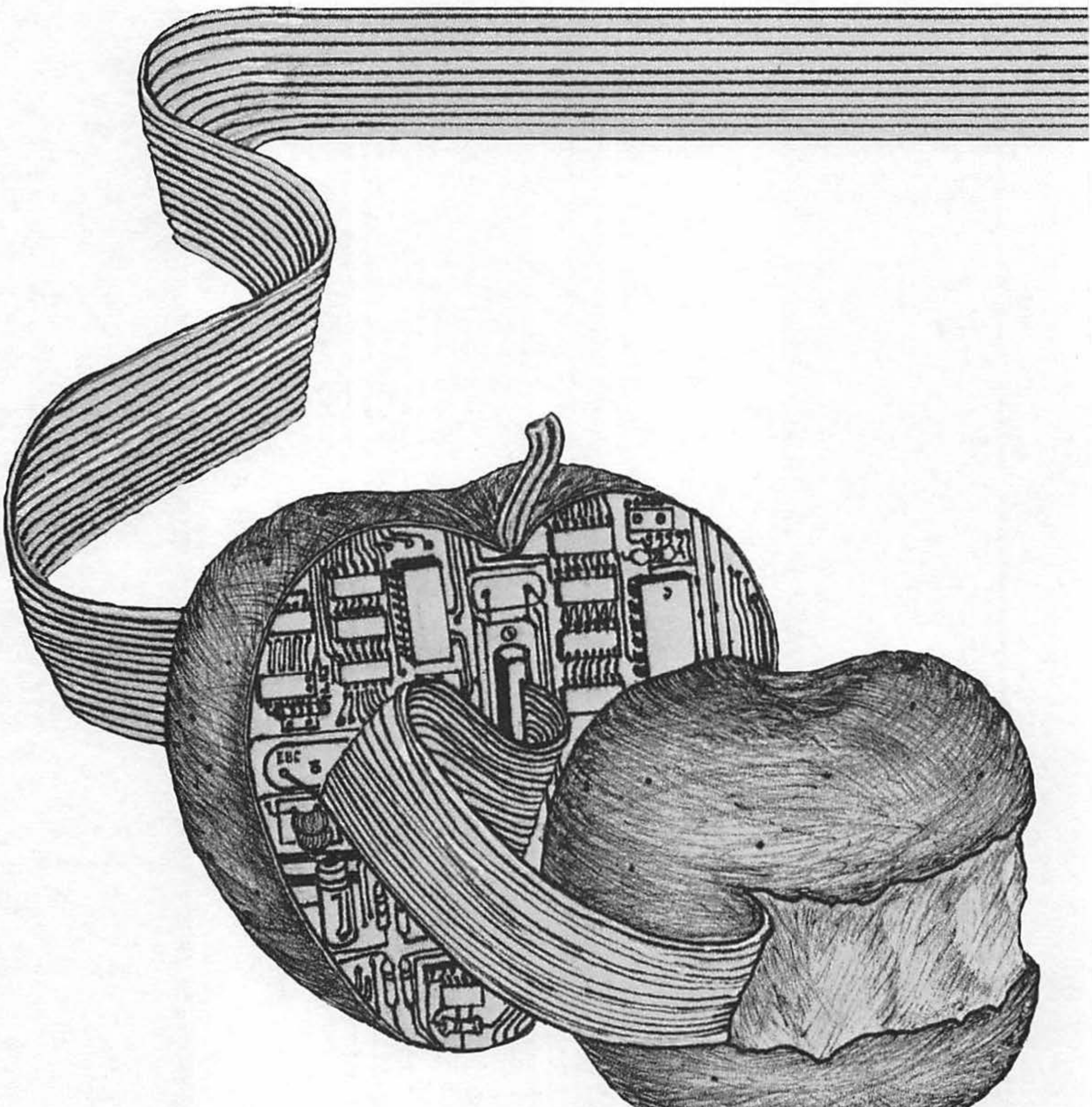
1.4: SUMMARY

In this chapter we have introduced the topics that will be covered in this text. The chapters to come will show how to correctly input and output information from the Apple computer to control external devices. We next discussed the two basic concepts of computer control, using the example of a home security system. In this example it was shown that the computer must send information out to the external device and the computer must be capable of getting information back from the external device.

The next section of the chapter was devoted to introducing some new vocabulary that is used when discussing computer control. This vocabulary is used in industry and in the literature. It is presented here so you may start to learn the language that is common to computer control. We will use all of these terms in this text. The definitions given are summarized below for your convenience.

Words Defined in This Chapter

1. *Input:* The computer receives information from an external device.
2. *Output:* The computer sends information to an external device.
3. *Transducer:* Any device that converts a physical event into electrical information.
4. *Digital:* Having discrete output values. The selection of a television station is digital because the channels that can be tuned in are a certain preset value. In a digital computer there are discrete voltage values for the information.
5. *Binary:* Having two possible states or values. In a digital computer there are two and only two voltage levels associated with the information processed. *Binary* refers to this fact.
6. *Data:* This is the information that is output, input, or processed by the Apple computer.
7. *Bit:* A bit is one of the eight places in a digital data word used by the Apple computer. A bit will be either a logical 1 or a logical 0.
8. *Byte:* A byte consists of eight parallel bits.
9. *Nibble:* A nibble consists of four parallel bits. Two parallel nibbles will equal one byte.

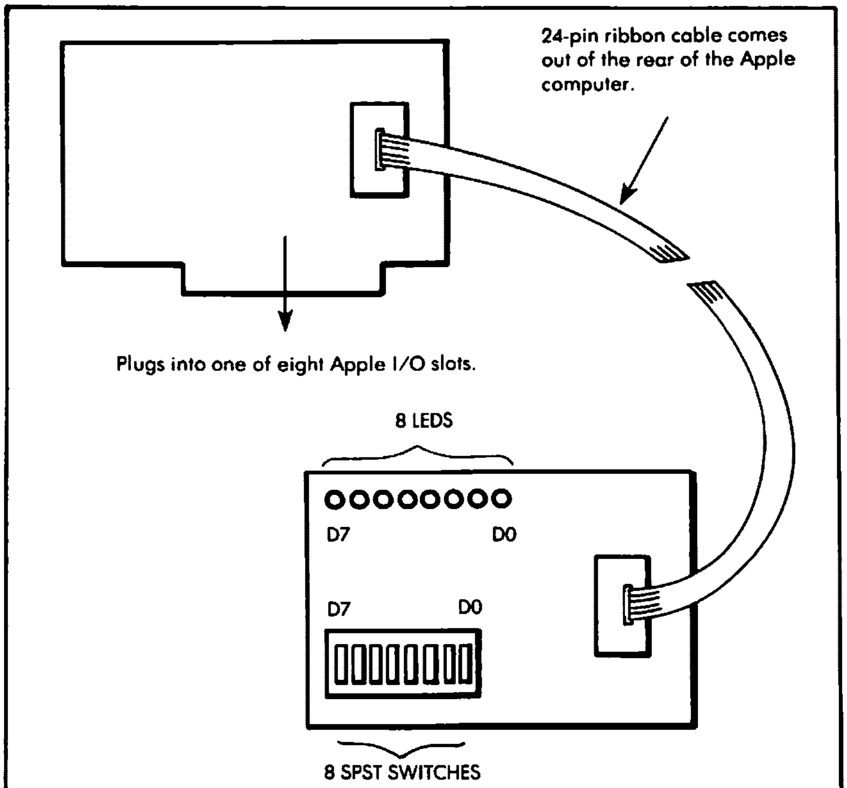

EBC

Chapter 2

Software for Output from the Apple

IN THIS CHAPTER we will discuss the programming necessary to output digital information from the Apple computer to the outside world. We start the discussion assuming the reader is familiar with the versions of the BASIC programming language used on the Apple. There are no other assumptions made. Each new topic will be clearly discussed, and examples will be given.

The examples given in this chapter are designed to be used with the Creative Microprocessor Systems (CMS) I/O system. The CMS I/O board is a visual means of testing your Apple I/O programs. It will install directly into the Apple computer, and data can be output and input with it. You do not have to purchase this board to benefit from this chapter. However, you will learn more about writing this kind of program if you use the CMS I/O board (or an equivalent device) while studying the examples given. Information regarding availability of the CMS I/O system for the Apple computer is given in Appendix D.


24-pin ribbon cable comes out of the rear of the Apple computer.
Plugs into one of eight Apple I/O slots.
8 LEDS
D7
D0
D7
D0
8 SPST SWITCHES

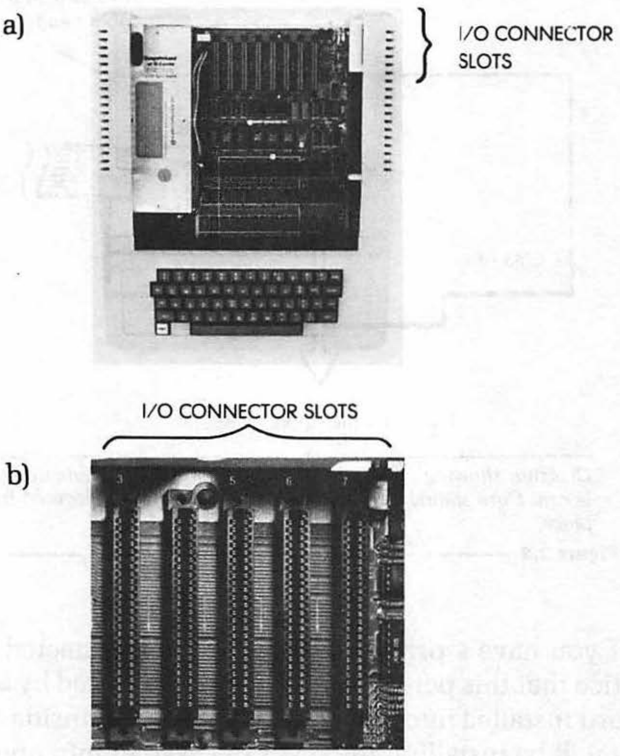
Pictorial diagram of the Creative Microprocessor Systems I/O system PC boards. One board, CMS AP1, is inserted into one of the eight I/O slots in the Apple computer. The second board, CMS AP2, is connected to board 1 via a 24-pin ribbon cable. The second board is intended for user interaction with the Apple I/O system.

Figure 2.1

2.1: INSTALLING THE CMS I/O SYSTEM

Before we start learning the software required for inputting and outputting data with the Apple computer, we must first learn how to install the hardware into the computer. This discussion will bring out some important general information about the physical connections necessary to perform computer control with the Apple.

The CMS I/O system comprises two printed circuit (PC) boards and an interconnecting cable. One of the PC boards plugs directly



Photograph above shows where the eight I/O connectors in the Apple computer are located. Photograph below shows several of the slots in detail. These I/O slots are the means by which external devices communicate with the computer.

Figure 2.2

into the Apple computer. The second PC board will connect to the first board via a 24-pin ribbon cable. See Figure 2.1 for a diagram of these two boards. Let us now go over the details of how to connect the first board directly to the Apple computer.

Figure 2.2 shows the inside of an Apple computer; the top cover has been removed. In Figure 2.2(a) there are eight physical connectors ("slots") in a row. (Figure 2.2(b) shows a section in detail.) Some of these connectors may already have PC boards installed. It is through these connectors that external instruments are controlled by the Apple computer.

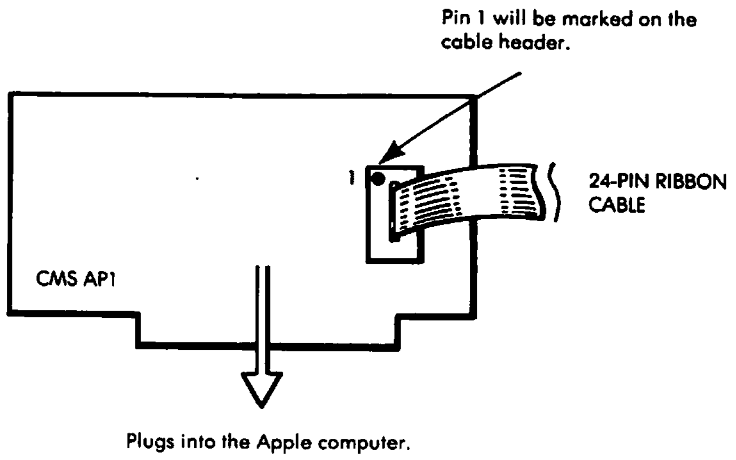

Pin 1 will be marked on the cable header.
1
24-PIN RIBBON CABLE
CMS AP1
Plugs into the Apple computer.

Diagram showing how to install the 24-pin ribbon cable into the CMS I/O board. Care should be taken to insure that pin 1 is located in the correct place.

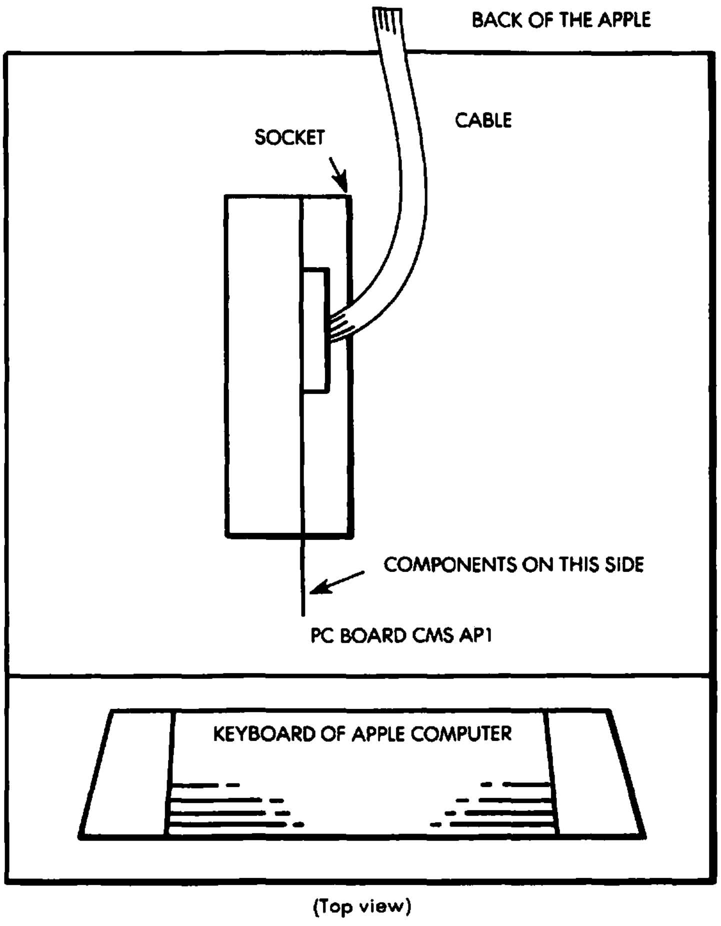
Figure 2.3

If you have a printer or a disk drive connected to the Apple, notice that this peripheral device is connected by a cable to a PC board installed into one of the eight sockets inside the computer. We will be installing the CMS I/O system into one of the empty sockets.

It should be mentioned before we begin that static electricity is easily generated, and can seriously damage a computer's circuitry. Rodnay Zaks' *Don't! (or How to Care for Your Computer)* (Sybex, 1981) describes ways of avoiding this problem.

The procedure for installing the CMS I/O system is as follows:

1. Turn off the power on your computer and any other peripherals. Always turn off the power when installing any hardware into the computer.
2. Remove the top cover to expose the eight connectors.
3. Install the 24-pin ribbon cable into the 24-pin socket on the CMS I/O board labeled "CMS AP1." It is important to install the 24-pin connector with pin 1 in the correct place. See Figure 2.3 for a diagram showing how to physically install the cable into the respective PC boards.



Block diagram showing how to install the CMS AP1 board into one of the I/O slots inside the Apple computer. Be certain the computer is turned off before you insert or remove any PC board from the computer I/O slots.

Figure 2.4

4. Install the PC board labeled "CMS AP1" into one of the empty sockets. I/O slot 5 would be a good choice as it is seldom used. It is important to install the PC board correctly, so that the cable is at the rear. Figure 2.4 shows a diagram of how the PC board is to be physically installed into the Apple computer.

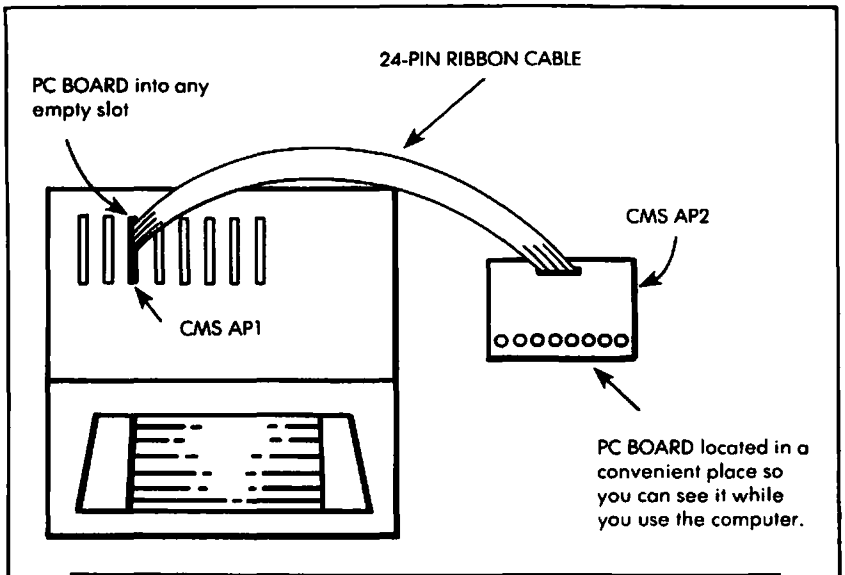

24-PIN RIBBON CABLE
PC BOARD into any empty slot
CMS AP2
CMS AP1
PC BOARD located in a convenient place so you can see it while you use the computer.

Diagram showing how the CMS I/O system will appear with the Apple computer once it is correctly installed.

Figure 2.5

5. Make a note of which I/O slot the PC board was installed into. The slots are numbered from 0 to 7. Refer to Figure 2.2 to see where the number of each I/O slot is printed. Later in this chapter we will see why this information is important.
6. Run the ribbon cable through one of the slots at the back of the Apple computer frame. The cable will then connect to the second PC board, labeled "CMS AP2."
7. Connect the remaining end of the 24-pin ribbon cable into the second PC board. Care should be taken to insure that pin 1 of the cable is connected to pin 1 of the socket on the second PC board.
8. At this time your system will appear as shown in Figure 2.5. Now your system is ready for use with the CMS input and output boards.
9. Replace the top cover of your system and turn on the power.

2.2: THE POKE INSTRUCTION

It was stated earlier that the reader is assumed to be familiar with BASIC. You are not expected to be an expert programmer, but you should know enough to be able to write simple BASIC software and run programs using the Apple computer. We will introduce new information based on that assumption.

To get started, you must know how to direct digital information from a BASIC program out to the CMS I/O board. If you can direct the information to the CMS I/O board, then you can direct information to any type of I/O board. The CMS I/O system is used as a tool to promote understanding. After this initial discussion, several examples and problems will be given. These are designed to let the reader get "hands-on" experience using the software necessary for outputting information on the Apple computer. The CMS I/O board will allow the user to instantly verify whether or not the software written is operating correctly.

Let's Get Started

To output digital information from a BASIC program, we will use an instruction that may be new to some Apple users: POKE. A full description of the POKE instruction can be found in the Apple user's manual. We will describe this instruction in enough detail here so that you will feel comfortable in its use.

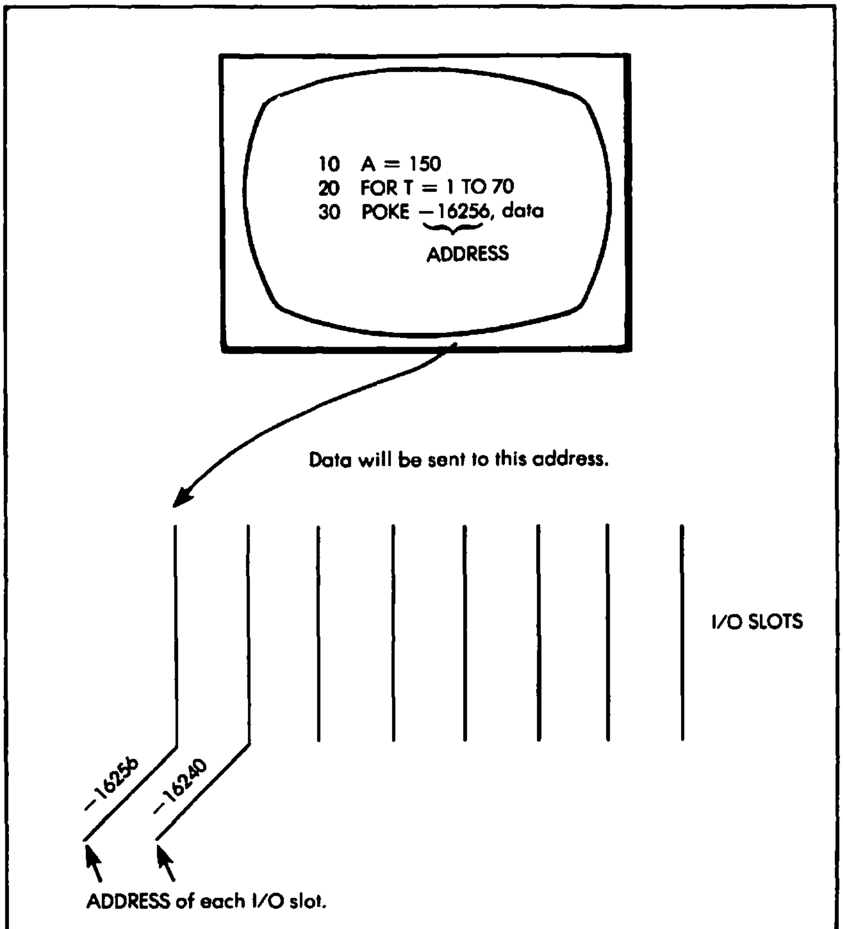
The form of the POKE instruction is:

```
POKE address, data
```

That is, if we were using the POKE instruction in a BASIC program, it would appear as shown. This will be made clearer as we present different examples of the POKE instruction. Let us now discuss the two parts of the POKE instruction, *address* and *data*.

The address used by the POKE instruction will indicate the physical space in the Apple computer where the information will be sent out from the program. For example, when we put the CMS I/O card into a particular I/O slot of the Apple computer, a number is specified as the "address" of that slot.

More than one number may be specified for a particular I/O slot.



As the program is executing, the POKE address will be electrically matched with the address of the correct I/O slot for outputting data.

Figure 2.6

However, the complexities of I/O addressing are beyond the scope of this book. Therefore, we will use only one of the available address numbers. A detailed discussion of the I/O system for the Apple computer can be found in the *Apple II Reference Manual* (Apple Computer Inc., 1979, 1981).

An address in computer programming can be likened to the address of a house. The only way the mail carrier knows where to deliver a letter is by matching the address on the envelope with the address on the house. To extend this analogy, think of the address on the envelope as the address specified in the POKE instruction. The address of the house will be the address of the I/O slot. An Apple computer will match the address of the "envelope" (POKE address) with the address of the "house" (I/O slot address) and deliver the information. (See Figure 2.6.)

The second element in the POKE instruction is the data. This will be the actual digital information to be sent to the address specified. In our analogy of the address and the mail carrier, we may think of the data as the actual letter that was delivered.

The address specifies where in the entire system to send the information. The data is the information. With this broad overview of the POKE instruction, let us get into some specifics.

2.3: FORMING THE ADDRESS FOR THE POKE

We have discussed the concept of the address in the POKE instruction, but we have not shown exactly how to use the address. In this section we will show how to calculate the correct address, depending on which I/O slot in the system your output device or PC board is plugged into. It should be noted that calculating an address can be much more complicated than we will show. However, if you are just starting to learn the interfacing of a computer, you will find this introduction to be a good entry point. What we will show here will work and work well. Only when you begin to tackle more sophisticated interfacing problems will a greater understanding of address calculation be required.

To begin, you must first know that each of the output slots in the Apple computer has a limited set of address numbers associated with it. Unlike a street address, the address in a computer system can be negative. That is, it may be a number preceded by a minus (—) sign. It is critical in a program not to leave off the minus sign on the address if it requires one.

Figure 2.7 is a list of the I/O slots and their corresponding addresses in the Apple computer.

Remember that there are actually several numbers which may be used to specify a particular I/O slot. The addresses listed in the table are those used in this text.

Slot Number	Address
0	−16256
1	−16240
2	−16224
3	−16208
4	−16192
5	−16176
6	−16160
7	−16144

The Apple I/O connector slots and their addresses.

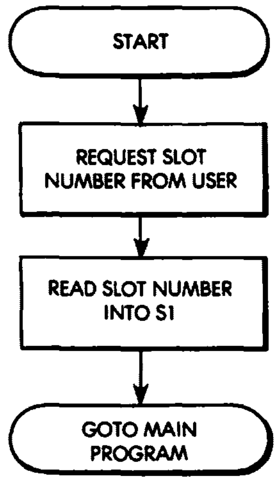
Figure 2.7

Using this information, we can select the proper address for the POKE instruction simply by knowing into which I/O slot the output device is installed. For example, if the CMS I/O board were plugged into slot 3, then the address would be equal to −16208. The corresponding form of the POKE instruction would be:

```
POKE −16208,data
```

Notice that in this example of the POKE instruction, only the address was specified. The content of the data is not important to us yet. The data will be sent to whichever output device "resides" at the specified address. In this case, it will be the CMS I/O board. Therefore, the data will be electrically sent to it.

You may want the flexibility of using the same output device in any of the Apple I/O slots. If your BASIC program includes a POKE statement with a specific address as an input value, then that address will need to be altered depending on which slot the output device is connected to when the program is run again. For example, if we had written a BASIC program to communicate with a CMS I/O board plugged into slot 4, the address in the POKE instruction would be −16192. Suppose we saved that program. At some later time we wish to recall the program and run it, but now the CMS I/O board has been changed to a different slot, say slot 5.



Flowchart showing the major steps involved in entering the POKE address as a BASIC variable.

Figure 2.8

Before the program will execute correctly, the address at the POKE instruction will have to be changed to −16176.

However, this particular problem can be avoided by letting the program determine the address for the POKE instruction. This could be done by following the flowchart shown in Figure 2.8. The BASIC program prints a message asking the user which I/O slot the card is installed into. Based on that answer, the program will set the variable S1 equal to the correct address obtained from a table of addresses.

Figure 2.9 shows one way a program could be written to realize the flowchart of Figure 2.8. A point to be stressed here is that the address of the POKE instruction can be a BASIC variable. After the address is calculated by the program the form of the POKE instruction would be:

```
POKE S1,data
```

where S1 is the number that corresponds to the address of a particular I/O slot.

a)

```
 10  PRINT "WHAT SLOT NUMBER ARE YOU USING?"
 20  INPUT A1
 40  IF A1 = 0 THEN S1 = -16256
 50  IF A1 = 1 THEN S1 = -16240
 60  IF A1 = 2 THEN S1 = -16224
 70  IF A1 = 3 THEN S1 = -16208
 80  IF A1 = 4 THEN S1 = -16192
 90  IF A1 = 5 THEN S1 = -16176
100  IF A1 = 6 THEN S1 = -16160
110  IF A1 = 7 THEN S1 = -16144
120  REM: GOTO main program
```

b)

```
 10  PRINT "INPUT THE SLOT NUMBER"
 20  INPUT A1
 30  LET S1 = -16256+(16*A1)
 40  REM: GOTO main program
```

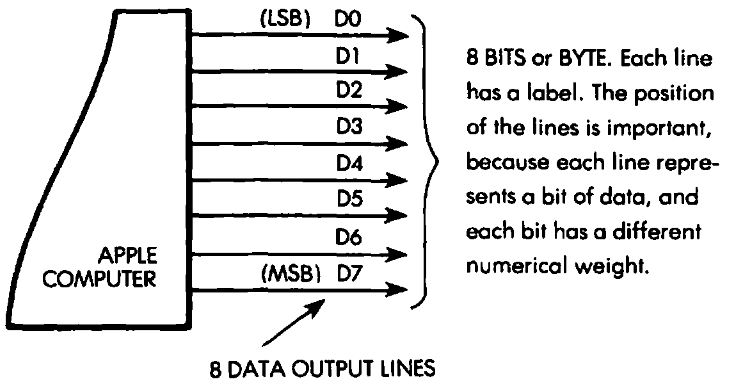
a) A BASIC program to allow the user to enter the slot number of the output device. The program will then look up the correct slot address. In this way the programmer can use the same program regardless of which I/O slot the I/O device may be installed into.

b) Another technique for realizing the flowchart of Figure 2.8.

Figure 2.9

2.4: CALCULATING DATA FOR THE POKE

Before we examine the calculation of the data portion of the POKE instruction in detail, let us discuss exactly what the data will do. The output section of the Apple computer *supports* (that is, performs or allows) what is called 8-bit, or byte, output. What this means is that there are eight physical, electrical lines over which information is passed from the Apple computer to the external output circuits. (See Figure 2.10.)


(LSB) D0
D1
D2
D3
D4
D5
D6
(MSB) D7
APPLE COMPUTER
8 BITS or BYTE. Each line has a label. The position of the lines is important, because each line represents a bit of data, and each bit has a different numerical weight.
8 DATA OUTPUT LINES

Block diagram showing that there are eight physical output lines over which data will be transferred from the Apple computer to the external output device. The output lines originate from any of the eight I/O slots located inside the Apple computer.

Figure 2.10

It is not really important to know *why* there are only eight lines. To perform computer interfacing and computer control, we simply make use of this fact. When the POKE instruction is executed in a BASIC program, the data that is output is contained in these eight bits.

All of the information that can be transferred to the output device in a single POKE instruction is included in the eight bits. This means that a person using an output device with the Apple must keep in mind what each of the eight bits in the transfer does, when using the POKE instruction.

For example, one of the eight bits might turn on a light. Another may sound an alarm. Yet another might open a door, etc. Further, all eight bits may be used together to form a unique combination to which the output device may respond. The main point is that the programmer using the POKE instruction must be able to generate any information wanted with the eight bits.

We will concentrate, in these early discussions, on showing how to set any bit that will be used in the POKE instruction to a logical 1 or a logical 0. These terms were introduced in Chapter 1, but if you are new to the field of digital logic, they still may not mean much. However, if you are interested in learning how to electrically control hardware with a computer, then you must become aware of exactly what these terms mean.

As you progress through the chapters of this book, you will develop a greater understanding of these and other related concepts. For now we will define logical 1 and logical 0 as precisely as possible, and use the terms in our explanation. When the examples of outputting information are given, you will have a good idea of exactly how to set any of the eight bits to a logical 1 or a logical 0. This will be true even if you do not yet fully understand exactly what a logical 1 or logical 0 means to the external device.

Logical 1 and 0 can be defined by the ranges of voltage levels they correspond to. These definitions will apply to the Apple computer and most other home computers. A logical 1 is a voltage level greater than 2.4 volts and less than 5.0 volts. Any time a digital line is set to a logical 1, the voltage on that line will be within these limits. A logical 0 is a voltage level of less than 0.8 volts and greater than 0.0 volts.

These definitions will help you relate physical actions to the logical definitions we will deal with in computer control. For example, suppose we say that all eight bits or lines are a logical 1. This means that all of the lines are set to a voltage that will fall within the range specified for a logical 1.

With this brief introduction, let us discuss how any of the eight output lines can be set to a logical 1 or a logical 0 under software control. There are two key facts to keep in mind at this time. First, we must remember that all eight bits are output to the Apple I/O slot connector at the same time. That is, all eight bits are output in "parallel." This was illustrated in Figure 2.10. The other important point to note is that the position of the lines within the 8-bit parallel output is significant. The lines are labeled D0–D7. The D stands for data, and the number 0–7 stands for the position of the line within the eight parallel bits. D7 is the most significant bit (MSB), and D0 is the least significant bit (LSB).

Using these two pieces of information, let us examine the eight bits or lines. Remember from Chapter 1 that when eight bits are taken together at the same time, they are called a "byte." We will

Bit Position	Weight
D0	1
D1	2
D2	4
D3	8
D4	16
D5	32
D6	64
D7	128

Each bit position has a numerical weight, corresponding to its value as a power of 2.

Figure 2.11

use this definition throughout the remainder of the text. In a single byte there are eight individual bits.

Since the bits are parallel, or independent of each other, we can use software to set any of the bits to a logical 1 or a logical 0, regardless of the logical state or value of any other bit. In fact, we *must* be able to do this to achieve computer control. To accomplish this, the software must have some means of logically setting any bit to a 1 or a 0. Each bit position in the byte is assigned a number or weight equal to a power of 2. (Remember that a byte represents a binary number.) Each bit position n is weighted 2^n. The weight of D0, 1, is equal to 2^0. The weight of D7, 128, is equal to 2^7. The bit positions and their corresponding weights are shown in Figure 2.11.

When we wish to set a particular data bit to a logical 1, the weight of the bit is summed and used as the data in a POKE instruction. For example, suppose the bits D0 and D2 were set to a logical 1. The weights of these bits would be summed, giving the following results:

$$D0=1, D2=4: SUM = 1+4 = 5$$

The resulting value to be used in the POKE instruction as data would be 5. The POKE instruction would appear as

```
POKE address,5
```

As we have seen, the address is set according to the specific I/O

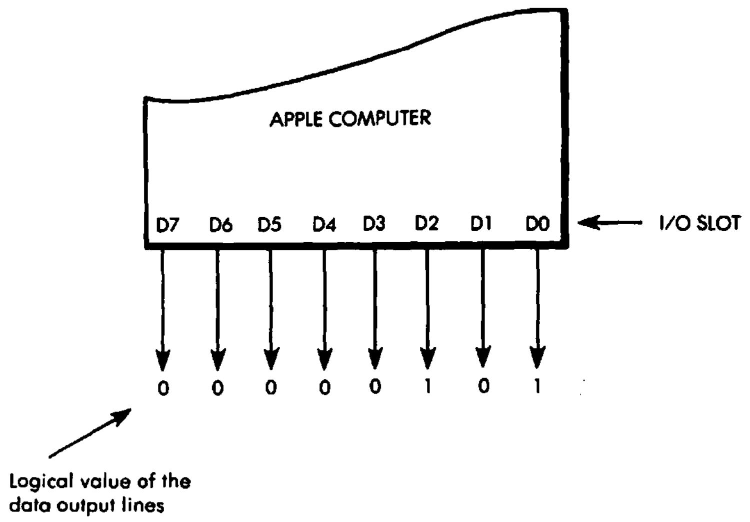


Diagram showing the logical conditions of the eight output lines that would result if the POKE data were equal to 5.

Figure 2.12

slot where data is to be sent.

If we were to look at the logical conditions of the data byte to be output with the sum equal to 5, the result would appear as shown in Figure 2.12. Notice, in this diagram, that all the data bits that were used in the summation are set to a logical 1. All other data bits are set to a logical 0.

In short, the smallest data byte is output when no weights are summed; that is, when all the bits in the byte are set to a logical 0. This weight is 0. The largest data byte is output when all the weights are summed; when all the bits are set to logical 1. This summation is equal to 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255. All possible valid weights are within the 0—255 range, and each number represents a unique combination of weights.

Let us try some examples of setting the correct data bits to a logical 1 by summing the appropriate weights. We wish to set data bits D0, D4, and D7 to a logical 1. To do this the weights of the bits we want are summed: D0=1, D4=16, D7=128. The resulting summation would be 1 + 16 + 128 = 145. A POKE instruction would appear as follows:

```
POKE address,145
```

One more example. Suppose we wish to set the data bits D2, D5, and D6 to a logical 1. We would again sum the weights of the bits we want: D2=4, D5=32, and D6=64. The resulting summation would be 4 + 32 + 64 = 100. A POKE instruction would appear thus:

```
POKE address,100
```

By combining the proper choice of data and the correct address we can set any bit at any output slot on the Apple to a logical 1 or logical 0. Further, we can accomplish this using only software. We will discuss the hardware in Chapter 4 of this text. At this time you should simply assume the hardware will respond correctly if the software is correct.

Before starting experiments with the CMS I/O board, let us examine one more aspect of setting the correct data. A program similar to the one that was written to input any address can be written to allow any bit to be set to a logical 1 or a logical 0. A flowchart for this program is shown in Figure 2.13.

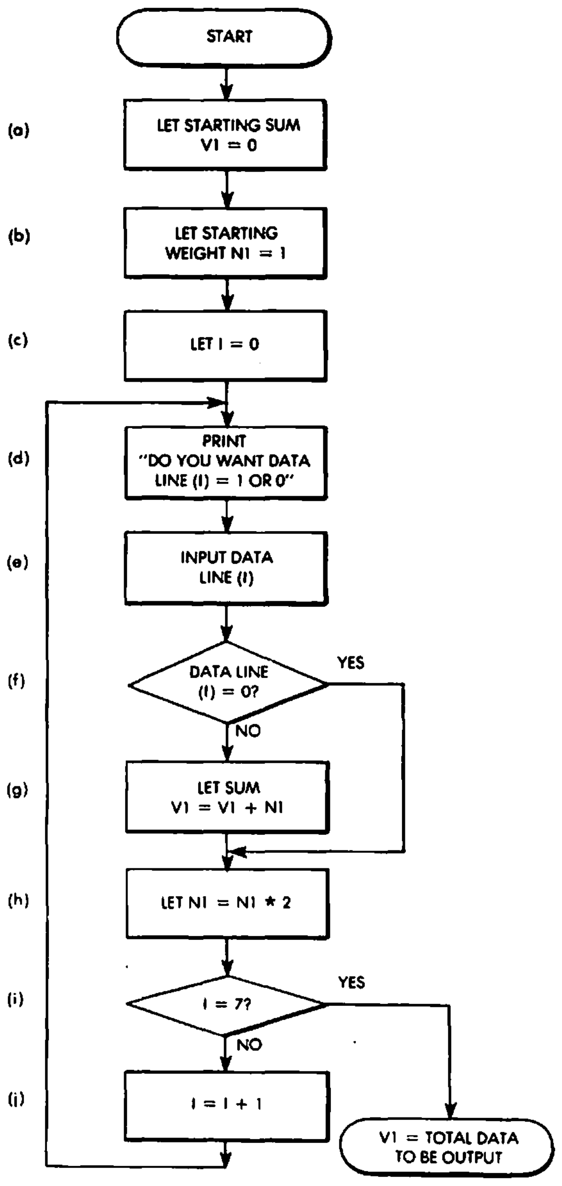
Let us examine this flowchart in some detail. In Figure 2.13 the first two steps, (a) and (b), will initialize the variables for the starting sum, V1, and the starting weight, N1. The flowchart then enters a loop starting with step (c). The loop variable will be I. In step (d) the program will ask the user for the logical value of each data bit. If the bit is to be set to a logical 1, then the weight of the bit will be added to a sum in step (g). The result of the sum is stored in the variable V1. In step (h) the starting weight (N1) is multiplied by 2. This will set the weight equal to the next bit to be tested. After V1 is computed, the POKE instruction will appear as:

```
POKE address, V1
```

A program to realize the flowchart of Figure 2.13 is given in Figure 2.14.

2.5: EXPERIMENTS WITH THE CMS I/O SYSTEM

In this section of the chapter we will actually write and execute programs that will perform output using the POKE instruction.



Flowchart showing the sequence of events necessary to set any of the data output lines to a logical 1 or a logical 0. The sequence begins with D0 (the lowest weight) and ends with D7 (the highest weight).

Figure 2.13

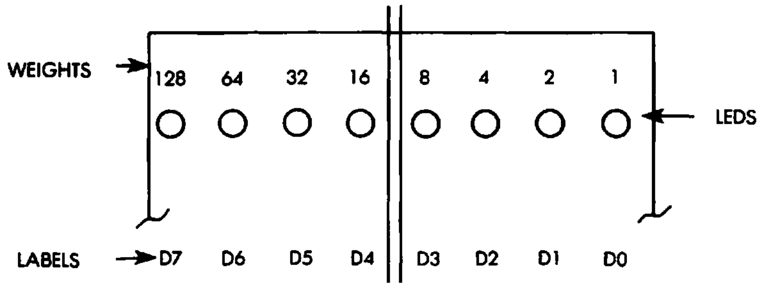
```
10  LET V1 = 0
20  LET N1 = 1
30  FOR I = 0 TO 7
40    PRINT "WHAT IS THE VALUE OF D";I;" 1 OR 0"
50    INPUT D(I)
60    IF D(I) = 0 THEN 80
70    LET V1 = V1 + N1
80    N1 = N1*2
90  NEXT I
```

BASIC program to realize the flowchart given in Figure 2.13.

Figure 2.14

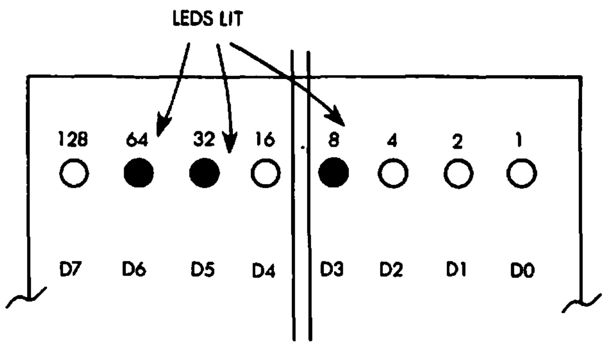
These programs are designed to show exactly how to use the POKE instruction with the Apple computer. All of the following experiments assume that the user has installed the CMS I/O system into the Apple computer in one of the eight output slots in the back of the machine, according to the procedure described at the beginning of the chapter.

The CMS output board (AP2) is designed with eight light-emitting diodes installed. These LEDs are labeled D0, D1, D2, D3, D4, D5, D6, and D7. (See Figure 2.15.) When an LED is lit, it in-



The output section of the CMS I/O board has eight LEDs. The weight and data line label of each LED are also shown.

Figure 2.15



(a)

D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	0	1	0	0	0

(b)

a) LEDs D6, D5 and D3 are lit. All other LEDs are off. The number input to produce this combination was 104.

b) Corresponding logical levels of the data lines needed to produce the LED pattern in (a).

Figure 2.16

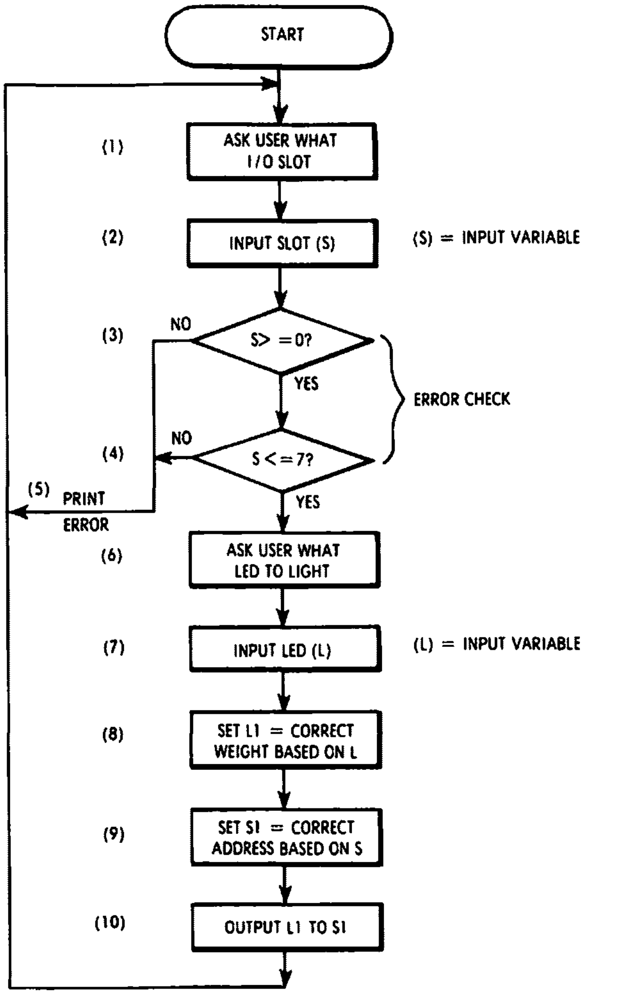
dicates that the corresponding bit position in the data byte was set to a logical 1. When an LED is not lit, the corresponding bit position is a logical 0.

For example, Figure 2.16 (a) shows a diagram of three LEDs that are lit. (The darkened LEDs are lit.) In this diagram bits D3, D5, and D6 are set to a logical 1. The data byte would appear as shown in Figure 2.16 (b).

2.6: EXAMPLE 1: LIGHTING A SINGLE LED

In this first "hands-on" example of Apple output, we will write a program that will turn on any of the eight LEDs located on the CMS I/O board.

The program we are about to write will first ask you which I/O slot the CMS system is installed into. Next, the program will ask



Flowchart for inputting the I/O slot number and the LED to be turned on at the external I/O device.

Figure 2.17

you which LED you want to light. Finally, the program will light the selected LED on the CMS I/O board.

A flowchart for the program is shown in Figure 2.17. Let us discuss the steps in the flowchart. After this discussion we will show the actual program listing.

Step 1. The program will write a message asking the user which I/O slot the CMS system is installed into.

Step 2. The program will now input the slot number.

Steps 3 and 4. The program now checks for a valid slot number between 0 and 7. If the slot number is not valid, the program will jump to step 5. If the slot number is valid, the program will go to step 6.

Step 5. In this step the program will print an error statement to the user indicating that the slot number was not valid. The program will then go back to step 1, and the user will enter another number.

Step 6. At this point the program will write a message to the user asking which LED the user wishes to light.

Step 7. The user will input the LED number.

Step 8. In this step the program will set the output byte to the correct weight corresponding to the LED that the user wants to light.

Step 9. In this step the program will set the variable S1 to the correct address corresponding to the I/O slot that was input in step 2.

Step 10. The software will now output the correct byte to the I/O slot. In examining the program listed in Figure 2.18, note that if a valid LED number in the range from 0 to 7 is not input in step 7, all of the LEDs on the CMS I/O board will light. This will be an error indication. As an alternative, we could change the program to look for this error condition in software. To do this, we would simply write another loop like the one in steps 3 and 4 above. A program to realize the flowchart of Figure 2.17 is presented in Figure 2.18.

2.7: EXAMPLE 2: LIGHTING A COMBINATION OF LEDS

The second hands-on example will allow the user to light any combination of the LEDs on the CMS I/O board. You are asked to determine which LEDs you want on, and then calculate the output variable weight necessary to accomplish this.

For example, suppose you wished to turn on LEDs 0, 5, and 7 on the CMS I/O board. The output weight to POKE would be

```
10   REM: THIS PROGRAM WILL OUTPUT A BYTE TO THE CMS I/O BOARD
20   REM: THAT IS INSTALLED INTO AN APPLE COMPUTER. THE BOARD IS
30   REM: INSTALLED INTO ANY I/O SLOT 0-7.
40   REM
50   REM: START OF THE PROGRAM
60   PRINT "WHICH I/O SLOT IS THE CMS I/O BOARD INSTALLED INTO? 0-7 ";
70   INPUT S
80   IF S>=0 OR S<=7 GOTO 110
90   PRINT "INPUT SLOT ERROR ";S
100  GOTO 60
110  PRINT "WHICH LED ON THE CMS I/O BOARD DO YOU WANT LIT? ";
120  INPUT L
130  LET L1=255
140  IF L=0 THEN L1=1
150  IF L=1 THEN L1=2
160  IF L=2 THEN L1=4
170  IF L=3 THEN L1=8
180  IF L=4 THEN L1=16
190  IF L=5 THEN L1=32
200  IF L=6 THEN L1=64
210  IF L=7 THEN L1=128
220  IF S=0 THEN S1=-16256
230  IF S=1 THEN S1=-16240
240  IF S=2 THEN S1=-16224
250  IF S=3 THEN S1=-16208
260  IF S=4 THEN S1=-16192
270  IF S=5 THEN S1=-16176
280  IF S=6 THEN S1=-16160
290  IF S=7 THEN S1=-16144
300  REM
310  POKE S1,L1
320  PRINT
330  GOTO 110
```

BASIC program to realize the flowchart given in Figure 2.17.

Figure 2.18

equal to the weight of D0 plus the weight of D5 plus the weight of D7. This equals D0=1 + D5=32 + D7=128 = 161. If the number 161 were POKED to the correct address, LEDs D0, D5 and D7 would light. This is analogous to saying that the output lines D0, D5, and D7 are set to a logical 1 in the output data byte.

A flowchart for the program to be written is shown in Figure 2.19. A program to realize this flowchart is given in Figure 2.20.

With the CMS I/O system installed in the Apple computer, try out the program given in Figure 2.20. This system is simply a means of visually verifying the calculated results. Use the LED patterns (a–g) shown and calculate the weight to be input. Verify your answer by executing the program of Figure 2.20. The answers to the patterns are given at the end of pattern (g).

a. LEDs D0, D4, D7 are lit.

b. LEDs D0, D3, D5, D6 are lit.

c. All LEDs are lit.

d. No LEDs are lit.

e. LED D6 is lit.

f. LEDs D1, D3, D5, D7 are lit.

g. LEDs D0, D2, D4, D6 are lit.

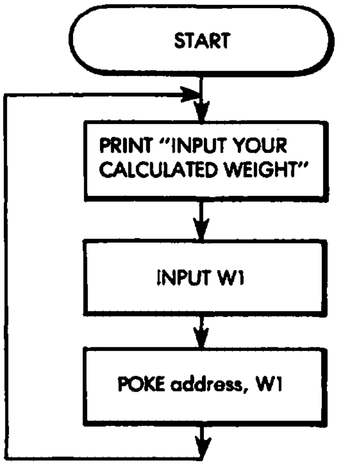
Answers to the LED patterns:

a = 145, b = 105, c = 255, d = 0, e = 64, f = 170, g = 85

2.8: EXAMPLE 3: A COUNTING PROGRAM

In this third hands-on example, we will write a program to light up the LEDs on the CMS I/O board in a specified sequence. The program will perform the following sequence. At first, all of the LEDs will be turned off. Next, the number 1 will be output from the computer to the I/O board. This will turn on the LED that corresponds to a weight of 1. Only the LED labeled D0 will be lit.

The program will delay long enough for the user to see that the correct LED is lit. Next, the program will output the number 2 from the computer to the I/O board. This will turn on the LED that corresponds to a weight of 2. The LED labeled D1 will be lit at this



Flowchart for inputting the weight to POKE at the output slot.

Figure 2.19

```
20  REM: FIRST INPUT THE CALCULATED WEIGHT
30  PRINT "INPUT THE CALCULATED WEIGHT"
40  INPUT W1
50  POKE -16176,W1
60  REM: ASSUME I/O SLOT 5 WAS USED
70  GOTO 30
```

BASIC program to realize the flowchart given in Figure 2.19.

Figure 2.20

time. Again, the program will delay long enough for the user to see the LED pattern on the I/O board.

The program now outputs the number 3. This will turn on the LEDs corresponding to a weight of 3. LEDs D0 and D1 will be lit. As before, the program will now delay to give the user time to see the LED pattern.

This process is repeated, each time incrementing by 1 the number to be output. The result will be the turning on and off of the LEDs in a manner that will show the output weight increasing by 1 for each POKE statement.

The flowchart for this program is shown in Figure 2.21. A BASIC program to realize this flowchart is given in Figure 2.22.

If you know a little more BASIC than we have tested so far, you might try the following variations. After you have had a chance to load and run the program of Figure 2.22, try to realize the program using fewer BASIC statements. Next, try speeding up and slowing down the wait time for the program. Note the effect on the output display LEDs.

2.9: EXAMPLE 4: A TRAVELING LIGHT

In our final example a program will be written to make the LEDs turn on and off in a certain sequence different from that of example 3. The sequence will give the visual illusion that the light on the CMS I/O board is traveling. This is the same effect one encounters in a marquee sign, where it appears that the light is moving around the outside of the sign.

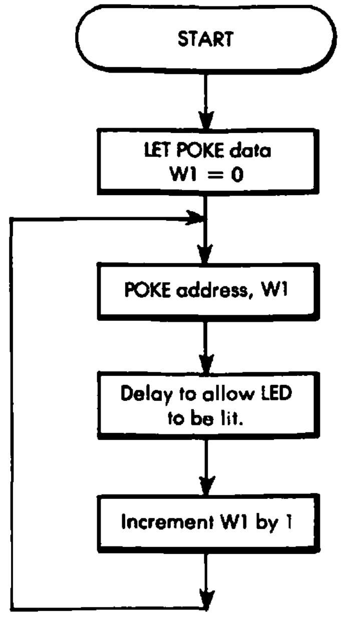
This program will be very similar to the one presented in example 3. That is, we will output a byte to the CMS I/O board and then delay for a while before outputting another.

The program starts by outputting a byte that will light only LED D0. An output number of 1 will accomplish this. Next, a byte is output that will light D1. This is the number 2. The number 4 is output next, as it will light only D2. The idea is to output a number that corresponds to a single bit weight only. The remaining numbers to be output will be 8, 16, 32, 64 and 128. A flowchart for this program is shown in Figure 2.23. Figure 2.24 shows a BASIC program to realize the flowchart.

Load this program and run it. Verify that the LEDs turn on and off in the specified sequence. After the program has run correctly, try these four variations:

a) Make the light travel backwards from the way it is now going.

b) Make the light "bounce." That is, when it gets to D7 make it travel back to D0. When it gets to D0 make it travel back to D7.



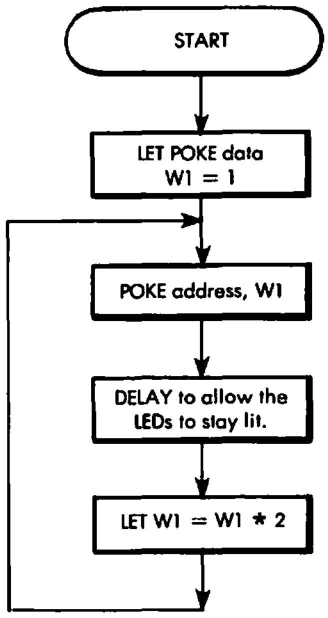
Flowchart for a counting program.

Figure 2.21

```
 20   REM: COUNTING PROGRAM FOR THE APPLE
 30   REM: SET THE FIRST POKE DATA EQUAL TO 0
 40   LET W1 =0
 50   POKE −16176,W1
 60   REM: ASSUME I/O SLOT 5 IS USED
 70   FOR I=1 TO 1000
 80   NEXT I
 90   REM: WE JUST DELAYED FOR A WHILE
100   LET W1 =W1+1
110   IF W1>255 THEN W1 =0
120   GOTO 50
130   END
```

BASIC program to realize the flowchart given in Figure 2.21.

Figure 2.22



Flowchart for a traveling-light program.

Figure 2.23

```
 20  REM: TRAVELING LIGHT PROGRAM
 30  REM: SET FIRST POKE DATA = 1
 40  LET W1=1
 50  POKE -16176,W1
 60  REM: NOW TO DELAY
 70  FOR I=1 TO 1000
 80  NEXT I
 90  REM: NOW TO SHIFT THE DATA BIT LEFT
100  LET W1=W1*2
110  IF W1>128 THEN W1=1
120  GOTO 50
130  END
```

BASIC program to realize the flowchart given in Figure 2.23.

