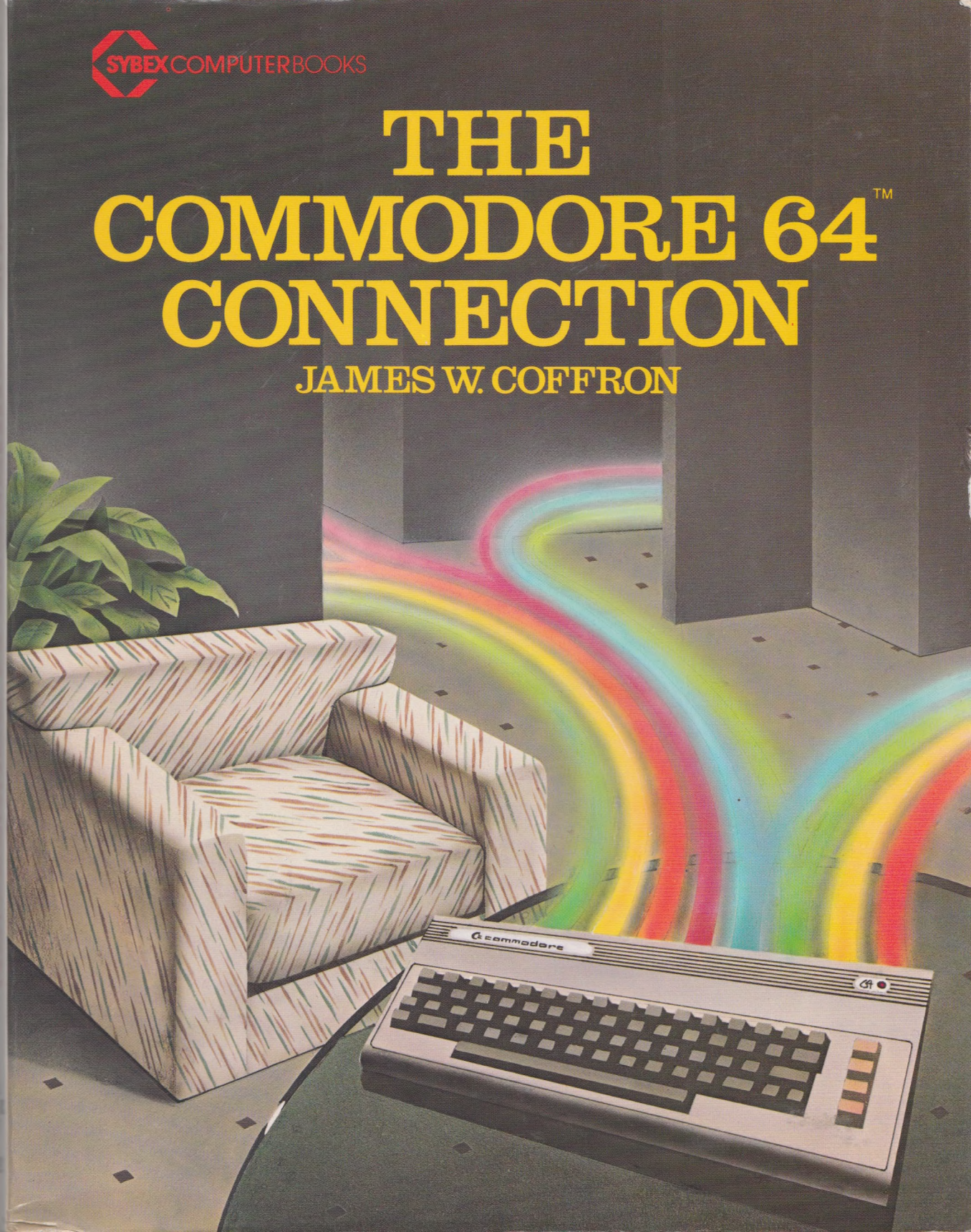
SYBEX COMPUTER BOOKS

THE COMMODORE 64™ CONNECTION

JAMES W. COFFRON



COFFRON

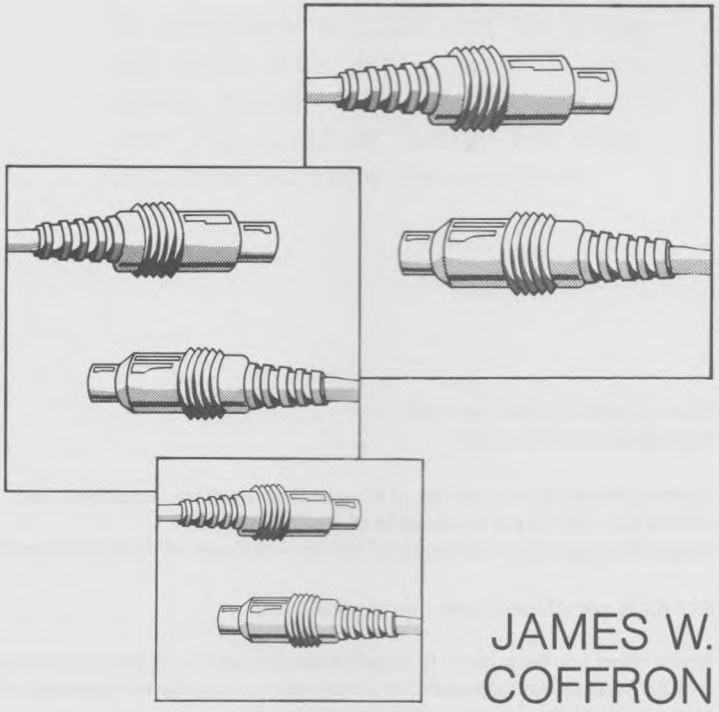
THE COMMODORE 64 CONNECTION

SYBEX

0-192

THE COMMODORE 64
CONNECTION

THE COMMODORE 64™ CONNECTION



JAMES W. COFFRON

SYBEX®

Berkeley • Paris • Düsseldorf

Cover design by Tom Cervenak
Book design by Lisa Amon

Library of Congress Card Number: 84-51242
ISBN 0-89588-192-6
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

To my brother Michael and his family; his lovely wife Jill, wonderful sons Brock, Robert, and Beau, and his charming daughter Tammy. For Mike, who does not know the word quit.

ACKNOWLEDGEMENTS

I wish to thank my editor, Lorraine Aochi, for her splendid work and excellent guidance in the production of this text. Good job, Lorraine! To my wife Carol, my appreciation for her help in preparing the preliminary drawings of the technical figures. To the staff of SYBEX, whose efforts always result in a first-class book, my admiration and respect for a difficult job well done.

TABLE OF CONTENTS

PREFACE

If you have recently purchased a Commodore 64™ Personal Computer, or are thinking of purchasing one, you probably have many questions about your computer's overall usefulness. While you bought it for a specific purpose in the first place, you may wonder what else it can do.

There are more potential future uses of the Commodore 64 than any purchaser can dream of today. As you sit in front of the computer, you can look forward to hours of enjoyment using the many applications programs and the various "off-the-shelf" games that are available for the system. If you are a beginner in home computers, these games and programs may seem difficult to master at first, but this difficulty will soon pass, so do not worry.

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 confidence improves as you discover that nothing drastic happens when you press the wrong key. The Commodore 64 is a very forgiving instrument.

Before long, you are skillfully running, modifying, and writing programs. The once-formidable task of using a home computer is now easier. Soon you find yourself looking for new challenges and new applications for the computer. You may wonder at this point, "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, if you feel it is far beyond your ability to accomplish them, you will soon see that you are wrong. The realization of these controls with your Commodore 64 is not beyond your capabilities. The required information may be different from that which you use everyday, but making the Commodore 64 connection is a straightforward, fairly simple process.

The designers of the Commodore 64 showed valuable foresight in anticipating those system users who do not know or care much about hardware, but who want to create new interfaces between their system and the outside world. The Commodore 64 architecture was designed to make interfacing easy. You do not have to be a computer expert to construct the hardware or write the software for controlling external devices. 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 to 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 Commodore 64 connection.

INTRODUCTION

This book is written for everyone who wants to understand how the Commodore 64, as well as other home computers, can be interfaced to the outside world. This book illustrates the essential concepts of computer control and interfacing. However, you can readily adapt the information and ideas presented here to most home computers.

This text assumes you can write simple programs in BASIC. An extensive knowledge of BASIC is not required to get the maximum benefit 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.

This book is organized so you can understand how all the pieces of the interfacing-and-control puzzle fit together. The path given in the following pages is straightforward and fairly simple, but you will have to learn some new, essential information and concepts in order to interface and control external devices with the Commodore 64 and other home computers.

Chapter 1 introduces and defines 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 Commodore 64. The programming language we will use is BASIC.

Chapter 3 discusses the software required to input information to the Commodore 64 from an external device. Again, we will use BASIC as the programming language.

Chapter 4 introduces the basic hardware concepts necessary to input and output information to and from the Commodore 64. This chapter is designed for readers who are unfamiliar with digital electronics, and who want to learn only as much 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 for home security.

Chapter 6 shows how to make the Commodore 64 produce speech, with an introduction to speech synthesis using phoneme speech techniques. This chapter then covers the software and hardware required to let your Commodore 64 speak when and what you desire.

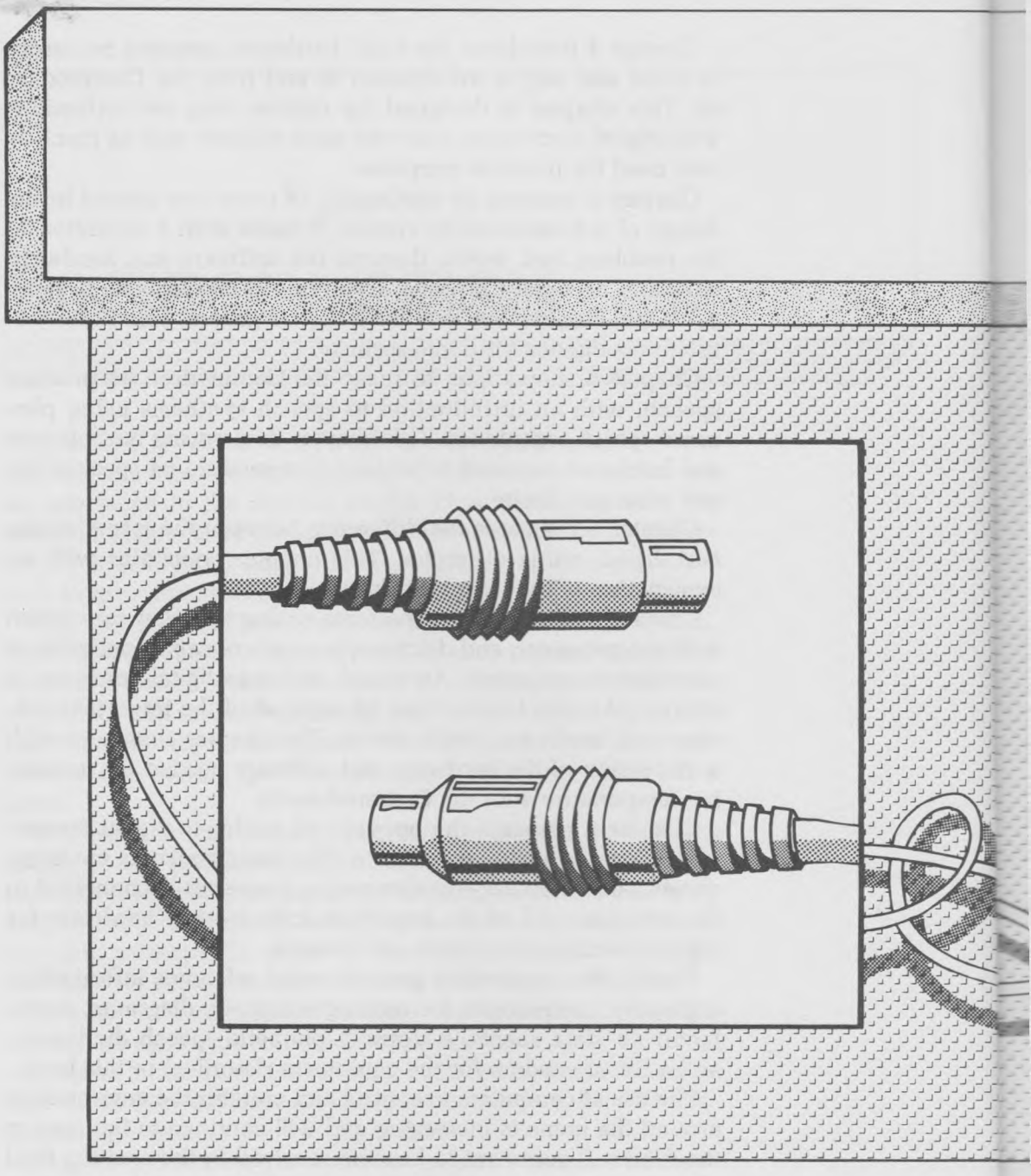
Chapter 7 discusses the difference between the terms *analog* and *digital,* using examples. This chapter concludes with an explanation of the term *transducer.*

Chapter 8 shows how to perform analog-to-digital conversion with the computer, and discusses general concepts applicable to most home computers. An actual analog-to-digital converter is connected to the Commodore 64, with all of the important software and hardware details shown. The chapter concludes with a discussion of the hardware and software needed for measuring temperature with the Commodore 64.

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 software and hardware for digital-to-analog conversion are covered.

Finally, the Appendices present useful reference information: a glossary, instructions for reading schematic diagrams, manufacturers' data sheets, a Votrax® phonetic speech dictionary, 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.



INTRODUCTION TO COMPUTER CONTROL

1

Before we launch into the discussion of actual computer control that begins in Chapter 2, let's 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 only scientists use—inevitable, but too complicated to understand. In fact, while complex applications of computer control are used in spacecraft and automated factories, 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 help you understand computer control. With this understanding will come new insight, allowing persons not directly involved with scientific uses of the computer to see many ways they can use computers in the home. As you develop these home applications, you will lose whatever fear you may have of automation, and gain insight into what computer control can do and how 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 Commodore 64 in this text as the means of control. However, the concept of computer control is applicable to almost any home computer. If you have a Commodore 64 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 see several examples of how a home computer is used. The simple concept of computer control is graphically illustrated in Figure 1.1, which shows the computer connected in some way to direct the action of another piece of hardware.

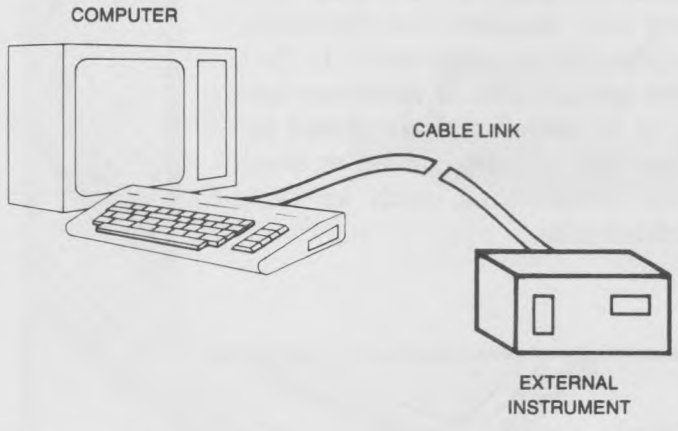


Figure 1.1: *The main concept of computer control is that the computer will direct the action of an external piece of hardware. The link shown here is made by a cable. However, the link may also be made by transmission of radio frequencies, without a physical connection.*

This, in essence, is what computer control is all about: the computer directs 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 responding to its control. Therefore, the computer must not only direct but also monitor the action of the external hardware. 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 two essential concepts of computer control:

1. The computer transmits instructions to the external hardware.
2. The computer receives information from the external hardware.

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

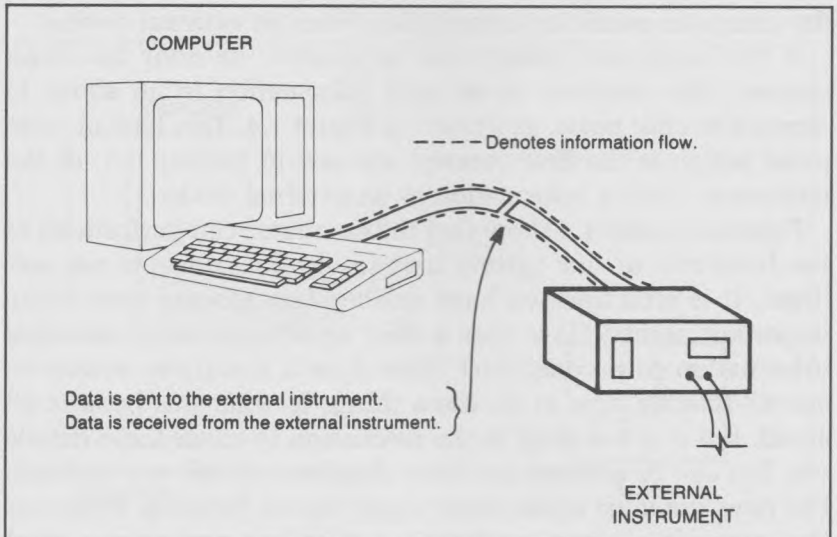


Figure 1.2: *The computer will send information to the external device to control the device's actions. Information can also be sent by the external device to be read by the computer.*

1.2 A Practical Example of the Two Basic Concepts

An example of computer control that most of us can imagine using is a home-security system. Through diagrams, we will see that you can achieve this goal using only the two concepts we have outlined. We will develop such a system in detail in Chapter 5.

Let's start by defining what we want the security system to do. In short, we want our home safe from intruders. Unfortunately, this statement is useless for our purpose, because the word "safe" conjures up entirely different meanings to different people. Let's try to be more precise: the system will detect any window or door being opened, will indicate which door or window it is, and will sound an alarm if any door or window is opened. This is the exact definition of what we wish the system to do. In Chapter 5, we will expand on this definition to show new ideas for a computer-controlled, home-security system.

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

If the computer detects that a window or door has been opened, the computer must send information to an alarm to direct it to emit noise, as shown in Figure 1.4. This type of computer action is the first concept we saw in Section 1.1, of the computer sending information to an external device.

Figures 1.3 and 1.4 show that the computer can perform all of the functions of our system using only the concepts we outlined. It is true that we have deliberately glossed over some important points: How does a door or window send electrical information to a computer? How does a computer sound an alarm? Exactly how to do these things is what this book is all about. But it is too early in the discussion to cover these details yet. You can be assured that later chapters will tell you in detail. For now, we must understand where we are heading. From our first example, we can see that we can reduce computer control to the repeated application of the two basic concepts. You must completely understand these concepts before you go on.

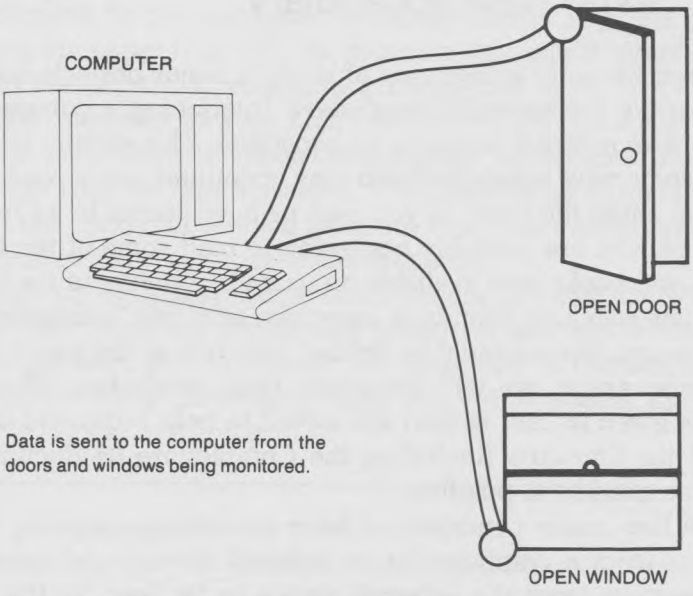


Figure 1.3: *Doors and windows send information to the computer to report their status, either "open" or "closed."*

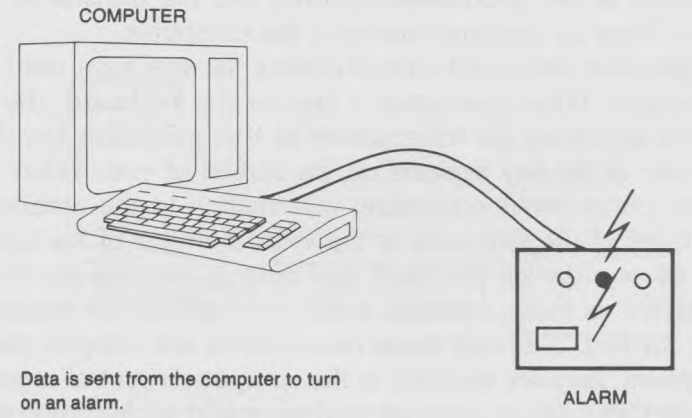


Figure 1.4: *A computer will send information to the alarm to inform the user that a window or door has been opened.*

1.3 Some New Vocabulary

When we enter a new area of study, a major obstacle we face is learning the essential vocabulary. Interfacing a computer to control an external device is no exception. This section will discuss some new words that you may encounter when reading or talking about the topic. If you own or have access to a Commodore 64, you are probably beginning to read some of the magazines and books now available on using computers in the home. You may also have friends or associates who talk "computerese." All the new terms cannot be defined here but, as the topics in the text warrant it, we will introduce new vocabulary. The new words given in this section are meant to help beginners understand the literature (including the Commodore 64 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 information itself, and to the communication of information between a computer and an external device. Thus, output is both the information sent and the transfer of information from the computer to the external device, and input is the information entered and the transfer of information from an external source to the computer.

If you have ever used a Commodore 64, you have used input and output. When you press a key on the keyboard, the computer is inputting the information of that particular key. When the letter of the key appears on the screen of your video monitor, the computer is outputting information to the monitor. We may think of the keyboard or the video monitor of the Commodore 64 as a device for input and output, because we think of the device as being external and connected by, for example, a cable. In fact, although these components are integral parts of the system, they are external to the computer's *central processing unit*, or CPU. Other external 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 Commodore 64 gives a printout, the operation of the printer is controlled by the computer. While 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 cassette recorder attached to your Commodore 64, the computer also controls these devices. When the computer writes your program onto the disk or tape cassette, it is outputting information. When the computer reads a program from the disk or cassette, information is being input to the computer.

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 "doing I/O."

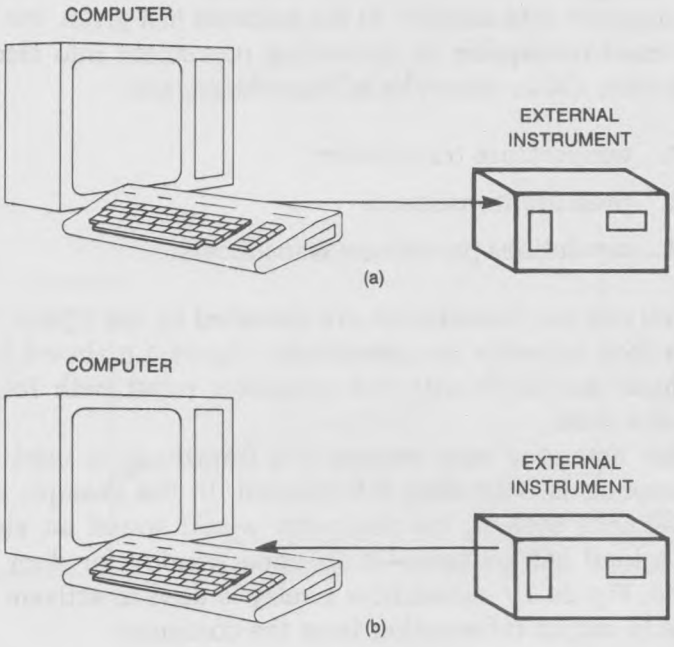


Figure 1.5(a): *This is an example of* output. *The computer is sending information to the external device.*

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

The next term we will discuss is *transducer*, which is used frequently when discussing I/O. To illustrate what a transducer is, let's 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 window. The information input to the computer is electrical, but the door or window only gives out information about physical movement, because all it can do is move. Therefore, we need some type of device that changes 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 a transducer.

This 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 that 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 transducers
2. pressure transducers
3. revolutions-per-minute transducers

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.

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—either a loud bell or siren—if an open window or door were detected. 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*, which may already be familiar to you. Generally, the term means that there are discrete values assigned to a particular event. For example, television channels are assigned certain frequencies out of an infinite range of frequencies. Selecting a TV station is a digital process,

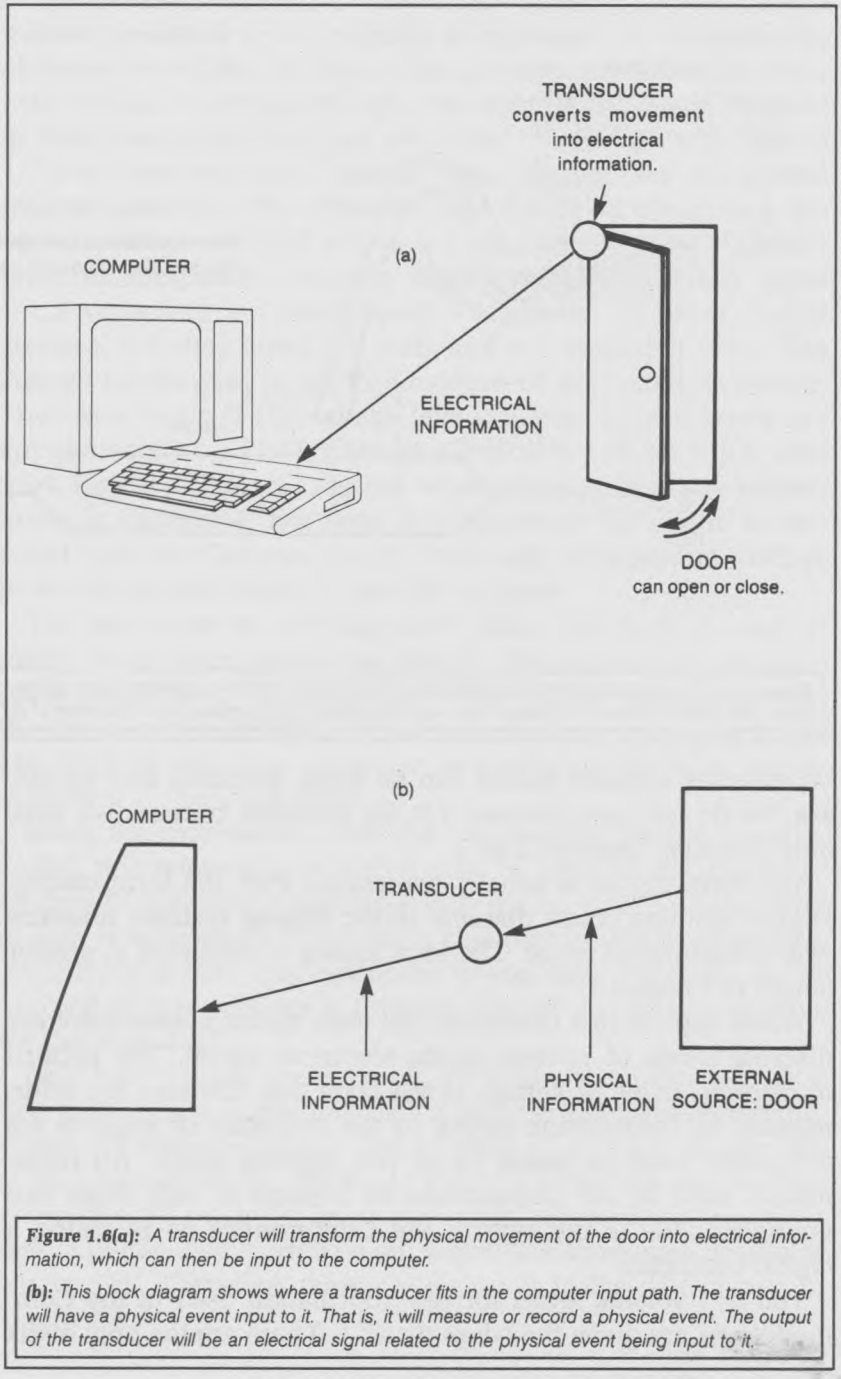


Figure 1.6(a): *A transducer will transform the physical movement of the door into electrical information, which can then be input to the computer.*

(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 measure or record a physical event. The output of the transducer will be an electrical signal related to the physical event being input to it.*

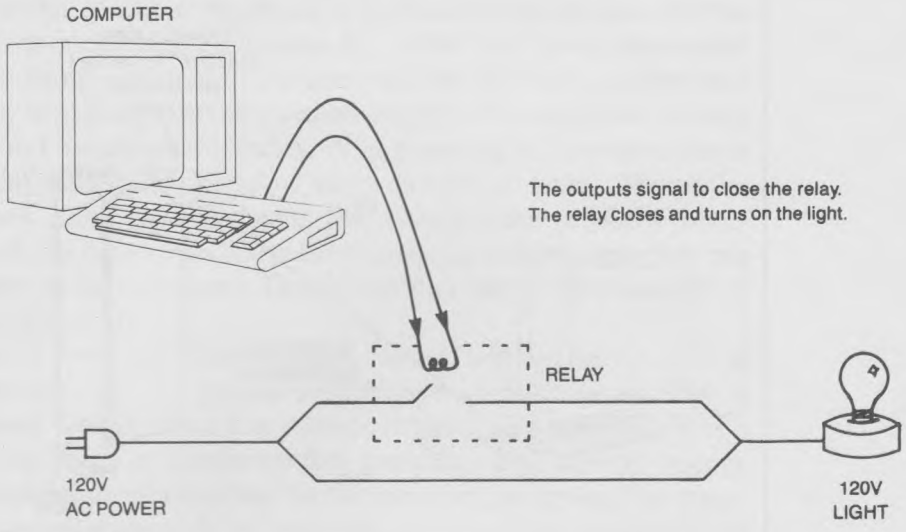


Figure 1.7: *This diagram shows a physical device that will transform the electrical output of the computer into a physical event. This device, a relay, will transform the electrical output of the computer into movement of the relay contacts.*

because the channel values can be those specified and no others. We do not have channel 2.5, for instance, because we have only specified 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. The term analog is discussed at greater length in Chapter 7.

When applied to a computer, the term *digital* means there are discrete levels of voltage in the electrical signal. The pattern of discrete levels of voltage is the code that contains the information. All information output by the computer or input to the computer must be made up of two voltage levels. All information used by the Commodore 64 consists of only these two voltages. This is the reason the Commodore 64 is called a *digital* computer.

The two voltage levels for the information used in the Commodore 64 are given the labels 0 and 1. These are the only digits

in the *binary* notation used in almost all computers. All information processed by a computer is expressed in combinations of these two digits. In binary logic, every statement is either true or false. In computer logic, the digits 0 and 1 are assigned to these two conditions, and are called "logical 0" and "logical 1." For these reasons, "binary" and "digital" are often used interchangeably in the computer field. In digital electronics, the actual voltage value of a 0 or a 1 may vary among different types of equipment. In some digital systems, a 0 may equal −1.9 volts and a 1 may equal −1.5 volts. In other digital systems, a 0 may equal 0.0 volts and a 1 equal 9.0 volts. The voltage values used in the Commodore 64 are called *Transistor-Transistor Logic (TTL)* 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 what is meant by these voltage levels in Chapter 4. For now, it is important for you to understand that the Commodore 64 uses only *two* distinct voltage levels when information is output or input.

The next term we will discuss is *data*. This term is used in many ways when discussing digital computers. For our purposes in this text, data will refer to the digital information that the Commodore 64 inputs or outputs when we are using it. We can think of data in a computer as a series of electrical pulses occurring at specified times.

Since the information that the Commodore 64 processes is digital, it is referred to as digital data. In other words, the Commodore 64 will output and input only digital data. No other type of data is acceptable. If you are a beginner in using computers, other types of data may not occur to you now. However, as we proceed to later sections in this text, we will see other meanings for the term data.

Let's now discuss the term *bit*. To define this term, we must first look at how digital data will appear in the Commodore 64. Figure 1.8 shows what one typical piece of data would look like to the Commodore 64. We see in Figure 1.8 that the data is composed of eight single 1s, 0s, or any combination of the two. Each unit of digital data is called a bit, which is a shortened version of *binary digit*. Therefore, there are eight bits in the data shown in Figure 1.8.

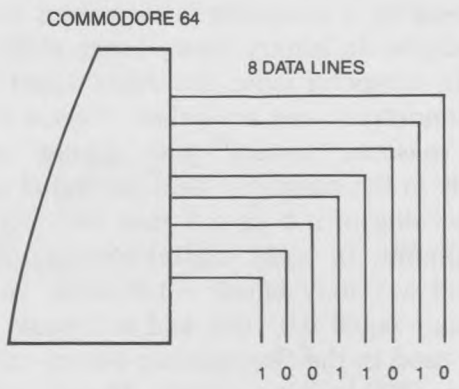


Figure 1.8: *Data output lines can be set to any combination of logical 1s or 0s.*

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

Figure 1.9: *These are examples of different data bytes. Each byte consists of a combination of eight digits, either logical 1 or logical 0, called bits.*

The next new term that we will cover is *byte*. A byte consists of eight bits that are communicated at the same time; the bits are said to be *parallel*. We can, therefore, describe the digital-data output and input by the Commodore 64 as data bytes. The Commodore 64 will output or input digital data in units of one byte, or 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 the position of the bits in a byte) determines their values as powers of 2. For example, in the base-10 number 357, the numeral 3 represents 3×10^2. In the number 35, the same numeral 3 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 numeral represents numbers raised to the power of zero. (Note that any number raised to the zero power equals 1.) As data bytes, all base-2 numbers are filled out to eight places by adding zeroes. Thus, 1×2^2 would be represented as 00000100, and 1×2^1 as 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)*.

When discussing data input and output, people will sometimes 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 Commodore 64 and other 8-bit computers, the data word will be 8 bits, and so data byte and data word are synonymous. For other types of computers, the data word may be longer than 8 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 Commodore 64, the data byte will consist of two parallel (simultaneous) nibbles, as shown in Figure 1.11.

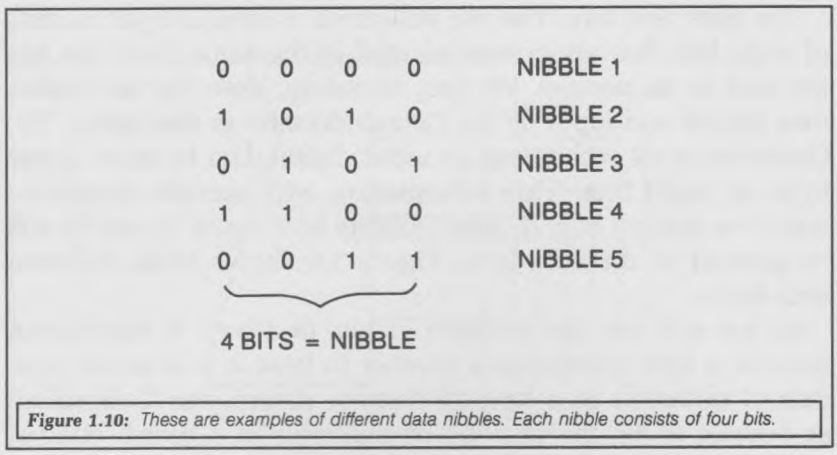


Figure 1.10: *These are examples of different data nibbles. Each nibble consists of four bits.*

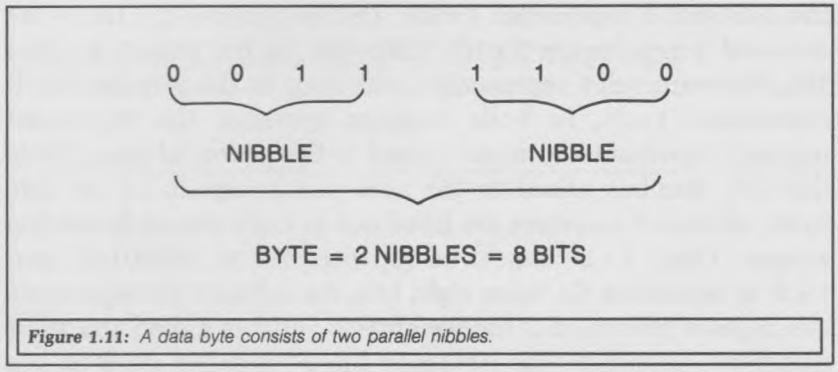


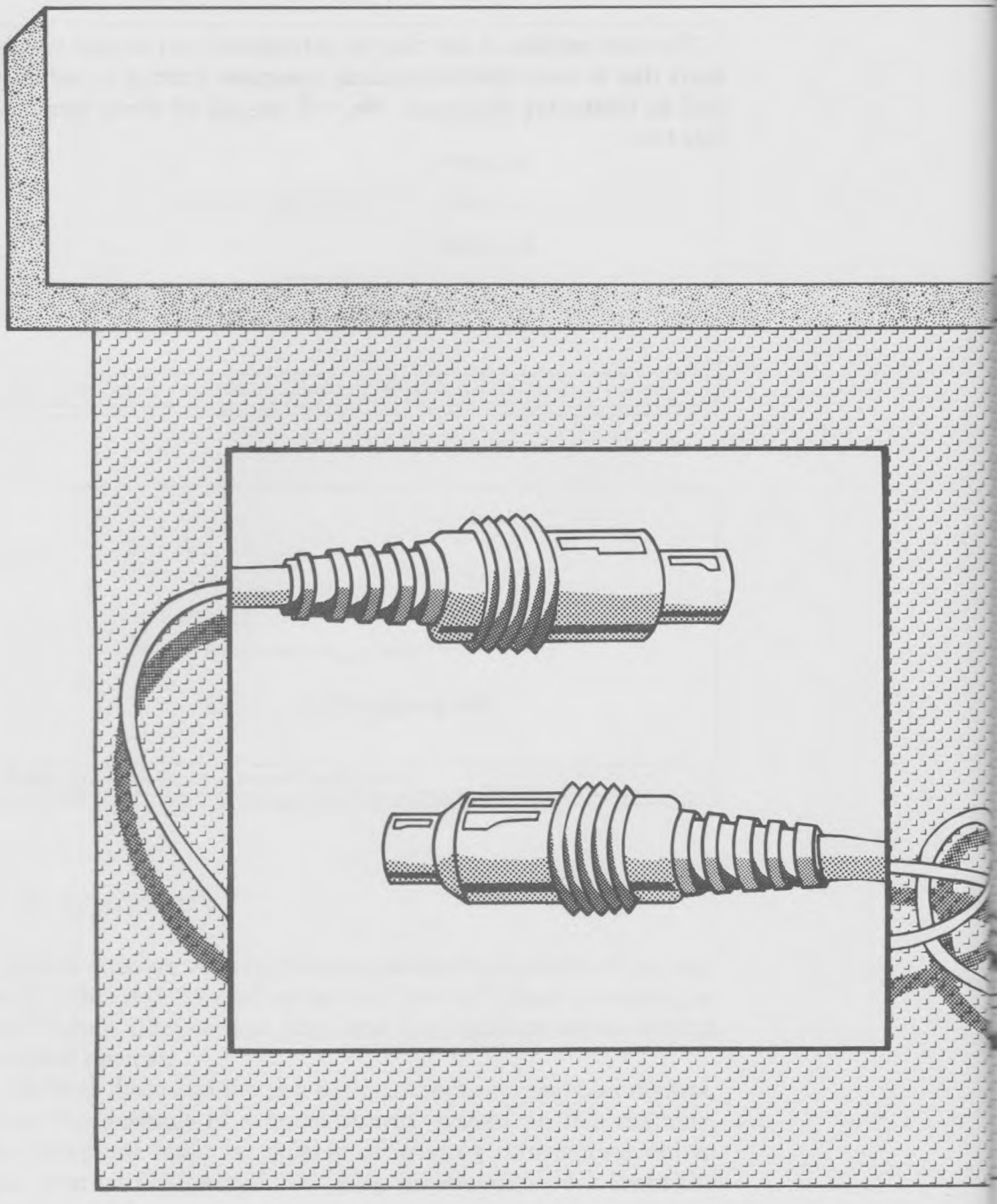
Figure 1.11: *A data byte consists of two parallel nibbles.*

1.4 Summary

In this chapter, we have introduced the topics that will be covered in this text. The following chapters will show how to input and output information from the Commodore 64 to control external devices.

We have discussed the two basic concepts of computer control, using the example of a home-security system. In this example, the computer must be capable of sending information out to the external device and receiving information back from the external device.

The next section of the chapter introduced you to new vocabulary that is used when discussing computer control in industry and in computer literature. We will use all of these terms in this text.



SOFTWARE FOR OUTPUT FROM THE COMMODORE 64

2

In this chapter, we will discuss the programming necessary to output digital information from the Commodore 64 to the outside world. We only assume that the reader is familiar with the versions of the BASIC programming language used on the Commodore 64. You will be given examples with each new topic that we cover.

We've designed the examples in this chapter to be used with the Creative Microprocessor Systems (CMS) I/O system. The CMS I/O board is a visual means of testing your Commodore 64's input and output programs. You can install it directly into the Commodore 64,

and output and input data with it. Although you do not have to purchase this board to benefit from the examples given, you will learn more about writing this kind of program if you use the CMS I/O board (or an equivalent device). Information regarding availability of the CMS I/O system for the Commodore 64 is given in Appendix E.

2.1 Installing the CMS I/O System

Before we discuss the software required for inputting and outputting data with the Commodore 64, we must first learn how to install the hardware into the computer. This discussion will bring out important general information about the physical connections necessary to use computer control with the Commodore 64.

The CMS I/O system comprises two printed-circuit boards (PC boards) and an interconnecting cable. One of the circuit boards plugs directly into the Commodore 64. The second circuit board will connect to the first board via a 24-pin ribbon cable. Figure 2.1 is a diagram of these two boards. Let's now go over the details of how to connect the first board directly to the Commodore 64.

Figure 2.2 shows the I/O expansion slot for a Commodore 64. It is through this slot that external instruments are controlled by the Commodore 64. The Commodore 64 has another slot labeled User Port. We will not make use of it because using the I/O expansion slot will give you a more general technique for connecting your computer to the outside world.

Static electricity, which is easily generated, can seriously damage a computer's circuitry. Rodnay Zaks' book *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:

Step 1. Turn off the power on your computer and any other peripherals. Always turn off the power when installing any hardware into the computer.

Step 2. Turn the Commodore 64 so that the I/O expansion slot at the rear of the computer faces you. This 44-pin, 22/22 (22 pins on top, 22 pins on bottom) slot is located on the far left, as shown in Figure 2.1.