Figure 2.24

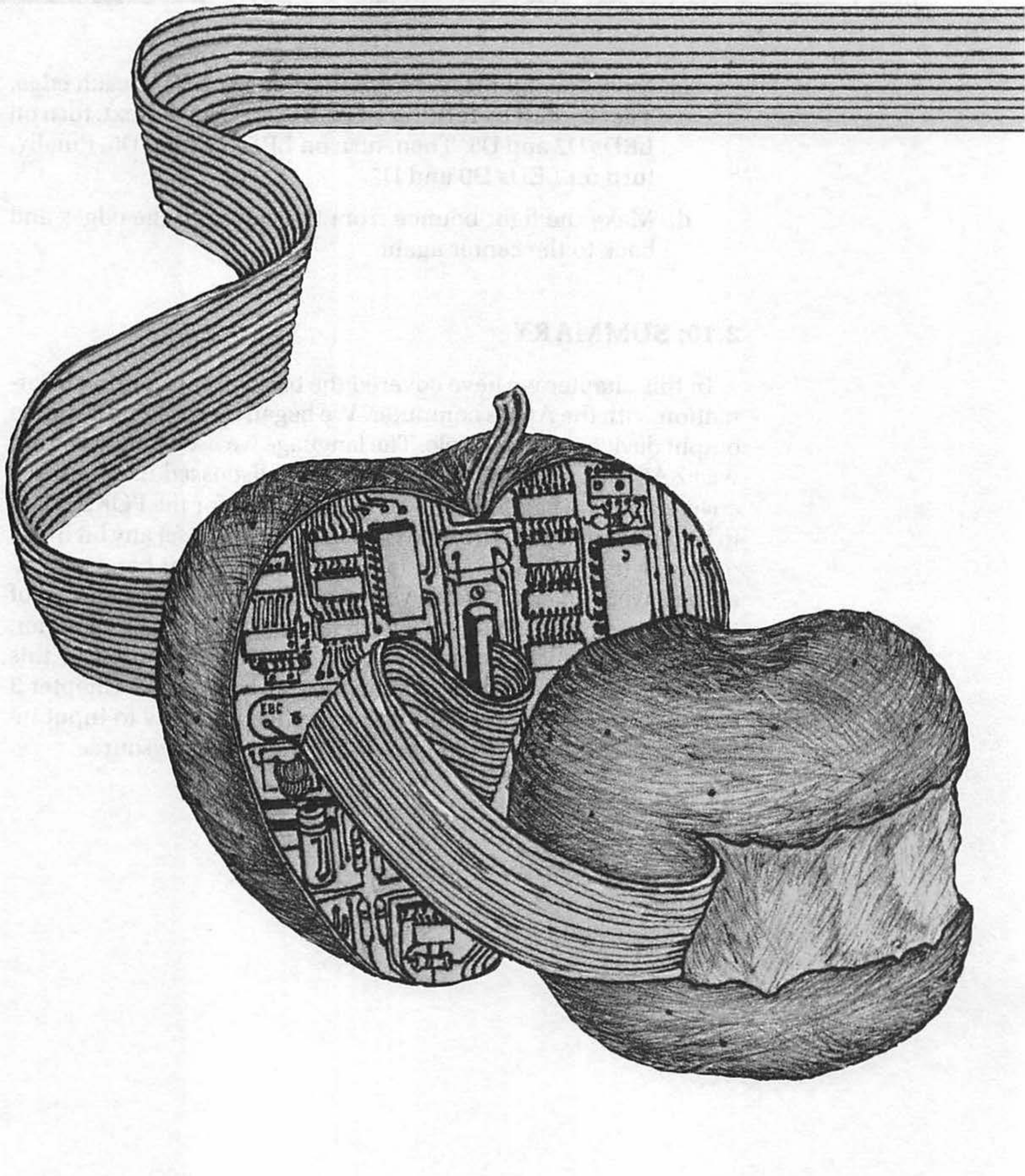
c) Make the light travel from the center LEDs to each edge. That is, start by turning on LEDs D3 and D4. Next, turn on LEDs D2 and D5. Then, turn on LEDs D1 and D6. Finally, turn on LEDs D0 and D7.

d) Make the light bounce from the center to the edges and back to the center again.

2.10: SUMMARY

In this chapter we have covered the basics of outputting information with the Apple computer. We began by plugging a simple output device into the Apple. The language we used for outputting was BASIC. The POKE instruction was discussed in detail. We covered how to calculate the address needed for the POKE. Data to be output was shown and we discussed how to set any bit in the output byte to a logical 1 or a logical 0. Finally, four hands-on examples were given to enable you to verify your understanding of the main points of outputting information with the Apple computer.

When you have mastered the information presented in this chapter, half of the Apple connection has been made. Chapter 3 will discuss the second half of this connection, how to input information to the Apple computer from an external source.



Chapter 3

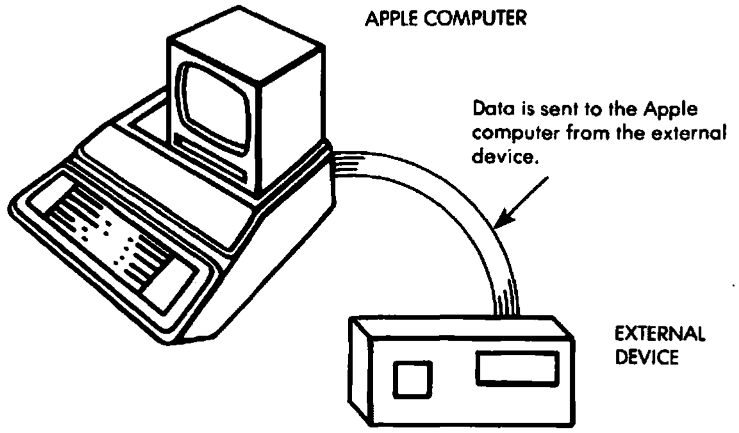
Inputting Data to the Apple Computer

IN THIS CHAPTER we will discuss how to input digital information to the Apple computer from an external device, by means of a BASIC program. After the information is input to the program, we will show ways of interpreting it using software.

3.1: OVERVIEW OF INPUTTING DATA

To begin our discussion of inputting information to the Apple computer, let us review exactly what our goal is, by referring to the block diagram in Figure 3.1. We see in this diagram that an external device will be sending digital information (data) to the Apple computer. In order to electrically input the data, the computer must be able to accept the information and then interpret what was accepted.

As you look at Figure 3.1, certain questions may come to mind. For example, "How does the Apple computer electrically know that the external device is ready to send information?" The answer to that question is covered in the broad topic called *handshaking*. Handshaking refers to the process by which data is exchanged between a computer and an external device in an organized


APPLE COMPUTER
Data is sent to the Apple computer from the external device.
EXTERNAL DEVICE

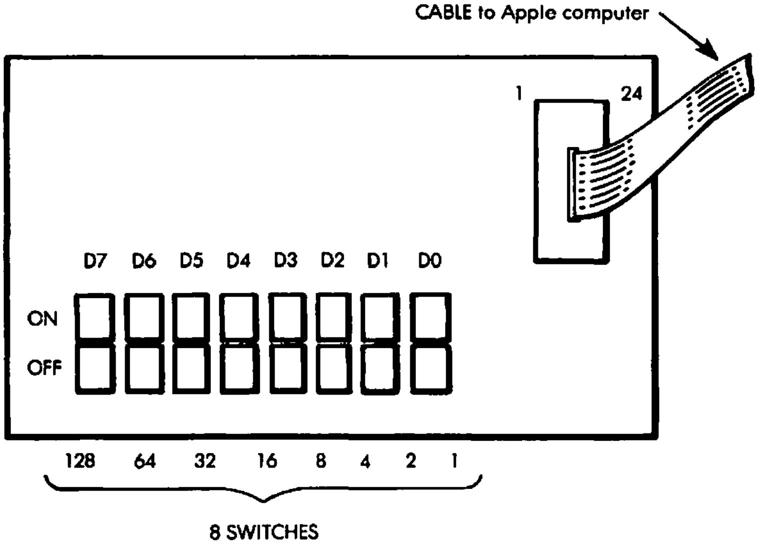
The object of inputting data is to send information from an external device to the Apple computer. The information must be in an electrical form the computer can understand.

Figure 3.1

fashion. In general, when the external device has information or data ready to send to the computer, another line connected to the computer will electrically indicate that the data is ready. At that time, the computer will input, or accept, the information. If the computer is outputting data, another "handshake" line will electrically inform the external device that data is ready to send.

There are other types of handshaking that may be performed, including *interrupts* and *Direct Memory Access (DMA)*. Those types of handshaking systems are beyond the scope of this book, but they are discussed elsewhere in the literature. However, in Chapter 8 we discuss a handshaking system in detail. For our purposes here, we can assume that the data from the external device is always present when the computer requests it.

Using this assumption we may concentrate our efforts on how to transfer the information into the Apple computer, using a BASIC program. In review, our job in this chapter will be to understand how data is input from an external source to a program running in BASIC on the Apple computer. Once the data is input, ways of processing it with software to make logical decisions will be shown.



Pictorial diagram showing the physical switches on the CMS I/O system input section. Each switch is marked with a data line label and the corresponding weight, and is either "ON" or "OFF".

Figure 3.2

3.2: THE CMS INPUT BOARD FOR THE APPLE COMPUTER

The physical connection used for inputting the electrical data to the Apple computer will be made via the CMS I/O system. We became familiar with the I/O board in Chapter 2, when we discussed the mechanics of outputting data from the Apple computer. Besides functioning as an output device, the CMS I/O system is also electrically capable of inputting data to the Apple computer. Figure 3.2 shows a pictorial diagram of the input section of the CMS I/O board hardware, located on board AP2. The I/O system should be installed in the Apple exactly as described in Section 2.1 of Chapter 2. The remainder of this discussion will assume the reader has installed the CMS I/O system correctly.

Notice, in Figure 3.2, that there are eight switches on the CMS I/O board input section. These switches will be set to an OFF or ON position. When the switch is OFF it corresponds to a setting of logical 0 on a particular input line. When the switch is ON it corresponds to a setting of logical 1 on a particular input line.

Each of the switches in Figure 3.2 is assigned a unique label: D0, D1, D2, D3, D4, D5, D6, and D7. The label corresponds to the physical signal line that will be input to the Apple computer.

It is important to know that the switches in the CMS I/O system input section operate in parallel. That is, *any signal line, D0 through D7, can be set to a logical 1 or a logical 0. Each line is set independently of any other signal line.*

3.3: INPUT SOFTWARE

The CMS I/O system will allow us to physically set the input lines to any desired logical state. Now we will turn our attention to the problem of how the programmer can electrically request information from the external device. The programming language is BASIC. The instruction used to accomplish the inputting of information is PEEK. Like the POKE instruction we used in Chapter 2, PEEK is explained fully in your user's manual and elsewhere, but we will briefly summarize its operation here. A PEEK instruction will appear in a BASIC program like this:

```
LET A = PEEK(address)
```

Let us discuss exactly what each part of this instruction does. The variable name "A" could be any BASIC numeric variable. We used "A" here only as an arbitrary example; it could have been replaced by T1, Z5, C(3) or any valid BASIC numeric variable name. The important point is that when information is input using the PEEK instruction, variable A will be set equal to it.

The (address) part of the PEEK instruction will electrically inform the Apple computer which I/O slot (0–7) the data will be input from. The definition of this address is exactly the same as the address definition used for the POKE instruction. This definition was discussed in Chapter 2. Remember from that discussion that the I/O slots in the Apple computer each have a limited set of addresses. In Chapter 2 we presented a list showing the I/O slot number and the corresponding system address used in this book. Refer to Figure 2.7 for that list. Using the list, we can input information from any of the I/O slots simply by choosing the correct address number. For example, suppose we wished to input information from slot 4 in the Apple computer. In a BASIC program

```
 10  PRINT "INPUT THE SLOT NUMBER THE I/O CARD IS INSTALLED INTO ";
 20  INPUT S2
 30  IF S2=0 THEN S3=-16256
 40  IF S2=1 THEN S3=-16240
 50  IF S2=2 THEN S3=-16224
 60  IF S2=3 THEN S3=-16208
 70  IF S2=4 THEN S3=-16192
 80  IF S2=5 THEN S3=-16176
 90  IF S2=6 THEN S3=-16160
100  IF S2=7 THEN S3=-16144
110  LET A1=PEEK(S3)
120  END
```

BASIC program to ask the user which I/O slot is being used in the Apple computer. The I/O slot number (0–7) is input to the BASIC variable S2. Next the program performs a "look-up" of the correct system address based on the I/O slot number.

Figure 3.3

an instruction that would allow this to occur is:

```
LET A1 = PEEK(-16192)
```

After this instruction is executed, the variable A1 will be equal to the information that was input from I/O slot 4.

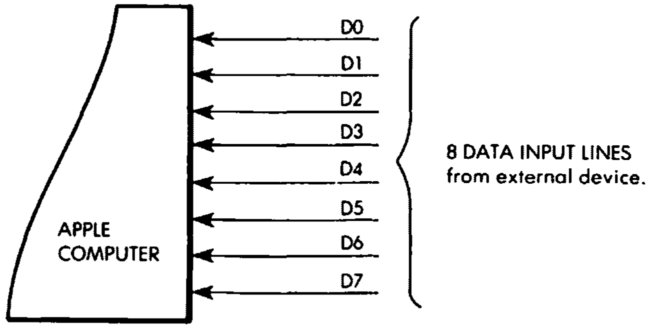
Let us take another example. Suppose the I/O slot address is a variable itself. That is, the BASIC program will prompt the user and ask for the I/O slot number. The I/O slot address will be stored in a BASIC variable. For this example we will assume that the I/O slot address was stored in the variable S3. The form of the PEEK instruction would be:

```
LET A1 = PEEK(S3)
```

This will let A1 be equal to the information read from the I/O slot address S3. Figure 3.3 shows a BASIC program that will operate in the way described for this example.

3.4: INTERPRETING THE INPUT INFORMATION

Up to this point in our discussion of inputting data, we have shown the software required to get the input information into a



Eight separate signal lines are connected to an Apple computer I/O slot in order to input external information.

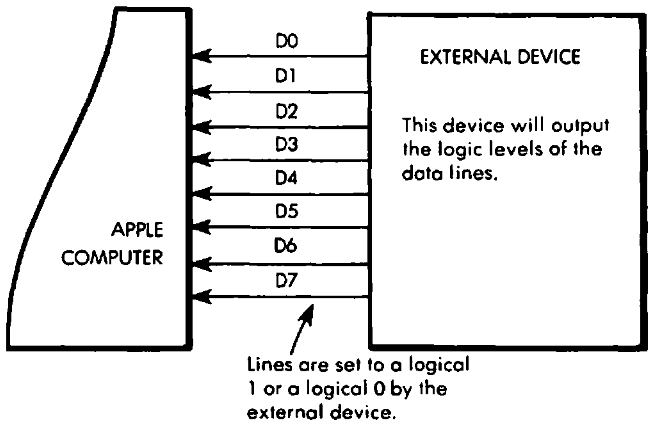
Figure 3.4

BASIC program. The information will reside in a valid BASIC variable. This section will focus on ways to interpret the information that was input to the BASIC variable. During this discussion we will be building on the information that was presented in Section 2.4 of Chapter 2.

When the Apple computer inputs data from an I/O slot, it is electrically inputting the logical voltage levels of eight separate signal lines. These signal lines are labeled D0, D1, D2, D3, D4, D5, D6, and D7. (See Figure 3.4.)

Each physical signal line is assigned a numerical weight by the computer. Since each signal line corresponds to a bit position, these weights are the same as those discussed in Chapter 2, and listed in Figure 2.11. (Again, it should be remembered that these weights are not arbitrarily assigned, but correspond to powers of two, that is, to binary numbers. A few minutes spent familiarizing yourself with the binary numbering system will be of great value in understanding how your computer works.) When an input signal line is a logical 1, its corresponding weight is summed. When an input signal line is a logical 0, its corresponding weight is not summed. The resulting sum is the value that will be stored into the BASIC variable used in the PEEK instruction.

For example, suppose the external hardware was connected to I/O slot 5. This would require an address of −16176 to be used in the PEEK instruction. It is further assumed that the external


APPLE COMPUTER
D0
D1
D2
D3
D4
D5
D6
D7
EXTERNAL DEVICE
This device will output the logic levels of the data lines.
Lines are set to a logical 1 or a logical 0 by the external device.

The external input device controls the logical voltage levels of the input lines to the Apple computer.

Figure 3.5

hardware is sending data that has lines D0, D4, D5 and D7 set to a logical 1. The remaining lines, D1, D2, D3 and D6, are set to a logical 0. External input lines are set under control of the device sending the data to the computer. (See Figure 3.5.)

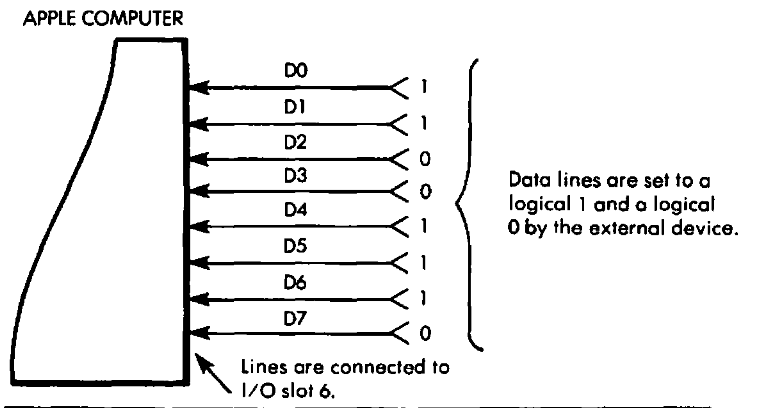
When we wish to obtain the information from the external device used in this example, the PEEK instruction will appear as:

```
LET A1 = PEEK(-16176)
```

After this instruction is executed, the BASIC variable A1 will be equal to the summation of the weights of all data input lines, D0–D7, to slot 5. The summation will include the weights of all the input lines that were a logical 1 during the execution of the PEEK instruction.

In our example the weights that will be summed are D7 = 128, D5 = 32, D4 = 16 and D0 = 1. The resulting summation would yield 128 + 32 + 16 + 1 = 177. Variable A1 would be equal to the value 177 after the execution of the PEEK instruction.

We can expect the variable used in the PEEK instruction to be greater than or equal to 0 and less than or equal to 255. A value of 0 is returned when no input lines are set to a logical 1 at the I/O slot, and a value of 255 is returned when all of the input lines are set to a logical 1 at the I/O slot.


APPLE COMPUTER
D0
D1
D2
D3
D4
D5
D6
D7
1
1
0
0
1
1
1
0
Data lines are set to a logical 1 and a logical 0 by the external device.
Lines are connected to I/O slot 6.

Data input lines D0, D1, D4, D5, and D6 are set to a logical 1 by the external device.
Figure 3.6

To further illustrate this point, let us consider another example. In this example it is assumed that the input device is setting data lines D6, D5, D4 and D1 to a logical 1. All other data input lines are set to a logical 0. (See Figure 3.6.) The input device is physically connected to I/O slot 6. Input information is read using the PEEK instruction. In BASIC the instruction for this example will appear as:

```
LET R = PEEK(-16160)
```

The number −16160 is derived from the address of I/O slot 6 in the Apple computer.

What will be the value of R after the PEEK instruction is executed? Recall that R will be equal to the sum of the weights of all input lines set to a logical 1 during the execution of the PEEK instruction. These weights will be as follows: D6 = 64, D5 = 32, D4 = 16, and D1 = 2. The resulting sum would be 64 + 32 + 16 + 2 = 114. Variable R would equal 114 after execution of the PEEK instruction. If we were to print the value of R at this time, the number 114 would appear. By using the PEEK instruction, we now have a means of inputting data from an external device. Further, the data is stored in a BASIC variable. By using software, we can then operate on the variable in exactly the same way as any other BASIC variable.

3.5: CALCULATING THE BITS FROM THE INPUT VARIABLE

In the preceding section we discussed how the user can calculate the value of the input variable used in the PEEK instruction. That calculation was based on the assumption that we knew the logical level of the data input lines at the selected I/O slot. However, in many applications of computer control, we do not know which data input lines were a logical 1 and which data input lines were a logical 0 at the time of the PEEK instruction, and we need to know. The reason we need to know the logical state of the input lines is that each line input by the external device may mean something different. For example, D0 may logically inform the user that the temperature is too high, D1 may be used to indicate that some lights in a home were left on, and so on. The main point is that we need to examine the logical conditions (1 or 0) of *each* data line input during the PEEK instruction. This can be accomplished using software.

There are several ways of doing this in a BASIC program. We will present only one method here. This technique is designed to help you understand exactly what is needed and what is occurring, rather than to operate as efficiently as possible. However, as a BASIC program, it *does* work. As you become more familiar with this type of processing, new and more efficient ways of performing the transformation and testing will become apparent.

Here is one way to do it. We can start by applying our knowledge of how the PEEK variable was originally formed to the problem of "deciphering" it. Recall that the variable was formed by summing the weights of the data input lines that were a logical 1 during execution of the PEEK instruction. What we will do is discover, by a process of subtraction, which individual weights were used to obtain the sum. Once these weights are known, the corresponding input lines that were set to a logical 1 are also known. All other input lines must have been a logical 0 during the execution of the PEEK instruction.

For example, suppose we executed this PEEK instruction:

```
LET A = PEEK(-16240)
```

After this instruction was executed, the variable A would be a value between 0 and 255, inclusive. The value would depend on which of the eight data input lines, D0–D7, were set to a logical 1. For purposes of illustration, we can assume that the variable A was equal to 183 after the PEEK instruction. At the outset, we can see that at least one of the data input lines was set to a logical 1, because the value of A is not 0. We also know that at least one of the data input lines is a logical 0, because the value of A is not 255. But we do not know which input lines were set to a logical 1 and which input lines were set to a logical 0 during the PEEK instruction. That is precisely the problem we are going to solve.

Our job now is to determine which of the data input lines was a logical 1 during the input instruction. This can be done in the following way. Subtract from the variable A the weight of each input line, starting with the weight of D7. If the result of the subtraction is less than 0, we know the weight subtracted was not used to obtain the sum. Let's go through some examples to show exactly what is meant.

Suppose a variable returned from the PEEK instruction was equal to 125. Using our plan, the weight of D7 is subtracted from 125. This would yield a subtraction of 125 − 128, which is less than 0. Therefore, we know that the weight of D7 was not used to obtain the original sum, and that D7 is a logical 0.

We then proceed with the next weight in line. This is the weight of D6. The weight of D6 is subtracted from 125. The result would be 125 − 64 = 61. Note that the result of this subtraction is not less than 0. Therefore, the weight of D6 was used in the original summation. D6 is a logical 1.

When we find that a weight was used in the summation, its value is subtracted from the starting number and the resulting value is tested against the next weight in the line. In this case the next weight in the line after D6 is D5. The weight of D5 is subtracted from the new value, 61, not the original value, 125. This will give 61 − 32 = 29. This value is not less than 0. Therefore, the weight of D5 was used to obtain the original sum.

At this time we know that the weights of D6 and D5 were used in obtaining the original sum of 125. Proceeding further in the process, we test the next weight, D4, against the new value, 29. The value 29 is equal to the original number, 125, minus the known weights, D6 and D5.

Subtracting 16 from 29, we get a result of 13. This result is not less than 0. Therefore, the weight of D4 was used in obtaining the original sum. We continue the process by subtracting the next weight from the new value, 13.

The next weight in line is D3. The subtraction 13 − 8 = 5 yields a number greater than 0. From this we know that the weight of D3 was used in obtaining the original sum. The next weight, D2, is tested against the new value, 5.

The resulting subtraction, 5 − 4 = 1, gives a number that is not less than 0. Again, we know that the weight of D2 was used to obtain the original sum. At this point we know that the weights of D6, D5, D4, D3 and D2 were used to obtain the original sum.

We next test against the weight of D1. This subtraction would yield 1 − 2 = −1. This result is less than 0. Because of this, we know that the weight of D1 was not used to obtain the original sum. Finally, we proceed to test against the weight of D0.

Since the subtraction of the weight of D1 yielded a negative result, the testing value, 1, is not changed. Subtracting the weight of D0 would give 1 − 1 = 0. The result is not less than 0, so we know that D0 was used to obtain the original sum of 125. At the conclusion of this process, we know that the weights of the data lines D6, D5, D4, D3, D2 and D0 were used to obtain the original sum. Therefore, these input data lines were a logical 1 during the execution of the PEEK instruction. Further, we know that the data input lines D7 and D1 were a logical 0 during the execution of the PEEK instruction.

The procedure described above is, essentially, how the weights of the data lines are tested. If at any point in the testing the result is 0, then we can stop and test no further. All the remaining data lines must be a logical 0. To further illustrate this procedure let us consider another example, detailing the steps in outline form. We will assume that the variable returned from the PEEK instruction this time was 183. We need to know which data lines were a logical 1 and which data lines were a logical 0. The procedure is outlined below:

1. Subtract the weight of D7 from the original variable, 183. This gives:

   183 − 128 = 55

   The result is greater than 0. Therefore the bit weight of D7 was used to obtain the original sum. D7 was a logical 1 during the PEEK instruction. We set the variable A2 equal to 55.

2. We now subtract the bit weight of D6 from the new value of variable A2.

   $55 - 64 = -9$

   The result is less than 0. This tells us that the bit weight of D6 was not used to obtain the original sum of 183. D6 was a logical 0 during the PEEK instruction.

3. We next subtract the weight of D5 from the variable A2. Note that A2 was not changed in step 2, because the result was less than 0 during the subtraction.

   $55 - 32 = 23$

   Since the result of the subtraction is not less than 0, we know that this weight was used to obtain the original sum. Therefore, D5 was a logical 1 during the PEEK instruction. The variable A2 is now set to 23 because the result of the subtraction was not negative. Notice that as a weight is used to obtain the sum, it is subtracted from the variable A2.

4. Next we subtract the bit weight of D4 from the variable A2.

   $23 - 16 = 7$

   The result is not less than 0. Therefore, the bit weight of D4 was used to obtain the original sum. Data input line D4 was a logical 1 during the PEEK instruction. The variable A2 is now equal to 7.

   Let us stop at this point to examine what information we have concerning the input data lines. From the preceding steps 1–4 we know that bit D7 = 1, D6 = 0, D5 = 1 and D4 = 1. The checking is resumed starting with the weight of D3.

5. We subtract the bit weight of D3 from the new variable A2.

   $7 - 8 = -1$

   The result is a number less than 0. Therefore, we know that the weight of D3 was not used in obtaining the original sum of 183. This means D3 was a logical 0 during the PEEK instruction.

6. Testing the next bit weight, D2, we obtain the result:

   $7 - 4 = 3$

   The result is a number that is not less than 0. Therefore, input line D2 was a logical 1 during the PEEK instruction. The variable A2 becomes 3.

7. Now test bit D1. The result of this subtraction is:

   $3 - 2 = 1$

   We see that the D1 input line was a logical 1 during the PEEK instruction.

8. The final bit to test is D0. The result of this test is:

   $1 - 1 = 0$

   Since the result of the subtraction was not less than 0, the weight of D0 was used to obtain the original sum.

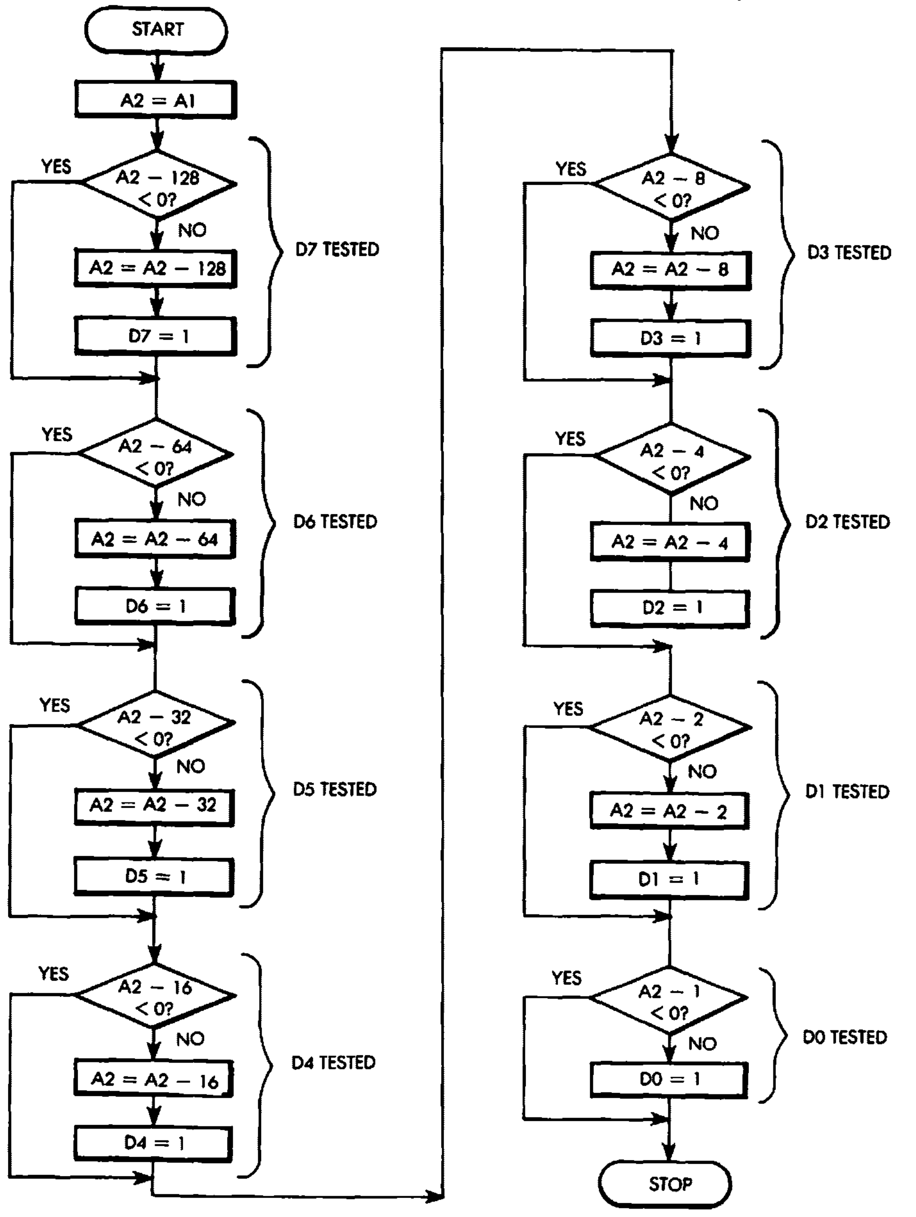
The results of the bit testing can be summarized thus:

D7	D6	D5	D4	D3	D2	D1	D0	*bit labels*
1	0	1	1	0	1	1	1	*logical values*

These results are easy to check as we can simply sum the weights of all the bits that are a logical 1. The result of this summation should be the number we started with, 183.

$$\begin{array}{rl} 128 &= D7 \\ +\ 32 &= D5 \\ +\ 16 &= D4 \\ +\ 4 &= D2 \\ +\ 2 &= D1 \\ +\ 1 &= D0 \\ \hline 183 & \quad \textit{The result checks.} \end{array}$$

When we read through the steps required to make these tests by hand, the procedure seems quite tedious. Fortunately, we have the computer to make these checks for us. Since we know what tasks the computer has to perform, the first step in writing a computer program to perform these checks is to design a task flowchart outlining the steps of the procedure. Such a flowchart is shown in Figure 3.7.



Flowchart for checking the logical value (1 or 0) of the external data input lines to the Apple computer. The data was input by using a PEEK instruction.

Figure 3.7

A BASIC program to realize this flowchart on the Apple computer is given in Figure 3.8. Another example of a BASIC program that will implement the flowchart of Figure 3.7 is given in Figure 3.9. The difference between the two programs is in the use of the FOR/NEXT loops employed to do the testing of the individual weights. In Figure 3.8 each weight is tested in a sequential fashion. Figure 3.9, on the other hand, tests each weight by reducing the variable A3 by half at each pass through the loop. In Figure 3.8 we simply wrote a statement that set up the new test value.

```
 10  PRINT "INPUT THE ORIGINAL NUMBER BETWEEN 0-255 ";
 20  INPUT A1
 30  LET A2=A1
 35  REM: SET ALL D VALUES EQUAL TO ZERO (LINES 40-110)
 40  LET D0=0
 50  LET D1 =0
 60  LET D2=0
 70  LET D3=0
 80  LET D4=0
 90  LET D5=0
100  LET D6=0
110  LET D7=0
195  REM: D7 TESTED (LINES 200-220)
200  IF A2-128<0 THEN 230
210  LET A2=A2-128
220  LET D7=1
225  REM: D6 TESTED (LINES 230-250)
230  IF A2-64<0 THEN 260
240  LET A2=A2-64
250  LET D6=1
255  REM: D5 TESTED (LINES 260-280)
260  IF A2-32<0 THEN 290
270  LET A2=A2-32
280  LET D5=1
285  REM: D4 TESTED (LINES 290-310)
```

Figure 3.8

```
290  IF A2-16<0 THEN 320
300  LET A2=A2-16
310  LET D4=1
315  REM: D3 TESTED (LINES 320-340)
320  IF A2-8<0 THEN 350
330  LET A2=A2-8
340  LET D3=1
345  REM: D2 TESTED (LINES 350-370)
350  IF A2-4<0 THEN 380
360  LET A2=A2-4
370  LET D2=1
375  REM: D1 TESTED (LINES 380-400)
380  IF A2-2<0 THEN 410
390  LET A2=A2-2
400  LET D1=1
405  REM: D0 TESTED (LINES 410-420)
410  IF A2-1<0 THEN 430
420  LET D0=1
430  PRINT "ORIGINAL NUMBER WAS ";A1
440  PRINT
450  PRINT "BINARY VALUE IS ";D7;D6;D5;D4;D3;D2;D1;D0
460  END
```

BASIC program to realize the flowchart of Figure 3.7.

Figure 3.8 (cont.)

```
10  DIM D(8)
20  PRINT "INPUT THE ORIGINAL NUMBER BETWEEN 0-255";
30  INPUT A1
40  FOR I=0 TO 7
50  LET D(I)=0
60  NEXT I
70  LET A2=A1
80  LET A3=128
90  FOR I= 7 TO 0 STEP -1
```

Figure 3.9

```
100      IF A2-A3<0 THEN 130
110      LET A2=A2-A3
120      D(I)=1
130      A3=A3/2
140   NEXT I
150   PRINT "ORIGINAL NUMBER WAS ";A1
160   PRINT
170   PRINT "THE BINARY VALUE IS EQUAL TO ";
180   FOR I= 7 TO 0 STEP -1
190      PRINT D(I);
200   NEXT I
210   PRINT
220   END
```

A shorter BASIC program to realize the flowchart of Figure 3.7. In lines 90–130 the variable A3 is the weight of each data line. A3 will start at 128, which is the weight of D7. This variable is then reduced by half each pass through the FOR/NEXT loop.

Figure 3.9 (cont.)

Using the testing technique described above, we will be able to tell exactly which data input lines were a logical 1 and which data input lines were logical 0 during the execution of a PEEK instruction. We are now ready to get some practice at inputting data to the Apple.

3.6: HANDS-ON EXAMPLE 1: CALCULATING THE WEIGHT OF THE INPUT WORD

In this first hands-on example of inputting data to the Apple computer from the CMS I/O board, we will calculate the value of the input weights by hand. The input weight will be calculated according to the method described in this chapter. After the weight is calculated, you should set the switches on the CMS I/O board to the correct value and run the program.

The program will print out the weight calculated by the computer. You may then verify whether the manual calculation was correct. Let us go over an example of exactly what is meant. First, set the input switches on the CMS I/O board so that D0 and D4 are a logical 1. All other switches are set to a logical 0. Remember that when the switch is in the OFF position it is a logical 0, and when it is in the ON position it is a logical 1.

Now calculate the input weight the Apple computer will see when this data is read. In this example the computer will electrically input D4 and D0 as a logical 1. This will correspond to a weight of D4=16 + D0=1 = 16 + 1 = 17.

After the computer has read this data it will print out the number 17. At this point we can check our calculated results against the actual computer results. In this way you can quickly verify that you understand how the input lines are weighted.

Try the program shown in Figure 3.10, using the different switch settings, a–f. The switches indicated will be a logical 1, and all other switches will be set to a logical 0. The correct weight for each switch setting is given after setting (f).

a. D1 and D7 are set to 1
b. D0 and D5 are set to 1
c. D4, D6, and D7 are set to 1
d. D1, D2, D3, D4, D6, D7 are set to 1
e. All switches are set to a logical 1
f. All switches are set to a logical 0

Answers: a = 130, b = 33, c = 208, d = 222, e = 255, f = 0

3.7: EXAMPLE 2: READ A BYTE AND DETERMINE WHICH BITS WERE A LOGICAL 1

In this example we will read an input word from the CMS I/O board and let the computer print out which data lines were equal to a logical 1 and which data lines were equal to a logical 0. This is exactly the same type of operation we discussed in Section 3.5. A general program to allow the Apple computer to perform this function is shown in Figure 3.11. Run this program and verify that the computer will print out the correct data lines that you have set on the CMS I/O board switches. Use the same switch settings given in Section 3.6.

```
 10  PRINT "INPUT THE SLOT NUMBER THE I/O CARD IS INSTALLED INTO ";
 20  INPUT S2
 30  IF S2=0 THEN S3= -16256
 40  IF S2=1 THEN S3= -16240
 50  IF S2=2 THEN S3= -16224
 60  IF S2=3 THEN S3= -16208
 70  IF S2=4 THEN S3= -16192
 80  IF S2=5 THEN S3= -16176
 90  IF S2=6 THEN S3= -16160
100  IF S2=7 THEN S3= -16144
110  LET A1 =PEEK(S3)
120  PRINT "THE DATA READ FROM I/O SLOT # ";S2;" IS ";S3
121  END
```

Sample program to read the external input data from the CMS I/O system. This program will print out the total weight and the slot number the I/O system is connected to.

Figure 3.10

```
 10  PRINT "INPUT THE SLOT NUMBER THE I/O CARD IS INSTALLED INTO ";
 20  INPUT S2
 30  IF S2=0 THEN S3= -16256
 40  IF S2=1 THEN S3= -16240
 50  IF S2=2 THEN S3= -16224
 60  IF S2=3 THEN S3= -16208
 70  IF S2=4 THEN S3= -16192
 80  IF S2=5 THEN S3= -16176
 90  IF S2=6 THEN S3= -16160
100  IF S2=7 THEN S3= -16144
110  LET A1 =PEEK(S3)
122  LET A2=S3
123  LET D0=0
125  LET D1 =0
127  LET D2=0
129  LET D3=0
131  LET D4=0
133  LET D5=0
135  LET D6=0
137  LET D7=0
```

Figure 3.11

```
200  IF A2-128<0 THEN 230
210  LET A2=A2-128
220  LET D7=1
230  IF A2-64<0 THEN 260
240  LET A2=A2-64
250  LET D6=1
260  IF A2-32<0 THEN 290
270  LET A2=A2-32
280  LET D5=1
290  IF A2-16<0 THEN 320
300  LET A2=A2-16
310  LET D4=1
320  IF A2-8<0 THEN 350
330  LET A2=A2-8
340  LET D3=1
350  IF A2-4<0 THEN 380
360  LET A2=A2-4
370  LET D2=1
380  IF A2-2<0 THEN 410
390  LET A2=A2-2
400  LET D1=1
410  IF A2-1 <0 THEN 430
420  LET D0=1
430  PRINT
440  PRINT "THE BINARY VALUES OF THE INPUT WERE"
450  PRINT
460  PRINT "D7 ="; D7
470  PRINT "D6 ="; D6
480  PRINT "D5 ="; D5
490  PRINT "D4 ="; D4
500  PRINT "D3 ="; D3
510  PRINT "D2 ="; D2
520  PRINT "D1 ="; D1
530  PRINT "D0 ="; D0
540  END
```

A BASIC program to read the external input data and print the logical value (1 or 0) of each input data line.

Figure 3.11 (cont.)

After you have run the program and verified that it works, make certain you understand each part of it. Now write a similar program using fewer BASIC statements. Use this new program to print out only the input lines that are a logical 1. For example, if D0 and D4 are set to a logical 1 position on the CMS I/O board, have the computer print out D0, D4.

Now reverse the sense of the problem. Have the computer print out the input lines that are set to a logical 0 instead of a logical 1.

3.8: EXAMPLE 3: READ A WORD AND PERFORM AN ACTION

In this example we will input a byte from the CMS I/O board to the Apple computer. If the byte we are inputting is equal to a certain value, we will perform a particular action; we will print a message. If the byte is not equal to that value, we will perform a different action; we will print a different message.

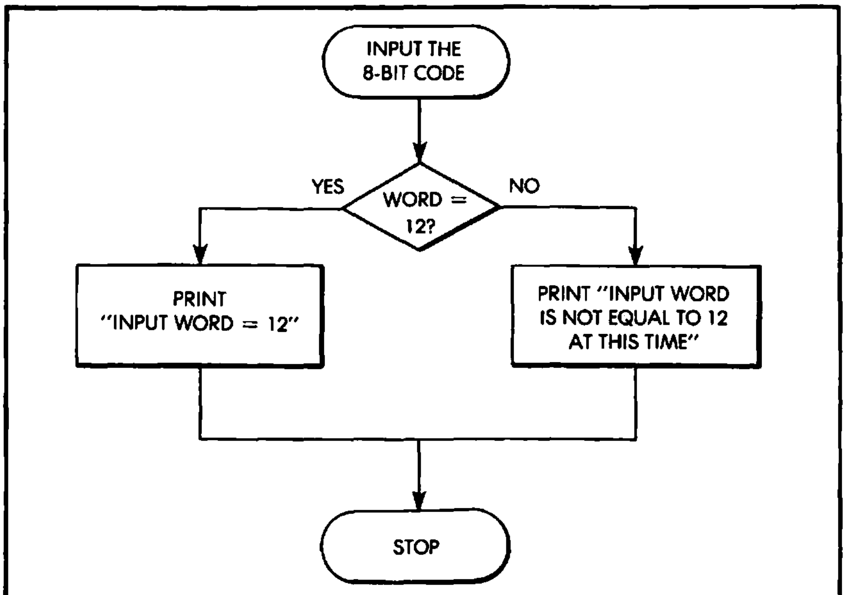
This type of example is very similar in principle to monitoring an input device with the Apple (or any other) computer. When the input condition is true, the computer will automatically perform some corrective action or simply perform some action based on this input. Of course, in a real application, the input would come from some piece of external hardware the computer was controlling or monitoring.

In our example the inputs will be generated by the user from the CMS I/O board. When the word input is equal to 12, the computer will print the message, "THE INPUT WORD IS EQUAL TO 12." If the input word is not equal to 12, then the computer will print the message, "THE INPUT WORD IS NOT EQUAL TO 12 AT THIS TIME."

A flowchart for this problem is shown in Figure 3.12, and a BASIC program to realize the flowchart is given in Figure 3.13.

Try this program out with your computer. Set the switches on the CMS I/O board to a number other than 12. Insure that the computer will print out the correct message. Now set the input word equal to 12 (that is, set D3 and D2 to a logical 1), and check to see if the computer will print out the correct message.

When you are able to get the program running, try the following



A task flowchart for the solution to the problem given in Section 3.8.

Figure 3.12

variations:

a. Have the computer check for an even number of ones in the input word. If the number of ones is even, the computer will print, "THERE ARE AN EVEN NUMBER OF ONES ." If the number of 1s in the input word is odd, the computer will print, "THE NUMBER OF ONES IS ODD."

b. Input a number, and have the computer subtract 25 from it; then print out the results. The output should be both your original number and the new number. Remember that it is possible to get a number that is less than 0 in this program. If the result is less than 0 then print out the message, "THE RESULT WAS LESS THAN 0."

3.9: EXAMPLE 4: A COMBINATION LOCK

In this example we will make the Apple computer into a combination lock. The lock will have three unique numbers that must

```
10  PRINT "INPUT THE SLOT NUMBER THE I/O CARD IS INSTALLED INTO ";
20  INPUT S2
30  IF S2=0 THEN S3=-16256
35  REM: SET ALL D VALUES EQUAL TO ZERO (LINES 40-60)
40  IF S2=1 THEN S3=-16240
50  IF S2=2 THEN S3=-16224
60  IF S2=3 THEN S3=-16208
70  IF S2=4 THEN S3=-16192
80  IF S2=5 THEN S3=-16176
85  REM: LOOP TO TEST ALL D VALUES (LINES 90-140)
90  IF S2=6 THEN S3=-16160
100 IF S2=7 THEN S3=-16144
110 LET A1=PEEK(S3)
120 IF S3=12 THEN 200
130 PRINT" THE INPUT WORD IS NOT EQUAL TO 12 AT THIS TIME"
140 STOP
200 PRINT "THE INPUT WORD IS EQUAL TO 12"
210 END
```

A BASIC program that will PEEK the data from the I/O board, and print one of two responses, depending on the value of the input word.

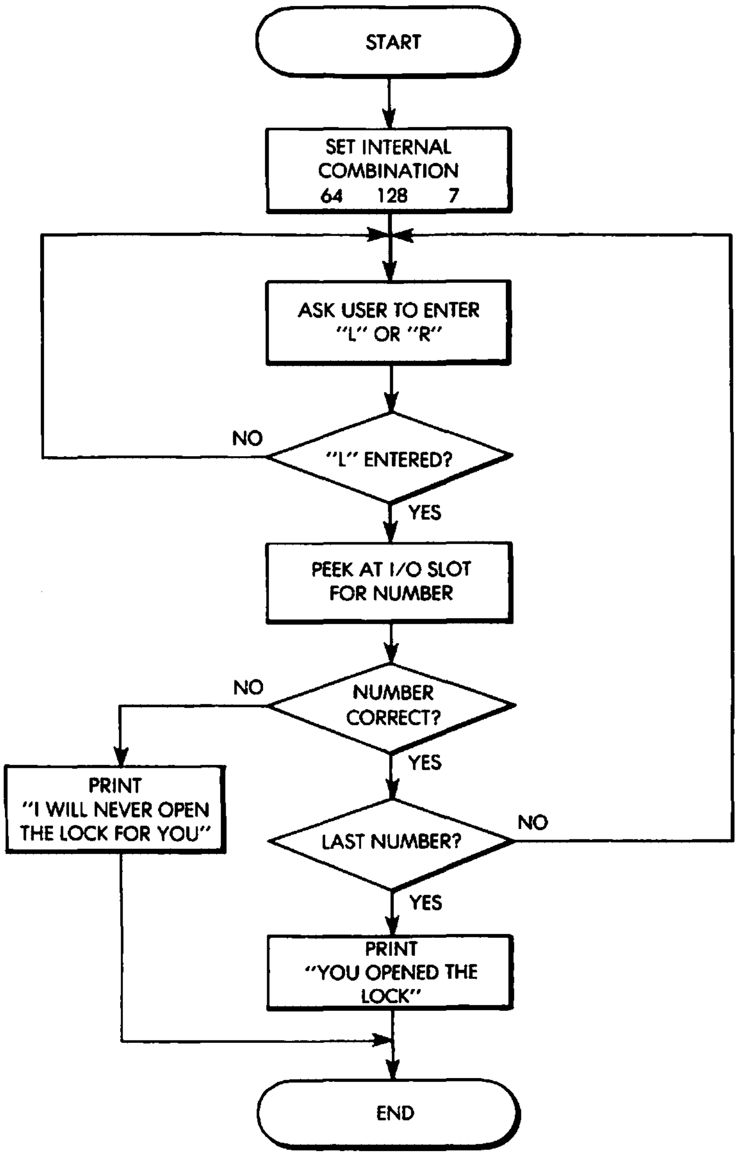
Figure 3.13

be entered in sequence. If any of the numbers (or their sequence) is not correct, the lock will not open.

The three numbers will be 64, 128, and 7. Here is how the lock will work. A BASIC computer program will ask you to enter the first number by setting the switches corresponding to that weight on the CMS I/O board. When the number is set you will enter "L" on the computer. This signals the computer that you have entered the number and are ready for the computer to read the number.

If the computer detects a wrong number, then the message, "I WILL NEVER OPEN FOR YOU" will be printed. If the number is correct, the computer will print the message, "SO FAR SO GOOD." When you enter the last number correctly, the computer will print, "YOU OPENED THE LOCK." If you enter an "R" you will reset the combination lock. This will allow you to start over.

A flowchart of this problem is shown in Figure 3.14. Figure 3.15


START
SET INTERNAL COMBINATION 64 128 7
ASK USER TO ENTER "L" OR "R"
NO
"L" ENTERED?
YES
PEEK AT I/O SLOT FOR NUMBER
NO
NUMBER CORRECT?
YES
PRINT "I WILL NEVER OPEN THE LOCK FOR YOU"
LAST NUMBER?
NO
YES
PRINT "YOU OPENED THE LOCK"
END

A task flowchart showing the general steps for controlling an imaginary combination lock.

Figure 3.14

```
10   DIM N(3),L(3),A$(5)
20   REM: THESE ARE THE COMBINATION NUMBERS
30   N(1)=64
40   N(2)=128
50   N(3)=7
60   REM: THESE ARE THE INPUT COMBINATION NUMBERS
70   L(1)=0
80   L(2)=0
90   L(3)=0
100  REM: START OF COMBINATION INPUTTING
110  I=1
120  PRINT "PUT SETTING # ";I;" ON CMS INPUT SWITCHES"
130  PRINT "INPUT 'L' WHEN READY, OR 'R' TO RESET"
140  INPUT A$
150  IF A$="L" THEN 180
160  IF A$="R" THEN 70
170  GOTO 120
180  L(I)=PEEK(-16208)
190  IF L(I)=N(I) THEN 220
200  PRINT "I WILL NEVER OPEN THE LOCK FOR YOU !!!"
210  GOTO 240
220  IF I=3 THEN 260
230  PRINT "SO FAR, SO GOOD"
240  I=I+1
250  GOTO 120
255  REM: THE LOCK WILL NOW OPEN
260  PRINT "YOU OPENED THE LOCK"
270  END
```

A BASIC program that will realize the computer control for the combination lock.

Figure 3.15

shows one BASIC program that will realize this flowchart. Try this program out with your Apple computer. Once you have this program running, try putting in the following variations:

a. Change the combination of the lock to 2, 4, 8.
b. Realize the program using fewer BASIC statements than shown in the example.

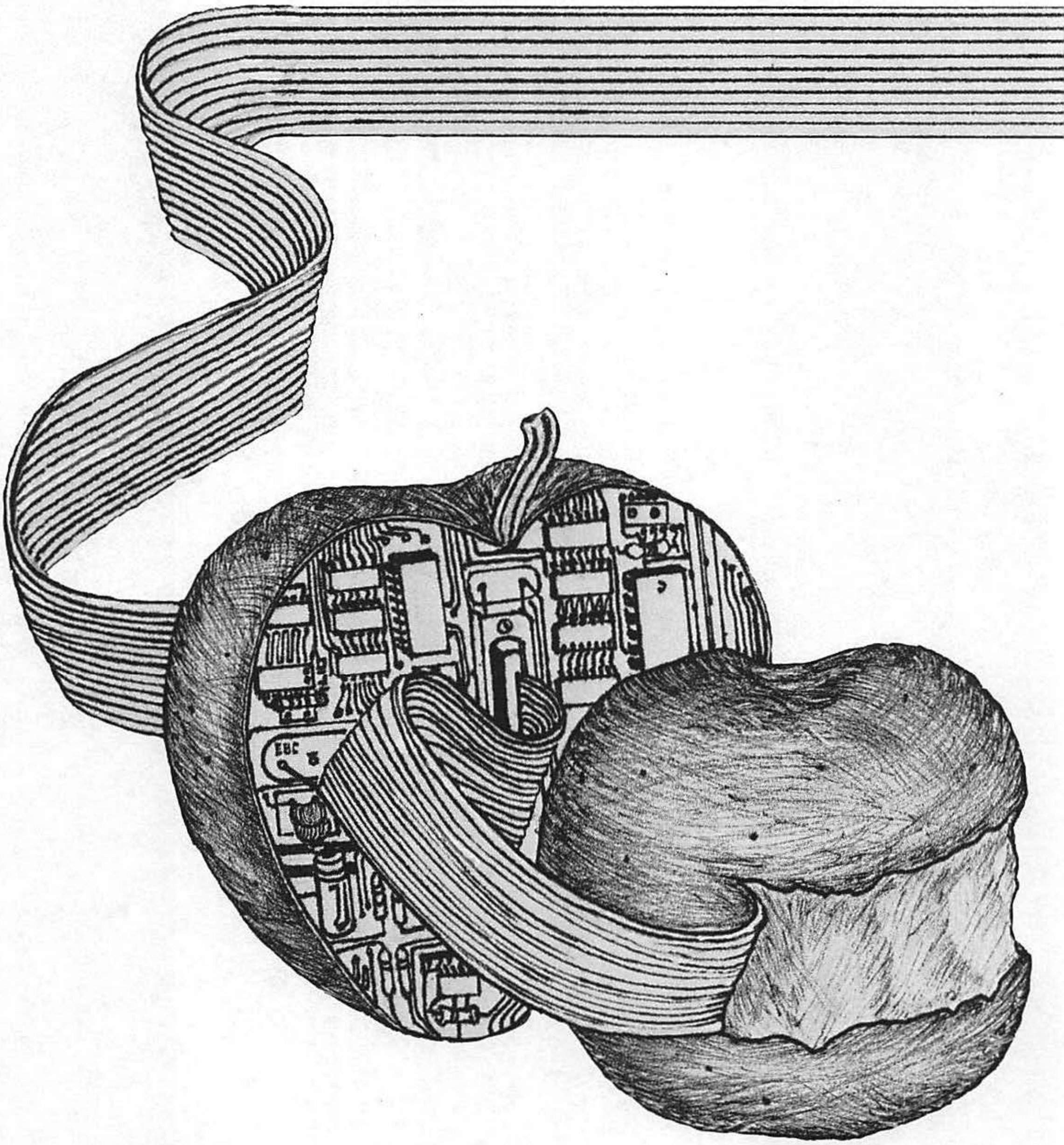
3.10: SUMMARY

In this chapter we have discussed in detail how to input information into a BASIC program from an external device. This device will be monitored by the Apple computer.

Once the information was input into the BASIC program, we discussed different methods of interpreting what the input number meant. A technique was described that will enable you to determine which input lines were a logical 1 and which input lines were a logical 0.

At the end of this chapter we presented some hands-on examples. These examples were designed to allow you to practice inputting information to a BASIC program, and to test your understanding of the methods outlined.

If you understand the solution to these examples, then you understand the principles of inputting data. In Chapters 4 and 5 we will present the hardware for this task and describe how the software will interact with it. Now you are ready for the next step along the path toward making *the Apple Connection*. That step is to actually connect the hardware to allow the computer to input and output data to and from an external device. How do we do that? To answer that question, *read on*.


EBC

Chapter 4

Input and Output Hardware for the Apple

IN THIS CHAPTER we will discuss the hardware, internal and external, necessary to input and output data to and from the Apple computer. It should be noted that this discussion is meant for the beginner in computer interfacing and/or computer control. These discussions are centered around the Apple computer's input and output hardware architecture.

The designs shown in this chapter do work, but they are not shown as the most elegant or efficient, "least-parts-count," circuits. Instead, their main function is to instruct. They are calculated to enable a person who has little or no experience in digital hardware to understand the major concepts involved in the input and output of electrical information between the Apple and an external device. In the chapters that follow we will make use of the information presented here.

The author is fully aware that this may be your first exposure to any digital hardware, and that you may feel apprehensive about the difficulties of the subject. Furthermore, you may not *want* to learn about the hardware of the computer in any great detail. But if you can grasp the essential facts and concepts presented in this chapter, understanding how other peripheral hardware is interfaced to the Apple computer will be much easier. That will be true even if you never actually design a computer interface for the

Apple computer. Using the information given in this chapter, you will better understand other manufacturers' interfaces to the Apple computer. As we proceed through the discussion, relationships will be pointed out between the hardware described here and the software given in Chapters 2 and 3.

4.1: BEGINNING OUTPUT ELECTRONICS FOR THE APPLE

There are four main digital electronic sections that are of concern when discussing the output hardware for the Apple computer. These are:

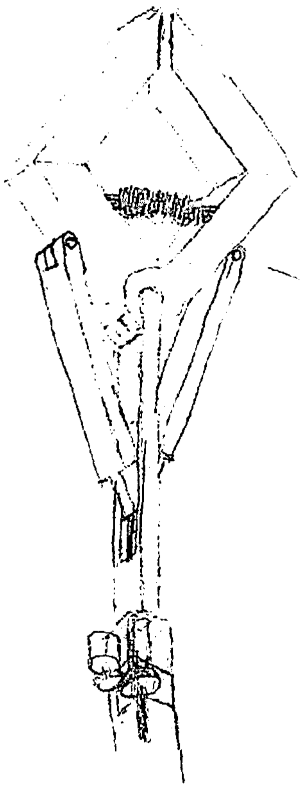
1. The enable circuit for the external device.
2. The READ/WRITE signal.
3. The output strobe signal.
4. The output storage latches.

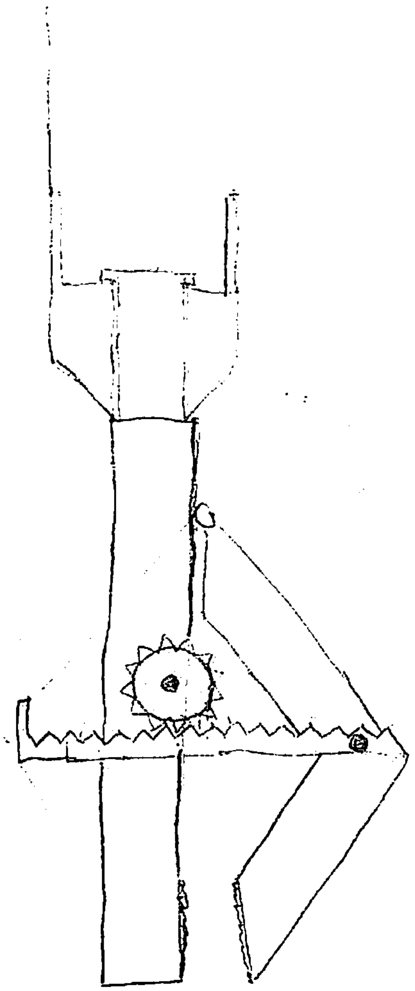
The four sections operate together to perform the overall output function. If any one of them fails to operate correctly, the physical output operation will not work. Let us discuss the function each section performs in an output operation. As each section is discussed, we will show how it can be realized with common digital electronic circuits for the Apple computer.