Step 3. Install the 24-pin ribbon cable into the 24-pin socket on the CMS I/O board labeled CMS-641. Be certain to install the 24-pin connector with pin 1 in the correct place. Figure 2.3 shows how to install the cable into the respective circuit boards.

Step 4. Install the CMS-641 circuit board into the I/O expansion slot so that the the ICs (integrated circuits) are on top. Figure 2.4 shows a diagram of how the circuit board is to be installed into the Commodore 64.

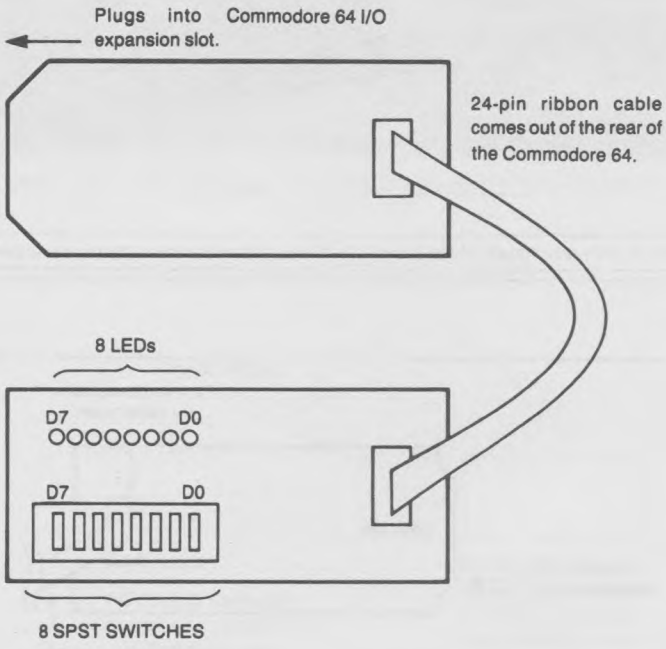


Figure 2.1: *This diagram shows the two printed-circuit boards that constitute the CMS I/O system. One board plugs directly into the Commodore 64, while the other board sits alongside the computer for easy viewing by the user.*

Step 5. Run the ribbon cable around the back of the Commodore 64. The cable will then connect to the second circuit board labeled CMS-642.

Step 6. Connect the remaining end of the 24-pin ribbon cable into the second circuit board. Be sure that pin 1 of the cable is connected to pin 1 of the socket on the second circuit board.

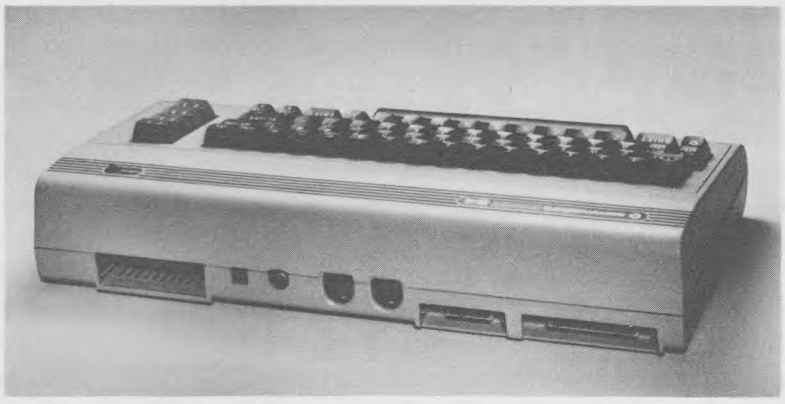


Figure 2.2: *This photograph of the back of the Commodore 64 shows the I/O expansion slot.*

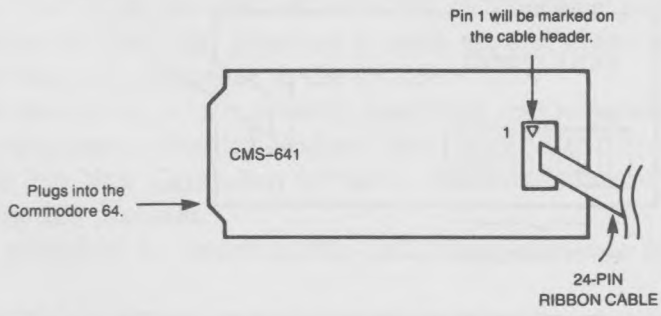


Figure 2.3: *The CMS-641 board will plug directly into the I/O expansion slot in the rear of the computer. A 24-pin cable will connect to this printed-circuit board.*

Step 7. At this time, your system will appear as shown in Figure 2.5. Now your system is ready for use with the CMS I/O system.

Step 8. Turn on the system power.

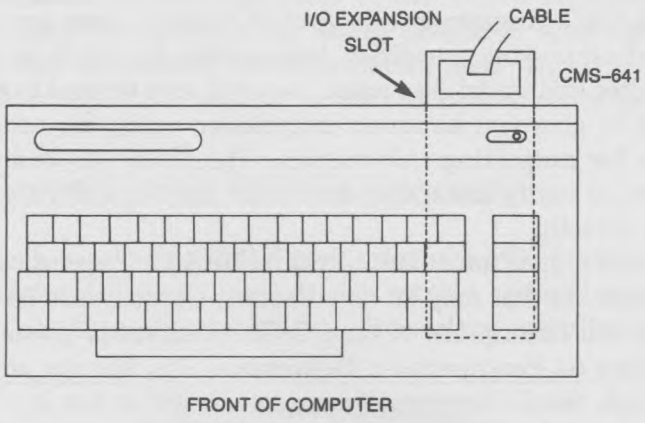


Figure 2.4: *This diagram shows exactly how the CMS-641 board is installed into the Commodore 64.*

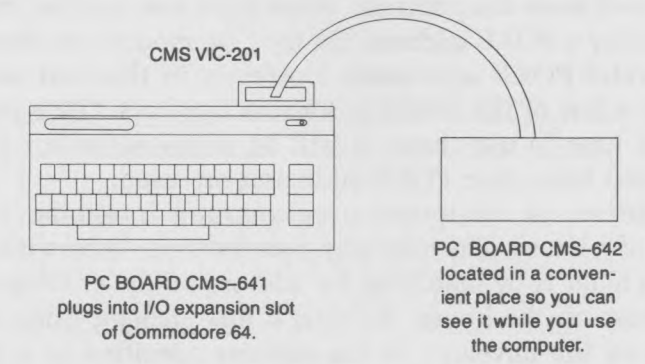


Figure 2.5: *When both boards are connected properly, the complete system will appear as shown.*

2.2 The POKE Statement

We will introduce new information based on the assumption that you are familiar with BASIC. You are not expected to be an expert programmer, but you should be able to write and run simple programs using the Commodore 64.

To start, you must know how to direct digital information from a BASIC program to the CMS I/O board. Once you know how to do this, you can direct information to any type of I/O board. After this initial discussion, we will give several examples designed to give you hands-on experience using the necessary software for outputting information. The CMS I/O board will allow you to verify instantly whether or not the software is operating correctly.

To output digital information from a BASIC program, we will use a statement that may be new to some Commodore 64 users: POKE. A full description of the POKE statement is given in the *Commodore 64 Programmer's Reference Guide,* but we will give you enough details here so that you can begin to use it.

The form of the POKE statement in a BASIC program is:

POKE address,data

We will later present different examples using POKE. Let's now discuss the two parts of the POKE statement, *address* and *data.*

The address used by the POKE statement will indicate the physical space in the Commodore 64 where the information will be sent from the program. More than one number may be specified for a POKE address. On the Commodore 64, there are 65,536 valid POKE addresses. However, in this text we will use only a few of the available address numbers. Once you understand how to use these, it will be much easier for you to understand how other POKE addresses are used.

An address in computer programming is similar to the address of a house. The only way a 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 a POKE statement, and the address of the house will be analogous to the address or numbered location we specify on our CMS I/O circuit. A Commodore 64 will match the POKE address

with the I/O circuit address and deliver the information. (See Figure 2.6)

The second element in a POKE statement is the data, which is the actual digital information to be sent to the address specified. In our analogy of the address and the mail carrier, the data is the actual letter that was delivered.

Let's summarize the two parts of the POKE statement: the address specifies where in the entire system to send the information, and the data is the information sent. With this broad overview of the POKE statement, we can turn to some specifics.

2.3 Forming the POKE Address

Although we discussed the concept of the POKE address, we have not shown how to use it. In this section, we will show how to calculate the correct address, depending upon which I/O

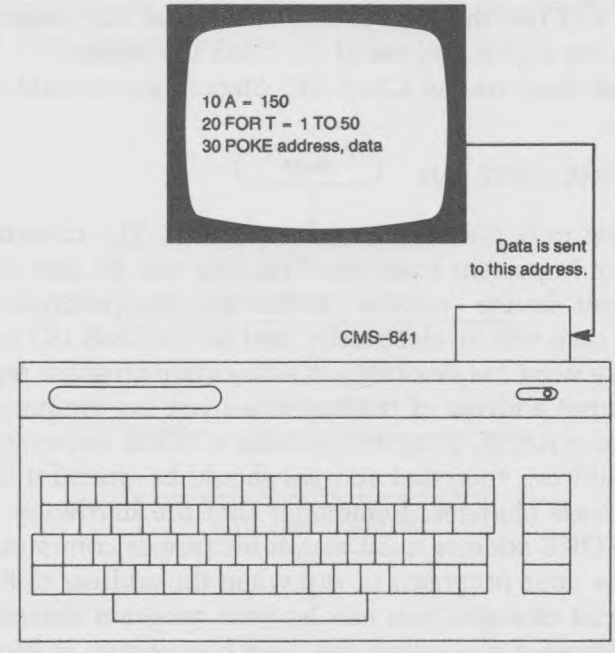


Figure 2.6: *When a POKE statement is used, the address specified will select the I/O expansion slot for communication.*

address you have chosen to use on the Commodore 64. Calculating an address can be much more complicated than we will show. However, if you are just starting to learn computer interfacing, this section will be a good entry point for you. Only when you begin to tackle more sophisticated interfacing problems will you need a greater understanding of address calculation.

In the I/O expansion slot on the Commodore 64, there are two I/O enable lines, labeled $\overline{\text{I/O1}}$ and $\overline{\text{I/O2}}$. (The rule over the label will be explained in section 4.2 of Chapter 4.) Each line has 256 unique addresses associated with them. $\overline{\text{I/O1}}$ has addresses from 56832 to 57087 inclusive. $\overline{\text{I/O2}}$ has addresses from 57088 to 57343 inclusive.

We can select the proper address for the POKE statement simply by knowing what the address is on the CMS I/O circuit. This holds true for any POKE statement. In order to send information, we must know whether the hardware we want to communicate with has an output address, and what the proper address is. The CMS I/O circuit will connect to $\overline{\text{I/O1}}$ line of the I/O expansion slot. We will use the first available address that asserts $\overline{\text{I/O1}}$, 56832, as the POKE address of the CMS I/O system.

To send data to the CMS I/O circuit you would use the statement:

```
POKE 56832,data
```

Notice that 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, the data will be electrically sent to the CMS I/O board.

You may want the flexibility of using your program regardless of the output address of the hardware you are communicating with. If your BASIC program includes a POKE statement with a specific address, then that address should be altered if the hardware address changes. Remember that the hardware address and the POKE address must match for proper communication.

To allow your programs to run when the address of the hardware circuit changes, you can let your program determine the POKE address if you follow the flowchart shown in Figure 2.7. This BASIC program prints a message asking which I/O address the user wishes to communicate with. Based on the answer, the

POKE statement will send data to the appropriate address.

Figure 2.8 shows one way to write a program to realize the flowchart of Figure 2.7. Remember that the address of the POKE statement can be a BASIC variable. After you enter the address, the form of the POKE statement would be:

```
POKE S1,data
```

where S1 corresponds to the address of a particular I/O circuit.

2.4 Calculating Data for the POKE Statement

Before we examine the details of calculating the data portion of the POKE statement, let's discuss exactly what the data will do. The output section of the Commodore 64 will use byte output. This means that there are eight electrical lines over which the Commodore 64 passes information to the external output circuits. (See Figure 2.9.)

Figure 2.7: *This flowchart shows the major steps to entering the POKE address as a BASIC variable.*

All information that can be transferred to the output device in a single POKE statement is included in these eight bits. A person using an output device with the Commodore 64 must remember what each bit does, when using the POKE statement. For example, one bit may turn on a light. Another may sound an alarm. Yet another may open a door. Further, all eight bits may be used together to form a unique combination to which the output device may respond.

In these early discussions, we will show how to set any bit

```
10 PRINT "INPUT I/O ADDRESS ";
20 INPUT S1
30 GOTO (LINE NUMBER OF MAIN PROGRAM)
```

Figure 2.8: *This BASIC program enters the I/O address as a numeric variable.*

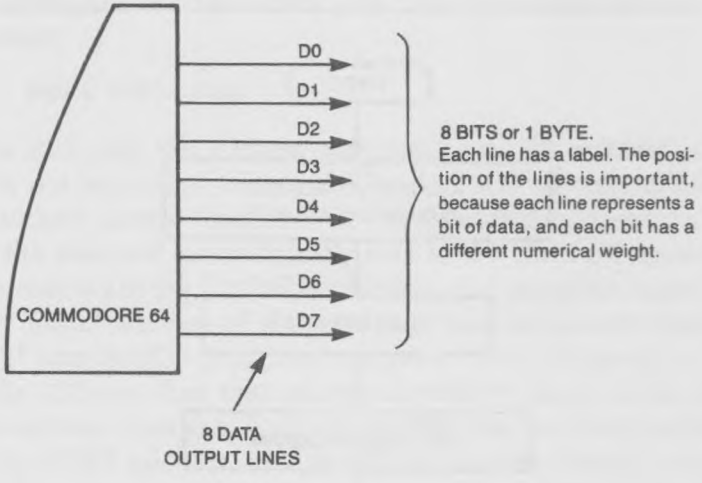


Figure 2.9: *This block diagram shows that there are eight physical output lines over which data will be transferred from the Commodore 64 to the external output device. The output lines originate from the I/O expansion slot.*

that will be used in the POKE statement to a logical 1 or a logical 0. Logical 1 and logical 0 were introduced in Chapter 1, but let's define them as precisely as possible now so that we can use them later.

We define logical 1 and logical 0 by the discrete voltage levels they correspond to. These definitions apply to the Commodore 64 and most other home computers. A logical 1 is a voltage level greater than 2.4 volts and less than 5.0 volts. Anytime 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 that 0.8 volts and greater than 0.0 volts.

For example, suppose that all eight bits or lines are a logical 1. This means that all are set to a voltage that will fall within the range specified for a logical 1.

With this brief introduction, let's discuss how any of the eight output lines can be set to a logical 1 or a logical 0 through software control. There are two key facts to remember. First, all eight bits are output to the I/O expansion slot in parallel; that is, all eight bits are output at the same time. This was illustrated in Figure 2.9. Second, 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 numbers 0–7 stand for the positions. D7 is the most significant bit (MSB), and D0 is the least significant bit (LSB).

Using these two pieces of information, let's examine the eight bits or lines. Remember from Chapter 1 that there are eight individual bits in a single byte.

Since the bits are 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. 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. For example, the weight of D0 is equal to 2^0, or 1. The weight of D7 is equal to 2^7, or 128. The bit positions and their corresponding weights are shown in Figure 2.10.

When we wish to set particular data bits to a logical 1, we add up the weights of the bits and use them as the data in a POKE

statement. For example, suppose that the bits D0 and D2 were set to a logical 1. We add up the weights of these bits (D0 = 1, and D2 = 4), and get the following result:

SUM = 1 + 4
= 5

The resulting value, 5, will be used in the POKE statement as data:

POKE address,5

Remember that we set the address according to the specific I/O address we want to send the data to.

If the POKE sum equals 5, the logical conditions of the data byte to be output would appear as in Figure 2.11. All the data bits used in the summation are set to a logical 1, while all others are set to a logical 0.

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

Let's try an example: suppose we wish to set data bits D0, D4, and D7 to a logical 1. Remember that we must add up the weights of the data bits. The weights of the bits we want are: D0 = 1, D4 = 16, D7 = 128. The resulting summation would

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

Figure 2.10: *The position of each bit is assigned a numerical weight, corresponding to its value as a power of 2.*

be 1 + 16 + 128 = 145. The POKE statement we would use appears as follows:

```
POKE address,145
```

Here's another example: suppose we wish to set data bits D2, D5, and D6 to a logical 1. We would again add the weights of the bits we want: D2 = 4, D5 = 32, and D6 = 64. The resulting summation would be 4 + 32 + 64 = 100. The POKE statement would be:

```
POKE address, 100
```

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

Before we experiment with the CMS I/O board, let's examine one more aspect of setting the correct data. We can write a program to set a bit to a logical 1 or logical 0. It is similar to the program that inputs an address. Figure 2.12 shows a flowchart for this program.

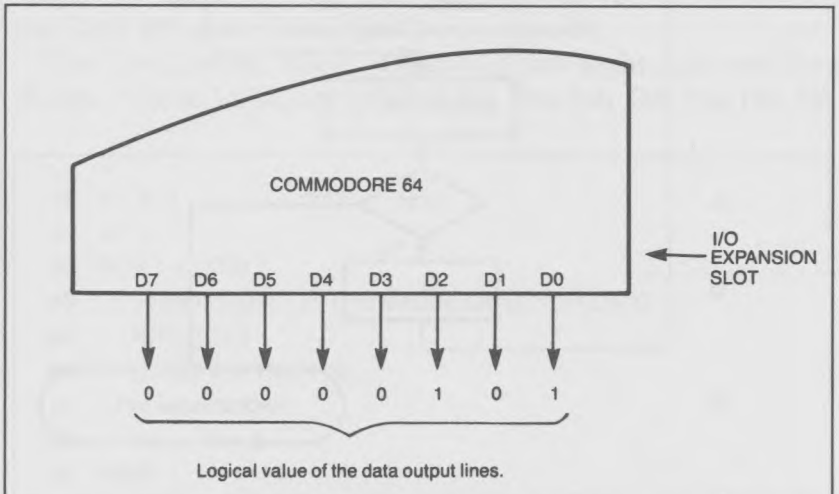


Figure 2.11: *This diagram shows the logical conditions of the eight output lines that would result if the POKE data equals 5.*

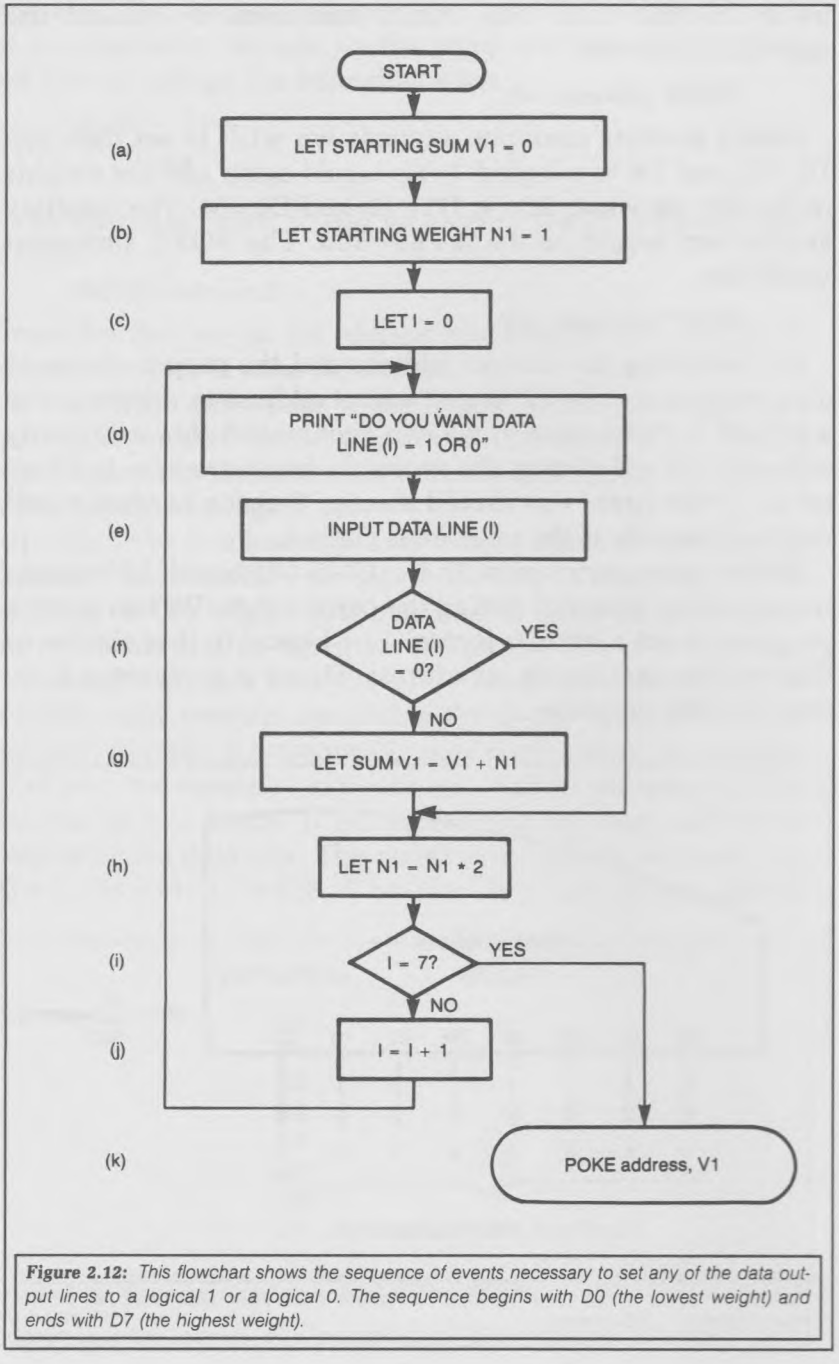


Figure 2.12: *This flowchart shows the sequence of events necessary to set any of the data output lines to a logical 1 or a logical 0. The sequence begins with D0 (the lowest weight) and ends with D7 (the highest weight).*

Let's examine this flowchart in detail. 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 user wants the bit to be set to a logical 1, then the weight of the bit will be added to the 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, which will set the weight equal to the next bit to be tested. After V1 is computed, the POKE statement will appear as:

```
POKE address,V1
```

A program to realize the flowchart of Figure 2.12 is shown in Figure 2.13.

2.5 Experiments with the CMS I/O System

In this section, we will write and execute programs that use the POKE statement to perform output on the Commodore 64. All of the following examples assume that you have installed the CMS I/O system into the Commodore 64.

The CMS output board (CMS-642) has eight light-emitting diodes. These LEDs are labeled D0, D1, D2, D3, D4, D5, D6,

```
10  V1 = 0
20  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 90
70      V1 = V1 + N1
80      N1 = N1*2
90  NEXT I
100 POKE address, V1
```

Figure 2.13: *This BASIC program realizes the flowchart given in Figure 2.12.*

and D7, as shown in Figure 2.14. A lighted LED indicates that the corresponding bit position in the data byte is set to a logical 1. An unlighted LED means that the corresponding bit position is a logical 0.

For example, Figure 2.15(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 2.15(b).

2.6 Example 1: Lighting a Single LED

In this first hands-on example of Commodore 64 output, we will write a program that will turn on any of the eight LEDs located on the CMS I/O board.

In this program, we will assume that the POKE address is 56832. The program will then ask you which LED you want to light. Finally, the program will light the selected LED on the CMS I/O board. Figure 2.16 shows the flowchart for this program. Let's examine each step before we look at the program listing:

Step 1. The program will write a message asking the user what the I/O address is.

Step 2. The user will input the correct I/O address.

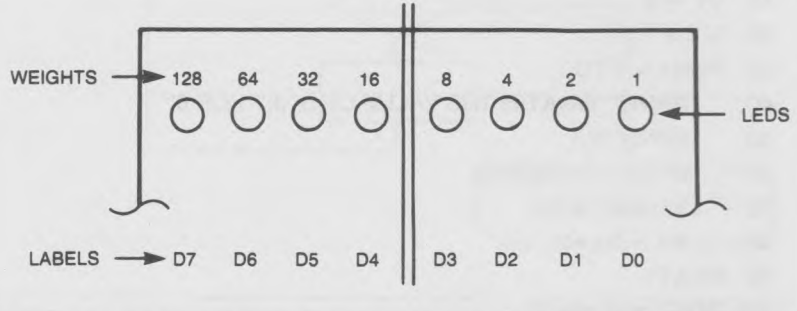


Figure 2.14: *The output section of the CMS I/O board has eight LEDs. The weight and data-line label of each LED are also shown.*

Step 3. The program will write a message asking which LED the user wishes to light.

Step 4. The user will input the LED number.

Step 5. The program will set the output byte to the correct weight that corresponds to the LED that the user wants to light.

Step 6. The program will set the address to the correct value that corresponds to the LED.

Step 7. The software will now output the correct byte to the I/O device.

A program to realize the flowchart of Figure 2.16 is presented in Figure 2.17.

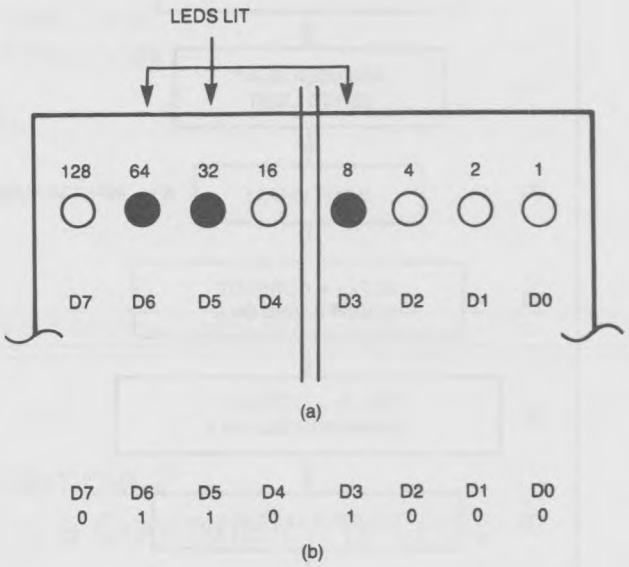


Figure 2.15(a): *LEDs D6, D5, and D3 are lit. All other LEDs are off. The number input to produce this combination was 104.*

(b): *These are the corresponding logical levels of the data lines needed to produce the LED pattern shown in (a).*

In examining Figure 2.17, note that if a valid LED number in the range from 0–7 is not input in step 2, all of the LEDs on the CMS I/O board will light. This indicates an error. As an alternative, we could change the program to check for this error condition by writing another loop.

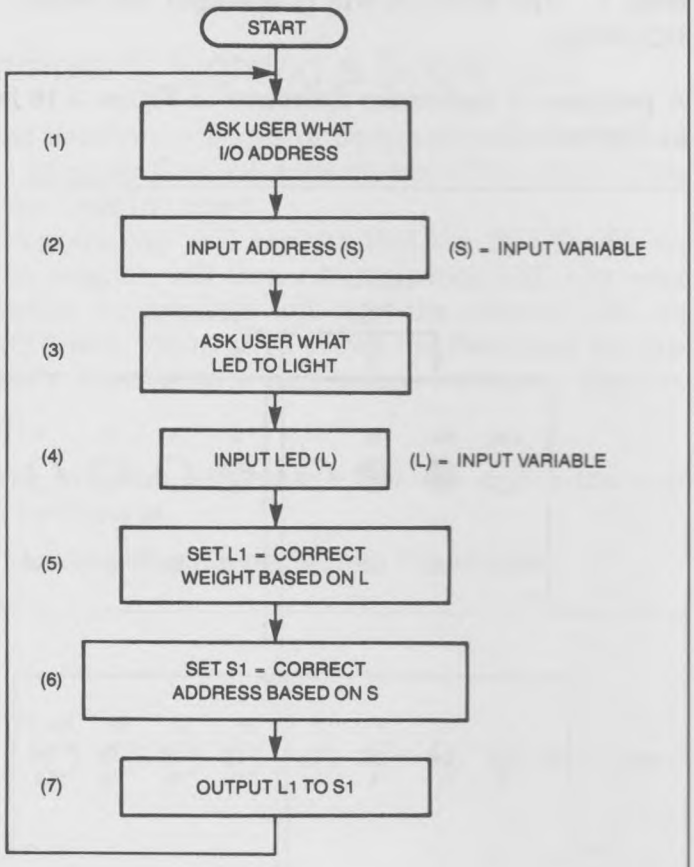


Figure 2.16: *This is a flowchart for inputting the I/O address and the LED to be turned on at the external I/O device.*

```
10  REM THIS PROGRAM WILL OUTPUT A BYTE TO THE CMS I/O BOARD
20  REM THAT IS INSTALLED IN A COMMODORE 64 COMPUTER.
50  REM START OF THE PROGRAM
60  PRINT "WHAT I/O ADDRESS? ";
70  INPUT S
80  PRINT "WHICH LED ON THE CMS I/O BOARD DO YOU WANT LIT? ";
90  INPUT L
100 L1 = 255
110 IF L = 0 THEN L1 = 1
120 IF L = 1 THEN L1 = 2
130 IF L = 2 THEN L1 = 4
140 IF L = 3 THEN L1 = 8
150 IF L = 4 THEN L1 = 16
160 IF L = 5 THEN L1 = 32
170 IF L = 6 THEN L1 = 64
180 IF L = 7 THEN L1 = 128
190 REM
200 POKE S,L1
210 PRINT
220 GOTO 80
```

Figure 2.17: *This is the BASIC program to realize the flowchart given in Figure 2.16.*

2.7 Example 2: Lighting a Combination of LEDs

The second hands-on example will allow the user to light any combination of LEDs on the CMS I/O board. The program asks you which LEDs you want on, and then what the necessary, output variable weight is to accomplish this.

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

weight of D0 (which is 1) plus the weight of D5 (32) plus the weight of D7 (128). This equals 1 + 32 + 128 = 161. If the number 161 were output to the correct address, LEDs D0, D5, and D7 are set to a logical 1 in the output data byte. Figure 2.18 shows a flowchart to accomplish this example and Figure 2.19 is the program based on the flowchart.

Using LED patterns a–g on the next page, calculate each weight to be input. You can visually verify your answer by

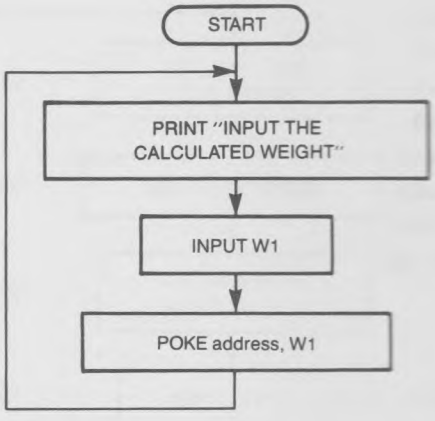


Figure 2.18: *This flowchart inputs the weight to POKE at the output connector.*

```
20 REM FIRST INPUT THE CALCULATED WEIGHT
30 PRINT "INPUT THE CALCULATED WEIGHT"
40 INPUT W1
50 POKE 56832,W1
60 GOTO 30
```

Figure 2.19: *This is the BASIC program to realize the flowchart given in Figure 2.18.*

executing the program of Figure 2.19. The answers are given at the end of the list.

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.

The answers to the LED patterns are:

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 the following sequence. At first, all of the LEDs will be turned off. The number 1 then will be output from the computer to the I/O board. This will turn on only the LED that corresponds to a weight of 1, which is D0. The program will pause long enough for you 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 LED D1, which corresponds to a weight of 2. Again, the program will delay long enough for you to see the LED pattern on the I/O board.

The program now outputs the number 3, which turns on the LEDs corresponding to a weight of 3. LEDs D0 and D1 will be lit, and you can check the pattern during the pause.

This process is repeated, each time incrementing the number to be output by 1. The result will be the LEDs turning on and off 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 2.20 and the corresponding BASIC program is given in Figure 2.21.

If you know more BASIC than we have used so far, you might

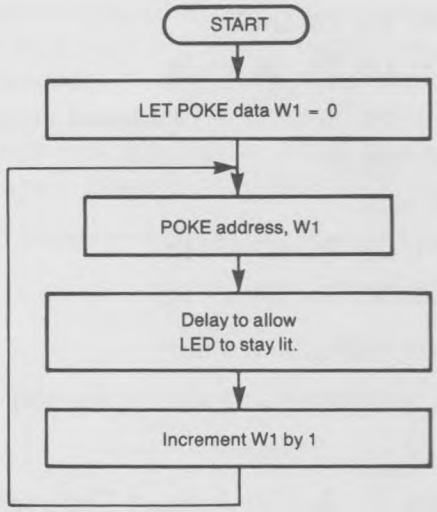


Figure 2.20: *This is the flowchart for a counting program.*

```
 20 REM COUNTING PROGRAM FOR THE COMMODORE 64
 30 REM SET THE FIRST POKE DATA EQUAL TO 0
 40 W1 = 0
 50 POKE 56832,W1
 60 REM
 70 FOR I = 1 TO 1000
 80 NEXT I
 90 REM WE JUST DELAYED FOR A WHILE
100 W1 = W1 + 1
110 IF W1 > 255 THEN W1 = 0
120 GOTO 50
130 END
```

Figure 2.21: *This is the BASIC program to realize the flowchart of Figure 2.20.*

try the following variations. After you have had a chance to load and run the program in Figure 2.21, try to get the same results using fewer BASIC statements. Next, try speeding up and slowing down the pauses in the program and note the effect on the output display LEDs.

2.9 Example 4: A Traveling Light

In our final example, we will write a program to make the LEDs turn on and off in a 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 as on 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: we will output a byte to the CMS I/O board and then pause before outputting another.

The program starts by outputting the number 1 to light only LED D0. Next, to light D1, the program outputs the number 2. The number 4 is output next, since it will light only LED D2. Therefore, the program should output only a number that corresponds to a single bit weight. The remaining numbers to be output will be 8, 16, 32, 64, and 128. Figure 2.22 and Figure 2.23 give a flowchart and a BASIC program for this example.

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

1. Make the light travel in the opposite direction from the way it is now going.
2. Make the light "bounce": When it gets to D7, make it travel back to D0. When it gets to D0, make it travel back to D7.
3. Make the light travel from the center LEDs to each edge. 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.
4. Make the light bounce from the center to the edges and back to the center again.

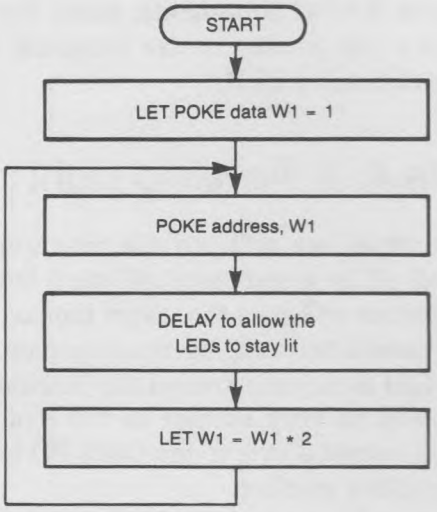


Figure 2.22: *This flowchart is for a traveling-light program.*

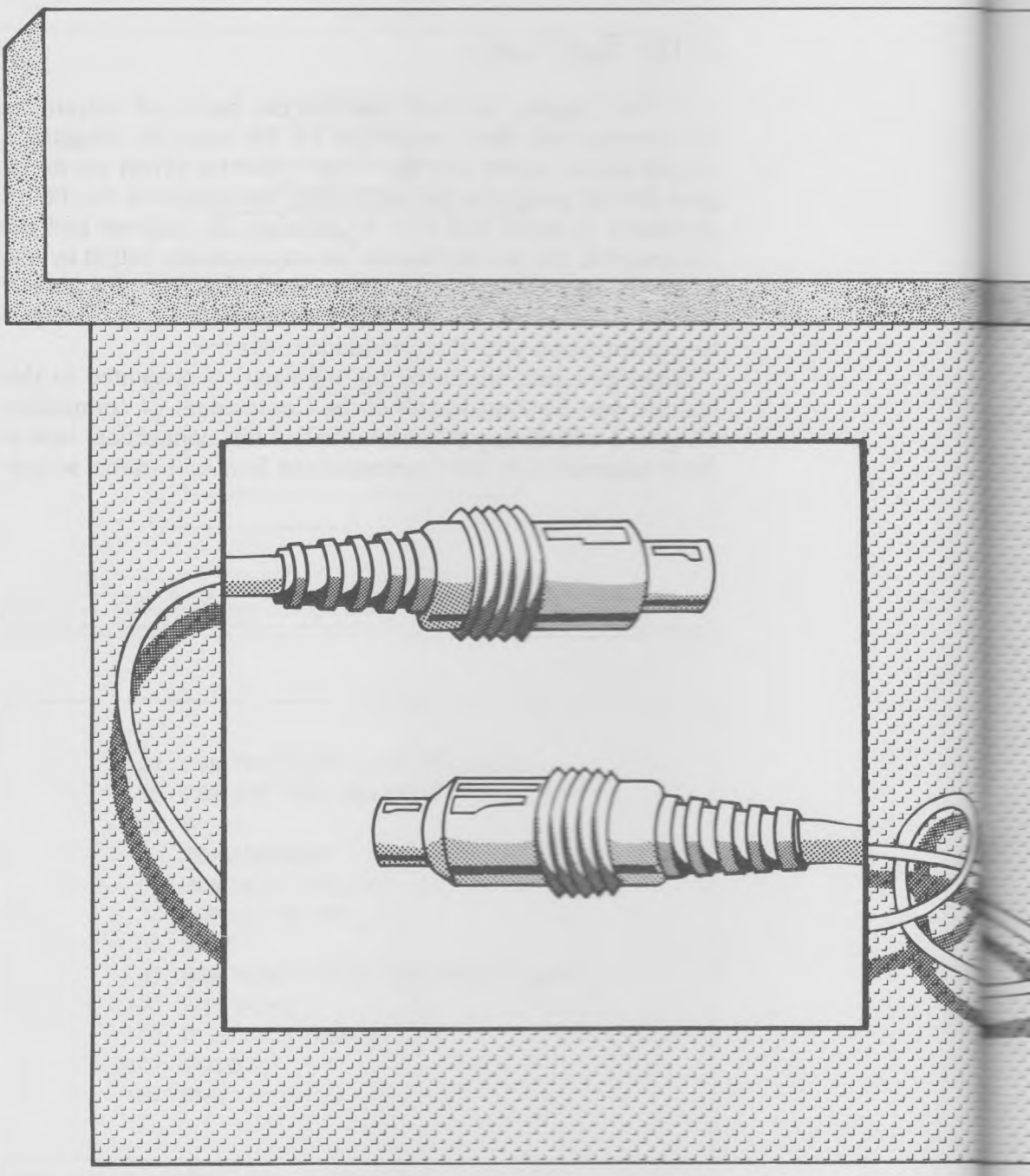
```
 20 REM TRAVELING LIGHT PROGRAM
 30 REM SET FIRST POKE DATA EQUAL TO 1
 40 W1 = 1
 50 POKE 56832,W1
 60 REM NOW TO DELAY
 70 FOR I = 1 TO 1000
 80 NEXT I
 90 REM NOW TO SHIFT THE DATA BIT LEFT
100 W1 = W1 * 2
110 IF W1 > 128 THEN W1 = 1
120 GOTO 50
130 END
```

Figure 2.23: *This is the BASIC program to realize the flowchart of Figure 2.22.*

2.10 Summary

In this chapter, we have covered the basics of outputting information with the Commodore 64. We began by plugging a simple output device into the Commodore 64. When we examined BASIC programs for outputting, we discussed the POKE statement in detail and how to calculate the address and the data needed. We covered how to set any bit in the output byte to a logical 1 or a logical 0. Finally we looked at four "hands-on" examples that allow you to verify your understanding of outputting information with the Commodore 64.

When you have mastered the information presented in this chapter, you have made half of the Commodore 64 connection. Chapter 3 will discuss the second half of this connection: how to input information to the Commodore 64 from an external source.



INPUTTING DATA TO THE COMMODORE 64

3

In this chapter, we will discuss how to input digital information using BASIC programs to the Commodore 64 from an external device connected to the I/O expansion slot. 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, let's review what our goal is. By referring to the block diagram in Figure 3.1, we see that an external device will send digital information (data) to the Commodore 64. 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, you may have questions, such as "How does the Commodore 64 electrically know that the external device is ready to send information?" The answer to the question is covered in the broad topic called *handshaking*. Generally, handshaking refers to the organized process by which either an external device or a computer informs the other of its condition. For example, the external device uses a handshake line to indicate to the computer that it has data ready to send. Conversely, the computer uses another handshake line to inform the external device that it is ready to input data.

There are other types of handshaking systems, including interrupts and Direct Memory Access (DMA), but they are beyond the scope of this book. However, in Chapter 9, we discuss another handshaking system in detail. For now, we will assume

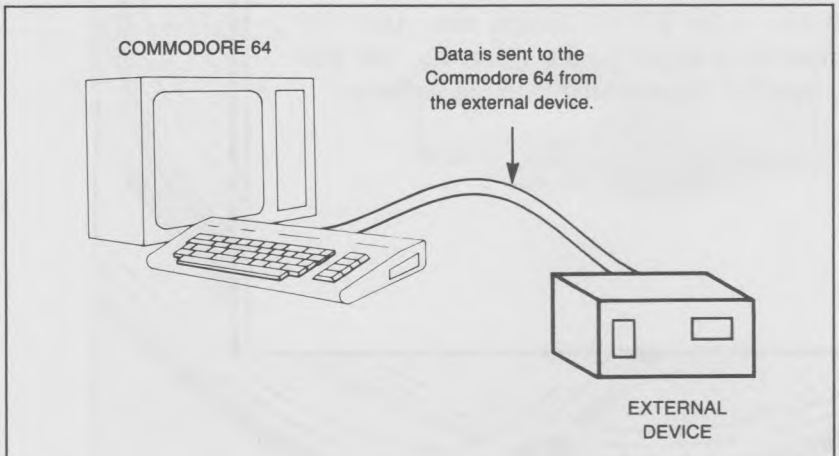


Figure 3.1: *The object of inputting data is to send information from an external device to the Commodore 64. The information must be in an electrical form that the computer will understand.*

that the data from the external device is always ready to send when the computer requests it.

Now let's turn to the topic of this chapter: how to transfer, or input, information from an external device to the Commodore 64 using a BASIC program. Once we transfer the data, we will examine ways of processing it with software to make logical decisions.