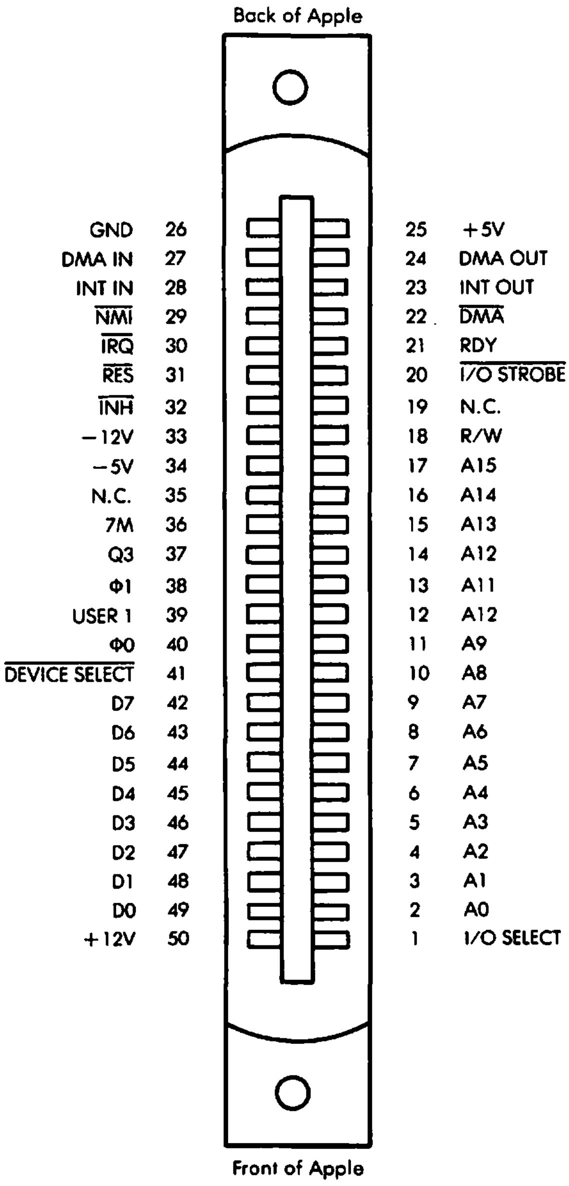
4.2: THE ENABLE CIRCUIT

To start, we concentrate on the function of the enable circuit in an output operation for the Apple computer. When an enable circuit is active, it means the computer will be electrically communicating with that output circuit. The computer is capable of electrically communicating with over 64,000 different circuits. Therefore, it must have some way of electrically informing any particular output circuit that it has been selected to communicate with.

The Apple computer has a built-in enable signal on each of the eight 50-pin I/O connectors, which were shown in Chapter 2. Figure 4.1 shows the pinout of an I/O connector slot, with the number and signal name of each of the 50 pins.







Pinout of the 50-pin I/O slot connector located at the rear of the Apple computer. Redrawn from Apple II Reference Manual, *by permission of Apple Computer, Inc.*

Figure 4.1

The enable signal we are interested in is called $\overline{\text{DEVICE SELECT}}$. It is output on pin 41 of the I/O connector shown in Figure 4.1. The bar over the top of the signal name indicates that when the signal is active, it is a logical 0. That is, when the signal is doing its electrical job it will be in the logical 0 state. At all other times this signal will be in the logical 1 state.

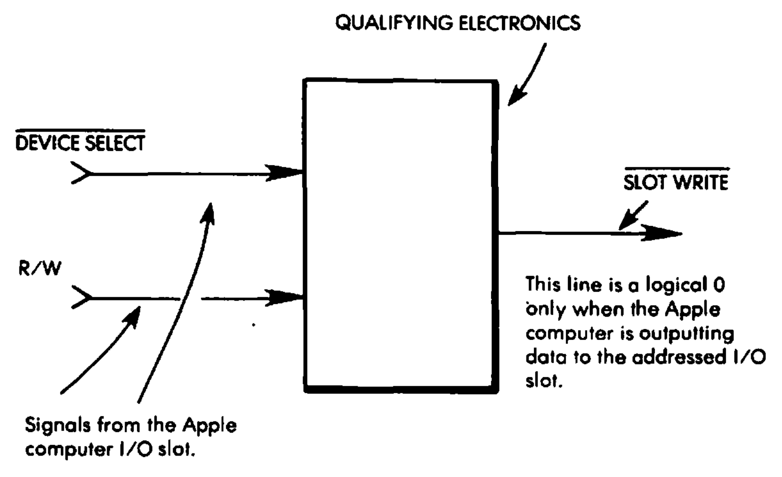
Whenever the computer selects a particular I/O slot to communicate with, the $\overline{\text{DEVICE SELECT}}$ line on that I/O connector will be a logical 0. The $\overline{\text{DEVICE SELECT}}$ line can be activated under software control, using the PEEK and POKE instructions. Recall that these two BASIC instructions each include an address. The logical state of the $\overline{\text{DEVICE SELECT}}$ output signal is dependent on the address specified in the PEEK or POKE instruction. For example, if we want the $\overline{\text{DEVICE SELECT}}$ line on I/O connector slot 3 to go to a logical 0, we can PEEK or POKE address −16208.

The $\overline{\text{DEVICE SELECT}}$ line is generated totally by the Apple computer. Let us assume our computer is operating correctly and the PEEK or POKE address is correct to generate the $\overline{\text{DEVICE SELECT}}$ signal. Further, let us assume that this line will go to a logical 0 whenever we wish to output information from the computer to the output circuit to which the active signal is applied.

4.3: THE READ/WRITE (R/W) LINE

Another hardware signal that we must make use of when performing computer output is the READ/WRITE or R/W line. This signal is output on pin 18 of the I/O connector shown in Figure 4.1. The R/W signal is a logical 1 whenever the Apple computer is reading information from the I/O slot, and a logical 0 whenever the Apple computer is writing data to the I/O slot. During a PEEK instruction, therefore, the R/W line is a logical 1, because the computer is reading data. During a POKE instruction, the R/W line is a logical 0 because the computer is writing data. This line is constantly toggling (that is, switching back and forth) between a logical 1 and a logical 0 even when the computer is not communicating with the output circuit we are designing. This is because every circuit in the computer makes use of the R/W signal.

The R/W line must be *qualified* in order for an output circuit to

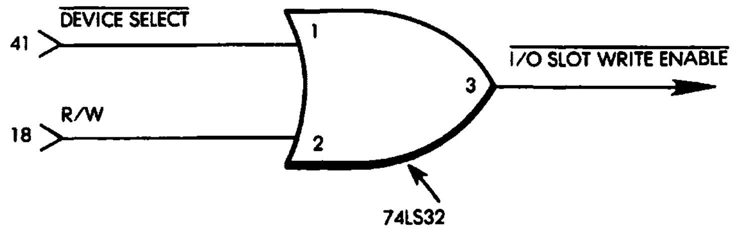

QUALIFYING ELECTRONICS
$\overline{\text{DEVICE SELECT}}$
R/W
$\overline{\text{SLOT WRITE}}$
This line is a logical 0 only when the Apple computer is outputting data to the addressed I/O slot.
Signals from the Apple computer I/O slot.

Block diagram showing how the R/W line must be qualified by the $\overline{\text{DEVICE SELECT}}$ line to generate the I/O SLOT WRITE signal.

Figure 4.2

make use of it. By that, we mean that the hardware must not only electrically examine the logical state of the R/W line to determine whether our circuit will be read or written to, but the circuit must also examine the logical state of the $\overline{\text{DEVICE SELECT}}$ line mentioned earlier. That is, the R/W line must be "qualified" by the $\overline{\text{DEVICE SELECT}}$ line before we can use it in the external circuit. Figure 4.2 shows a block diagram of exactly what we mean.

Figure 4.3 shows a hardware realization for the qualification of the R/W line to be used by an output circuit. In Figure 4.3 we see that the $\overline{\text{DEVICE SELECT}}$ signal is input to pin 1 of the 74LS32, an OR gate. The 74LS00 family of integrated circuits is widely available and will be used throughout this book. An OR gate is the name given to a digital logic device that will produce a logical 1 at its output only when input A OR input B is a logical 1. When both inputs are a logical 0, the output is a logical 0. Input A and input B are shown as pin 1 and pin 2 in Figure 4.3. The 74LS32 integrated circuit contains four individual OR gates. The R/W signal is input



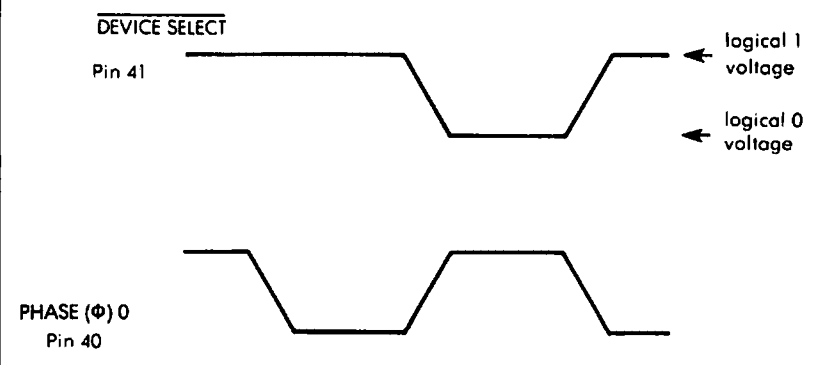
Schematic diagram showing how to realize the block diagram of Figure 4.2.
Figure 4.3

to pin 2 of the same OR gate. The output line of the OR gate, pin 3, will be a logical 0 only when both the R/W and the DEVICE SELECT line are a logical 0. That is, we have now "qualified" the R/W line. The output of the OR gate in Figure 4.3 is given the name I/O SLOT WRITE ENABLE. This line is a logical 0 only when the Apple computer is outputting data to the external circuit installed in the selected I/O slot.

4.4: THE EXTERNAL OUTPUT STROBE SIGNAL

The next section of the output hardware we will discuss is the external output strobe signal. Some confusion may arise here, because there is a signal on the 50-pin I/O slot connector labeled I/O STROBE (pin 20 on the pinout shown in Figure 4.1). We are *not* discussing that signal. The signal we *are* discussing is timed by PHASE (or Φ) 0. Do not confuse the strobe signal we are discussing with the I/O STROBE signal.

The external output strobe signal we are discussing informs the hardware of the exact moment when the computer is presenting the data intended for the selected output circuit. In a home



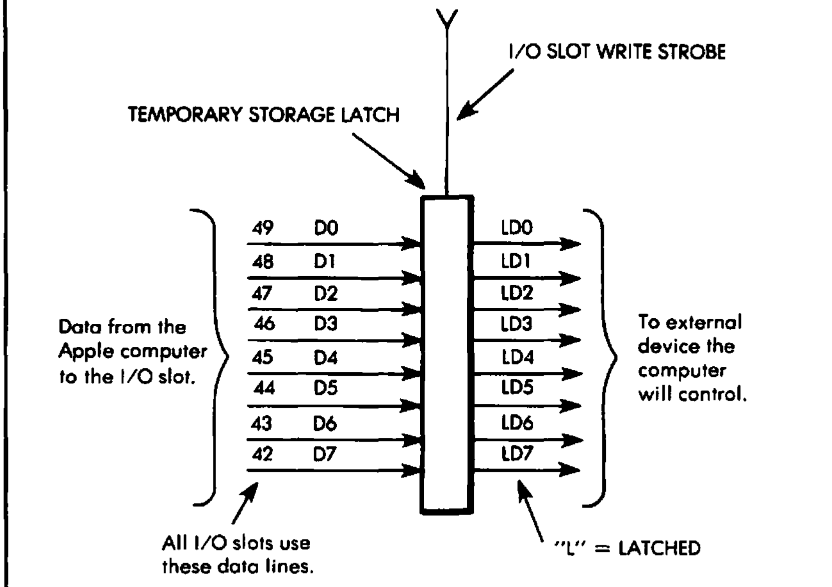
Timing diagram showing the relationship between the $\overline{\text{DEVICE SELECT}}$ line and the PHASE (ϕ) 0 clock. $\overline{\text{DEVICE SELECT}}$ is in its active (logical 0) state when PHASE 0 is in its logical 1 state.

Figure 4.4

computer (and most other computer systems), the data to be written to the output circuit is present for less than 2 millionths of a second. During that period, the hardware for which the data is intended must be electrically informed of the presence of data in order to be "turned on" to accept it. In the Apple computer, the PHASE 0 signal is used to time the output strobe function. When the PHASE 0 signal goes to a logical 0 level, the data from the computer will be present at the enabled output circuit.

Further, the output hardware must use this signal to *strobe* the data. By strobing the data, we mean that the data must be electrically stored temporarily in some hardware. The strobe signal is the write signal that will perform the temporary storing of the data into the actual hardware.

The Apple computer uses the PHASE 0 signal in the generation of the $\overline{\text{DEVICE SELECT}}$ signal. (This feature makes the Apple simpler to interface than many other microcomputers, which are not designed this way.) Therefore, whenever the $\overline{\text{DEVICE SELECT}}$ line is active, it is also timed properly to be used by many I/O devices to strobe data. Figure 4.4 is a timing diagram showing the relationship between the $\overline{\text{DEVICE SELECT}}$ signal and the PHASE 0 clock.


I/O SLOT WRITE STROBE
TEMPORARY STORAGE LATCH
49 D0
48 D1
47 D2
46 D3
45 D4
44 D5
43 D6
42 D7
LD0
LD1
LD2
LD3
LD4
LD5
LD6
LD7
Data from the Apple computer to the I/O slot.
To external device the computer will control.
All I/O slots use these data lines.
"L" = LATCHED

Block diagram showing how the data output lines from the Apple computer will connect to the temporary storage latches located in the I/O slot output circuit.

Figure 4.5

4.5: THE OUTPUT LATCHES

The final section of the output hardware we will discuss comprises the output latches. Output latches have the very important function of temporarily storing the data sent by the Apple computer to an external circuit. Recall that the POKE instruction included an address and data. The address will select the I/O slot and the data specified will be sent to the external circuit. The output latches will store this data.

The computer will present the data to the input pins of the output latches. (See Figure 4.5.) At this time, the $\overline{\text{I/O SLOT WRITE STROBE}}$ signal will strobe the latches. In our system, the $\overline{\text{I/O SLOT WRITE ENABLE}}$ and the $\overline{\text{I/O SLOT WRITE STROBE}}$ are the same signal. After the data has been written into the latches, it will remain stored there until the computer is again instructed to write other data to the circuit. To put it another way, when data is POKED into the storage latches, it will remain there until another POKE instruction directs new data to the same address.

The data at the output of the latches can now be used to control any hardware that we desire. At this point, you may not have a clear picture of exactly what hardware can be controlled. Don't worry. We will introduce some examples in later chapters. The main point to be stressed is that we now have a means of forcing any single logical line to switch between a logical 0 and a logical 1 level under computer control. We have achieved an important objective in the interfacing problem. When you understand how this was done, a major step has been completed.

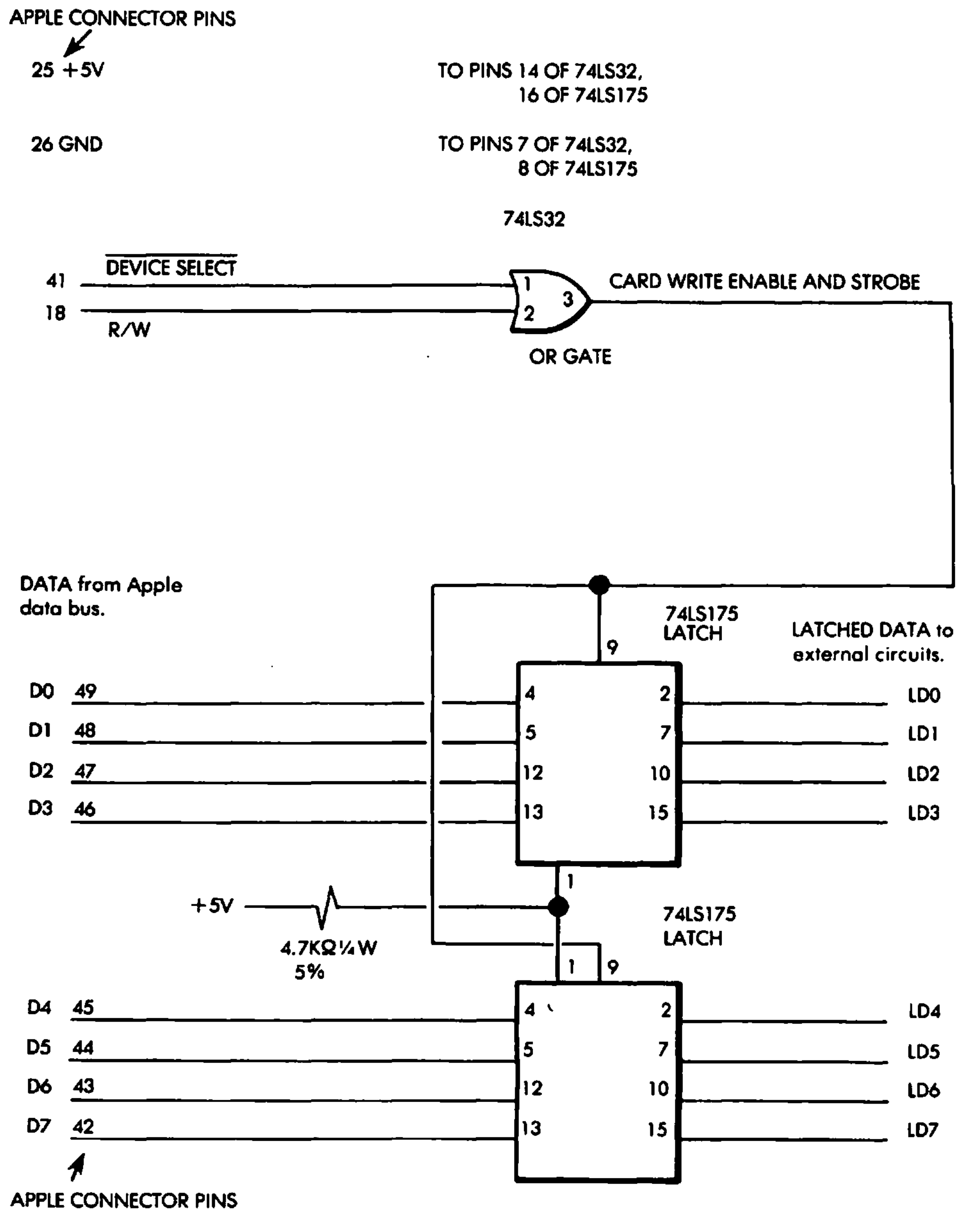
Now let us take a closer look at the hardware of the output latch circuits. Figure 4.6 shows the latch devices we will use and the pin numbers that incoming and outgoing data lines will connect to. The latches used are 4-bit 74LS175s. An Apple computer will transfer eight bits of data during every output operation. For this application we need two 4-bit latches in parallel to accommodate the eight data lines. Each latch is strobed at the same time. Data from the computer is input to the latches from the Apple data bus lines D7–D0.

Data from the Apple will be applied to the data input pins of the 74LS175s. Pin 9 of each 74LS175 is the strobe input. When this line is active it goes from a logical 0 to a logical 1. The data present at the input of the latch is "captured" and stored internally in the latch. Recall that when the strobe line is activated, it is in a logical 0 state. When this line returns to the logical 1 state, the data is latched.

We have followed the data, along with the $\overline{\text{DEVICE SELECT}}$ and R/W signals, out from the pins of the Apple's I/O slot, and into the latches of an I/O board. The data is now latched, and we are ready to use this data to control an output device.

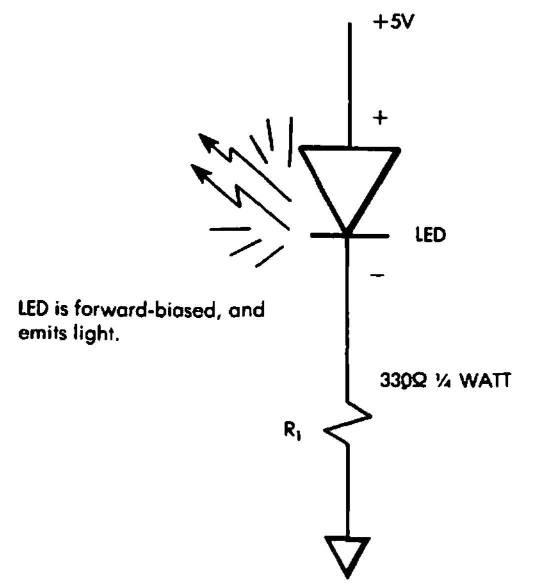
4.6: THE LIGHT-EMITTING DIODES

The output device that we will be controlling in this first example will be a series of light-emitting diodes, or LEDs. Eight LEDs are used, one for each data output line. By using LEDs, we can gain



Complete schematic showing the hardware required to interface an output circuit to the Apple computer.

Figure 4.6



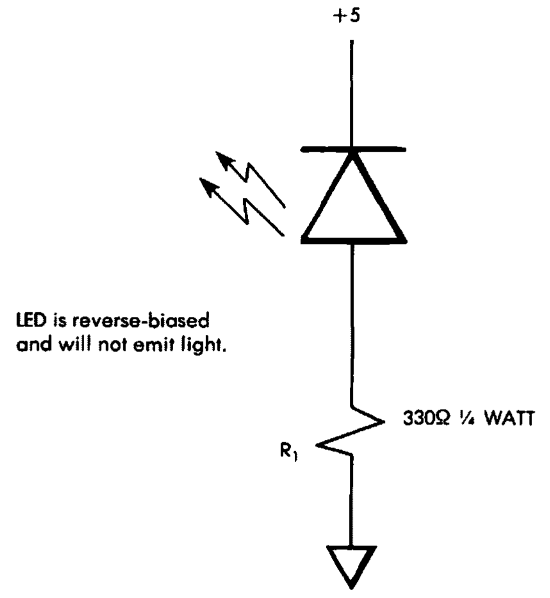
Schematic diagram of a typical LED connected in the forward-biased mode.
Figure 4.7

experience in turning on and off selected bits of the output device. We did this in Chapters 2 and 3 by using the CMS I/O system for the Apple computer. At this time, you may choose to construct your own I/O system using the schematics given in this chapter.

When we turn on and off selected LEDs, we are performing exactly the same function as when we control most types of external devices. We will now concentrate on how the hardware does its part of the job.

Figure 4.7 shows a schematic diagram of how the LEDs are electrically driven to light. We see in this diagram that the LED has approximately the same electrical symbol as a standard diode. The difference is that the LED symbol has some arrows drawn out from it, an indication that the diode is emitting light.

The LED performs approximately the same electrical function as a standard diode. The important major difference is that when



Schematic diagram of an LED connected in the reverse-biased mode.

Figure 4.8

the LED is forward-biased, it will emit light. The color of light is determined by the materials used to fabricate the LED. Typical LED colors are red, green and yellow. In Figure 4.7 we see that the *anode* of the LED is connected to +5 volts. The anode is the positive (+) side of the device. The *cathode*, or negative (−) side of the LED, is connected to one end of a resistor. To turn on the LED, the other side of the resistor must be connected to ground potential, or approximately 0.0 volts.

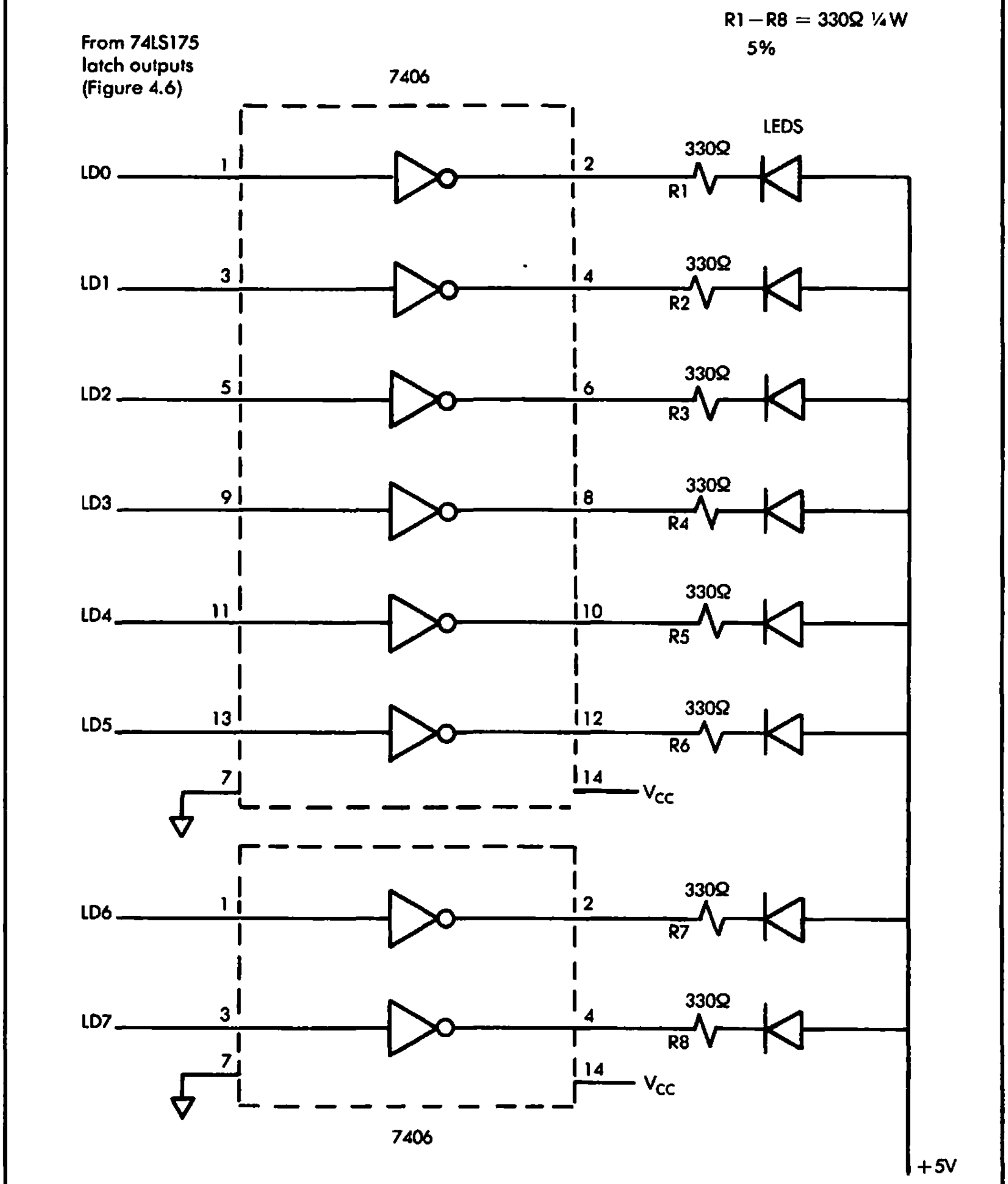
If you have never used an LED before, it might be fun and instructive to construct the circuit shown in Figure 4.7 and manually turn on and off the diode. This is done by connecting and disconnecting the anode to +5 volts. When you do this, try reversing the connection by interchanging the diode leads, and note what happens. That is, connect the cathode (−) side of the LED to +5 volts and the anode (+) side of the LED to the resistor. Connect the other side of the resistor to ground. (See Figure 4.8.) The LED should not light under these conditions, because it is now connected in a *reverse-biased* mode.

Here is how the circuit of Figure 4.7 works. With the anode of the LED connected to +5 volts, the cathode must be connected to ground *potential* to light the diode. There must not be a direct connection of the cathode to ground. If there were, too much current would be passed through the LED, and it would burn up. Therefore, a current-limiting resistor, R1, is required. The function of R1 is to limit the current to a safe value in the LED when it is connected in the forward-biased mode. Typical LED currents are around 10 milliamperes or .01 amperes. This amount of current will usually give a nice degree of brightness.

Expanding the concept of a single LED to eight, we have the circuit shown in Figure 4.9. In this schematic diagram, the resistors R1–R8 are connected at one end to the LEDs, and at the other end to the outputs of a 7406 integrated circuit. The 7406 will act as a switch to connect the resistors to ground potential. For example, when the input pin 1 of the 7406 shown in Figure 4.9 is a logical 1, the output pin 2 is connected to ground potential. Under these conditions, the LED is turned on, or forward-biased.

When the input pin 1 of the 7406 is a logical 0, the output pin 2 is not connected to ground. The output is essentially an open circuit; that is, no current is passing through it. Under this condition the LED is turned off. From these two conditions, we see that all we need to do to turn the LED on and off is to place a logical 1 or a logical 0 at the input of the 7406. As long as the input stays at a logical 1 or a logical 0, the LED will remain either on or off.

What we have described and discussed in the preceding sections is really the essential idea of output-hardware interfacing. The problem we solved was how to get any amount of information from a specific place or location to another specific place, with every bit under precisely planned control in order to accomplish a particular task. We place a logical 1 or a logical 0 on some digital input line, and the result is an action taking place in the output device. In the case just discussed, the action was simply turning on and off an LED. Although this action is simple, it accurately reflects the basic principles of output interfacing.



Schematic diagram of the driving circuits for eight LEDs.

Figure 4.9

The inputs to the 7406 shown in Figure 4.9 will come from the outputs of the 74LS175 latches (shown in Figure 4.6) that we discussed earlier. Latched outputs, once set, will remain at a logical 1 or a logical 0 until we write another byte to the device using the POKE instruction. In this way we can turn on and off the LEDs under computer control. A BASIC program can be generated that will write data to the output device, causing the hardware to respond in the manner described. We discussed this type of software in Chapter 2.

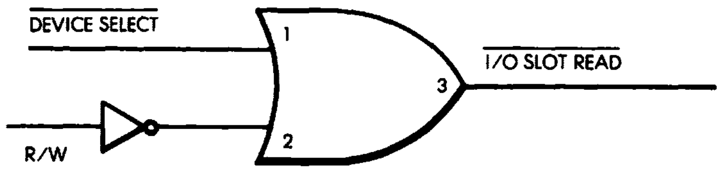
The overall result will be a turning on and off of different LEDs under our control via the program written. Simple as it appears, this experiment is an excellent place to begin addressing the problem of interfacing different types of external hardware because it embodies typical elements involved in all present-day computer output transfers of information.

4.7: HARDWARE FOR INPUTTING DATA TO THE APPLE

In the preceding discussion, we examined the hardware for outputting data from the Apple computer. Now, we will turn our attention to inputting data to the Apple from an outside source. That is, some external device will output data; this data will be input to the Apple. The Apple computer can make decisions based on the input data, store the data, or alter the data in any manner we desire. We discussed these operations in Chapter 3. Let us now discuss how to design the hardware that will allow the input of external data to the Apple computer.

The same general comments we made about the output hardware at the beginning of this chapter apply here. This hardware is presented mainly for its educational value, to illustrate the basic concepts involved in computer interfacing. The hardware shown is very simple and it does work. Like the output hardware discussed earlier, it has been designed to use standard "off-the-shelf" integrated circuits, so you can actually construct and experiment with it. In fact, experimentation of this type is highly recommended prior to designing any interface circuits.

To input data to a BASIC program, we will use the PEEK instruction described in Chapter 3. The PEEK instruction will


$\overline{\text{DEVICE SELECT}}$
R/W
1
2
3
$\overline{\text{I/O SLOT READ}}$

Schematic diagram of the hardware necessary to qualify the R/W line during an input read operation from the I/O slot on the Apple computer.

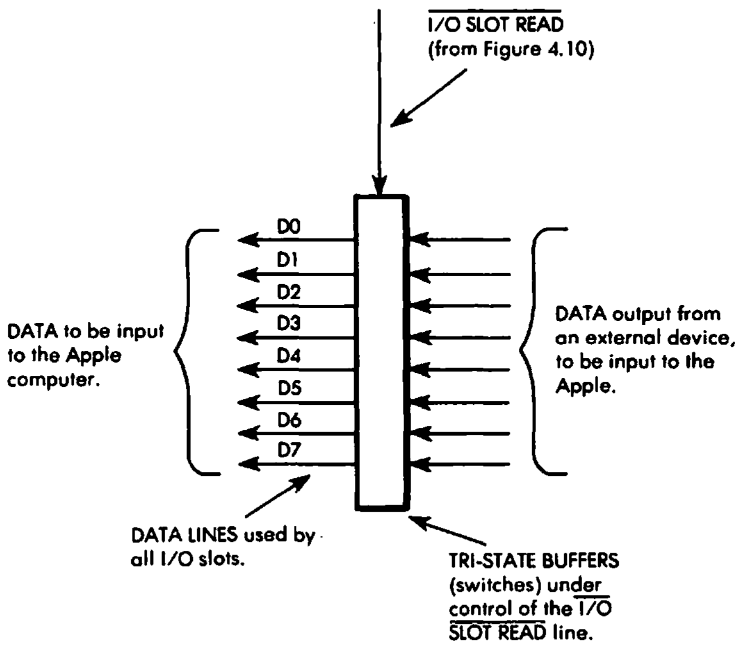
Figure 4.10

use an address to PEEK from. This address will be logically decoded to enable the $\overline{\text{DEVICE SELECT}}$ line, which we discussed in detail in the output section. That is, whenever the software is performing a PEEK at the correct address for the I/O slot, the $\overline{\text{DEVICE SELECT}}$ line will be a logical 0 at the I/O slot connector.

The PEEK instruction will be performing a read function and the R/W output line on the I/O slot connector will, therefore, be a logical 1. (Recall that a logical 0 was used for the write function on the same line.) Figure 4.10 shows how the $\overline{\text{DEVICE SELECT}}$ line and the R/W line are logically combined to generate an $\overline{\text{I/O SLOT READ}}$ signal. The $\overline{\text{I/O SLOT READ}}$ signal shown in Figure 4.10 will be a logical 0 whenever the Apple computer is reading data from the address of the selected I/O slot.

During the time the Apple is reading the data from the selected I/O slot, data from the I/O slot is electrically enabled onto the Apple data lines, as shown in Figure 4.11. In this diagram we see that the data from the external device will be electrically placed on the Apple data lines when the $\overline{\text{I/O SLOT READ}}$ signal of Figure 4.10 is active.

Figure 4.12 shows a complete schematic of a circuit that will input data to the Apple computer from an external source. This circuit will employ the circuits of Figures 4.10 and 4.11. As you can see from this schematic, there is very little hardware required.


$\overline{\text{I/O SLOT READ}}$
(from Figure 4.10)
D0
D1
D2
D3
D4
D5
D6
D7
DATA to be input to the Apple computer.
DATA output from an external device, to be input to the Apple.
DATA LINES used by all I/O slots.
TRI-STATE BUFFERS (switches) under control of the $\overline{\text{I/O SLOT READ}}$ line.

Block diagram indicating that external data is first input to a tri-state buffer. The buffer outputs are connected to the data lines of the Apple computer. The $\overline{\text{I/O SLOT READ}}$ line (shown in Figure 4.10) will enable the buffer outputs onto the Apple data bus at the correct time.

Figure 4.11

Further, the hardware is simple in its operation. Let us discuss exactly how the hardware shown in Figure 4.12 operates.

At the center of Figure 4.12 is the 74LS244 octal, tri-state buffer. This buffer takes input on pins 2, 4, 6, 8, 11, 13, 15 and 17. These inputs are labeled CD0–CD7. (The CD is short for *card data,* which is the logical state of the data lines to be input to the Apple computer.) These inputs are held at either a logical 1 or a logical 0 level. The input pins can hold any digital information desired.

We are not now interested in what data might be input, but for the purpose of discussion we will assume that these input lines actually represent some digital information to be sent to the Apple computer. The objective of this discussion is to show *how* that information can be input to the Apple computer. You must be able to master this hardware before proceeding on to the next step in the interfacing process.

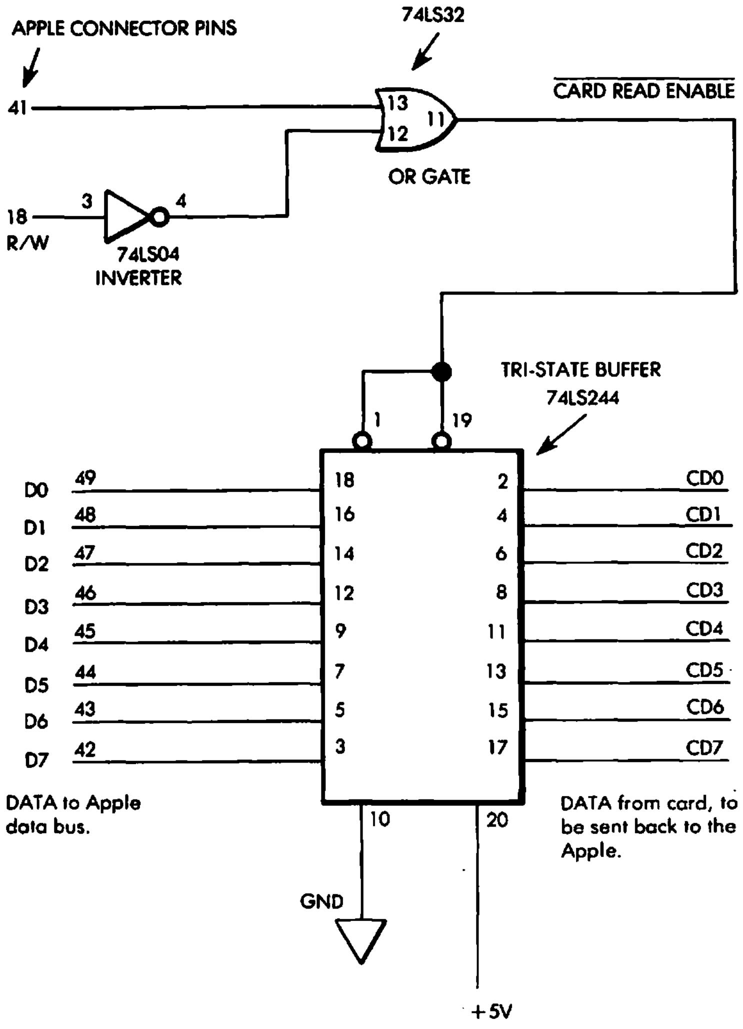
In our circuit the lines CD0–CD7 will be connected to a DIP switch. DIP stands for *Dual In-line Package.* A schematic of this type of switch is shown in Figure 4.13. We see in this diagram that one side of the switch is connected to ground. The other side of the switch is connected to the CD input lines of the 74LS244 buffer shown in Figure 4.12. When the switch is closed, the input to the 74LS244 is a logical 0. When it is open, the input to the 74LS244 is a logical 1. We are making use of the fact that an open, or floating, TTL (Transistor-Transistor Logic) input is a logical 1. TTL is a family of digital electronic circuits. The Apple computer uses TTL for some of its internal circuits.

By using the switch, we can force any of the eight inputs to a logical 1 or a logical 0. This data will be sent to the Apple computer. Note that this is exactly how the CMS I/O system we described in Chapter 3 operated.

The outputs of the 74LS244 shown in Figure 4.12 are connected to the Apple computer data bus lines, D7–D0. The outputs of the 74LS244 must not be enabled or turned on unless the Apple computer is electrically requesting the data from the input slot. At all other times, the data outputs from the buffer are set into a tri-state, or off, mode. This mode will electrically remove the 74LS244 outputs from the Apple data bus. The only time the outputs of the latch will be placed on the Apple data bus is when the input pins 1 and 19 of the 74LS244 are set to a logical 0. These two pins are connected to the OR gate, and thus to the Apple's R/W and $\overline{\text{DEVICE SELECT}}$ lines.

4.8: ENABLING THE TRI-STATE BUFFER

The output of the 74LS244 buffer will be electrically placed on the data bus only when pins 1 and 19 (the enable pins) are a logical 0. When this occurs, it is because the Apple computer is electrically requesting data from the selected I/O slot. This will occur during a PEEK instruction. Let us examine how the enable pins are set to a logical 0 under control of the computer.



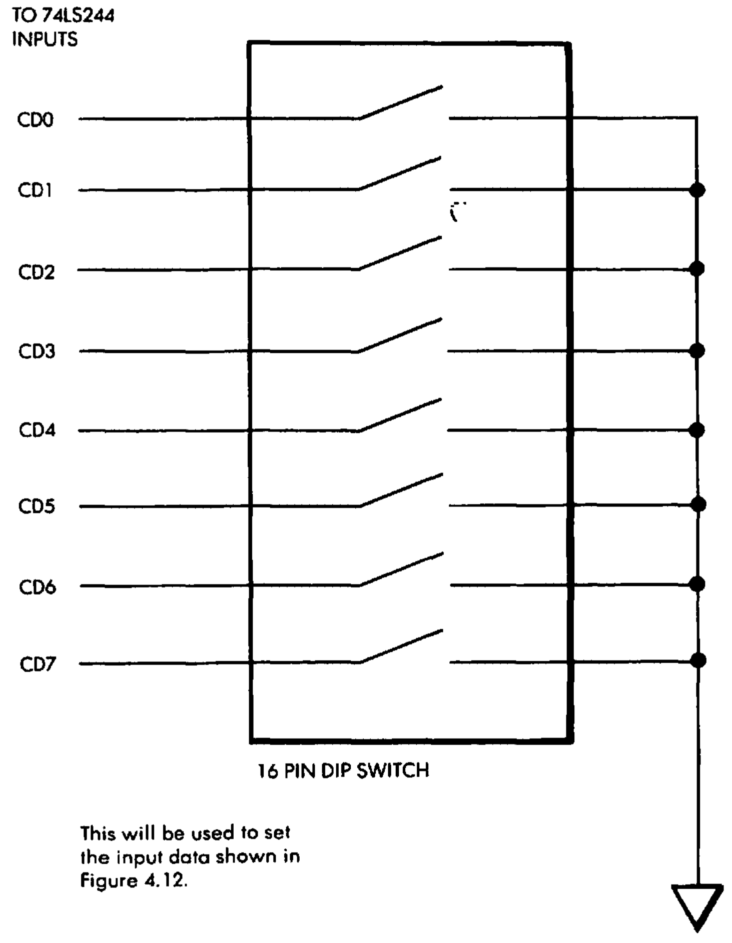
Complete schematic of the hardware necessary to input an external data byte to the Apple computer from an I/O slot.

Figure 4.12

When the computer sends out an address that is equal to the address for the I/O slot, the $\overline{\text{DEVICE SELECT}}$ line becomes active, as we saw in our discussion of the mechanics of output. This line is input to pin 13 of the 74LS32 of Figure 4.12. Remember that this line is active during a write or POKE operation also. Therefore, we must use the R/W line as a qualifier. When the R/W line is a logical 1, the Apple computer is electrically expecting some external device to send data to it. In Figure 4.12 the R/W line is inverted via the 74LS04. When the R/W line is a logical 1, the input pin 12 of the 74LS32 is a logical 0. With both pins 12 and 13 of the 74LS32 a logical 0, pin 11, the output, is also a logical 0. This pin is connected to the enable input pins 1 and 19 of the 74LS244.

In review, the following will occur during a read or PEEK operation from an I/O slot:

1. The Apple computer will output the correct address that will enable the proper $\overline{\text{DEVICE SELECT}}$ line.
2. The R/W line will go to a logical 1. This action will electrically inform the system hardware that the Apple computer is expecting an external device to place data on the system data bus. When the R/W line goes to a logical 1, the enable input pins 1 and 19 of the 74LS244 are set to a logical 0. At this time, the data at the inputs of the buffer are placed on the Apple data bus.
3. During the time the external data is enabled onto the system data bus, the Apple computer will automatically read the data. In Chapter 3 we discussed how the software will use the data that was read.
4. After a fixed period of time (approximately 1 millionth of a second), the Apple computer will place another address on the system address bus, and the $\overline{\text{DEVICE SELECT}}$ signal will go to a logical 1. The address change is performed automatically by the internal circuits of the Apple computer. A user need not be concerned about forcing this event to occur. When it does occur, the enable input pins 1 and 19 of the 74LS244 will be set to a logical 1. This action will tri-state, or turn off, the 74LS244 buffer outputs, which will electrically remove them from the Apple data bus lines.


TO 74LS244
INPUTS
CD0
CD1
CD2
CD3
CD4
CD5
CD6
CD7
16 PIN DIP SWITCH
This will be used to set the input data shown in Figure 4.12.

Schematic diagram of eight switches used to set the logical level of each external input line at the I/O slot.

Figure 4.13

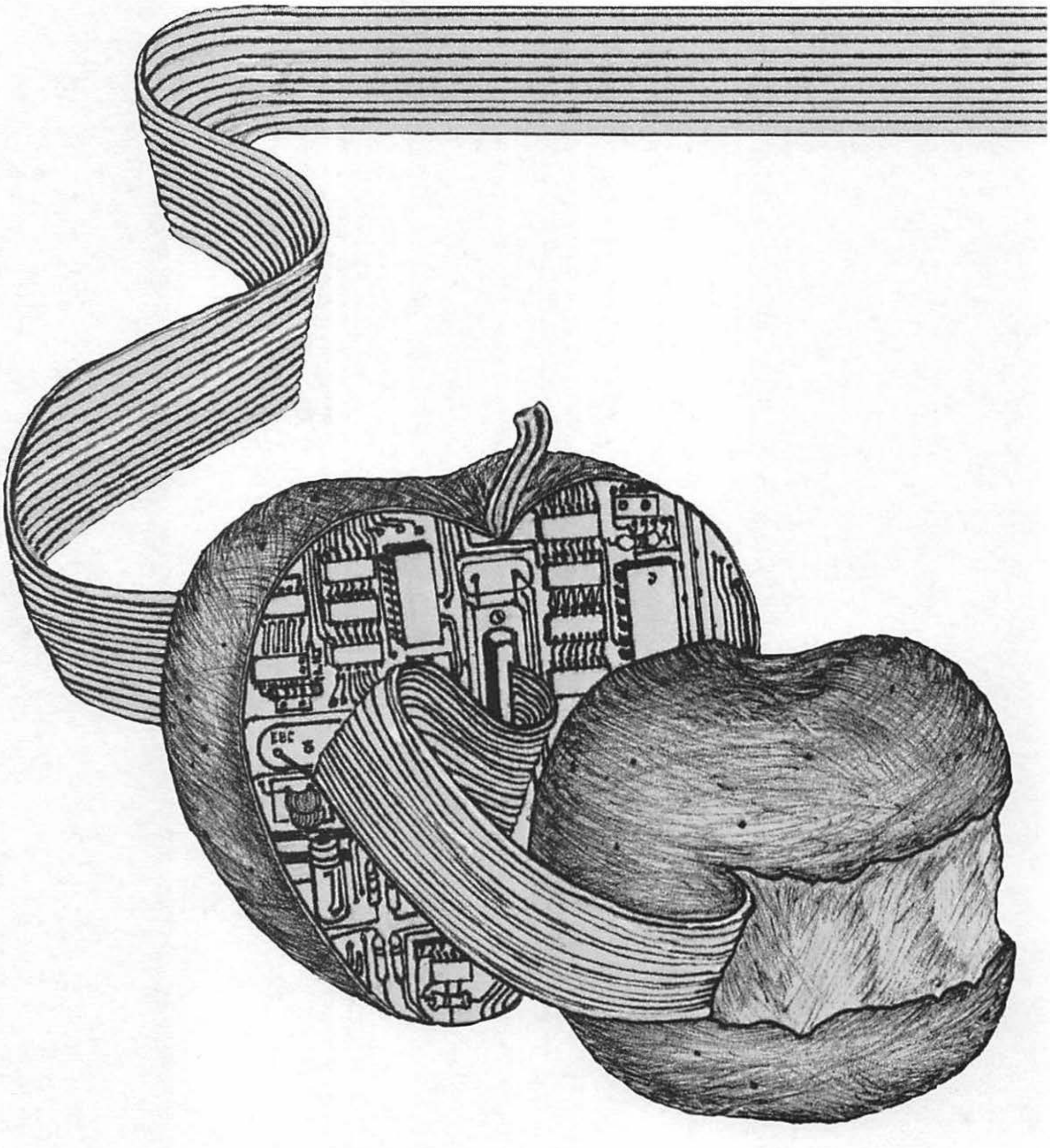
4.9: SUMMARY OF INPUT AND OUTPUT

In this chapter we have presented the essential details of inputting and outputting data with the Apple computer. The schematics for the actual hardware were shown and discussed. You can build these circuits if you want to.

If you understand the information presented in this chapter, you have taken an important step in the interfacing process. The next step will be to connect the input and output lines to different types of external hardware. This will allow the computer to control other hardware besides the LEDs shown here.

In the next chapter we make use of the hardware and software concepts we have explored to design a home security system. In Chapter 6 we will discuss how to use the digital outputs to control home appliances. We will show how to turn on and off home lights, coffee pots, toasters and other appliances that plug directly into the wall.

However, before you attempt to accomplish this, you must have a good understanding of the information presented in this chapter. It is recommended that the beginner in computer interfacing take the time to construct the circuits presented here. All of the hardware components are readily available from many suppliers (see Appendix D), and there are circuit boards available for wiring up circuits that will plug directly into the Apple computer's I/O slots.


EBC

Chapter 5

An Application of Computer Interfacing: A Home Security System

IN THIS CHAPTER we will present the complete hardware and software system for a possible home security system. The controlling software for this application will be written in BASIC. We will use information that was given in Chapters 1–4. This chapter will help to bring together all of the important points covered in the first four chapters of this text.

The system to be discussed does work, but like the simple circuits presented earlier, its primary purpose is instruction. The intention is to show you one way such a system can be designed. Once the problem has been solved in some manner, adapting the solution to a particular case will be much easier. It is hoped that, after you have been shown one way the Apple computer can be used to monitor and secure a home, applying this information to your own home will be straightforward, exciting, and fun. You can use this chapter as a starting point for designing other external systems to be controlled by the Apple computer. By the conclusion of the chapter, you will see that this is not too difficult a task, and is a rather enjoyable challenge.

5.1: DEFINITION OF THE PROBLEM

Let us start this overall design with a definition of the problem. The first step in any design problem is to define exactly what the system is to do. If the problem is clearly defined at the outset, then we have a direct path to follow. If we do not know exactly what we want from the beginning, it will be extremely difficult to decide as we go along.

A first, general definition is this: *The system to be designed and realized will monitor the status of all doors and windows in a home.* Already we see a problem: there may be some windows that cannot be opened. These windows will not be monitored. Also, there are many doors in a home that do not directly lead to the outside. So let us refine our definition thus: *The system will monitor the status of any door that leads to the outside of the home and any window that can be opened.* Unfortunately, the system definition is still too vague. What exactly is meant by the term "to monitor the status" of something? In this case, the term means *to indicate whether the door or window is open or closed.*

The security system will determine whether a window or door is open or closed. If the system determines that a door or window is open, what action will be taken? That is the next part of the problem we must define. If a door or window is open, the system will indicate visually on the CRT screen of the Apple computer which door or window it is.

Here, now, is the complete definition of the system.

1. The system will monitor every door in the house that opens directly to the outside.
2. The system will monitor every window in the house that can be opened.
3. If the system detects an open window or door, it will visually indicate this on the CRT screen of the Apple computer.

There are certainly more functions you could design into the system. It depends on how complex you wish the system to become. Some systems have been designed to incorporate a modem and a voice synthesizer to call the police if one of the windows or doors is tampered with. This example is given only as an illustration of what can be done. The definition of the system that we have given

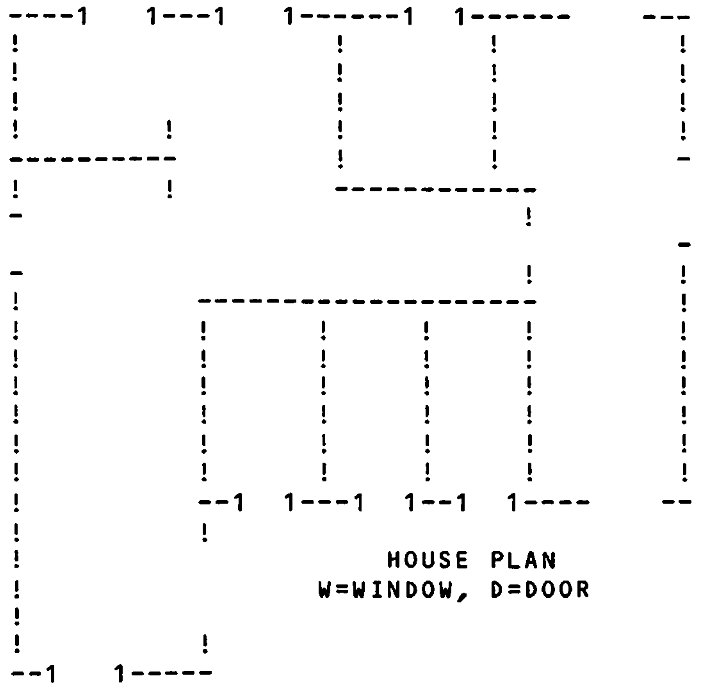


Diagram of the outline of a house. This outline was drawn by the computer. There are no labels for the doors and windows. These will be added later.

Figure 5.1

will enable us to highlight the important details of interfacing and computer control with the Apple, without making the task needlessly complex.

5.2: DRAWING THE HOUSE WITH THE COMPUTER

Part of the definition of the problem is to indicate on the screen of the Apple computer if any of the doors or windows being monitored are open. This can be done in any number of ways. The technique we will use starts with an outline of the house, similar to an architect's blueprint. This outline will be displayed on the screen of the computer. Each window or door being monitored will be shown. Figure 5.1 shows exactly what we will draw on the screen of the CRT.

```
200 PRINT"----1   1---1   1------1  1------   ---"
210 PRINT"!                  !        !         !"
220 PRINT"!                  !        !         !"
230 PRINT"!                  !        !         !"
240 PRINT"!        !         !        !         !"
250 PRINT"----------         !        !         -"
260 PRINT"!        !         ------------        "
270 PRINT"-                             !        "
280 PRINT"                                      -"
290 PRINT"-                             !       !"
300 PRINT"!          --------------------       !"
310 PRINT"!          !      !     !     !       !"
320 PRINT"!          !      !     !     !       !"
330 PRINT"!          !      !     !     !       !"
340 PRINT"!          !      !     !     !       !"
350 PRINT"!          !      !     !     !       !"
360 PRINT"!          !      !     !     !       !"
370 PRINT"!          --1  1---1  1--1  1----   --"
380 PRINT"!          ! "
390 PRINT"!                     HOUSE  PLAN"
400 PRINT"!                 W=WINDOW,  D=DOOR "
410 PRINT"!
420 PRINT"!          ! "
430 PRINT"--1   1-----"
```

BASIC program for drawing Figure 5.1 on the Apple computer. This program consists of 24 print statements. Notice that each print statement has 39 characters (including blanks) between the quote marks.

Figure 5.2

We see in Figure 5.1 that each room of the house is shown. This program will be written using standard PRINT statements on the Apple computer. We could make use of the graphics capability of the Apple computer for this task, but we will not do that because we also need text printed. The standard Apple computer will not allow text to be mixed with graphics without a special software package.

Also shown in Figure 5.1 is the location of each window and door that is being monitored. Each door and window is given a label. When the system detects that a door or window is open, the label associated with that door or window will blink in inverse video. This gives a quick visual indication of the exact status of any door or window in the house.

Another feature of the program we will write is that the user will have the option of ignoring any window or door that is open. For example, suppose it is very hot, and you are purposely leaving a window in the bedroom open. The program will ask which windows or doors you wish to ignore. The display on the screen will show that the door or window is open by staying in inverse video. However, the label will not blink.

Figure 5.2 is a BASIC program for drawing the layout of Figure 5.1 on the screen of the Apple computer.

5.3: PHYSICAL CONNECTIONS TO THE DOORS AND WINDOWS

We have now drawn a layout of the house on the screen of the computer. Next, let us discuss how the physical and electrical connection to the windows and doors is made. We need a device that will transform the motion or position of a window or door into an electrical signal. A common switch will do this.

There are various types of switches that can be used. The type we will use is installed so that the window or door position will open or close the switch. Figure 5.3 shows a schematic diagram of the type of switch to be used.

We see in Figure 5.3 that when a window or door is closed, its switch will be closed. When that window or door is open, the switch will be open. We now have a device that will transform a physical event, a change in position, into an electrical event, the signal to the computer.

The electrical quantity the switch uses is resistance. When the switch is closed, corresponding to a closed door or window, the resistance of the switch is approximately zero ohms. When the switch is open, corresponding to an open door or window, the resistance is infinite ohms.

We will use the characteristics of a switch to generate a digital

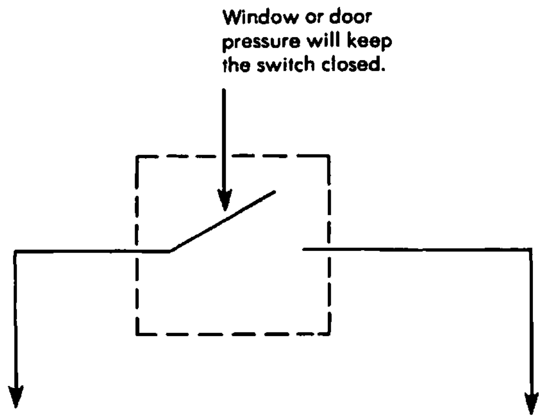
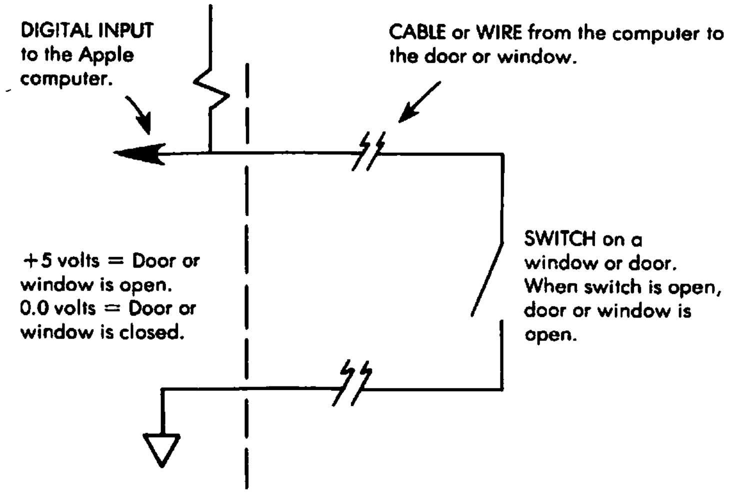


Diagram showing one type of switch that can be used to monitor doors and windows. The pressure of the door or window will keep the switch closed.