3.2 The CMS Input Board for the Commodore 64

The physical connection used for inputting the electrical data to the Commodore 64 will be made through the CMS I/O system. We covered this I/O board in Chapter 2, when we discussed the mechanics of outputting data. We will assume that you have installed the CMS I/O system according to Section 2.1 of Chapter 2. Besides functioning as an output device, the CMS I/O system is also electrically capable of inputting data to the computer. Figure 3.2 shows a pictorial diagram of the input

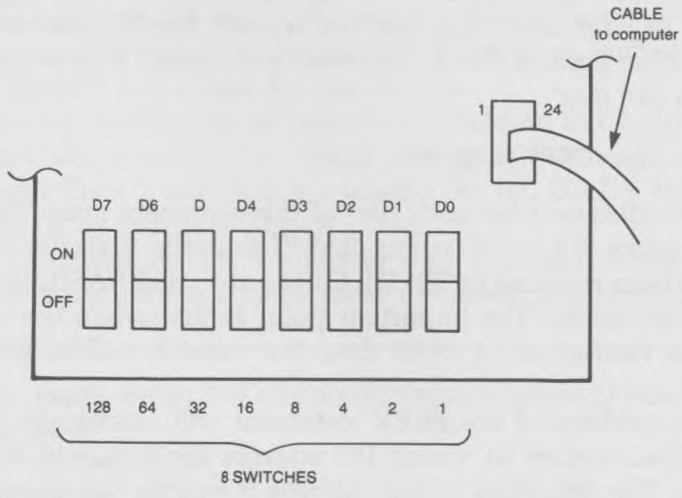


Figure 3.2: *This pictorial diagram shows the physical switches on the CMS I/O system input section. Each switch is marked with a "D" label and its corresponding weight, and is either "ON" or "OFF."*

section of the CMS I/O board hardware, located on board CMS-642.

In Figure 3.2, there are eight switches on the CMS I/O board input section. These switches are either *off* or *on*. 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.

Each switch in Figure 3.2 has a unique label: D0, D1, D2, D3, D4, D5, D6, and D7. The label corresponds to the input line to the Commodore 64.

The switches in the CMS I/O system input section operate in parallel. This means that any signal line, D0–D7, can be set to a logical 1 or a logical 0, independent of any other signal line.

3.3 Input Software

Let's look at how a programmer using BASIC can electrically request information from the external device. The statement used to accomplish the inputting of information is PEEK. Like the POKE statement we used in Chapter 2, PEEK is explained fully in your user's manual, but we will briefly summarize its operation here. A PEEK statement will appear in a BASIC program like this:

```
A1 = PEEK (address)
```

Let's discuss what each part of this statement does. The variable name A1 could be any BASIC numeric variable; it could have been replaced by T1, Z5, C(3) or any valid BASIC numeric-variable name. The important point is that when we use the PEEK instruction to input data, the variable will be set equal to it.

The address of the PEEK statement will electrically inform the Commodore 64 which I/O address the data will be input from. The definition of this address is exactly the same as the POKE address definition we discussed in Chapter 2. Remember that we can input information from any I/O circuit simply by using the correct I/O address, which is 56832 for the CMS I/O

system. In a BASIC program, the statement that allows this to happen is:

```
A1 = PEEK (56832)
```

After the computer executes this statement, A1 will equal the information that was input from address 56832.

Let's look at another example. Suppose the I/O expansion slot address is itself a variable. The BASIC program will ask the user for the I/O address, which will be stored in a BASIC variable. For this example, the I/O address was stored in the variable S3. The form of the PEEK statement would be:

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

This will let A1 equal the information read from the I/O address S3. Figure 3.3 shows a BASIC program for this example.

3.4 Interpreting the Input Information

So far, we have covered how to store the input information in a valid variable in a BASIC program. This section will focus on ways to interpret the input information. We will use the information on calculating data for the POKE statement from section 2.4 of Chapter 2 as a base for this section.

When the Commodore 64 inputs data from an I/O address, it is electrically inputting the logical voltage levels of eight separate signal lines. These lines are labeled D0, D1, D2, D3, D4, D5, D6, and D7 as shown in Figure 3.4.

The computer assigns a numeric weight to each signal line. These weights are the same as those discussed in Chapter 2 and

```
10 PRINT "INPUT THE I/O ADDRESS FOR COMMUNICATION";
20 INPUT S3
30 A1 = PEEK(S3)
40 END
```

Figure 3.3: *This BASIC program asks the user which I/O address to input data from.*

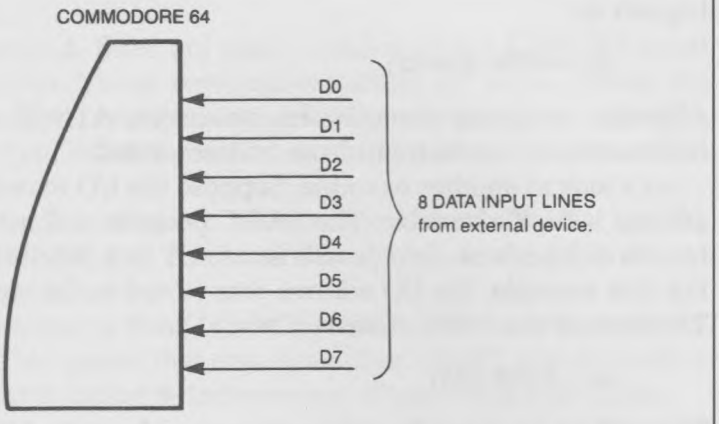


Figure 3.4: *Eight separate signal lines are connected to the Commodore 64 I/O expansion slot in order to input external information.*

listed in Figure 2.10. (Remember that these weights correspond to binary numbers. You may find that a review of the binary numbering system will help you understand how your computer functions.) The weight of a signal line is summed when it is a logical 1. But its weight is not summed when it is a logical 0. The sum is the value that will be stored in the BASIC variable in the PEEK statement.

For example, suppose the external hardware was connected to I/O address 56832. We would use 56832 as the PEEK address. Let's assume that the external 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. The device sending the data to the computer will set the external input lines. (See Figure 3.5.)

To obtain the information from the external device used in this example, the PEEK statement we use is:

```
A1 = PEEK (56832)
```

After this statement is executed, A1 will equal the sum of the weights of all data input lines, D0–D7, at I/O address 56832. The sum will include the weights of the lines that were a logical 1 during the execution of the PEEK statement.

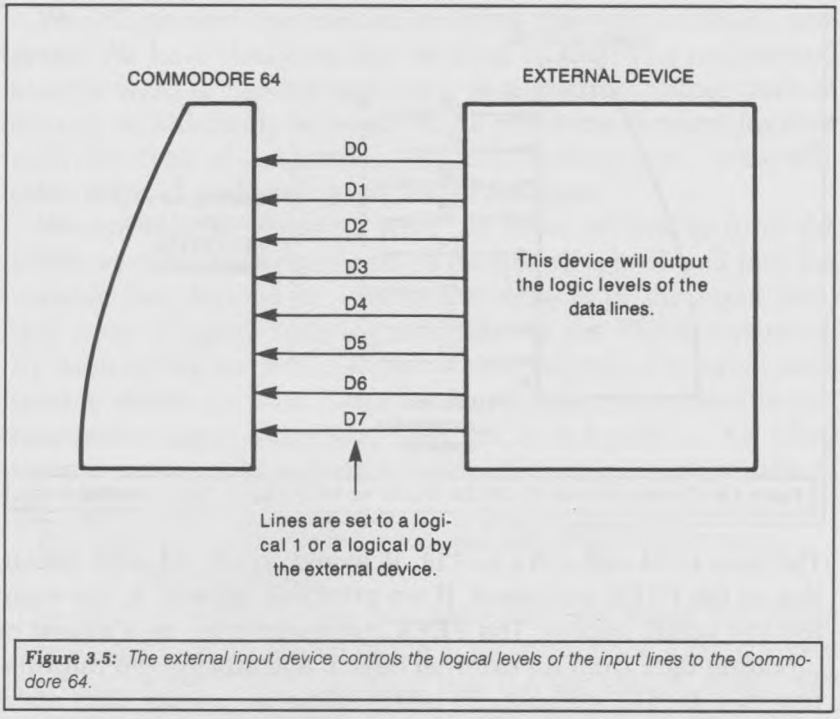


Figure 3.5: *The external input device controls the logical levels of the input lines to the Commodore 64.*

In our example, the weights that will be added are 128 (for D7), 32 (D5), 16 (D4), and 1 (D0). The result is 128 + 32 + 16 + 1 = 177. A1 equals 177 after the execution of the PEEK statement.

The variable in the PEEK statement is greater than or equal to 0, and less than or equal to 255. A1 equals 0 when all input lines are set to a logical 0 at the I/O address, and A1 equals 255 when all input lines are a logical 1.

Let's consider another example: the input device sets data lines D6, D5, D4, and D1 to a logical 1. All other data input lines are set to a logical 0. This is shown in Figure 3.6. The input device is connected to I/O address 56832. Our PEEK statement to read the input information will appear as:

```
R = PEEK (56832)
```

What will be the value of R after the PEEK statement is executed? Remember R equals the sum of the weights of all input lines set to a logical 1 during the execution of the PEEK statement. These weights are: D6 = 64, D5 = 32, D4 = 16, and D1 = 2.

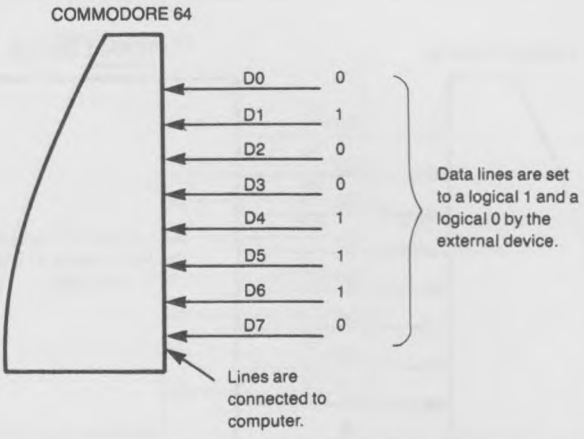


Figure 3.6: *External input lines D1, D4, D5, and D6 are set to a logical 1 by the external device.*

The sum is 64 + 32 + 16 + 2 = 114. R would equal 114 after execution of the PEEK statement. If we print the value of R, the number 114 would appear. The PEEK statement gives us a means of inputting data from an external device and storing the information in a BASIC variable. By using software, we can then operate on the variable in the same way as any other BASIC variable.

3.5 Calculating the Bits from the Input Variable

We have discussed how you can calculate the value of the input variable in a PEEK statement, based on the assumption that we knew the logical level of the data input lines at the selected I/O address. However, in many applications of computer control, we do not know which data input lines were a logical 1 and which were a logical 0 when we use a PEEK statement. We need to know this because each line input by the external device may mean something different. For example, D0 may signal that the temperature is too high, D1 may indicate that some lights in a home were left on, and so on. We need a way to find out the logical conditions (1 or 0) of each data line input during the PEEK statement. We can accomplish this by using software.

We will present one method of doing this with a BASIC program. We have designed this method to help you understand exactly what is needed and what is occurring, rather than to operate as efficiently as possible. As you become more familiar with this type of processing, you can develop new, more efficient ways of performing the transformation.

We can start by applying what we know of how to form the PEEK variable to the problem of deciphering it. Recall that the variable was formed by adding the weights of the input lines that were a logical 1 during execution of the PEEK statement. By subtraction, we will discover which individual weights were used to obtain the sum. Once we know these, we know the corresponding input lines that were set to a logical 1. All other input lines must have been a logical 0 during the execution of the PEEK statement.

For example, suppose we executed this PEEK statement:

```
A = PEEK (56832)
```

The variable A would then be a value between 0 and 255, inclusive, depending on which of the eight input lines, D0–D7, were set to a logical 1. Let's assume that the variable A equals 183 after the PEEK statement. We can first see that at least one of the input lines was set to a logical 1, because A does not equal 0. We also know that at least one of the input lines is a logical 0, because A is not 255. But which input lines were set to a logical 1 during the PEEK statement, and which were a logical 0?

Our first job is to determine which of the input lines was a logical 1 during the input instruction. This can be done by subtracting the weight of each input line, starting with the weight of D7, from the variable A. If the result is less than 0, we know that that particular weight was not used to obtain the sum.

Let's look at an example: suppose a PEEK variable equals 125. In our plan, the weight of D7 is subtracted from 125. This is $125 - 128 = -3$, which is less than 0. This tells us that the weight of D7 was not added to the original sum. Therefore, D7 is a logical 0.

In the next step, we then subtract the weight of D6 from 125. The result is $125 - 64 = 61$, which is not less than 0. This means that the weight of D6 was used in the original summation and that D6 is a logical 1.

When we find that a weight was used in the summation, its value is subtracted from the PEEK variable and the result is tested against the next weight in line. In our example, the next weight, D5, is subtracted from the new value of 61, and not the original value of 125. This gives $61 - 32 = 29$, which is not less than 0. Therefore, the weight of D5 was used to obtain the original sum and D5 is a logical l.

So far, we know that the weights of D6 and D5 were used in obtaining the original sum of 125. To find our new value for our test, we subtract the known weights of D6 and D5 from 125. The new value is 29.

Our next step is to subtract the next weight, D4, from 29. Our result is $29 - 16 = 13$. Since this result is not less than 0, we know that the weight of D4 was used in obtaining the original sum and D4 is a logical 1.

The next weight in line is D3, which we subtract from our new value, 13. The result of $13 - 8 = 5$ is more than 0. We know that the original sum used the weight of D3 and that D3 is a logical 1.

The next weight, D2, is tested against the new value, 5. The result, $5 - 4 = 1$, is a number that is more than 0. Again, we know that the weight of D2 was used in the original sum.

We next test the weight of D1. However, this result, $1 - 2 = -1$, is less than 0. Because of this, we know that the weight of D1 was not used to obtain the original sum.

Finally we test D0. Since subtracting the weight of D1 gave us a negative result, we keep the testing value of 1. When we subtract the weight of D0, 1, the result is $1 - 1 = 0$, which is not less than 0. We know that D0 was used to obtain the original sum of 125.

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 lines were a logical 1 during the execution of the PEEK statement. We also know that the input lines D7 and D1 were a logical 0.

This example shows us how the weights of the data lines are tested. If the result becomes 0 during our test, then we can stop testing because it means that all the remaining data lines must be a logical 0. With 183 as the value of the variable from the

PEEK statement, let's consider another example, in order to outline the steps to find out which data lines were a logical 1 and which data lines were a logical 0:

Step 1. Subtract the weight of D7, 128, 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 and D7 was a logical 1 during the PEEK statement. We set the variable A2 equal to 55.

Step 2. We now subtract the bit weight of D6, 64, from the new value of variable A2.

55 – 64 = –9

The result is less than 0. This means that the bit weight of D6 was not used for the original sum of 183. D6 was a logical 0 during the PEEK statement.

Step 3. We next subtract the weight of D5, 32, from the variable A2. Remember that we do not change A2 because the result was less than 0 in step 2.

55 – 32 = 23

Since the result is not less than 0, we know that this weight was used for the original sum, and D5 was a logical 1 during the PEEK statement. We set the variable A2 to 23 because the result of the subtraction was positive.

Step 4. Next we subtract the bit weight of D4, 16, from the variable A2.

23 – 16 = 7

Since the result is not less than 0, we know that the bit weight of D4 was used for the original sum and D4 was a logical 1 during the PEEK statement. The variable A2 is now equal to 7.

Step 5. We subtract the bit weight of D3, 8, from the new variable A2.

7 – 8 = –1

The result is less than 0. Therefore, we know that the weight

of D3 was not used in obtaining the original sum of 183, and that D3 was a logical 0 during the PEEK statement.

Step 6. When we test the next bit weight, D2, the result is:

7 – 4 = 3

Since this means that the weight of D2 was used for the sum, input line D2 was a logical 1 during the PEEK statement. A2 now becomes 3.

Step 7. Now we test bit D1. The result is:

3 – 2 = 1

D1 was a logical 1 during the PEEK statement.

Step 8. The final bit to test is D0. The result is:

1 – 1 = 0

Since the result is not less than 0, the weight of D0 was used to obtain the original sum, and D0 is a logical 1.

We can summarize our results like this:

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

There is a simple way to check our results. Add up the weights of all bits that are a logical 1. The result should be the value of our variable, 183.

```
  128 = D7
+  32 = D6
+  16 = D4
+   4 = D2
+   2 = D1
+   1 = D0
-------------
  183   The result checks.
```

The procedure may seem quite tedious when we read over the steps. Fortunately, we can use the computer to make these checks. The first step in writing a program for this procedure is to design a flowchart as shown in Figure 3.7.

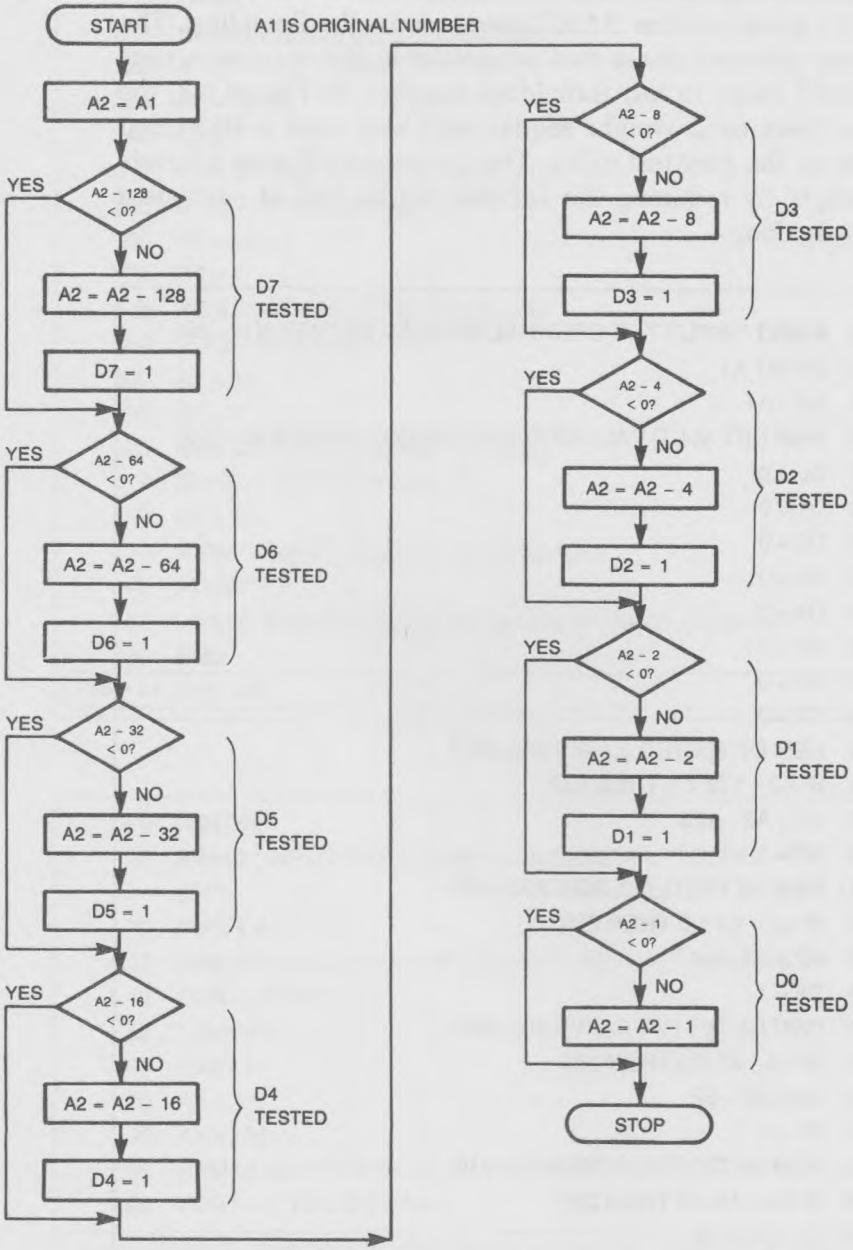


Figure 3.7: *This flowchart checks the logical value (1 or 0) of the external data input lines to the Commodore 64. The data was input by using a PEEK statement.*

A BASIC program for this flowchart is shown in Figure 3.8. Figure 3.9 gives another BASIC program for the flowchart. The difference between these two programs is the way each uses FOR/NEXT loops to test individual weights. In Figure 3.8, the program tests each weight sequentially and uses a statement that sets up the new test value. The program in Figure 3.9 tests each weight by reducing the variable A3 by half at each pass through the loop.

```
 10 PRINT "INPUT THE ORIGINAL NUMBER BETWEEN 0 - 255 "
 20 INPUT A1
 30 A2 = A1
 35 REM SET ALL D VALUES EQUAL TO ZERO (LINES 40-110)
 40 D0 = 0
 50 D1 = 0
 60 D2 = 0
 70 D3 = 0
 80 D4 = 0
 90 D5 = 0
100 D6 = 0
110 D7 = 0
195 REM D7 TESTED (LINES 200-220)
200 IF A2 - 128 < 0 THEN 230
210 A2 = A2 - 128
220 D7 = 1
225 REM D6 TESTED (LINES 230-250)
230 IF A2 - 64 < 0 THEN 260
240 A2 = A2 - 64
250 D6 = 1
255 REM D5 TESTED (LINES 260-280)
260 IF A2 - 32 < 0 THEN 290
270 A2 = A2 - 32
280 D5 = 1
285 REM D4 TESTED (LINES 290-310)
290 IF A2 - 16 < 0 THEN 320
300 A2 = A2 - 16
```

Figure 3.8: *This BASIC program realizes the flowchart of Figure 3.7.*

```
310  D4 = 1
315  REM D3 TESTED (LINES 320–340)
320  IF A2 – 8 < 0 THEN 350
330  A2 = A2 – 8
340  D3 = 1
345  REM D2 TESTED (LINES 350–370)
350  IF A2 – 4 < 0 THEN 380
360  A2 = A2 – 4
370  D2 = 1
375  REM D1 TESTED (LINES 380–400)
380  IF A2 – 2 < 0 THEN 410
390  A2 = A2 – 2
400  D1 = 1
405  REM D0 TESTED (LINES 410–420)
410  IF A2 – 1 < 0 THEN 430
420  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
```

Figure 3.8 *(continued)*

```
10  DIM D(8)
20  PRINT "INPUT THE ORIGINAL NUMBER BETWEEN 0 AND
    255"
30  INPUT A1
35  REM SET ALL D VALUES EQUAL TO ZERO (LINES 40–60)
40  FOR I = 0 TO 7
50    D(I) = 0
60  NEXT I
70  A2 = A1
80  A3 = 128
85  REM LOOP TO TEST ALL D VALUES (LINES 90–140)
90  FOR I = 7 TO 0 STEP – 1
```

Figure 3.9: *This is a shorter BASIC program to realize the flowchart of Figure 3.7. In lines 80–130, the variable A3 is the weight of D7. This variable is then reduced by half in each pass through the FOR/NEXT loop.*

```
100    IF A2 - A3 < 0 THEN 130
110    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 ";
180  FOR I = 7 TO 0 STEP -1
190    PRINT D(I);
200  NEXT I
210  PRINT
220  END
```

Figure 3.9 (continued)

We can now determine which data input lines were a logical 1 and which data input lines were a logical 0 during the execution of a POKE instruction. Let's turn to some examples of inputting data.

3.6 Example 1: Calculating the Weight of the Input Word

In this example, we will calculate the value of the input weights by hand, according to the procedure described in the last section. After you calculate the weight, set the switches on the CMS I/O board to the correct value and run the program given in Figure 3.10. This program prints out the weight calculated by the computer so that you can verify whether your manual calculation was correct.

Let's go through these steps together on an example. First, set D0 and D4 to a logical 1. All other switches are a logical 0. Remember that the switch is a logical 0 when in the OFF position, and a logical 1 when in the ON position.

Now let's calculate the input weight. The weight of D0 is 1 and the weight of D4 is 16. Therefore the input weight is 1 + 16 = 17.

```
10 PRINT "INPUT THE I/O ADDRESS FOR COMMUNICATION"
20 INPUT S3
30 A1 = PEEK(S3)
40 PRINT "THE DATA READ FROM I/O ADDRESS # ";S3;" = ";A1
50 END
```

Figure 3.10 (continued)

When we run the program, the computer reads this data and prints out the number 17. At this point we can check our calculations against the computer's. You can use this check to verify whether or not you understand how the input lines are weighted.

Try the program in Figure 3.10, using the different switch settings, a–f. Start by setting the switches to a logical 0. The answers are given at the end of the list.

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, and D7 are set to 1

e. All switches are set to a logical 1

f. All switches are set to a logical 0

The answers are:

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 the same type of procedure discussed in Section 3.5. A general program for the Commodore 64 to do this is shown in Figure 3.11. Run this program to verify that the computer

prints out the correct data lines that you set on the CMS I/O board switches. Use the same switch settings that were listed in Section 3.6.

```
10  PRINT "INPUT THE I/O ADDRESS FOR THE CMS BOARD "
20  INPUT S3
30  A1 = PEEK(S3)
40  A2 = A1
50  REM INITIALIZE THE VARIABLES
60  D0 = 0:D1 = 0:D2 = 0:D3 = 0:D4 = 0:D5 = 0:D6 = 0:D7 = 0
70  IF A2 - 128 < 0 THEN 100
80  A2 = A2 - 128
90  D7 = 1
100 IF A2 - 64 < 0 THEN 130
110 A2 = A2 - 64
120 D6 = 1
130 IF A2 - 32 < 0 THEN 160
140 A2 = A2 - 32
150 D5 = 1
160 IF A2 - 16 < 0 THEN 190
170 A2 = A2 - 16
180 D4 = 1
190 IF A2 - 8 < 0 THEN 220
200 A2 = A2 - 8
210 D3 = 1
220 IF A2 - 4 < 0 THEN 250
230 A2 = A2 - 4
240 D2 = 1
250 IF A2 - 2 < 0 THEN 280
260 A2 = A2 - 2
270 D1 = 1
280 IF A2 - 1 < 0 THEN 300
290 D0 = 1
300 PRINT
310 PRINT "THE BINARY VALUES OF THE INPUT WERE "
```

Figure 3.11: *This BASIC program reads the input data and then prints out the logical value of each data line, D0–D7.*

```
320 PRINT
330 PRINT "D7 = ";D7
340 PRINT "D6  = ";D6
350 PRINT "D5  = ";D5
360 PRINT "D4  = ";D4
370 PRINT "D3  = ";D3
380 PRINT "D2  = ";D2
390 PRINT "D1  = ";D1
400 PRINT "D0  = ";D0
410 END
```

Figure 3.11 *(continued)*

After you have verified that the program works, make certain you understand each part of it. Try to write a similar program using fewer BASIC statements to print out only the input lines that are a logical 1. For example, if D0 and D4 are set to a logical 1, have the computer print out D0, D4.

After you are able to do this, have the computer print only the input lines that are set to a logical 0.

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 Commodore 64. If the byte is equal to a certain value, we will print a message. If the byte is not equal to that value, we will print a different message.

This example uses a principle that is very similar to monitoring an input device with the Commodore 64. If the input condition is true, the computer will automatically take some action based on the input. Of course, in a real application, the input would come from an external hardware that the computer was monitoring.

In our example, you will generate the inputs from the CMS I/O board. When the word input equals 12, the computer prints the message, "THE INPUT WORD IS EQUAL TO 12." If the

input word does not equal 12, the computer will print the message, "THE INPUT WORD IS NOT EQUAL TO 12 AT THIS TIME."

Figure 3.12 gives a flowchart for this problem, and Figure 3.13 gives the corresponding BASIC program. Try this program out with your Commodore 64. Set the switches to a number other than 12 to see that the computer will print out the correct message. Now set the input word equal to 12 by setting D3 and D2 to a logical 1, and check if the computer will print out the correct message.

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

a. Have the computer check whether the number of bits that are a logical 1 is even in the input word. If it is, the computer will print, "THERE ARE AN EVEN NUMBER OF ONES." If the number in the input word is odd, the computer will print, "THE NUMBER OF ONES IS ODD."

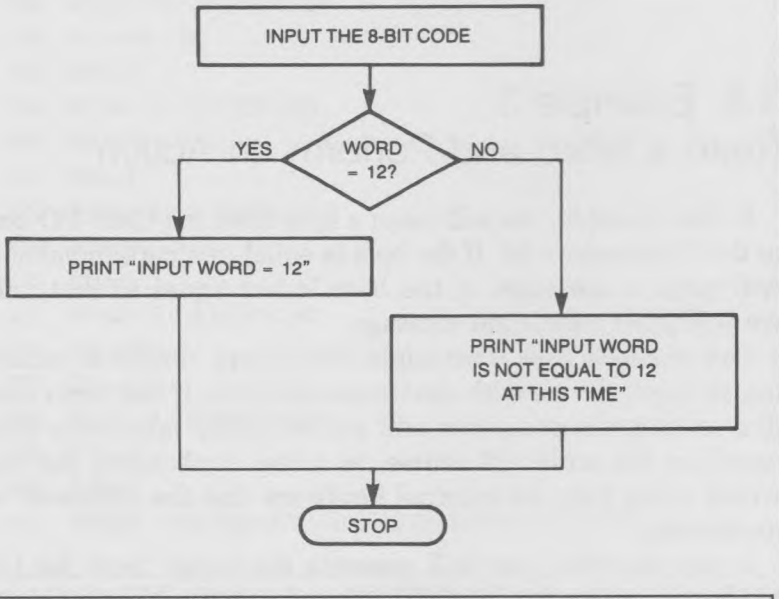


Figure 3.12: *This is a task flowchart to solve the problem given in Section 3.8.*

b. Input a number, have the computer subtract 25 from it, and then print out the results. The output should be both your original number and the new number. If the new number is less than 0, print the message, "THE RESULT WAS LESS THAN 0."

3.9 Example 4: A Combination Lock

In this example, we will make the Commodore 64 into a combination lock. The lock will have three unique numbers that must be entered in sequence. If any of the numbers or the sequence is not correct, the lock will not open.

Here is how the lock will work. Our program will ask you to enter the first number. You will set the switches that correspond to that weight on the CMS I/O board, and then enter "L" on the computer. This signals the computer that you are ready for the computer to read the number.

If the computer detects that a number is wrong, it will print "I WILL NEVER OPEN FOR YOU." Entering "R" will allow you to reset the lock and start over. If the number you entered is correct, the computer will print the message, "SO FAR SO GOOD." After you enter the last number correctly, the computer will print, "YOU OPENED THE LOCK."

Figure 3.14 presents a flowchart of this procedure. Figure 3.15 shows one BASIC program based on this flowchart. The

```
10 PRINT "INPUT THE I/O ADDRESS FOR THE CMS BOARD "
20 INPUT S3
30 A1 = PEEK(S3)
40 IF A1 = 12 THEN 70
50 PRINT "THE INPUT WORD IS NOT EQUAL TO 12 AT THIS TIME"
60 STOP
70 PRINT "THE INPUT WORD IS EQUAL TO 12"
80 END
```

Figure 3.13: *This BASIC program is based on the flowchart of Figure 3.12.*

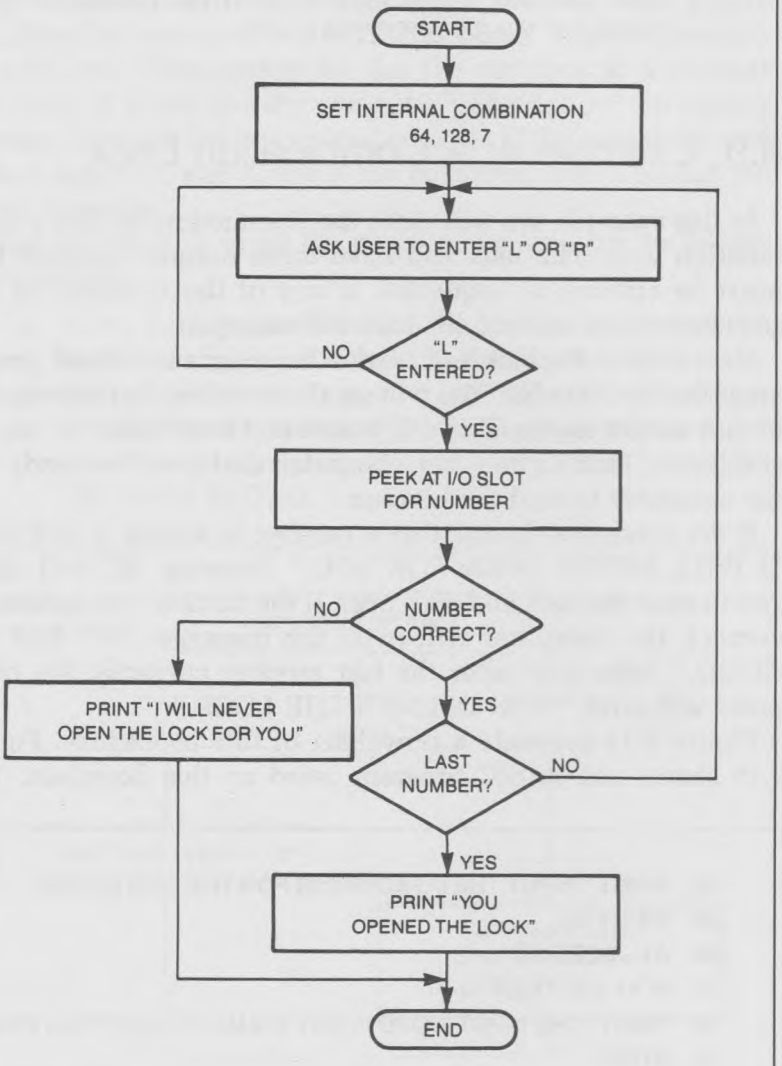


Figure 3.14: *This task flowchart shows the general steps for controlling an external device that acts as a combination lock.*

three numbers we used are 64, 128, and 7. After you run this program on your Commodore 64 successfully, try putting in the following variations:

a. Change the combination of the lock to 2, 4, 8.

b. Change the program so that it uses fewer BASIC statements than in the example.

```
 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(56832)
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
```

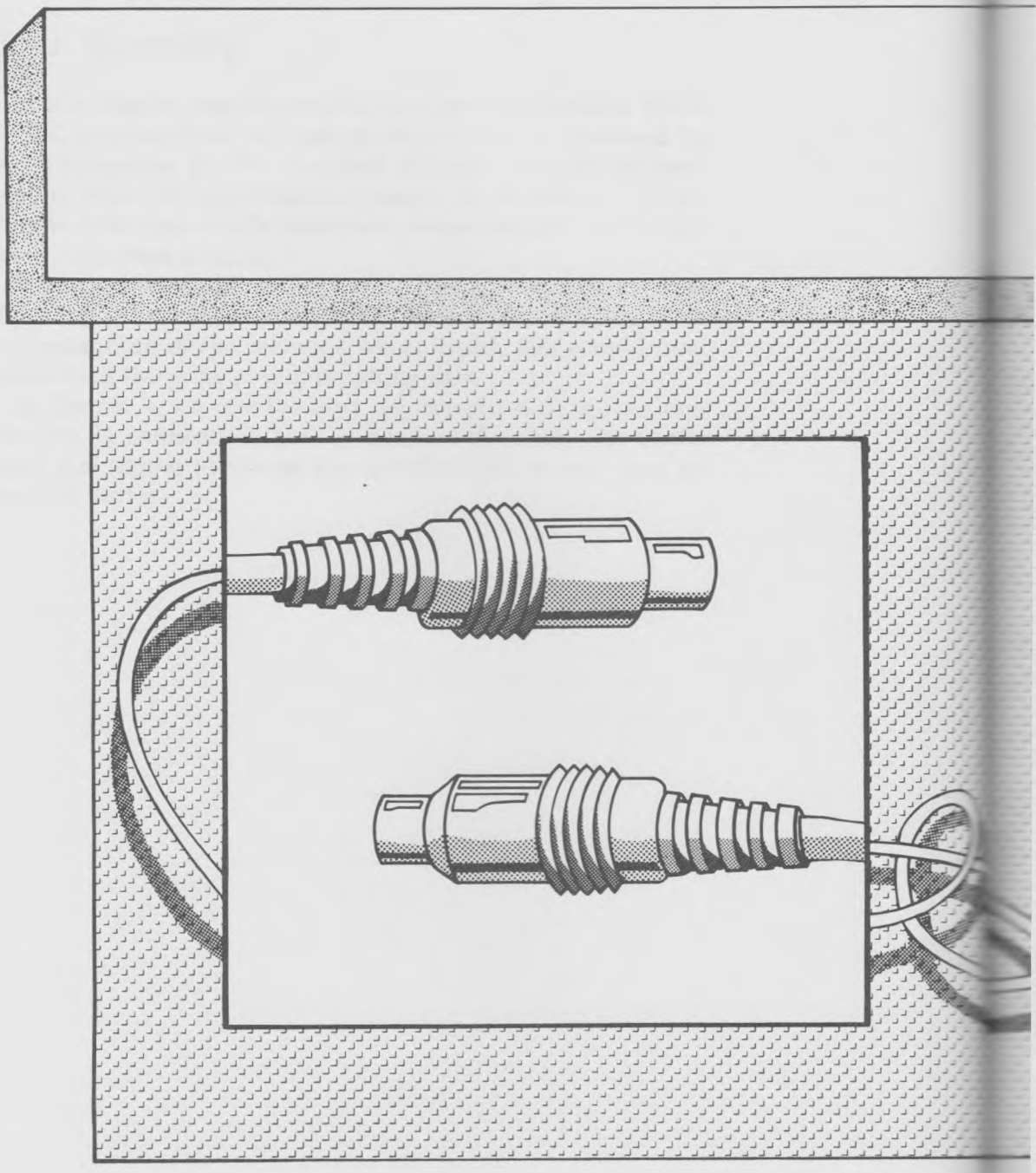
Figure 3.15: *This is a BASIC program to realize the flowchart of Figure 3.14.*

3.10 Summary

In this chapter, we covered how to input information into a BASIC program from an external device that is monitored by the Commodore 64. We discussed different methods of interpreting what the input number meant. We described a procedure to determine which input lines were a logical 1 and which input lines were a logical 0.

The examples at the end of the chapter gave you practice on inputting information to a BASIC program, and tested your understanding of the methods we outlined. You should now understand the principles of inputting data.

In Chapter 4, we take the next step toward making the Commodore 64 connection: how to connect the computer to the input and output hardware that transfers data to and from an external device.



INPUT AND OUTPUT HARDWARE FOR THE COMMODORE 64

4

In this chapter, we will discuss the hardware—internal and external—necessary to input and output data to and from the Commodore 64. Designed for the beginner in computer interfacing and computer control, these discussions focus on the architecture of the I/O expansion slot.

The designs shown in this chapter are not the most elegant or efficient. Instead, they are designed to help a person who has little or no experience with digital hardware to understand the major concepts of input and output of electrical information between the Commodore 64 and an external device. We will

frequently use the information presented here in later chapters. You may feel apprehensive if this is your first exposure to any digital hardware, and you may not want to learn about the hardware in detail. But if you can understand the essential concepts in this chapter, it will be easier to understand how to interface other peripheral hardware to the Commodore 64. Throughout this chapter, we will point out relationships between the hardware described here and the software we discussed in Chapters 2 and 3.

4.1 Beginning Output Electronics for the Commodore 64

The four digital-electronic sections we will cover when discussing the output hardware for the Commodore 64 are:

1. The enable circuit for the external device
2. The READ/$\overline{\text{WRITE}}$ signal
3. The write strobe for the circuit
4. The output storage latches

These operate together to perform the overall output function. If any of them fails to operate correctly, the output operation will not work. Let's discuss what function each electronic section performs during an output operation, and how we can implement common digital electronic circuits for each.

4.2 The Enable Circuit

Let's begin with the function of the enable circuit during an output operation for a Commodore 64. When active, an enable circuit indicates that the computer will be electrically addressing the output circuit. Because it is capable of communicating with over 64,000 different circuits, the computer must have a means of electrically informing any particular output circuit that the computer selects to communicate with.

This communication passes through the internal computer lines called *address lines.* These lines are output on the pins of the I/O expansion slot. Figure 4.1 shows a pinout of the Commodore 64 I/O expansion slot with the pin number and signal name of each pin.

We will discuss the enable signals $\overline{I/O1}$ and $\overline{I/O2}$, which are output on pins 7 and 10 of the I/O expansion slot shown in Figure 4.1. The bar over the signal indicates that when the signal is

PIN #	TYPE
12	$\overline{BA}$
13	$\overline{DMA}$
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND

PIN #	TYPE
1	GND
2	+5V
3	+5V
4	$\overline{IRQ}$
5	R/$\overline{W}$
6	Dot Clock
7	I/O1
8	$\overline{GAME}$
9	$\overline{EXROM}$
10	I/O2
11	$\overline{ROML}$

PIN #	TYPE
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

PIN #	TYPE
A	GND
B	$\overline{ROMH}$
C	$\overline{RESET}$
D	NMI
E	S 02
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10

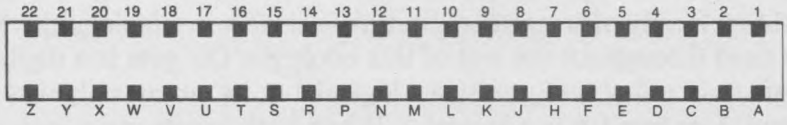


Figure 4.1: *This is the pinout of the Commodore 64, 44-pin, I/O expansion slot.*

active, it is a logical 0. At all other times, this signal is a logical 1.

Whenever the computer selects the $\overline{\text{I/O1}}$ or the $\overline{\text{I/O2}}$ address to communicate with, the $\overline{\text{I/O1}}$ or $\overline{\text{I/O2}}$ line will be set to a logical 0. Either of these two select lines can be activated by using the PEEK and POKE statements. The logical state of the I/O output signals depends on the address specified in the PEEK or POKE statement. For example, if we want to set the $\overline{\text{I/O1}}$ line to a logical 0, we PEEK or POKE an address of 56832 through 56832 + 255. To activate the $\overline{\text{I/O2}}$ line, we specify an address of 57088 through 57088 + 255. We will use the $\overline{\text{I/O1}}$ line in our example, but you can relate the information to the $\overline{\text{I/O2}}$ line.

4.3 The READ/$\overline{\text{WRITE}}$ (R/$\overline{\text{W}}$) Line

When performing computer output, we must make use of the READ/$\overline{\text{WRITE}}$ or R/$\overline{\text{W}}$ line. Its signal is output on pin 5 of the I/O expansion slot. The R/$\overline{\text{W}}$ signal is a logical 1 whenever the Commodore 64 reads data from the I/O expansion slot, and logical 0 whenever it writes data to the slot. During a PEEK statement, therefore, the R/$\overline{\text{W}}$ line is a logical 1 because the computer is reading data. During a POKE statement, the R/$\overline{\text{W}}$ line is a logical 0 because the computer is writing data. The output circuit must not only electrically examine the logical state of the R/$\overline{\text{W}}$ line to determine whether our circuit will be read or written to, but the circuit must also examine the logical state of the $\overline{\text{I/O1}}$ line. The R/$\overline{\text{W}}$ line must be "qualified" by the $\overline{\text{I/O1}}$ 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 the necessary hardware for the qualification of the R/$\overline{\text{W}}$ line to be used by an output circuit. We see that the $\overline{\text{I/O1}}$ 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 the rest of this book. An OR gate is 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 pins 1 and 2 in Figure 4.3. The 74LS32 integrated circuit contains four individual OR gates. R/$\overline{\text{W}}$ signal is input to pin 2 of the same OR

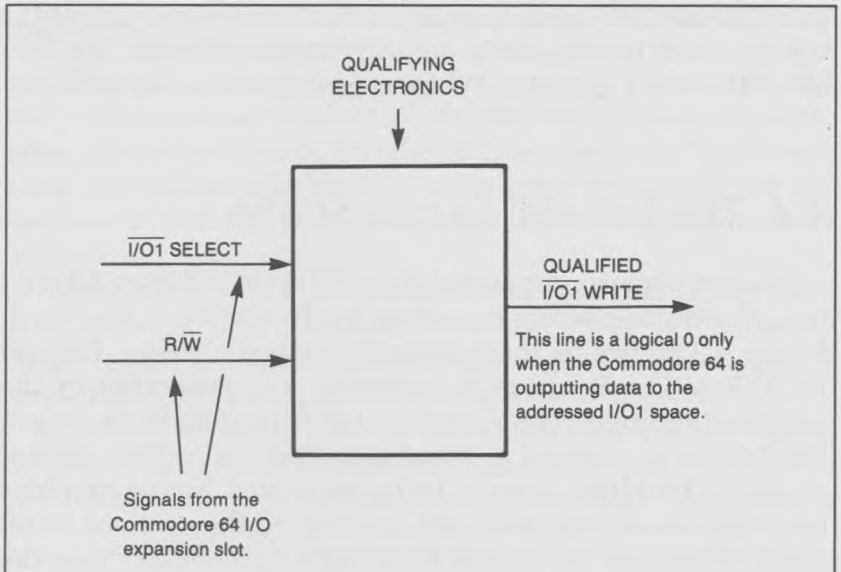


Figure 4.2: *This block diagram shows how the R/W line must be qualified by the I/O1 select line.*

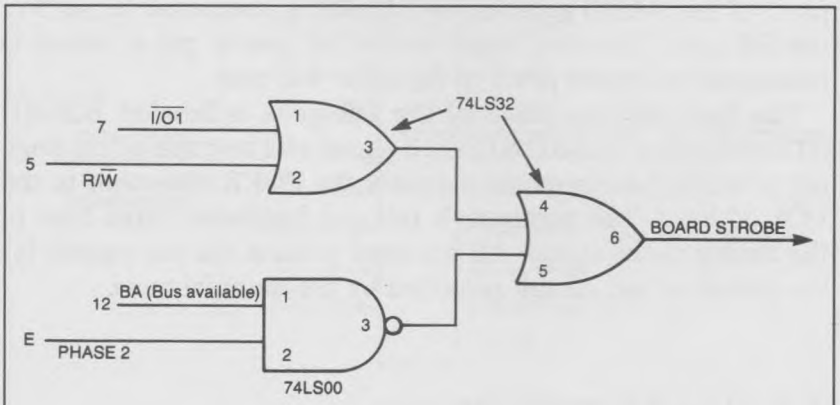


Figure 4.3: *This schematic diagram shows how to realize the circuit that generates the board strobe signal.*

gate that the $\overline{I/O1}$ signal is input to. Pin 3, the output line of the OR gate, will be a logical 0 only when both R/$\overline{W}$ and the $\overline{I/O1}$ select line are a logical 0. We have now qualified the R/$\overline{W}$ line.

4.4 The External Output Strobe

We must combine the output of the OR gate in Figure 4.3 with two other signals before the output hardware circuit can use it. Figure 4.3 shows us these other two signals labeled *BA* and *PHASE 2*. The BA (Bus Available) line is generated by the graphics microprocessor used in the Commodore 64. When the BA line is a logical 0, it indicates that the address present on the I/O address lines is being generated by the graphics microprocessor and must not be used by the external hardware. When the BA line is a logical 1, it indicates that the address is a "normal address" and can be used by the external hardware. Therefore, we must only use the I/O address lines when the BA line is a logical 1.

The other signal we must use is the PHASE 2 clock line. The PHASE 2 clock times any data transfer made with the 6510 microcprocessor that the Commodore 64 uses. In Figure 4.3, pin 3 of the NAND gate output (74LS00) is connected to pin 5 of the OR gate. The other input to the OR gate is pin 4, which is connected to output pin 3 of the other OR gate.

The final output, pin 6 of the OR gate, is labeled $\overline{BOARD\ STROBE}$. This $\overline{BOARD\ STROBE}$ signal will become active (logical 0) when the computer executes the POKE statement to the $\overline{I/O1}$ address. The computer's internal hardware takes care of the timing of the signal. All we need to do is use the signals for the design of our circuit provided by the manufacturer.

4.5 The Output Latches

This final electronic section comprises the output latches. Output latches temporarily store the data sent by the computer to an external circuit.

When we send data to the input pins of the output latches, the BOARD STROBE signal ends a write input pulse so that the data will be written into the latches. (See Figure 4.4.) The data will be stored there until we tell the computer to write other data to the circuit. In other words, after our POKE statement stores data into the latches, it stays there until we use another POKE instruction to send new data to the same address.

We can use the data at the output latches to control any hardware we want. This means we now have a way of forcing any single logical line to switch between a logical 0 and a logical 1 level. This is an important interfacing step to understand.

Let's look at the hardware of the output latch circuits. Figure 4.5 shows the latch devices we will use and the pin numbers we will connect incoming and outgoing data lines to. These latches

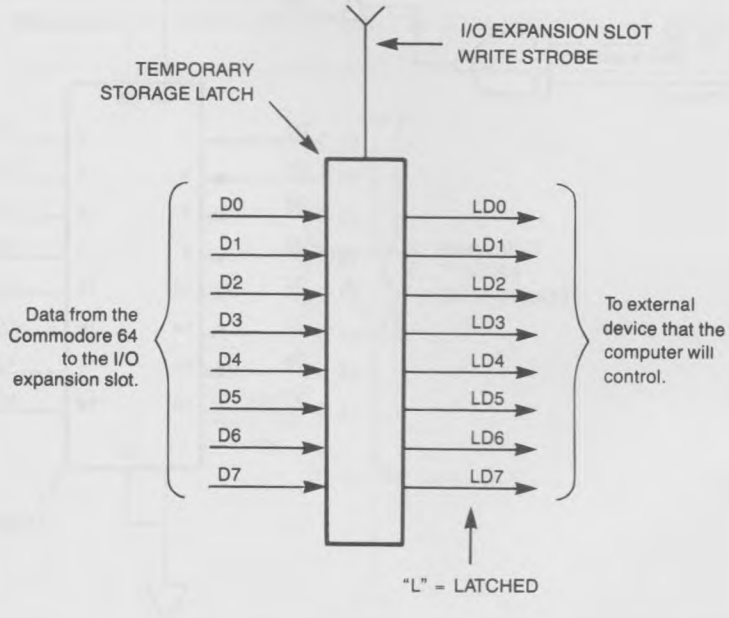


Figure 4.4: *This block diagram shows how the data output lines from the Commodore 64 will connect to the temporary storage latches located in the output circuit of the I/O expansion slot.*

are 8-bit 74LS374s. The Commodore 64 will transfer eight bits of data during every output operation using data lines D0–D7.

Pin 11 of the 74LS374 latch is the *strobe input.* When active, this line goes from a logical 0 to a logical 1. When the data is sent to the input pin of the latch, it is stored internally in the latch. Therefore, the strobe line is a logical 0 when activated. The line returns to a logical 1 when the data is latched.

We have now followed the data—along with the R/$\overline{W}$, $\overline{I/O1}$, PHASE 2, and BA control lines—from the pins of the I/O expansion slot to the latches of an I/O circuit. Now that the data is latched, we are ready to use this data to control an output device.

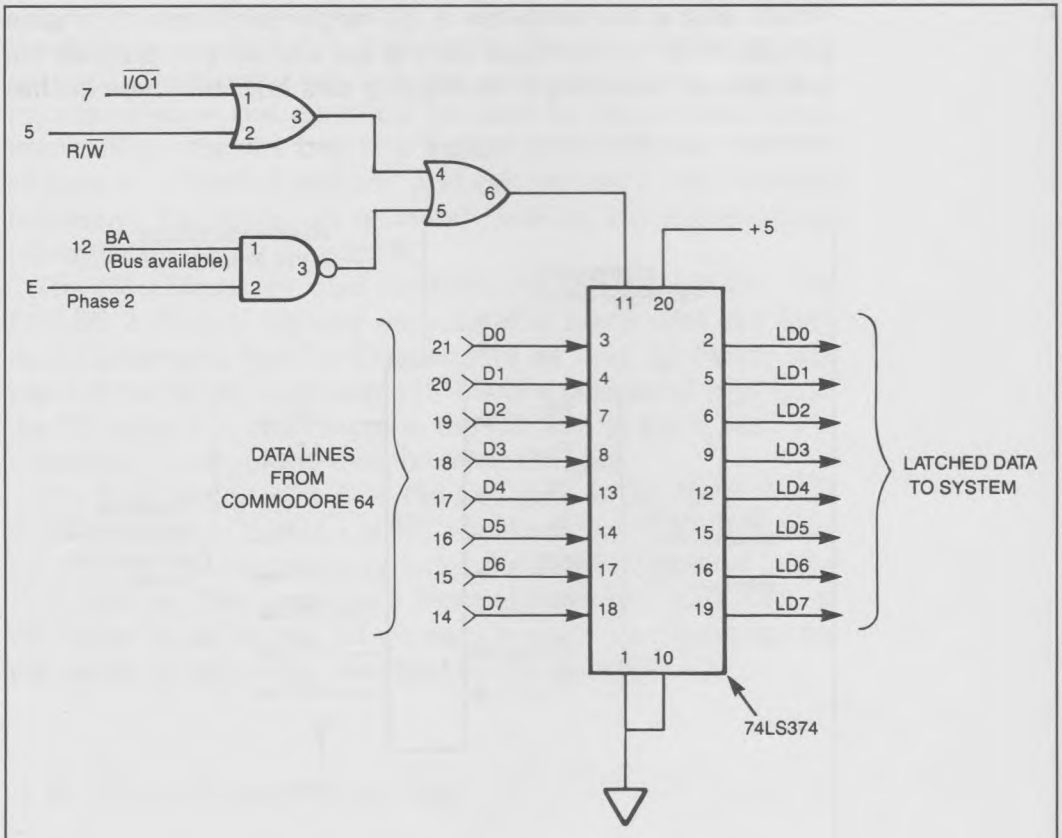


Figure 4.5: *This diagram shows the connection of the 74LS374 latch to the I/O expansion slot.*

4.6 The Light-Emitting Diodes

In this first example, we will learn to control a series of eight LEDs, one LED for each data-output line. Using LEDs will give us experience in turning on and off selected bits of an output device. We covered examples of this in Chapters 2 and 3 using the CMS I/O system. Let's look at what the hardware does when we control an external device.

Figure 4.6 is a schematic diagram of how we make an LED light. Although the LED has approximately the same electrical symbol as a standard diode, the difference is that the LED symbol has arrows pointing from it, which indicate that the diode is capable of emitting light.

The LED performs approximately the same electrical function as a standard diode. An important major difference is that when the LED is forward biased, it will emit light. The materials used to make the LED determine the color of the light. Typical LED colors are red, green, and yellow. In Figure 4.6, the anode—the positive (+) side—of the LED is connected to +5 volts. The

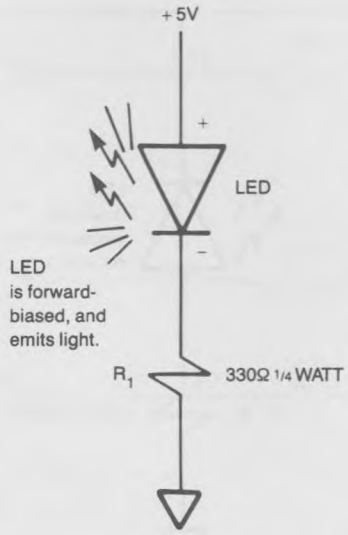


Figure 4.6: *This diagram shows a typical LED connected in the forward-biased mode.*

cathode—the negative (−) side—must not be directly connected to ground. If it is, too much current would pass through the LED and would burn it up. Therefore, the cathode must be connected to one end of a resistor, R1, so that the current would be limited to a safe value of about 10 milliamperes, or 0.01 amperes. To turn on the LED, the other end must be connected to ground potential, or approximately 0.0 volts.

Let's try to construct the circuit shown in Figure 4.6 and turn the diode on and off manually by connecting and disconnecting the anode to +5 volts. Try reversing the connection by interchanging the diode leads to see what happens. Connect the cathode of the LED to +5 volts and the anode to the resistor. Then connect the other side of the resistor to ground. The LED should not light under these conditions, because it is now connected in a reverse-biased mode, as shown in Figure 4.7. The reversed biased mode for an LED is when the anode is connected to a more negative voltage than the cathode. In this condition, no current will flow to light the diode.

If we apply what we've learned so far to eight LEDS, we have the circuit shown in Figure 4.8. Each of the resistors R1–R8 is

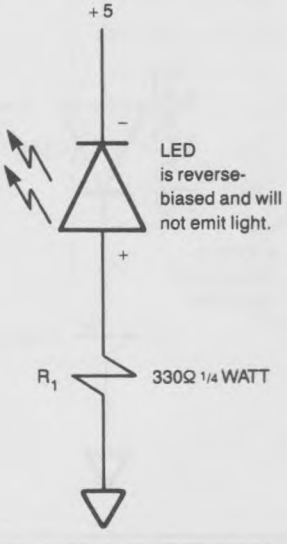


Figure 4.7: *This schematic diagram shows an LED connected in the reverse-biased mode.*

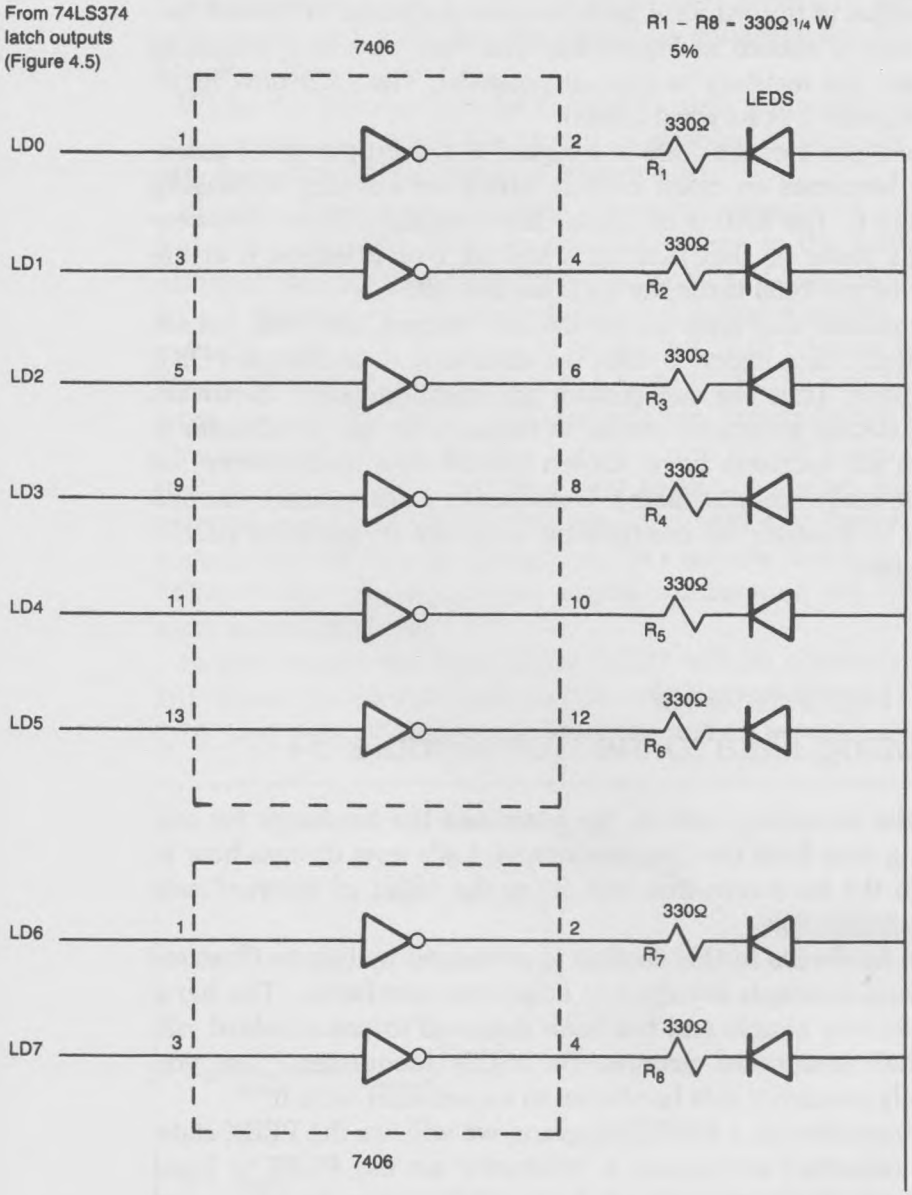


Figure 4.8: *This schematic diagram shows the driving circuits for eight LEDs.*

connected to the LEDs at one end, and to the outputs of a 7406 integrated circuit at the other. The inputs to the 7406 come from the output of the 74LS374 latch that we discussed in the last section and as shown in Figure 4.5. The 7406 acts as a switch to connect the resistors to ground potential. The LED now lights up, because it is forward biased.

When pin 1 of the 7406 is a logical 0, the output pin 2 essentially becomes an open circuit, since no current is passing through it. The LED is off under this condition. These two conditions show us that placing a logical 1 or a logical 0 at the input of the 7406 turns the LED on and off.

Remember that once we set latched outputs, they will remain a logical 1 or a logical 0 until we send new data using a POKE statement. Thus we can control the function of the hardware using BASIC programs similar to the ones we saw in Chapter 2. These six sections have shown us the essential concept of output-hardware interfacing: transferring information from one place to another by controlling each bit to perform particular tasks.

4.7 Hardware for Inputting Data to the Commodore 64

In the preceding section, we examined the hardware for outputting data from the Commodore 64. Let's now discuss how to design the hardware that will allow the input of external data to the computer.

The hardware in this section is presented mainly to illustrate the basic concepts involved in computer interfacing. The hardware is very simple and has been designed to use standard, off-the-shelf integrated circuits. We highly recommend that you actually construct this hardware to experiment with it.

To input data to a BASIC program, we will use the PEEK statement described in Chapter 3. Whenever we use PEEK to input data from the correct address for the I/O circuit, the $\overline{\text{I/O1}}$ select line will be an active logical 0. We discussed the process in Section 4.2 on the enable circuit.

Because the PEEK statement will be performing a read function, the R/$\overline{W}$ line on the I/O expansion slot will be a logical 1. Figure 4.9 shows that the R/$\overline{W}$ and the $\overline{I/O1}$ select line are logically combined with the BA signal to generate the $\overline{BOARD\ ENABLE}$ line.

While the Commodore 64 is reading data from the selected I/O circuit, data from the circuit is enabled electrically onto data lines D0–D7, as shown in Figure 4.10.

Figure 4.11 shows a complete schematic diagram of a circuit that will input data from an external source to the computer. Although this circuit will employ the circuits of Figures 4.9 and 4.10, it requires very little hardware and operates simply.

At the center of Figure 4.11 is the 74LS240 tri-state buffer. This buffer takes input on pins 2, 4, 6, 8, 11, 13, 15, and 17 from input lines XCD0–XCD7. (XCD stands for *External Card Data.*) These input lines are either a logical 1 or a logical 0 and represent digital information being sent to the computer from an external instrument. The input pins can hold any digital information desired. For our discussion, let's assume that input lines XCD0–XCD7 represent some digital information we want to send to the computer.

In our circuit, the lines XCD0–XCD7 will be connected to a DIP (Dual In-line Package) switch, which is shown in Figure 4.12. One side of the switch is connected to ground, and the

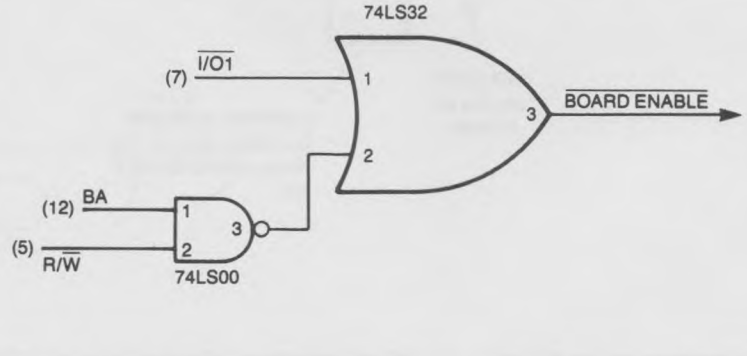


Figure 4.9: *This schematic diagram shows the hardware necessary to qualify the R/$\overline{W}$ line during an input read operation from the I/O expansion slot.*

other side to the XCD input lines of the 74LS240 buffer shown in Figure 4.11. When the switch is closed, the input to the 74LS240 is a logical 0. When the switch is open, the input is a logical 1. The input we use here is called TTL (*Transistor-Transistor Logic*) input. The Commodore 64 uses TTL, which is a family of digital electronic circuits, for some of its internal circuits. A 74LS240 will logically invert the information input before it outputs data to the data lines. This is exactly how the CMS I/O system described in Chapter 3 operated.

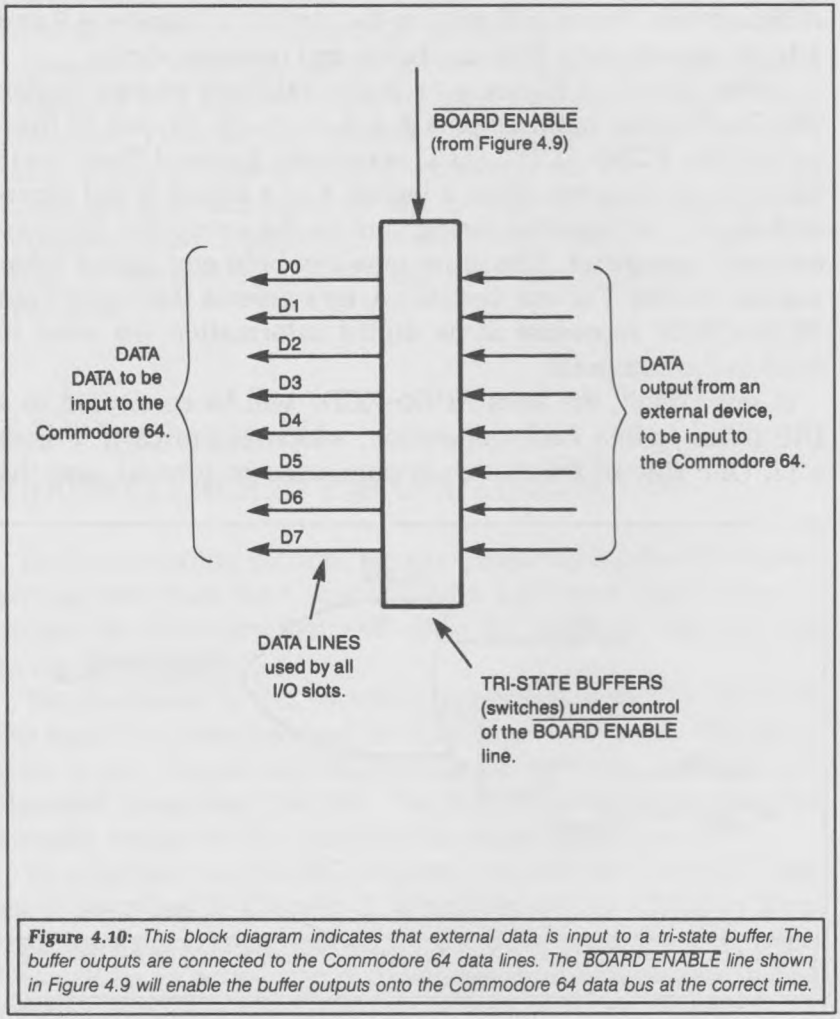


Figure 4.10: *This block diagram indicates that external data is input to a tri-state buffer. The buffer outputs are connected to the Commodore 64 data lines. The BOARD ENABLE line shown in Figure 4.9 will enable the buffer outputs onto the Commodore 64 data bus at the correct time.*

As we can see in Figure 4.11, the outputs of the 74LS240 are connected to the data lines, D0–D7. We can enable, or turn on, the outputs of the 74LS240 only when the computer electrically requests the data from the input circuit. At all other times, the data outputs from the buffers are set into a tri-state, or off, mode, which electrically removes the 74LS240 outputs from the

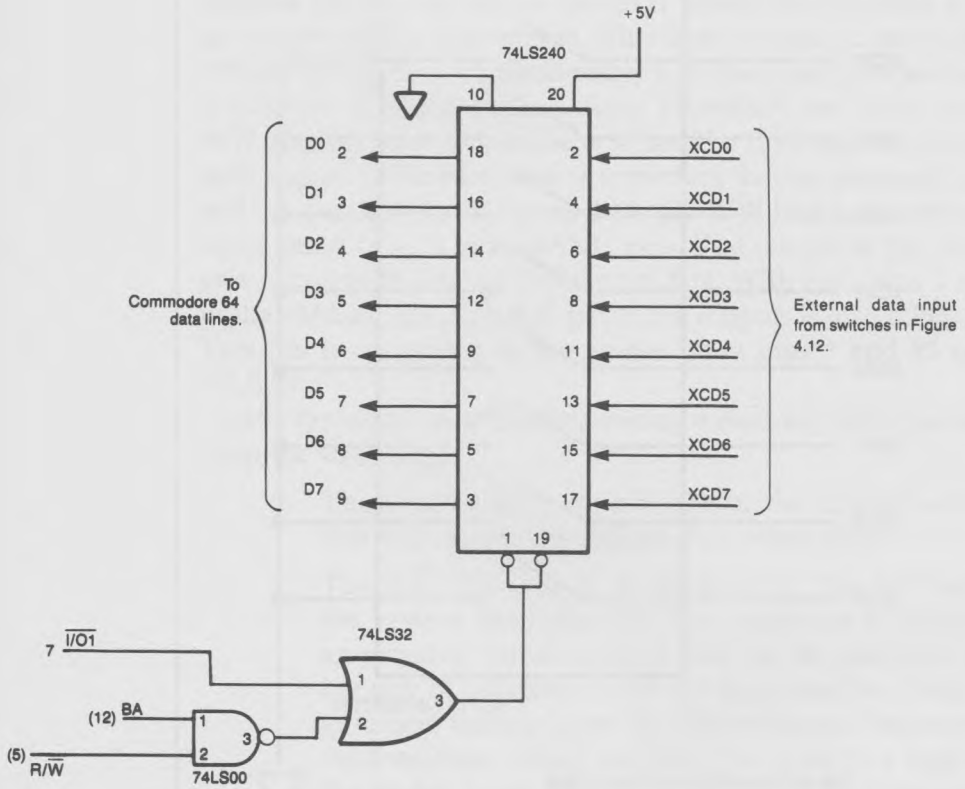


Figure 4.11: *This is a complete schematic of the hardware necessary to input an external data byte to the Commodore 64 from the I/O expansion slot.*

data bus lines. The outputs of the buffer will be placed on the data bus only when input pins 1 and 19 are a logical 0. Pins 1 and 19 are connected to the output pin 3 of the OR gate (74LS32), and thus will be controlled by the R/$\overline{W}$, BA, and $\overline{I/O1}$ select lines of the I/O expansion slot.

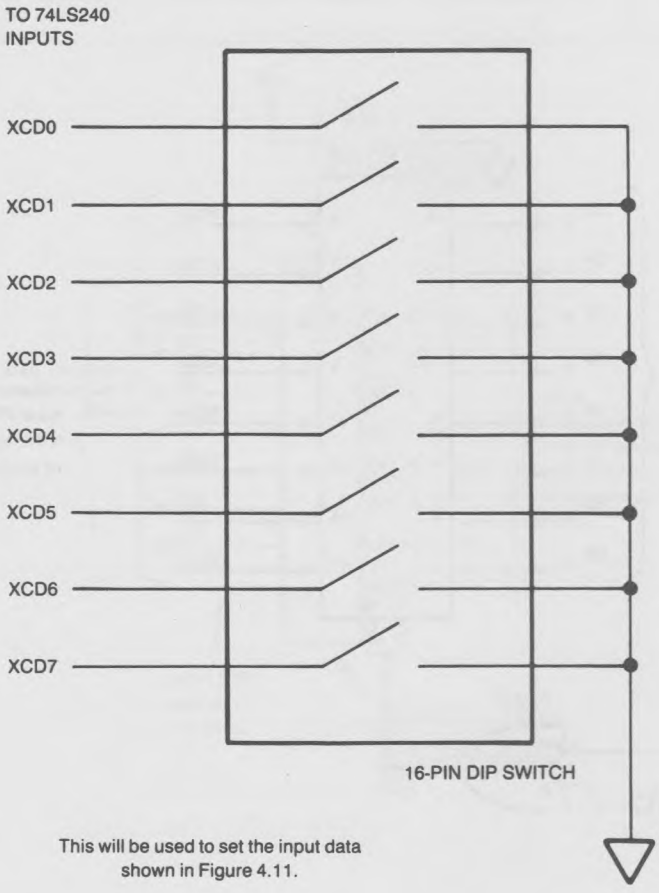


Figure 4.12: *This is a schematic diagram of eight switches used to set the logical level of the eight external input lines at the I/O expansion slot.*

4.8 Enabling the Tri-state Buffer

The previous section showed us that the 74LS240 output is placed on the data bus only when pins 1 and 19 are a logical 0. This occurs during a PEEK instruction because the computer is requesting data from the selected I/O address. In this section, we will examine how we use the computer to set the enable pins to logical 0, based on what we have learned in the previous section. This procedure will enable the tri-state buffer.

When the computer sends out an address that matches the address for the I/O circuit, the $\overline{\text{I/O1}}$ select line becomes active, as we saw in the last section. This line is input to pin 1 of the 74LS32 of Figure 4.11. (Remember that this line is active during a write or POKE operation also.) Therefore, we must use the $\text{R}/\overline{\text{W}}$ and BA lines as qualifiers. When the $\text{R}/\overline{\text{W}}$ and BA lines are both logical 1, the computer is expecting that an external device will send data to it. In Figure 4.11, the $\text{R}/\overline{\text{W}}$ line is connected to input pin 2 of a 74LS00 NAND gate. The output of the NAND gate is connected to pin 2 of the 74LS32. With both pins 1 and 2 of the 74LS32 at a logical 0, pin 3, the output, is also a logical 0. This pin is connected to the enable input pins 1 and 19 of the 74LS240.

Let's review what will occur during a read or PEEK operation from the I/O circuit.

1. The Commodore 64 will output the correct address that will enable the proper $\overline{\text{I/O1}}$ select line.
2. The $\text{R}/\overline{\text{W}}$ line will go to a logical 1. This will inform the system hardware that the computer is expecting an external device to send data on the data bus. During any I/O operation, the BA line must be a logical 1 to inform the hardware that the computer has output a valid address. When the $\overline{\text{I/O1}}$ line goes to a logical 0, the enable input pins 1 and 19 of the 74LS240 are set to a logical 0. At this time, the data at the buffer inputs are placed on the data bus.
3. Meanwhile, the Commodore 64 will automatically read the data. Chapter 3 discussed how the software will use this data.

4. The circuits in the Commodore 64 automatically set the $\overline{I/O1}$ select line to a logical 1; therefore, you do not have to worry about making this happen. The enable input pins 1 and 19 of the 74LS240 will be then set to a logical 1. The action will tri-state, or turn off, the 74LS240 buffer outputs, which will electrically remove them from the data bus lines.

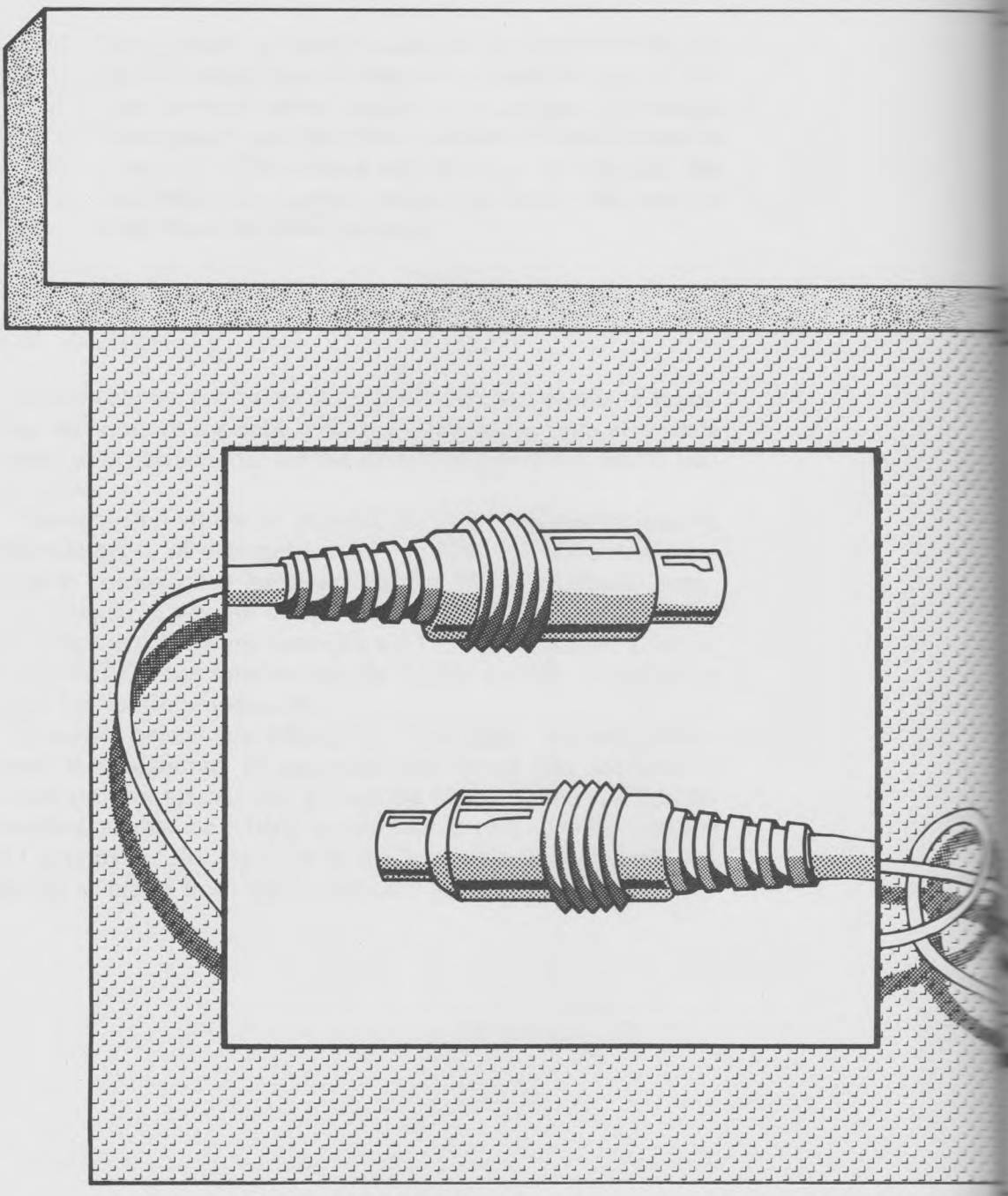
4.9 Summary

In this chapter, we have covered the essential details of inputting and outputting data with the Commodore 64, and have looked at the schematics for the necessary hardware, which you can build.

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 Chapter 5, we will design a home security system with the hardware and software concepts we have explored. In Chapter 6, we will discuss how to use the digital outputs to control a voice for the Commodore 64.

However, before you attempt to accomplish this, we recommend that beginners in computer interfacing take the time to construct the circuit we presented here. There are circuit boards available for wiring up circuits to plug directly into the I/O expansion slot, and all of the hardware components are readily available from many suppliers (see Appendix E).



AN APPLICATION OF COMPUTER INTERFACING: A HOME-SECURITY SYSTEM

5

In this chapter, we will present all of the hardware and software necessary for a home-security system. We will use BASIC programs as the controlling software. This chapter will bring together all of the important points covered in the first four chapters.

While it does work, the system's primary purpose is to show you one way the Commodore 64 can be used to monitor and secure a home. We hope that when you apply this information to your own home, you will find the process fun and easy. You can use the information in this chapter as a starting point for using the Commodore 64 to control other external systems.

5.1 A Definition of the Problem

Let's start by trying to define our problem: we must design and build a system that will monitor the status of all doors and windows in a home. There are already problems with our definition: there may be some windows that cannot be opened which we will not monitor. There are also many doors in a home that do not open to the outside. So let's refine our problem: *the system will monitor the status of any door that opens to the outside of the home and any window that can be opened.* For our system, the term "to monitor the status of" means "to indicate whether the door or window is open or closed."

Our security system will determine whether a window or door is open or closed. If a door or window is open, the system will indicate on the Commodore 64 screen which door or window it is.

Here is the complete definition of our system:

1. The system will monitor every door that opens directly to the outside.
2. The system will monitor every window that can be opened.
3. If the system detects an open window or door, it will indicate this on the Commodore 64 screen.

There are certainly more functions you could design into the system, depending on how complex you wish the system to be. Some systems have been designed that incorporate a modem and a voice synthesizer to call the police if a window or door is tampered with. Remember that our example is given only as an illustration of what can be done and will highlight the important details of interfacing and computer control.

5.2 Drawing the House with the Computer

Part of our definition requires that the Commodore 64 indicate on screen if any of the doors or windows being monitored are open. This can be done in any number of ways, but the

technique we will use starts with a screen display of an outline of the house, similar to an architect's blueprint, showing each window or door being monitored. Figure 5.1 shows exactly what we will draw on the screen.

We will give each door and window a label. When the system detects that a door or window is open, the label for that door or window will be highlighted in a special color on the screen. This will give you a quick visual check of the status of any door or window in the house.

Another feature of our program is that the user will have the option of ignoring any window or door that is open. For example, suppose you are purposely leaving a bedroom window open because it is very hot. The program will ask which windows or doors you wish to ignore, so that the screen display will show which are open but not generating an alarm. The display

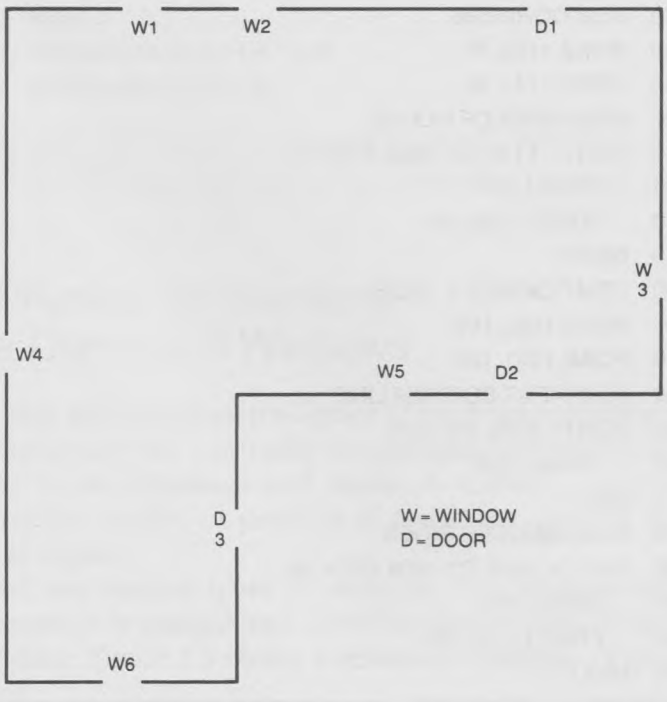


Figure 5.1: *This is the outline of the house that will be drawn by the Commodore 64.*

will highlight those differently than other open windows or doors which are causing an alarm.

Figure 5.2 shows a program written in BASIC for drawing the layout in Figure 5.1 on the screen. This program uses the special graphic characters of the Commodore 64.

```
170 REM DRAW THE SCREEN NOW
180 PRINT CHR$(147)
190 REM SET COLOR OF EACH SCREEN CELL BLACK
200 FOR I = 55296 TO 56295
210   POKE I,0
220 NEXT I
230 REM TOP LINE OF SCREEN
240 FOR I = 1106 TO 1140
250   POKE I,99
260 NEXT I
270 REM CORNERS
280 POKE 1105,79
290 POKE 1141,80
300 REM SIDES OF HOUSE
310 FOR I = 1145 TO 1584 STEP 40
320   POKE I,101
330   POKE I + 36,103
340 NEXT I
350 REM CORNER + SIDE
360 POKE 1585,101
370 POKE 1621,122
380 REM FIRST BOTTOM LINE
390 FOR I = 1600 TO 1620
400   POKE I,100
410 NEXT I
420 REM SMALLER SIDES
430 FOR I = 1625,TO 1906 STEP 40
440   POKE I,101
450   POKE I + 14,103
460 NEXT I
```

Figure 5.2: *The BASIC program draws the outline of the house shown in Figure 5.1. Special graphics characters of the Commodore 64 were used.*

```
470 REM CORNERS
480 POKE 1945,76
490 POKE 1959,122
500 REM BOTTOM LINE
510 FOR I = 1946 TO 1958
520   POKE I,100
530 NEXT I
540 REM SCREEN IS NOW DRAWN, NO LABELS IN YET
900 REM WRITE W = WINDOWS, D = DOORS
910 FOR I = 1809 TO 1816
920   READ V1
930   POKE I,V1
940 NEXT I
950 FOR I = 1849 TO 1854
960   READ V1
970   POKE I,V1
980 NEXT I
1100 DATA 23,61,23,9,14,4,15,23
1110 DATA 4,61,4,15,15,18
```

Figure 5.2 (continued)

5.3 Physical Connections to the Doors and Windows

Now that we have drawn a layout of the house on the screen, let's discuss how we can make the physical and electrical connections to the windows and doors. A common switch will transform the motion or position of a window or door into an electrical signal.

We can use various types of switches. The type we will use for our system is opened and closed by the position of the window or door. Figure 5.3 shows a schematic diagram of this type of switch.