Figure 5.3

voltage signal that can be input directly to the Apple computer. Figure 5.4 shows how this will be accomplished. One side of the switch will be connected to the ground of the computer. (We will explain exactly how to do this later in the discussion.) The other side of the switch is connected to an input circuit on the Apple computer, exactly like the circuit described in Chapter 4.

When the switch in Figure 5.4 is open, there will be no electrical path to ground through the switch. This will allow the input line connected to the 4.7k-ohm resistor to be pulled up to +5 volts. This voltage level corresponds to a logical 1 voltage level in the Apple computer. When the window or door is closed, the switch will close. This forces the input line connected to the resistor to ground. Now there is an electrical path established through the switch to ground. A voltage of ground potential is equal to a logical 0 in the Apple computer. Another way of saying this is that when the door or window is open, the digital input line to the Apple computer is equal to +5 volts, which is a logical 1. When the door or window is closed, the digital input line to the Apple computer will equal 0.0 volts, or a logical 0.


DIGITAL INPUT to the Apple computer.
CABLE or WIRE from the computer to the door or window.
+5 volts = Door or window is open.
0.0 volts = Door or window is closed.
SWITCH on a window or door. When switch is open, door or window is open.

Schematic diagram showing how the switch will be wired. One side of the switch will be connected to the Apple computer ground and the other side of the switch will connect to a 4.7k-ohm resistor. It does not matter which side of the switch is connected to ground.

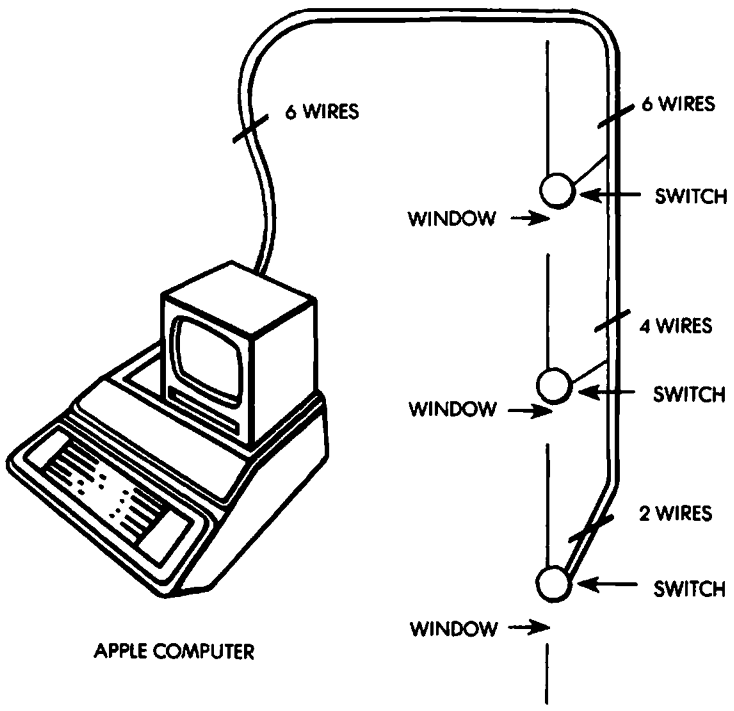
Figure 5.4

Knowing this will enable us to examine the status of each door or window under software control. We can do this using the PEEK instruction and the software discussed in Chapter 3. Later, we will give the complete program for controlling the system.

We have discussed in general terms how the switch will be placed on the door or window to be either opened or closed by the position of the door or window. Unfortunately, we cannot give an exact, detailed description of how to install these switches in your home, because everyone's windows and doors, and the type of switches used, may be slightly different.

However, there is a positive side to this. It gives you, the user, complete freedom to decide exactly how you want to install the system in your home. It has been the author's experience that the end user will usually improve on any method described in a text. Therefore, this book will give general guidelines that will get you started in the correct direction.

If you have a number of doors and windows that you wish to

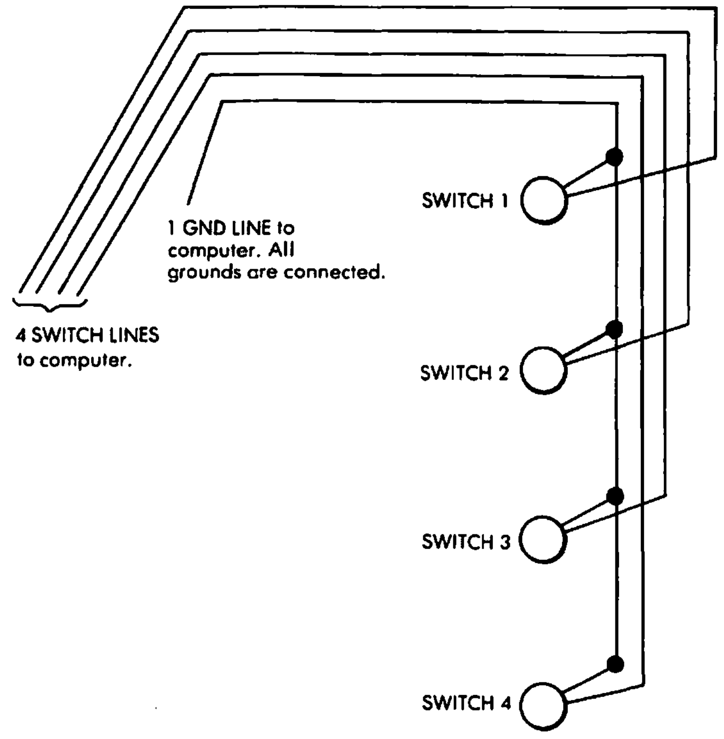


Pictorial diagram indicating that each switch to be monitored will require two wires sent back to the Apple computer. There are three switches shown here.

Figure 5.5

monitor there will need to be quite a few electrical connections to the Apple computer. The voltages and currents in the circuits are very low, so you can use light-gauge wire in the connections. What is sold as "speaker wire" has been used very successfully in an application such as this.*

*A note of caution: Although these wires are *insulated*, they are not *shielded*. A sufficient length of unshielded wire can act as an antenna, and attract induced pulses, often of very high voltages that can damage your computer. The chance of this happening in most installations is remote, but in areas where thunderstorms are common it should be considered. Other sources of induced pulses can include static electricity, residential power surges and spikes, and local radio transmitters. *Don't! (or How to Care for Your Computer)* by Rodnay Zaks (Sybex, 1981), describes these risks and ways of reducing them.


SWITCH 1
1 GND LINE to computer. All grounds are connected.
4 SWITCH LINES to computer.
SWITCH 2
SWITCH 3
SWITCH 4

One method of reducing the number of wires that must be connected to the Apple computer. In this scheme we use only a single ground wire for all of the switches.

Figure 5.6

Figure 5.5 shows a block diagram of exactly what is occurring when one connects the switches to the doors and windows. We see in this diagram that each switch will require two wires connected to it. This means that for every switch installed, you will have two wires running back to the computer. This could be a bit cumbersome.

Figure 5.6 shows one way to reduce the number of wires that need to be connected to the computer. The idea behind this diagram is to connect the ground lines of each of the switches together. After the grounds are connected, a single ground wire can be connected back to the computer. It should be noted that it will make no electrical difference which side of the switch is connected to ground.

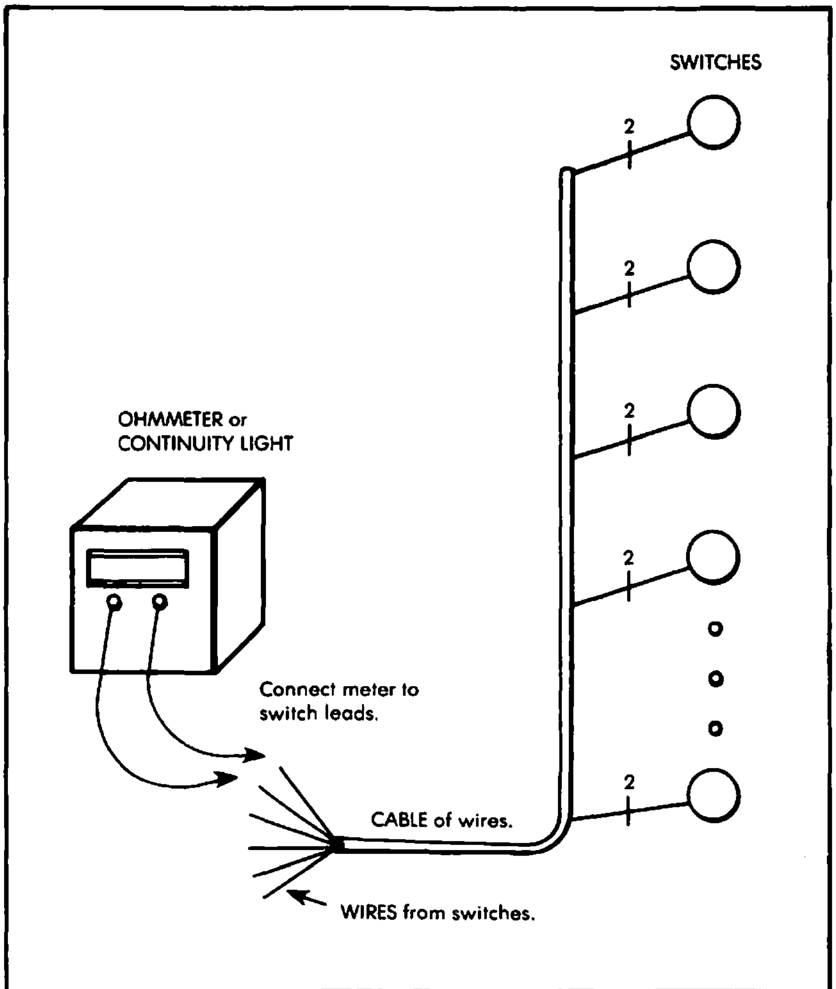
5.4: CONNECTING THE HARDWARE TO THE COMPUTER

At this time we have a bundle of wires from the various door and window switches ready to be input to the computer. The next step would seem to be to electrically connect the wires to the Apple computer. However, before we discuss how to do that, we should examine how to verify that all the wiring to the switches is correct. We want to insure that all of the switches will open and close, and we want to determine if the wiring from the switches to the computer is complete or not.

The procedure for checking this involves using an ohmmeter or a continuity light. An ohmmeter is an instrument that will allow measurement of resistance in a circuit. In this application, a closed connection will be approximately zero ohms and an open connection will be (theoretically) infinite ohms. A continuity light is a light bulb that will turn on when the resistance is approximately zero ohms, indicating a closed connection. The light will remain off for an open connection. The test leads are placed across the two wires that are connected to any of the switches just installed. When the door or window is opened and then closed, the ohmmeter or continuity light will indicate this. That is, the ohmmeter or continuity light will visually inform you if the switch located at the door or window is opening and closing at the correct time. Figure 5.7 shows in pictorial form what we are accomplishing.

If the preceding verification is valid for every switch, we have assurance that everything is operational up to the point where the lines are connected to the computer. We will now show how these lines are input to the Apple computer. Figure 5.8 shows the electrical connections and the digital electronics necessary to input the signal lines to the computer.

Let us discuss this schematic diagram in detail. On the right-hand side of Figure 5.8 are the electrical input wires from all of the switches, labels W1–9 (the windows) and D1–4 (the doors). One side of each switch is connected to a particular input pin of

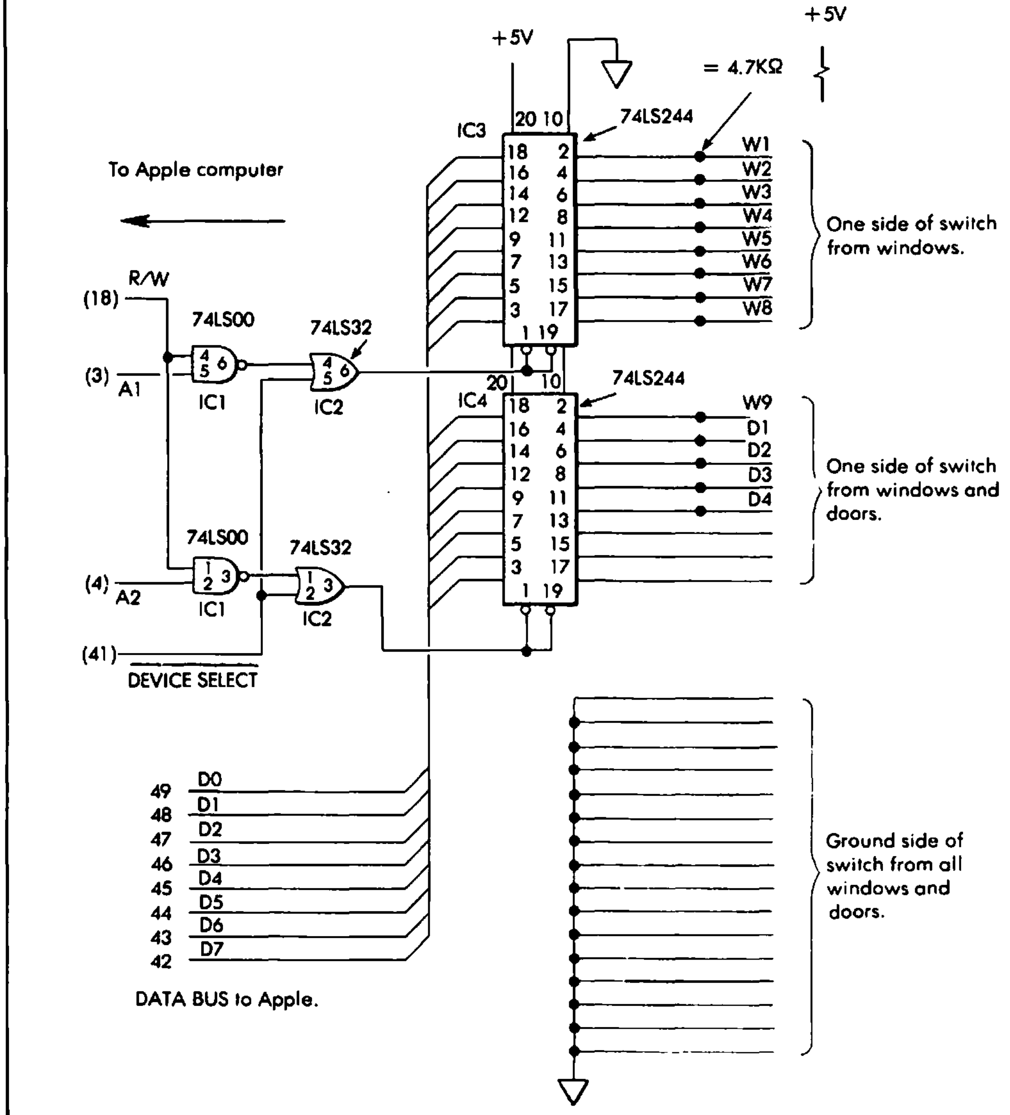


The physical and electrical connections to the switches can be verified by using an ohmmeter or a continuity light.

Figure 5.7

one of the two 74LS244 buffers, IC3 and IC4. Remember that this line is also connected to a 4.7k-ohm resistor, as shown in Figure 5.4. The other side of each switch wire is connected to the Apple computer ground.

Let us now concentrate on exactly how the "W" and "D" input lines are read by the Apple computer. In Chapter 4 we discussed the hardware necessary to allow data to be input to the Apple. We



Complete schematic for the I/O circuit to connect the Apple computer to the external switches on the doors and windows.

Figure 5.8

have essentially duplicated those circuits given in Chapter 4 twice for this application.

The data will be read into the Apple computer with the PEEK instruction. One main difference between this circuit and the circuit discussed in Chapter 4 is that two lines, labeled A1 and A2, have been added. The lines are input to 74LS00 NAND gates, IC1 of Figure 5.8. These two lines are called address lines. They allow us to have several addresses associated with each I/O slot in the Apple computer. In Chapter 4 these lines were not used, because at that time we only used one address per I/O slot. Now we have two buffers, and we need two addresses.

The new address for enabling IC3 of Figure 5.8 will be equal to:

PEEK address = I/O slot address + 2

The 2 sets address line A1 to a logical 1. To enable IC4 of Figure 5.8, the PEEK address will be equal to:

PEEK address = I/O slot address + 4

The 4 sets address line A2 to a logical 1. For example, if we want to read the logical level of the input lines connected to IC3 of Figure 5.8, we could do it as follows. (We will assume the circuit board is installed in I/O slot 5, so the I/O address is −16176.) In BASIC the instruction would be:

```
LET B1 = PEEK(-16176 + 2)
```

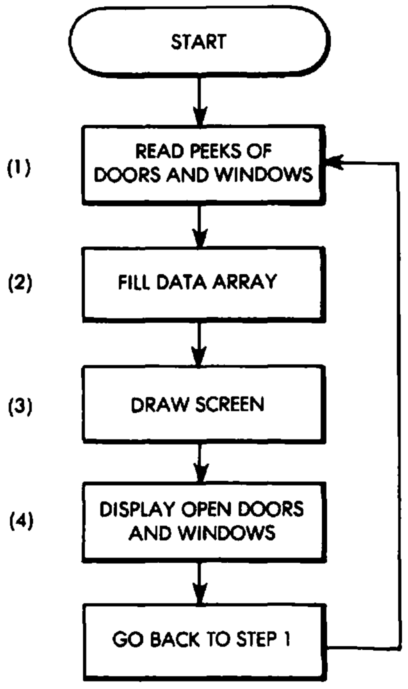
To read the logical levels of the input lines connected to IC4, we would use the BASIC statement:

```
LET B2 = PEEK(-16176 + 4)
```

At the conclusion of these two instructions, the variables B1 and B2 would be equal to the input weights associated with the open and closed position of each door and window. What must be done now is to write the BASIC language software that will examine the position of each door and window individually.

5.5: SOFTWARE FOR INTERPRETATION OF THE INPUT LINES

Now that we have an electrical means of entering the logical conditions of the windows and doors into the program, what do


START
(1)
READ PEEKS OF DOORS AND WINDOWS
(2)
FILL DATA ARRAY
(3)
DRAW SCREEN
(4)
DISPLAY OPEN DOORS AND WINDOWS
GO BACK TO STEP 1

Task flowchart for the major jobs that must be accomplished with the control software.

Figure 5.9

we do next? We must pause here and realize exactly where we stand. At this point in the design of our security system, we are able to input into a BASIC variable the logical condition of each switch in the home. That is, we are able to determine with software whether a window or door is open or closed.

From this point on, what is done with that particular information is totally dependent on the imagination. In our case we want to inform the user whether any particular window or door is open or closed. The point to be stressed at this time is that you are free to do with the data what you please. This fact leaves the territory wide open for applying any of the software graphics and clever tricks you may know how to do.

Figure 5.9 shows a block diagram flowchart of exactly what the software to be written next will do. In this flowchart the tasks shown are large ones, corresponding to the subroutines we will write. In the first block, the PEEK instruction is used to get the status of the digital signals connected to the windows and doors into the Apple computer.

In the next block an array called "DATA ARRAY" will be filled. Each element of this array will have a 1 or 0 in it, based on the logical value of the corresponding window or door that is being monitored. In our data array there will be 16 elements, D(1) to D(16). Each element will correspond to the logical input value of the signal line shown in Figure 5.8. For example, D(1) equals the logical value of the W1 input from the external window monitor, D(2) equals the logical value of the W2 input, D(3) equals the logical value of the W3 input, and so on.

Notice, in Figure 5.8, that not all of the 16 possible input lines are used. Therefore, not all of the D array values will be of use. This is acceptable because it is not necessary to examine all of the D array values.

Next in the flowchart of Figure 5.9, the software will draw the outline of the house on the screen. Finally, the software will write the status of each window and door to the screen.

These are the major software tasks that need to be accomplished in order to complete the design of our system. In this section we will discuss the software to implement the first two blocks of the flowchart in Figure 5.9. These two blocks will read the data at the I/O slot and then fill the data array with the proper 1s and 0s.

We have already seen in Section 4.7 how to read the data into the Apple computer from an external I/O slot. However, it might be a good idea to summarize the important points of that discussion here. The BASIC instruction used to input the data is PEEK. There will be two PEEK instructions necessary to read the entire 16 data lines that were shown in Figure 5.8.

The data read into the Apple computer from the input slot will be stored into an array. We will name the array "A," in order to allow for any future expansion of the number of input lines for the system. Array A will have only two entries at this time, A(1) and A(2).

A(1) will store the summed weight of the data lines from IC3 of Figure 5.8, and A(2) will store the summed weight of the data lines from IC4 of Figure 5.8.(These integrated circuits are 74LS244s, octal buffers with eight output lines each.) Recall that the summed weight can be any number from 0 to 255, inclusive.

You should also remember that the PEEK address is dependent on the address of the I/O slot into which the external card is installed. These addresses were given in Chapter 2, in Figure 2.7. In the program we are writing we will use the more general name "I/O add" to mean the number that corresponds to the I/O address of a particular slot in the Apple computer. For the program to actually work, of course, you would need to use a *real* address, such as −16192 for I/O slot 4, for example.

The two PEEK instructions used are:

```
LET A(1) = PEEK(I/O add + 2)
```

and

```
LET A(2) = PEEK(I/O add + 4)
```

These two instructions will store the summed weight of the 16 data input lines into the array A.

Data is now stored into the BASIC program. The next step will be to fill the data array D with the proper 1s and 0s. The list of the corresponding D array elements and the window or door each represents is shown in Figure 5.10.

The subroutine for filling the data array with the correct values is shown in Figure 5.11. We will first show the entire routine and then break each instruction down and provide further explanation where it is required.

In lines 500 and 510 the variable R2 will be set equal to the number of the A array element set by the value of T. This subroutine will be called for each element of the A array. In our program this subroutine will be called twice: once with T equal to 1, and another time with T equal to 2.

The instruction:

```
520  FOR I = F TO F + 7
```

will be the start of the FOR/NEXT loop. The variable F will be set prior to calling this subroutine, and will determine the starting element of the data array. Each time the subroutine is called, eight elements of the data array are filled. The first call or GOSUB will be with F equal to 1, the next GOSUB will be with F equal to 9.

Data Array Value		Window or Door
D(1)	=	W1
D(2)	=	W2
D(3)	=	W3
D(4)	=	W4
D(5)	=	W5
D(6)	=	W6
D(7)	=	W7
D(8)	=	W8
D(9)	=	W9
D(10)	=	D1
D(11)	=	D2
D(12)	=	D3
D(13)	=	D4
D(14)	=	NOT USED
D(15)	=	NOT USED
D(16)	=	NOT USED

List of the "D" (data) array elements and the corresponding "W" or "D" input line each one represents with software.

Figure 5.10

```
500  LET R1=128
510  LET R2=A(T)
520  FOR I=F TO F+7
530     IF R2-R1 < 0 THEN 560
540     LET R2=R2-R1
550     LET D(I)=1
560     LET R1=R1/2
570  NEXT I
580  RETURN
```

This subroutine will fill the data array.

Figure 5.11

```
530      IF R2 - R1 < 0 THEN 560
540      LET R2 = R2 - R1
550      LET D(I) = 1
```

The value of D(I) was originally set to zero. It will be changed by the weight of any D array element tested if the weight was used in the summation. This is exactly the same type of operation we described in Section 3.5 of this text. A program similar to this was shown in Figure 3.9.

```
560      LET R1 = R1/2
570      NEXT I
580   RETURN
```

Line 560 will compute a new value of R1, which will be the weight of the next lower data input line to test. R1 will start at 128, which is the weight of D7. R1 is set to 128 in line 500 of the subroutine.

The way in which the subroutine just described will be called by the main program is shown in Figure 5.12.

At this time the data has been input and the data array D is filled with the corresponding 1s and 0s for the logical inputs of the windows and doors being monitored by the computer.

5.6: SIMULATION OF ALL WINDOWS AND DOORS FOR PROGRAM DEVELOPMENT

Now we are ready to start writing the software to display the results on the screen. This type of software development is an empirical, trial-and-error process. That is, we make a first pass at what we think the output display should look like. Then, based on what we see, we adjust the program to make the output more visually pleasing.

When we are doing this type of program development, it would be useful to be able to simulate the opening and closing of any combination of doors and windows. Of course, we could simply go into the room where the door or window is and open or close it. However, the computer is usually located in a different spot than the doors and windows being monitored. It would be tedious and frustrating to have to get up from the computer each time we

```
 95  REM: READ WINDOWS AND DOORS (LINES 100—110)
100  LET A(1)=PEEK(I/O add + 2)
110  LET A(2)=PEEK(I/O add + 4)
115  REM: SET D ARRAY EQUAL TO ZERO (LINES 120—140)
120  FOR I = 1 TO 16
130    LET D(I)=0
140  NEXT I
150  LET T=1
160  LET F=1
170  GOSUB 500 REM: CALL THE SUBROUTINE
180  LET T=2
190  LET F=9
200  GOSUB 500 REM: CALL THE SUBROUTINE AGAIN
```

A section of the main program with the subroutine calls.

Figure 5.12

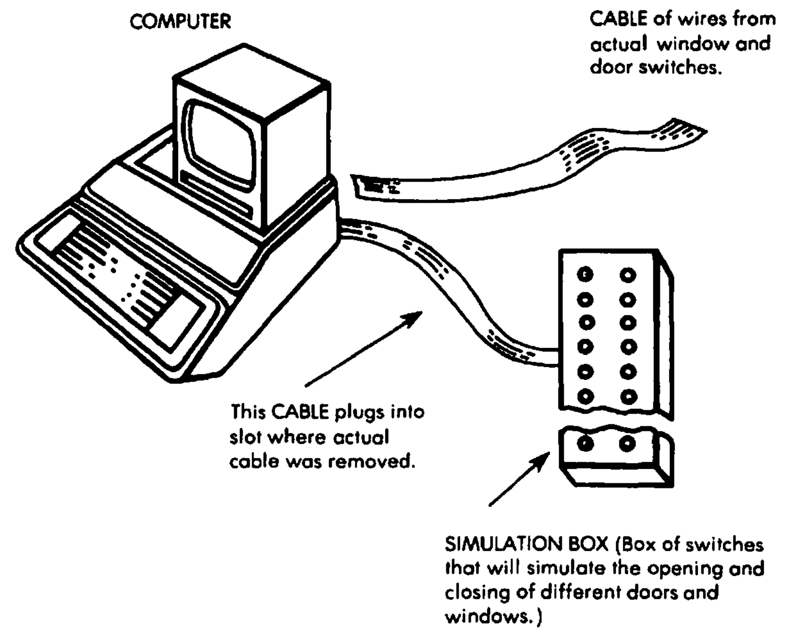
wanted to try a different combination of doors and window settings.

There are two possible ways to remedy this situation. One way involves hardware, and the other way involves software. The objective of each technique is to allow the data array, D, to be filled with any combination of 1s or 0s that we wish. This will simulate any combination of windows or doors being open or closed.

The hardware technique involves building a switch box that will connect to the I/O slot in place of the cable from the switches located in the doors and windows. This is shown in Figure 5.13. Using this technique, we can simply run the existing program and flip a switch on the box to change the status of any door or window being monitored.

The software technique for simulation can be as exotic as you wish. However, it should be kept in mind that this is a short-term effort. The program will not be used much, if at all, once the system is developed. Let this fact guide you in the amount of time spent on its creation.

The program we will show asks the user what the value of each element of the D array is to be. The user will enter the value 1 or 0


COMPUTER
CABLE of wires from actual window and door switches.
This CABLE plugs into slot where actual cable was removed.
SIMULATION BOX (Box of switches that will simulate the opening and closing of different doors and windows.)

The cable from the external window and door switches can be disconnected from the Apple computer and a simulator "switch" box can be connected in its place. This type of simulating hardware can be used for software development, and for system debugging.

Figure 5.13

for each element. In some cases the user will want to change only one of the array values. This program has the ability to do this also.

After the D array values are set, the program will jump directly to the section of the main program that will output the display. This program is appended to the main program during the software development stage. When the development is complete, the program may be deleted. This program is shown in Figure 5.14.

At line 200, the final GOTO, you would jump to the location in the main program where the system is outputting the screen information based on the values in the data array.

If you choose to perform the simulation using hardware, you can follow the schematic shown in Figure 5.15. We see in this

```
10   DIM D(16)
20   PRINT "DO YOU WANT TO CHANGE ALL OF THE VALUES"
30   INPUT G$
40   IF G$="Y" THEN 100
50   PRINT "ENTER THE DATA ARRAY NUMBER TO TOGGLE"
60   INPUT Y
70   IF D(Y)=1 THEN 85
75   LET D(Y)=1
80   GOTO 200
85   LET D(Y)=0
90   GOTO 200
100  FOR I=1 TO 16
110     PRINT "DO YOU WANT D(";I;") = 1"
120     INPUT G$
130     IF G$="Y" THEN 160
140     LET D(I)=0
150     GOTO 170
160     LET D(I)=1
170  NEXT I
200  REM: GOTO main program
```

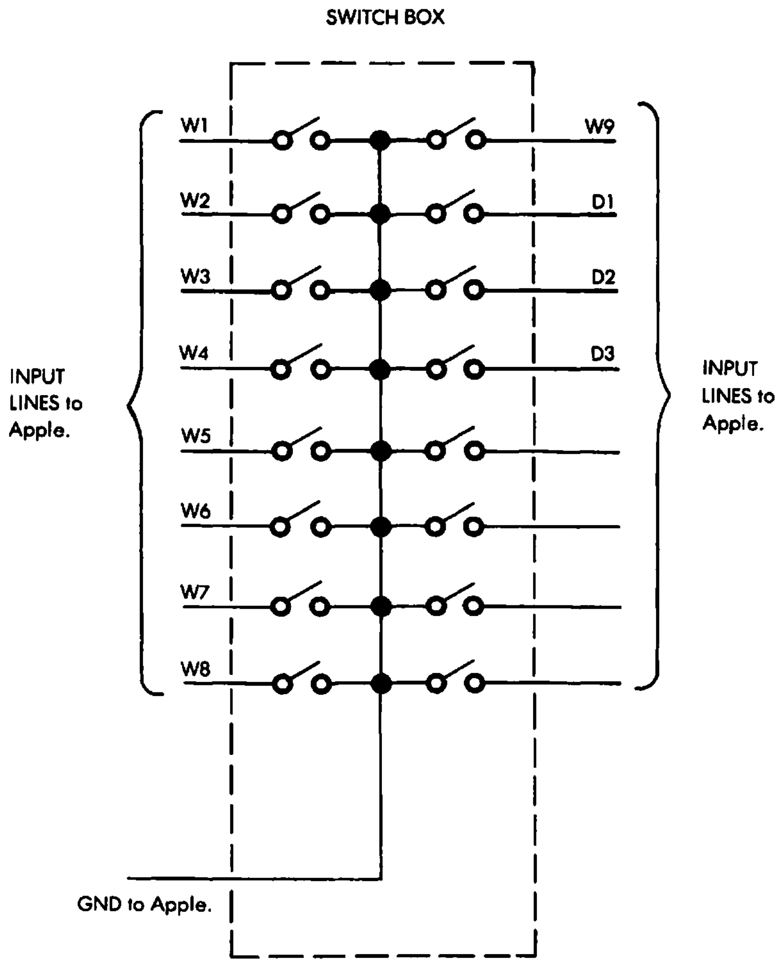
This routine will simulate the opening and closing of various doors and windows. When software development is completed, you can delete these lines.

Figure 5.14

diagram that each switch corresponds to one of the windows or doors being examined in the system. This type of simulation also has the advantage that it can be used as a debugging tool in the event your system becomes defective. The simulation box can be installed and used to verify that all of the interface hardware in the I/O slot is operational.

5.7: MASKING OFF THE ALARMS WITH SOFTWARE

Sometimes, you will not want the fact that a window or door is open to be the cause of an alarm. For example, when it is very hot, you may wish to leave a window or door open. It is possible to



Schematic diagram of a hardware switch box.

Figure 5.15

mask off any window or door that you do not wish to alarm. (If you design a more elaborate security system with a sound alarm, or even a connection to the police, the masking capability will be especially important.) In this section we will write the software to do that.

The program will ask the user which window number or door

```
 10  DIM M(16)
 15  REM: INITIALIZE THE MASK ARRAY TO ZERO (UNMASK ALL ALARMS)
 20  FOR I=1 TO 16
 30    LET M(I)=0
 40  NEXT I
 50  FOR I=1 TO 16
 60    PRINT "DO YOU WANT TO MASK OFF M(";I;")"
 70    ENTER A$
 80    IF A$="N" THEN 100
 90    M(I)=1
100  NEXT I
```

This routine will mask off any alarm you choose.

Figure 5.16

number to mask off. The results of these questions will reside in an array labeled M. This array will have as many elements as the data array we discussed earlier. If an element of the mask, or M, array is equal to a logical 0 (that is, if the user types "N"), then the alarm is not masked. If the entry is equal to a logical 1, then the alarm is masked. For example, if we wanted to mask the alarm on door D1, then M(9) = 1. The program is shown in Figure 5.16.

At the end of this program, all of the masks will be set. Remember that if the entry into the M array is a logical 1, then the mask is set. If the entry is a logical 0, then the mask is cleared. This array will be used when the computer generates the correct display on the Apple screen.

5.8: THE COMPLETE SYSTEM

So far in this chapter, we have separately presented most of the software and hardware pieces for the system. In this section we will bring all of these pieces together and show the software for the entire system. The software will be broken into functional parts. Each part will be nearly the same as a previous section we have presented. The new sections of the program will be explained fully. The program shown in Figure 5.21 comprises the software necessary to complete the entire system. First, let us look at the program so far, shown in Figure 5.17. Statements 10–620 will ask for masked alarms to be input, and then draw the outline of the house on the screen. Finally, the system will input the status of the windows and doors. This will be the print section to fill in the display outline for the screen.

```
  5  REM: LINES 10—150 WERE DESCRIBED IN SECTION 5.8
 10  DIM A(5), D(16), M(16), W$(32), P$(2)
 20  HOME
 25  REM: INITIALIZE THE MASK AND DATA ARRAYS (LINES 30—60)
 30  FOR I=1 TO 16
 40     LET D(I)=0
 50     LET M(I)=0
 60  NEXT I
 70  PRINT "DO YOU WANT TO MASK ANY WINDOWS OR DOORS ?"
 80  INPUT A$
 90  IF A$="N" THEN 200
100  FOR I=1 TO 16
110     PRINT "DO YOU WANT TO MASK M(";I;")";
120     INPUT A$
130     IF A$="N" THEN 150
140     LET M(I)=1
150  NEXT I
195  REM: LINES 200—430 WERE DESCRIBED IN SECTION 5.3
200  PRINT"----1   1---1   1-------1  1-------   ---"
210  PRINT"!                   !        !          !"
220  PRINT"!                   !        !          !"
230  PRINT"!                   !        !          !"
240  PRINT"!        !          !        !          !"
250  PRINT"----------          !        !          -"
260  PRINT"!        !          ------------         "
270  PRINT"-                              !         "
280  PRINT"                                        -"
290  PRINT"-                              !        !"
```

Figure 5.17

```
300  PRINT"!             --------------------             !"
310  PRINT"!             !       !      !      !          !"
320  PRINT"!             !       !      !      !          !"
330  PRINT"!             !       !      !      !          !"
340  PRINT"!             !       !      !      !          !"
350  PRINT"!             !       !      !      !          !"
360  PRINT"!             !       !      !      !          !"
370  PRINT"!             --1    1---1   1--1   1----      --"
380  PRINT"!             ! "
390  PRINT"!                              HOUSE     PLAN"
400  PRINT"!                          W=WINDOW,     D=DOOR"
410  PRINT"!
420  PRINT"!             ! "
430  PRINT"--1     1-----"
440  LET A(1)=PEEK(I/O add + 2 ) : REM GET THE STATUS (SECTION 5.6)
450  LET A(2)=PEEK ( I/O add + 4 ) : REM GET THE STATUS (SECTION 5.6)
460  LET T=1
470  LET F=1
480  GOSUB 530 : REM FILL DATA ARRAY (SECTION 5.6)
490  LET T=2
500  LET F=9
510  GOSUB 530 : REM FILL DATA ARRAY (SECTION 5.6)
520  GOTO 700 : REM END OF THIS FIRST SECTION OF CODE
525  REM: LINES 530-610 FILL THE DATA ARRAY (SECTION 5.6)
530  LET R1=128
540  LET R2=A(T)
550  FOR I= F TO F+7
560     IF R2-R1<0 THEN 610
570     R2=R2-R1
580     D(I)=1
590     R1=R1/2
610  NEXT I
620  RETURN
```

The program so far.

Figure 5.17 (cont.)

The following will be a new section of code. This section will compare the logical input data from the windows and doors against the mask data and, based on this comparison, will write different information to the screen. There are three possible conditions to write to the display:

1. ALARM. The display will blink the window or door number.
2. ALARM but MASKED. The display will show the number in inverse video.
3. NO ALARM. The display will show the window or door number without blinking or reverse video.

To set up this new section of code, several items of detail must be taken care of. First, we have not yet labeled the various doors and windows on our screen display of the outline of the house. We must now assign the letters and numbers that will define the label for each door or window. In our system we have used two letters for this purpose: windows are labeled W1, W2, etc., and doors are labeled D1, D2, etc. We need to incorporate these labels into a string variable, labeled W$.

(It should be stressed here that this method, using a string variable, is only one way to realize this function. Like many other solutions to particular problems offered in this book, it is meant to show guidelines for the reader who wants the simplest solution. After you have seen the problem solved one way, it will be easy to modify the solution to suit a particular application.)

The string variable W$ was dimensioned in line 10 of this program to a length of 32. This will give us room for 16 different labels of two characters each. We will use only 13 of the possible labels. Again, this may be expanded for any number of different labels.

W$ will equal:

```
W$="W1W2W3W4W5W6W7W8W9D1D2D3D4"
```

We can see from this variable that all of the labels are embedded in the single string W$. The software will extract the two-letter label as needed.

The next detail to be taken care of is the location of the various labels on the Apple screen. Using the standard Apple PRINT function, not graphics, the screen is composed of a grid 40 characters wide by 24 characters deep. This grid is shown in Figure 5.18. We must decide where in the grid each label will reside.

We stated earlier that this type of program development is usually done empirically. That is, we take a best guess, and then, based on how the display looks, we adjust the software according to which numbers must move. Each position on the screen grid will be defined by its x-y coordinates. Notice from Figure 5.18 how the grid is numbered.

After we decide what the x-y placement of the first character will be for each label, the information is entered into the program using data statements. For our program the data statement will appear as:

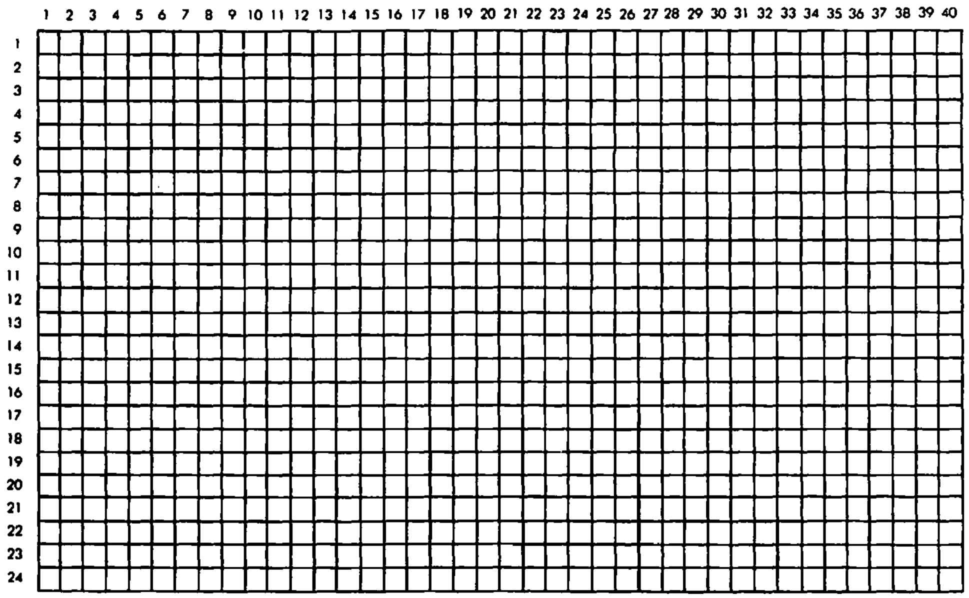
```
DATA (x,y,x,y,x,y,x,y)
```

Each x and y pair in the data statement will represent the x-y coordinates of a label to be printed on the screen. The printing starts with the label W1 and proceeds through all the labels shown in W$.

With this introduction, let us write the next section of code, shown in Figure 5.19. This section of code will examine the D and M arrays and decide if an alarm is present. There are three possibilities, as stated before. The variable shown as Q1 will be set to a 1, 2, or 3 depending on the alarm. Q1 will be equal to 1 if there is no alarm, to 2 if there is an alarm but it is masked, and to 3 if there is an alarm. H1 and V1 are the horizontal and vertical positions.

The following section of code, shown in Figure 5.20, will be the subroutine that will print the label of the door or window on the screen in the correct video format. The subroutine will assume that H1 and V1 are set to the correct horizontal and vertical positions on the screen grid. O1 will be the variable that determines the video format for the print. Finally, P$ will be set to the two letters of W$ that will be printed.

At the end of this subroutine the screen will be updated and the status of all windows will be shown. The program can then be run as often as desired. If you do not want to enter the alarm masks or draw the house outline each time the program is run, then start execution from line 440. The complete system program is shown in Figure 5.21.



Grid layout of the Apple screen. The screen on the Apple is 40 characters wide by 24 lines deep. (That is why there are 24 print statements in Figure 5.2.)

Figure 5.18

```
700  DATA 6,1,14,1,24,1,37,7,27,17,20,17,14,17,2,22,2,9
710  DATA 9,22,35,7,34,1,32,9
720  RESTORE
730  LET T=1
740  FOR I=1 TO 13 : REM: 13 IS THE MAX NUMBER OF LINES TO CHECK
750  READ H1,V1
760  IF D(I)=0 THEN Q1 =1
770  IF D(I)=1 AND M(I)=0 THEN Q1 =3
780  IF D(I)=1 AND M(I)=1 THEN Q1 =2
790  P$=MID$(W$,T,T+1) : REM: P$ IS A TEMPORARY VARIABLE EQUAL TO
```

Figure 5.19

```
 795  REM: THE LABEL OF THE DOOR OR WINDOW EXTRACTED FROM W$
 800  T=T+2
 810  GOSUB 900
 820  NEXT I
 830  STOP : REM CONTROL IS COMPLETE
```

This routine examines the D and M arrays to decide if an alarm is present.

Figure 5.19 (cont.)

```
 900  VTAB(V1)
 910  HTAB(H1)
 920  IF Q1=1 THEN NORMAL: PRINT P$
 930  IF Q1=2 THEN INVERSE: PRINT P$
 940  IF Q1=3 THEN FLASH: PRINT P$
 950  RETURN
2000  END
```

This section of code will print the labels of the doors and windows in the correct video format.

Figure 5.20

```
  10  DIM A(5),D(16),M(16),W$(32), P$(2), A$(5)
  15  W$ = "W1W2W3W4W5W6W7W8W9D1D2D3D4"
  20  HOME
  30  FOR I = 1 TO 16
  40     LET D(I) = 0
  50     LET M(I) = 0
  60  NEXT I
  65  GOTO 1000
  70  PRINT "DO YOU WANT TO MASK ANY DOORS"
  72  PRINT "OR WINDOWS"
  80  INPUT A$
  90  IF A$ = "N" THEN 200
 100  FOR I = 1 TO 16
 110     PRINT "DO YOU WANT TO MASK M(";I;")";
 120     INPUT A$
```

Figure 5.21

```
130     IF A$ = "N" THEN 150
140     LET M(I) = 1
150  NEXT I
200  HOME
205  PRINT"- - - - 1           1 - - - 1       1 - - - - - - 1      1 - - - - - -       - - -"
210  PRINT"!                                          !                  !                    !"
220  PRINT"!                                          !                  !                    !"
230  PRINT"!                                          !                  !                    !"
240  PRINT"!                     !                    !                  !                    !"
250  PRINT"- - - - - - - - - - - -                    !                  !                    -"
260  PRINT"!                     !                   - - - - - - - - - - - - -                 "
270  PRINT"-                                                                 !                 "
280  PRINT"                                                                                   -"
290  PRINT"-                                                                 !                !"
300  PRINT"!                         - - - - - - - - - - - - - - - - - - - - -                !"
310  PRINT"!                         !              !           !            !                !"
320  PRINT"!                         !              !           !            !                !"
330  PRINT"!                         !              !           !            !                !"
340  PRINT"!                         !              !           !            !                !"
350  PRINT"!                         !              !           !            !                !"
360  PRINT"!                         !              !           !            !                !"
370  PRINT"!                         - - 1      1 - - - 1      1 - - 1     1 - - - -       - -"
380  PRINT"!                         ! "
390  PRINT"!                                               HOUSE        PLAN"
400  PRINT"!                                      W=WINDOW,             D=DOOR "
410  PRINT"!
420  PRINT"!                         ! "
430  PRINT"- - 1           1 - - - - - -"
440  LET A(1)=PEEK(-16176 + 2) : REM I/O ADDRESS = SLOT 5
450  LET A(2)=PEEK(-16176 + 4)
460  LET T = 1
470  LET F = 1
480  GOSUB 530
490  LET T = 2
```

Figure 5.21 (cont.)

```
500  LET F = 9
510  GOSUB 530
520  GOTO 700
530  LET R1 = 128
540  LET R2 = A(T)
550  FOR I = F TO F + 7
560     IF R2 - R1 < 0 THEN 610
570     R2 = R2 - R1
580     D(I) = 1
590     R1 = R1 / 2
610  NEXT I
620  RETURN
700  DATA 7,1,15,1,26,1,37,9,29,18,23,18,15,18,4,22,2,10
710  DATA 9,21,36,18,35,2,30,10
720  RESTORE
730  LET T = 1
740  FOR I = 1 TO 13
750  READ H1,V1
760  IF D(I) = O THEN Q1 = 1
770  IF D(I) = 1 AND M(I) = O THEN Q1 = 3
780  IF D(I) = 1 AND M(I) = 1 THEN Q1 = 2
790     P$ = MID$ (W$,T,2)
800     T = T + 2
810  GOSUB 900
820  NEXT I
830  GOTO 400 : REM FINISHED WITH A SINGLE PASS
900  VTAB (V1)
910  HTAB (H1)
920  IF Q1 = 1 THEN NORMAL : PRINT P$
930  IF Q1 = 2 THEN INVERSE : PRINT P$
940  IF Q1 = 3 THEN FLASH : PRINT P$
950  NORMAL
960  RETURN
970  END
```

A complete software program for controlling the security system presented in this chapter.

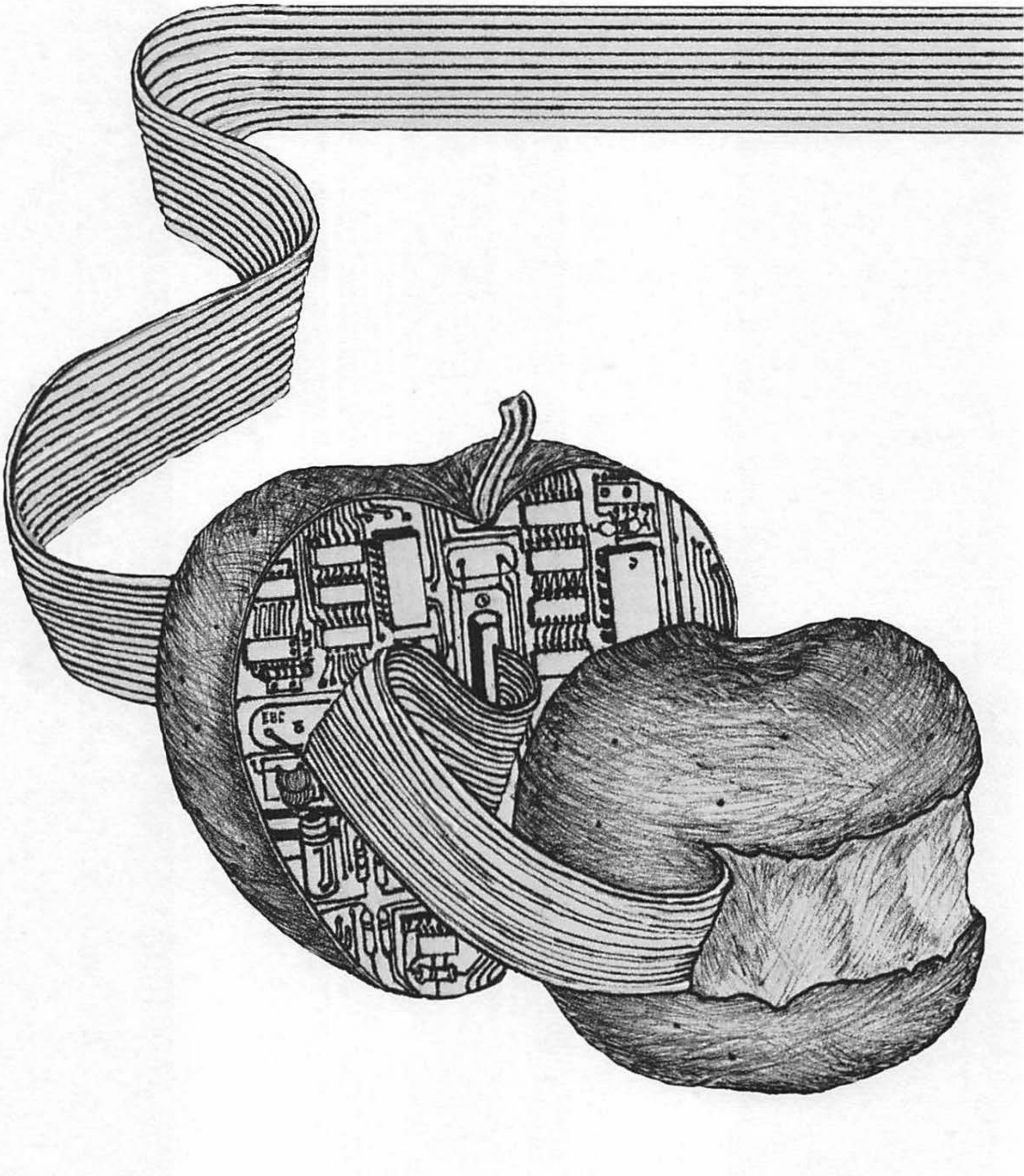
Figure 5.21 (cont.)

5.9: SUMMARY

In this chapter we have presented the complete design of a home security system using the Apple computer as the system controller. The chapter started with a clear definition of the design. From there we discussed the hardware necessary to realize the interface, and after the hardware was shown, the software was presented.

The details presented in this chapter are very typical of computer-controlled systems. That is, once the hardware interface is designed and connected, the problem becomes one of writing the software to control it.

We have also presented some guidelines for program development. This is useful when the external hardware is cumbersome to use or is not complete. The system shown in this chapter was designed to illustrate general principles, and its primary use is for instruction. However, if you can understand the major parts of this system, then construction of another, more elaborate system will be a much easier task.



Chapter 6

Interfacing the Apple to Home Appliances

WE HAVE LEARNED how to connect the Apple computer so as to monitor a system of switches and display their output on its screen. These switches were electrically connected directly to the Apple, and needed no other power supply in order to operate. But you may have imagined applications that require the Apple to turn on and off some home appliance that draws power from a wall socket. Can this be done? Yes. The Apple (or any other computer) is certainly capable of performing this task. However, the task cannot be performed directly. There must be some intermediate hardware used that allows the computer to turn on and turn off the appliance. The purpose of this chapter is to show how the computer can be interfaced so as to accomplish this control. It may come as a surprise to some that the job is not difficult. By using the information given in this chapter, you will be able to turn on and off most 120/220-volt AC appliances in the home.

Note—For the only time in this book, we are discussing systems that involve connections to a standard 120/220-volt AC wall socket. Working with AC line voltages requires care and prudence. Errors made here can have much more serious consequences than errors made in executing the other designs shown in this book.

6.1: BLOCK DIAGRAM OF THE PROBLEM

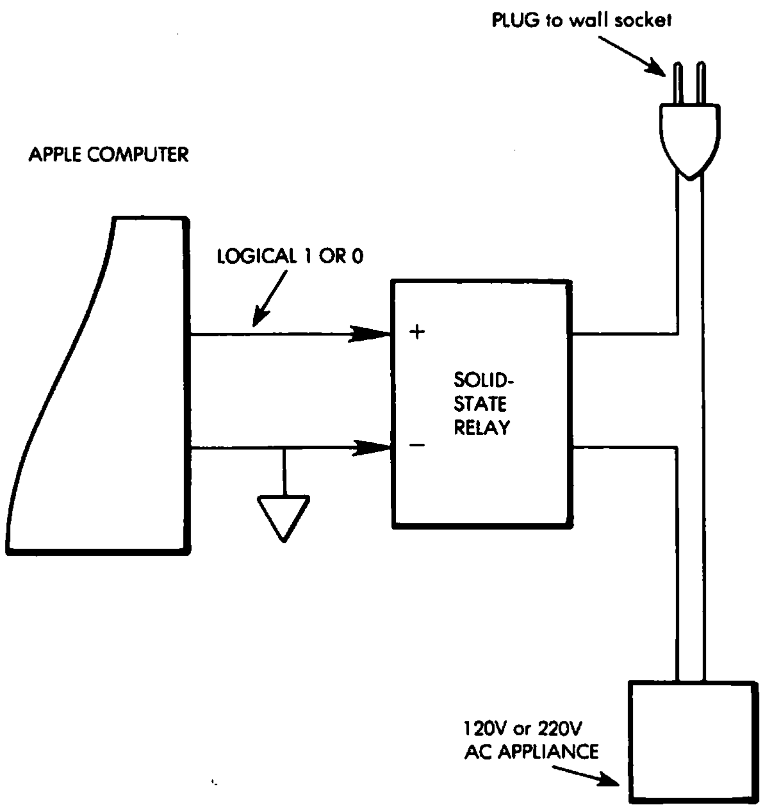
We have stated that the computer can be used to turn on and off home appliances by means of external hardware. The additional hardware needed will be explained in block diagram form in this section. We begin with a block diagram so you may see the overall outline of the problem before studying the details.

Figure 6.1 shows a block diagram of a system that allows the computer to turn on and off the AC appliances in your home. Notice, in this diagram, that the Apple computer will output a logical 1 or a logical 0 voltage. We have already discussed what was meant by these voltage levels, in Chapters 2 and 4 of this text. In review, a logical 0 voltage for the Apple computer (and for most home and personal computers) will be approximately 0.0 volts, and a logical 1 voltage will be approximately 5.0 volts. These two voltages are what the Apple computer is capable of outputting at the I/O slots. For the Apple computer to directly control any external piece of hardware, that device must electrically "accept" either 5.0 volts or 0.0 volts. That is, these voltages must control the flow of the much higher voltages the appliances actually use. To do this, another, intermediate hardware device is necessary.

In Figure 6.1 the logical 1 or logical 0 output from the Apple computer is input to a block labeled "solid state relay." The solid-state relay (SSR) transforms the +5 or 0.0 volts output from the Apple computer in a manner that will control an appliance that requires 120/220 volts.

This is not a new concept. It is very similar to the low current in an automobile ignition system that closes a starter relay, allowing many amperes of current to flow into the starter motor. The starter switch controls the smaller current flowing into the starter relay, and the starter relay controls the high current. This concept is illustrated in Figure 6.2. We can think of the Apple computer as the starter switch controlling the solid-state relay, which is analogous to the starter relay.

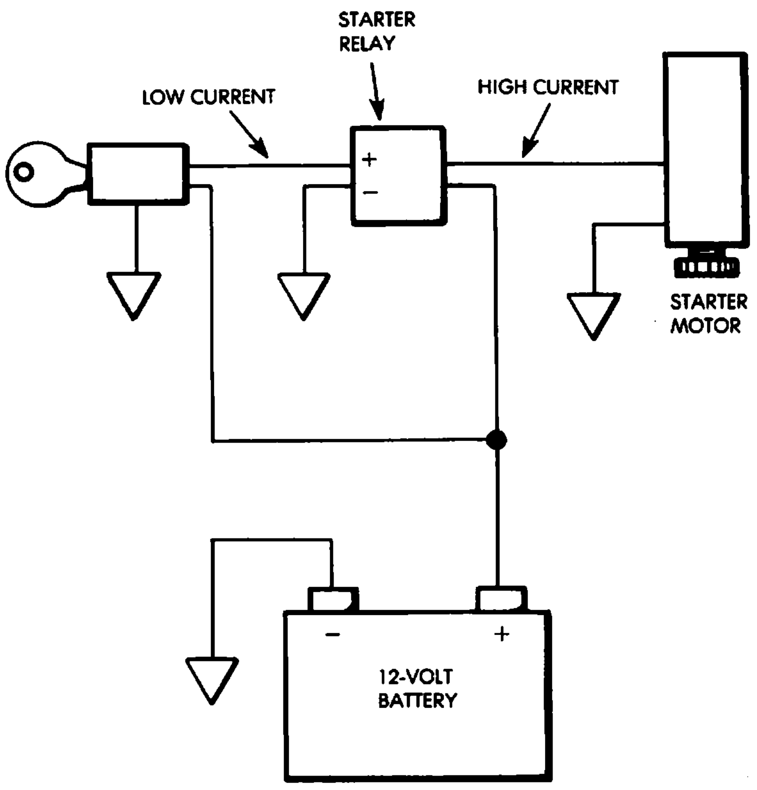
In Figure 6.1 the Apple computer outputs a low control voltage, a logical 1 or a logical 0 that allows the solid-state relay to control the higher voltage required for home appliances. The solid-state relay performs essentially the same function as a switch. When



Block diagram showing the Apple computer controlling a solid-state relay, which in turn controls the AC appliance. Therefore, the Apple computer controls the AC appliance.

Figure 6.1

current is output from an Apple computer I/O circuit to the solid-state relay, the switch is closed, and when current is not output, the switch is open. When the switch is open, no AC current will flow, and the appliance is turned off. When the switch is closed, the standard house current (120/220 volts) will flow, and the appliance is turned on. In effect, when current flows from the

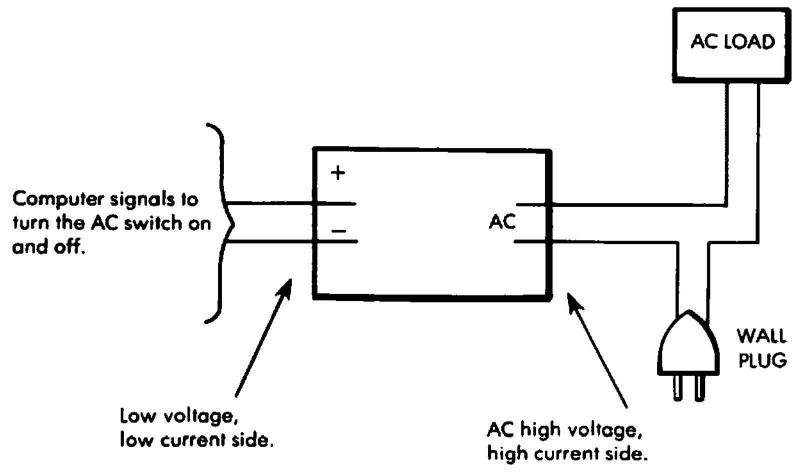


A common example of a low-current switch controlling a high-current switch is the starter system in an automobile. The low-current switch is the starter switch (or ignition); the high-current switch is the starter relay.

Figure 6.2

Apple computer to the solid-state relay, the appliance is turned on. When current does not flow from the Apple computer to the solid-state relay, the appliance is turned off.

Ultimately, our software problem, after the hardware is all connected, will be to generate a logical 1 or a logical 0 at an I/O slot in the Apple computer. This is exactly the same problem we solved in Chapter 2 of this text. First, let us examine the hardware, beginning with the solid-state relay.


AC LOAD
+
–
AC
Computer signals to turn the AC switch on and off.
WALL PLUG
Low voltage, low current side.
AC high voltage, high current side.

Block diagram of the typical external connections on a solid-state relay. One side of the relay is for the low-voltage control lines. The other side is for connecting the AC load.

Figure 6.3

6.2: HOW THE SOLID-STATE RELAY OPERATES

Without going into more detail than necessary, we will explain how the solid-state relay operates. This discussion is undertaken so you will have a user's knowledge of the operation of solid-state relays. Figure 6.3 shows a block diagram of the solid-state relay.

We see in Figure 6.3 that one side of the solid-state relay is used for the AC connection. This will be connected in the same way as a switch: one line to the AC source, the wall socket, and one line to the AC load, the appliance. The other side of the solid-state relay is used for the low-voltage input control lines. A data sheet for a typical solid-state relay is shown in Figure 6.4.

Let us focus our attention on the low-voltage side of the relay diagram shown in Figure 6.3. Figure 6.5 shows a pictorial diagram of what is internal to the solid-state relay on the low-voltage

Electrical Specifications (25°C unless otherwise specified)

INPUT CHARACTERISTICS	DC INPUT MODELS (with "D" Prefix)
Control Voltage Range	3 to 32 VDC
Max. Reverse Voltage	-32 VDC
Max. Turn-On Voltage (-30°C ≤ TA ≤ 80°C)	3.0 VDC
Min. Turn-Off Voltage (-30°C ≤ TA ≤ 80°C)	1.0 VDC
Min. Input Impedance	1500 Ohms
Max. Input Current	4mA (@5 VDC)
	20mA (@28 VDC)
Max. Turn-On Time (60 Hz)	8.3 msec
Max. Turn-Off Time (60 Hz)	8.3 msec

OUTPUT CHARACTERISTICS			240 VAC			
MODEL NUMBERS — AC Control / DC Control	A2402 D2402	A2410 D2410	A2425 D2425	A2440 D2440	- D2475	Units
Operating Voltage Range 47-63 Hz			48-280			V(RMS)
Max. Load Current (See derating curves)	2.5	10	25	40	75	A(RMS)
Min. Load Current			50			mA(RMS)
Transient Overvoltage			500			V(peak)
Max. Surge Current (Non-Repetitive) 16.6 ms (See surge curves)	22.5	120	250	625	1000	A(peak)
Max. Over Current (Non-Repetitive) 1 sec.	5	22	40	80	150	A(RMS)
Max. On-State Voltage Drop @ Rated Current	3.5	1.6	1.6	1.6	1.8	V(peak)
Max. I^2T for Fusing (8.3 ms)	2.1	60	260	1620	4150	A^2sec
Thermal Resistance, Junction-to-Case, $R_{\theta JC}$ (T_J Max. = 115°C)	8.5	1.48	1.02	0.63	0.18	°C/W
Power Dissipation @ Max. Current (See dissipation curves)	6.3	12	29	46	82	Watts
Max. Zero Voltage Turn-on			30			V(peak)
Max. Peak Repetitive Turn-on Voltage			12			V(peak)
Max. Off-State Leakage Current @ Rated Voltage (-30°C ≤ TA ≤ 80°C)			10			mA(RMS)
Min. Off-State dv/dt (Static) @ Max. Rated Voltage			200			V/µS

GENERAL CHARACTERISTICS—ALL MODELS

Dielectric Strength 50/60 Hz:	2500 $V_{(RMS)}$
Insulation Resistance @ 500 VDC:	10^{10} Ohms
Max. Capacitance Input/Output:	8 pf
Ambient Temperature Range: Operating:	-30°C to 80°C
Storage:	-40°C to 120°C

All models are U.L. recognized and CSA certified except D2475, D4840 and D4875 which are pending.

Photograph of a typical solid-state relay, with relevant sections of the manufacturer's data sheet. (Courtesy of International Rectifier, Crydom Division of IR Corporation.)

Figure 6.4

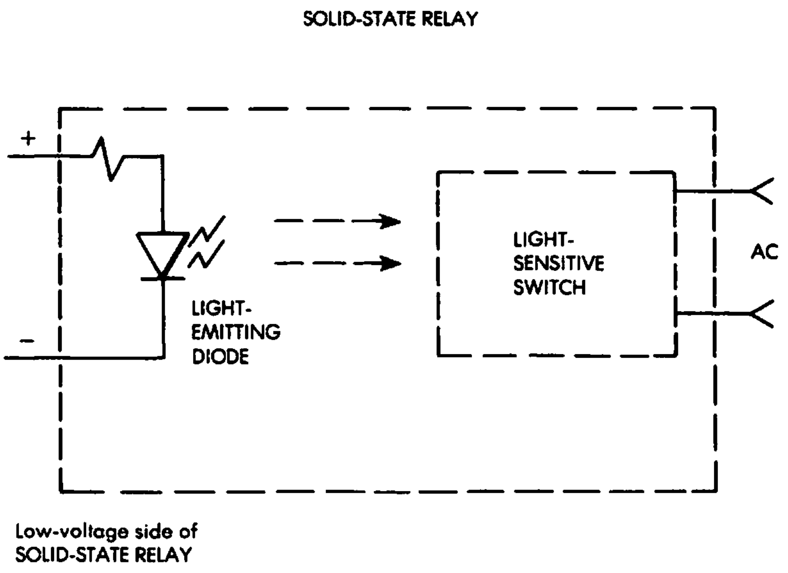


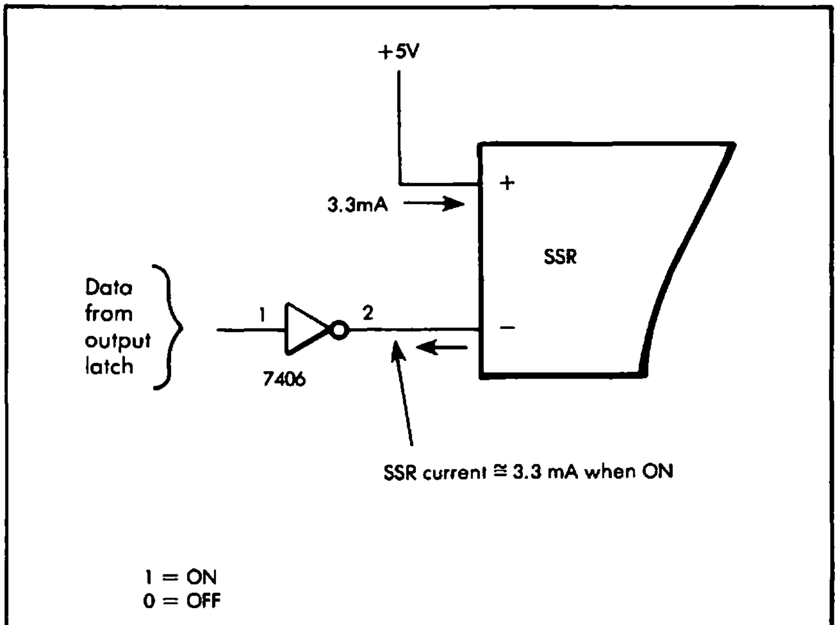
Diagram showing what is inside the solid-state relay package. The low-voltage side is internally connected to an LED. The AC side is connected to a light-sensitive switch.

Figure 6.5

side. In Figure 6.5 the input terminals of the SSR on the low-voltage side are connected to a light-emitting diode (LED).

One side of the SSR input is labeled "+", and is connected internally to the anode of the LED. This side will be externally connected to +5 volts, a convenient voltage output from the Apple computer. The specifications shown in Figure 6.4 give a "control voltage range" from 3 to 32 volts on the positive input terminal, so the Apple's output voltage is acceptable.

The negative (−) low-voltage input to the SSR shown in Figure 6.5 is connected internally to the cathode of the light-emitting diode. When the negative terminal of the circuit is connected to ground potential, the internal light-emitting diode shown in Figure 6.5 will light and emit photons. The photons will strike an internal light-sensitive switch. This is the switch that will actually conduct the AC current. While the photons are striking the light-sensitive



Logic circuit for turning the solid-state relay on and off.

Figure 6.6

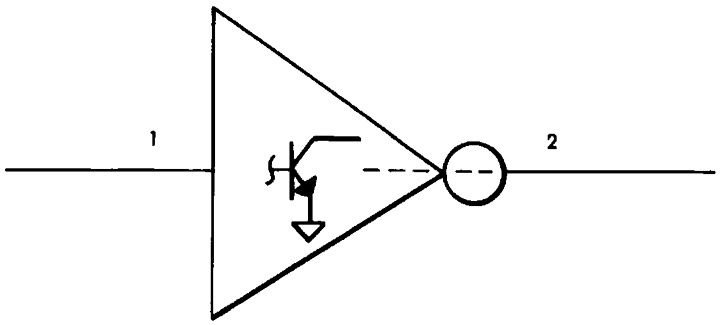
switch within the SSR, the switch will stay closed. During this time the AC current will flow through the SSR.

When the low current ceases to flow through the light-emitting diode of the SSR, no more photons will be emitted, and the internal light-sensitive switch will open. At this time there will be no AC current flow.

From this brief discussion of how the SSR operates, it can be seen that there is no direct electrical connection between the Apple computer and the AC line voltage. In effect, the computer system is "optically isolated" from the AC voltage. That is, the two are electrically isolated, and optically coupled by the LED and light-sensitive switch.

6.3: ELECTRICALLY CLOSING THE SOLID-STATE RELAY

We have stated previously that the low-voltage input to the SSR is the controlling side of the relay. It will now be shown how to


1
2
The output pin of a 7406 is the collector of a transistor. The emitter is connected to ground internally.

Pictorial diagram showing what is meant by an "open collector" integrated circuit. The output pin of the device is simply the collector of a transistor. The emitter of the transistor is connected to ground.

Figure 6.7

electrically turn the relay on and off under the control of a logical 1 voltage or a logical 0 voltage.

Figure 6.6 shows the circuit used to turn on and off the SSR. In this diagram a logical 1 or logical 0 output from an I/O circuit (like that described in Chapter 4) in the Apple computer is input to the 7406 inverter. The output of the 7406 will appear as shown in Figure 6.7. When a logical 1 is input to pin 1 of the 7406, output pin 2 will essentially connect to ground potential.

With the output of the 7406 at ground potential, there is a current path established from the SSR's positive terminal through the internal light-emitting diode to ground. At this time the SSR will turn on, for the reasons we discussed in Section 6.2. The current required to turn on the AC switch is approximately 3.3 milliamperes, or .0033 amperes. This is calculated from the input voltage, 5 volts, divided by the input resistance, 1500 ohms. The specification for the input resistance is given in Figure 6.4.

During the time a logical 1 is input to the 7406 shown in Figure 6.6, the AC appliance will be powered by current from the wall socket through the SSR. When the input pin 1 of the 7406 shown in Figure 6.6 is a logical 0, the output pin 2 will open and there will be no physical connection to ground. The SSR will be off, causing the appliance to turn off.

Our problem now is to output a logical 1 or a logical 0 to the input pin 1 of the 7406 from the Apple computer. This can be accomplished with the circuit shown in Figure 6.8. Figure 6.8 is very similar to the circuit discussed in Chapter 4, and can be compared to Figure 4.6.

6.4: AN OUTPUT PORT

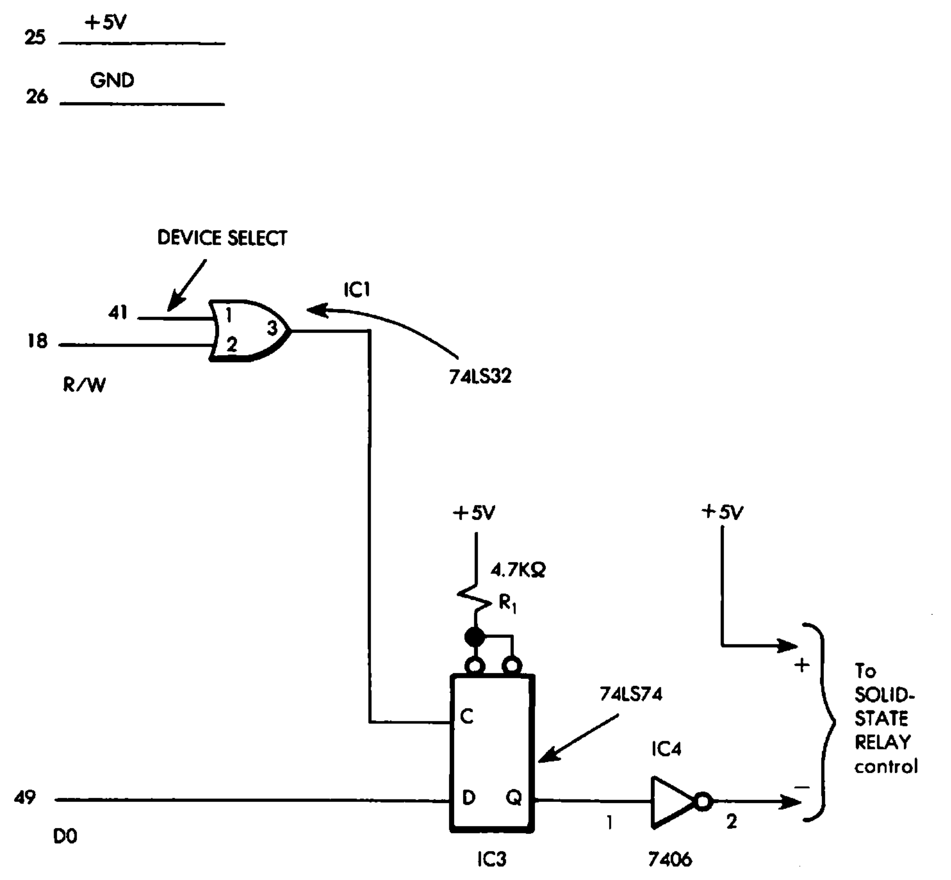
For review we will discuss how the circuit shown in Figure 6.8 will operate. In this diagram the temporary storage device is a 74LS74 single-bit latch (or flip-flop), IC3. (This single-bit latch is identical to the 74LS175 presented in Chapter 4, with the exception that only one bit of data can be stored at a time. The 74LS175 permitted four bits of data storage at the same time.) The data input line to the latch is connected to the D0 data line of the Apple computer. This means that when we want the Q output of the latch to be a logical 1, the D0 line of the Apple computer will be a logical 1 during the POKE instruction. (The Q output is a label given to the signal line of the latch that represents the output pin.) If we want the Q output to be a logical 0, the D0 line will be a logical 0 during the POKE.

The clock input to the 74LS74 flip-flop is generated by the $\overline{\text{DEVICE SELECT}}$ line and the R/W output from the Apple. The logical conditions that will enable data to be written to the flip-flop are the following:

1. $\overline{\text{DEVICE SELECT}}$ = logical 0.
2. R/W = logical 0.

The $\overline{\text{DEVICE SELECT}}$ line will go to a logical 0 when the correct address is specified in the POKE instruction. The R/W line will go to a logical 0 during a POKE instruction, because the computer is writing data.

When the Q output of the flip-flop is a logical 1, the output pin 2 of the 7406 will be on. If the Q output of the flip-flop is a logical 0, the 7406 output will be off. The 7406 will operate as we discussed



Schematic diagram of a complete I/O circuit for the Apple computer for controlling the solid-state relay. The low-voltage side of the SSR will be connected to + 5 volts and the output pin 2 of the 7406.

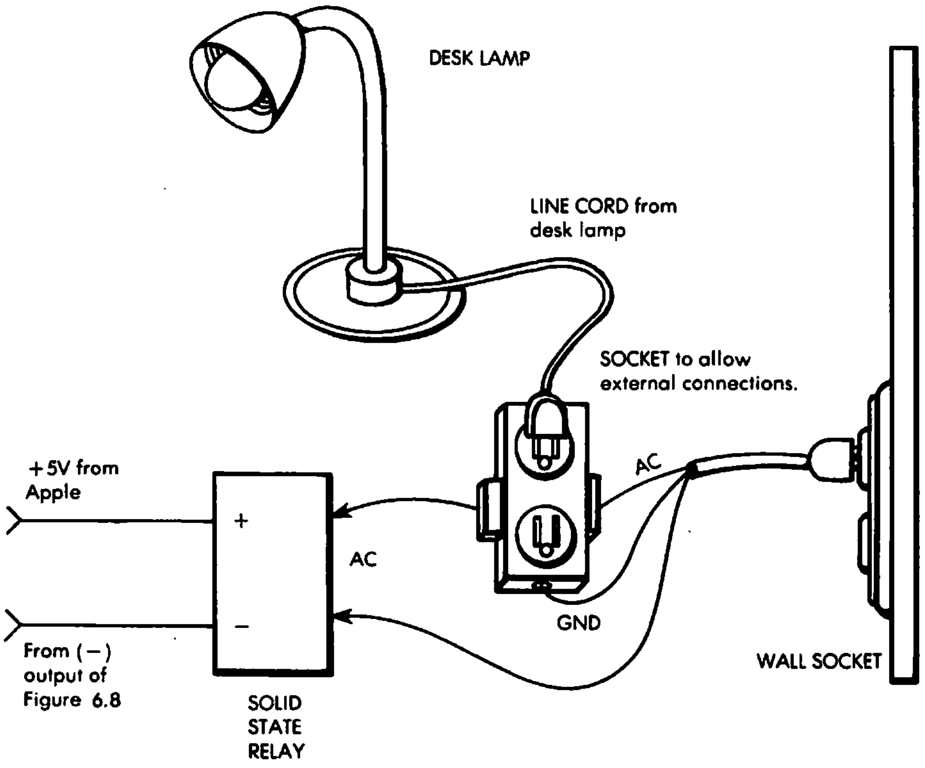
Figure 6.8

in section 6.3.

From this discussion we see that the hardware will respond to the following two POKE instructions:

POKE address, 1
POKE address, 0

where "address" will equal the correct address number of the I/O



Circuit diagram showing how to connect a desk lamp, line cord from the wall socket, and the SSR together, for computer control of the lamp.

Figure 6.9

slot. A 1 in the POKE instruction will set the output of the 7406 to a logical 0. A 0 in the POKE instruction will force the output pin 2 of the 7406 open.

6.5: SOFTWARE FOR CONTROL

Let us now discuss how to control the SSR with software. Once the hardware is connected, the problem is reduced to one of writing software. We will assume that a desk lamp with a standard 100-watt light bulb is connected to the SSR. This will be an easy example to try in your own home. The circuit diagram for this connection is shown in Figure 6.9.

The maximum AC current the SSR can handle is specified in Figure 6.4. For the SSR we are using, the maximum AC current is 10 amperes. In our example, the 100-watt light will require less than one ampere. The current can be calculated as follows:

$$\text{light current} = \frac{100 \text{ watts}}{120 \text{ volts}} = .833 \text{ amperes}$$

The solid-state relay can handle this current easily.

The software will blink the light on and off. More extensive software could be written to turn on any of your lights at regular time intervals if you were not at home. We give this software only as an example.

It is assumed that the I/O slot used in the Apple computer is slot 5. This slot requires a system address of −16176 for the POKE instruction. In the interface hardware shown in Figure 6.8, the data bit D0 is the computer output data line that controls the switching of the SSR. When we wish the SSR to turn on, a 1 will be POKED to address −16176, and when we wish the SSR to turn off, a 0 is POKED to address −16176.

An example of a program to turn the light on and off under control of the Apple computer is presented in Figure 6.10.

```
 10  REM: TURN THE LAMP OFF
 20  POKE -16176,0
 30  REM: DELAY FOR A WHILE
 40  FOR I= 1 TO 1000
 50  NEXT I
 60  REM: TURN THE LIGHT ON
 70  POKE -16176,1
 80  REM: DELAY FOR A WHILE
 90  FOR I=1 TO 1000
100  NEXT I
110  GOTO 20
```

This short program will turn a desk lamp on and off.

Figure 6.10

In the preceding BASIC program, the lamp will be turned on and off with a delay time set by the FOR/NEXT loops. If you wish to change the length of time the lights are on or off, then decrease or increase the number 1000 in lines 40 and 90.

6.6: SUMMARY

In this chapter we have discussed how to connect the Apple computer to control 120/220-volt AC appliances, using a solid-state relay, or SSR. The problem was presented in block diagram form. Next, an explanation of how the SSR operates was given. It was then shown how to electrically connect the Apple to the SSR. Finally, a sample BASIC program to turn on and off a desk lamp was given.

It is highly recommended that you try out the use of an SSR on a small scale first. That is, try out the desk lamp example that was given in this chapter. This will give you the opportunity to experiment. After that initial experiment you will be able to use the SSR in some larger-scale applications around the home. For instance, you might try some of the following home applications, which would use a solid-state relay interfaced to the computer:

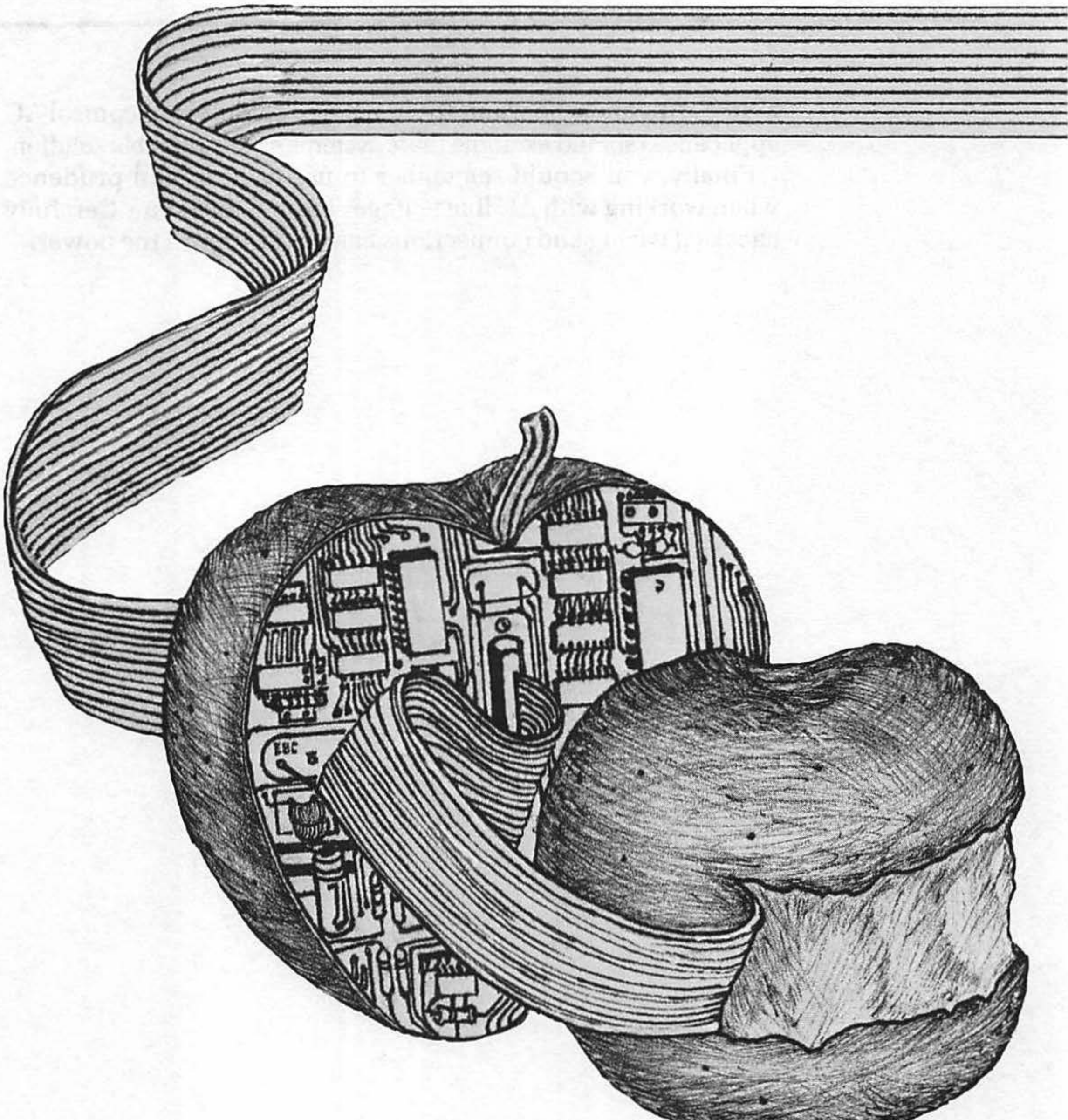
1. Turn on the stereo. You could let the Apple computer turn on the stereo at different programmed times when you are not at home.
2. Your television could be turned on at different times by the computer when you are not at home.
3. Outside lights around your home could be turned on when the computer detects dusk and turned off when the computer detects morning light.

These are just a few of the many applications that can be accomplished using the computer and a solid-state relay. Virtually any appliance that uses 120/220 volts AC can be controlled automatically with the computer using the techniques described in this chapter.

It should be pointed out that, besides the kind of system we have described, there are also "off-the-shelf" AC control systems available for the home computer. An SSR may or may not be used in these systems. One very popular AC control system is the BSR

X-10™. Anyone who wants to use a home computer to control AC applicances should examine these systems as one possible solution.

Finally, you should remember to use caution and prudence when working with AC line voltages around the home. Carefully check *all* wiring and connections before turning on the power.



Chapter 7

Analog vs. Digital, and Transducers

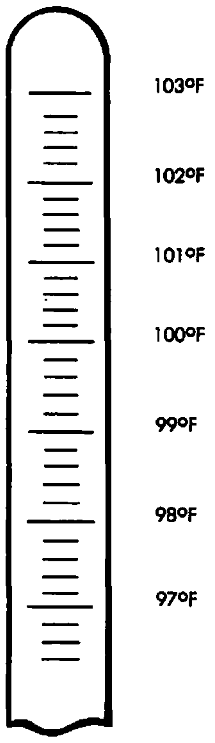
IN THIS CHAPTER we will examine two concepts: 1) the difference between analog and digital events, and 2) the transducer. This discussion is intended to give readers not familiar with these concepts a good understanding of them. If you are already familiar with the difference between analog and digital events, and the related concept of the transducer, you may want to skip this chapter.

However, it is important for anyone who wishes to learn about computer control to understand the difference between analog and digital events and the importance of transducers. As we will see, external devices are mostly analog in nature, and some conversion is needed before they can operate under digital control.

7.1: ANALOG EVENTS

Briefly, an analog event is one whose output is variable. The following examples will illustrate this idea and demonstrate that we live in an analog world.

Nearly everyone is familiar with the common mercury thermometer, which, of course, measures temperature. A typical



Typical thermometer showing the temperature scale in degrees Fahrenheit. The reading can be anywhere on the scale shown; there are an infinite number of potential output values on this continuous scale.

Figure 7.1

thermometer scale is shown in Figure 7.1. As we see, the temperature reading may vary anywhere on the scale: 98.6, 98.4, 98.2, 97.1, 100.3, 100.6, etc.

The thermometer is an analog device because its output (the temperature reading) can have virtually any value. The range of values is limited by the scale on the thermometer (97–103° F), but within these limits there are an infinite number of possible readings. That is, the range is *continuous.* The point is that the temperature reading, like any analog output, can be *anywhere* between these limits.

A point of confusion may arise at this time. The term "digital" has been applied to thermometers that display numbers (digits) and use digital electronics in their display circuitry. In that sense (now the most common use of the term), they are indeed "digital" devices. As measuring instruments, however, these thermometers are analog devices, just as mercury thermometers are, because temperature is an analog phenomenon.

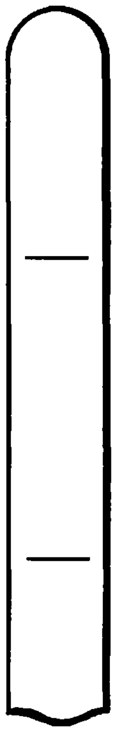
Sense perceptions may be thought of as analog events; consider the feeling of hunger. There are varying degrees of hunger, which seem to merge into each other. We can be full, almost full, a little bit hungry, very hungry, famished, or any number of different states in between, but we cannot tell where one state ends and another begins. Because the range of sensations is apparently continuous, hunger can be considered an analog event.

7.2: DIGITAL EVENTS

Digital events are not continuous but discrete. Their outputs are not infinitely variable within a range, but instead are limited to a finite number of predefined states. To illustrate this idea, let us re-examine our examples of analog events and consider what they would be like as digital events. In the two events we will describe, the number of discrete output values will be limited to two. We could have specified more values, but we have chosen only two, because that is the number of states possible in the binary logic used in digital electronics.

First, let's examine the thermometer. If this device were limited in the way we have described, it would have only two temperature output readings. The scale would appear as it does in Figure 7.2, where the temperature reading is either 70° F or 80° F. These two numbers were chosen at random to illustrate the point. A thermometer such as this would be quite useless to us, because temperature is an analog event. It cannot be measured using a digital scale, because it varies. It would be a strange world indeed if there were only two possible values for temperature.

Now let's examine hunger as if it were a digital event, again limited to the two possible output values we have specified. You would either feel full or hungry. This might not seem so bad, but the change from one to the other would be immediate, and it



A thermometer with a binary, digital scale. The temperature reading can only be 70° F or 80° F.

Figure 7.2

would be absolute. One second you would feel "stuffed," the next you would feel "famished." Fortunately, hunger is an analog event and actually varies in imperceptible stages.

The preceding examples were intended to show that a digital event has discrete output values. The contrast between digital and analog phenomena can also be illustrated by comparing a staircase and a ramp. The staircase represents digital phenomena, and the ramp represents analog. On a staircase we must ascend or descend by fixed, predefined steps. On a ramp we can move any amount we desire. The closer the stairs are to each other, the smaller the steps we proceed by, and the closer we can come to some desired level. However, as the *size* of the steps decreases, the *number* of steps required to reach the same level must increase. If there were an infinite number of stairs in the staircase, we would have an analog event again. We will return to this analogy in Chapter 9 (see Figure 9.7).

We have tried to show readers who may be unfamiliar with the concepts *analog* and *digital* what the difference between them is. For these readers we will restate our definition of the terms. In an analog event, there are an infinite number of possibile output values. In a digital event, there are a finite, limited number of possible output values. In our examples we used two possible output values, but we could have used more.

7.3: TOTALLY DIGITAL EVENTS

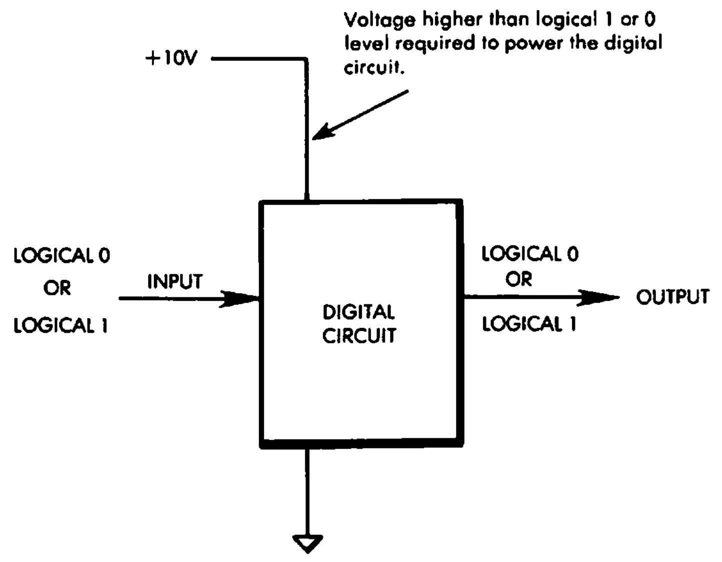
Although most events in the real world are analog, some common digital events occur. Let's examine some.

One of the most common digital events is the turning on and off of a light. The light output has only two possible values: totally on, or totally off. (Of course, if you have dimmer switches installed in your home, the light output will vary, because you are using an analog device, a rheostat.)

A home furnace fan in a forced-air heating system is another digital event. Either the fan is on, running at a fixed, constant RPM, or it is off, not spinning at all. With a thermostat (another analog device) we can vary the temperature at which the fan will turn on. However, when it does turn on it will always run at the same speed. A further example of a digital, binary device is the ON-OFF switch used in television sets and home computers. The switch has two positions, ON or OFF. Television tuners and fans with more than one speed are digital, but not binary. There are a finite number of possible states, but more than two.

7.4: ANALOG AND DIGITAL ELECTRONICS

So far in this chapter, we have tried to illustrate the difference between analog and digital events with examples from everyday life. Let us now examine what the terms *analog* and *digital* mean

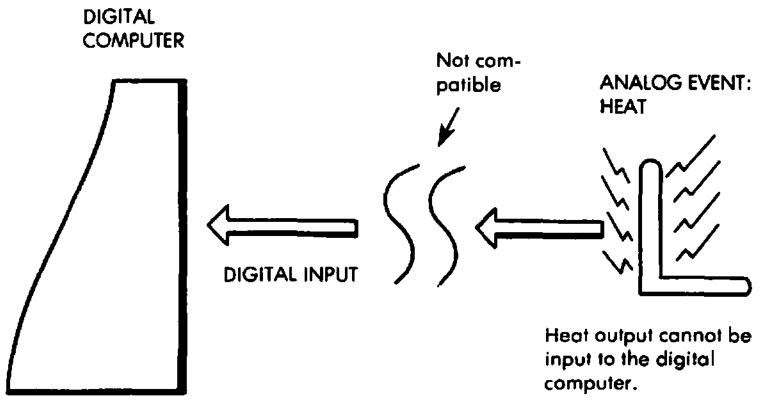

Voltage higher than logical 1 or 0 level required to power the digital circuit.
+10V
LOGICAL 0
OR
LOGICAL 1
INPUT
DIGITAL CIRCUIT
LOGICAL 0
OR
LOGICAL 1
OUTPUT

A block diagram showing that non-logical-level voltages may be required to power the digital circuits. However, the digital circuit will only process input and output at the logical 0 or logical 1 voltage levels.

Figure 7.3

when applied to electronics. In electronics, the terms refer to electrical quantities, such as resistance, current, and voltage. The quantity that you may be most familiar with is voltage. For example, the output of a car battery is an analog voltage. A battery rated at 12 volts may actually produce 12.02 V, 12.1 V, 11.9 V, 12.6 V or any value near 12 volts. As the battery ages, the voltage output may drop: 11.9 V, 10.88 V, 10.73 V. The voltage output of the battery is analog, not digital, because it varies. A digital voltage output would have a finite number of possible values. A battery would have to produce exactly 12.0 V, 12.5 V, or 12.75 V. A battery does not meet these conditions; it is an analog device.

Home computers are called digital because there are a finite number of possible voltage levels for the electrical information



An analog event, such as temperature, is not directly compatible with a digital computer; it cannot be measured or monitored without some intermediate device.

Figure 7.4

being processed within the system. We saw what these levels were when discussing input and output in Chapters 2 and 3 of this text; there were only two possible voltage levels in the system. All digital circuits used in computers have this characteristic, but they may not use the same actual voltage levels to process information. A power supply (battery) of different voltages may be used to run the digital system, but the digital circuits will output and input data at only two different voltages. (See Figure 7.3.)

In a digital electronic circuit, the two possible output levels are called "logical 1" and "logical 0." Because there are only two states, digital circuits are designed to follow the logic of binary (or Boolean) mathematics, a number system that has only two digits, 0 and 1. (We normally work in the decimal number system, which has ten digits, 0–9.)

Home computers and personal computers are digital, binary machines. They will operate on digital information only. Most of the events that occur in nature are analog. If we wish to control and monitor these analog events with a digital computer, there is a conflict. For a graphic illustration of this conflict, see Figure 7.4. If you can understand the difference between analog and digital events, you will learn to resolve the incompatibility between them more easily.

7.5: TRANSDUCERS

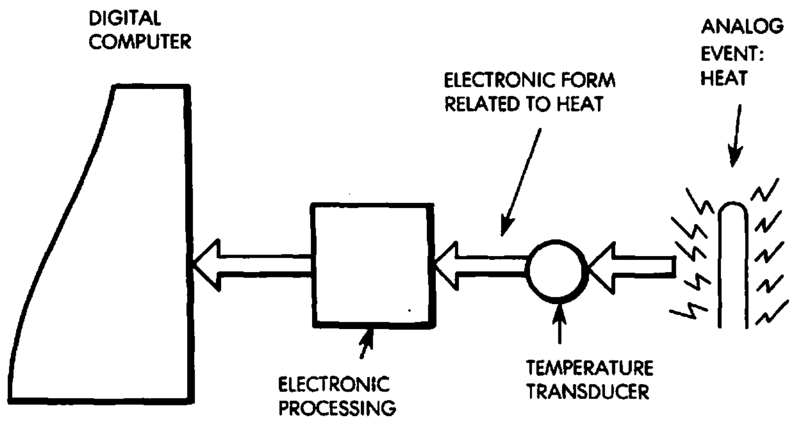
In the remaining chapters of this text we will discuss how to bridge the gap between the analog events typical of the real world and the digital electronics used to control or monitor them. Before we can begin to explore this problem, another piece of the puzzle must be explained. That remaining piece is the *transducer*.

We introduced the transducer in Chapter 1 and mentioned it again in Chapter 5. We return to it here to discuss this device in the context of ADC. At this point you probably have a better understanding of how the transducer fits into the computer control problem, of what such a device should do.

It has been stated that most of the phenomena we wish to monitor and control in the world are analog, or continuous. That is, they can be measured anywhere on a continuous scale of values. However, these values represent different physical characteristics. For example, the various scales of temperature measure the physical form of energy, heat. There are an infinite number of different temperature readings, but the form of energy they all represent is heat. Pressure is another physical quality, another form of energy, that we sometimes wish to monitor with the digital computer. It is also an analog event, because there are an infinite number of pressure outputs.

What we need is some way to transform the various physical forms of energy into a form that can be used in an electronic environment. That new form should be some electrical quantity: voltage, current, resistance, capacitance, or inductance. After the physical form of energy has been changed into electricity, it can be further processed using electronic techniques. Finally, we will need to input the form into a digital computer. The entire process is shown in the block diagram of Figure 7.5.

The device that transforms the physical quantity into an electrical quantity is called a *transducer*. Transducers are generally classified according to the physical form of energy that they transform into an electrical quantity. For example, a transducer that will transform pressure into an electrical form is called a



The flow of information in the monitoring system is from the analog event, heat, into the physical transducer. The transducer will transform the temperature output into its electrical equivalent. The electrical signal can then be processed electronically and, finally, input to the digital computer.

Figure 7.5

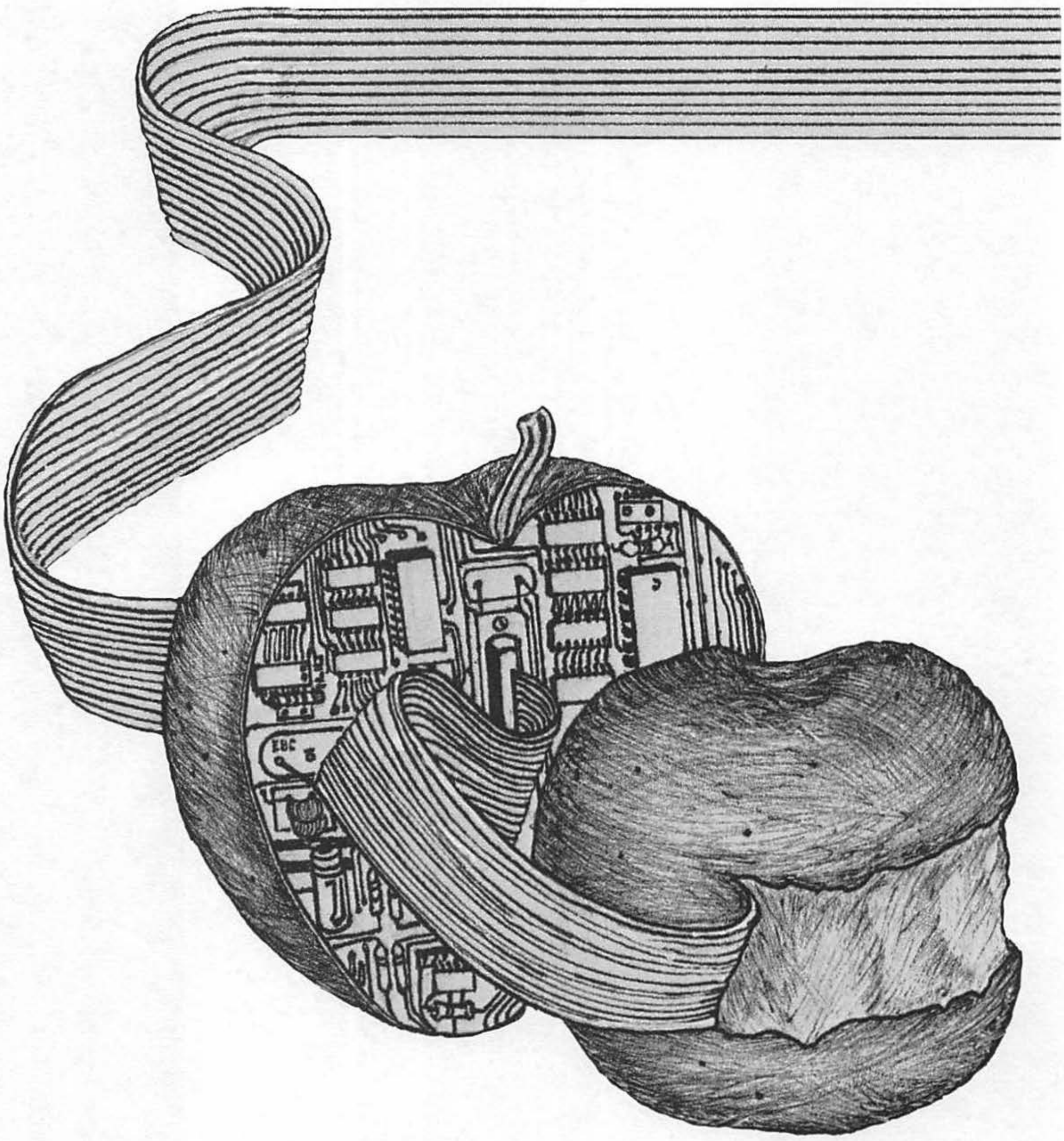
pressure transducer. One that will transform temperature into an equivalent electrical quantity is called a temperature transducer, and so on. Often, however, the qualifying term, "pressure" or "temperature," is omitted in the literature, since it is generally known what type of energy has to be measured. The reader must infer the type of transducer from the surrounding text.

In Chapter 8 we will describe a temperature transducer and use it to enable the Apple computer to measure an analog quantity, temperature.

7.6: SUMMARY

In this chapter we have discussed the meaning of the terms *analog* and *digital*, illustrating them with common events that occur in the world. After this introduction, we applied these concepts to electronics, and discussed digital and analog electrical quantities. Finally, we saw what a *transducer* is, and discussed how it fits into an automatic computer control environment.

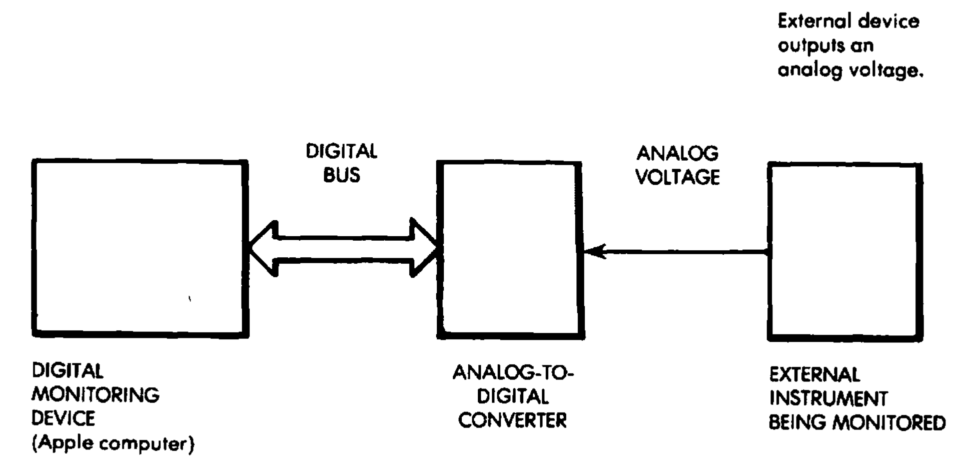
When we undertake the job of designing or using a computer-controlled system, we will need to know the difference between analog and digital quantities, and how each of the external devices being controlled or monitored outputs information. Sometimes the output is analog, and sometimes it is digital. If the external device produces an analog output, some type of conversion will be necessary. In the following chapter, we will discuss analog-to-digital conversion, and show how to input a physical quantity, temperature, to the Apple computer. In doing so, we will use information presented in this chapter.



Chapter 8
Analog to Digital Conversion For the Apple

IN THIS CHAPTER we will show how to use the Apple computer to monitor any external instrument which outputs an analog voltage. The discussion starts off by showing a block diagram of the overall concept involved in this type of computer monitoring. From there the discussion focuses on how an analog-to-digital converter operates. The discussion will be basic enough to enable the beginner to understand the principles of analog-to-digital conversion.

After the analog-to-digital converter is presented, this chapter will show how to electrically connect an analog-to-digital converter to the Apple computer via the I/O slots. Finally, a complete system for monitoring temperature will be explained. At the conclusion of this chapter, you will have a good understanding of how devices that output analog voltages may be monitored using a digital computer like the Apple.



Block diagram showing an overall view of the concepts involved in monitoring an analog voltage with a digital system.

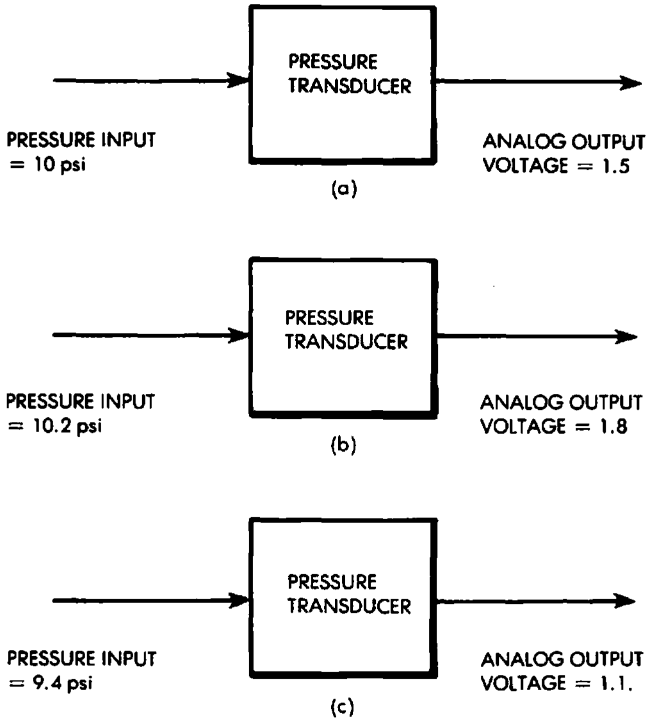
Figure 8.1

8.1: BLOCK DIAGRAM OF THE PROBLEM

Figure 8.1 shows a block diagram of the overall concept that we must use. In this diagram there are three major blocks shown:

1. Block 1 is the digital monitoring device. In this case it will be the Apple computer.
2. Block 2 will transform the analog voltage output from the external device to an equivalent digital word.
3. Block 3 is the external device itself. This device is really a transducer of some sort. The transducer will output a voltage that has an amplitude dependent on the physical event being monitored.

In the last chapter we discussed what was meant by an analog event. In review, an analog event is one that can be measured at an infinite number of possible output levels (within a given range). An example is pressure. Pressure can be measured at any value



a,b,c) Each of these block diagrams shows the analog voltage output from a given pressure input. Note that as the pressure changes at the input of the transducer, the analog voltage must change to reflect this.

Figure 8.2

from a few pounds per square inch (psi) to many hundreds of psi. Further, the value can be any number of psi between these two limits.

If the event to be monitored by the digital computer is analog, a transducer capable of accurately reporting the analog changes is necessary. For example, as the pressure being measured changes, the transducer must reflect this change in its output. When the pressure changes a little, either increasing or decreasing, the voltage output from the pressure transducer must increase or decrease in the same proportion. (See Figure 8.2.)

These changes are in contrast to a "digital" transducer. Such a transducer was used in Chapter 5 for the home security system. We needed these transducers to indicate whether the windows and doors were open or closed. Although it is true that a window or door can be wide open or just cracked open, in the example of Chapter 5 we did not need to distinguish between the degrees of openness; the event we were monitoring was "digital." That is, for our purposes, the door or window was either open or closed. The transducers used in Chapter 5 were switches.

The transducers we are concerned with now are analog output transducers. This means the transducer's output will vary in voltage anywhere from a low voltage, 0.0 volts, to a high voltage, approximately 5.0 volts. We use this voltage range simply as an example. Typically, the outputs of transducers are variable from any negative voltage to any positive voltage. Voltage variations depend on the type of transducer used. An actual, practical example will be presented later in this chapter.

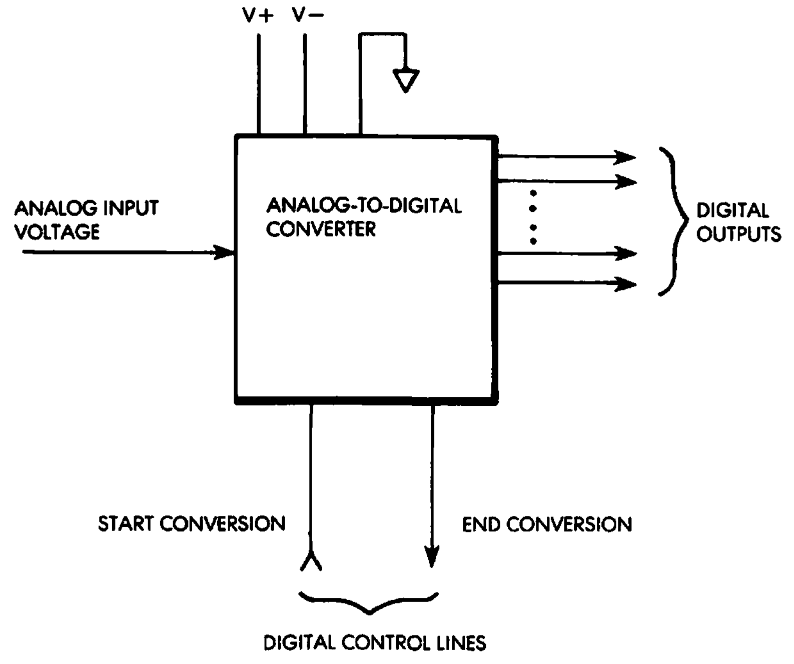
For the present time let us assume that an analog transducer will output a certain range of voltage. This analog voltage will be input to the block of Figure 8.1 labeled "analog-to-digital converter." The digital output of the analog-to-digital converter is then input to the Apple computer block.

The Apple computer will electrically input digital information being output from the analog-to-digital converter. As we will see, this digital information will be equivalent to the analog voltage that is input to the analog-to-digital converter.

8.2: THE ANALOG-TO-DIGITAL CONVERTER

Let us now define exactly what is meant by the term analog-to-digital conversion, and discuss how we can make use of this principle. In general terms, the function of analog-to-digital conversion is to transform (or convert) an analog input voltage into the digital output word that is its mathematical representation.

This function must be performed before a digital computer can input the analog voltage. A device that performs this type of function is called an *analog-to-digital converter,* or simply ADC. The remainder of this section will be devoted to explaining how a typical



Block diagram of a typical analog-to-digital converter.

Figure 8.3

ADC operates. After this discussion, we will present some examples of how the ADC device operates in a digital system environment.

We can begin the discussion by looking at Figure 8.3, a block diagram of a typical ADC. This diagram shows general ADC inputs and outputs. Let us concentrate on the function of each input and output shown in Figure 8.3, beginning with the power outputs. The ADC consists of electronic circuits, which require a source of Direct Current (DC) power. The power input lines are shown at the top of Figure 8.3. Typical power connections will be + and −12 or 15 volts, +5 volts, and ground.

The + and −12-to-15 volt DC supplies are used to power some of the internal circuits, such as operational amplifiers and voltage comparators, located inside the ADC package. A +5 volt supply will usually power the digital electronics inside the ADC package. Data sheets on any particular ADC will specify any DC power required for proper electrical operation.

In Figure 8.3 we also see the input line labeled "ANALOG VOLTAGE." This is the analog voltage that will be transformed into its equivalent digital word. Analog-to-digital converters have a specified range of analog voltages they can accept as input. Typical ADC analog input voltage ranges are 0 to +5 volts, −5 to +15 volts, −10 to +10 volts, and 0 to +10 volts. These are just a few of the many different voltage ranges that are commonly available. Refer to the manufacturer's data sheet for exact specifications on the analog input voltage range for a particular ADC.

Some of the input voltage ranges listed are negative (−) and positive (+), while some are only positive. If the input voltage to the ADC can be positive and negative, the ADC is said to be *bipolar*, meaning that it has two electrical poles. If the input voltage range is from zero to a positive value, the ADC is said to be *unipolar*, meaning that it has only one pole. The ADC we will use later in the chapter is unipolar.

The next section of Figure 8.3 to examine shows the eight digital output lines. The number of digital output lines on an ADC will vary. Typical numbers of output lines are 6, 8, 10, 12 and 16. In general, the greater the number of output lines, the more expensive the ADC.

The digital output lines on the ADC will be set to a logical 1 or a logical 0, as determined by the analog input voltage. Later in the discussion, it will be shown how to calculate which digital outputs will be a logical 1 and which will be a logical 0. For now, think of the digital output lines as the physical connections to the digital monitoring system. In our case, the connections are to the Apple computer.