In Figure 5.3, we see that when a window or door is closed, its switch is closed. When that window or door is open, the

switch is 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. However, we cannot give you details of how to install these switches because everyone's windows, doors, and the type of switch used may differ. We have found that many readers will take the methods we describe and improve them. Therefore, use these general guidelines to help you get started in the right direction.

The switch uses resistance as the basis for its measurement. When the switch is closed, the resistance of the switch is approximately zero ohms. When the switch is open, the resistance is infinite ohms.

We will use the characteristics of a switch to generate a digital voltage signal that we can input directly to the Commodore 64. 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 chapter.) The other side is connected to an input circuit, exactly like the circuit described in Chapter 4.

When the switch in Figure 5.4 is open, there is no electrical path to ground through the switch. This changes the input line connected to the 4.7K-ohm resistor to +5 volts. This voltage level corresponds to a logical 1 in the computer. When the window or door is closed, the switch is closed which forces the input line connected to the resistor to ground. Now, an electrical path is established through the switch to ground. The voltage of ground potential equals a logical 0 in the computer. Therefore, when the door or window is open, the digital input line to the Commodore 64 equals +5 volts, which is a logical 1. When the door or window is closed, the digital input line equals 0.0 volts, or a logical 0.

Now that we know this, we can examine the status of each door or window using the PEEK instruction and the software discussed in Chapter 3. We will give the complete program for controlling the system later in this chapter.

If you have many doors and windows to monitor, you will need quite a few electrical connections to your Commodore 64. The voltages in the circuits are very low, so you can use light-gauge wire—such as "speaker wire"—in the connections.

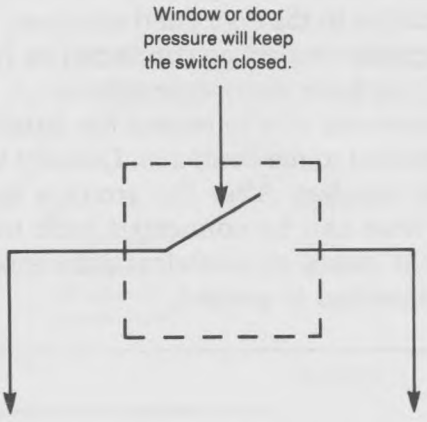


Figure 5.3: *This diagram shows one type of switch that can be used to monitor doors and windows. The pressure of the door or window will keep the switch closed.*

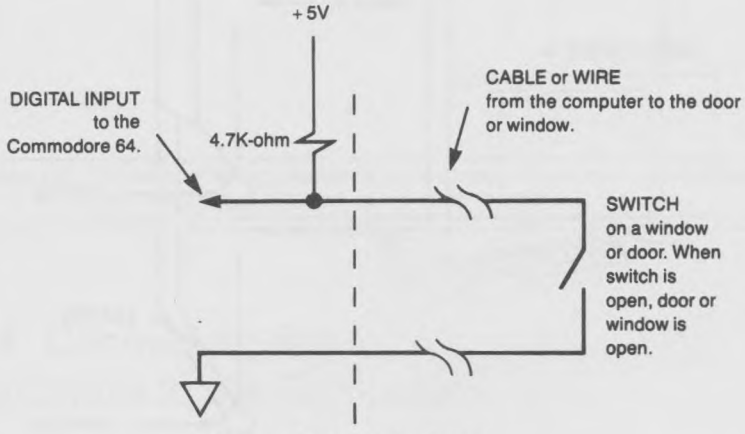


Figure 5.4: *This diagram shows how the switch will be wired. One side of the switch will be connected to the Commodore 64 ground, and the other side will be connected to a 4.7K-ohm resistor. It does not matter which side of the switch is connected to ground.*

Figure 5.5 shows a block diagram of what happens when you connect the switches to the doors and windows. In this diagram each switch requires two wires connected to it. This could be cumbersome if you have many connections.

Figure 5.6 shows one way to reduce the number of wires that need to be connected to the computer. Connect the ground lines of the switches together. After the grounds are connected, a single ground wire can be connected back to the computer. Remember that it makes no electrical difference which side of the switch is connected to ground.

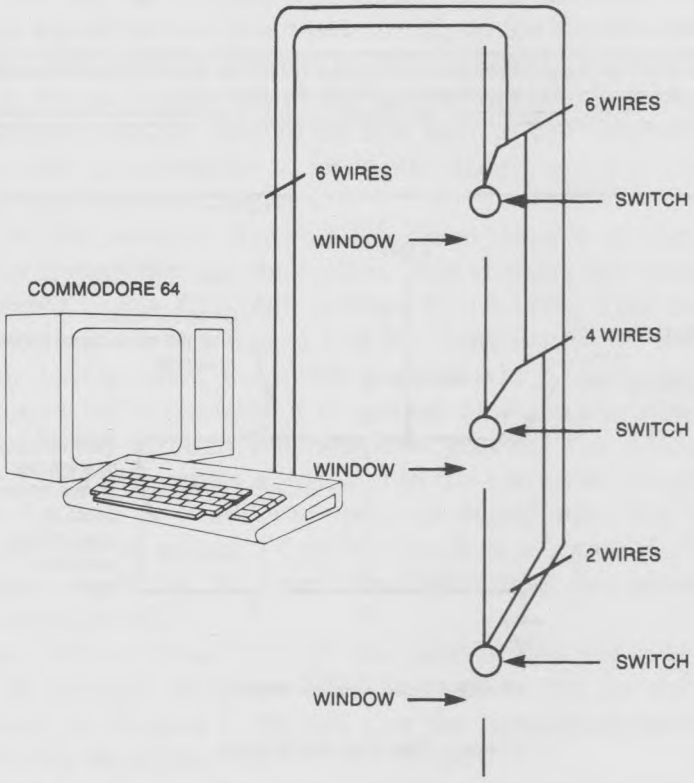


Figure 5.5: *This pictorial diagram indicates that each switch to be monitored will require two wires sent back to the Commodore 64. There are three switches shown here.*

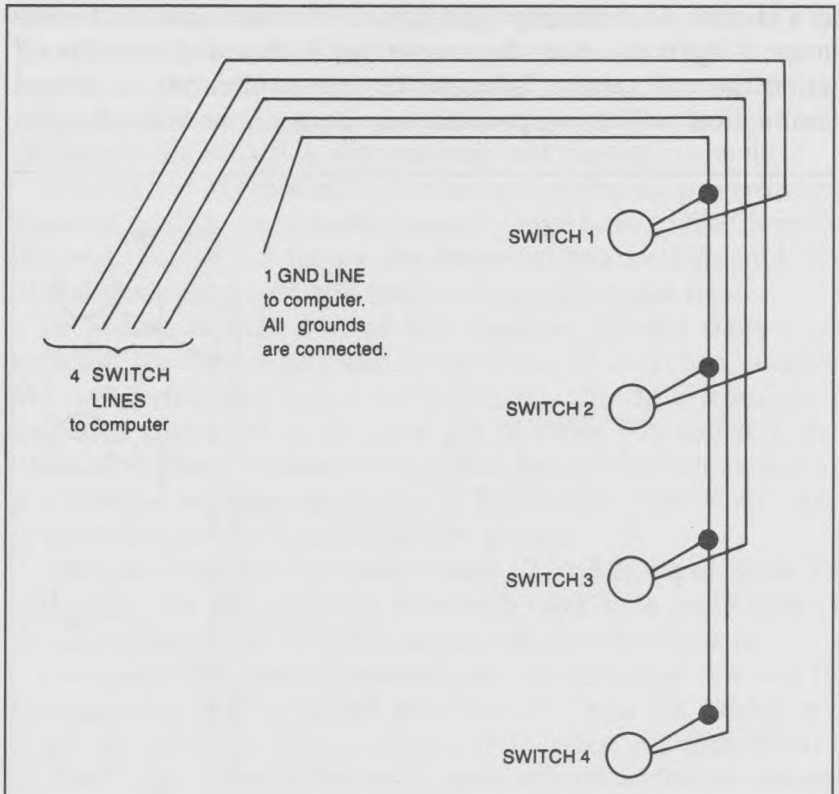


Figure 5.6: *This method of reducing the number of wires must be connected to the Commodore 64. In this scheme, we use only a single ground wire for all the switches.*

5.4 Connecting the Hardware to the Computer

We now have a bundle of wires from the door and window switches ready to be connected to the computer. However, before we discuss how to make those connections, we should verify that all of the switches will open and close, and that the wiring from the switches to the computer is complete.

The verification procedure involves using either an *ohmmeter* or a *continuity light*. An ohmmeter measures resistance

in a circuit. A continuity light also checks the amount of resistance: it lights up when the connection is closed or remains off when the connection is open. In this application, a closed connection will be approximately zero ohms and an open

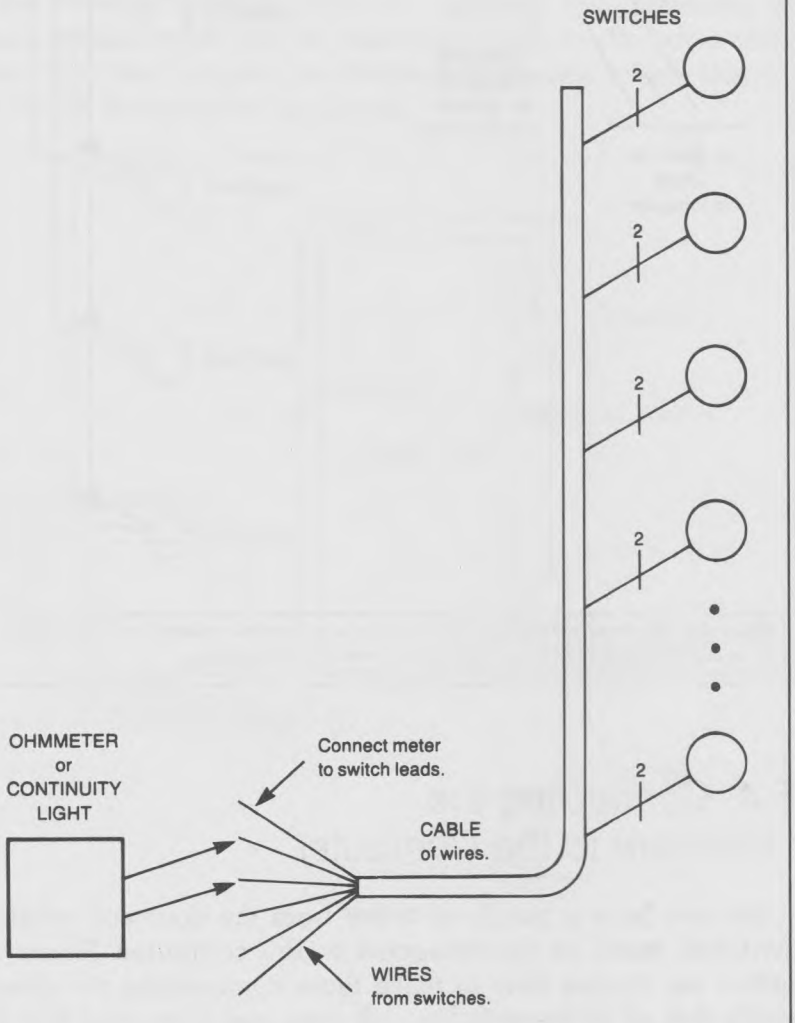


Figure 5.7: *The physical and electrical connections to the switches can be verified by using an ohmmeter or a continuity light.*

connection will be (theoretically) infinite ohms. As shown in Figure 5.7, we place the test leads across the two wires that are connected to any installed switches. If the door or window is opened and then closed, your ohmmeter or continuity light will indicate if the switch is also opening and closing correctly.

If we can verify that every switch is functioning properly, we are now ready to see how to connect signal lines to the Commodore 64. Figure 5.8 shows the electrical connections and the digital electronics you will need to input the signal lines.

Let's discuss each part of this diagram. On the right-hand side are the electrical input wires from all switches, labeled W1–W6 (windows) and D1–D3 (doors). One line from each switch is connected to an input pin of either IC3 and IC4, the 74LS244 buffers. Remember that this line is also connected to a 4.7K-ohm resistor, as shown in Figure 5.4. The other wire is connected to the Commodore 64 ground.

Let's see how the computer reads W and D input lines. In Chapter 4, we discussed the necessary circuits to input data to the Commodore 64. We will use two similar circuits here.

One main difference between our circuits and the one in Chapter 4 is that we added two lines, A1 and A2, which are input to the IC1s, the 74LS00 NAND gates. A1 and A2 are address lines. They allow us to have several addresses associated with each I/O circuit. In Chapter 4, these lines were not used, because we were only using one address per I/O circuit. Now that we have two buffers, we need two addresses.

When we use the PEEK statement, the new address for enabling IC3 of Figure 5.8 will be equal to:

```
PEEK address = I/O address + 2
```

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

```
PEEK address = I/O address + 4
```

The 4 will set A2 to a logical 1.

For example, if we want to read the logical level of the input lines connected to IC3, we could do it as follows. (We will assume that the I/O address of the circuit is 56832 decimal.) In

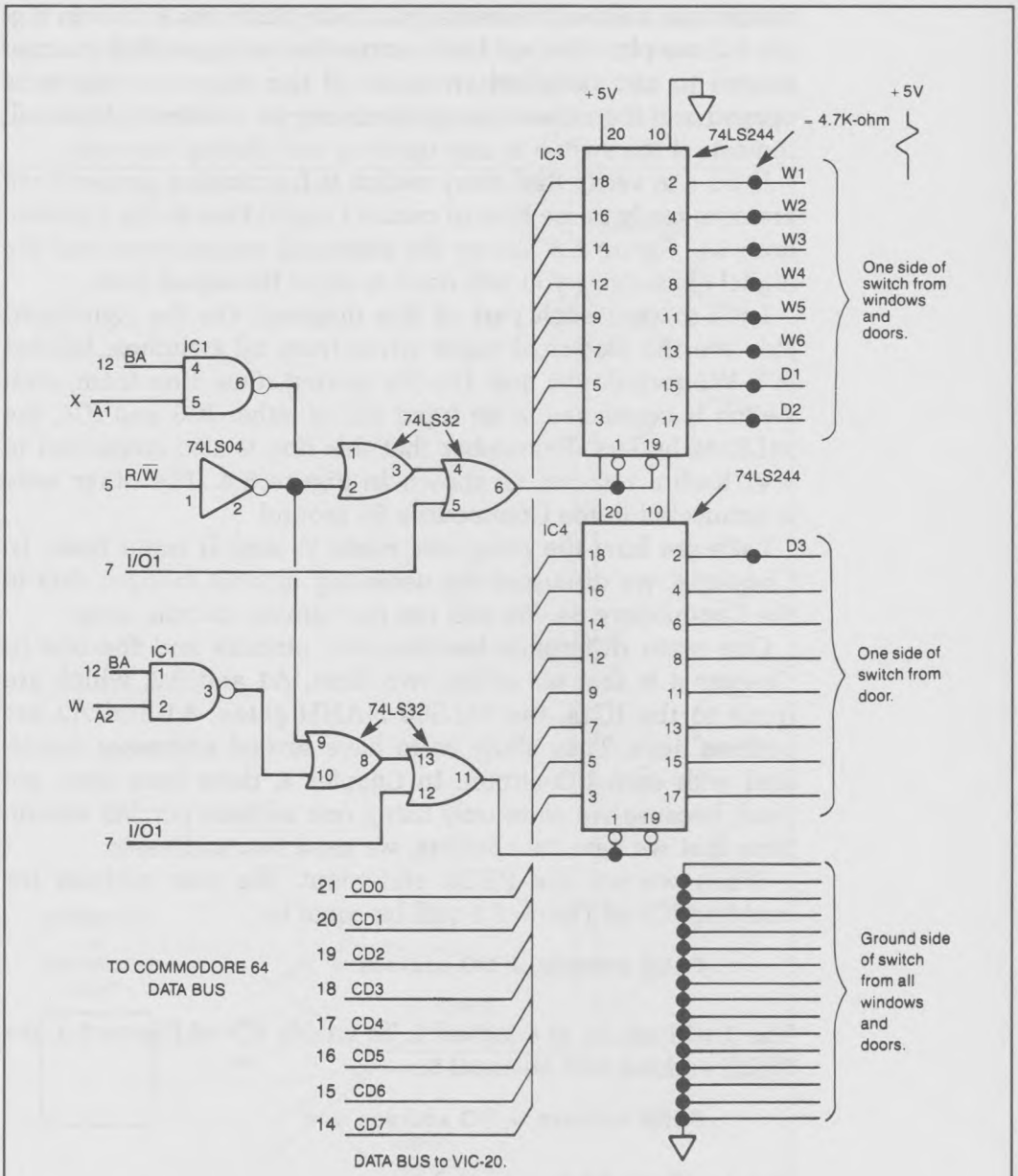


Figure 5.8: *This is the complete schematic for the I/O circuit used to connect the Commodore 64 to the external switches on the doors and windows.*

BASIC, the PEEK statement would be:

```
B1 = PEEK (56832 + 2)
```

or

```
B1 = PEEK (56834)
```

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

```
B2 = PEEK (56832 + 4)
```

or

```
B2 = PEEK (56836)
```

After we use these statements, the variables B1 and B2 are equal to the input weights associated with the open and closed positions of each door and window. We must now write the BASIC program that will examine the position of each door and window individually.

5.5 Software for Interpretation of the PEEK Input Lines

At this point in designing our security system, we can input the logical condition of each switch into a BASIC variable. This is the way we determine whether a window or door is open or closed. Our next step is to tell the user what the status of each door and window is.

We can use the data we have been given so far toward any imaginable application. For example, to tell the user that a window has been opened you could use special graphics programs to draw a window opening with a warning flashing on the screen. The possibilities are endless.

Figure 5.9 is a flowchart of what our program must do next. This flowchart shows large tasks, which correspond to the routines we will write. In the first block, we draw the outline of the house on the screen, as shown in Figure 5.1. In the second block, we use the PEEK statement to read the digital signals of the windows and doors into the Commodore 64.

In the following block (3), we will fill an array called "DATA ARRAY" with either a 1 or 0, based on the logical value of the window or door we are monitoring. Therefore the 16 elements, D(1) to D(16), of our data array will correspond to the logical input values of the signal lines 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 W2, D(3) equals the logical value of W3, and so on.

In the last block, our software will write the status of each window and door on the screen.

In this section, we will write the software for the tasks in the second and third blocks of the flowchart: reading the data from the windows and doors, and filling the data array. We have already discussed how to draw the house on the screen in Section 5.2.

Section 4.7 showed us how to use the PEEK statement to input the data in the Commodore 64 from an external I/O

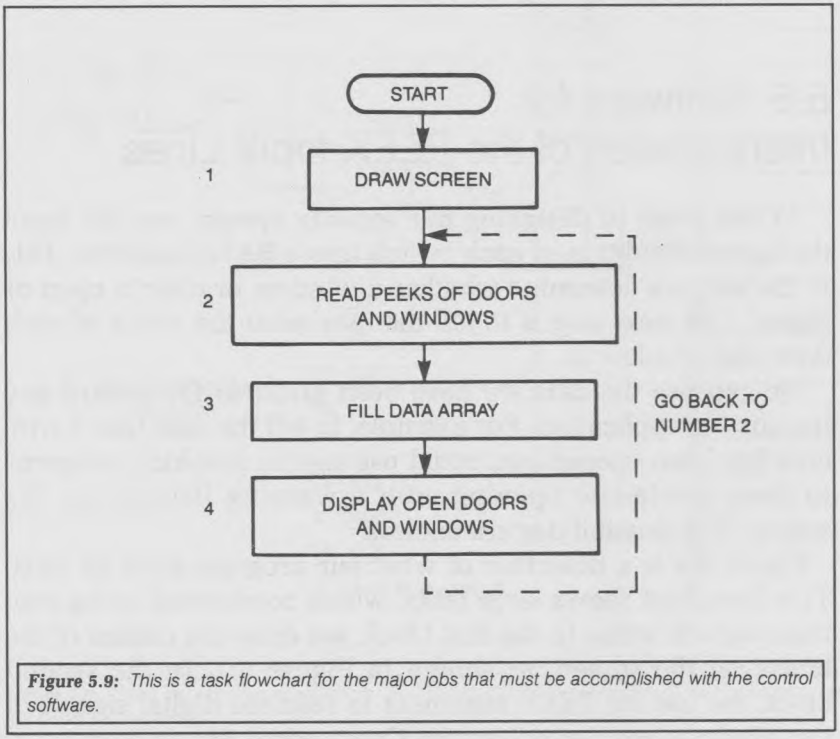


Figure 5.9: *This is a task flowchart for the major jobs that must be accomplished with the control software.*

circuit. We will need two PEEK statements to read all 16 data lines that were shown in Figure 5.8.

The data read from the input connector will be stored into an array A. The name "A" allows 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. Remember that these integrated circuits are 74LS244s, octal buffers with eight output lines each, and that the summed weight can be any number from 0 to 255, inclusive.

Recall also that the PEEK address is dependent on the address of the I/O circuit. We will use the general name "I/O add" to mean the number that corresponds to the I/O address of a particular circuit. For our program to actually work, of course, you must use a real address. The two PEEK statements used are:

```
A(1) = PEEK (I/O add + 2)
A(2) = PEEK (I/O add + 4)
```

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

We have now stored into our BASIC program. The next step is to fill data array D with the proper 1s and 0s. Figure 5.10 shows the list of the array-D elements and the corresponding window or door that each represents.

Figure 5.11 shows the section of software for filling the data array with the correct values. After we show you the entire routine, we will then break each statement down and provide further explanation when necessary.

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

The statement:

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

marks the beginning of the FOR/NEXT loop. Variable F will be set prior to calling this subroutine, and will determine the starting element of data array D. Each time our program calls the

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)	=	D1
D(8)	=	D2
D(9)	=	D3
D(10)	=	NOT USED
D(11)	=	NOT USED
D(12)	=	NOT USED
D(13)	=	NOT USED
D(14)	=	NOT USED
D(15)	=	NOT USED
D(16)	=	NOT USED

Figure 5.10: *This is a list of the "D" (data) array elements and the corresponding "W" or "D" input line each one represents with software.*

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

Figure 5.11: *This subroutine will fill the data array.*

subroutine, it fills eight elements of data array D. The first GOSUB will have F equal to 1; the next will have F equal to 9.

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

The value of D(I) was originally zero. It will be changed by the weight of any array-D element tested if the summation used the weight. We described the same type of operation in Section 3.5, and gave a similar program in Figure 3.9.

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

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

Figure 5.12 shows how the main program will call the subroutine we just described.

At this stage in the program, the data has been input, and data array D is filled with 1s and 0s that correspond to the logical inputs of the windows and doors we are monitoring.

```
 95  REM READ WINDOWS AND DOORS (LINES 100 - 110)
100  A(1) = PEEK(I/O add + 2)
110  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    D(I) = 0
140  NEXT I
150  T = 1
160  F = 1
170  GOSUB 500 REM CALL THE SUBROUTINE
180  T = 2
190  F = 9
200  GOSUB 500 REM CALL THE SUBROUTINE AGAIN
```

Figure 5.12: *This section of the main program shows the subroutine calls.*

5.6 Simulation of all Windows and Doors for Program Development

Now we can begin to write the software that will display our results on screen. This is an empirical, trial-and-error process: we first try for what we think the output display should look like and then, based on what we see, we adjust our program to make the output more visually attractive.

During our program development, it would be useful to simulate the opening and closing of any combination of doors and windows, as shown in Figure 5.13. Of course, we could simply go to the door or window, and open or close it. But it would be frustrating to get up from the computer whenever we tried a different combination of door and window settings. By using either hardware or software to fill data array D with any combination of 1s or 0s we could simulate the opening or closing of any combination of windows or doors.

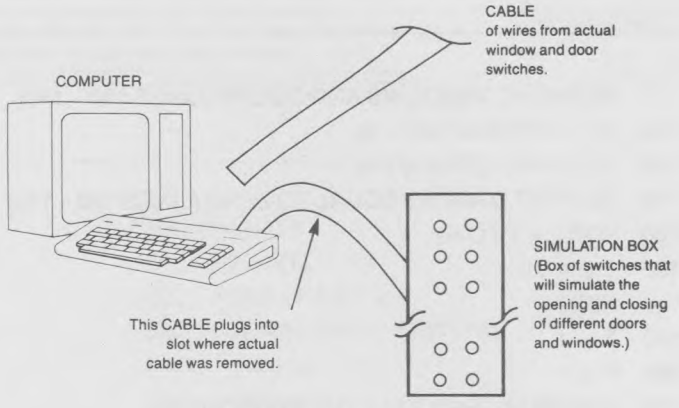


Figure 5.13: *The cable from the external window and door switches can be disconnected from the Commodore 64 and a simulation switch box can be connected in its place. This type of hardware box can be used for software development and for system debugging.*

The hardware technique uses a switch box, which we can build, that will connect to the I/O circuit in place of the cable from the door or window switches. By using this technique, we can simply run our program and flip a switch on the box to change the status of any door or window.

The software technique for simulating can be as exotic as you wish. However, remember that this is a short-term effort: You probably will not use the program once the system is developed.

Our program asks you to enter the value of each array D element, 1 or 0. The program will allow you to change only one array value if you want.

After setting the array-D values, the program will jump directly to the subroutine that will output the display. We add this subroutine to the main program during software development. When we finish the development stage, we can delete this section, which is shown in Figure 5.14.

```
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  D(Y) = 1
80  GOTO 200
85  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   D(I) = 0
150   GOTO 170
160   D(I) = 1
170 NEXT I
200 REM GOTO main program
```

Figure 5.14: *This subroutine will simulate the opening and closing of various doors and windows. When software development is completed, you can delete these lines.*

At line 200, which is the final GOTO, you would jump to the line in the main program which outputs information on a screen based on the value in data array D.

If you choose to use hardware for the simulation, you can follow the schematic in Figure 5.15. In this diagram, each switch

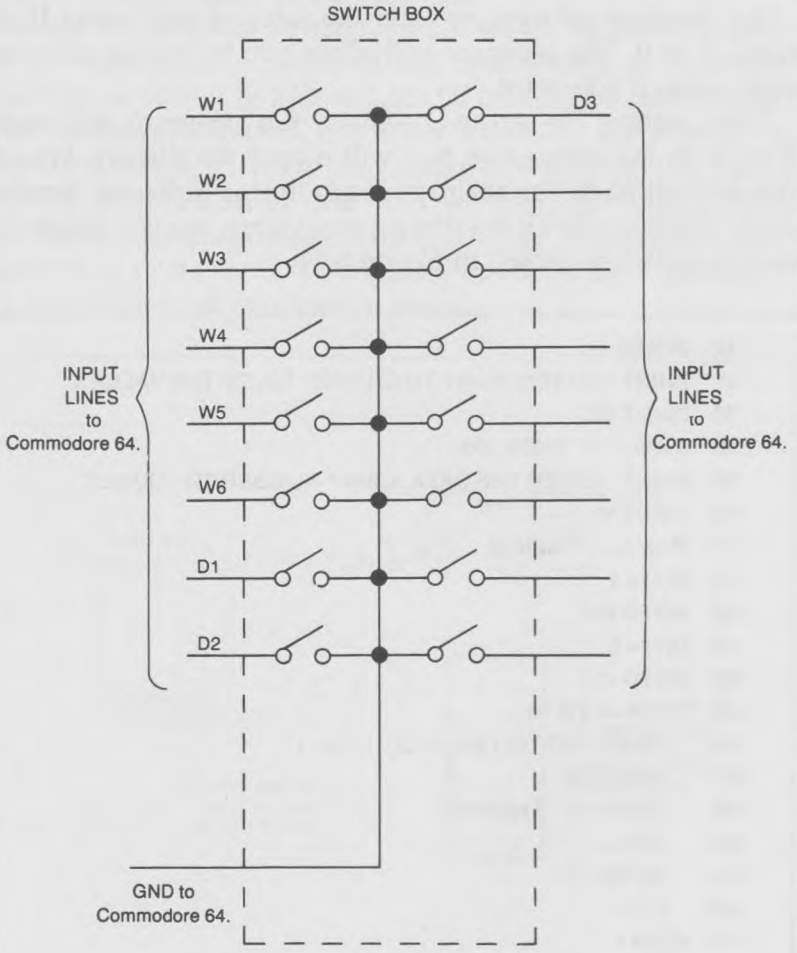


Figure 5.15: *This is a schematic diagram of a hardware switch box.*

corresponds to a window or door that we are monitoring. Hardware simulation can be used as a debugging tool in case your system becomes defective. You can use the simulation box to verify that all of the interface hardware in the I/O circuit is operational.

5.7 Masking Off the Alarms with Software

Sometimes, you will not want an open window or door to cause an alarm. When it is very hot, for example, you may wish to leave a window or door open. In this section, we will write a program to 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 is especially important.)

The program will ask you if you want to mask off a certain window or door. The numbers will reside in a mask array labeled M, which will have as many elements as data array D that we discussed earlier. If an element of mask array M equals a logical 0 (that is, if the user types "N"), then the alarm is not masked. If the element equals a logical 1, then the alarm is masked. For example, if we want to mask the alarm of door D1, then M(7) = 1. Figure 5.16 lists the program.

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

Figure 5.16: *This routine will mask off any alarm you choose.*

At the end of this program, all masks will be set. Remember that if the entry in mask array M is a logical 1, then the mask is set. If the entry is a logical 0, then the mask is cleared. The array is also used when the Commodore 64 generates the corrected display on screen.

5.8 The Complete System

So far in this chapter, we have presented most of the software and hardware pieces for the system separately. In this section, we will bring all of these pieces together. The software will be broken into functional parts, with explanations of any changes. First, let's look at the program so far, shown in Figure 5.17.

```
 10 DIM A(5),D(16),M(16),A$(3)
 20 PRINT CHR$(147)
 30 FOR I = 1 TO 16
 40   D(I) = 0
 50   M(I) = 0
 60 NEXT I
 70 REM THE ABOVE ZEROED OUT ARRAYS
 80 PRINT "DO YOU WANT TO MASK ANY DOOR OR WINDOW?"
 90 INPUT A$
100 IF A$ = "N" THEN 180
110 FOR I = 1 TO 16
120   PRINT "DO YOU WANT TO MASK M(";I;")";
130   INPUT A$
140   IF A$ = "N" THEN 180
150   M(I) = 1
160 NEXT I
170 REM DRAW THE SCREEN NOW
180 PRINT CHR$(147)
190 REM SET COLOR OF EACH SCREEN CELL BLACK
```

Figure 5.17: *This is our program so far.*

```
200  FOR I = 55296 TO 56295
210    POKE I,0
220  NEXT I
230  REM TOP LINE OF SCREEN
240  FOR I = 1106 TO 1140
250    POKE I,99
260  NEXT I
270  REM CORNERS
280  POKE 1105,79
290  POKE 1141,80
300  REM SIDES OF HOUSE
310  FOR I = 1145 TO 1584 STEP 40
320    POKE I,101
330    POKE I + 36,103
340  NEXT I
350  REM CORNER + SIDE
360  POKE 1585,101
370  POKE 1621,122
380  REM FIRST BOTTOM LINE
390  FOR I = 1600 TO 1620
400    POKE I,100
410  NEXT I
420  REM SMALLER SIDES
430  FOR I = 1625 TO 1906 STEP 40
440    POKE I,101
450    POKE I + 14,103
460  NEXT I
470  REM CORNERS
480  POKE 1945,76
490  POKE 1959,122
500  REM BOTTOM LINE
510  FOR I = 1946 TO 1958
520    POKE I,100
530  NEXT I
540  REM SCREEN IS NOW DRAWN, NO LABELS IN YET
```

Figure 5.17 *(continued)*

```
550  REM NOW TO GET THE PEEK DATA
560  A(1) = PEEK (56832 + 2)
570  A(2) = PEEK (56832 + 4)
580  T = 1
590  F = 1
600  GOSUB 670
610  T = 2
620  F = 9
630  GOSUB 670
640  GOTO 770
650  REM SUBROUTINE TO FILL DATA ARRAY
660  REM
670  R1 = 128
680  R2 = A(T)
690  FOR I = F TO F + 7
700    IF R2 - R1 < 0 THEN 730
710    R2 = R2 - R1
720    D(I) = 1
730    R1 = R1/2
740  NEXT I
750  RETURN
```

Figure 5.17 (continued)

Statements 10–640 will ask you to input the masked alarms, and then will draw the outline of the house on screen. Finally, the system will input the status of the windows and doors in the print section to fill in the screen-display outline.

We now need a new section that will compare the logical input data from the windows and doors against the mask data, and that will write different information to the screen based on this comparison. Three possible conditions to write to the display are:

1. NO ALARM. The display will show the window or door number in *black*.
2. ALARM. The display will show the window or door label in *yellow*.

3. ALARM but MASKED. The display will show the window or door label in *light blue.*

We must take care of several details before we set up this section. First, we need to assign letters and numbers that will define the labels for each door or window on our screen outline of the house. We have used two-character labels in earlier chapters: for example, a window could be W1, and a door could be D1. We will incorporate these labels into DATA statements.

We will describe only one method to accomplish this. The DATA statement for each label will be formatted like this:

```
DATA  23,1034,49,1035
      ↓  ↓   ↓  ↓
      W mem  1 mem
        add    add
```

Each label has four data parts, two for each character in the label. The first part consists of the decimal equivalent of the first letter, which is either W or D, and its video memory location. The second character of the label also has two numbers associated with it. We will use four values to poke the data into the screen memory. Figure 5.18 shows the memory locations and layout of the Commodore 64 video memory.

Because there are nine labels, W1–W6 and D1–D3, we will use nine data statements. Each statement will contain the necessary information for one label. You do not have to use nine data statements, but it makes the organization of the data much simpler to understand in our case.

After we write the label into the screen location, we must set it to the correct color. We will use black, yellow, and light blue. To make the label the correct color, we must fill the color array for the screen with the appropriate color number: black = 0, light blue = 14, and yellow = 7. The color memory grid looks like the grid shown in Figure 5.18, except all addresses are offset by 54272.

To put the correct color into the color memory array, we must check if there was an alarm at the specified window or door. Figure 5.19 shows the software necessary to write the correct color labels to the screen of the Commodore 64. Figure 5.20 gives the complete program for our home-security system.

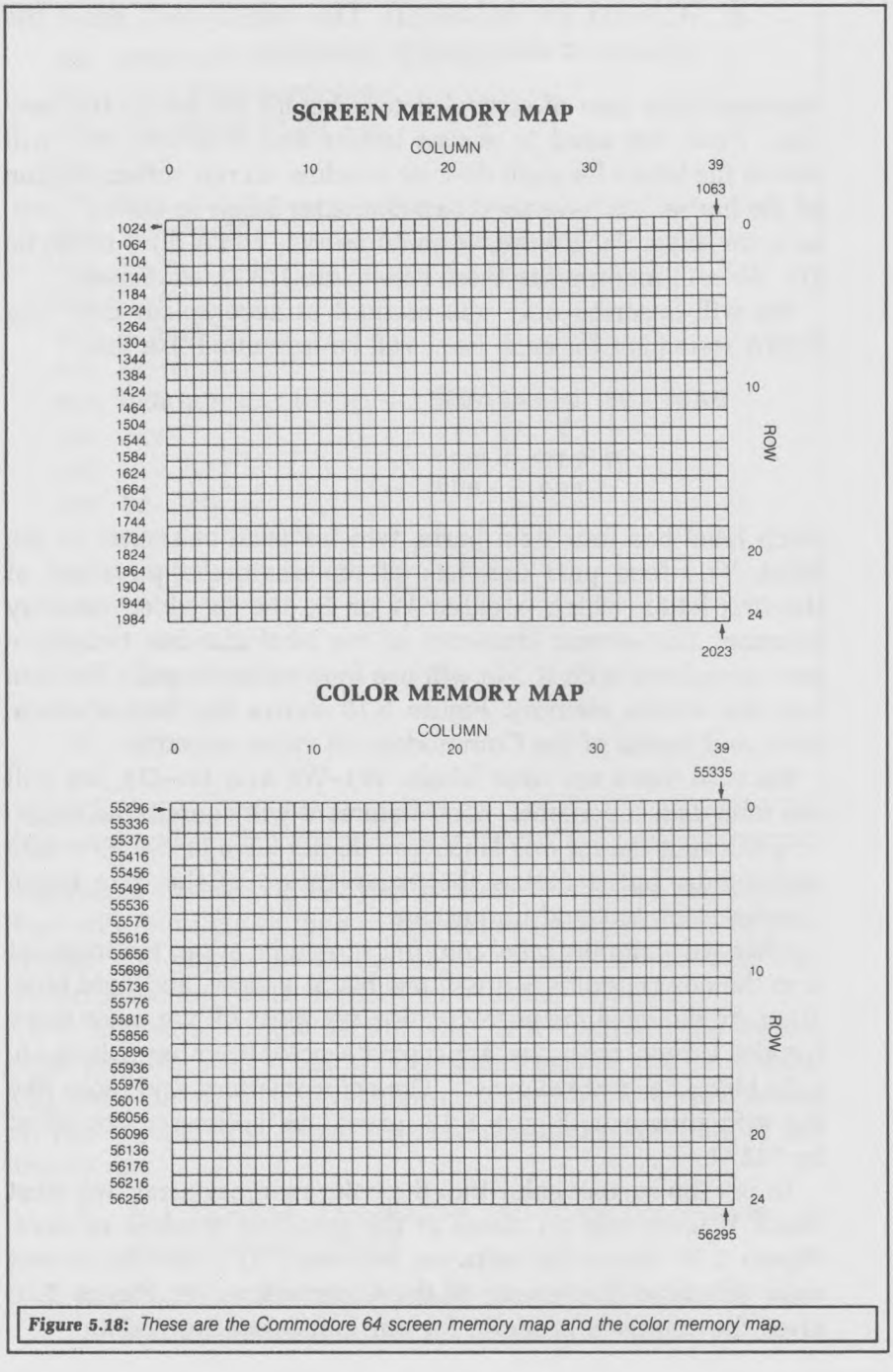


Figure 5.18: *These are the Commodore 64 screen memory map and the color memory map.*

```
760  REM THIS WILL PUT IN LABELS OF CORRECT COLOR
770  RESTORE
780  FOR I = 1 TO 9
790    REM MAX NUMBER OF LINES TO CHECK
800    READ V1,V2
810    READ V3,V4
820    IF D(I) = 0 THEN Q1 = 0 : REM NO ALARM
830    IF D(I) = 1 AND M(I) = 0 THEN Q1 = 7 : REM ALARM
840    IF D(I) = 1 AND M(I) = 1 THEN Q1 = 14 : REM ALARM BUT
       MASKED
850    POKE V2,V1
860    POKE V4,V3
870    POKE V2 + 54272,Q1
880    POKE V4 + 54272,Q1
890  NEXT I
900  REM WRITE W = WINDOWS, D = DOORS
910  FOR I = 1809 TO 1816
920    READ V1
930    POKE I,V1
940  NEXT I
950  FOR I = 1849 TO 1854
960    READ V1
970    POKE I,V1
980  NEXT I
990  REM FINISHED WITH A SINGLE PASS
1000 GOTO 560
1010 DATA 23,1110,49,1111
1020 DATA 23,1118,50,1119
1030 DATA 4,1134,49,1135
1040 DATA 23,1421,51,1461
1050 DATA 23,1545,52,1546
1060 DATA 23,1605,53,1606
1070 DATA 4,1612,50,1613
```

Figure 5.19: *This routine will examine the D and M arrays to see if an alarm is present. It then will print the labels in their correct colors, based on the alarm conditions.*

```
1080 DATA 4,1799,51,1839
1090 DATA 23,1952,54,1952
1100 DATA 23,61,23,9,14,4,15,23
1110 DATA 4,61,4,15,15,18
1120 END
```

Figure 5.19 (continued)

```
 10 DIM A(5),D(16),M(16),A$(3)
 20 PRINT CHR$(147)
 30 FOR I = 1 TO 9
 40    D(I) = 0
 50    M(I) = 0
 60 NEXT I
 70 REM THE ABOVE ZEROED OUT ARRAYS
 80 PRINT "DO YOU WANT TO MASK ANY DOOR OR WINDOW?"
 90 INPUT A$
100 IF A$ = "N" THEN 180
110 FOR I = 1 TO 9
120    PRINT "DO YOU WANT TO MASK M(";I;")";
130    INPUT A$
140    IF A$ = "N" THEN 160
150    M(I) = 1
160 NEXT I
170 REM DRAW THE SCREEN NOW
180 PRINT CHR$(147):REM HOME CURSOR
190 REM SET COLOR OF EACH SCREEN CELL BLACK
200 FOR I = 55296 TO 56295
210    POKE I,0
220 NEXT I
230 REM TOP LINE OF SCREEN
240 FOR I = 1106 TO 1140
```

Figure 5.20: *This is the complete program for the home-security system presented in this chapter.*

```
250    POKE 1,99
260  NEXT I
270  REM CORNERS
280  POKE 1105,79
290  POKE 1141,80
300  REM SIDES OF HOUSE
310  FOR I = 1145 TO 1584 STEP 40
320    POKE I,101
330    POKE I + 36,103
340  NEXT I
350  REM CORNER + SIDE
360  POKE 1585,101
370  POKE 1621,122
380  REM FIRST BOTTOM LINE
390  FOR I = 1600 TO 1620
400    POKE I,100
410  NEXT I
420  REM SMALLER SIDES
430  FOR I = 1625 TO 1906 STEP 40
440    POKE I,101
450    POKE I + 14,103
460  NEXT I
470  REM CORNERS
480  POKE 1945,76
490  POKE 1959,122
500  REM BOTTOM LINE
510  FOR I = 1946 TO 1958
520    POKE I,100
530  NEXT I
540  REM SCREEN IS NOW DRAWN, NO LABELS IN YET
550  REM NOW TO GET THE PEEK DATA
560  A(1) = PEEK (56832 + 2)
570  A(2) = PEEK (56832 + 4)
580  T = 1
590  F = 1
```

Figure 5.20 *(continued)*

```
600 GOSUB 670
610 T = 2
620 F = 9
630 GOSUB 670
640 GOTO 770
650 REM SUBROUTINE TO FILL DATA ARRAY
660 REM
670 R1 = 128
680 R2 = A(T)
690 FOR I = F TO F + 7
700   IF R2 - R1 < 0 THEN 730
710   R2 = R2 - R1
720   D(I) = 1
730   R1 = R1/2
740 NEXT I
750 RETURN
760 REM THIS WILL PUT IN LABELS OF CORRECT COLOR
770 RESTORE
780 FOR I = 1 TO 9
790   REM MAX NUMBER OF LINES TO CHECK
800   READ V1,V2
810   READ V3,V4
820   IF D(I) = 0 THEN Q1 = 5
830   IF D(I) = 1 AND M(I) = 0 THEN Q1 = 2
840   IF D(I) = 1 AND M(I) = 1 THEN Q1 = 7
850   POKE V2,V1
860   POKE V4,V3
870   POKE V2 + 54272,Q1
880   POKE V4 + 54272,Q1
890 NEXT I
900 REM WRITE W = WINDOWS, D = DOORS
910 FOR I = 1809 TO 1816
920   READ V1
930   POKE I,V1
940 NEXT I
```

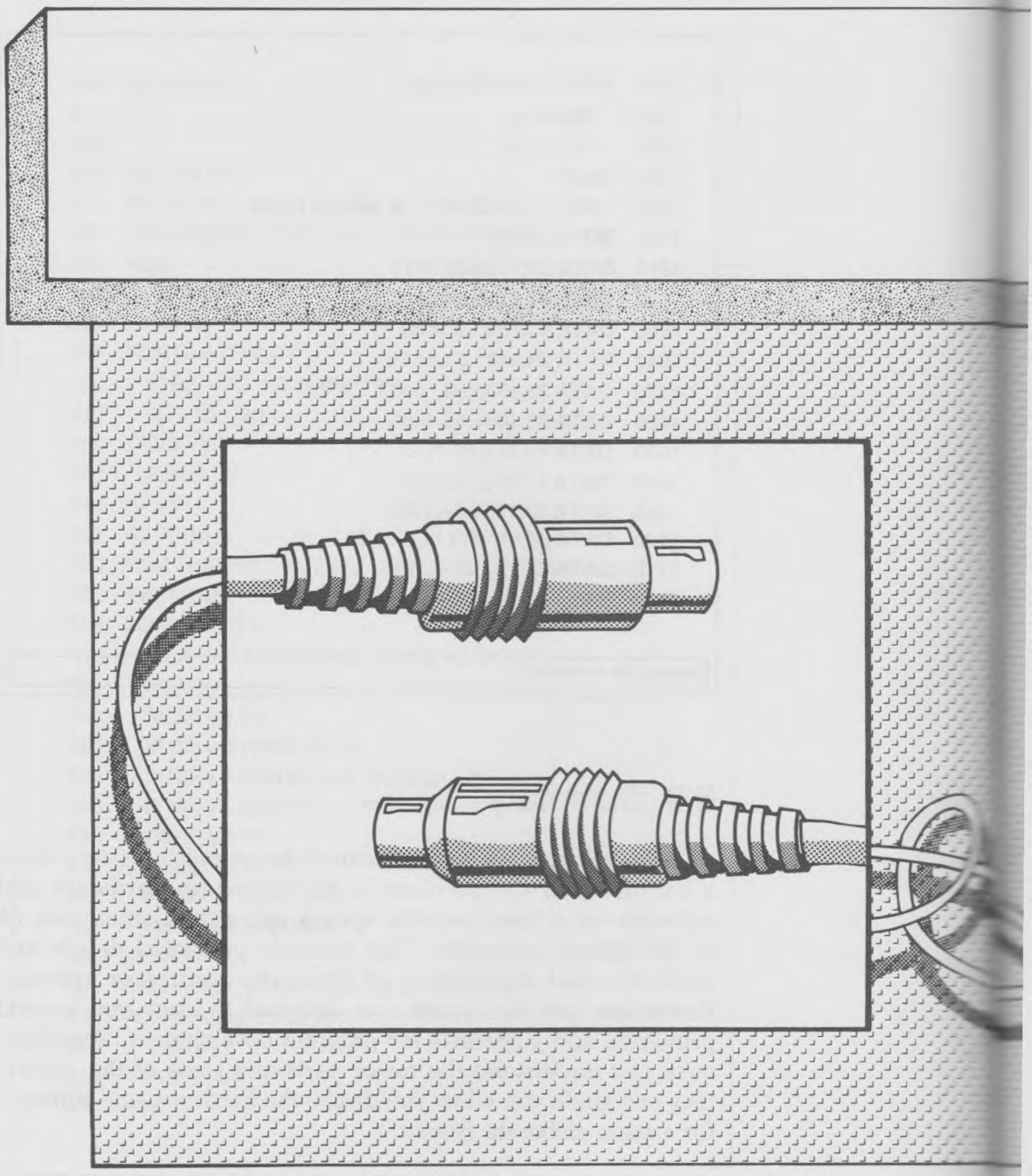
Figure 5.20 *(continued)*

```
 950 FOR I = 1849 TO 1854
 960   READ V1
 970   POKE I,V1
 980 NEXT I
 990 REM FINISHED WITH A SINGLE PASS
1000 GOTO 560
1010 DATA 23,1110,49,1111
1020 DATA 23,1118,50,1119
1030 DATA 4,1134,49,1135
1040 DATA 23,1421,51,1461
1050 DATA 23,1545,52,1546
1060 DATA 23,1605,53,1605
1070 DATA 4,1612,50,1613
1080 DATA 4,1799,51,1839
1090 DATA 23,1952,54,1953
1100 DATA 23,61,23,9,14,4,15,23
1110 DATA 4,61,4,15,15,18
1120 END
```

Figure 5.20 *(continued)*

5.9 Summary

In this chapter, we have presented the complete design—from a definition of the problem to the necessary hardware and software—of a home-security system using the Commodore 64 as the system controller. This example presented details and problems that are typical of computer-controlled systems. Remember that the system was designed to teach you general principles and guidelines for program development. Therefore, once you understand the major parts and steps of the system, they can guide you when you design the hardware and software for a more elaborate system.