Finally, in Figure 8.3, we see two digital control lines, labeled "START CONVERSION" and "END CONVERSION." Each line has a unique function. The "START CONVERSION" input control line is a digital signal that will electrically inform the ADC to start converting the analog input to a digital output word. This signal is generated by the digital monitoring device (in this case, the Apple computer).

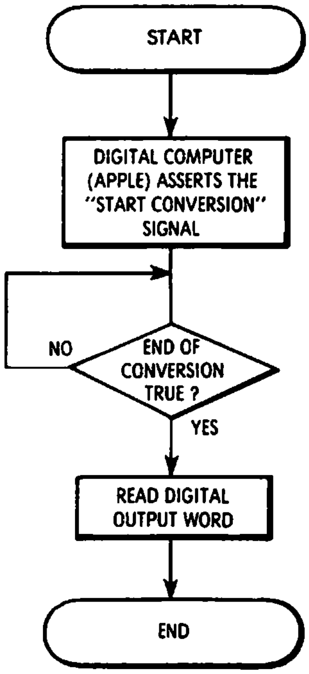
When the Apple computer is electrically prepared to accept the digital output word from the ADC, the "START CONVERSION" input control line to the device is *asserted;* that is, the input signal will be placed into the logical state that will start the action. Since the logical state could be a 1 or a 0, the term "asserted" is used to generalize.

The "END CONVERSION" output line shown in Figure 8.3 will be used to electrically inform the digital monitoring device that an analog-to-digital conversion is complete. Notice that the ADC is itself a small electronic system. It will take a short time for the ADC to adjust the digital outputs to new values based on the analog input voltage. The exact time required will vary from ADC to ADC. Typical times are between 10 and 200 microseconds, or .000010 and .000200 seconds.

When the Apple computer electrically requests an analog-to-digital conversion, it must electrically monitor the "END OF CONVERSION" output line during the conversion. This monitoring will electrically determine when the conversion is complete. Figure 8.4 shows a flowchart of how the computer must electrically communicate with the ADC.

In Figure 8.4 the analog-to-digital conversion will be started when the Apple computer asserts the "START CONVERSION" input line to the ADC. Once the hardware is connected, this will be accomplished using software. After the conversion is started, the Apple computer will electrically monitor the "END OF CONVERSION" line from the ADC. When this line is true, the Apple will read (input) the digital output lines of the ADC. This flowchart will be realized later in this chapter when the ADC is electrically connected to the Apple.

You have probably noticed that the discussion thus far has not gone into the details of how the ADC internally performs the analog-to-digital conversion. That topic is beyond the scope of this text. We present the ADC at the user's level, treating it as a "black box" that will perform a certain function in a certain prescribed manner. This is precisely the way ADCs are used. All of the electronics that actually perform the conversion are usually out of the user's control or access. This chapter, therefore, simply discusses how to make the ADC perform its prescibed function. Understanding this introduction to the ADC inputs and outputs


START
DIGITAL COMPUTER (APPLE) ASSERTS THE "START CONVERSION" SIGNAL
NO
END OF CONVERSION TRUE ?
YES
READ DIGITAL OUTPUT WORD
END

Flowchart showing the sequence of events required for electrical communication with an ADC.

Figure 8.4

has been our first step toward using this type of device with the Apple computer.

8.3: CALCULATING THE DIGITAL OUTPUTS OF THE ADC

In this section we will discuss how to determine the relationship between the logical conditions of the digital outputs and the voltage level of the analog input. For this discussion we will use an ADC that is unipolar. The acceptable voltage input range will be from 0.0 volts to +10.0 volts. There will be eight digital output lines. A block diagram of this type of device was shown in Figure 8.3.

The digital outputs will be labeled D0–D7, and will be assigned the same numerical weights we have given to bit positions in data

Digital Output Line	Weight
D0	1
D1	2
D2	4
D3	8
D4	16
D5	32
D6	64
D7	128

The ADC's output lines have the same numerical weights as the Apple's data bus lines.

Figure 8.5

bytes elsewhere in this text. A table of the digital output lines and the corresponding weights is shown in Figure 8.5.

As we have seen before, the minimum weight at the digital outputs will occur when all are a logical 0, and the maximum weight at the digital outputs will occur when all are a logical 1. The minimum and maximum weights are 0 and 255, respectively.

The minimum analog voltage input to the ADC is 0.0 volts. The maximum analog input voltage to the ADC is 10.0 volts. If we relate the maximum and minimum input voltage to the maximum and minimum digital output weights, we have:

0.0 volts input = weight 0 output
10.0 volts input = weight 255 output

All input voltages between 0 and 10 volts will have a corresponding digital weight between 0 and 255.

There are 256 different weights available for the input voltages, because there are eight digital outputs. The number of different weights available with a particular ADC is calculated by the following equation:

number of digital weights = 2 raised to the power of (number of digital output lines)

In our case we had eight digital output lines, so the number of digital weights was equal to 2^8, or 256. If we had an ADC with 10 digital output lines, the number of digital weights would equal 2^{10}, or 1024.

Since we know how many unique digital weights are available for the analog input voltage range, we can divide that range into equal increments. In our example we have 255 available digital weights. (One weight must be subtracted because of the 0 at the start.) Therefore, the range between 0 and 10 volts will be divided into 255 equal pieces by the following equation:

$$\frac{\text{voltage range}}{255} = .03921569$$

Using this information, we can generate a list of input voltages and their corresponding digital output weights. This list is shown in Figure 8.6.

We see in Figure 8.6 that the equivalent input voltage increases in discrete steps of approximately .04 volts. For example, suppose an input voltage to the ADC is equal to 1.5 volts. There is no digital output weight exactly equivalent to 1.5 volts. From the list of Figure 8.6 we must choose an output weight that will yield the voltage closest to 1.5 volts. We see that 1.5 volts is between 1.49 volts (a weight of 38) and 1.53 volts (a weight of 39). The weight of

Digital Weights and Voltages

D	V	D	V	D	V	D	V	D	V
0	0.00	1	0.04	2	0.08	3	0.12	4	0.16
5	0.20	6	0.24	7	0.27	8	0.31	9	0.35
10	0.39	11	0.43	12	0.47	13	0.51	14	0.55
15	0.59	16	0.63	17	0.67	18	0.71	19	0.75
20	0.78	21	0.82	22	0.86	23	0.90	24	0.94
25	0.98	26	1.02	27	1.06	28	1.10	29	1.14
30	1.18	31	1.22	32	1.25	33	1.29	34	1.33
35	1.37	36	1.41	37	1.45	38	1.49	39	1.53
40	1.57	41	1.61	42	1.65	43	1.69	44	1.73
45	1.76	46	1.80	47	1.84	48	1.88	49	1.92
50	1.96	51	2.00	52	2.04	53	2.08	54	2.12
55	2.16	56	2.20	57	2.24	58	2.27	59	2.31

D = DIGITAL WEIGHT
V = VOLTAGE

Figure 8.6

Digital Weights and Voltages

D	V	D	V	D	V	D	V	D	V
60	2.35	61	2.39	62	2.43	63	2.47	64	2.51
65	2.55	66	2.59	67	2.63	68	2.67	69	2.71
70	2.75	71	2.78	72	2.82	73	2.86	74	2.90
75	2.94	76	2.98	77	3.02	78	3.06	79	3.10
80	3.14	81	3.18	82	3.22	83	3.25	84	3.29
85	3.33	86	3.37	87	3.41	88	3.45	89	3.49
90	3.53	91	3.57	92	3.61	93	3.65	94	3.69
95	3.73	96	3.76	97	3.80	98	3.84	99	3.88
100	3.91	101	3.96	102	4.00	103	4.04	104	4.08
105	4.12	106	4.16	107	4.20	108	4.24	109	4.27
110	4.31	111	4.35	112	4.39	113	4.43	114	4.47
115	4.51	116	4.55	117	4.59	118	4.63	119	4.67
120	4.71	121	4.75	122	4.78	123	4.82	124	4.86
125	4.90	126	4.94	127	4.98	128	5.02	129	5.06
130	5.10	131	5.14	132	5.18	133	5.22	134	5.25
135	5.29	136	5.33	137	5.37	138	5.41	139	5.45
140	5.49	141	5.53	142	5.57	143	5.61	144	5.65
145	5.69	146	5.73	147	5.76	148	5.80	149	5.84
150	5.88	151	5.92	152	5.96	153	6.00	154	6.04
155	6.08	156	6.12	157	6.16	158	6.20	159	6.24
160	6.27	161	6.31	162	6.35	163	6.39	164	6.43
165	6.47	166	6.51	167	6.55	168	6.59	169	6.63
170	6.67	171	6.71	172	6.75	173	6.78	174	6.82
175	6.86	176	6.90	177	6.94	178	6.98	179	7.02
180	7.06	181	7.10	182	7.14	183	7.18	184	7.22
185	7.25	186	7.29	187	7.33	188	7.37	189	7.41
190	7.45	191	7.49	192	7.53	193	7.57	194	7.61
195	7.65	196	7.69	197	7.73	198	7.76	199	7.80
200	7.84	201	7.88	202	7.92	203	7.96	204	8.00
205	8.04	206	8.08	207	8.12	208	8.16	209	8.20
210	8.24	211	8.27	212	8.31	213	8.35	214	8.39
215	8.43	216	8.47	217	8.51	218	8.55	219	8.59
220	8.63	221	8.67	222	8.71	223	8.75	224	8.78
225	8.82	226	8.86	227	8.90	228	8.94	229	8.98
230	9.02	231	9.06	232	9.10	233	9.14	234	9.18
235	9.22	236	9.25	237	9.29	238	9.33	239	9.37
240	9.41	241	9.45	242	9.49	243	9.53	244	9.57
245	9.61	246	9.65	247	9.69	248	9.73	249	9.76
250	9.80	251	9.84	252	9.88	253	9.92	254	9.96
255	10.00								

D = DIGITAL WEIGHT
V = VOLTAGE

List of digital output weights and the corresponding analog input voltages they represent.

Figure 8.6 (cont.)

the ADC digital outputs for an input voltage of 1.5 volts is closest to 38, so we can use that number. However, it is not an exact representation of the analog input voltage.

The point is that the ADC can only digitally represent analog voltages between 0 and 10 volts to a resolution of approximately .04 volts. This error is dependent on the number of digital output lines and the analog voltage input range. If the value of .04 volts is too great a range, then a different ADC, with more digital output lines, must be chosen.

When using the ADC, we do not want to have to look up the equivalent voltage each time a conversion takes place. It would be much easier if the voltage could be calculated based on the digital output word. This would lend itself well to computer control. These calculations can be accomplished using the following equation: The analog voltage is equal to the digital output weight multiplied by the resolution of the ADC.

analog voltage = digital weight * (10/255)

For example, suppose the weight read from the ADC was equal to 125. This would equal an input voltage of:

125 * (10/255) = 4.90 volts

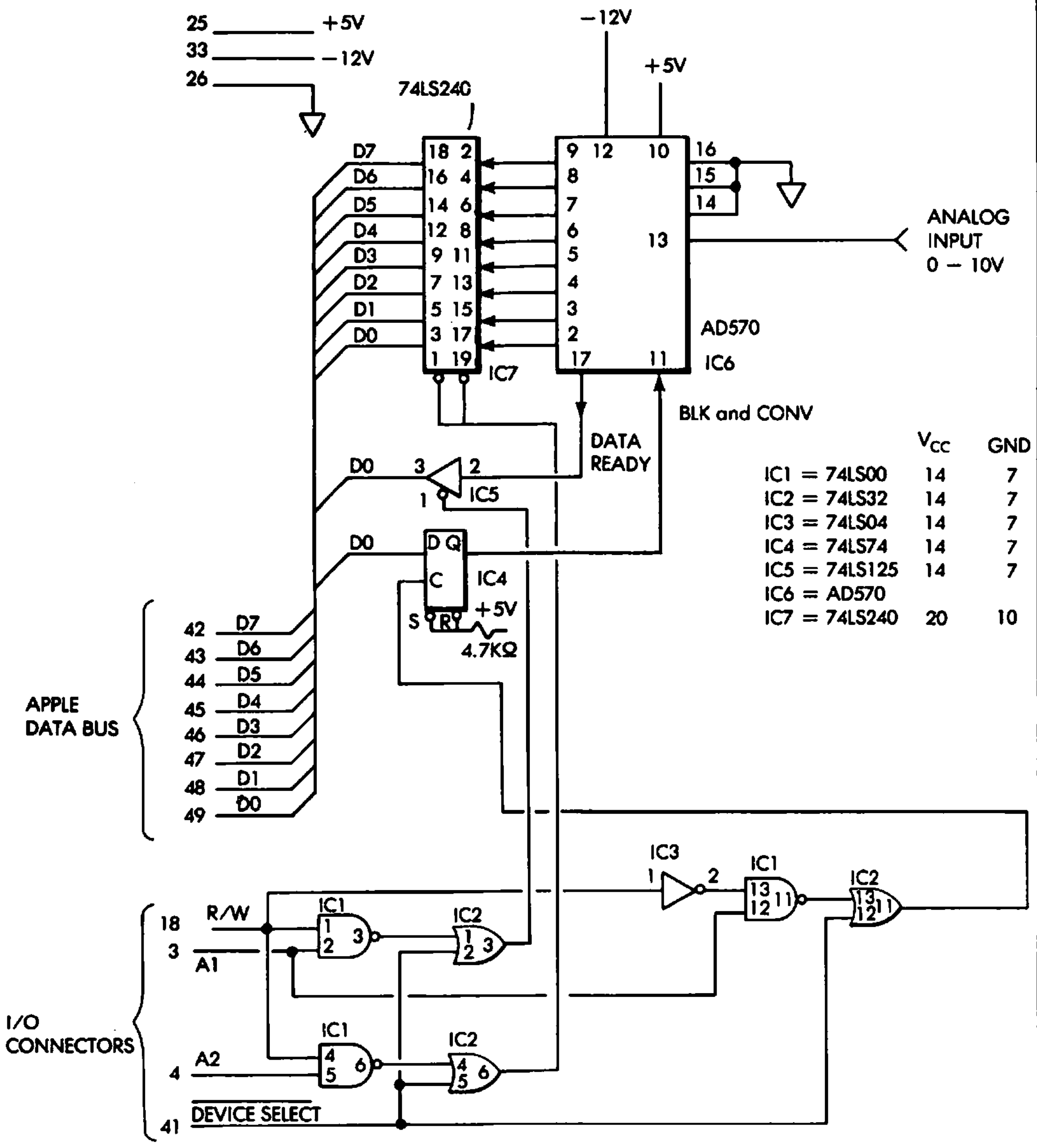
Remember that this voltage may actually equal 4.90 volts or slightly less than 4.94 volts, because of the resolution of the ADC.

In its present form, our equation is dependent on the input voltage range and the number of digital output lines on the ADC. These values must be modified for the equation to fit a particular application.

8.4: CONNECTING THE ADC TO THE APPLE COMPUTER

Figure 8.7 shows a complete schematic for connecting an actual ADC, Analog Devices' model AD570, to the Apple computer. In this section we will discuss exactly how each part of Figure 8.7 operates. The AD570 is very similar in its operating characteristics to the general example we gave in Figure 8.3. A data sheet for this device is included in Appendix A.

The analog-to-digital converter is labeled IC6 in Figure 8.7. The device is powered by the +12 volts, +5 volts and ground lines



	V_{CC}	GND
IC1 = 74LS00	14	7
IC2 = 74LS32	14	7
IC3 = 74LS04	14	7
IC4 = 74LS74	14	7
IC5 = 74LS125	14	7
IC6 = AD570		
IC7 = 74LS240	20	10

Complete schematic for an ADC interface to the Apple computer via an I/O slot.

Figure 8.7

output by the Apple computer. These power supply lines are specific pins on the I/O slot connectors in the computer. (The pin numbers for these connections are shown in Figure 8.7. Refer to Figure 4.1 for a display of the pinout of the Apple I/O slots.)

On the right side of the ADC (IC6) is the analog input voltage to be converted. This is the voltage input from an external source, between 0 and 10 volts. The digital outputs of the ADC are input to a tri-state buffer, a 74LS240, labeled IC7. We discussed how this type of buffer operated in Chapter 4. A similar circuit was also shown in Chapter 5.

The outputs of the buffer (IC7) are connected directly to the Apple data bus lines, D0–D7. When the Apple computer electrically reads the data at the ADC outputs, the buffer will become enabled. The Apple will read the data using the PEEK instruction as described in Chapter 3.

In order to start the analog-to-digital conversion, the input pin 11 of IC6 in Figure 8.7 must be set to a logical 1 and then reset to a logical 0. This is the "START CONVERSION" input signal for this particular ADC. When input pin 11 is reset to a logical 0, the analog-to-digital conversion will start. This is graphically shown by the wave form next to the signal line at IC6.

Input pin 11 of the ADC is connected to the Q output of a 74LS74 D flip-flop, IC4. This flip-flop is acting as a single-bit latch. The data input to the flip-flop is connected to the D0 data line on the Apple computer. Whenever the user wishes to set the Q output to a logical 1, the D0 line will go to a logical 1. At the same time, the flip-flop will be clocked.

We can perform all the steps necessary to start the conversion by using the POKE instruction in BASIC. When we wish to set the Q output of the flip-flop to a logical 1, the weight of D0 is used in the POKE instruction. For example, the following instruction:

```
POKE address, 1
```

will set the Q output of the D flip-flop to a logical 1, and the instruction:

```
POKE address, 0
```

will set the Q output of the D flip-flop to a logical 0. Later in the chapter, we will discuss how the address of these two instructions is computed. The important point here is that we can control an individual hardware line via a software instruction. Specifically, we can assume that the input pin 11 of IC6 can be set to a logical 1, or a logical 0, under software control.

To know when the analog-to-digital conversion is complete, the Apple computer must electrically monitor the output pin 17 of IC6. This pin is the "END OF CONVERSION" signal for the ADC. Output pin 17 of IC6 in Figure 8.7 is connected to input pin 2 of IC5. IC5 is a 74LS125 tri-state buffer. The data sheet for the 74LS125 is given in Appendix A. Pin 17 of the ADC will be a logical 0 when the analog-to-digital conversion is complete. At all other times it will be a logical 1.

The 74LS125 will perform a function similar to that of the 74LS240 mentioned earlier. That is, when the Apple computer executes a PEEK instruction to the correct address, the 74LS125 will become enabled. This will allow the logical state of the output pin 17 of IC6 to be enabled onto the Apple data bus. At that time the Apple computer will electrically input the logical level into a BASIC program.

We have now seen how the "END OF CONVERSION" signal is monitored by the computer. We have also seen how the computer will set the input pin 11 of IC6 to a logical 1, or a logical 0, under control of the software.

Each of these events will be accomplished using either a PEEK or a POKE instruction. A major part of both of these instructions is the address. We will now discuss how the address of the PEEK or POKE instruction will be related to the schematic diagram given in Figure 8.7.

To read the data from the ADC, the outputs of IC7 of Figure 8.7 must be enabled. This means pins 1 and 19 of IC7 must be held at a logical 0 voltage level. If we follow the connection from pins 1 and 19 of IC7, we can see that they are connected to the output pin 6 of IC2. Output pin 6 will be a logical 0 when the $\overline{\text{DEVICE SELECT}}$ line is a logical 0 and pin 6 of IC1 is a logical 0. Pin 6 of IC1 will be a logical 0 when the R/W line and the A2 address line are both a logical 1. Therefore, the logical conditions that will enable IC7 pins 1 and 19 are the following:

R/W = 1 (PEEK instruction)
A2 = 1 ($\overline{\text{DEVICE SELECT}}$ address + 4)
$\overline{\text{DEVICE SELECT}}$ = 0 ($\overline{\text{DEVICE SELECT}}$ address)

Let us assume that our I/O card is installed in I/O slot 4. The $\overline{\text{DEVICE SELECT}}$ address for this slot is −16192. The PEEK instruction needed to read the data from the ADC will be:

```
LET A = PEEK (−16192 + 4)
```

To read the logical conditions of output pin 17 of IC6, the address line A1 (instead of address line A2) must be a logical 1. The PEEK instruction to read this line will be:

```
LET A = PEEK (−16192 + 2)
```

The last signal we must control is the Q output of the flip-flop, IC4 in Figure 8.7. To write a logical 1 or a logical 0 to the D flip-flop, a POKE instruction is used. If we follow the clock (C) input line from the D flip-flop, we can see that it will connect to the output pin 4 of IC3. Whenever pin 4 of IC3 goes from a logical 0 to a logical 1, the information at the D input of the flip-flop will be latched at the Q output.

Address decoding of the clock line for the flip-flop is shown in the lower-right corner of Figure 8.7. Let us follow the logic of this decoding. IC2 pin 8 will become active when pin 10 is set to a logical 0. This is accomplished in the following way.

The R/W line from the Apple computer will be a logical 0 during a POKE instruction. This logical 0 is inverted and input to pin 13 of IC1. Pin 12 of IC1 is connected to address line A1 from the Apple computer. When address line A1 is a logical 1 and the R/W line is a logical 0, output pin 11 of IC1 is a logical 0.

Output pin 11 of IC1 is connected to input pin 13 of IC2. Input pin 12 of IC2 is connected to the $\overline{\text{DEVICE SELECT}}$ line output from the Apple computer. When the $\overline{\text{DEVICE SELECT}}$ output is a logical 0, the output pin 11 of IC2 is a logical 0. Output pin 11 will enable the following gate via pin 10. This output is inverted by IC3. Finally, the inverted output is the clock input to the 74LS74, IC4.

In review, the logical conditions that will enable the clock line to the flip-flop in Figure 8.7 are:

R/W = logical 0 (POKE instruction)
A1 = logical 1 ($\overline{\text{DEVICE SELECT}}$ address + 2)
$\overline{\text{DEVICE SELECT}}$ = 0 ($\overline{\text{DEVICE SELECT}}$ address)

All of these hardware conditions will be true when the POKE instruction is used as follows:

```
POKE I/O slot address, data
```

The data will be a 1 if we wish to set the Q output to a logical 1, and a 0 if we wish to set the Q output to a logical 0.

We have discussed the hardware action taking place in the schematic shown in Figure 8.7 in some detail, to relate hardware events to software instructions. The discussion was meant for those who wished to know exactly how the hardware of the interface operates.

8.5: SOFTWARE FOR ANALOG-TO-DIGITAL CONVERSION

Now that we have the ADC connected to the Apple computer I/O slot, let's control it with the software. We will show the BASIC software required to input a digital value from the ADC. A flowchart for the steps of this program is shown in Figure 8.8.

The following discussion will realize the flowchart of Figure 8.8 with BASIC programming statements. It is assumed that the ADC I/O device is located in slot 3 of the Apple computer. This will yield an I/O slot address of −16208. (Refer to Figure 2.7 for the address numbers of the eight I/O slots in the Apple computer.)

Step 1:

The first step is to write a logical 1 to the flip-flop on the ADC I/O board. We can accomplish this using a POKE instruction and setting address line A1 equal to a logical 1.

```
POKE -16208 + 2, 1
```

The value of 2 added to the I/O slot address will set address line A1 to a logical 1.

Step 2:

In this step the Q output of the flip-flop is reset to a logical 0. This can be accomplished in the same way as in step 1, except that the data is now a 0 instead of a 1:

```
POKE -16208 + 2, 0
```

The analog-to-digital conversion has now started.

Step 3:

We must now read the logical level of pin 17 of IC6 to determine if the conversion is complete. This can be accomplished using the following instruction:

```
LET R1 = PEEK (-16208 + 2)
```

Notice that when we read R1, its weight will be the summation of all the data lines, D0–D7. However, as we see in Figure 8.7, we are physically controlling the logical level of only one input line during this PEEK instruction. This input line is D7. The other input lines can be either a logical 1 or a logical 0. Further, the other input lines may not show the same value during each PEEK instruction. The actual logical level will depend on many factors, most of which are out of the user's control.

Therefore, we would like to logically ignore data lines D0–D6 during the PEEK instruction. Although we cannot electrically do this, it can be done using software. In the system software we can ignore data lines D0–D6 by testing for a value greater than 127. (If the value of R1 is at least 128, we know that D7 must have been a logical 1.) It can be done as follows:

```
IF R1 > 127 THEN GOTO (line number of PEEK instruction above)
```

After this instruction is executed, the value of R1 will be 1 if the D7 input line is a logical 1, or 0 if the D7 input line is a logical 0. Here is the reason why:

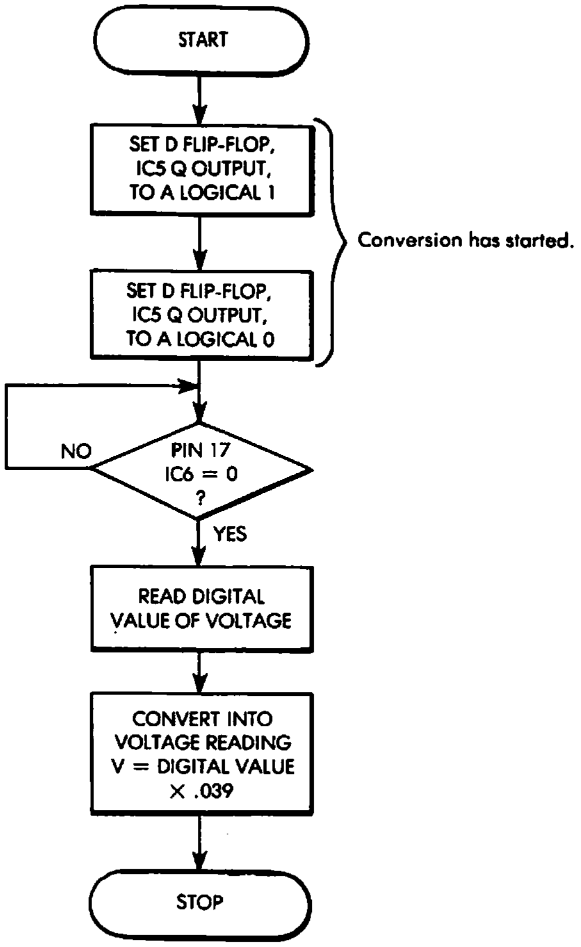
When R1 was read into the computer, by means of the PEEK instruction, it was made up of eight data input lines. Only one of these lines is of importance, D7. The input data will logically appear in one of two ways:

D7	D6	D5	D4	D3	D2	D1	D0
1	X	X	X	X	X	X	X

or

D7	D6	D5	D4	D3	D2	D1	D0
0	X	X	X	X	X	X	X

In these two cases the logical state of D7 was either a 1 or a 0. The other data lines are shown as "X". This means we do not care what the logical state of these data input lines is. However, their


START
SET D FLIP-FLOP, IC5 Q OUTPUT, TO A LOGICAL 1
SET D FLIP-FLOP, IC5 Q OUTPUT, TO A LOGICAL 0
Conversion has started.
NO
PIN 17 IC6 = 0 ?
YES
READ DIGITAL VALUE OF VOLTAGE
CONVERT INTO VOLTAGE READING V = DIGITAL VALUE X .039
STOP

Flowchart showing the sequence of events required to electrically communicate with the circuit shown in Figure 8.7.

Figure 8.8

logical state is important, as it will affect the overall computed weight of the input byte.

By testing whether R1 is greater than 127 or not, we can effectively ignore all data lines DO–D6. If those lines were all a logical 1 and D7 was a logical 0, R1 would be equal to 127. Therefore, it will not matter if lines D0–D6 are a logical 1 for this test. If R1 is a 1, then the analog-to-digital conversion is not complete and the program must loop, waiting until it is.

Step 4:

If the conversion is complete, the value of R1 will be less than 127. The digital outputs of the ADC can now be read by the computer. This will be done by the following statement:

```
LET V1 = PEEK (-16208 + 4)
```

The variable V1 will be assigned the combined weight of the input data lines.

Step 5:

The final operation is to compute the analog voltage that was input. This is accomplished by multiplying the weight of V1 by the constant (10/255). We discussed this procedure earlier in this chapter. The instruction that will do this is the following:

```
LET V2 = V1 * (10/255)
PRINT "WEIGHT = ";V1;" VOLTAGE IN VOLTS = ";V2
```

If we now put the entire set of instructions together, they will appear as shown in Figure 8.9. (It is assumed that the starting line number for this routine is 100.) The RETURN statement in line 170 was used on the assumption that all of the statements, 100–170, were contained in a BASIC subroutine. This would allow us to access the ADC at any time during a program by executing a GOSUB 100 statement.

8.6: TEMPERATURE MEASURING CIRCUIT (TRANSDUCER)[1]

In this section we will discuss a circuit that can be used to sense temperature. The circuit will produce an analog output voltage that is a function of the temperature. To put it another way, the

[1]This temperature-control application is based on the 8085 microprocessor version which appeared as experiments 9-5 and 9-6 in *The 8085/SDK-85 (Hands-on) Volume 2, 54 Control Experiments*, by Howard Boyet, New York: Microprocessor Training Inc.

```
100  POKE -16208 + 2, 1
110  POKE -16208 + 2, 0
120  LET R1 = PEEK (-16208 + 2)
130  IF R1 > 127 THEN 120
140  LET V1 = PEEK (-16208 + 4)
150  LET V2 = V1 * (10/255)
160  PRINT "WEIGHT = ";V1;" VOLTAGE = ";V2
170  RETURN
```

A complete routine for analog-to-digital conversion. This routine implements the flowchart shown in Figure 8.8.

Figure 8.9

voltage produced by the temperature-sensing device will be mathematically related to the temperature measured.

The circuit we will use is shown in Figure 8.10 parts (a) and (b). Part (a) shows the device pinout, and part (b) shows how to connect the device as a temperature-sensing transducer. The device is a National Semiconductor LM135, LM235, or LM335. Each device will work; the difference is in the temperature range (and the price) of each one.

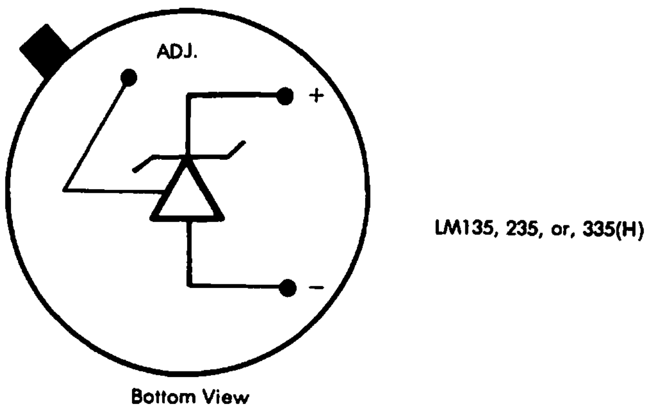
The specified temperature ranges are:

LM135H	−55 to +150° C
LM235H	−40 to +125° C
LM335H	−10 to +100° C

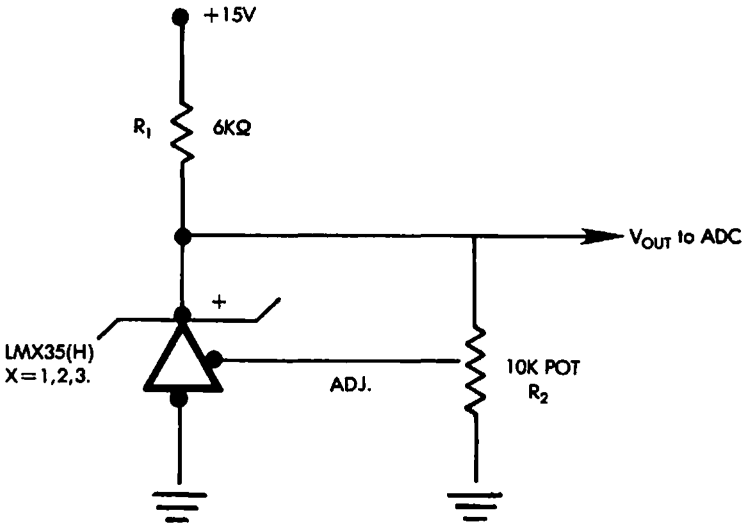
The "H" denotes that the device is packaged in metal. A model number ending in "Z" would indicate that the device was packaged in ceramic. Either of these packages will work equally well for this application.

All of the circuit components shown in Figure 8.10(b), such as the resistors R1 and R2, were located at a distance from the transducer, as shown in Figure 8.11. The reason for taking this precaution is simply that resistors may change their value as a function of heat, which could affect the accuracy of the temperature sensor's reading if the two devices are placed too close together.

The voltage out (V_{OUT} in Figure 8.10(b)) is connected to the ADC circuit we discussed in Section 8.4. The mathematical relationship



(a)

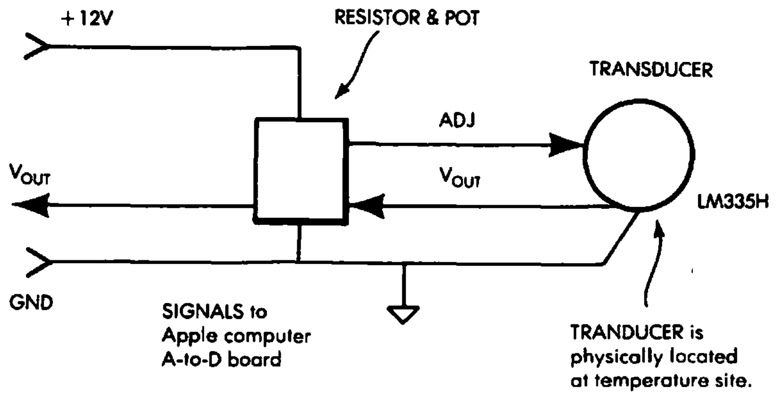


(b)

a) Pinout of the LM335.

b) Schematic diagram showing how the LM335 can be connected to act as a temperature transducer.

Figure 8.10


+12V
RESISTOR & POT
TRANSDUCER
ADJ
V_{OUT}
V_{OUT}
LM335H
GND
SIGNALS to Apple computer A-to-D board
TRANDUCER is physically located at temperature site.

Block diagram showing how the circuit components of Figure 8.10(b) are located at a distance away from the actual temperature measuring site.

Figure 8.11

between V_{OUT} and the temperature to be measured is:

$$\frac{T\text{ (in degrees Kelvin)}}{100} = V_{OUT}$$

where degrees Kelvin = 273.2 + degrees Centigrade. Using this equation we can calculate the voltages of some known temperatures, as shown in Figure 8.12. This table is presented to show you the relationship that exists between the different units that we must work with in this application. When the temperature is measured, converted, and input to the Apple, we will know the digital weight. From this, the output voltage of the transducer can be calculated as shown in the previous section of this chapter.

Once the voltage is calculated, the temperature can be calculated in degrees Centigrade, using the following set of equations:

a. $\dfrac{273.2 + \text{degrees Centigrade}}{100} = V_{OUT}$

b. $100 * V_{OUT} = 273.2 + \text{degrees Centigrade}$

c. $(100 * V_{OUT}) - 273.2 = \text{degrees Centigrade}$

d. $V_{OUT} = (\text{digital word}) * (10/256)$

e. $\text{Degrees Centigrade} = (100 * (\text{digital word}) * 10/256) - 273.3$

Temperature		
Degrees Centigrade	Degrees Kelvin	Voltage V_{OUT}
0	273.2	2.73
25	298.2	2.98
50	323.2	3.23
75	348.2	3.48
100	373.2	3.73

This table illustrates the relationship between the temperature measured, and the voltage output, by the temperature transducer.

Figure 8.12

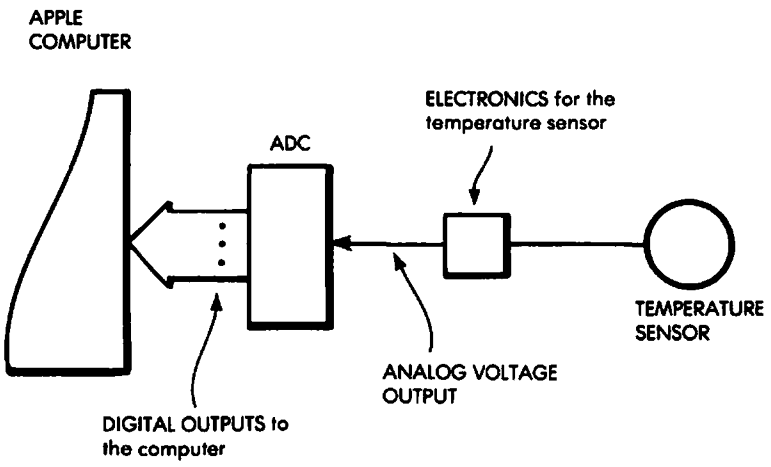
This type of mathematics is very simple to perform using BASIC on the Apple computer. In fact, as we will see, only the last equation and a PRINT statement must be added to our subroutine to display the temperature measured.

8.7: THE COMPLETE SYSTEM FOR TEMPERATURE MEASUREMENT

We are now ready to connect the entire system together to perform the function of automatic temperature measurement. When the system is connected via the hardware, the software will enable you to measure and record any temperature in degrees Centigrade. Some examples of temperature monitoring are given at the end of this chapter.

Figure 8.13 shows the block diagram of the complete hardware system. We have already discussed each major block of Figure 8.13. Using this block diagram let us write a complete BASIC program that will accept the data from the ADC, and calculate and finally print the temperature in degrees Centigrade.

It should be pointed out that no attempt has been made to minimize the number of BASIC statements in the program. The program is intended to instruct. In that light it is written in a very straight-


APPLE COMPUTER
ADC
ELECTRONICS for the temperature sensor
TEMPERATURE SENSOR
ANALOG VOLTAGE OUTPUT
DIGITAL OUTPUTS to the computer

Block diagram of the complete system required to automatically measure temperature with the Apple computer.

Figure 8.13

forward manner, and documented at each step with REM lines. The program shown in Figure 8.14 assumes that the ADC hardware is connected to I/O slot 3 of the Apple computer. The subroutine at line 200 may be called whenever a temperature is needed.

8.8: SOME PRACTICAL ADC APPLICATIONS

One application for monitoring temperature in your home is to chart a temperature profile of the house. This will allow you to better determine where problem areas of heat escape are located. To accomplish this you can connect one or more temperature probes to the Apple computer. Each probe can be placed in a certain area of the home. When the probes are in place you can program the computer to monitor the temperature for 24 hours.

The temperature could be sampled at 10-minute intervals, with the computer storing the resulting values on a disk. At the end of the 24-hour period, the computer could be programmed to plot

```
 10  GOSUB 200
 20  PRINT "TEMPERATURE IN DEGREES CENTIGRADE = ";T1
 30  STOP
 40  REM: THE SUBROUTINE FOR TEMPERATURE MEASURING
200  POKE -16208+2,1
210  POKE -16208+2,0
215  REM: THE ABOVE WILL START THE A/D CONVERSION
220  LET R1=PEEK(-16208+2)
230  LET R1=R1 AND 1
240  IF R1=1 THEN 220
245  REM: THE ABOVE WILL WAIT UNTIL CONVERSION IS COMPLETE
250  LET V1=PEEK(-16208+4)
255  REM: THE ABOVE WILL READ THE DIGITAL WEIGHT
260  LET V2=V1 * (10 / 255)
265  REM: THE ABOVE WILL CALCULATE THE VOLTAGE INPUT
270  LET T1= (100 * V2)-273.2
280  RETURN
290  REM: THE ABOVE CALCULATES THE TEMP IN DEGREES CENTIGRADE
```

This program will accept an input voltage from an ADC, and calculate and finally print the corresponding temperature in degrees Centigrade.

Figure 8.14

the temperature against the time of day (or any other meaningful axis). When all of the important rooms and areas in the home have been profiled, you could then make whatever physical adjustments are necessary to maintain the desired temperature.

A second application for a temperature probe would be to monitor the temperature outside the home vs. the temperature inside the home. If the desired inside temperature were cooler than the outside temperature, the computer could be made to let the warmer outside air heat up the home instead of the furnace.

Besides the temperature probe, there are many home computer applications for using analog-to-digital conversion. Some of these applications are listed below:

1. Sound detection. Sound could be converted into an analog voltage by a microphone (another kind of transducer). If

the sound reached a certain level, the computer could take some action. This could be part of a security system that detected the noise of an intruder.

2. Wind direction. You could make a transducer that would produce an analog voltage directly proportional to the direction of the wind. For example, 5.0 volts = north, 3.75 volts = east, 2.5 volts = south and 1.25 volts = west. All other compass directions would be scaled accordingly.
3. A barometer. You could connect a transducer that would produce an analog voltage proportional to the barometric pressure.
4. Moisture measurement. Using the correct transducer, the moisture of the soil could produce an equivalent analog voltage. This would allow the computer to determine when the sprinkler system should be turned on.

These are just a few of the many applications for temperature measurement and analog-to-digital conversion that can be accomplished with the home computer.

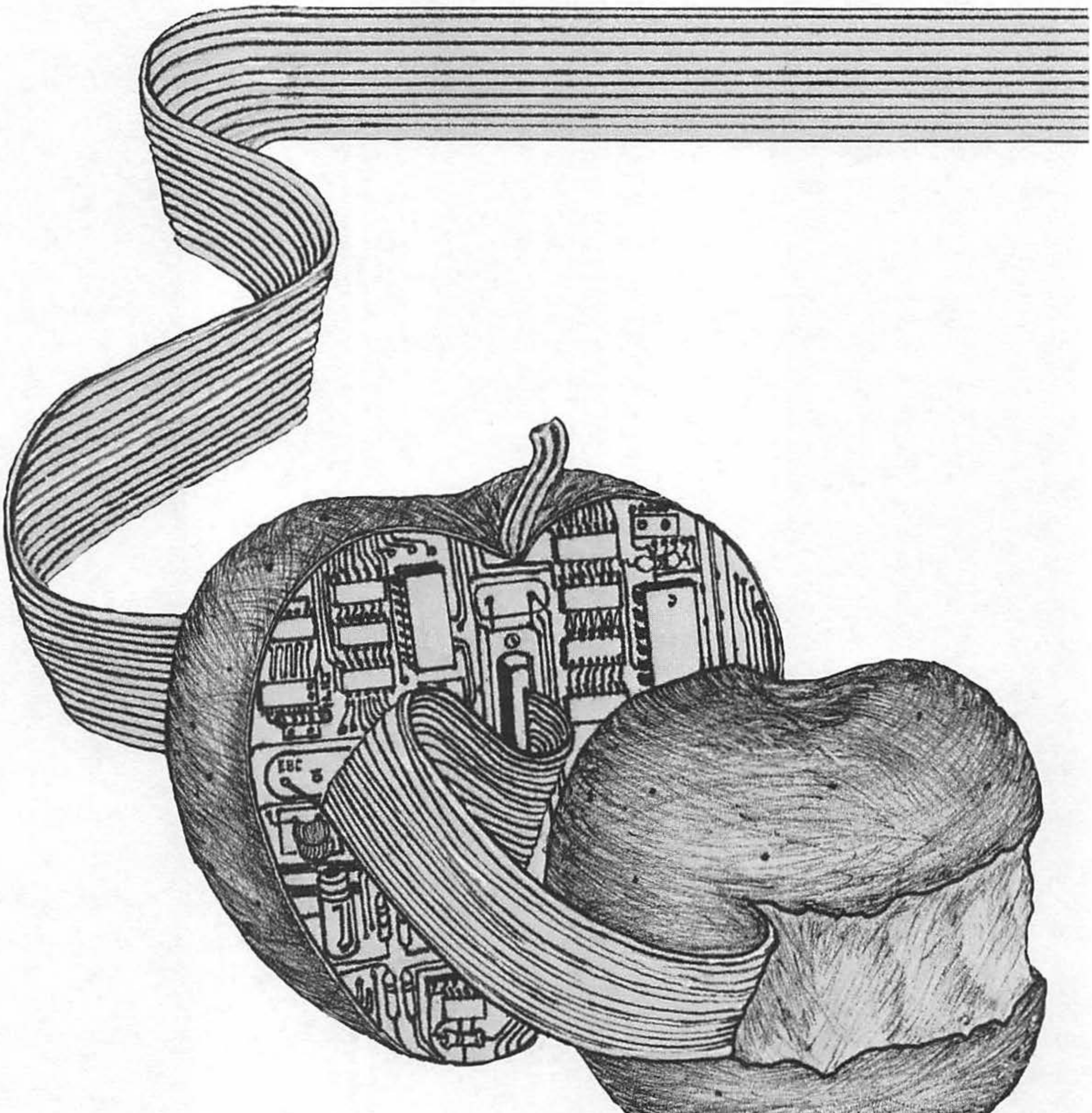
8.9: SUMMARY

In this chapter we have discussed the details of analog-to-digital conversion. The discussion started by explaining the need for analog-to-digital conversion. Next we discussed conceptually how it is accomplished. This discussion was at the user's level. We deliberately did not attempt to explain the details of how an ADC operates internally.

Next we proceeded to connect a real analog-to-digital converter, the AD570, to the Apple computer. Actual circuits were designed and discussed. It was shown how to calculate the input voltage based on the digital word or weight read from the ADC. A temperature transducer was then connected to the ADC. A complete temperature-monitoring system was presented, along with the software for controlling the system. Finally, we suggested a few practical applications for a system involving an ADC and an Apple.

The main focus of this chapter was to show how to perform ADC using an Apple computer. The example of inputting a temperature was meant only as a single illustration of how ADC may be employed in computer control. Whatever they measure, all tranducers generate electricity, and whenever an analog voltage needs to be monitored by a digital computer, it can be done in a manner similar to that shown in this chapter.

There are several available "off-the-shelf" analog-to-digital converters for the Apple computer. If you plan on using one of these, the information given in this chapter will help in understanding how to apply it to the Apple computer.

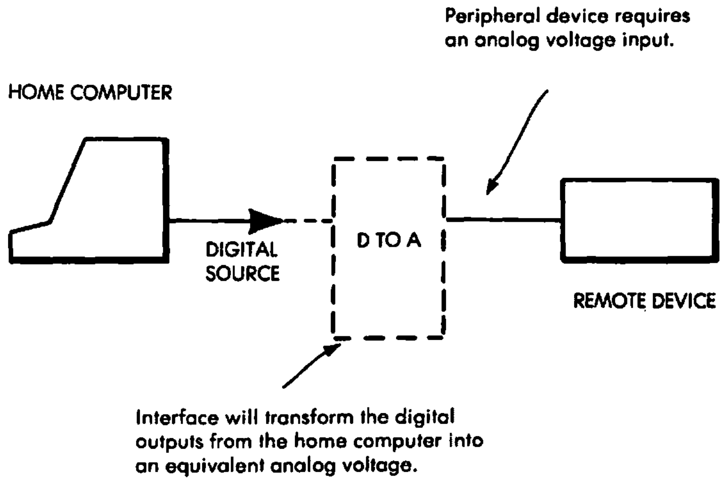

EBC

Chapter 9
Digital to Analog Conversion for the Apple

CERTAIN PERIPHERAL DEVICES that can be controlled by a home computer require an analog input voltage in order to operate. (We discussed what is meant by an analog voltage in Chapter 7.) A home computer system is completely digital, which presents us with an interfacing problem. What we need is a means of producing an analog voltage from a digital source or controller. Such a process is called *digital-to-analog conversion*, or more simply, DAC. Figure 9.1 shows a block diagram of this concept.

One type of external device that might require an analog input voltage would be a direct current (DC) motor. Most DC motors have the operating characteristic of rotating faster when a higher voltage is applied to the input. This type of external device, along with its analog input, is shown in the block diagram of Figure 9.2.

We have shown only a single example of an external device that would require an analog voltage, but there are many other applications using devices that require an analog voltage for control. For instance, communications and recording systems utilize digital-to-analog conversion and analog-to-digital conversion at their input and output. Electronic music and waveform-generation


Peripheral device requires
an analog voltage input.
HOME COMPUTER
DIGITAL
SOURCE
D TO A
REMOTE DEVICE
Interface will transform the digital
outputs from the home computer into
an equivalent analog voltage.

Block diagram showing the interfacing problem that occurs when a completely digital source is to be used to control a peripheral device that requires an analog voltage input. A DAC will solve this interfacing problem.

Figure 9.1

systems also use digital-to-analog conversion as a part of the overall system. These systems are growing in popularity. In short, any device whose output (speed, musical pitch, brightness, etc.) is variable will need digital-to-analog conversion to be controlled by digital electronics.

The intention of this chapter is to show how you can control and set an analog output voltage from a digital source, such as the Apple computer. Our discussions will cover the basics of digital-to-analog conversion. We will design and discuss the hardware and software necessary to allow the Apple computer to automatically control an analog output voltage from any I/O slot. The circuits shown are inexpensive and available from many suppliers. These circuits will show one way to achieve digital-to-analog conversion. Nothing in this chapter will be left up to the reader; no important detail will be omitted. All pin numbers, connecting lines and types of devices will be labeled.

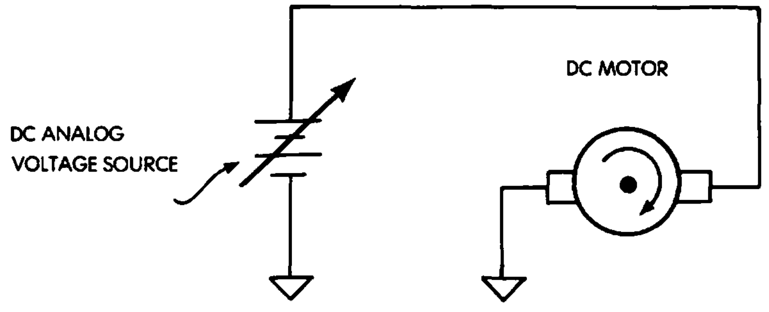


Diagram showing a DC motor being controlled by an analog voltage. When the voltage increases, the motor will increase in speed. This is only one type of peripheral device that can be controlled by an analog input voltage.

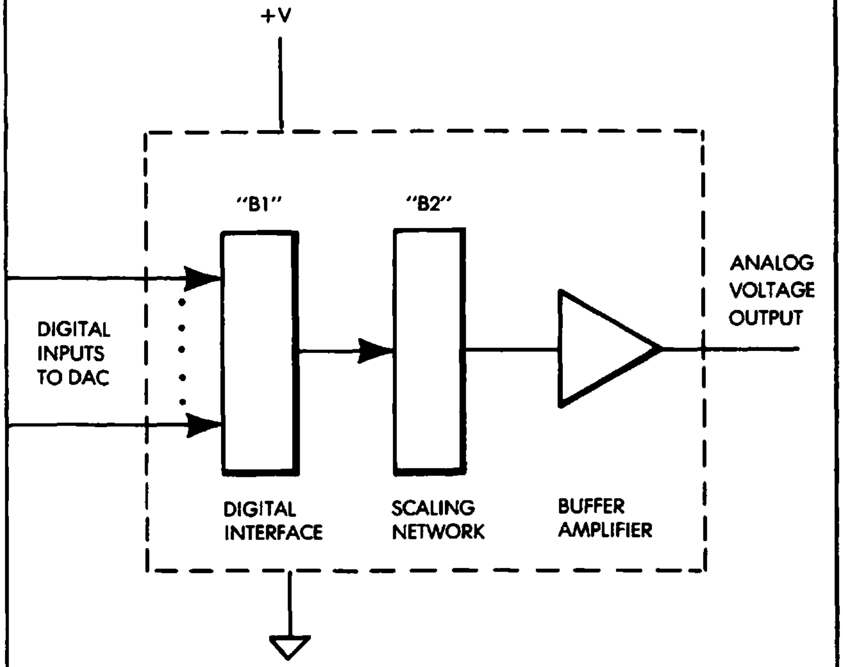
Figure 9.2

9.1: WHAT IS DIGITAL-TO-ANALOG CONVERSION?

Before we design and discuss a digital-to-analog conversion interface in detail, let us examine the overall concept. In the introduction to this chapter a very informal definition was given. This section will bring that definition into sharper focus.

Figure 9.3 shows a general block diagram of a digital-to-analog converter. In this diagram the block labeled "B1" is the electrical interface to the digital source that will input data to, and control, the converter. In our case the source will be the digital output lines from one of the Apple computer I/O slots. Block B1 will control the block labeled "B2" in Figure 9.3.

Block B2 is the precision scaling network. The job of the scaling network is to electrically determine how much of the reference voltage or current will appear at the converter output. For example, suppose the reference voltage is equal to 8.00 volts. The scaling network is capable of setting the output voltage to an exact portion of the reference, say 2.04 volts. The reference could also be a



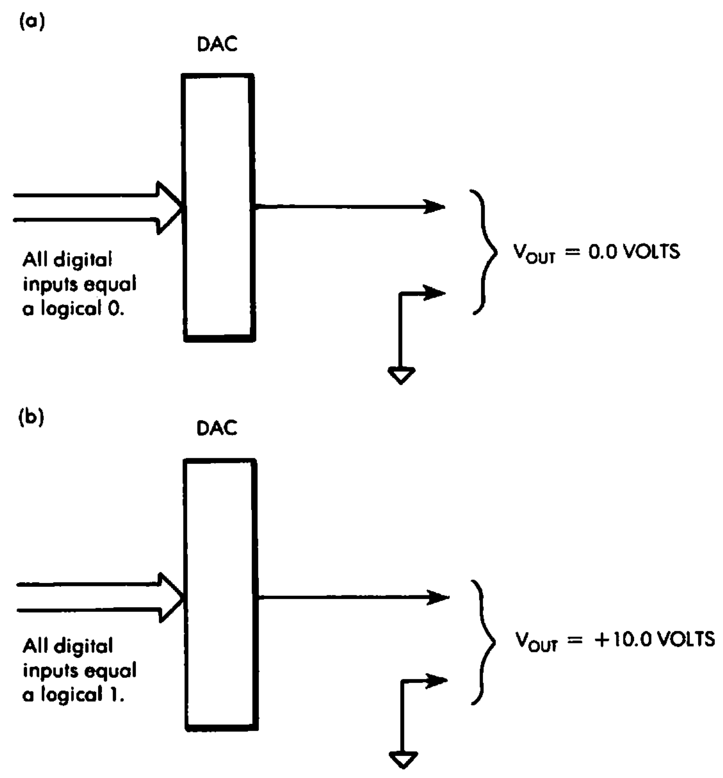
Block diagram of a general digital-to-analog converter. Most digital-to-analog converters will have a block diagram similar to this one.

Figure 9.3

smaller voltage than the output device requires, such as 1.5 volts. In this case, the scaling network would select a portion of the 1.5 volts, and this would be amplified before being output from the DAC.

Digital inputs from the computer will electrically control the scaling network, determining how much of the reference will be applied to the output. By setting the proper digital information on the DAC inputs, we could force the DAC output voltage to be equal to exactly half of the 8.00-volt reference, or 4.00 volts. We cannot give a formal, general definition of what voltage will appear at the output as a function of the digital inputs, because every digital-to-analog converter may have different specifications.

Using what we know about digital-to-analog conversion, let us take an example and show how to set the analog output voltage as

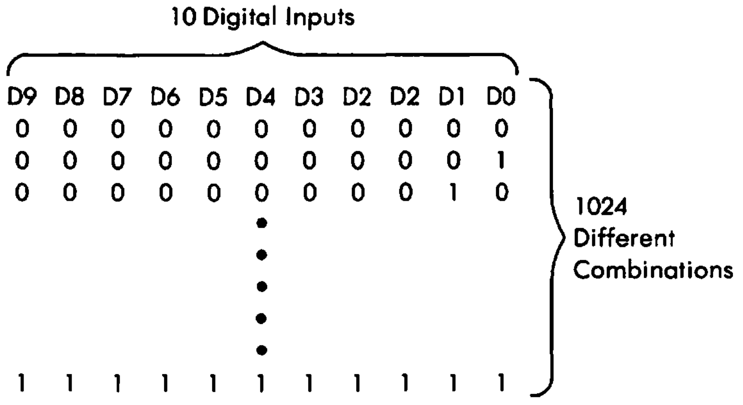

(a)
DAC
All digital inputs equal a logical 0.
V_{OUT} = 0.0 VOLTS
(b)
DAC
All digital inputs equal a logical 1.
V_{OUT} = +10.0 VOLTS

a) When all digital inputs to the DAC are a logical 0, the output voltage will equal 0.00 volts.

b) When all digital inputs to the DAC are a logical 1, the output voltage will equal 10.00 volts. These are the extreme ranges of this DAC's output voltage swing.

Figure 9.4

a function of the digital inputs. Suppose we have a digital-to-analog converter that will output 10.00 volts when all of its digital inputs are set to a logical 1; that is, suppose we have a DAC whose reference voltage is 10.00 volts. (We use this as an illustration only. Every DAC has a unique set of specifications.) Let us further assume that the DAC will output 0.0 volts when all of the digital inputs are a logical 0. Figure 9.4 shows these two conditions. A DAC with this type of voltage output specification is called unipolar. The



D9	D8	D7	D6	D5	D4	D3	D2	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0
					•					
					•					
					•					
					•					
					•					
1	1	1	1	1	1	1	1	1	1	1

With 10 digital inputs to the DAC, there are 1024 different combinations that may be applied. The number 1024 is derived from raising 2 to the tenth power.

Figure 9.5

output voltage will swing in one direction (toward one electrical pole) only. In this case the swing was from 0.0 volts to +10.00 volts. The term "unipolar" also describes an analog-to-digital converter with the same voltage characteristics.

Next we need to know how many digital inputs the DAC has. We will assume that our hypothetical converter has 10 digital inputs. There are therefore 2^{10}, or 1024, different input combinations that can be used. (See Figure 9.5.) From this number we can calculate the minimum voltage swing, the increment by which the voltage will change for each unique digital combination on the input.

To calculate how much the DAC output voltage will change when the digital input combination changes by only one least significant bit, we can use the following formula:

1. Minimum output voltage change = maximum output voltage change/maximum number of states − 1
2. Maximum output voltage change = V_{OUT} max − V_{OUT} min
3. V_{OUT} min = 0.0 volts, V_{OUT} max = 10.0 volts
   Maximum output voltage change = 10.0 − 0.0 = 10.0

4. Maximum number of unique digital combinations with 10 inputs = 1024. One of these combinations is used to output 0.0 volts. Therefore, we have actually 1023 states that produce a voltage.
5. Minimum voltage swing = 10.0 volts/1023 = .009775 volts, or 10 millivolts.

This means the output voltage will change 10 millivolts for each digital input bit that changes. (See Figure 9.6.) This DAC would be described as a unipolar, 10-bit, voltage DAC. (The term "voltage" indicates that the output is a voltage and not a current.) Suppose our hypothetical DAC had 12 inputs instead of 10. Using the formulas given, the minimum voltage change at the output would be equal to:

6. Minimum voltage change = 10.0 volts/4096−1 = 10/4095 = .0024 V

Let us now suppose that this same DAC had only 6 inputs. The minimum voltage change at the output pin would be:

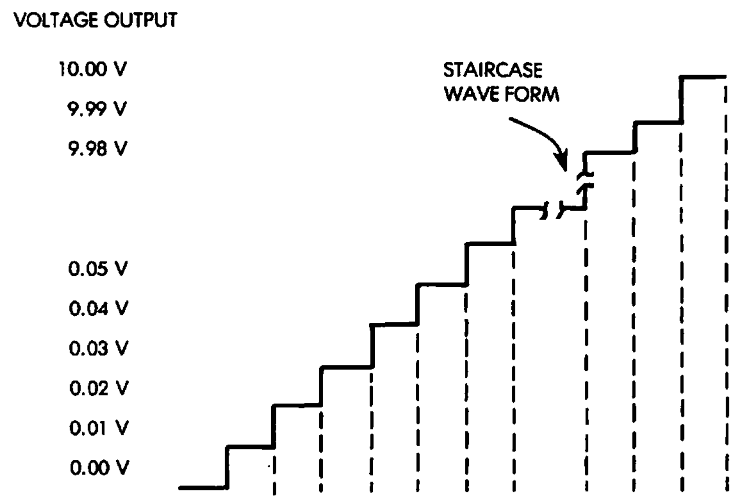
7. Minimum voltage change = 10.0 volts/64 − 1 = 10.0/63 = .158 V

Notice that, for a given output voltage range, the greater the number of digital inputs, the smaller the minimum voltage change at the DAC output. (See Figure 9.7.) In general, the greater the number of digital input lines, the better the output voltage "resolution"; that is, the DAC can resolve a smaller increment of voltage. This means we can come closer to obtaining the "smooth" staircase waveform shown at the top of Figure 9.7.

9.2: AN ACTUAL DIGITAL-TO-ANALOG CONVERTER

To illustrate the points we have covered about digital-to-analog conversion, let us examine a real DAC that is available "off the shelf." The DAC chosen for this example is the AD558, manufactured by Analog Devices. A complete data sheet for this device is given in Appendix A. A block diagram of the AD558 is shown in Figure 9.8; let us discuss it in detail.

In this figure we see the digital input block, labeled DB0–DB7. These digital inputs will connect to the data lines of an Apple

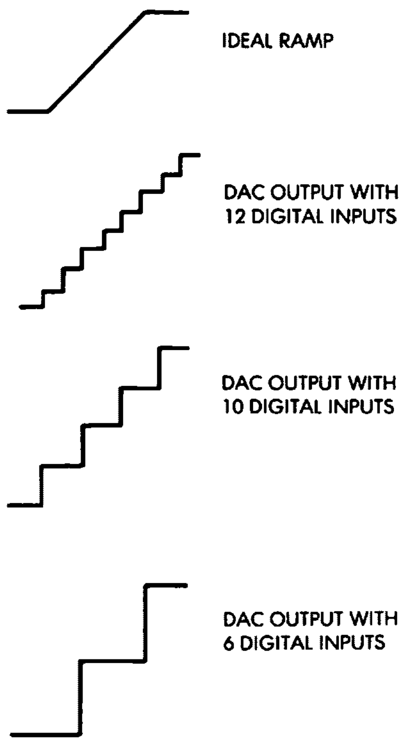


DIGITAL WORD INPUT									
D0	0	1	0	1	0	1	1	0	1
D1	0	0	1	1	0	0	0	1	1
D2	0	0	0	0	1	1	1	1	1
D3	0	0	0	0	0	0	1	1	1
D4	0	0	0	0	0	0	1	1	1
D5	0	0	0	0	0	0	1	1	1
D6	0	0	0	0	0	0	1	1	1
D7	0	0	0	0	0	0	1	1	1
D8	0	0	0	0	0	0	1	1	1
D9	0	0	0	0	0	0	1	1	1

The relationship between the digital input value and the analog output voltage for a DAC with 10 inputs and an output voltage swing between 0.00 and +10.00 volts.

Figure 9.6

computer I/O slot. This DAC has built-in latches that allow the data output from the Apple computer to be strobed and electrically stored without the use of an external latch. (See Figure 9.9.) This is a very helpful feature of the AD558, because it reduces the total number of parts necessary. We will show exactly how to connect the data lines of the Apple computer to the DAC digital inputs in a later section of this chapter.

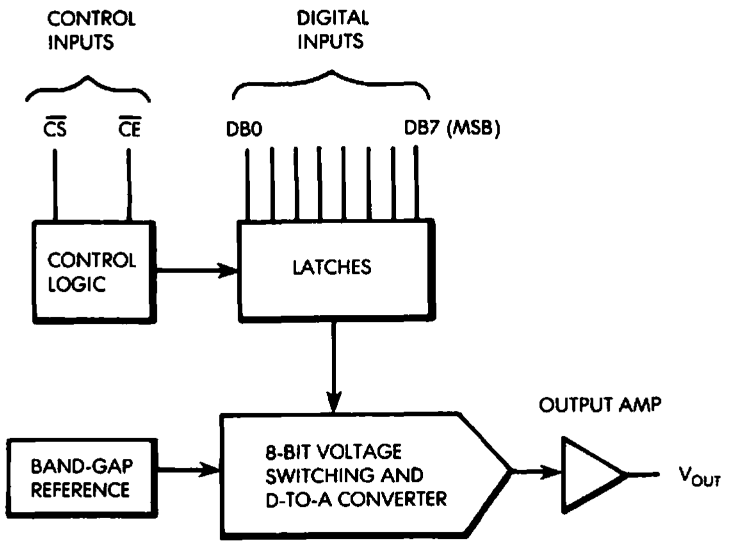

IDEAL RAMP
DAC OUTPUT WITH 12 DIGITAL INPUTS
DAC OUTPUT WITH 10 DIGITAL INPUTS
DAC OUTPUT WITH 6 DIGITAL INPUTS

Different "staircase" wave forms for the analog voltage output of the DAC. As the digital input word changes by a single count, the output will increase by the minimum voltage change. Notice that as the number of digital inputs to the DAC increases, the staircase more closely resembles the ideal ramp.

Figure 9.7

The next block shown in Figure 9.8 is the *control logic*. This logic block will provide the strobe signal to the input storage latches at the correct time. The Apple computer will output a strobe pulse to the control logic block.

Another important block of Figure 9.8 is the *band-gap reference*. This is the internal reference voltage for the DAC. With the reference voltage inside the package, we do not have to provide an



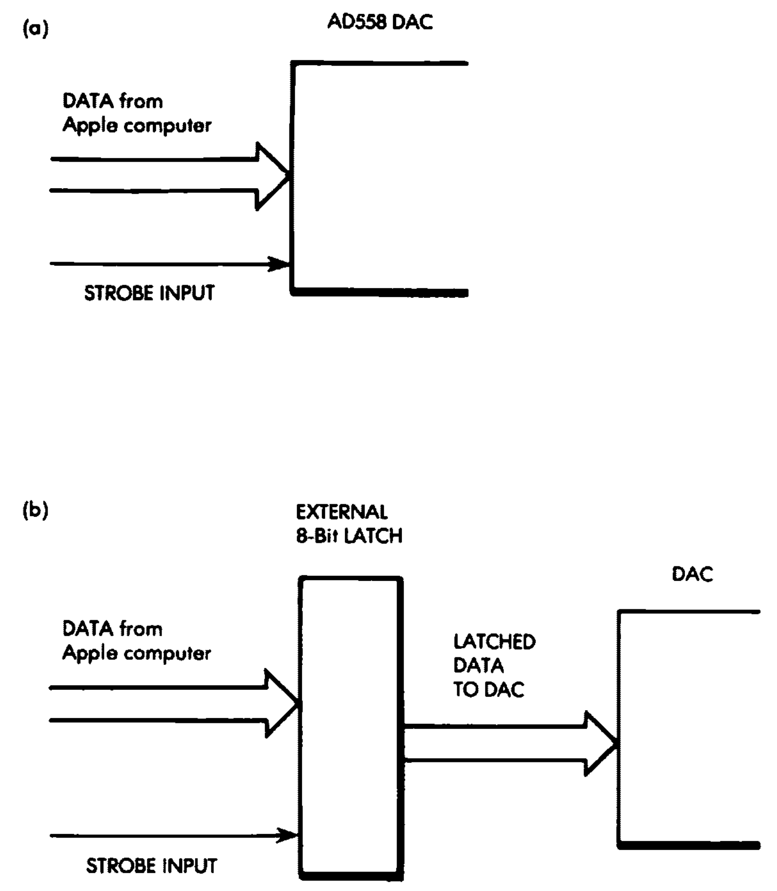
Block diagram for the AD558 digital-to-analog converter. Compare this to the general block diagram given in Figure 9.2.

Figure 9.8

external precision reference for the DAC. This is useful because it means that standard power supplies may be used to power the device. Most digital electronic systems, such as the Apple computer, have power supplies to suit this application.

A large block of Figure 9.8 is the *8-bit voltage switching and D-to-A converter.* This block is the scaling network that applies a portion of the reference voltage to the output. In this device the voltage is output to the outside world from an amplifier, labeled "output amp" in Figure 9.8. At this point in the discussion you should compare the block diagram of Figure 9.8 to the general block diagram that was given in Figure 9.2.

There are two modes of operation for the AD558. One mode will allow the output voltage to swing between 0.00 volts and +2.56 volts. The second mode allows the output voltage to swing between

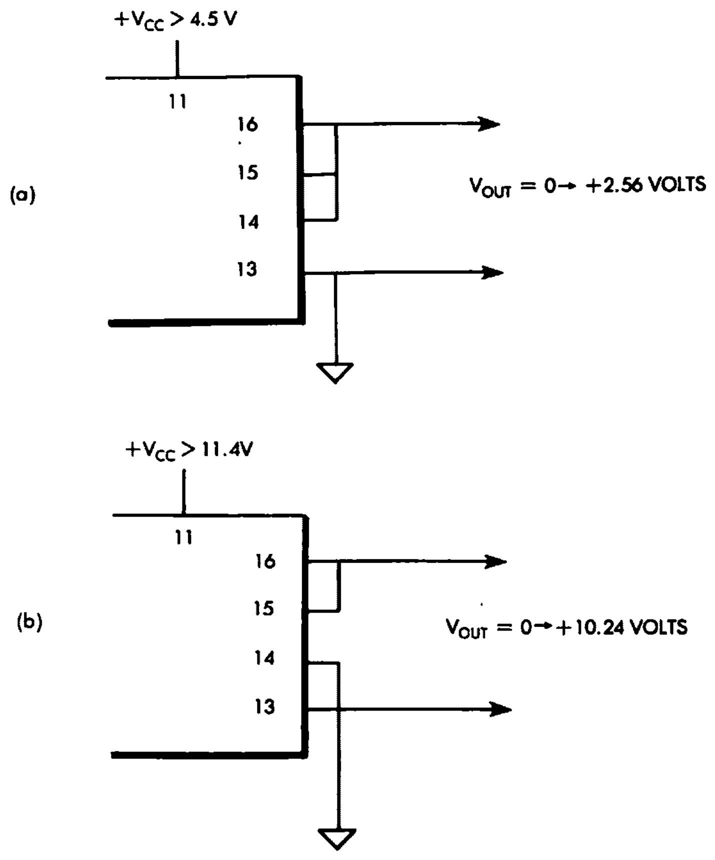


a) The AD558 will connect directly to the Apple computer data lines. There is an internal 8-bit storage latch on the device.

b) Other DACs may require an external storage latch to capture the logical state of the Apple data lines at exactly the correct time.

Figure 9.9

0.00 volts and +10.24 volts. Figure 9.10 shows how to connect the pins of the AD558 for an output voltage swing in each of the two modes. The 10.24-volt mode requires the V_{CC} input voltage pin to have a potential between 11.4 and 16.5 volts. The 2.56-volt mode requires a minimum potential of 4.6 volts.



(a)
$+V_{CC} > 4.5$ V
11
16
15
14
13
$V_{OUT} = 0 \rightarrow +2.56$ VOLTS
(b)
$+V_{CC} > 11.4$V
11
16
15
14
13
$V_{OUT} = 0 \rightarrow +10.24$ VOLTS

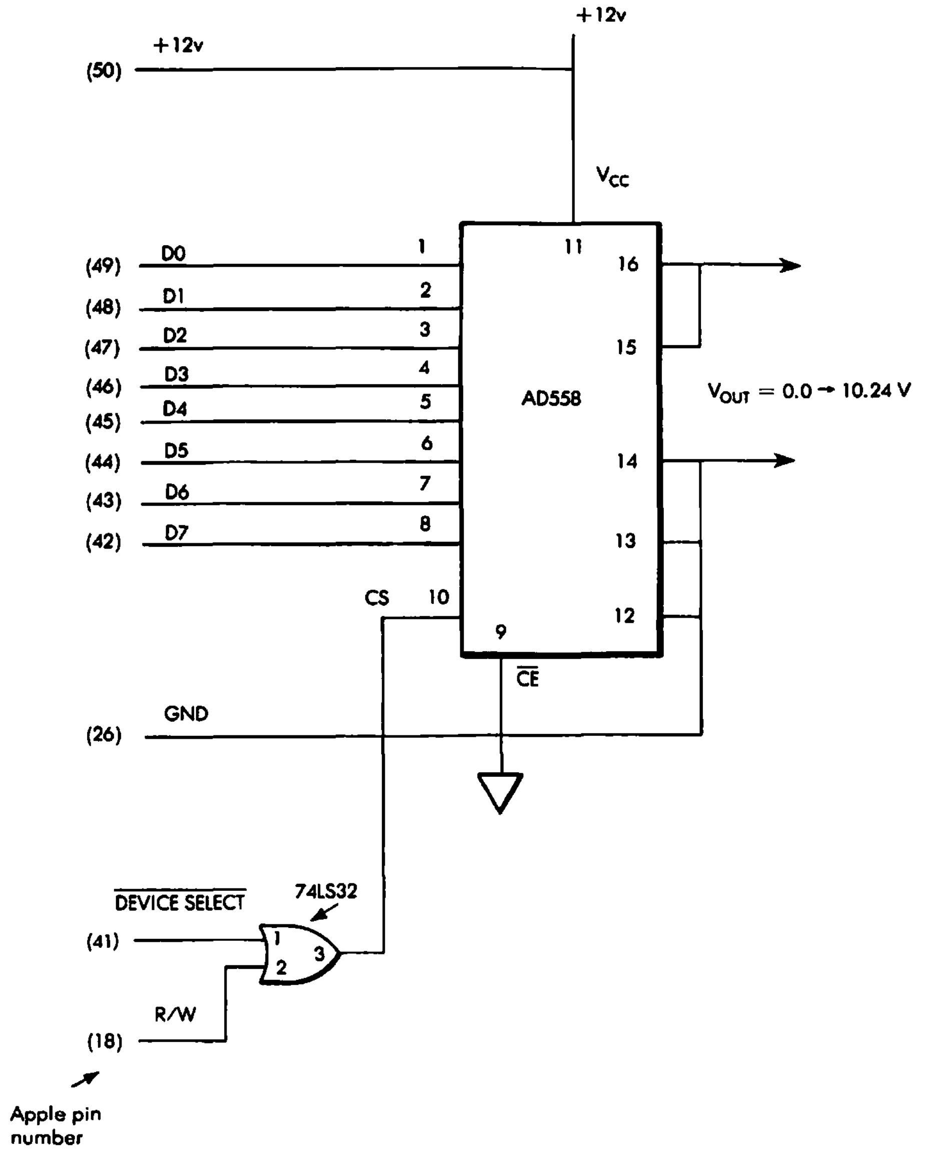
a) Wiring diagram for using the AD558 with a maximum output voltage of 2.56 volts.

b) Wiring diagram for using the AD558 with a maximum output voltage of 10.24 volts.

Figure 9.10

9.3: CONNECTING THE DAC TO THE APPLE

Let us now discuss how to connect the AD558 to the Apple computer for automatic control. This discussion will describe how to physically wire the AD558 to an Apple I/O slot. The information presented can be applied to connecting the Apple to most other DACs as well. Figure 9.11 shows the complete schematic



Complete schematic for interfacing the AD558 to the Apple computer at any I/O slot.

Figure 9.11

diagram for connecting the AD558 to an Apple I/O slot.

We see in Figure 9.11 that the data lines from the Apple computer, D0 – D7, are connected directly to the data input pins of the device. We can do this because of the AD558's built-in 8-bit latch, which will store the logical conditions of the data lines at the correct time. Other DACs may not have this feature. Digital-to-analog converters without built-in storage latches require an external 8-bit latch, used as shown in Figure 9.12.

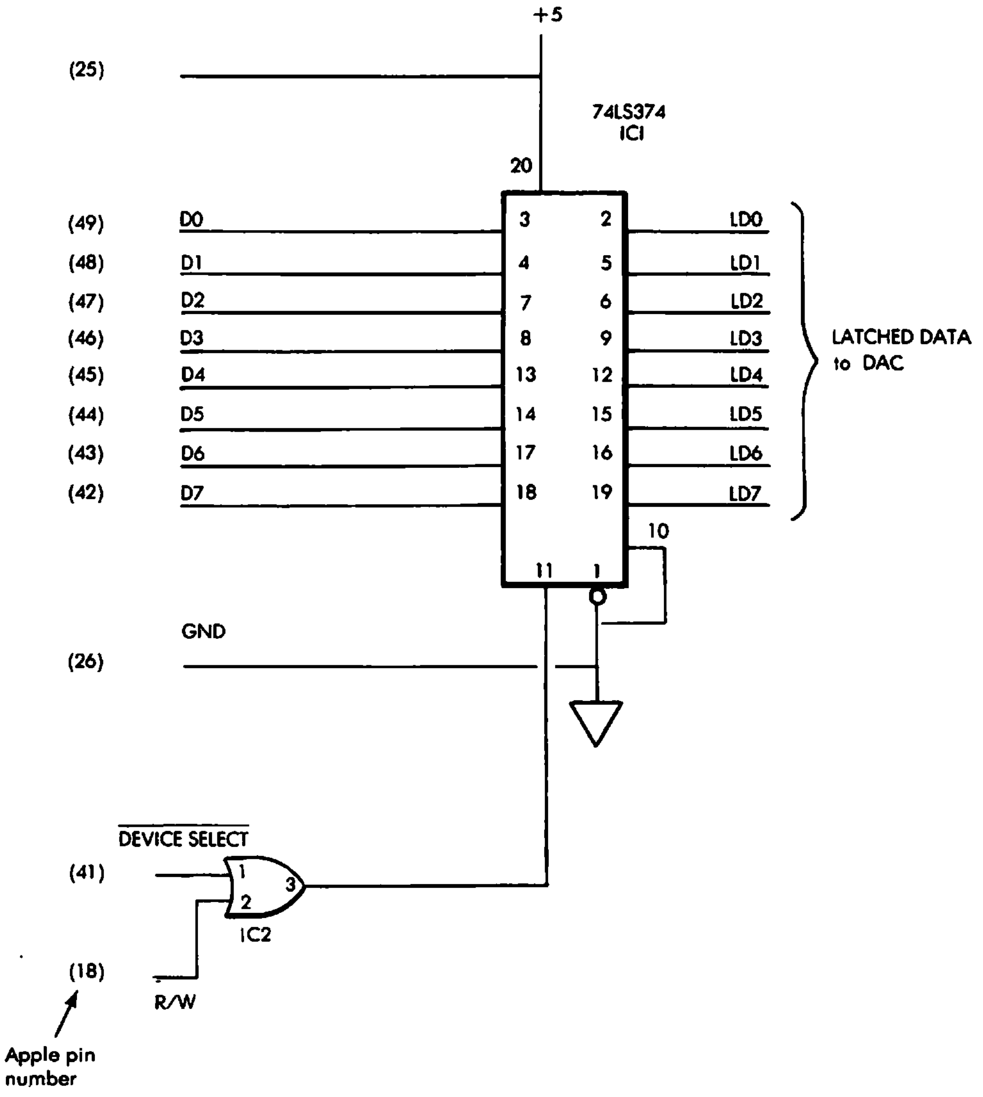
To strobe the data into the DAC circuit shown in Figure 9.11 or 9.12, the Apple computer will execute a POKE instruction. The use of the POKE instruction was described in Chapter 2. Timing considerations for latching the data during a POKE instruction were discussed in Chapter 4. When the POKE instruction is executed, the data specified by the instruction will be placed at the AD558 data inputs via the eight Apple data lines.

At this time the $\overline{\text{DEVICE SELECT}}$ line, as well as the R/W control line, will go to a logical 0. When this occurs, the CS input pin 10 of the AD558 will be set to a logical 0. After the POKE instruction is complete, the CS input line to the AD558 will be set to a logical 1, because the $\overline{\text{DEVICE SELECT}}$ line will go to a logical 1. The operation of the $\overline{\text{DEVICE SELECT}}$ and R/W lines was discussed in Chapter 4.

When the CS input line to the AD558 goes to a logical 1, the data at the input pins will be strobed into the internal 8-bit latch. The output voltage at pin 16 will depend on the data written to the DAC during the execution of the POKE instruction. By using the POKE instruction and the circuit shown in Figure 9.11, we can set any digital input combination between 0 and 255 at the DAC input pins. Now we need to know how to calculate exactly what data should be written to the DAC.

9.4: SETTING ANY OUTPUT VOLTAGE ON THE DAC

We will assume that the Apple computer system can control the digital inputs to the DAC as discussed in Section 9.3. The DAC will be physically connected to the system as shown in Figure 9.11. If the Apple computer can control the digital inputs to the DAC, how can we calculate the digital byte needed to set



Schematic diagram showing how an external latch could be connected to the Apple computer for use with a DAC that, unlike the AD558, does not have an internal storage latch.

Figure 9.12

the correct voltage output? That is the question we will answer in this section.

In Section 9.1 we discussed briefly how to calculate the minimum voltage change that will occur at the output of the DAC. That discussion assumed there were 10 digital inputs to the device, but the AD558 has only 8 inputs. Therefore, the equations given in Section 9.1 will need to be modified slightly to handle the differences in the two examples. We could interface a 10-bit DAC to the Apple computer, but it would require additional hardware, because the Apple output data bus handles only 8 bits at a time. If a 10-bit DAC were used, you would have to program the device using two 8-bit words, or POKE instructions—eight bits from one word and two bits from the other. Each 8-bit word would need to be latched at the DAC inputs. Further, DACs or ADCs handling any number of bits can be used with the Apple computer. We have used 8-bit DACs and ADCs here to illustrate the concepts because these devices interface directly to the Apple's 8-bit data bus.

Our objective in this section is to calculate the total weight of the digital inputs required to output a certain voltage from the DAC. For example, suppose we wish the DAC's output voltage to be equal to 4.8 volts. The computer will be programmed to output a certain combination of logical 1s and 0s to obtain the correct DAC output voltage. The digital output byte in this case would be equal to 120. Let us go through the steps needed to perform this calculation. We will be using the AD558 DAC as the example. However, you can easily modify the equations given to fit any DAC with a different number of digital inputs and a different output voltage swing.

The first step is to calculate the minimum voltage change at the DAC output. This may be accomplished using the procedure outlined in Section 9.1, and the appropriate specifications for the AD558.

$$\text{Minimum voltage swing} = \frac{\text{Maximum voltage swing}}{\begin{array}{c}\text{Total number of digital}\\ \text{input combinations} - 1\end{array}}$$

The maximum voltage swing on the AD558 will be equal to 10.24 volts. There are eight digital inputs to the AD558. (That is why it is called an *8-bit* DAC.) With eight digital inputs there are a total of

256 different input combinations. One of these combinations is used to set the DAC to its lowest value, so we use 255 (256 − 1) combinations to generate the output voltage swing.

> Minimum voltage swing = 10.24 / 255 = .040 volts, or 40 millivolts.

This means that if the digital input byte to the DAC were 00000001, the output voltage would be 40 millivolts.

Given the digital input word, we can calculate the DAC output voltage by the equation:

$$V_{OUT} = (\text{Digital input word}) \times .040 \text{ volts}$$

For example, let us suppose we have a digital input word equal to 65. The output voltage is calculated as 65 × .040 = 2.60 volts.

However, we are interested in the converse of the preceding case: Given a voltage needed, what digital input word is required by the DAC? To calculate this we use the relationship between the voltage desired and the known voltage output change for each digital input change. The analog output voltage for the DAC will change .040 volts for each digital count. Therefore, we can divide the voltage needed or desired at the DAC output by .040. The result will be the digital value needed to produce the output voltage.

For example, suppose we wished the DAC output voltage to be equal to 8.00 volts. To find out what digital word is needed by the DAC we set up the relationship:

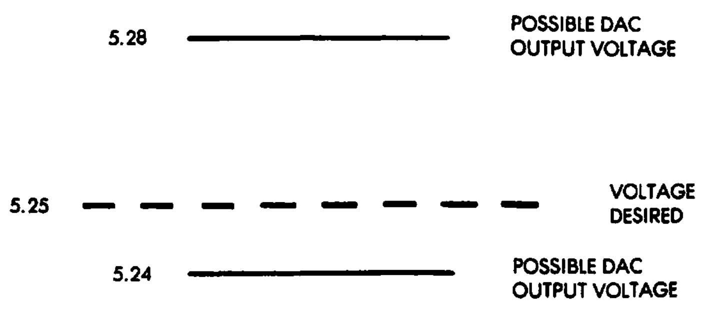
$$\text{Digital word} = \frac{V_{OUT}\text{ wanted}}{.040}$$

In our example the equation would be:

$$\text{Digital word} = \frac{8.00}{.040} = 200$$

If we input the digital weight of 200 to the DAC data input lines, the DAC output voltage will be equal to 8.00 volts. This type of calculation is quite easy to do with the Apple computer. We will present a program in the next section that will perform this calculation.

Let us suppose that we wished the DAC to output a voltage equal to 5.25 volts. To calculate the digital word we would need to



The desired output voltage is 5.25 volts. The DAC will output a voltage of either 5.24 volts or 5.28 volts. We must choose which of these two output voltages is the most desirable for the application.

Figure 9.13

use the equation:

$$\text{Digital word} = \frac{5.25}{.040} = 131.25$$

Notice that the digital word is not an integer, but a real number. That is, the number has some digits that are not zero after the decimal point. To control the DAC with the Apple, we must output digital words as integers: 1, 2, 234, 179, etc. The exact voltage we require at the output of the DAC cannot be obtained. We must choose whether we wish the output voltage to be a little larger than 5.25 or a little less than 5.25. (See Figure 9.13.)

The integer numbers that we must choose between are 131 and 132. Neither of these two numbers will give the exact output voltage of 5.25 volts. The number 131 will give an output voltage equal to $131 \times .040 = 5.24$ volts. The number 132 will give an output voltage equal to $132 \times .040 = 5.28$ volts. It is clear that the integer 131 gives an output voltage value closer to 5.25 volts than does 132.

As a general rule, we may round off to the closest whole number and use the rounded-off value as the digital word to be output. If the decimal part of the required digital word is less than but not equal to .5, use the smaller integer. If the decimal part of the digital word is greater than or equal to .5, use the greater integer as the output word.

To illustrate, let us suppose the required digital word was equal to 156.34. The decimal part of this word is equal to .34 so we will use the smaller integer, 156, as the output word. If the required digital word was equal to 156.67, we would use the greater integer, 157, as the output word. This process, testing the decimal value and outputting the nearest integer, can be accomplished quite easily with the Apple computer, using BASIC's INT function. The program given in the next section, which calculates the digital value needed, uses this function.

The same type of calculations for the digital input word and the rounding-off of the word to integer values may be applied to a DAC with any number of input lines and any output voltage swing.

9.5: CONTROLLING THE DAC WITH A BASIC PROGRAM

In this section we will present a BASIC program that will allow the AD558 to be programmed to any output voltage between 0.00 and 10.24 volts. The program will perform the calculations described in Section 9.5, using the information presented there. The program is shown in Figure 9.14.

```
 10  REM: THIS PROGRAM WILL INPUT A NUMBER FROM THE KEYBOARD
 20  REM: AND POKE THE FORMATTED DATA TO THE I/O SLOT.
 30  REM: WE ASSUME THE DAC BOARD OF FIGURE 9.11 IS CONNECTED
 40  REM: TO I/O SLOT 4. THIS REQUIRES A POKE ADDRESS OF
 50  REM: -16192.
 60  REM
 70  REM
 80  PRINT"INPUT THE VOLTAGE FOR THE DAC"
 90  PRINT"IT MUST BE BETWEEN 0 AND 10.24 VOLTS"
100  PRINT
110  INPUT V1
120  REM
```

Figure 9.14

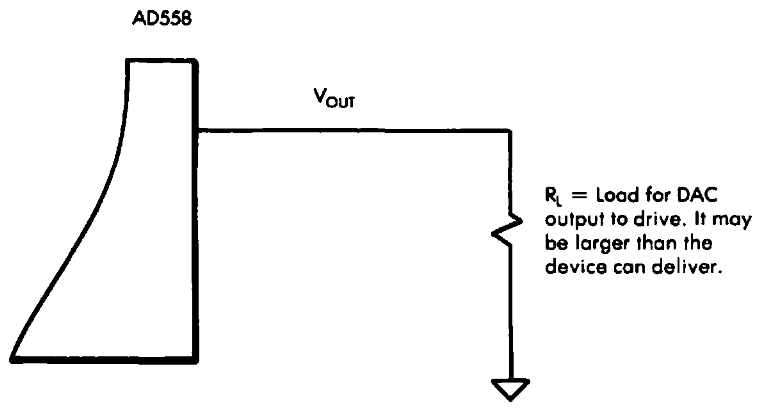
```
 130  REM: NOW TO CHECK FOR A VALID INPUT VOLTAGE
 140  REM
 150  IF V1 < 0 OR V1 > 10.24 GOTO 1000
 160  REM
 170  REM: NOW TO CALCULATE THE DIGITAL WORD FOR THE DAC
 180  REM
 190  LET X= V1/.040
 200  REM
 210  REM: NOW TO ROUND OFF THE NUMBER
 220  REM
 230  LET X=INT(X+.5)
 240  REM
 250  REM: NOW TO OUTPUT THE WORD TO THE DAC
 260  REM
 270  POKE -16192,X
 280  REM
 290  REM: THE DAC VOLTAGE IS NOW SET. LOOP BACK TO START
 300  REM
 310  GOTO 80
 800  REM
 810  REM: THIS SECTION PRINTS AN ERROR STATEMENT FOR A
 820  REM: VOLTAGE THAT WAS NOT BETWEEN 0.00 AND +10.24
 830  REM
1000  PRINT
1010  PRINT"THE VOLTAGE "V1;" WAS NOT BETWEEN 0.00 AND 10.24 V"
1020  PRINT
1030  GOTO 80
```

This program will calculate and output the digital value necessary for the DAC to produce a given voltage.

Figure 9.14 (cont.)

9.6: INCREASING THE OUTPUT DRIVE CAPABILITY OF THE DAC

If we want the DAC output voltage to drive a heavy current load, the current required may be greater than the rated amount


AD558
V_{OUT}
R_L = Load for DAC output to drive. It may be larger than the device can deliver.

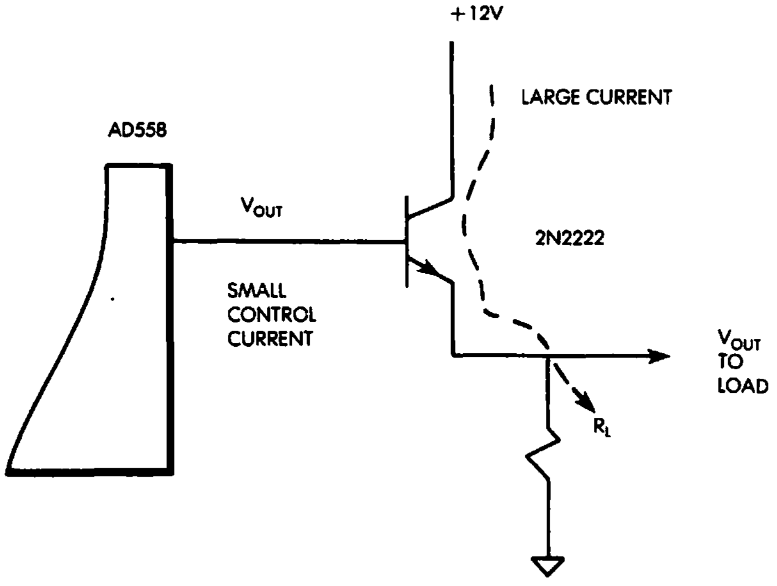
In some applications the DAC output voltage will be required to drive a load that needs more current than the DAC output pin can deliver. Usually, DACs will output only a few milliamperes of current to a load.

Figure 9.15

of the device. (See Figure 9.15.) The AD558 can output 5 milliamperes to a load with no electrical problems. In this section we will show how to increase the output current drive capability of the DAC from 5 milliamperes to several hundred milliamperes. In music applications the current required by the DAC is usually less than 5 milliamperes. For motor control applications, the current will be in the range from 100 to several hundred milliamperes.

One way to do this is by using the circuit shown in Figure 9.16. In this circuit we take advantage of the current gain of a transistor, the 2N2222, manufactured by many companies. A disadvantage of this technique is that we do not know exactly what the output voltage at the load will be, because some voltage will drop across the base-emitter junction of the transistor (see Figure 9.17).

Another technique that can be used to increase the current drive capability of the AD558 is shown in Figure 9.18. This circuit makes use of an operational amplifier (or op amp) in addition to the transistor to compensate for any voltage drop across the base-emitter junction. At this point we should briefly describe the



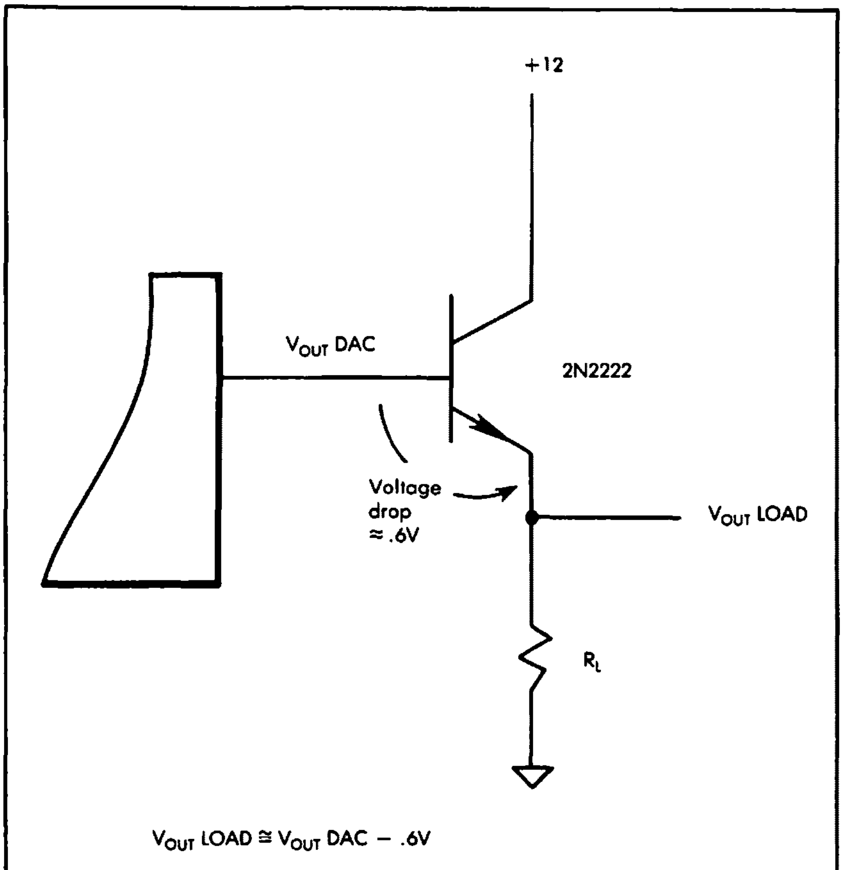
One way to increase the output drive current of the DAC is by using the circuit shown above. The heavy current is controlled by the smaller current in the base of the transistor. This base current is output by the DAC.

Figure 9.16

function of an operational amplifer. The output of the op amp will automatically adjust itself so the two inputs are at the same voltage. This will allow the emitter of the transistor to be at the desired voltage potential, and let the transistor handle the heavy current. This circuit is a current amplifier with a voltage gain of 1. The voltage at the load in Figure 9.17 will be equal to the output voltage of the DAC.

9.7: SUMMARY

In this chapter we began by explaining the nature of digital-to-analog conversion (DAC). We showed that the number of unique output voltages of a converter depends on the number of digital input lines and the converter's range of output voltage levels. Also shown in this chapter was a general block diagram of a DAC.



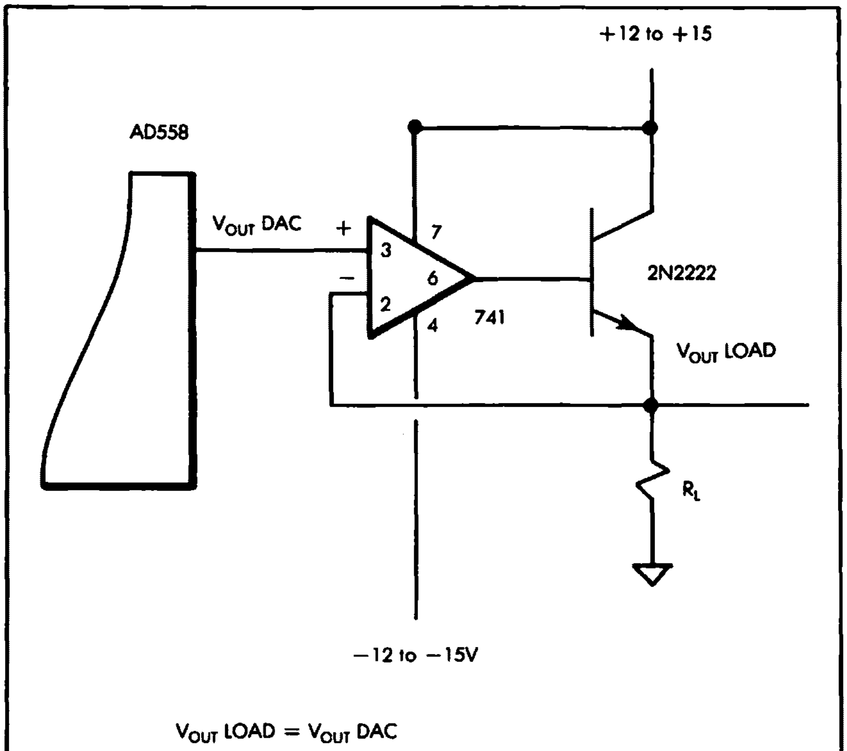
In the circuit of Figure 9.16, we do not know exactly what voltage will be delivered to the load, because of the voltage drop across the base-emitter junction of the transistor. The voltage drop is approximately .6 volts.

Figure 9.17

When you encounter a DAC for the first time, it will be helpful to keep in mind this general block diagram.

From this general discussion of the DAC, a formula was derived for calculating the minimum voltage change for a DAC with any number of digital inputs and any maximum output voltage swing. We then discussed how to determine what digital word to POKE to the DAC to set a desired output voltage.

The chapter finished by showing a sample program for controlling a DAC with the Apple computer and giving general schematics



Schematic diagram of a circuit that will increase the output current drive of the DAC and compensate for the base-emitter voltage drop.

Figure 9.18

for increasing the output drive capability of the device. Digital-to-analog converters are becoming quite popular in many home computer peripheral devices. If you understand the information given in this chapter, then using and controlling the peripheral devices that use DACs will be a much easier task.

Now that you have finished this text, the prospect of controlling external hardware with the Apple computer should seem a much simpler problem than it did at first. We have presented and discussed many examples of interfacing the computer to peripheral equipment. Important timing and control signals used by the Apple computer were clearly outlined, and you learned exactly how to use these signals. Throughout this text the two prevailing concepts were those stated in Chapter 1. That is, computer control is made up of hardware and software that will:

1. Send electrical information to an external device, and
2. Receive electrical information from an external device.

At this point you know enough about controlling peripheral equipment with the Apple computer to make your own Apple connection. This will open the door to the applications described in this text as well as many others you may be able to imagine. Good luck and have fun making the Apple connection.

9.8: FURTHER STUDY

In this text we have covered the basic topics involved in computer control with the Apple. If you are interested in going deeper into this subject, there are other topics which are important. (A good source to read for some of these topics is *Microprocessor Interfacing Techniques*, by R. Zaks and A. Lesea, Sybex, 3rd ed. 1979.)

One area we have not discussed which is used often in computer interfacing is that of *interrupts*. Using interrupts, the peripheral device can request service from the computer only when necessary. At all other times, the computer will perform other tasks and will not service the external device at all.

Direct memory access (DMA) is another area that may be useful for further study. DMA is an electrical mode in which the computer's internal microprocessor is electrically removed from the system, allowing the peripheral device to control the system. In this way, the peripheral device can directly access the system's memory circuits without first going through the computer's central processing unit.

The last topic of study we should mention in the context of computer control is the use of different *standard interface buses* with external devices. Examples of these standard buses are *IEEE-488* and *RS-232*. These buses will allow you to directly connect your computer to a peripheral device with no hardware modifications or special designs, which enables you to concentrate on software development.

These are some of the main topics you may want to study now that you are familiar with the essential elements of interfacing and computer control.

Appendix A

Appendix A:

Manufacturers' Data Sheets

- 74LS00 series integrated circuits
- LM135/LM235/LM335 temperature sensor
- AD570 analog-to-digital converter
- AD558 digital-to-analog converter

54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

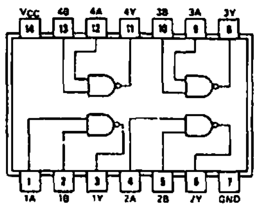
PIN ASSIGNMENTS (TOP VIEWS)

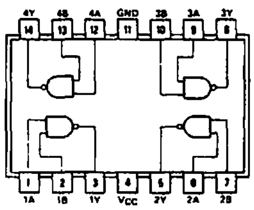
QUADRUPLE 2-INPUT
POSITIVE-NAND GATES

00

positive logic:
Y = $\overline{AB}$

See page 6-2





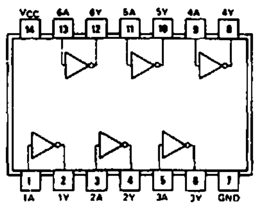
SN5400 (J)	SN7400 (J, N)	SN5400 (W)
SN54H00 (J)	SN74H00 (J, N)	SN54H00 (W)
SN54L00 (J)	SN74L00 (J, N)	SN54L00 (T)
SN54LS00 (J, W)	SN74LS00 (J, N)	
SN54S00 (J, W)	SN74S00 (J, N)	

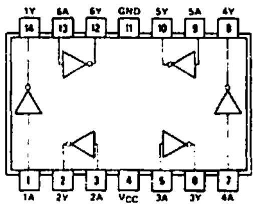
HEX INVERTERS

04

positive logic:
Y = $\overline{A}$

See page 6-2





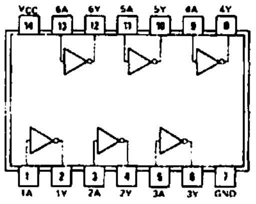
SN5404 (J)	SN7404 (J, N)	SN5404 (W)
SN54H04 (J)	SN74H04 (J, N)	SN54H04 (W)
SN54L04 (J)	SN74L04 (J, N)	SN54L04 (T)
SN54LS04 (J, W)	SN74LS04 (J, N)	
SN54S04 (J, W)	SN74S04 (J, N)	

HEX INVERTER BUFFERS/DRIVERS
WITH OPEN-COLLECTOR
HIGH-VOLTAGE OUTPUTS

06

positive logic:
Y = $\overline{A}$

See page 6-24



SN5406 (J, W) SN7406 (J, N)

54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

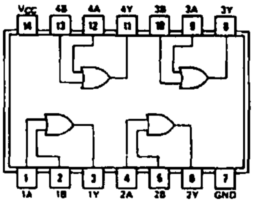
PIN ASSIGNMENTS (TOP VIEWS)

QUADRUPLE 2-INPUT POSITIVE-OR GATES

32

positive logic:
Y = A+B

See page 6-28



SN5432 (J, W) SN7432 (J, N)
SN54LS32 (J, W) SN74LS32 (J, N)
SN54S32 (J, W) SN74S32 (J, N)

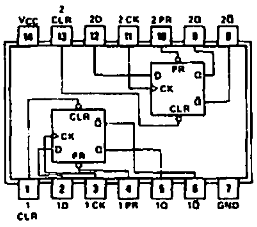
DUAL D-TYPE POSITIVE-EDGE-TRIGGERED FLIP-FLOPS WITH PRESET AND CLEAR

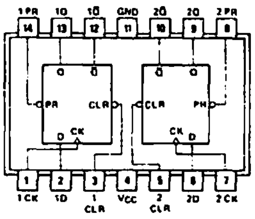
74

FUNCTION TABLE

INPUTS				OUTPUTS	
PRESET	CLEAR	CLOCK	D	Q	$\bar{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H*	H*
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\bar{Q}_0$

See pages 6-46, 6-50, 6-54, and 6-56





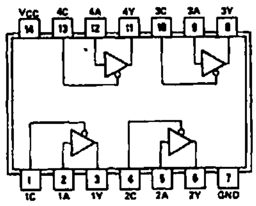
SN5474 (J) SN7474 (J, N) SN5474 (W)
SN54H74 (J) SN74H74 (J, N) SN54H74 (W)
SN54L74 (J) SN74L74 (J, N) SN54L74 (T)
SN54LS74A (J, W) SN74LS74A (J, N)
SN54S74 (J, W) SN74S74 (J, N)

QUADRUPLE BUS BUFFER GATES WITH THREE-STATE OUTPUTS

125

positive logic:
Y = A
Output is off (disabled) when C is high.

See page 6-33



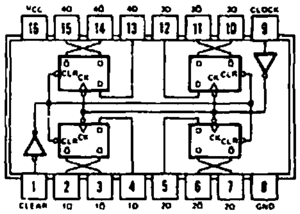
SN54125 (J, W) SN74125 (J, N)
SN54LS125A (J, W) SN74LS125A (J, N)

54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

PIN ASSIGNMENTS (TOP VIEWS)

QUAD D-TYPE FLIP-FLOPS

175 COMPLEMENTARY OUTPUTS
COMMON DIRECT CLEAR

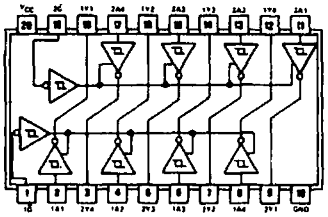


See page 7-253

SN54175 (J, W) SN74175 (J, N)
SN54LS175 (J, W) SN74LS175 (J, N)
SN54S175 (J, W) SN74S175 (J, N)

OCTAL BUFFERS/LINE DRIVERS/LINE RECEIVERS

240 INVERTED 3-STATE OUTPUTS

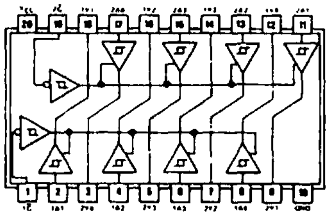


See page 6-83

SN54LS240 (J) SN74LS240 (J, N)
SN54S240 (J) SN74S240 (J, N)

OCTAL BUFFERS/LINE DRIVERS/LINE RECEIVERS

244 NONINVERTED 3-STATE OUTPUTS



See page 6-83

SN54LS244 (J) SN74LS244 (J, N)

From *The TTL Data Book for Design Engineers*

National Semiconductor

November 1980

LM135/LM235/LM335, LM135A/LM235A/LM335A Precision Temperature Sensors

General Description

The LM135 series are precision, easily-calibrated, integrated circuit temperature sensors. Operating as a 2-terminal zener, the LM135 has a breakdown voltage directly proportional to absolute temperature at +10 mV/°K. With less than 1Ω dynamic impedance the device operates over a current range of 400 μA to 5 mA with virtually no change in performance. When calibrated at 25°C the LM135 has typically less than 1°C error over a 100°C temperature range. Unlike other sensors the LM135 has a linear output.

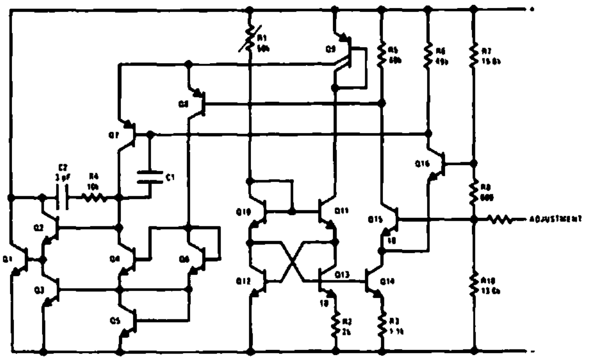
Applications for the LM135 include almost any type of temperature sensing over a −55°C to +150°C temperature range. The low impedance and linear output make interfacing to readout or control circuitry especially easy.

The LM135 operates over a −55°C to +150°C temperature range while the LM235 operates over a −40°C to +125°C temperature range. The LM335 operates from −40°C to +100°C. The LM135/LM235/LM335 are available packaged in hermetic TO-46 transistor packages while the LM335 is also available in plastic TO-92 packages.

Features

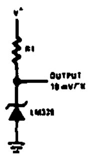
- Directly calibrated in °Kelvin
- 1°C initial accuracy available
- Operates from 400 μA to 5 mA
- Less than 1Ω dynamic impedance
- Easily calibrated
- Wide operating temperature range
- 200°C overrange
- Low cost

Schematic Diagram

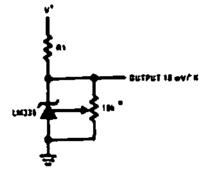


Typical Applications

Basic Temperature Sensor

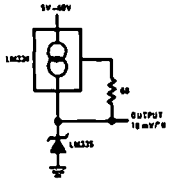


Calibrated Sensor



*Calibrate for 2.982V at 25°C

Wide Operating Supply



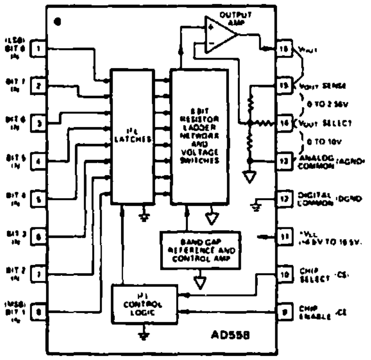
ANALOG DEVICES

DACPORT™ Low Cost Complete µP-Compatible 8-Bit DAC

AD558*

FEATURES
Complete 8-Bit DAC
Voltage Output – 2 Calibrated Ranges
Internal Precision Band-Gap Reference
Single-Supply Operation: +5V to +15V
Full Microprocessor Interface
Fast: 1µs Voltage Settling to ±1/2LSB
Low Power: 75mW
No User Trims
Guaranteed Monotonic Over Temperature
All Errors Specified T_{min} to T_{max}
Small 16-Pin DIP Package
Single Laser-Wafer-Trimmed Chip for Hybrids
Low Cost

AD558 FUNCTIONAL BLOCK DIAGRAM



TO-116

PRODUCT DESCRIPTION

The AD558 DACPORT is a complete voltage-output 8-bit digital-to-analog converter, including output amplifier, full microprocessor interface and precision voltage reference on a single monolithic chip. No external components or trims are required to interface, with full accuracy, an 8-bit data bus to an analog system.

The performance and versatility of the DACPORT is a result of several recently-developed monolithic bipolar technologies. The complete microprocessor interface and control logic is implemented with integrated injection logic (I^2L), an extremely dense and low-power logic structure that is process-compatible with linear bipolar fabrication. The internal precision voltage reference is the patented low-voltage band-gap circuit which permits full-accuracy performance on a single +5V to +15V power supply. Thin-film silicon-chromium resistors provide the stability required for guaranteed monotonic operation over the entire operating temperature range (all grades), while recent advances in laser-wafer-trimming of these thin-film resistors permit absolute calibration at the factory to within ±1LSB; thus no user-trims for gain or offset are required. A new circuit design provides voltage settling to ±1/2LSB for a full-scale step in 800ns.

The AD558 is available in four performance grades. The AD558J and K are specified for use over the 0 to +70°C temperature range, while the AD558S and T grades are specified for -55°C to +125°C operation. The hermetically-sealed ceramic package is standard. Processing to MIL-STD-883, Class B is optional on S and T grades.

PRODUCT HIGHLIGHTS

1. The 8-bit I^2L input register and fully microprocessor-compatible control logic allow the AD558 to be directly connected to 8- or 16-bit data buses and operated with standard control signals. The latch may be disabled for direct DAC interfacing.
2. The laser-trimmed on-chip SiCr thin-film resistors are calibrated for absolute accuracy and linearity at the factory. Therefore, no user trims are necessary for full rated accuracy over the operating temperature range.
3. The inclusion of a precision low-voltage band-gap reference eliminates the need to specify and apply a separate reference source.
4. The voltage-switching structure of the AD558 DAC section along with a high-speed output amplifier and laser-trimmed resistors give the user a choice of 0V to +2.56V or 0V to +10V output ranges, selectable by pin-strapping. Circuitry is internally compensated for minimum settling time on both ranges; typically settling to ±1/2LSB for a full-scale 2.55 volt step in 800ns.
5. The AD558 is designed and specified to operate from a single +4.5V to +16.5V power supply.
6. Low digital input currents, 100µA max, minimize bus loading. Input thresholds are TTL/low voltage CMOS compatible over the entire operating V_{CC} range.
7. The single-chip, low power I^2L design of the AD558 is inherently more reliable than hybrid multi-chip or conventional single-chip bipolar designs. The AD558S and T grades, which are specified over the -55°C to +125°C temperature range, are available processed to MIL-STD-883, Class B.
8. All AD558 grades are available in chip form with guaranteed specifications from +25°C to T_{max}. MIL-STD-883, Class B visual inspection is standard on Analog Devices bipolar chips. Contact the factory for additional chip information.

SPECIFICATIONS (typical @ T_A = +25°C, V_{CC} = +5V to +15V unless otherwise specified)

MODEL	AD558J	AD558K	AD558S[1]	AD558T[1]
RESOLUTION	8 Bits	•	•	•
RELATIVE ACCURACY[2]				
0 to +70°C	±1/2LSB max	±1/4LSB max	•	••
-55°C to +125°C	–	–	±3/4LSB max	±3/8LSB max
OUTPUT				
Ranges	0V to +2.56V	•	•	•
	0V to +10V [3]	•	•	•
Current, Source	+5mA	•	+5mA min	•••
Sink	Internal Passive Pull-Down to Ground[4]	•	•	•
OUTPUT SETTLING TIME[5]				
0 to 2.56 volt range	0.8μs (1.5μs max)	•	•	•
0 to 10 volt range[3]	2.0μs (3.0μs max)	•	•	•
FULL SCALE ACCURACY				
@ 25°C	±1.5LSB (±0.6%) max	±0.5LSB (±0.2%) max	•	••
T_{min} to T_{max}	±2.5LSB (±1.0%) max	±1LSB (±0.4%) max	•	••
ZERO ERROR				
@ 25°C	±1LSB max	±1/2LSB max	•	••
T_{min} to T_{max}	±2LSB max	±1LSB max	•	••
MONOTONICITY[6]				
T_{min} to T_{max}	Guaranteed	•	•	•
DIGITAL INPUTS				
T_{min} to T_{max}				
Input Current	±100μA max	•	•	•
Data Inputs, Voltage				
Bit On – Logic "1"	2.0V min	•	•	•
Bit Off – Logic "0"	0.8V max	•	•	•
Control Inputs, Voltage				
On – Logic "1"	2.0V min	•	•	•
Off – Logic "0"	0.8V max	•	•	•
Input Capacitance	4pF	•	•	•
TIMING[7]				
T_{min} to T_{max}				
t_W (Strobe Pulse Width)	100ns min	•	•	•
t_{DH} (Data Hold Time)	10ns min	•	•	•
t_{DS} (Data Set-Up Time)	100ns min	•	•	•
POWER SUPPLY				
Operating Voltage Range (V_{CC})				
2.56 Volt Range	+4.5V to +16.5V	•	•	•
10 Volt Range	+11.4V to +16.5V	•	•	•
Current (I_{CC})	15mA typ, 25mA max	•	•	•
Rejection Ratio	0.03%/% max	•	•	•
POWER DISSIPATION, V_{CC} = 5V	75mW (125mW max)	•	•	•
V_{CC} = 15V	225mW (375mW max)	•	•	•
OPERATING TEMPERATURE RANGE				
T_{min}	0°C	•	-55°C	•••
T_{max}	+70°C	•	+125°C	•••

Reprinted with permission of Analog Devices, Inc., Norwood MA 02062.