ADDING A VOICE TO THE COMMODORE 64

6

In this chapter, we will try to add a computer voice to the Commodore 64. This topic falls under the broad catagory of "Voice Synthesis." After we examine a general block diagram of a speech circuit, we will present the hardware for a speech synthesizer, which is extremely simple and easy to use with your Commodore 64. We then will begin to write the software for voice control, giving examples of BASIC programs for generating words, phrases, or commands. The speech synthesizer circuit allows you to use an unlimited vocabulary of words.

This chapter will show you just how easy it is to make your system talk when you want and say what you want. The circuit we present is very easy to construct, even if you are a hardware beginner. This chapter will also introduce you to how to use speech synthesis for robotics and machine control.

6.1 Phoneme Speech

The type of speech synthesis we will use is called *phoneme speech synthesis* (PSS). A *phoneme* is defined as a language's smallest fundamental unit. Each word we speak is comprised of a set of phonemes—a group of sounds—that, when spoken in succession, produce the entire word.

Let's look at the word *baby* as an example to show the different sounds that comprise that word. Of course, the sounds will depend on where you are from and what accent you speak with. You can use phonemes to tailor a particular word to suit your taste.

Baby is made of four individual phonemes. The first is "B," which does not sound like the letter of the alphabet, but which sounds more like the letters "BI" with the I being short. Next, the phoneme "AY" is used. This sounds like the long A in the word *made*. The last two are the "BI" and "EE" phonemes. "EE" sounds like the long "E" in the word *eat*. By stringing these four sounds together, we can form the complete word *baby*.

Any word may be broken down into its respective phonemes. To reproduce a word, we only need to output its phonemes in the correct sequence. The speed at which the phonemes are output will change the pitch of the word.

6.2 The Set of Phonemes

Figure 6.1 shows a set of 64 phonemes that are available for the Votrax phoneme-synthesizer chip, the SC-01, which we will use to generate a voice for our computer. Figure 6.1 gives the phoneme hexadecimal code, phoneme symbol, output duration, and example word for each phoneme. The hexadecimal

code is the 8-bit binary code, written in hexadecimal notation, which will produce the corresponding phoneme sound when applied to the SC-01 chip. We will use all four categories later in this chapter. However, at this point, Figure 6.1 may be easier to

Phoneme Code	Phoneme Symbol	Duration (ms)	Example Word	Phoneme Code	Phoneme Symbol	Duration (ms)	Example Word
00h	EH3	59	jacket	20h	A	185	day
01h	EH2	71	enlist	21h	AY	65	day
02h	EH1	121	heavy	22h	Y1	80	yard
03h	PAØ	47	no sound	23h	UH3	47	mission
04h	DT	47	butter	24h	AH	250	mop
05h	A2	71	made	25h	P	103	past
06h	A1	103	made	26h	O	185	cold
07h	ZH	90	azure	27h	I	185	pin
08h	AH2	71	honest	28h	U	185	move
09h	13	55	inhibit	29h	Y	103	any
0Ah	12	80	inhibit	2Ah	T	71	tap
0Bh	11	121	inhibit	2Bh	R	90	red
0Ch	M	103	mat	2Ch	E	185	meet
0Dh	N	80	sun	2Dh	W	80	win
0Eh	B	71	bag	2Eh	AE	185	dad
0Fh	V	71	van	2Fh	AE1	103	after
10h	CH*	71	chip	30h	AW2	90	salty
11h	SH	121	shop	31h	UH2	71	about
12h	Z	71	zoo	32h	UH1	103	uncle
13h	AW1	146	lawful	33h	UH	185	cup
14h	NG	121	thing	34h	02	80	for
15h	AH1	146	father	35h	01	121	aboard
16h	001	103	looking	36h	IU	59	you
17h	00	185	book	37h	U1	90	you
18h	L	103	land	38h	THV	80	the
19h	K	80	trick	39h	TH	71	thin
1Ah	J*	47	judge	3Ah	ER	146	bird
1Bh	H	71	hello	3Bh	EH	185	get
1Ch	G	71	get	3Ch	E1	121	be
1Dh	F	103	fast	3Dh	AW	253	call
1Eh	D	55	paid	3Eh	PA1	185	no sound
1Fh	S	90	pass	3Fh	STOP	47	no sound

/T/ must precede /CH/ to produce CH sound.
/D/ must precede /J/ to produce J sound.

Figure 6.1: *This is a list of the 64 phoneme codes that can be produced by the Votrax SC-01 chip. Reproduced by permission of Votrax, a division of Federal Screw Works.*

understand if you concentrate on the phoneme symbol and the example word.

The example word indicates what the phoneme will sound like when generated electronically. We can use the phonemes that produce no sound—PA0, PA1, and STOP—for pauses between words. We will show examples of this later.

6.3 How are the Correct Phonemes Chosen?

Choosing the correct phonemes for a particular word can be very complicated. There are reference tables to help you get started. However, because of the many subtle differences in dialect, phoneme selection becomes more difficult when you try to tailor words to match your own speech.

A good method of selecting the correct phonemes for a particular word is to use a ready-made list, which is usually supplied by the manufacturer of the particular PSS chip you are using. Figure 6.2 gives two words using phonemes supplied by Votrax for the SC-01 chip. Appendix D gives a more complete list of sample words.

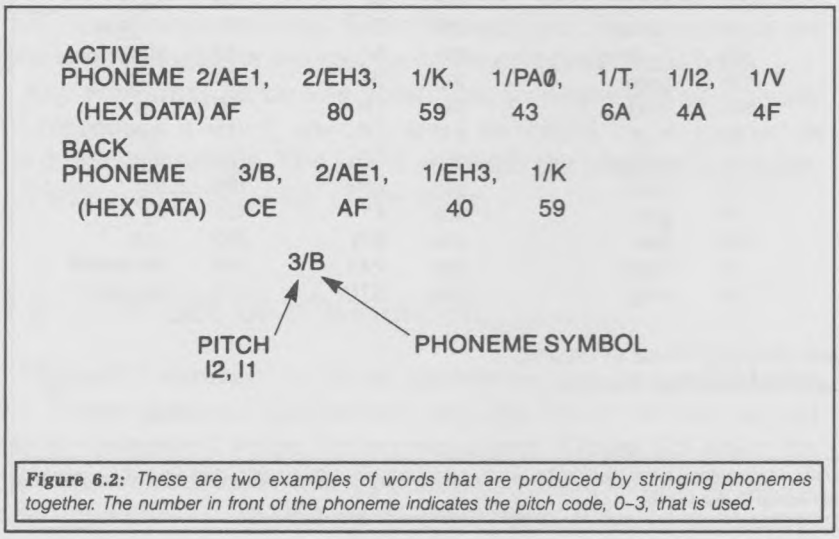


Figure 6.2: *These are two examples of words that are produced by stringing phonemes together. The number in front of the phoneme indicates the pitch code, 0–3, that is used.*

For words not included in the ready-made list, try to make your new words from "pieces" of ready-made words. This technique will usually get you in the ball park. For example, suppose you wanted the phonemes for *Saturn*, a word not on the ready-made list. We can create this word using the first part of *Saturday* and the last part of *return*. Figure 6.3 shows the phonemes of *Saturday* and *return* that will be used to produce the phonemes of *Saturn*.

6.4 The Votrax SC-01 Chips

So far, we have discussed the concept of phoneme speech synthesis. Let's now put that concept to use. We must produce electrically the selected phonemes in the correct order with the correct timing. Fortunately, the SC-01 chip has dramatically simplified this task. Figure 6.4 shows a flowchart of the necessary steps to produce a word using phonemes.

Figure 6.5 shows another flowchart and a pinout of the SC-01 chip. In this figure, let's see how we interface the chip to a control device. Our discussion is intended only to show you how to

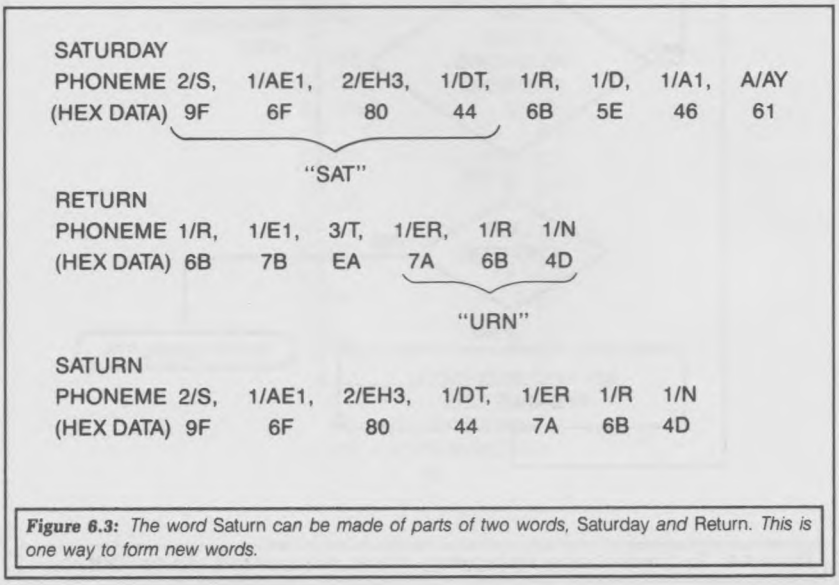


Figure 6.3: *The word* Saturn *can be made of parts of two words,* Saturday *and* Return. *This is one way to form new words.*

use the SC-01 chip to generate phoneme speech. We will not discuss in detail how the device operates internally.

POWER SUPPLY (V_P, V_G)

V_P is the positive voltage supplied to the SC-01 chip. The value is between +7 and +14 volts. Normal operating voltage equals +12 volts. V_G is connected to ground, or zero potential.

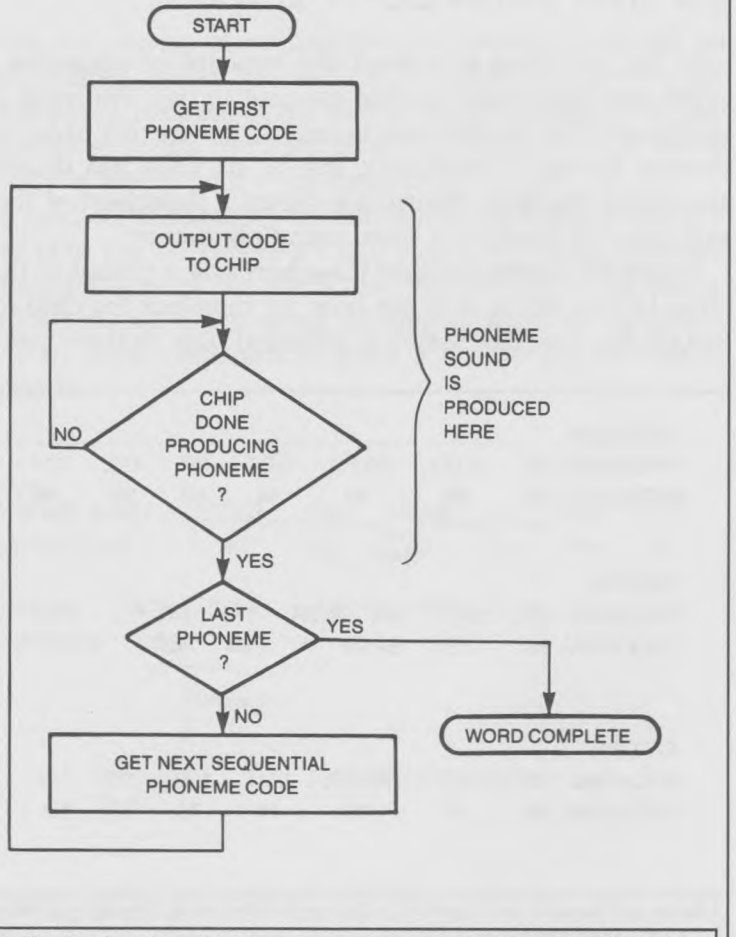
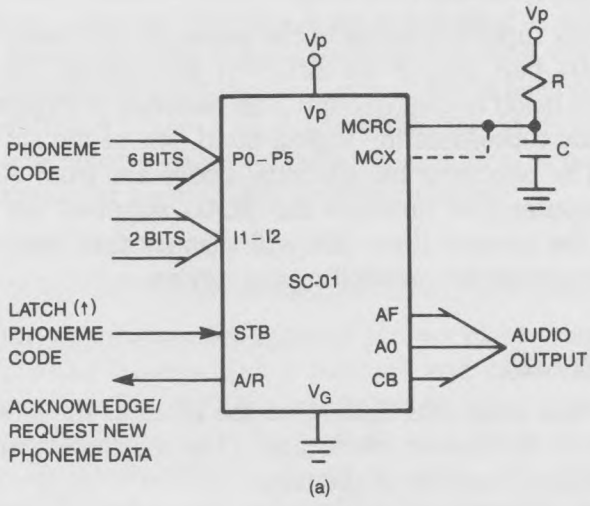
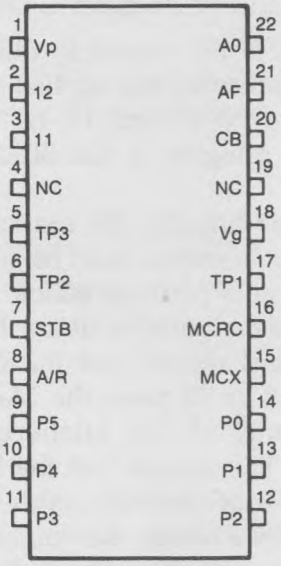


Figure 6.4: *This flowchart shows the necessary steps to produce speech with the SC-01 chip.*





NC = NO CONNECTION
TPX = NO CONNECTION
(b)

Figure 6.5: *This shows a pinout and a flowchart of the SC-01 device.*

PHONEME CODE (P0–P5)

These six input bits point to the phoneme you want to output. The SC-01 can output 64 different phonemes, the codes of which are listed in the first and fifth columns in Figure 6.1. Phoneme code represents the logical condition of the six input bits P0–P5. The phoneme hexadecimal codes are from 00h to 3Fh. The computer that controls the SC-01 supplies the phoneme code at the correct time. We will discuss this later when we show the circuit for controlling the device.

PITCH (I1,I2)

These two input bits determine the pitch of the selected phoneme. 00 is the lowest pitch, and 11 is the highest pitch. I1 is the least-significant bit of the two.

STB (STROBE FOR PHONEME CODE)

This input signal is the strobe to the SC-01 that indicates when the external controller has applied the phoneme code and the inflection bits to P0–P5 and I1, I2. When the signal goes from a logical 0 to a logical 1, the SC-01 will operate on the applied inputs.

The phoneme code bits (P0–P5) are latched with the strobe input. The I1, I2 bits, however, must be latched externally to the device. This is a simple process, which we will discuss when we examine the actual circuit for using the SC-01.

The STB input must remain low for a minimum amount of time that is specified as 72 times the Master Clock Frequency. At a clock frequency of 720 kilohertz, the period equals 1.40 microseconds. This means that the STB must remain low for longer than 100 microseconds. You may not think this is very long, but when we design the controller for the SC-01, we must use a circuit to meet this specification.

A/R (ACKNOWLEDGE/REQUEST)

This is the handshake line between the SC-01 and the control circuit. When the STB input strobes a new phoneme code into

the SC-01, the output is set low. After the SC-01 has processed the phoneme code, the A/R line is set to a logical 1, which indicates the device is ready to accept a new phoneme code.

The control hardware will monitor the logical level of the A/R line to determine when a new phoneme should be input. We will give examples of how we can use this line in a *polled* or *interrupt* mode for a controlling microprocessor.

MCRC (MASTER CLOCK RESISTOR-CAPACITOR)

This input determines the internal master clock frequency. It is accomplished by selecting a resistor and capacitor whose respective valves (*R* and *C*) are multiplied together giving an RC time constant. In this case, the resistor and capacitor are connected in series forming a series RC network. *Resistance-capacitance* (RC) values are selected for an operating frequency of 720 Kilohertz. The frequency of the master clock is approximately equal to 1.25/RC. Note that the maximum R value is specified at 6.8 K-ohms. When using this RC network as the clock frequency, the MCRC input is connected to the MCX input. If you use an external clock input, the MCRC input is connected to ground.

MCX (MASTER CLOCK EXTERNAL)

This input allows you to use an external clock to determine the operating frequency of the SC-01. If you don't use an external clock, connect this input to the MCRC input line.

AO (AUDIO OUTPUT)

This supplies the analog signal to an audio output device. This type of device is an audio amplifier similar in function to the amplifier in your stereo system.

AF (AUDIO FEEDBACK)

This output signal is used with a class A or class B transistor audio amplifier.

CB (CURRENT SOURCE FOR CLASS B)

This output is the current source for a Class B amplifier.

6.5 Connecting Up the SC-01

In this section, we will discuss how to connect the SC-01 to perform as a phoneme speech synthesizer. We will look at a circuit that is universal and that we can connect not only to our Commodore 64 but to most home or business computers in any computer-controlled system.

Figure 6.6 shows one way we can connect the SC-01 to the Commodore 64. This is a general connection to an output latch from some type of microprocessor system, such as a personal computer or a stand-alone microprocessor system. Let's now review the important points of Figure 6.6.

In Figure 6.6, the power supply pin V_P for the SC-01 chip is connected to +12 volts. The V_G pin is connected to ground. A 0.1-microfarad capacitor is placed between the V_P and V_G pins. The oscillator RC is equal to 6.8 K-ohms and approximately 330 picofarads.

The STB input and pins P0–P5, I1, and I2 are connected to output lines from a microprocessor-controlled system. The 7407 buffer device will translate the 0–5 volt output signals from the Commodore 64 to signals that switch from 0–12 volts, which are required by the SC-01 device. The A/R output from the SC-01 is connected to the computer as an input to be monitored (polled) or as an external interrupt.

We connect the audio output AO from the SC-01 to a simple audio amplifier. Our amplifier circuit must current-buffer A0 before we can use it to drive a speaker. However, if you have a favorite amplifier circuit, do not hesitate to experiment with it in place of the one we show.

Using the circuit shown in Figure 6.6, the SC-01 is capable of an unlimited vocabulary of speech. In the next section, we will discuss the interface to the Commodore 64, before we go on to the different software to control the SC-01.

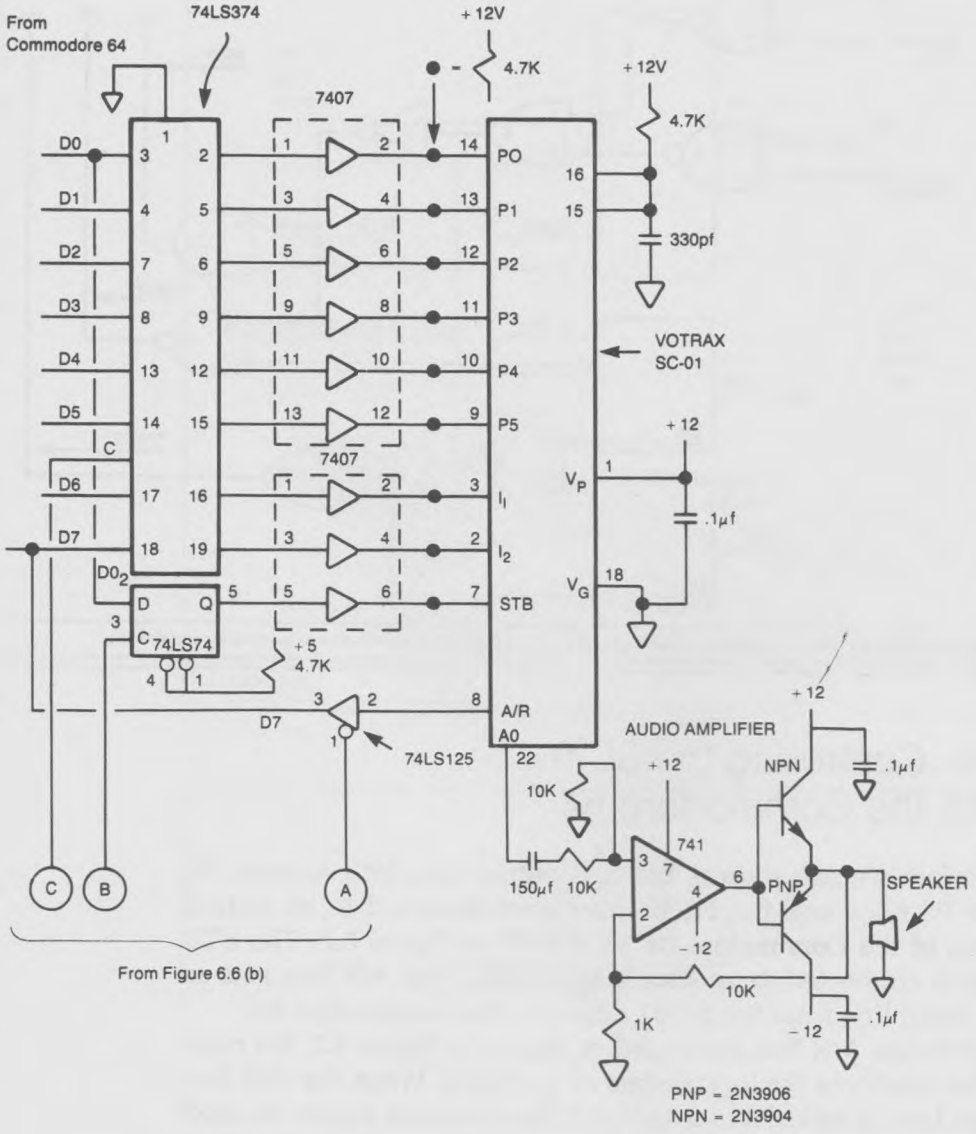


Figure 6.6 (a, b): *This complete schematic shows how to connect the SC-01 device to the Commodore 64.*

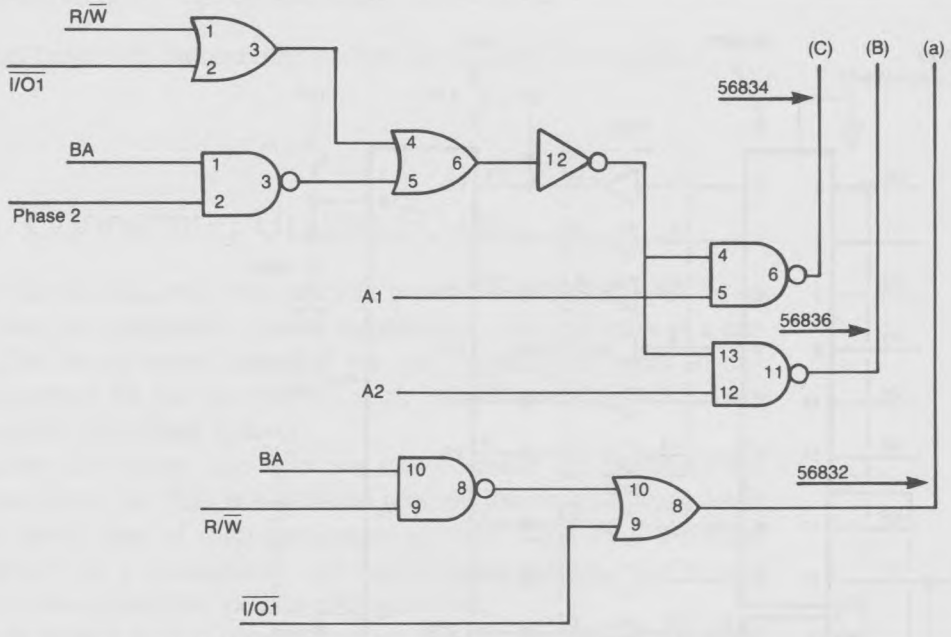


Figure 6.6 (a, b): *This complete schematic shows how to connect the SC-01 device to the Commodore 64.*

6.6 Controlling the SC-01 with the Commodore 64

Before we can discuss the A/R output line, let's assume the PO–P5 lines and the I1, I2 lines are connected to an output latch of the Commodore 64, as shown in Figure 6.7. The STB line is connected to another output latch. The A/R line acts as an input line from the SC-01 device to the Commodore 64.

With the A/R line connected as shown in Figure 6.7, the computer monitors the logical state of the input. When the A/R line goes from a logical 0 to a logical 1, the computer inputs the next phoneme to the SC-01. The A/R output is set to a logical 0 when the STB line goes from a logical 0 to a logical 1.

Figure 6.8 shows a flowchart of the necessary sequence of events to input phonemes to the SC-01 device with the A/R line connected as an input line.

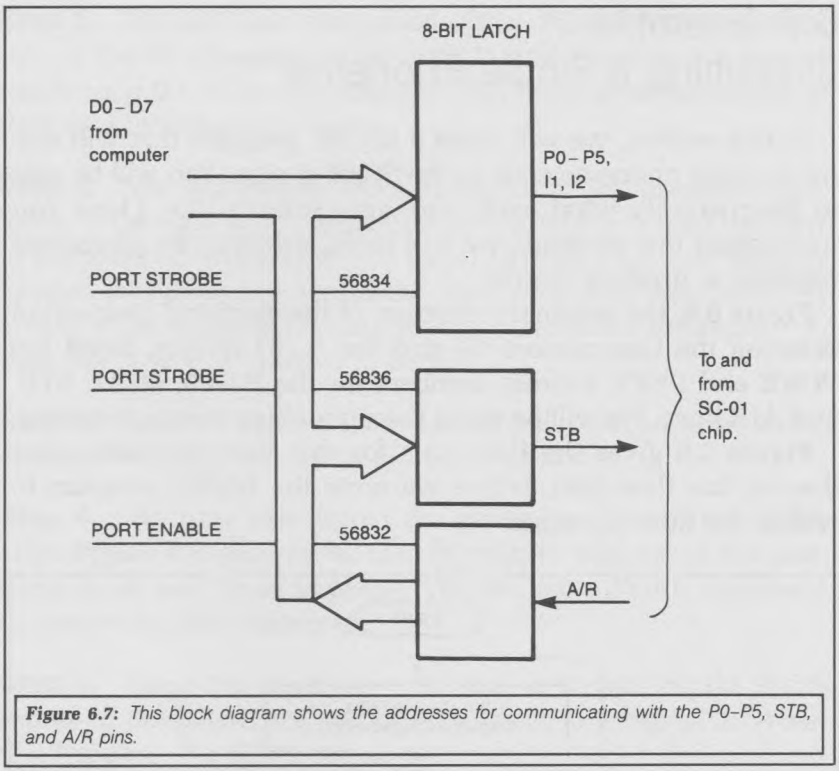


Figure 6.7: *This block diagram shows the addresses for communicating with the P0–P5, STB, and A/R pins.*

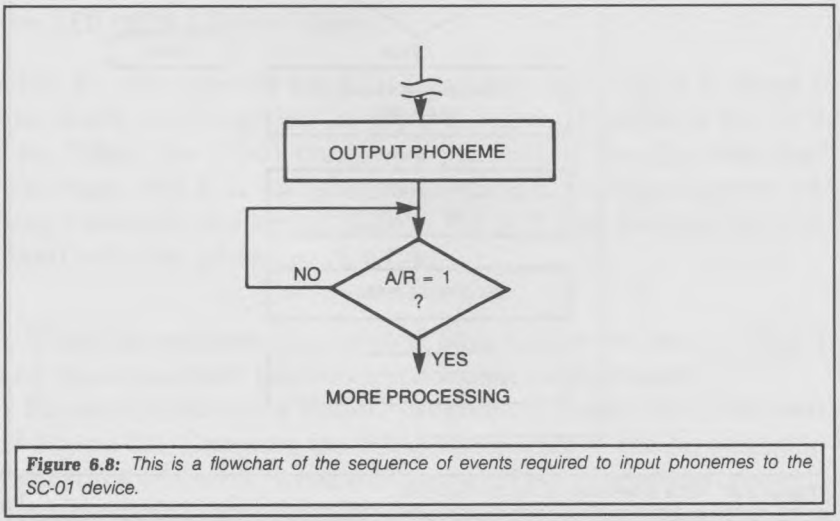


Figure 6.8: *This is a flowchart of the sequence of events required to input phonemes to the SC-01 device.*

6.7 Example 1: Outputting a Single Phoneme

In this section, we will write a BASIC program that will output a single phoneme code to the SC-01 device. You will be able to hear exactly what each phoneme sounds like. Once you understand this program, we will begin stringing the phonemes together to produce words.

Figure 6.6, the schematic diagram of the electrical connection between the Commodore 64 and the SC-01 device, listed the POKE and PEEK address numbers for the P0–P5, I1, I2, STB, and A/R lines. We will be using these numbers in our programs.

Figure 6.9 gives the flowchart for this first program. Let's discuss this flowchart, before we write the BASIC program to realize the flow of events.

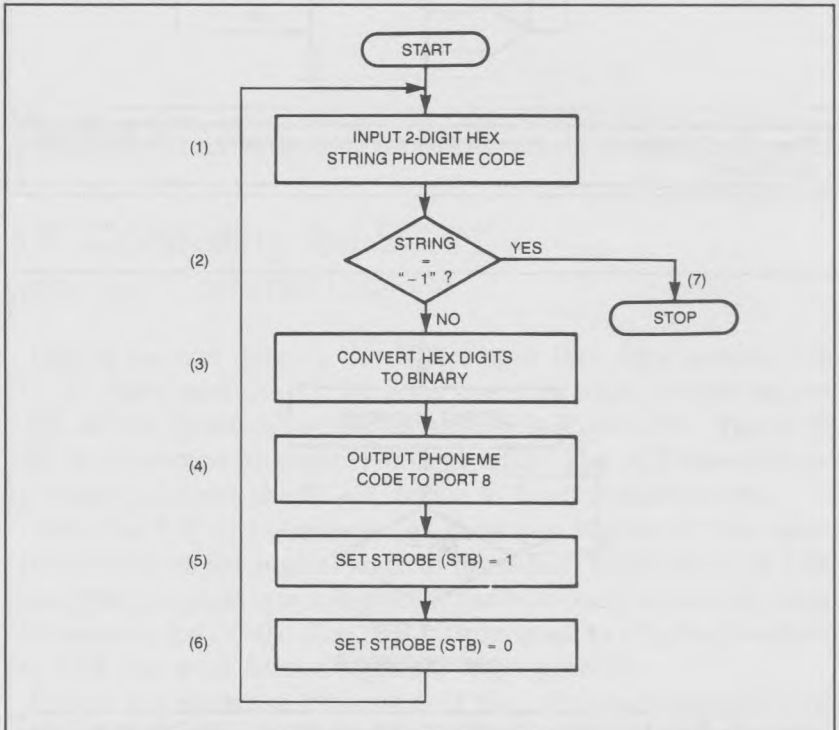


Figure 6.9: *This is a flowchart for the first example.*

Step 1. We will input two hexadecimal digits, which represent one of the 64 phoneme codes, with I1 and I2 set to the correct patch code 00-11. In this step, you can input your hexadecimal data as a two-character string.

Step 2. We will decide whether or not to terminate the program. If the data you entered in Step 1 equals −1, then the flow of the program will proceed to Step 7 (STOP). (You may use any symbol you desire to terminate your program. We chose −1 simply as an example.) If the data you entered did not equal −1, then the flow will proceed to Step 3.

Step 3. We now convert the hexadecimal, two-character string entered in Step 1 into 8 bits of binary data.

Step 4. We now can output the phoneme code to the SC-01 chip. Figure 6.6 showed us that the output address of the phoneme code was equal to 56834. We can use a POKE statement to output the phoneme code.

Step 5. Since the phoneme code now is present on the P0-P5 inputs to the SC-01, we set the STB input to a logical 1. When this happens, the SC-01 device processes the input data and outputs the desired phoneme. The A/R output line will go to a logical 0, when the STB output goes to a logical 1. The address of the STB output line is 56836.

Step 6. We now set the STB input line to a logical 0. Steps 5 and 6 are used together to apply a logical 1 pulse to the STB line. When the SC-01 completes processing the phoneme that was input, the A/R line goes to a logical 0. In this program, we won't examine the logical state of the A/R line, because we will input only one phoneme at a time.

When the program has finished Step 6, it jumps back to Step 1 and inputs another phoneme code from the keyboard.

Figure 6.10 shows a BASIC program to realize the flowchart of Figure 6.9. Compare the REM statements to the flowchart to help you understand what will happen. Our subroutine converts hexadecimal data to decimal data one character at a time.

```
10  DIM T$(2)
30  REM INPUT THE 2-DIGIT HEX STRING
40  PRINT "INPUT THE HEX VALUE FOR EACH
    CHARACTER. -1 = END"
50  INPUT T$
60  IF T$ = "-1" THEN 290
70  REM IF INPUT = -1 THEN STOP
80  GOSUB 190
90  REM SUBROUTINE TO CONVERT HEX TO DECIMAL
100 REM NOW TO OUTPUT PHONEME CODE TO THE SC-01
110 POKE 56834,T1
120 REM NOW TO PULSE THE STB INPUT TO THE SC-01
130 POKE 56836,1
140 POKE 56836,0
150 REM DO IT AGAIN
160 GOTO 40
170 REM THIS ROUTINE WILL CONVERT ASCII TO DECIMAL
180 REM EACH DIGIT IS CONVERTED ONE AT A TIME
190 C$ = MID$(T$,1,1)
200 M1 = 48
210 IF C$ >= "A" THEN M1 = 55
220 X = 16 * (ASC(C$) - M1)
230 C$ = MID$(T$,2,2)
240 M1 = 48
250 IF C$ >= "A" THEN M1 = 55
260 Y = ASC(C$) - M1
270 T1 = X + Y
280 RETURN
290 STOP
```

Figure 6.10: *This is the BASIC program to realize the flowchart of Figure 6.9.*

6.8 Example 2: Outputting Words with the SC-01 Chip

In this example, we will output words with the SC-01 device by inputting all of the phoneme hexadecimal codes required to synthesize a particular word. We will store these hexadecimal

codes in an array in which the last byte of data will be −1.

When you enter −1, the program will output all the phonemes to the SC-01 chip in the order you entered them. If the codes were correct, you will hear a word over and over. For now, it is best to let the chip repeat the word over and over so that you will be certain to hear it. If the chip only outputs the word once, you may miss it. However, this is up to you. Try it different ways to see how you like it.

Figure 6.11 is the flowchart for this program. Let's discuss the points of this flowchart that we did not cover in the preceding example.

Steps 1–5. These input the phoneme codes that will comprise the entire word. After you enter all of the phoneme codes, you will enter a −1.

For example, the phoneme codes for the word *second* are 5F, 82, 59, 71, 4D, 6A. First, you enter these in the order shown. Then you add a 3E, −1. The 3E creates a pause with no sound for approximately 185 milliseconds between the repeating of the word. A −1 logically informs the program that this is the last phoneme code to be entered. The complete data entry for the word *second* is 5F, 82, 59, 71, 4D, 6A, 3E, −1.

Step 6. The program jumps here after we enter the −1.

Steps 7–7A. These output the phoneme data to the SC-01.

Steps 8–9. Here we test the logical level of the A/R output. If this output is a logical 0, the SC-01 is not finished processing the phoneme and we will continue to test the line until it goes to a logical 1. When the A/R line is a logical l, the program goes to Step 10.

Step 10. The program prepares to output the next phoneme.

Step 11. The program will test the output code for a −1. If it equals −1, then the entire word has been output. The program will return to Step 6 and output the word again.

If the code does not equal −1, the program will go to Step 7 to output the next sequential phoneme code that you entered.

Figure 6.12 shows a complete BASIC program to realize the flowchart shown in Figure 6.11.

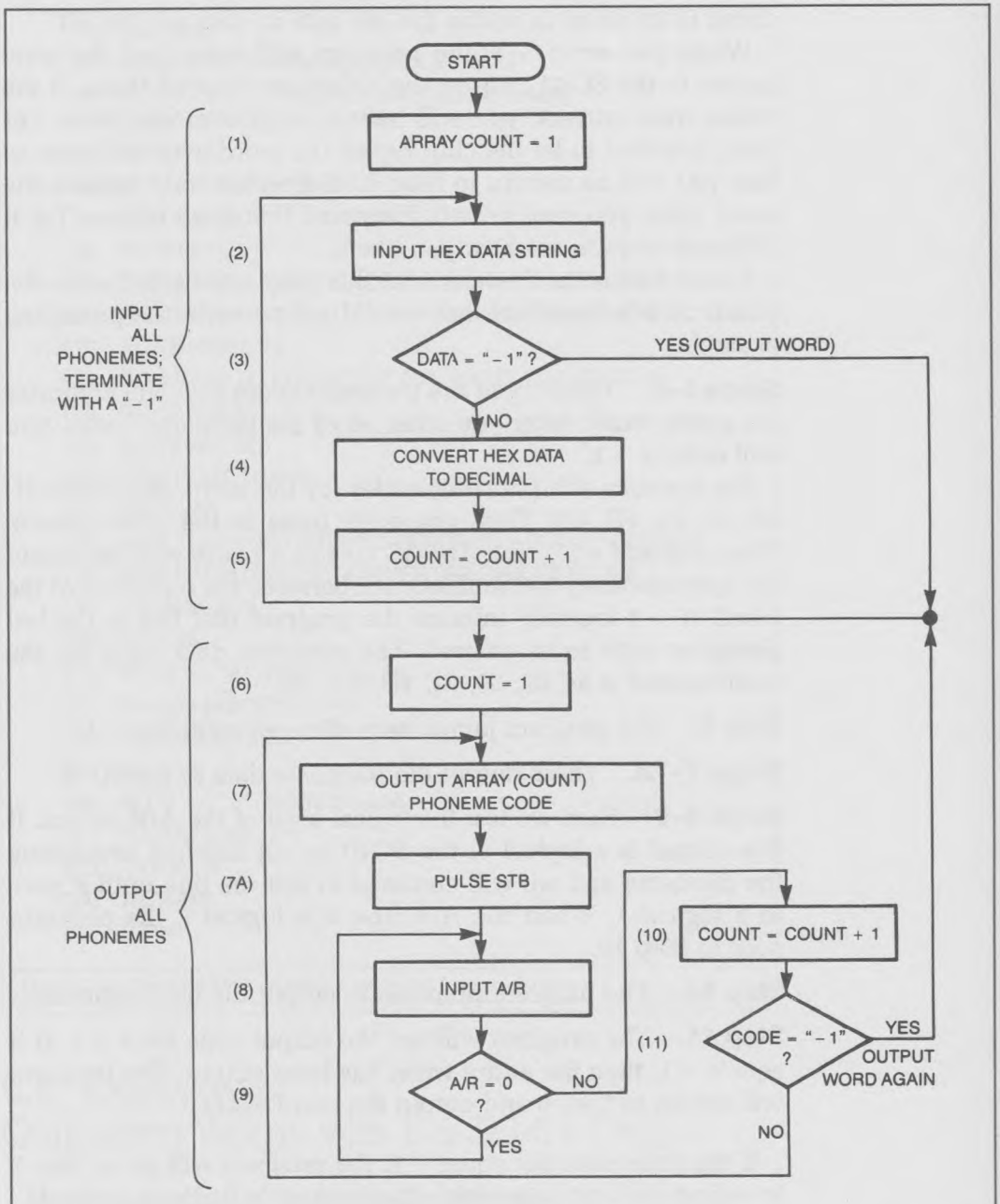


Figure 6.11: *This is the flowchart for a program that will output the same word on the SC-01 chip repeatedly.*

```
10  DIM T$(2),V1(100)
20  C1 = 1
30  REM INPUT THE 2 DIGIT HEX STRING
40  PRINT "INPUT THE HEX VALUE FOR EACH
    CHARACTER – 1 = END"
50  INPUT T$
60  IF T$ = " – 1" THEN 110
70  REM IF INPUT = – 1 THEN END OF STRING
80  GOSUB 320
90  REM SUBROUTINE TO CONVERT HEX TO DECIMAL
100 GOTO 40
110 V1(C1) = – 1
120 REM SET LAST ELEMENT OF ARRAY EQUAL TO – 1
130 REM NOW TO OUTPUT THE PHONEME CODES IN THE ARRAY
140 C1 = 1
150 POKE 56834,V1(C1)
160 REM NOW TO PULSE THE STB INPUT TO THE SC-01
170 POKE 56836,1
180 POKE 56836,0
190 REM INPUT THE LOGICAL VALUE OF A/R
200 A = PEEK(56832)
210 REM A/R IS CONNECTED TO D7 INPUT, < 128 = 0
220 IF A < 128 THEN 200
230 REM CHECK FOR LAST ELEMENT OF ARRAY
240 C1 = C1 + 1
250 IF V1(C1) = – 1 THEN C1 = 1
260 REM IF LAST ELEMENT, THEN START OVER IN ARRAY
270 REM IF NOT LAST ELEMENT, THEN OUTPUT NEXT ELEMENT
280 GOTO 150
290 REM SUBROUTINE FOR HEX TO DECIMAL
300 REM THIS ROUTINE WILL CONVERT ASCII TO DECIMAL
310 REM EACH DIGIT IS CONVERTED ONE AT A TIME
320 C$ = MID$(T$,1,1)
330 M1 = 48
340 IF C$ > = "A" THEN M1 = 55
350 X = 16 * (ASC(C$) – M1)
```

Figure 6.12: *This BASIC program realizes the flowchart of Figure 6.11.*

```
360  C$ = MID$(T$,2,2)
370  M1 = 48
380  IF C$ >= "A" THEN M1 = 55
390  Y = ASC(C$) - M1
400  T1 = X + Y
410  V1(C1) = T1
420  C1 = C1 + 1
430  RETURN
```

Figure 6.12 *(continued)*

6.9 Example 3: Outputting a Sentence with the SC-01

In the first two examples, we have output single phonemes and individual words. Now we are ready to output complete sentences or commands. This program will be very similar to the second example. We will input all the phonemes for each word. Between words, we will enter a 3E phoneme code that will allow for a pause between words. A −1 will mark the end of the sentence.

Most sentences require many hexadecimal phoneme codes. Therefore, we will place the codes into DATA statements instead of inputting the codes while the program is running, as we did before. For example, the sentence "Today is Saturday" will require the phoneme codes:

AA, A8, 5E, 46, 61, 3E, 67, 92, 3E, 9F, 6F, 80, 44, 6B, 5E, 46, 61, 3E, −1

TODAY IS SATURDAY

In this string, notice that there is a 3E between words and a −1 at the end of the sentence.

Figure 6.13 gives the flowchart for the program to output a sentence with the SC-01. Since the data is treated as string data, the program will convert the entire string to decimal data, before outputting the data to the SC-01. Figure 6.14 shows a BASIC program for the flowchart of Figure 6.13.

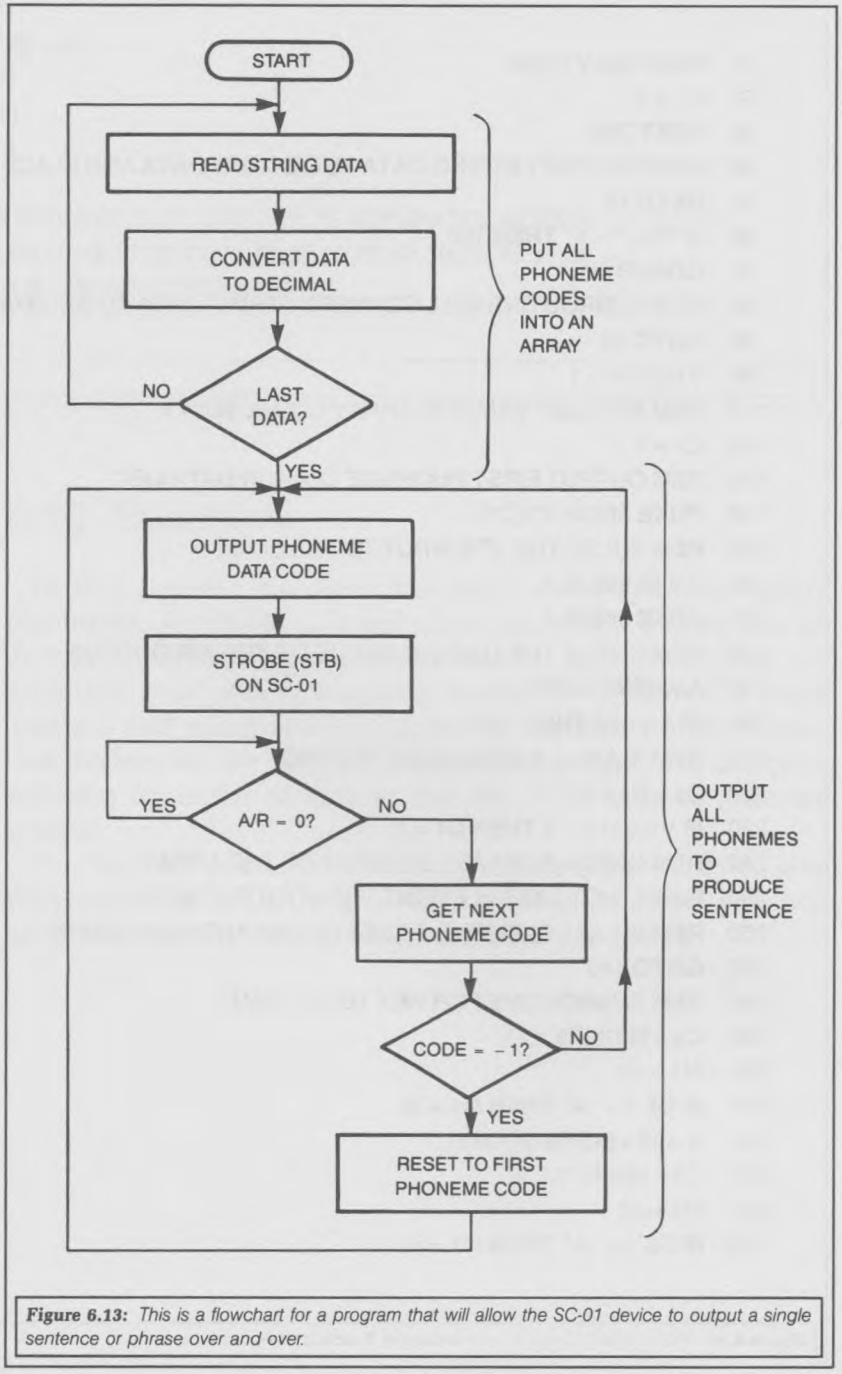


Figure 6.13: *This is a flowchart for a program that will allow the SC-01 device to output a single sentence or phrase over and over.*

```
10  DIM T$(2),V1(300)
20  C1 = 1
30  RESTORE
40  REM CONVERT STRING DATA TO DECIMAL DATA AND PLACE IN ARRAY
50  READ T$
60  IF T$ = " – 1" THEN 100
70  GOSUB 290
80  REM SUBROUTINE WILL CONVERT STRING DATA TO DECIMAL
90  GOTO 50
100 V1(C1) = – 1
110 REM SET LAST VALUE IN ARRAY EQUAL TO – 1
120 C1 = 1
130 REM OUTPUT FIRST PHONEME CODE IN DATA LIST
140 POKE 56834,V1(C1)
150 REM PULSE THE STB INPUT TO THE SC-01
160 POKE 56836,1
170 POKE 56836,0
180 REM CHECK THE LOGICAL VALUE OF THE A/R OUTPUT
190 A = PEEK(56832)
200 IF A < 128 THEN 190
210 REM IF A/R = 0 THEN KEEP TESTING
220 C1 = C1 + 1
230 IF V1(C1) = – 1 THEN C1 = 1
240 REM CHECK FOR LAST ELEMENT OF THE ARRAY
250 REM IF NOT LAST ELEMENT THEN OUTPUT NEXT
260 REM IF LAST ELEMENT, RESET COUNT AND DO IT AGAIN
270 GOTO 140
280 REM SUBROUTINE FOR HEX TO DECIMAL
290 C$ = MID$(T$,1,1)
300 M1 = 48
310 IF C$ > = "A" THEN M1 = 55
320 X = 16 * (ASC(C$) – M1)
330 C$ = MID$(T$,2,2)
340 M1 = 48
350 IF C$ > = "A" THEN M1 = 55
```

Figure 6.14: *This is a BASIC program to realize the flowchart of Figure 6.13.*

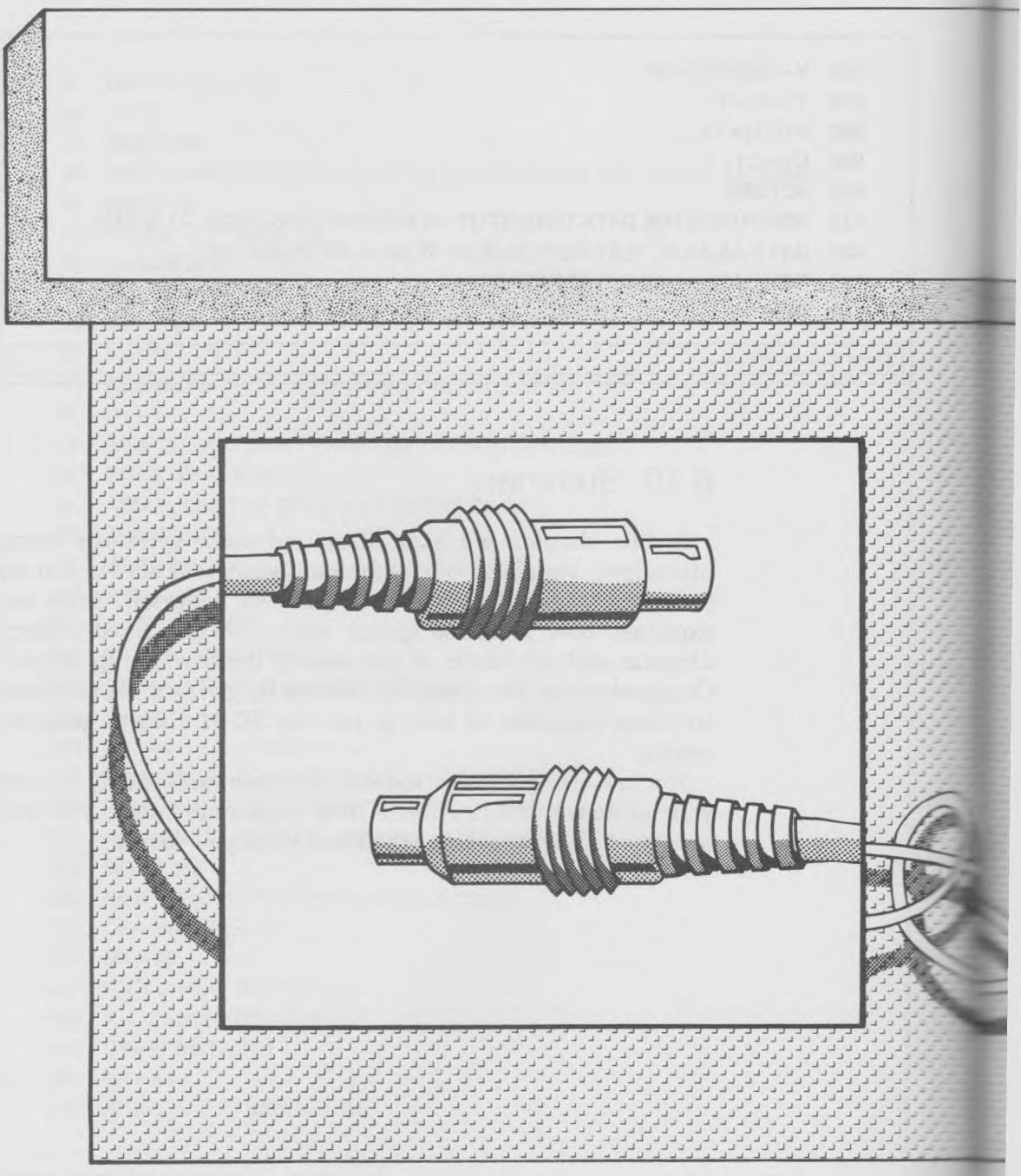
```
360  Y = ASC(C$) - M1
370  T1 = X + Y
380  V1(C1) = T1
390  C1 = C1 + 1
400  RETURN
410  REM THIS IS THE DATA TO OUTPUT. 3E SEPARATES WORDS, - 1 IS END
420  DATA AA,A8,5E,46,61,3E,67,92,3E,9F,6F,80,44,6B,5E,46,61,3E, - 1
430  REM DATA IS "TODAY IS SATURDAY"
440  REM
```

Figure 6.14 *(continued)*

6.10 Summary

In this chapter, we have discussed voice synthesis using phonemes. We chose this technique because it allows you an unlimited vocabulary which is easy to generate. After we explained how phoneme speech works, we presented a block diagram and schematic of connecting the SC-01 chip to your Commodore 64. We closed the chapter by giving three complete software examples of how to use the SC-01 chip to generate speech.

You can use phoneme speech synthesis quite easily to give your system a voice. After a little experimentation, you will have your voice speaking when and what you desire.



THE ANALOG-VERSUS-DIGITAL DIFFERENCE AND TRANSDUCERS

7

In this chapter, our discussion will introduce two concepts: 1) the difference between analog and digital events, and 2) the transducer. If you are already familiar with these concepts, you may want to skip this chapter. However, anyone who wishes to learn about computer control must understand the difference between analog and digital events, and the importance of transducers. As we will see, external devices are mostly analog in nature. We need to convert some features before the devices can operate under digital control.

7.1 Analog Events

Briefly, an *analog* event is one whose output is variable. Let's look at a few examples of the analog concept that also prove that we live in an analog world.

The common mercury thermometer is an analog device because its output (the temperature reading) can have virtually any value. A typical thermometer scale is shown in Figure 7.1. The range of values is limited by the scale on the thermometer (97–103 degrees F), but within these limits there are an infinite range of possible readings. The temperature reading, like any analog output, can be anywhere between these limits.

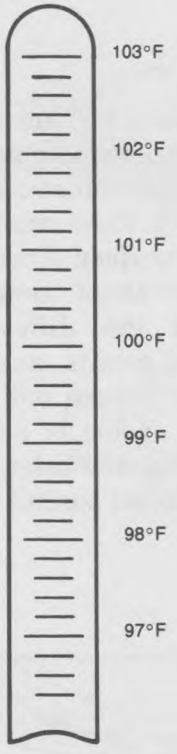


Figure 7.1: *This typical thermometer shows the temperature scale in degrees Fahrenheit. The reading can be anywhere on the scale, which has an infinite number of potential output values.*

You may be confused by our terms. The word *digital* has been applied to thermometers that display numbers (digits) and use digital electronics in their display circuitry. In that sense (which is how *digital* is commonly used), 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: full, almost full, a little hungry, very hungry, famished, or any number of different states in between. Because the range of sensations is apparently continuous, we consider hunger to be an analog event.

7.2 Digital Events

The outputs of *digital* events are not infinitely variable within a range, but are limited instead to a finite number of predefined states. Let's re-examine our examples of analog events and consider what they would be like as digital events. We will limit the number of discrete output values to two, because there are only two possible states in the binary logic used in digital electronics.

First, let's examine the thermometer. If we limit this device in the way just described, it would have only two temperature output readings, such as 70 degrees F and 80 degrees F. The scale would appear as in Figure 7.2. This thermometer would be quite useless to us, because temperature as an analog event cannot be measured on a digital scale. 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, limited to two possible output values: full or hungry. You may not think this is so bad, but imagine how you would feel if one second you were "stuffed," and the next you were "famished." Now imagine if this happens all day long. Fortunately, hunger is an analog event that varies in imperceptible stages.

We can also illustrate the difference between digital and analog phenomena by comparing a staircase and a ramp. The staircase represents digital phenomena, and the ramp represents

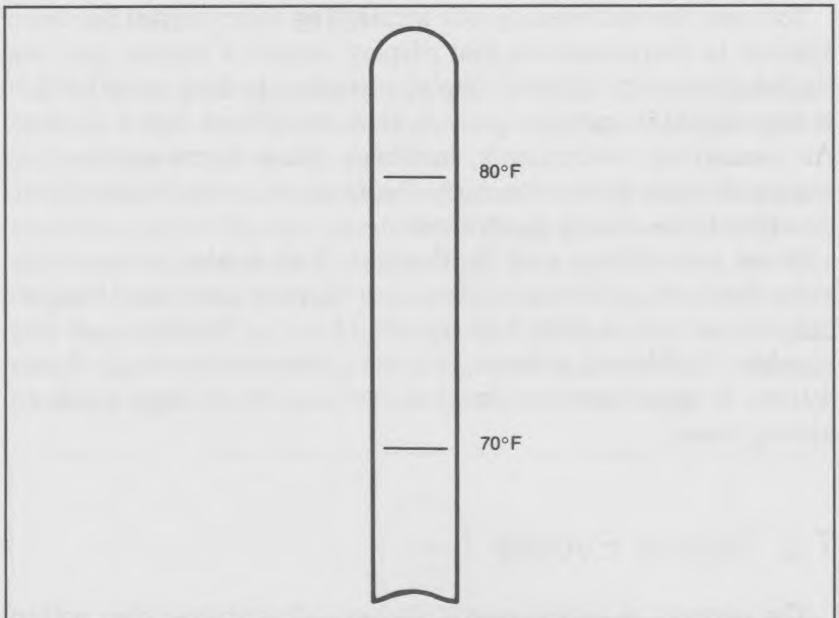


Figure 7.2: *This is a thermometer with a binary scale. The temperature reading can only be 70 degrees F or 80 degrees F.*

analog. On a staircase, we must ascend or descend by fixed, predefined steps. On a ramp we can move any amount we desire. The more steps there are in a given distance, the smaller the steps we proceed by, and the more accurately we can move to a particular level. If there were an infinite number of stairs in the staircase, we would have an analog event. We will use this analogy again in Chapter 9 (see Figure 9.7).

7.3 Common Digital Events

Although most events in the real world are analog, there are some common digital events. One common digital event occurs when you turn a light on and off. The light output only has two possible values: completely on or completely off. (Of course, if you have dimmer switches installed in your home, the light output will vary, because you are using an analog device called a rheostat.)

A home furnace fan in a forced-air heating system is also a digital event. The fan is either on—running at a fixed, constant RPM—or off—not spinning at all. With a thermostat (another analog device), we can vary the temperature at which the fan will turn on. However, it always will run at the same speed.

Another digital, binary device is the ON-OFF switch used in television sets and home computers. The switch has only two positions, on or off. Television tuners and fans with more than one speed are digital, but not binary. There is a finite number of possible states, but more than two.

7.4 Analog and Digital Electronics

Now that we have discussed the difference between analog and digital events using everyday examples, let's look at what these terms mean in relation to electronics and our Commodore 64. In electronics, analog and digital refer to electrical quantities, such as resistance, current, and 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, 12.10 V, 11.90 V, 12.60 V, or any value near 12 volts. As the battery ages, the voltage output may drop to 11.90 V, 10.88 V, or 10.73 V. To be digital, the voltage output of the battery would have to have a finite number of possible values. A battery would have to produce exactly 12.00 V, 12.50 V, or 12.75 V. A battery is an analog device because it does not meet these conditions.

Home computers are called digital because there are a finite number of possible voltage levels for the electrical information being processed within the system. When discussing input and output in Chapter 2 and 3, we saw 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. You can use a power supply (battery) of different voltages to run the digital system, but the digital circuits will output and input data only at two different voltages, as shown in Figure 7.3.

Remember that, in a digital electronic circuit, the two possible output levels are logical 1 and logical 0, and that digital circuits

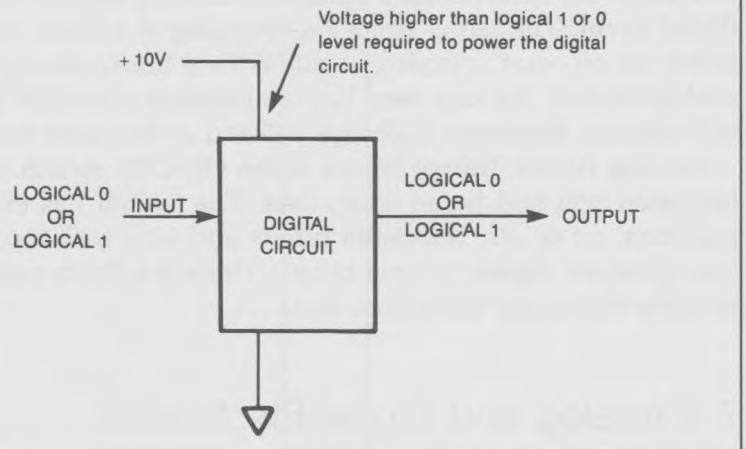


Figure 7.3: *This block diagram shows that non-logical-level voltages may be required to power the digital circuits. However, the digital circuit will only process input and output at the logical 0 or logical 1 voltage levels.*

are designed to follow the logic of the binary number system, which has only two digits, 0 and 1. Recall also that we normally work in the decimal number system, which has ten digits, 0–9. Home and personal computers are digital, binary machines that will operate on digital information only. Since most events that occur in nature are analog, there is a conflict if we wish to control them with a digital computer, as shown in Figure 7.4. Understanding the difference between analog and digital events will help you learn to resolve any incompatibility more easily.

7.5 Transducers

In later chapters, we will discuss how to bridge the gap between everyday analog events and the digital electronics used to control them. Before we explore this problem, we must explain another piece of the puzzle: the *transducer.*

We mentioned the transducer in Chapters 1 and 5 as a device that changes one form of energy to another. We discuss it here in the context of *analog-to-digital conversion,* or ADC.

Remember that many of the events we wish to control are analog and can be measured anywhere on a continuous scale of

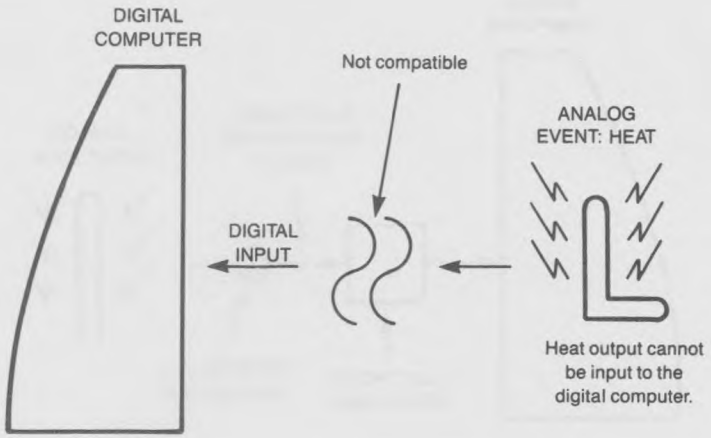


Figure 7.4: *An analog event, such as heat or temperature, is not directly compatible with a digital computer; it cannot be measured or monitored without some intermediate device.*

values. However, these values represent different physical characteristics. For example, temperature scales measure heat, a physical form of energy. Pressure is another form of energy that we sometimes will monitor with the digital computer. Remember that both pressure and temperature are analog events, because there is an infinite number of pressure outputs and temperature readings.

We must transform the various physical forms of energy into some electrical quantity that we can use in an electronic environment: voltage, current, resistance, capacitance, or inductance. After we change physical energy into electricity, we must further process it using electronic techniques. At this point, we are ready to input the form into a digital computer. The block diagram of Figure 7.5 shows the entire process.

The device that transforms a physical quantity into an electrical quantity is called a *transducer.* Transducers are classified according to the physical form of energy that they transform. For example, a pressure transducer transforms pressure into an electrical form, and a temperature transducer will transform temperature into an electrical quantity. However, you will find that the qualifying term—"pressure" or "temperature"—often is

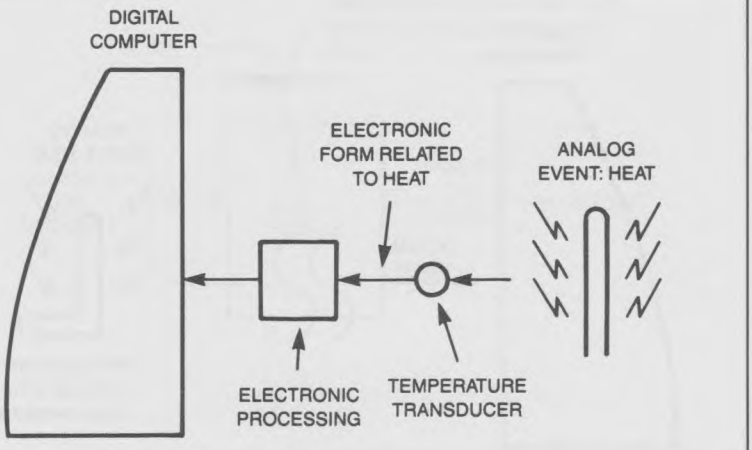


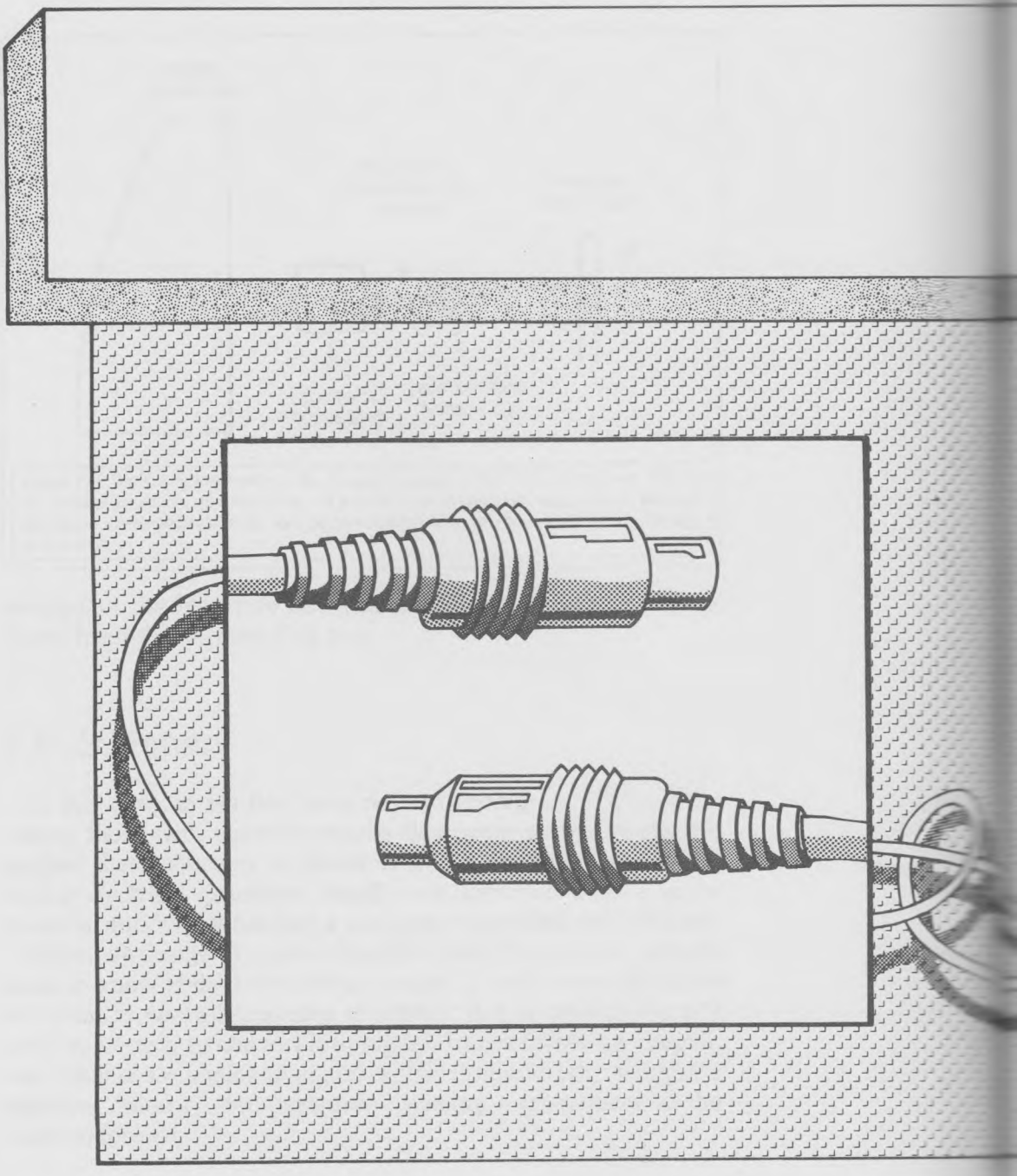
Figure 7.5: *The flow of information in the monitoring system is from the analog event, heat, to the physical transducer. The transducer will transform the temperature output into its electrical equivalent. The electrical signal can then be processed electronically and, finally, input to the digital computer.*

omitted in the literature and you must infer the type of transducer from the surrounding text.

7.6 Summary

In this chapter, we first have defined *analog* and *digital*, illustrating them with common events that occur in the world. We applied these concepts to electronics, and discussed digital and analog electrical quantities. Finally, we discussed what a transducer is, and how it fits into a computer-controlled environment.

When we design or use a computer-controlled system, we will need to know if the information output by each external device we want to control is analog or digital. If it is analog, we will need some way to convert it to digital. In the following chapter, we will discuss analog-to-digital conversion and how to input a physical quantity—temperature—by using a transducer to the Commodore 64.



ANALOG-TO-DIGITAL CONVERSION FOR THE COMMODORE 64

8

In this chapter, we will show how to use the Commodore 64 to monitor any external instrument, which outputs an analog voltage. We have designed the chapter to introduce beginners to the basic principles of analog-to-digital conversion. We will begin with a block diagram of the overall concept and then discuss how an analog-to-digital converter operates. Next, we will discuss the steps we will take when designing a complete system for monitoring temperature. After reading this chapter, you should have a clear understanding of how to monitor devices with a digital computer such as your Commodore 64.

8.1 Block Diagram of the Problem

We will start by showing a block diagram of the overall concept that we must use when monitoring an external device that gives analog output. In Figure 8.1, there are three major blocks shown:

1. Block 1 is the digital monitoring device, which will be our Commodore 64.
2. Block 2 transforms the analog voltage output from the external device into an equivalent digital word.
3. Block 3 is the external device itself, which is a type of transducer. It will output a voltage whose amplitude depends on the physical event being monitored.

In the last chapter, we defined an analog event as one that can be measured at an infinite number of possible output levels (within a given range). For example, pressure can have any value between a few pounds per square inch (psi) to many hundreds of psi. If we want to monitor an analog event with a digital computer, we need a transducer capable of accurately reporting the analog changes. If the pressure we are measuring increases or decreases, the voltage output from the pressure transducer must increase or decrease in the same proportion. (See Figure 8.2.)

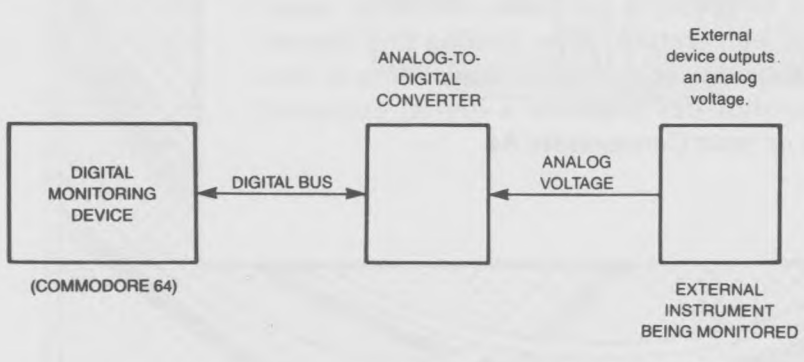


Figure 8.1: *This block diagram gives an overall view of the concepts involved in monitoring an analog voltage with a digital system.*

Let's compare an analog transducer to a digital transducer, which is like the switches we used for the home security system in Chapter 5. These transducers indicated whether the windows and doors were open or closed. Although a window or door may have been wide open or just cracked open, our transducer did not indicate the degree to which it was open. We were monitoring a digital event, and we only cared if the door or window was open or closed.

In this chapter, we will work with analog output transducers. This means that the transducer's output will vary in voltage anywhere from a low voltage, 0.0 volts, to a high voltage, approximately 5.0 volts. This voltage range is simply an example. Typically, the outputs of transducers vary from any

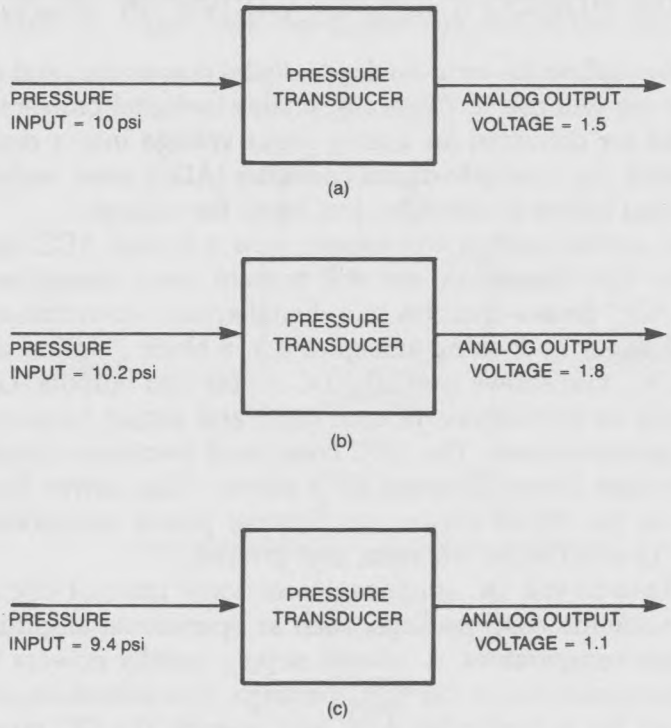


Figure 8.2 (a,b,c): *Each of these block diagrams shows the analog voltage output from a given pressure input. Note that as the pressure changes at the input of the transducer, the analog voltage must change in proportion.*

negative voltage to any positive voltage and depend on the type of transducer used. We will present an actual, practical example later in this chapter.

Let's assume here that an analog transducer will output a certain range of voltage. This analog voltage will be input to the block labeled "analog-to-digital converter" in Figure 8.1. The digital output of the analog-to-digital converter is then input to the Commodore 64 block.

The Commodore 64 will input digital information being output from the analog-to-digital converter. As we will see, this digital information equals the analog voltage that is input to the analog-to-digital converter.

8.2 The Analog-to-Digital Converter (ADC)

Let's now define the term *analog-to-digital conversion,* and discuss how we will use it. Generally, analog-to-digital conversion transforms (or converts) an analog input voltage into a digital output word. An *analog-to-digital converter* (ADC) must perform this function before a computer can input the voltage.

The rest of this section will explain how a typical ADC operates. After this discussion, we will present some examples of how the ADC device operates in a digital-system environment.

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

The ±12-to-15-volt DC supplies power some internal circuits located inside the ADC package, such as operational amplifiers and voltage comparators. A +5-volt supply usually powers the digital electronics inside the ADC package. The manufacturer's data sheets for a particular ADC will specify the DC power required for proper electrical operation.

Figure 8.3 also shows the input line labeled "Analog Input Voltage." The ADC transforms analog voltage into its equivalent

digital word. ADCs have specific ranges of analog voltages they can accept as input. Typical ranges are 0 to +5 volts, −5 to +5 volts, −10 to +10 volts, and 0 to +10 volts. These are just a few of the many ranges that are available. The manufacturer's data sheet will give exact specifications for the analog input voltage range of a particular ADC.

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

We will examine the digital output lines of Figure 8.3 next. The number of digital output lines on an ADC will vary. Typical numbers are 6, 8, 10, 12, and 16. In general, the greater the number of output lines, the greater the cost of the ADC.

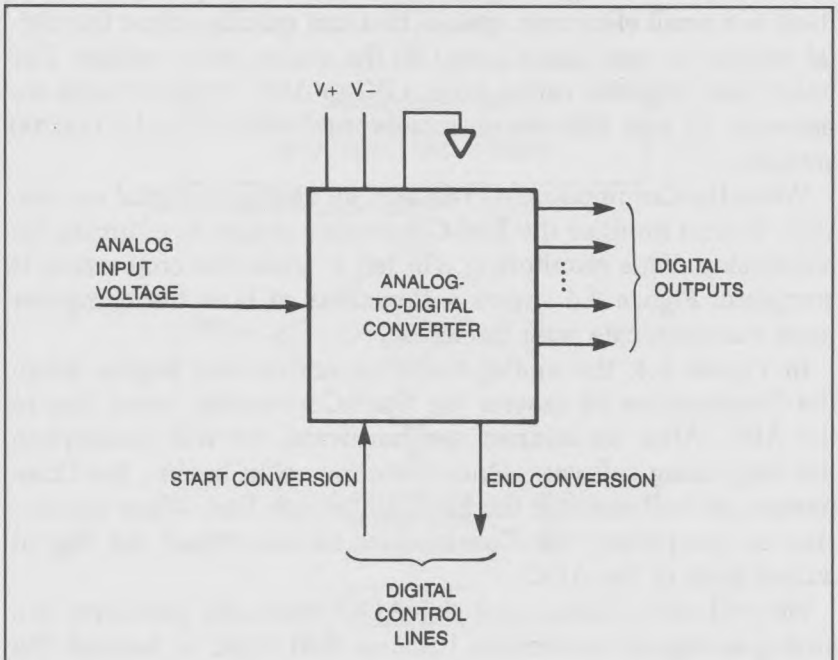


Figure 8.3: *This is a block diagram of a typical analog-to-digital converter.*

The digital output lines are set to a logical 1 or a logical 0, as determined by the analog input voltage. Later we will show you how to calculate which digital outputs are a logical 1 and which are a logical 0. For now, think of the digital output lines as the physical connections to the digital monitoring system, the Commodore 64.

In Figure 8.3, we see two digital control lines, labeled "Start Conversion" and "End Conversion." The Start-Conversion input line is a digital signal from the Commodore 64 to the ADC to start converting the analog input to a digital output word.

When the Commodore 64 is ready to accept the digital output word from the ADC, it asserts the Start-Conversion line to the ADC, by setting the input signal to the logical state that will start the action. Since the logical state could be a 1 or a 0, we use the term "assert" as a generalization.

The End-Conversion output line informs the digital monitoring device that an analog-to-digital conversion is complete. The ADC itself is a small electronic system that can quickly adjust the digital outputs to new values based on the analog input voltage. The exact time required varies from ADC to ADC. Typical times are between 10 and 200 microseconds, or 0.000010 and 0.000200 seconds.

When the Commodore 64 requests an analog-to-digital conversion, it must monitor the End-Conversion output line during the conversion. This monitoring will tell it when the conversion is complete. Figure 8.4 shows a flowchart of how the computer must communicate with the ADC.

In Figure 8.4, the analog-to-digital conversion begins when the Commodore 64 asserts the Start-Conversion input line to the ADC. After we connect the hardware, we will accomplish this step using software. Once the conversion begins, the Commodore 64 will monitor the End-Conversion line. When conversion is completed, the Commodore 64 will input the digital output lines of the ADC.

We will not discuss how the ADC internally performs the analog-to-digital conversion because that topic is beyond the scope of this text. We present the ADC as a "black box" and show how to make it perform its prescribed function. All of the electronics in the ADC you will use are usually out of the user's

reach. Understanding the ADC inputs and outputs is our first step toward using the ADC with a Commodore 64.

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 levels of the analog input. We will use a unipolar ADC with a voltage input range of 0.0 volts to +10.0 volts. There will be eight digital-output lines. Figure 8.3 gave a block diagram of this type of device.

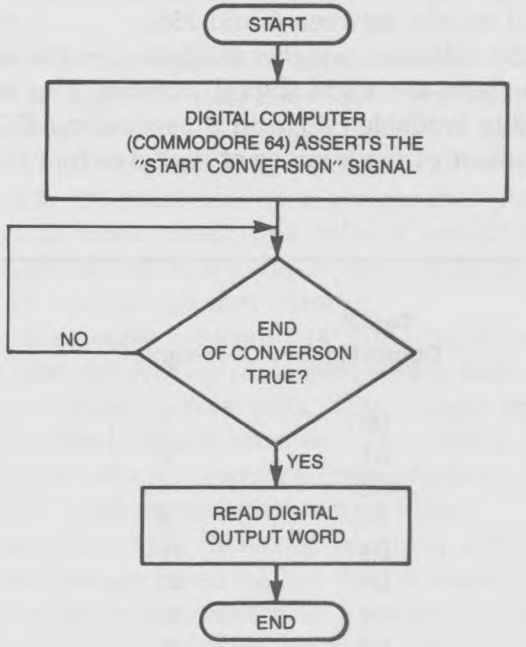


Figure 8.4: *This flowchart shows the sequence of events required for electrical communication with an ADC.*

We will label the digital outputs D0–D7, and will assign the same numerical weights that we gave to bit positions in data bytes for other examples. Figure 8.5 gives a table of the digital output lines and their corresponding weights.