ABSOLUTE MAXIMUM RATINGS

V_{CC} to Ground	0V to +18V
Digital Inputs (Pins 1-10)	0 to +7.0V
V_{OUT}	Indefinite Short to Ground Momentary Short to V_{CC}
Power Dissipation	450mW
Storage Temperature Range	
D (ceramic) Package	-55°C to +150°C
Lead Temperature (soldering, 10 second)	300°C
Thermal Resistance	
Junction to Ambient/Junction to Case	
D (ceramic) Package	100/30°C/W

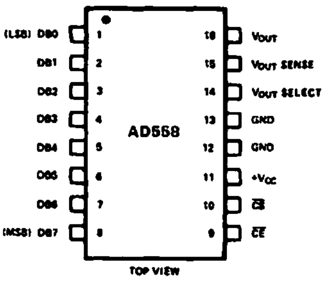


Figure 1. AD558 Pin Configuration

AD558 ORDERING GUIDE

Model	Package	Temperature	Relative Accuracy Error Max T_{min} to T_{max}	Full-Scale Error, Max T_{min} to T_{max}	Package Style[1]
AD558JN	Plastic	0 to +70°C	±1/2LSB	±2.5LSB	N16A[2]
AD558KN	Plastic	0 to +70°C	±1/4LSB	±1LSB	N16A[2]
AD558JD	Ceramic	0 to +70°C	±1/2LSB	±2.5LSB	D16A
AD558KD	Ceramic	0 to +70°C	±1/4LSB	±1LSB	D16A
AD558SD	Ceramic	-55°C to +125°C	±3/4LSB	±2.5LSB	D16A
AD558SD/883B	Ceramic	-55°C to +125°C	±3/4LSB	±2.5LSB	D16A
AD558TD	Ceramic	-55°C to +125°C	±3/8LSB	±1LSB	D16A
AD558TD/883B	Ceramic	-55°C to +125°C	±3/8LSB	±1LSB	D16A

[1] See Section 20 for package outline information.
[2] To be available June, 1982.

CIRCUIT DESCRIPTION

The AD558 consists of four major functional blocks, fabricated on a single monolithic chip (see Figure 2). The main D to A converter section uses eight equally-weighted laser-trimmed current sources switched into a silicon-chromium thin-film R/2R resistor ladder network to give a direct but unbuffered 0mV to 400mV output range. The transistors that form the DAC switches are PNPs; this allows direct positive-voltage logic interface and a zero-based output range.

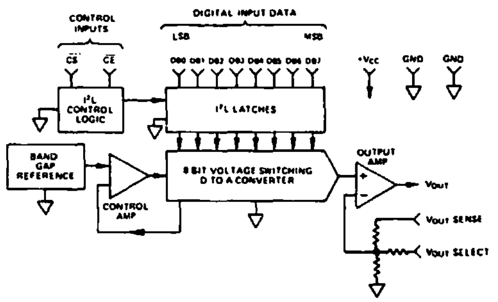


Figure 2. AD558 Functional Block Diagram

The high-speed output buffer amplifier is operated in the non-inverting mode with gain determined by the user-connections at the output range select pin. The gain-setting application resistors are thin-film laser-trimmed to match and track the DAC resistors and to assure precise initial calibration of the two output ranges, 0V to 2.56V and 0V to 10V. The amplifier output stage is an NPN transistor with passive pull-down for zero-based output capability with a single power supply.

The internal precision voltage reference is of the patented band-gap type. This design produces a reference voltage of 1.2 volts and thus, unlike 6.3 volt temperature-compensated zeners, may be operated from a single, low-voltage logic power supply. The microprocessor interface logic consists of an 8-bit data latch and control circuitry. Low-power, small geometry and high-speed are advantages of the I^2L design as applied to this section. I^2L is bipolar process compatible so that the performance of the analog sections need not be compromised to provide on-chip logic capabilities. The control logic allows the latches to be operated from a decoded microprocessor address and write signal. If the application does not involve a μP or data bus, wiring $\overline{CS}$ and $\overline{CE}$ to ground renders the latches "transparent" for direct DAC access.

AD558 Applications

CONNECTING THE AD558

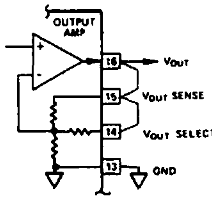
The AD558 has been configured for ease of application. All reference, output amplifier and logic connections are made internally. In addition, all calibration trims are performed at the factory assuring specified accuracy without user trims. The only connection decision that must be made by the user is a single jumper to select output voltage range. Clean circuit-board layout is facilitated by isolating all digital bit inputs on one side of the package; analog outputs are on the opposite side.

Figure 3 shows the two alternative output range connections. The 0V to 2.56V range may be selected for use with any power supply between +4.5V and +16.5V. The 0V to 10V range requires a power supply of +11.4V to +16.5V.

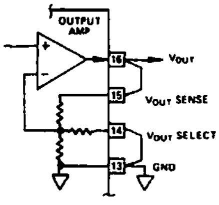
Because of its precise factory calibration, the AD558 is intended to be operated without user trims for gain and offset; therefore no provisions have been made for such user-trims. If a small increase in scale is required, however, it may be accomplished by slightly altering the effective gain of the output buffer. A resistor in series with V_{OUT} SENSE will increase the output range.

For example if a 0V to 10.24V output range is desired (40mV = 1LSB), a nominal resistance of 850Ω is required. It must be remembered that, although the internal resistors all ratio-match and track, the *absolute* tolerance of these resistors is typically ±20% and the *absolute* TC is typically -50ppm/°C (0 to -100ppm/°C). That must be considered when re-scaling is performed. Figure 4 shows the recommended circuitry for a full-scale output range of 10.24 volts. Internal resistance values shown are nominal.

NOTE: Decreasing the scale by putting a resistor in series with GND will not work properly due to the code-dependent currents in GND. Adjusting offset by injecting dc at GND is not recommended for the same reason.



a. 0V to 2.56V Output Range



b. 0V to 10V Output Range

Figure 3. Connection Diagrams

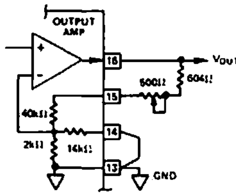


Figure 4. 10.24V Full-Scale Connection

GROUNDING AND BYPASSING*

All precision converter products require careful application of good grounding practices to maintain full rated performance. Because the AD558 is intended for application in microcomputer systems where digital noise is prevalent, special care must be taken to assure that its inherent precision is realized.

The AD558 has two ground (common) pins; this minimizes ground drops and noise in the analog signal path. Figure 5 shows how the ground connections should be made.

It is often advisable to maintain separate analog and digital grounds throughout a complete system, tying them common in one place only. If the common tie-point is remote and accidental disconnection of that one common tie-point occurs due to card removal with power on, a large differential voltage between the two commons could develop. To protect devices that interface to both digital and analog parts of the system, such as the AD558, it is recommended that common ground tie-points should be provided at *each* such device. If only one system ground can be connected directly to the AD558, it is recommended that analog common be selected.

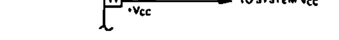


Figure 5. Recommended Grounding and Bypassing

POWER SUPPLY CONSIDERATIONS

The AD558 is designed to operate from a single positive power supply voltage. Specified performance is achieved for any supply voltage between +4.5V and +16.5V. This makes the AD558 ideal for battery-operated, portable, automotive or digital main-frame applications.

The only consideration in selecting a supply voltage is that, in order to be able to use the 0V to 10V output range, the power supply voltage must be between +11.4V and +16.5V. If, however, the 0V to 2.56V range is to be used, power consumption will be minimized by utilizing the lowest available supply voltage (above +4.5V).

TIMING AND CONTROL
The AD558 has data input latches that simplify interface to 8- and 16-bit data buses. These latches are controlled by Chip Enable ($\overline{CE}$) and Chip Select ($\overline{CS}$) inputs, pins 9 and 10 respectively. $\overline{CE}$ and $\overline{CS}$ are internally "NORed" so that the latches transmit input data to the DAC section when both $\overline{CE}$ and $\overline{CS}$ are at Logic "0". If the application does not involve a data bus, a "00" condition allows for direct operation of the DAC. When either $\overline{CE}$ or $\overline{CS}$ go to Logic "1", the input data is latched into the registers and held until both $\overline{CE}$ and $\overline{CS}$ return to "0". (Unused $\overline{CE}$ or $\overline{CS}$ inputs should be tied to ground.) The truth table is given in Table I. The logic function is also shown in Figure 6.

Input Data	$\overline{CE}$	$\overline{CS}$	DAC Data	Latch Condition
0	0	0	0	"transparent"
1	0	0	1	"transparent"
0	∫	0	0	latching
1	∫	0	1	latching
0	0	∫	0	latching
1	0	∫	1	latching
X	1	X	previous data	latched
X	X	1	previous data	latched

Notes: X = Does not matter
∫ = Logic Threshold at Positive-Going Transition

Table I. AD558 Control Logic Truth Table

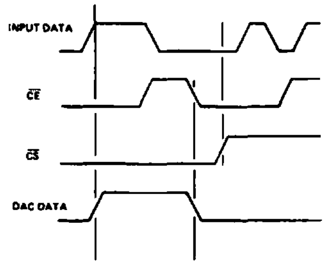


Figure 6. AD558 Control Logic Function

Figure 7 shows the timing for the data and control signals; $\overline{CE}$ and $\overline{CS}$ are identical in timing as well as in function.

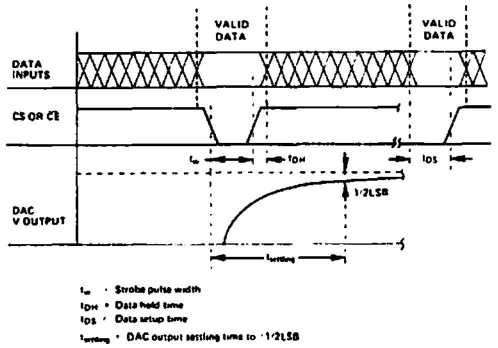
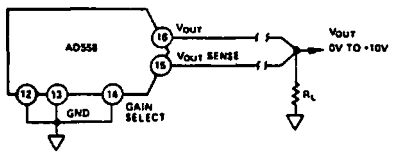
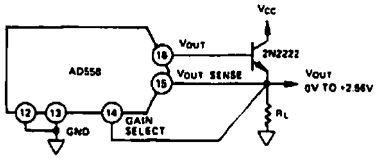

VALID DATA
VALID DATA
DATA INPUTS
$\overline{CS}$ OR $\overline{CE}$
t_W
t_{DH}
t_{DS}
1/2LSB
DAC V OUTPUT
$t_{settling}$
t_W = Strobe pulse width
t_{DH} = Data hold time
t_{DS} = Data setup time
$t_{settling}$ = DAC output settling time to ±1/2LSB

Figure 7. AD558 Timing

USE OF V_{OUT} SENSE
Separate access to the feedback resistor of the output amplifier allows additional application versatility. Figure 8a shows how I X R drops in long lines to remote loads may be cancelled by putting the drops "inside the loop". Figure 8b shows how the separate sense may be used to provide a higher output current by feeding back around a simple current booster.



a. Compensation for I x R Drops in Output Lines



b. Output Current Booster

Figure 8. Use of V_{OUT} Sense

Applying the AD558

OPTIMIZING SETTLING TIME

In order to provide single-supply operation and zero-based output voltage ranges, the AD558 output stage has a passive "pull-down" to ground. As a result, settling time for negative-going output steps may be longer than for positive-going output steps. The relative difference depends on load resistance and capacitance. If a negative power supply is available, the negative-going settling time may be improved by adding a pull-down resistor from the output to the negative supply as shown in Figure 9. The value of the resistor should be such that, at zero voltage out, current through that resistor is 0.5mA max.

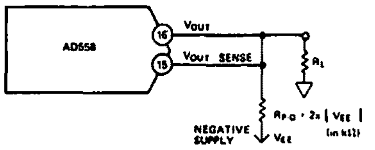
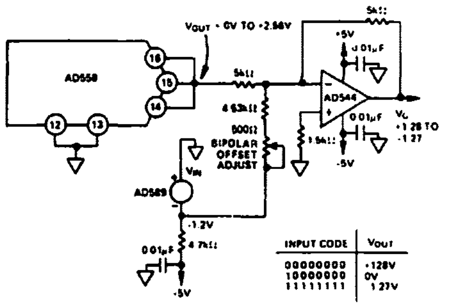


Figure 9. Improved Settling Time

BIPOLAR OUTPUT RANGES

The AD558 was designed for operation from a single power supply and is thus capable of providing only unipolar (0V to +2.56 and 0V to 10V) output ranges. If a negative supply is available, bipolar output ranges may be achieved by suitable output offsetting and scaling. Figure 10 shows how a ±1.28 volt output range may be achieved when a -5 volt power supply is available. The offset is provided by the AD589 precision 1.2 volt reference which will operate from a +5 volt supply. The AD544 output amplifier can provide the necessary ±1.28 volt output swing from ±5 volt supplies. Coding is complementary offset binary.

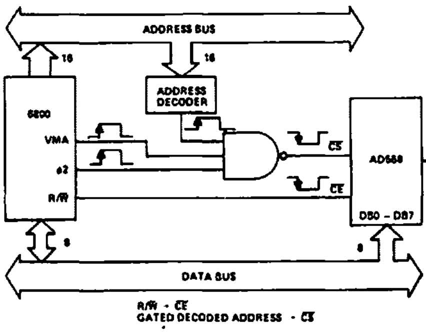


INPUT CODE	VOUT
00000000	+1.28V
10000000	0V
11111111	-1.27V

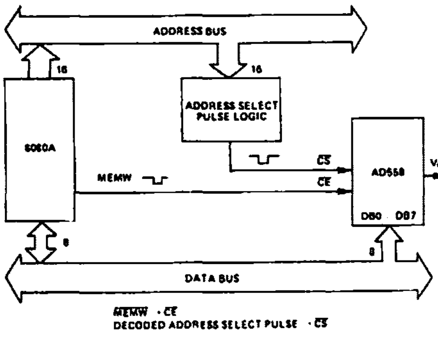
Figure 10. Bipolar Operation of AD558 from ±5V Supplies

INTERFACING THE AD558 TO MICROPROCESSOR DATA BUSES*

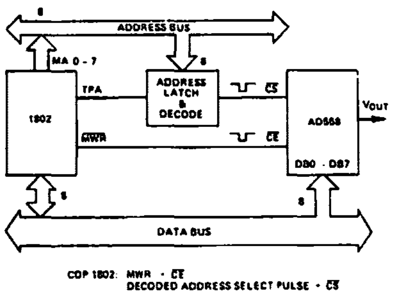
The AD558 is configured to act like a "write only" location in memory that may be made to coincide with a read only memory location or with a RAM location. The latter case allows data previously written into the DAC to be read back later via the RAM. Address decoding is partially complete for either ROM or RAM. Figure 11 shows interfaces for three popular microprocessor systems.



a. 6800/AD558 Interface



b. 8080A/AD558 Interface



c. 1802/AD558 Interface

Figure 11. Interfacing the AD558 to Microprocessors

*The microprocessor-interface capabilities of the AD558 are extensive. A comprehensive application note, "Interfacing the AD558 DACPORT™ to Microprocessors" is available from any Analog Devices Sales Office upon request, free of charge.

AD558 Performance (typical @ +25°C, V_{CC} = +5V to +15V unless otherwise noted)

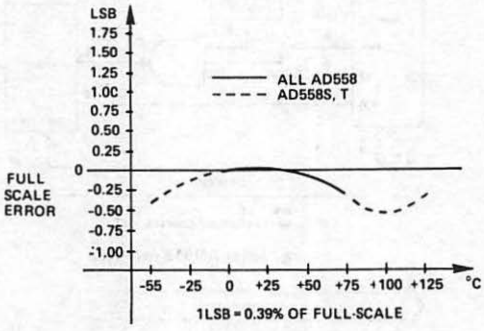


Figure 12. Full Scale Accuracy vs. Temperature Performance of AD558

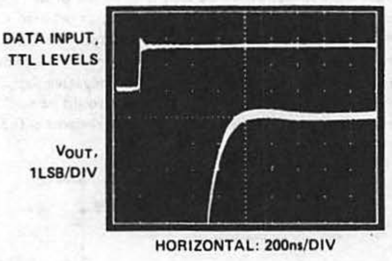


Figure 15. AD558 Settling Characteristic Detai 0V to 2.56V Output Range Full-Scale Step

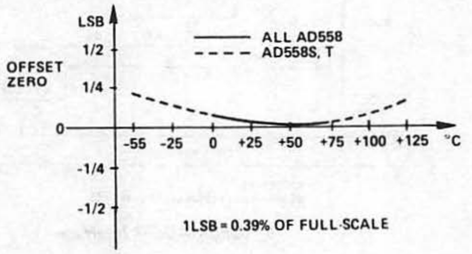


Figure 13. Zero Drift vs. Temperature Performance of AD558

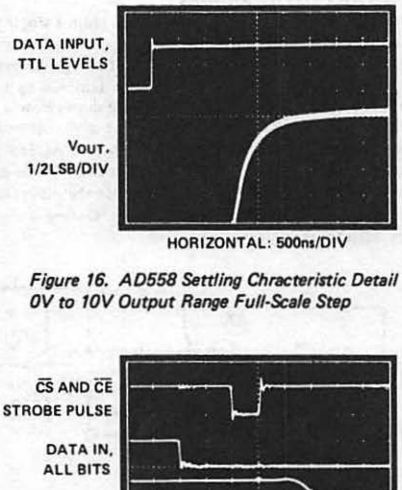


Figure 16. AD558 Settling Chracteristic Detail 0V to 10V Output Range Full-Scale Step

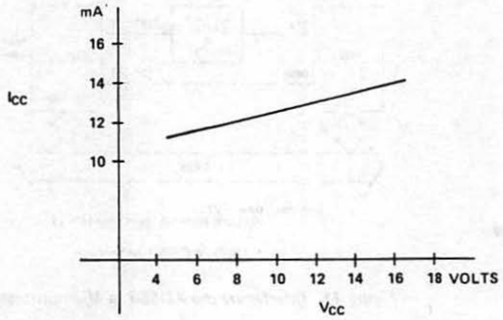


Figure 14. Quiescent Current vs. Power Supply Voltage for AD558

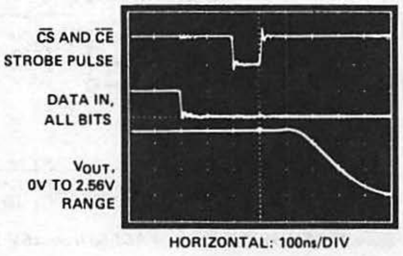


Figure 17. AD558 Logic Timing

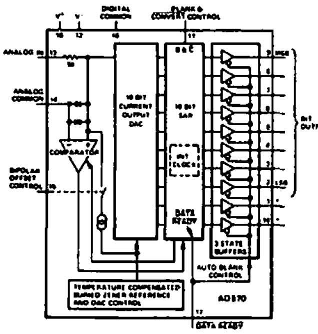
ANALOG DEVICES

Low Cost, Complete IC 8-Bit A to D Converter

AD570*

FEATURES
Complete A/D Converter with Reference and Clock
Fast Successive Approximation Conversion – 25μs
No Missing Codes Over Temperature
0 to +70°C – AD570J
-55°C to +125°C – AD570S
Digital Multiplexing – 3 State Outputs
18-Pin DIP
Low Cost Monolithic Construction

AD570 FUNCTIONAL BLOCK DIAGRAM



18-PIN DUAL IN LINE PACKAGE

PRODUCT DESCRIPTION

The AD570 is an 8-bit successive approximation A/D converter consisting of a DAC, voltage reference, clock, comparator, successive approximation register and output buffers – all fabricated on a single chip. No external components are required to perform a full accuracy 8-bit conversion in 25μs.

The AD570 incorporates the most advanced integrated circuit design and processing technology available today. I^2L (integrated injection logic) processing in the fabrication of the SAR function along with laser trimming of the high stability SiCr thin film resistor ladder network at the wafer stage (LWT) and a temperature compensated, subsurface Zener reference insures full 8-bit accuracy at low cost.

Operating on supplies of +5V and -15V, the AD570 will accept analog inputs of 0 to +10V unipolar or ±5V bipolar, externally selectable. As the BLANK and $\overline{\text{CONVERT}}$ input is driven low, the three state outputs will be open and a conversion will commence. Upon completion of the conversion, the $\overline{\text{DATA READY}}$ line will go low and the data will appear at the output. Pulling the BLANK and $\overline{\text{CONVERT}}$ input high blanks the outputs and readies the device for the next conversion. The AD570 executes a true 8-bit conversion with no missing codes in approximately 25μs.

The AD570 is available in two versions; the AD570J is specified for the 0 to 70°C temperature range, the AD570S for -55°C to +125°C. Both guarantee full 8-bit accuracy and no missing codes over their respective temperature ranges.
The AD570J is also offered in an 18-pin plastic DIP.

PRODUCT HIGHLIGHTS

1. The AD570 is a complete 8-bit A/D converter. No external components are required to perform a conversion. Full scale calibration accuracy of ±0.8% (2LSB of 8 bits) is achieved without external trims.
2. The AD570 is a single chip device employing the most advanced IC processing techniques. Thus, the user has at his disposal a truly precision component with the reliability and low cost inherent in monolithic construction.
3. The AD570 accepts either unipolar (0 to +10V) or bipolar (-5V to +5V) analog inputs by simply grounding or opening a single pin.
4. The device offers true 8-bit accuracy and exhibits no missing codes over its entire operating temperature range.
5. Operation is guaranteed with -15V and +5V supplies. The device will also operate with a -12V supply.
6. The AD570S is also available with processing to MIL-STD-883, Class B. The single chip construction and functional completeness make the AD570 especially attractive for high reliability applications.

*Protected by Patent Nos. 3940760, 4213806 and 4136349.

SPECIFICATIONS

(typical @ +25°C with V+ = +5V, V- = -15V, all voltages measured with respect to digital common, unless otherwise indicated)

MODEL	AD570J	AD570S[1]
RESOLUTION[2]	8 Bits	•
RELATIVE ACCURACY @ 25°C[2,3,4]	±1/2LSB max	•
T_{min} to T_{max}	±1/2LSB max	•
FULL SCALE CALIBRATION[4,5] (With 15Ω Resistor In Series With Analog Input	±2LSB (typ)	•
UNIPOLAR OFFSET (max)[4]	±1/2LSB	•
BIPOLAR OFFSET (max)[4]	±1/2LSB	•
DIFFERENTIAL NONLINEARITY (Resolution for Which no Missing Codes are Guaranteed)		
+25°C	8 Bits	•
T_{min} to T_{max}	8 Bits	•
TEMPERATURE RANGE	0 to +70°C	-55°C to +125°C
TEMPERATURE COEFFICIENTS[4]		
Guaranteed max Change		
T_{min} to T_{max}		
Unipolar Offset	±1LSB (88ppm/°C)	±1LSB (40ppm/°C)
Bipolar Offset	±1LSB (88ppm/°C)	±1LSB (40ppm/°C)
Full Scale Calibration[6] (With 15Ω Fixed Resistor or 200Ω Trimmer)	±2LSB (176ppm/°C)	±2LSB (80ppm/°C)
POWER SUPPLY REJECTION[4]		
Max Change In Full Scale Calibration		
TTL Positive Supply +4.5V≤V+≤+5.5V	±2LSB max	•
Negative Supply -16.0V≤V-≤-13.5V	±2LSB max	•
ANALOG INPUT RESISTANCE	3kΩ min	•
	5kΩ typ	•
	7kΩ max	•
ANALOG INPUT RANGES (Analog Input to Analog Common)		
Unipolar	0 to +10V	•
Bipolar	-5V to +5V	•
OUTPUT CODING		
Unipolar	Positive True Binary	•
Bipolar	Positive True Offset Binary	•
LOGIC OUTPUT		
Bit Outputs and $\overline{\text{Data Ready}}$		
Output Sink Current	3.2mA min	•
(V_{OUT} = 0.4V max, T_{min} to T_{max})	(2TTL Loads)	•
Output Source Current (Bit Outputs)[7] (V_{OUT} = 2.4V min, T_{min} to T_{max})	0.5mA min	•
Output Leakage When Blanked	±40μA max	•
LOGIC INPUT		
Blank and Convert Input $0 \leq V_{in} \leq V+$	±40μA max	•
Blank – Logic "1"	2.0V min	•
Convert – Logic "0"	0.8V max	•
CONVERSION TIME	15μs min	•
	25μs typ	•
	40μs max	•

Reprinted with permission of Analog Devices, Inc., Norwood MA 02062.

ALL MODELS	
POWER SUPPLY	
Absolute Maximum	
V+	+7V
V-	-16.5V
Specified Operating – Rated Performance	
V+	+5V
V-	-15V
Operating Range	
V+	+4.5V to +5.5V
V-	-12.0V to -16.5V
Operating Current	
Blank Mode	
V+ = +5V	2mA typ (10mA max)
V- = -15V	9mA typ (15mA max)
Convert Mode	
V+ = +5V	5mA
V- = -15V	10mA

*Specifications same as AD570J
Specifications subject to change without notice.

NOTES
[1] The AD570S is available processed and screened to the requirements of MIL-STD-883B, Class B. When ordering, specify the AD570SD/883B.
[2] The AD570 is a selected version of the AD571 10-bit A to D converter. As such, some devices may exhibit 9 or 10 bits of relative accuracy or resolution, but that is neither tested nor guaranteed. Only TTL logic inputs should be connected to pins 1 and 18 (or no connection made) or damage may result.
[3] Relative accuracy is defined as the deviation of the code transition points from the ideal transfer point on a straight line from the zero to the full scale of the device.
[4] Specifications given in LSB's refer to the weight of a least significant bit at the 8-bit level, which is 0.39% of full-scale.
[5] Full scale calibration is guaranteed trimmable to zero with an external 200Ω potentiometer in place of the 15Ω fixed resistor. Full scale is defined as 10 volts minus 1 LSB, or 9.961 volts.
[6] Full Scale Calibration Temperature Coefficient includes effects of unipolar offset drift as well as gain drift.
[7] The Data output lines have active pull-ups to source 0.5mA. The $\overline{\text{DATA READY}}$ line is open collector with a nominal 6kΩ internal pull-up resistor.

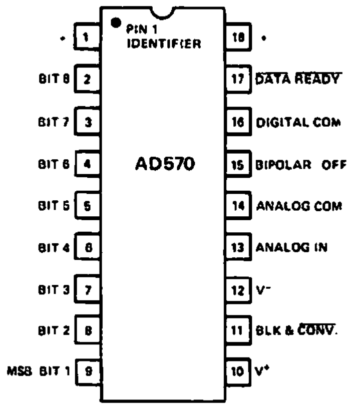
ABSOLUTE MAXIMUM RATINGS

V+ to Digital Common	0 to +7V
V- to Digital Common	0 to -16.5V
Analog Common to Digital Common	±1V
Analog Input to Analog Common	±15V
Control Inputs	0 to V+
Digital Outputs (Blank Mode)	0 to V+
Power Dissipation	800mW

AD570 ORDERING GUIDE

Model	Package Number[1]	Temperature Range
AD570JN	18-Pin Plastic DIP (N18A)[2]	0 to +70°C
AD570JD	18-Pin Ceramic DIP (D28A)	0 to +70°C
AD570SD	18-Pin Ceramic DIP (D18A)	-55°C to +125°C
AD570SD/883B	18-Pin Ceramic DIP (D18A)	-55°C to +125°C

[1] See Section 20 for package outline information.
[2] To be available June 1982.



*SEE NOTE 2, SPEC TABLE

Figure 1. AD570 Pin Connections

CONNECTING THE AD570 FOR STANDARD OPERATION

The AD570 contains all the active components required to perform a complete A/D conversion. Thus, for most situations, all that is necessary is connection of the power supply (+5 and -15), the analog input, and the conversion start pulse. But, there are some features and special connections which should be considered for achieving optimum performance. The functional pin-out is shown in Figure 1.

FULL SCALE CALIBRATION

The 5kΩ thin film input resistor is laser trimmed to produce a current which matches the full scale current of the internal DAC—plus about 0.3%—when a full scale analog input voltage of 9.961 volts (10 volts – 1LSB) is applied at the input. The input resistor is trimmed in this way so that if a fine trimming potentiometer is inserted in series with the input signal, the input current at the full scale input voltage can be trimmed down to match the DAC full scale current as precisely as desired. However, for many applications the nominal 9.961 volt full scale can be achieved to sufficient accuracy by simply inserting a 15Ω resistor in series with the analog input to pin 14. Typical full scale calibration error will then be about ±2LSB or ±0.8%. If a more precise calibration is desired a 200Ω trimmer should be used instead. Set the analog input at 9.961 volts, and set the trimmer so that the output code is just at the transition between 11111110 and 11111111. Each LSB will then have a weight of 39.06mV. If a nominal full scale of 10.24 volts is desired (which makes the LSB have weight of exactly 40.00mV), a 50Ω resistor in series with a 200Ω trimmer (or a 500Ω trimmer with good resolution) should be used. Of course, larger full scale ranges can be arranged by using a larger input resistor, but linearity and full scale temperature coefficient may be compromised if the external resistor becomes a sizeable percentage of 5kΩ.

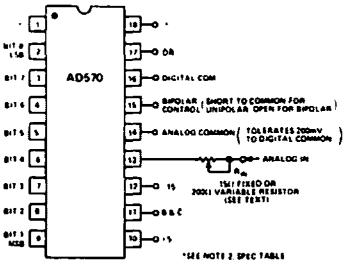

AD570
BIT 8 LSB
BIT 7
BIT 6
BIT 5
BIT 4
BIT 3
BIT 2
BIT 1 MSB
DR
DIGITAL COM
BIPOLAR CONTROL (SHORT TO COMMON FOR UNIPOLAR OPEN FOR BIPOLAR)
ANALOG COMMON (TOLERATES 200mV TO DIGITAL COMMON)
ANALOG IN
15Ω FIXED OR 200Ω VARIABLE RESISTOR (SEE TEXT)
B & C
*SEE NOTE 2, SPEC TABLE

Figure 2. Standard AD570 Connections

BIPOLAR OPERATION

The standard unipolar 0 to +10V range is obtained by shorting the bipolar offset control pin to digital common. If the pin is left open, the bipolar offset current will be switched into the comparator summing node, giving a -5V to +5V range with an offset binary output code. (-5.00 volts in will give a 8-bit code of 00000000; an input of 0.00 volts results in an output code of 10000000 and 4.96 volts at the input yields the 11111111 code.)

ZERO OFFSET

The apparent zero point of the AD570 can be adjusted by inserting an offset voltage between the Analog Common of the device and the actual signal return or signal common. Figure 3 illustrates two methods of providing this offset. Figure 3A shows how the converter zero may be offset by up to ±3 bits to correct the device initial offset and/or input signal offsets. As shown, the circuit gives approximately symmetrical adjustment in unipolar mode. In bipolar mode R2 should be omitted to obtain a symmetrical range.

Figure 3B shows how to offset the zero code by 1/2LSB to provide a code transition between the nominal bit weights.

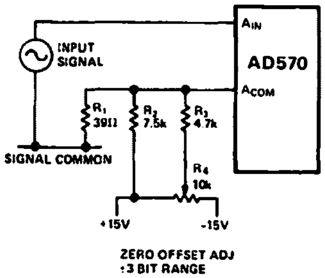


Figure 3A.

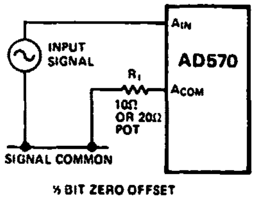


Figure 3B.

CONTROL AND TIMING OF THE AD570

There are several important timing and control features on the AD570 which must be understood precisely to allow optimal interfacing to microprocessor or other types of control systems. All of these features are shown in the timing diagram in Figure 4.

The normal stand-by situation is shown at the left end of the drawing. The BLANK and $\overline{\text{CONVERT}}$ (B & $\overline{\text{C}}$) line is held high, the output lines will be "open", and the $\overline{\text{DATA READY}}$ ($\overline{\text{DR}}$) line will be high. This mode is the lowest power state of the device (typically 150mW). When the (B & $\overline{C}$) line is brought low, the conversion cycle is initiated; but the $\overline{DR}$ and Data lines do not change state. When the conversion cycle is complete (typically 25μs), the $\overline{DR}$ line goes low, and within 500ns, the Data lines become active with the new data.

About 1.5μs after the B & $\overline{C}$ line is again brought high, the $\overline{DR}$ line will go high and the Data lines will go open. When the B & $\overline{C}$ line is again brought low, a new conversion will begin. The minimum pulse width for the B & $\overline{C}$ line to blank previous data and start a new conversion is 2μs. If the B & $\overline{C}$ line is brought high during a conversion, the conversion will stop, and the $\overline{DR}$ and Data lines will not change. If a 2μs or longer pulse is applied to the B & $\overline{C}$ line during a conversion, the converter will clear and start a new conversion cycle.

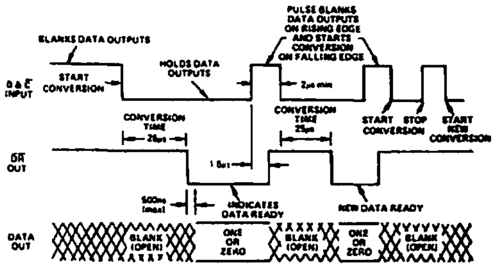


Figure 4. AD570 Timing and Control Sequence

Appendix B

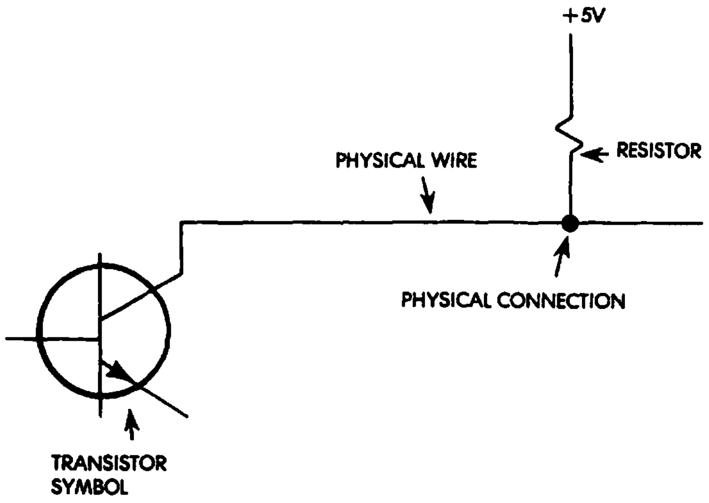
Appendix B:

Tips on Reading a Schematic Diagram

In this text we use a number of schematic diagrams to illustrate certain aspects of connecting the Apple computer to the outside world. For those readers who are not familiar with electronic schematics, the following tips are given to help you understand what information is being presented.

To start, the symbols used in schematic diagrams represent physical devices. They are not meant to resemble the components they represent. Instead, they are standard, stylized symbols meant to be understood by convention as standing for their devices. The lines that interconnect the symbols represent actual wires. In Figure B.1, the symbols shown represent a transistor connected to a resistor. Each physical device is represented in the diagram by a companion schematic symbol.

In digital logic many different symbols are used to represent the various components of a circuit. Since in this book we use only a few of these components, we will discuss only those symbols that will aid you in reading the schematics presented in this text. However, if you understand these symbols it will be much easier to read schematics presented elsewhere. (You will just have to learn a few more symbols.) Let us discuss the schematic shown in Figure B.2 and explain all of the important points.

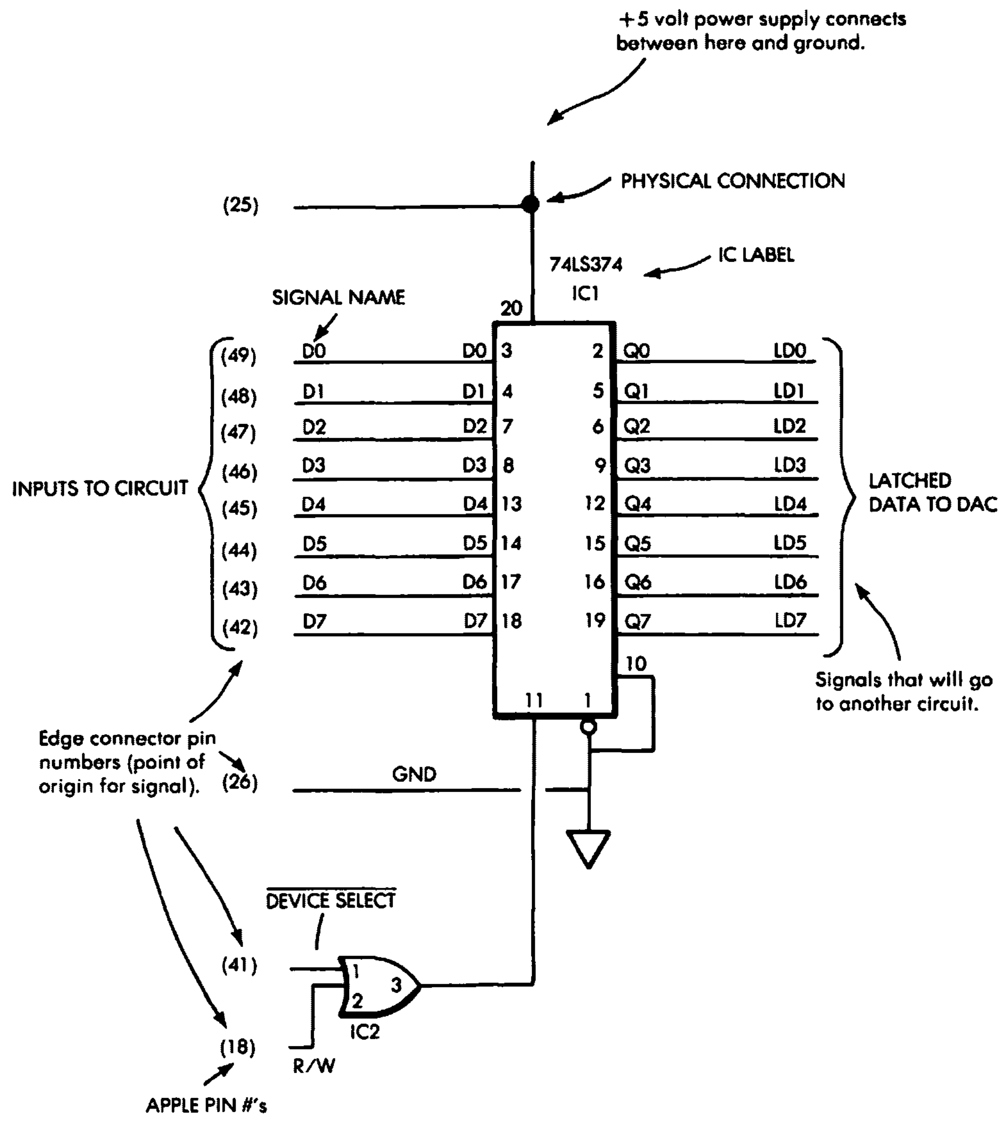


Schematic diagram of a transistor connected to a resistor. Each element is identified.
Figure B.1

The first point to notice in Figure B.2 is that the schematic should be read from left to right, just like the words on a printed page. That is, digital information, in the form of electricity, flows from the components shown on the left to those on the right. This is true of most schematics. On the left side of Figure B.2 are the inputs to the circuit. These inputs are given a signal name, such as R/W or D7, so they may be identified wherever they are used in the schematic.

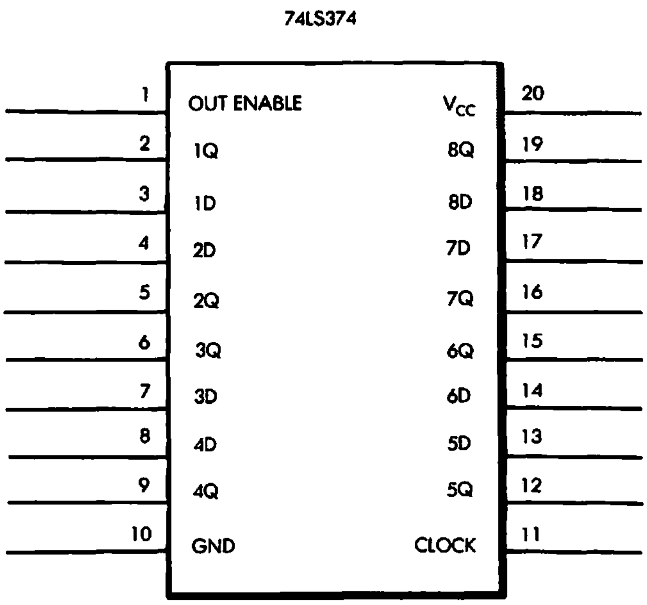
Each signal input in Figure B.2 also has a point of origin, indicating where the signal starts from. In this case, the inputs to the schematic will start from the edge connector pins of an Apple I/O slot. The edge connector pin number is shown in parentheses next to the signal name. For example, the data lines, D0–D7, originate from edge connector pin numbers 42–49. All the various signals are listed in the Apple computer documentation.

Following the signals D0-D7 farther to the right of the schematic, we see that they will connect to a rectangle. This rectangle represents a single integrated circuit. In this case the integrated circuit is a latch, labeled 74LS374. Each data line is shown connected to a specific number on the rectangle. These numbers correspond to



A more complex schematic, with the elements identified.

Figure B.2



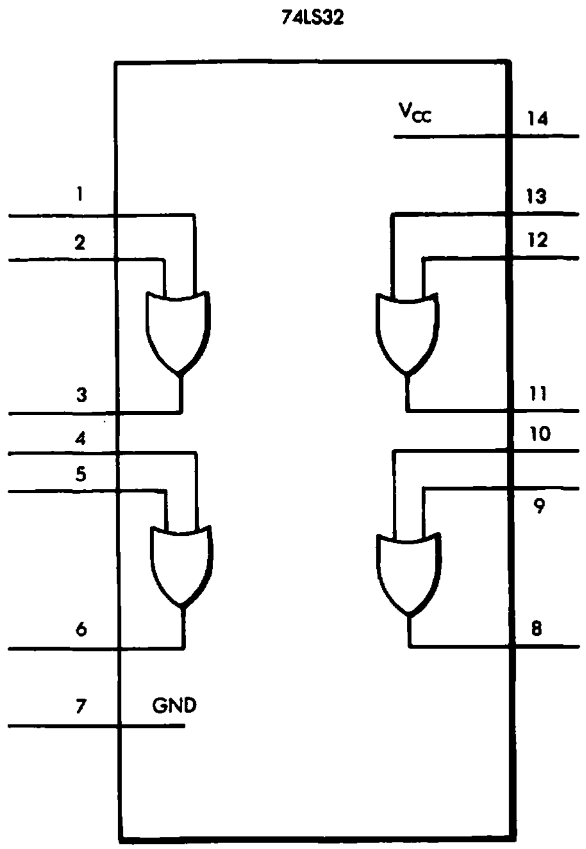
The pins on this integrated circuit are shown in their correct numerical order. In Figure B.2, the layout was altered to simplify the drawing and make it more legible.

Figure B.3

the pin numbers of the integrated circuit. The actual pinout of this IC is shown in Figure B.3.

We see in Figure B.3 that the integrated circuit has pin numbers that are labeled in a "U" arrangement. Pin 1 will always be located at the upper left hand corner. Notice that Figure B.2 does not show the pin numbers in the same numerical order as Figure B.3. The pin numbers in Figure B.2 are obtained from Figure B.3, but the placement of the numbers in Figure B.2 does not represent numerical order. Instead, the numbers are arranged to allow the schematic to be drawn easily.

Let us now discuss another input line to the schematic diagram B.2. This line is labeled $\overline{\text{DEVICE SELECT}}$. Notice the bar over the signal name. This is an indication that this signal will perform its

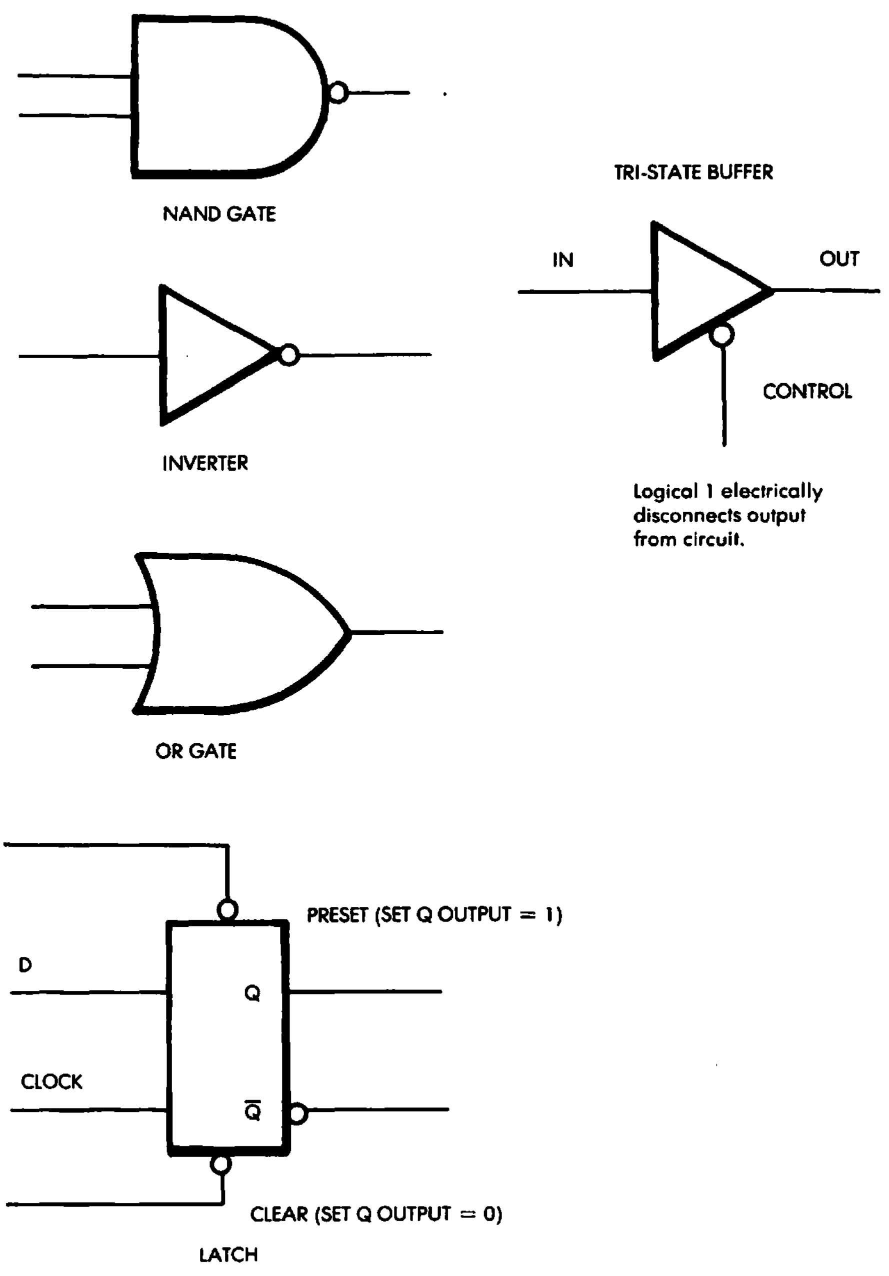


All four OR gates are located in the same device package.

Figure B.4

specified function (in this case, to select the I/O slot addressed) when it is in the logical 0 state. When the $\overline{\text{DEVICE SELECT}}$ signal is in the logical 1 state, the I/O slot is not selected for electrical communication with the computer.

The $\overline{\text{DEVICE SELECT}}$ line originates from edge connector pin 41. It is then connected to the symbol for a logical OR gate, shown in Figure B.2. Pin 1 of the OR gate represents the physical pin



The logic symbols used in this book.

Figure B.5

number of the integrated circuit package. A single integrated circuit package will contain up to four logical OR gates, as shown in Figure B.4.

Output pin 3 of the OR gate is connected to pin 4 of another OR gate. Both of these OR gates are contained in the same integrated-circuit package. We know this because both of the OR gates are labeled IC2. The "IC" stands for Integrated Circuit. Labeling of this nature is often used to denote gates that reside in the same physical package on the circuit board.

Output pin 6 of the OR gates is connected to input pin 11 of the 74LS374 integrated circuit we discussed previously. The outputs of the 74LS374 are not connected to anything shown in schematic B.2. If they were, the lines would be connected to some other symbol in the drawing. Instead, they lead off the page, sending latched data to an unseen DAC, for example. Again, the flow of the schematic is from left to right, input to output.

In the figures used in this text (as well as most schematics) the power supplies are not shown. Power is necessary to allow the system to function, but the connections to each individual IC and component in the schematic are "assumed." To find out where the power connects to each IC, refer to the manufacturer's data sheets. These sheets will indicate what pins the power is connected to and what voltage value is used. In general, most digital ICs use +5 volts and ground, while most operational amplifiers use +12 volts. However, you should always refer to data sheets to determine exact details.

Using the information presented here and throughout the book, you should be able to read and comprehend the schematic diagrams shown in this text. All of the special symbols used, like the symbol for a light-emitting diode, are discussed in the section of the text where they are used. This text uses only a few of the many logic symbols available. Those used are shown in Figure B.5. All of the pinouts for each of the integrated circuits used in this text are given in the data sheets contained in Appendix A.

Appendix C

Appendix C: Glossary of Selected Terms

AC appliance Any appliance that operates on Alternating Current, the type of electrical current that reverses direction at regular intervals, and is normally supplied to homes and businesses.

Analog event An event in nature that can have any value for its output. Some common analog events are temperature, pressure and brightness.

Analog-to-Digital Conversion (ADC) An electrical process by which an analog voltage is converted into its digital equivalent to be input to a home computer (or any type of digital computer).

Analog voltage An analog voltage is a voltage output from any source that can take on any numeric value. For example, 15.2345 V is an analog voltage.

BASIC (Beginner's All-purpose Symbolic Instruction Code) A computer programming language used by the Apple computer and most home computers.

Binary A description given to a set of values that may have two and only two possible outcomes. For example, the binary number system uses only two digits, 1 and 0.

Bit A single binary digit in a computer word. The bit may have the values 1 or 0.

Black box This colloquial term is used to describe any electronic circuit that performs a certain function, but whose internal operation need not be understood by the user, and usually isn't. A black box is inherently mysterious to its user.

Board An abbreviation for the term *circuit board.*

Byte A group of eight bits. An example of a byte would be 00101100. For an 8-bit computer like the Apple, *byte* and (data) *word* are synonymous.

Card Another way of saying *board.*

Circuit A collection of electronic components wired together to perform a certain function, or the electrical path between them. A circuit may consist of resistors, capacitors, transistors, digital IC's, or any other electronic elements.

Computer control The operation of a device or system under the direction of a computer.

Data Information output from or input to the computer. It takes the physical form of electrical pulses at one of two possible voltage levels.

$\overline{\text{DEVICE SELECT}}$ line A single logical line that will become active when a particular I/O slot on the Apple computer is selected with the software.

Digital-to-Analog Conversion (DAC) An electrical process by which an analog output voltage is produced by a specific combination of binary inputs to a piece of hardware called a digital-to-analog converter.

DIP An abbreviation for *Dual In-line Package,* an electronic package used to mount integrated circuits. It is rectangular and has leads (or "pins") extending from both long sides in a symmetrical pattern.

Flip-flop A group of digital electronic components that have the characteristic of storing information. Flip-flops are *bistable;* that is, they have two stable states, 1 and 0.

Flowchart A visual representation of the logical paths a computer program will follow as the instructions are executed.

4.7 kΩ An abbreviation for 4700 ohms, a resistance used in some circuits in this book. The letter is the metric symbol for 1000. Omega is the symbol for ohm, the unit of resistance.

Ground The point in a system that has a voltage potential of 0.0 volts.

Ground potential A voltage potential of approximately 0.0 volts. *Potential* is the difference in voltage between two points.

I/O Abbreviation for Input/Output.

I/O address A logical memory address that is assigned to a specific peripheral device.

I/O slot One of the eight physical connectors in the back of the Apple computer.

Input A signal sent to a computer or other device, or the act of sending a signal. During an I/O operation the computer can input, or take in, data.

Integrated Circuit (IC) An electronic component consisting of one or more circuit elements fabricated on a single chip of silicon, and usually packaged in a DIP.

Latch An electronic device capable of storing a single bit of binary information when electrically instructed to do so. An 8-bit latch will store eight bits of binary information at the same time.

Light-Emitting Diode (LED) An electronic device that has the physical property of emitting a certain wavelength of light when current is passed through it in the forward direction.

Logical 0 One of the two possible binary states that a digital logic circuit can reside in. For the Apple computer, a logical 0 is equivalent to a voltage level of less than or equal to .8 volts.

Logical 1 The second of the two possible binary states that a digital logic circuit can reside in. In the Apple computer, a logical 1 is equivalent to a voltage output greater than 2.0 volts.

Logic gate A digital hardware element that will perform one of the Boolean logical functions, such as AND or OR.

Nibble A group of four bits. 0011 is an example of a nibble. A byte consists of two nibbles.

Output Information sent from a computer to another device, or the action of sending it. When the computer is sending data to a peripheral device for control, it is outputting. Broadly, an electrical quantity produced by a circuit or the physical action that results. The output of a lamp is light; the output of a loudspeaker is sound.

Output port Any output address that the computer can send data to.

PEEK address The address used in a PEEK instruction.

PEEK instruction An instruction in BASIC that will allow data to be input from a valid memory address.

Phase (Φ) 0 A free-running clock that can be used to time the writing of data to the peripheral circuits connected to the Apple computer.

POKE address The memory address that is specified during the execution of the POKE instruction.

POKE instruction An instruction in BASIC that allows data to be output to a valid memory address.

Resistance An electrical property that impedes the flow of electrons.

Resistor A physical element that has the electrical property of resistance. It is usually tubular, with leads extending from both ends.

R/W line A signal line that is used to electrically inform the I/O circuits whether the computer is reading or writing during this memory cycle.

Schematic diagram A drawing using electronic symbols to represent the component interconnections of a circuit.

74LS____ A numeric label given to a family of integrated circuits used in this book. LS is an abbreviation for Low-power Schottky, a type of design technology. The blank in the label can be any 2- or 3-digit number. This number makes it possible to look the integrated circuit up in a data book.

Solid-State Relay (SSR) An electronic device that will open and close two electronic contacts without any mechanical parts.

Transducer An electronic device that will change a physical action into an electrical equivalent. For example, a temperature transducer will change the temperature it senses into an equivalent electrical value.

Transistor An electronic component that can be made to amplify electronic signals.

Transistor-Transistor Logic (TTL) The type of logic family used in the circuits of the Apple computer.

Appendix D

Appendix D: List of Vendors

Information concerning the prices and availability of various hardware devices described in this book can be obtained by writing to the outlets listed below. The list is not meant to be exhaustive.

TTL components (logic gates, inverters, latches, etc.):

Jameco Electronics
1355 Shoreway Rd.
Belmont, CA 94002

Quest Electronics
PO Box 4430
Santa Clara, CA 95054

Analog-to-digital and digital-to-analog converters, along with several different peripheral devices for microcomputer I/O:

Microprocessor Training Inc.
14 East Eighth St.
New York, N.Y. 10003

CMS I/O board for the Apple computer:

Creative Microprocessor Systems, Inc.
PO Box 1538
Los Gatos, CA 95030

Index

SOFTWARE

BAS 65™ CROSS-ASSEMBLER IN BASIC
8″ diskette, Ref. BAS 65
A complete assembler for the 6502, written in standard Microsoft BASIC under CP/M®.

8080 SIMULATORS
Turns any 6502 into an 8080. Two versions are available for APPLE II.

APPLE II cassette, Ref. S6580-APL(T)
APPLE II diskette, Ref. S6580-APL(D)

Connect your Apple to the real world!

The Apple® Connection

. . . teaches you to program and connect your Apple computer to real appliances and devices.

You will sharpen your BASIC programming skills as you learn elementary interfacing techniques.

In particular, you will learn how to use . . .

- PEEK and POKE instructions in BASIC
- Apple I/O slots
- Transducers and solid-state relays

and . . .

- How to perform digital-to-analog and analog-to-digital conversions
- How to build real applications, such as a home security system, and a complete temperature monitoring system

About the Author:

James W. Coffron is a computer systems engineer specializing in the design and testing of microprocessor-based systems. He has taught seminars in both academic and industrial settings, and is the author of several books about microprocessors. He holds an MSEE degree from Santa Clara University.

ISBN 0-89588-085-7