The minimum weight at the digital outputs will occur when all outputs equal a logical 0, and the maximum weight will occur when all are a logical 1. The minimum and maximum weights are 0 and 255, respectively.

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

0.0 volts input = weight 0 output

+10.0 volts input = weight 255 output

All input voltage 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 (W) with a particular ADC equals 2 raised to the power of the number of digital output lines (L):

$$W = 2^L$$

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

Figure 8.5: *The output lines of the ADC have the same numerical weights as the Commodore 64 data bus lines.*

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

Since we know how many unique digital weights are available, we can divide that range into equal increments. Each increment is called the *resolution of the ADC.* In our example, we have 255 available digital weights. (One weight must be subtracted because of the 0 at the start.) Therefore, we can divide the range between 0 and +10 volts into 255 equal pieces by the following equation:

$$\frac{\text{voltage range}}{255} = 0.03921569$$

We can use this information to generate a list of input voltages and their corresponding digital output weights, which is shown in Figure 8.6.

In Figure 8.6, the equivalent input voltage increases by discrete steps of approximately 0.04 volts. For example, suppose an input voltage to the ADC equals 1.50 volts. However, there is no digital output weight exactly equal to 1.50 volts. From the list of Figure 8.6, we must choose a weight that yields a voltage closest to 1.50 volts: either 1.49 volts (a weight of 38) or 1.53 volts (a weight of 39). Since the output weight for 1.50 volts is closest to 38, we can use that number.

For analog voltages between 0.00 and +10.00 volts, we must remember that our ADC gives digital values with an error margin of approximately +0.04 volts. This margin depends on the number of digital output lines and the analog voltage input range. If +0.04 volts is too great a range, then we must choose a different ADC with more digital output lines.

When using the ADC, it would be much easier if we could calculate the voltage based on the digital output word, instead of looking up the voltage each time a conversion took place. We can use the following equation for these calculations: the analog voltage equals the digital output weight multiplied by the resolution of the ADC.

$$\text{analog voltage} = \text{digital weight} \times (10/255)$$

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
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

D = DIGITAL WEIGHT
V = VOLTAGE

Figure 8.6: *This is a list of digital output weights and the corresponding analog input voltages they represent.*

Digital Weights and Voltages

D	V	D	V	D	V	D	V	D	V
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

Figure 8.6 *(continued)*

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

$$125 \times (10/255) = 4.9 \text{ volts}$$

Remember that this voltage actually may be slightly less than 4.94 volts or equal to 4.9 volts, because of the resolution of the ADC.

In its present form, our equation depends on the input voltage range and the number of digital output lines on the ADC. We must modify these values for the equation to fit a particular application.

8.4 Connecting the ADC to the Commodore 64

Figure 8.7 shows a complete schematic for connecting an actual ADC—Analog Devices' model AD570™—to the Commodore 64. Our discussion is for readers who want to know how the hardware of the interface operates. The AD570 is very similar

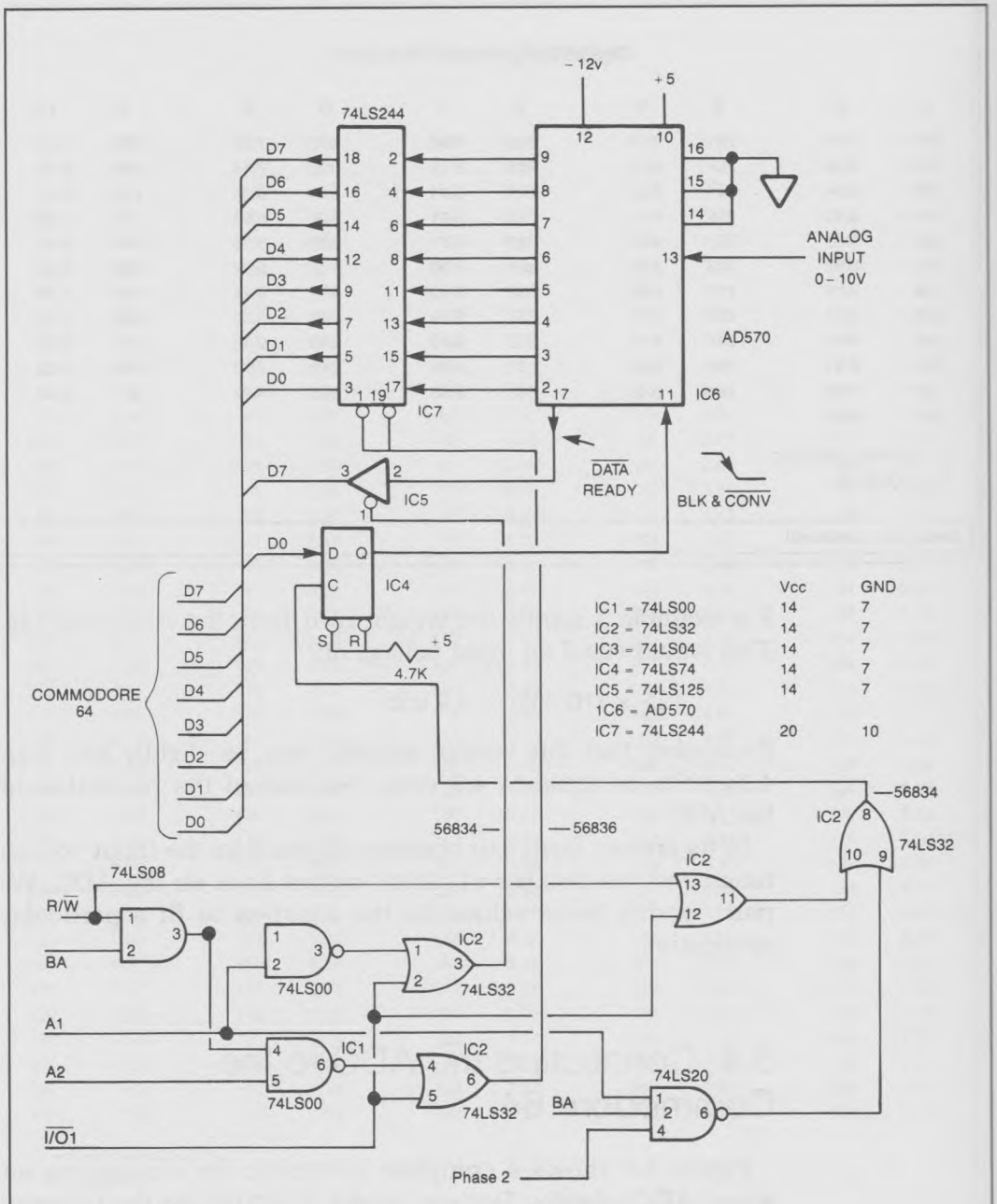


Figure 8.7: *This is the complete schematic for an ADC interface to the Commodore 64.*

in its operating characteristics to the general example in Figure 8.3. Appendix C includes a data sheet for this device.

The analog-to-digital converter is labeled IC6 in Figure 8.7. The device is powered by +12-volt, −5-volt, and ground lines. These power supply voltages must be input from an external source since the Commodore 64 only outputs +5 volts on the expansion connector. The ground of the external power supply will be connected to the ground line of the Commodore 64.

On the right side of the ADC (IC6) is the analog input voltage that we need to convert. 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 74LS244, labeled IC7. We discussed this type of buffer in Chapters 4 and 5.

The outputs of the buffer (IC7) are connected directly to the Commodore 64 data bus lines, D0–D7. When the computer uses the PEEK statement to read the data at the ADC outputs, the buffer will become enabled.

To start the analog-to-digital conversion, input pin 11 of IC6 must be set to a logical 1, then reset to a logical 0. This is the Start-Conversion input signal for this particular ADC. Resetting input pin 11 to a logical 0 starts the analog-to-digital conversion. This is shown by the waveform symbol next to the signal line to input pin 11.

Input pin 11 of the ADC is connected to output pin Q of a 74LS74 D flip-flop, IC4. This flip-flop acts as a single-bit latch. The data input to the flip-flop is connected to the D0 data line. Whenever you wish to set the Q output to a logical 1, the D0 line will go to a logical 1, while the flip-flop will be clocked.

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

```
POKE address,1
```

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

```
POKE address,0
```

will set the Q output of the D flip-flop to a logical 0. We now can control an individual hardware line through a software instruction: specifically, we can set input pin 11 of IC6 to a logical 1 or

a logical 0, using software control. Later we will discuss how to compute the address of the two POKE statements.

To know when the analog-to-digital conversion is complete, the Commodore 64 monitors output pin 17 of IC6. This pin is the End-Conversion signal for the ADC. In Figure 8.7, output pin 17 of IC6 is connected to input pin 2 of IC5, a 74LS125 tri-state buffer. Appendix C gives the data sheet for the 74LS125. 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.

When the Commodore 64 executes a PEEK statement to the correct address, the 74LS125 will be enabled. This will enable the logical state of output pin 17 of IC6 onto the Commodore 64 data bus. The computer will then input the logical level into a BASIC program. The function of the 74LS125 is similar to that of the 74LS244.

We can use PEEK or POKE statements to monitor the End-Conversion signal and to logically set input pin 11 of IC6. A major part of both statements is the address. We will now discuss how to relate the address of the PEEK or POKE statement to the schematic diagram given in Figure 8.7.

To read the data from the ADC, we must enable the outputs of IC7 of Figure 8.7. This means pins 1 and 19 of IC7 must be held at a logical 0. These pins are connected to output pin 6 of IC2. Output pin 6 will be a logical 0 when the $\overline{I/O1}$ line and pin 6 of IC1 are a logical 0. Pin 6 of IC1 will be a logical 0 when the R/$\overline{W}$ line, BA, and address line A2 are a logical 1. Therefore, the logical conditions that will enable IC7 pins 1 and 19 are:

$\overline{I/O1}$ = logical 0
R/$\overline{W}$ = logical 1 (PEEK)
BA = logical 1
A2 = logical 1 (I/O1 + 4)

The I/O circuit has a PEEK address equal to 56832 decimal. The PEEK statement we will use to read the data from the ADC will be:

```
A = PEEK (56832 + 4)
```

To read the logical conditions of output pin 17 of IC6, address line A1 (instead of address line A2) must be a logical 1. Our

PEEK statement to read this line will be:

```
A = PEEK (56832 + 2)
```

The last signal we must control is the Q output of the D flip-flop, IC4. To write a logical 1 or a logical 0 to the D flip-flop, we use a POKE statement. If we follow the clock (C) input line from the D flip-flop, we can see that it is connected to output pin 8 of IC2. Whenever pin 8 of IC2 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 section of Figure 8.7. IC2 pin 9 will become a logical 0 when A1, BA, and the Phase 2 clock are equal to a logical 1. Pin 10 of IC2 will be a logical 0 when $\overline{I/O1}$ and R/$\overline{W}$ are both equal to a logical 0. These inputs will set pin 8 of IC2 to a logical 0. Remember that pin 8 is connected to the clock input of the D flip-flop.

All of the preceding conditions will be true when we use the following POKE statement:

```
POKE 56832 + 2,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 it to a logical 0.

8.5 Software for Analog-to-Digital Conversion

Now that we have the ADC connected to the Commodore 64, let's look at the BASIC program we'll need to input a digital value from the ADC. Figure 8.8 shows a flowchart for the steps of this program. For our discussion, we assume that the ADC I/O device is located at address 56832 in the Commodore 64. Let's look at each step:

Step 1. Our first step is to write a logical 1 to the flip-flop on the ADC I/O circuit. We can use a POKE statement that will set address line A1 equal to a logical 1.

```
POKE 56832 + 2,1
```

Adding 2 to the I/O address will set address line A1 to a logical 1.

Step 2. Here, we reset the Q output of the flip-flop to a logical 0. We can use the same POKE statement as in step 1, except that the data is now a 0 instead of a 1:

```
POKE 56832 + 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. We can do this if we use the following PEEK statement:

```
R1 = PEEK(56832 + 2)
```

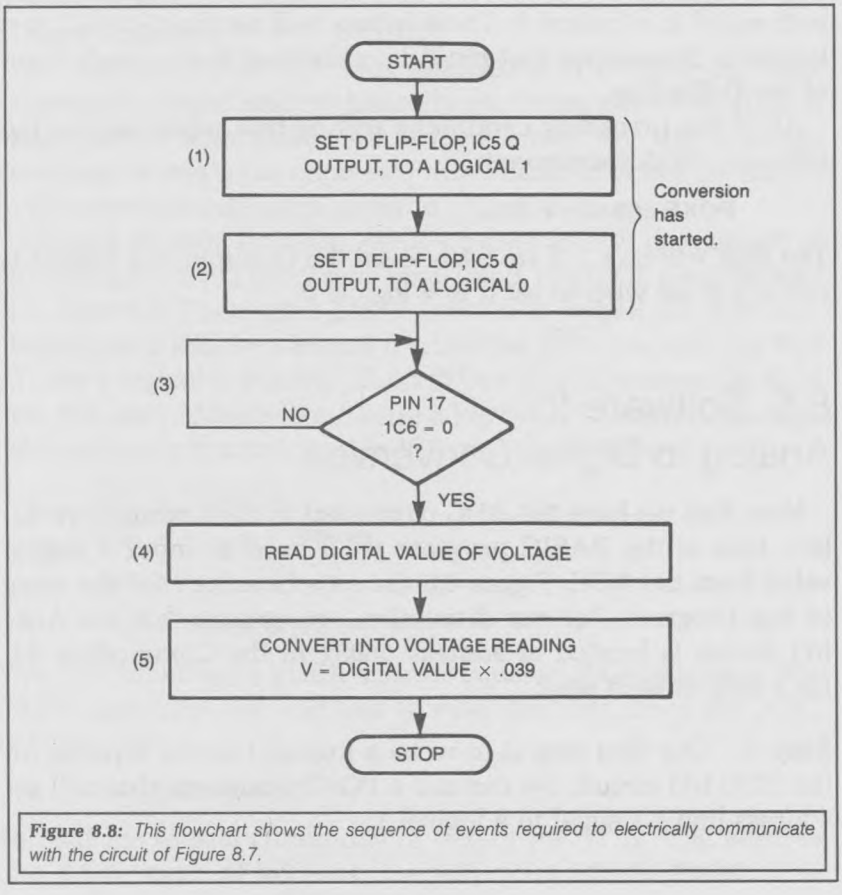


Figure 8.8: *This flowchart shows the sequence of events required to electrically communicate with the circuit of Figure 8.7.*

When we read R1, its weight will be the total of all of 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 execution of the PEEK statement. The other input lines are either a logical 1 or a logical 0, and may not show the same value during each PEEK statement. The actual logical level depends on many factors, most of which are out of our control.

Therefore, we want to logically ignore data lines D0–D6 during the PEEK statement. Although we could do this with hardware, it is much easier to do with software. In the system software, we can ignore data lines D0–D6 by testing for a value greater than 127. (If R1 is at least 128, we know that D7 must have been a logical 1.) We can use the statement:

IF R1>127 **THEN GOTO** (line number of PEEK statement above)

After this statement is executed, the value of R1 will be greater than 127 if D7 equals 1, or less than 127 if D7 equals 0. Let's examine why this happens. When the computer used a PEEK statement to read R1, the computer had eight data input lines. However, only one line, D7, is important. The input data will logically appear as either:

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, D7 was either a logical 1 or a logical 0. We marked the other data lines with an X, because we do not care what each logical state is at this time. However, the logical state of each line will affect the overall computed weight of the input byte.

By testing whether R1 is greater than 127 or not, we can ignore all data lines D0–D6. If they were all a logical 1 and D7 was a logical 0, R1 would equal 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 incomplete and the program must loop until it is complete.

Step 4. If the conversion is complete, the value of R1 will be less than 127. The computer can now read the digital outputs of the ADC, if we use the following statement:

```
V1 = PEEK (56832 + 4)
```

The variable V1 will stand for the combined weight of the input data lines.

Step 5. Our last step is to compute the analog voltage that was input. We can do this by multiplying the weight of V1 by the constant (10/255) which is a procedure we discussed earlier. Our statements are:

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

Figure 8.9 shows all of the steps of the BASIC program together. (We assume that the starting line number for this routine is 100.) We used the RETURN statement in line 170 on the assumption that all statements 100–170 would be in a BASIC subroutine. This gives us access to the ADC at any time during a program by executing a GOSUB 100 statement.

8.6 A System for Temperature Measurement

Let's suppose that we had a transducer that produces an analog voltage proportional to the temperature we are measuring.

```
100  POKE 56834,1
110  POKE 56834,0
120  R1 = PEEK (56834)
130  IF R1 = 1 THEN 120
140  V1 = PEEK (56836)
150  V2 = V1 * (10/255)
160  PRINT "WEIGHT = ";V1;" VOLTAGE = ";V2
170  RETURN
```

Figure 8.9: *This BASIC program for an analog-to-digital conversion implements the flowchart shown in Figure 8.8.*

There are off-the-shelf temperature transducers available that will operate like this. For our example, our temperature transducer will measure temperature from 0 to 150 degrees Centigrade. The analog output will be from 0 to 3 volts.

We can relate the output voltage to the temperature by this equation:

$$\text{Temperature in degrees C} = (150/3) \times V_{out}$$

We can use this equation to connect the temperature sensor output to the ADC and measure the temperature with the Commodore 64. Figure 8.10 shows the block diagram of the complete hardware system. (We have already discussed each major block of Figure 8.10.) This diagram lets us write the complete BASIC program in Figure 8.11, that will accept the data from the ADC, and calculate and print out the temperature in degrees Centigrade.

As with other programs we listed earlier, we designed this program to instruct you. We did not attempt to minimize the number of BASIC statements and we documented each step with REM lines. You can call the subroutine at line 200 whenever you need a temperature.

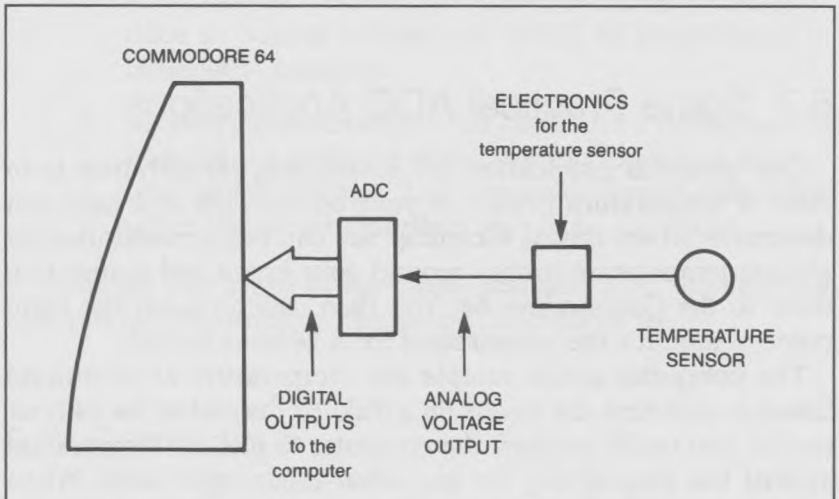


Figure 8.10: *This is a block diagram of the complete system required to automatically measure temperature with the Commodore 64.*

```
 10 GOSUB 200
 20 PRINT "TEMPERATURE IN DEGREES CENTIGRADE = ";T1
 30 STOP
 40 REM THE SUBROUTINE FOR TEMPERATURE MEASURING
200 POKE 56834,1
210 POKE 56834,0
215 REM THE ABOVE WILL START THE ADC CONVERSION
220 R1 = PEEK(56834)
230 R1 = R1 + 1
240 IF R1 = 1 THEN 220
245 REM THE ABOVE WILL WAIT UNTIL CONVERSION IS COMPLETE
250 V1 = PEEK(56836)
255 REM THE ABOVE WILL READ THE DIGITAL WEIGHT
260 V2 = V1 * (10/255)
265 REM THE ABOVE WILL CALCULATE THE VOLTAGE INPUT
270 T1 = (150/3)*V2
280 RETURN
290 REM THE ABOVE CALCULATES THE TEMP IN DEGREES CENTIGRADE
```

Figure 8.11: *This program will accept an input voltage from an ADC, and calculate and print out the corresponding temperature in degrees Centigrade.*

8.7 Some Practical ADC Applications

One practical application for monitoring temperature is to chart a temperature profile of your home. This will help you determine where heat is escaping. You can begin monitoring by placing temperature probes around your home and connecting them to the Commodore 64. You then can program the computer to monitor the temperature for a 24-hour period.

The computer could sample the temperature at 10-minute intervals and store the results on a disk. At the end of the 24-hour period, you could program the computer to plot the temperature against the time of day (or any other meaningful axis). When you have tested all of the important rooms and areas in your home, you could then make whatever adjustments are necessary to maintain the temperature you want.

You could also use a temperature probe to monitor the difference between the temperature outside your home and the temperature inside. If the inside temperature were cooler than the outside temperature, you then could program the computer to use the warmer, outside air to heat up the home instead of using the furnace.

There are many other practical ADC applications. Some of them are:

1. Sound detection. You could convert sound into an analog voltage by using a microphone (another kind of transducer). If the sound reached a certain level, you could program the computer to take some action. This could be part of a security system that detected the noise of an intruder.

2. Wind direction. You could construct a transducer to produce an analog voltage directly proportional to wind direction; for example, 5.00 v = north, 3.75 v = east, 2.50 v = south, and 1.25 v = west. The transducer would scale all other compass directions accordingly.

3. A barometer. You could connect a transducer to produce an analog voltage that would be proportional to barometric pressure.

4. Moisture measurement. You could use a transducer to monitor the moisture of the soil to produce an equivalent analog voltage. The computer would determine when the sprinkler system should be turned on.

These are just a few of the many practical ADC applications that you can accomplish with your Commodore 64.

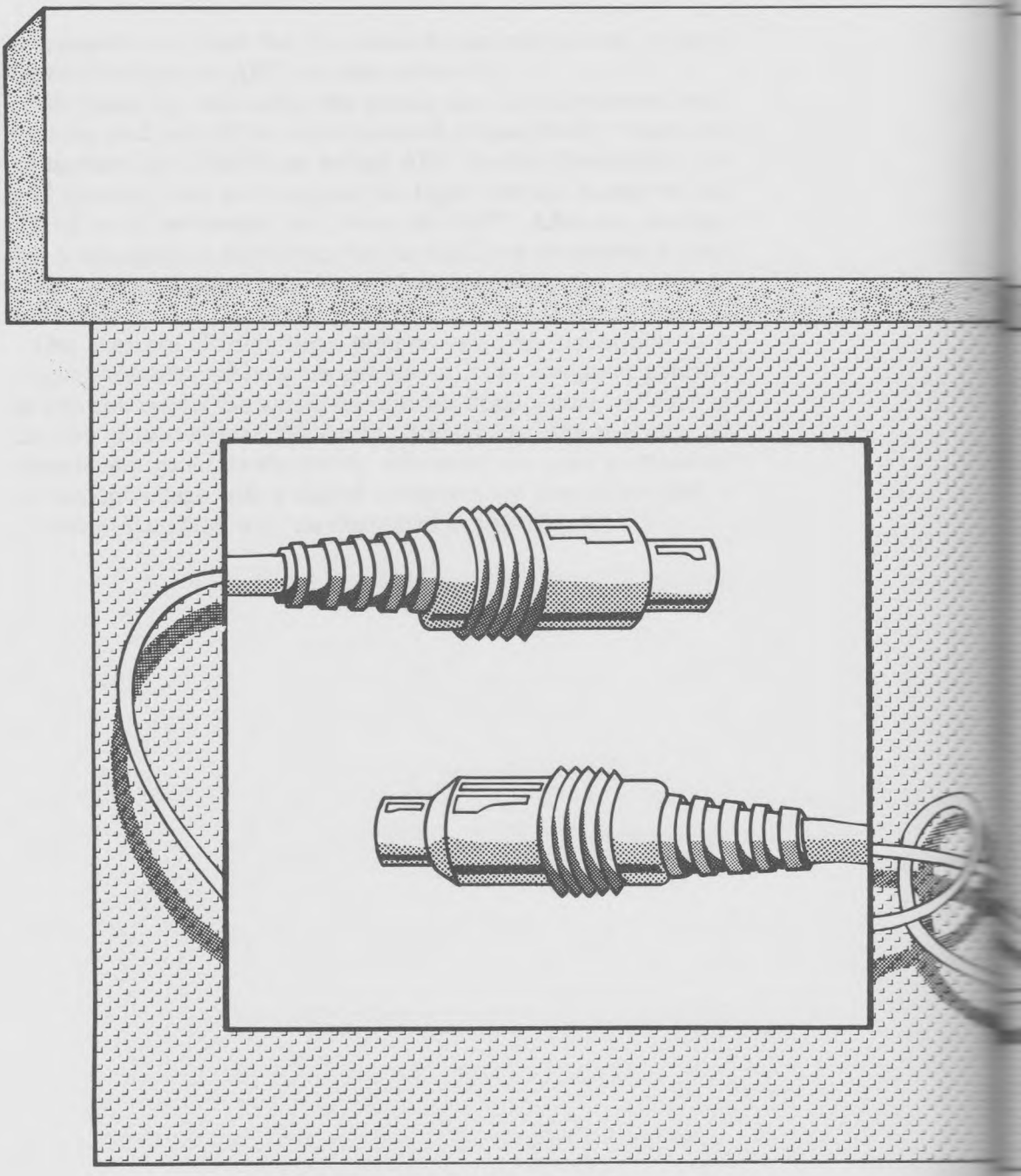
8.8 Summary

In this chapter, we focused on how to perform analog-to-digital conversion using the Commodore 64. Although we covered almost all aspects of ADC—from concept to connecting

a converter—we kept the discussion to the user's level, without explaining how an ADC operates internally.

We began by explaining the reason for analog-to-digital conversion and how it is accomplished conceptually. Next, we connected the AD570, an actual ADC, to the Commodore 64 and showed how to calculate the input voltage based on the digital word or weight read from the ADC. After we connected a temperature transducer to the ADC, we presented a complete temperature-monitoring system including the necessary software.

Our example of inputting a temperature was meant only as a single illustration of how we can use ADC in computer control, as you can see by the examples of other applications we gave at the end of the chapter. Whatever a transducer may measure, all transducers generate electricity; whenever we need to measure an analog voltage with a digital computer, we can accomplish it in almost the same way we described in this chapter.



DIGITAL-TO-ANALOG CONVERSION FOR THE COMMODORE 64

9

Certain peripheral devices that we can control with a home computer require an analog input voltage in order to operate. (We discussed what is meant by an analog voltage in Chapter 7.) This means we have an interfacing problem because a home computer system is completely digital. We need a means of producing an analog voltage from a digital source or controller. Such a process is called *digital-to-analog conversion*, or DAC. Figure 9.1 shows a block diagram of this concept.

A direct-current motor is one type of external device that might require an analog input voltage. Most DC motors rotate faster when you apply a higher voltage to the input. Figure 9.2 shows a block diagram of a DC motor with its analog input.

There are many other applications using devices that require an analog voltage for control. For instance, communication and recording systems use DAC and ADC at their input and output. Electronic-music and waveform-generation systems, which are growing in popularity, also use digital-to-analog conversion. In short, we need DAC to use digital electronics to control any device whose output is variable.

This chapter will show how you can control and set an analog output voltage from a digital source, such as the Commodore 64. We will first cover the basics of digital-to-analog, then design the hardware and software necessary for the Commodore 64 to control an analog output voltage from an output connector. The circuits shown are inexpensive and available from many suppliers.

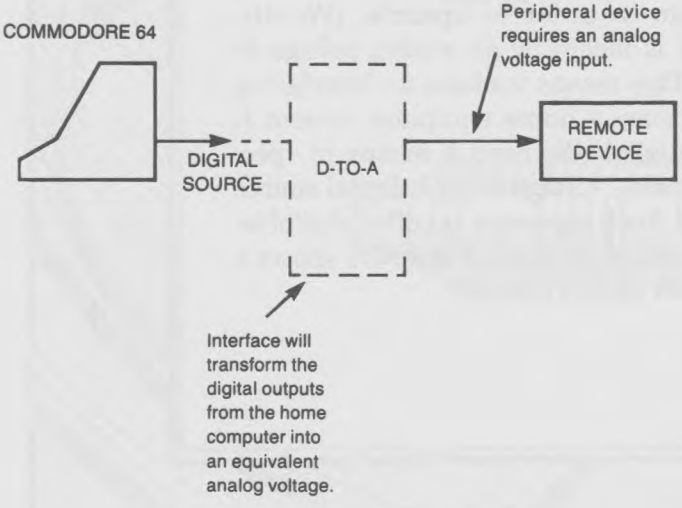


Figure 9.1: *This block diagram shows the interfacing problem that occurs when a completely digital source needs to control a peripheral device that requires an analog voltage input. A DAC will solve this interfacing problem.*

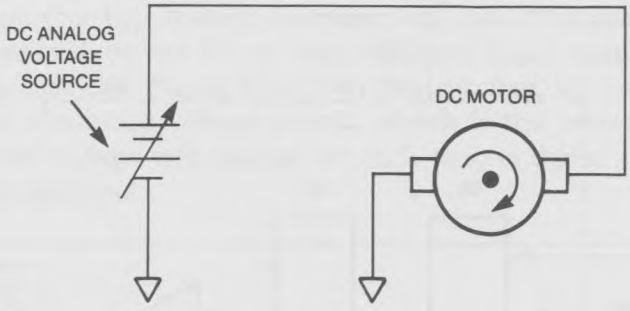


Figure 9.2: *This diagram shows a DC motor being controlled by an analog voltage. When the voltage increases, the motor will increase in speed. This is only one type of peripheral device that can be controlled by an analog input voltage.*

9.1 What is Digital-to-Analog Conversion?

Let's examine the overall concept of digital-to-analog conversion. The introduction gave a very informal definition that we will now bring into sharper focus.

Figure 9.3 shows a general block diagram of a digital-to-analog converter. Block B1 is the electrical interface to the digital source that will input data to the converter. Our source here will be the digital output lines from the Commodore 64.

Block B1 will control block B2, the precision scaling network. The scaling network electrically determines how much of the reference voltage or current will appear at the converter output. For example, suppose the reference voltage equals 8.00 volts. The scaling network is capable of setting the output voltage to an exact portion of the reference, such as 2.04 volts. The reference could also be a smaller voltage than the output device requires, such as 1.50 volts. In this case, the scaling network would select a portion of the 1.50 volts, which would be amplified before being output from the DAC.

Digital inputs from the computer will 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 equal

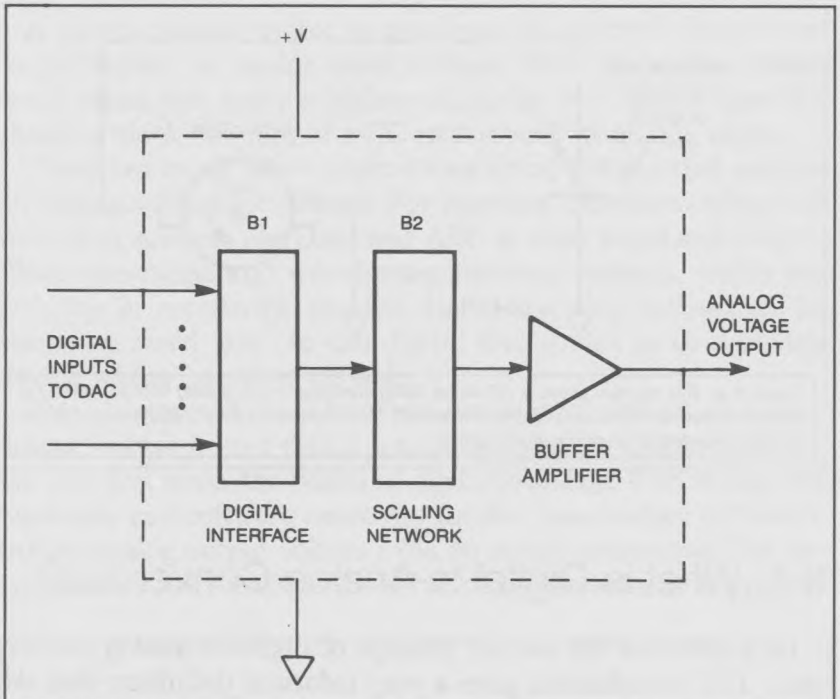


Figure 9.3: *This is a block diagram of a general digital-to-analog converter. Most DACs will have a block diagram similar to this one.*

4.00 volts, exactly half of the 8.00-volt reference. We cannot give a general definition of what voltage will be at the output as a function of the digital inputs, because every digital-to-analog converter has different specifications.

Let's look at an example that shows how to set the analog output voltage as a function of the digital inputs. Suppose we have a DAC that will output 10.00 volts when all digital inputs are a logical 1; that is, its reference voltage is 10.00 volts. (Remember that every DAC has a unique set of specifications.) Let's assume that the DAC will output 0.00 volts when all 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*, because the output voltage will swing toward one electrical pole only. In this case, the swing was from 0.00 volts to +10.00 volts. Unipolar also describes an analog-to-digital converter with the same voltage characteristics.

Next we need to know the number of digital inputs that the DAC has. Our hypothetical converter will have 10 digital inputs. This means there are 2^{10}, or 1024, different input combinations we can use. (See Figure 9.5.) With this number, we can calculate the minimum voltage swing, which is the increment by which the voltage will change for each unique digital combinations on the input.

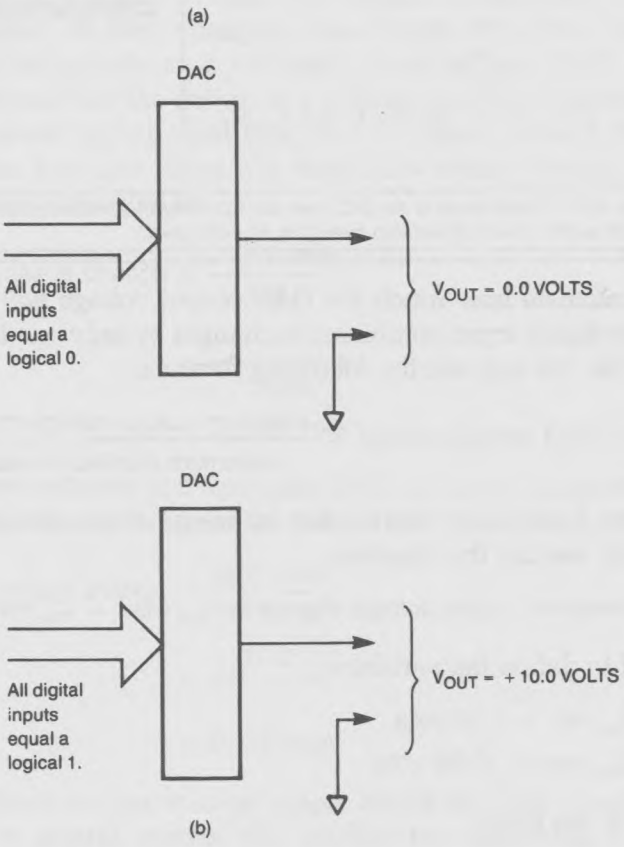


Figure 9.4 (a): *When all digital inputs to the DAC are a logical 0, the output voltage will equal 0.00 volts.*

(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 DACs output voltage swing.*

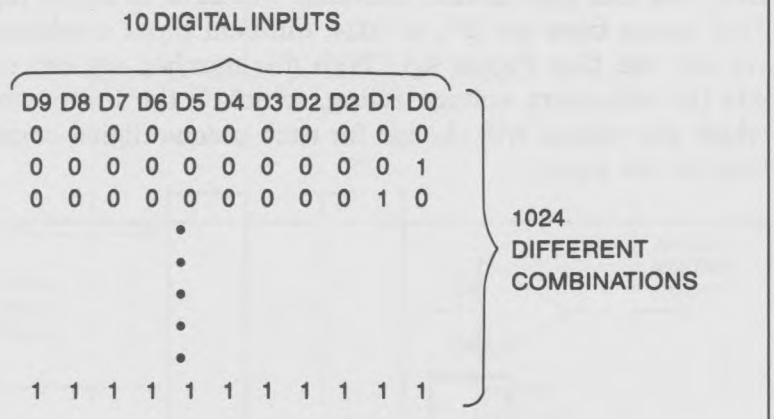


Figure 9.5: *With 10 digital inputs to the DAC, there are 1024 different combinations that may be applied. The number 1024 is derived from raising 2 to the tenth power.*

Let's 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:

$$\text{Minimum output voltage change} = \frac{\text{minimum output voltage change}}{\text{maximum number of states}}$$

Before we know what the answer is, we have to calculate each part. First, we use the equation:

$$\text{Maximum output voltage change} = V_{out}\ \text{max} - V_{out}\ \text{min}$$

We need to define the variables:

$$V_{out}\ \text{min} = 0.00\ \text{volts}$$
$$V_{out}\ \text{max} = 10.00\ \text{volts}$$

Therefore, we have:

$$\text{Maximum output voltage change} = 10.0 - 0.0$$
$$= 10.0$$

Remember that the maximum number of unique digital combinations with 10 inputs is 1024. Since one of these combinations

is used to output 0.0 volts, we have actually 1023 states that produce a voltage.

Let's substitute what we now know into the original equation:

$$\text{Minimum voltage swing} = \frac{10.0 \text{ volts}}{1023}$$

$$= 0.009775 \text{ volts}$$

or approximately 10 millivolts

This means the output voltage will change 10 millivolts for each digital input bit that changes. (See Figure 9.6.) Our DAC is described technically as a unipolar, 10-bit, voltage DAC. ("Voltage" indicates that the output is a voltage and not a current.)

Suppose our hypothetical DAC had 12 inputs instead of 10. If we use the formulas given, the minimum voltage change at the output would be equal to:

$$\text{Minimum voltage change} = \frac{10.0 \text{ volts}}{(4096 - 1)}$$

$$= \frac{10}{4095}$$

$$= 0.0024 \text{ volts}$$

Let's now suppose that this same DAC had only six inputs. The minimum voltage change at the output pin would be:

$$\text{Minimum voltage change} = \frac{10.0 \text{ volts}}{(64 - 1)}$$

$$= \frac{10.0}{63}$$

$$= 0.158 \text{ volts}$$

For a given output voltage range, notice that 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," because the DAC can resolve a smaller increment of voltage. This means we can come closer to obtaining the "smooth" staircase waveform at the top of Figure 9.7.

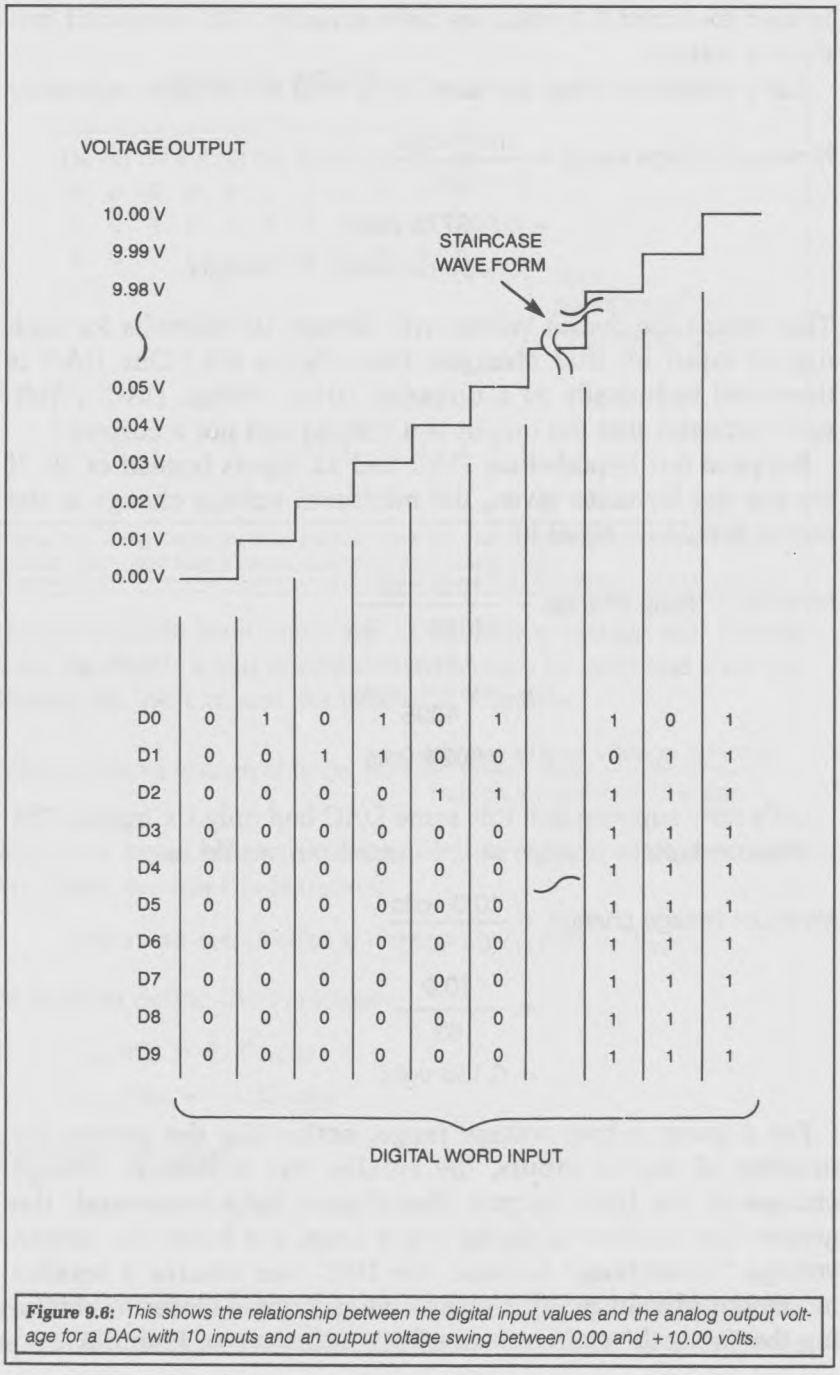


Figure 9.6: *This shows the relationship between the digital input values and the analog output voltage for a DAC with 10 inputs and an output voltage swing between 0.00 and +10.00 volts.*

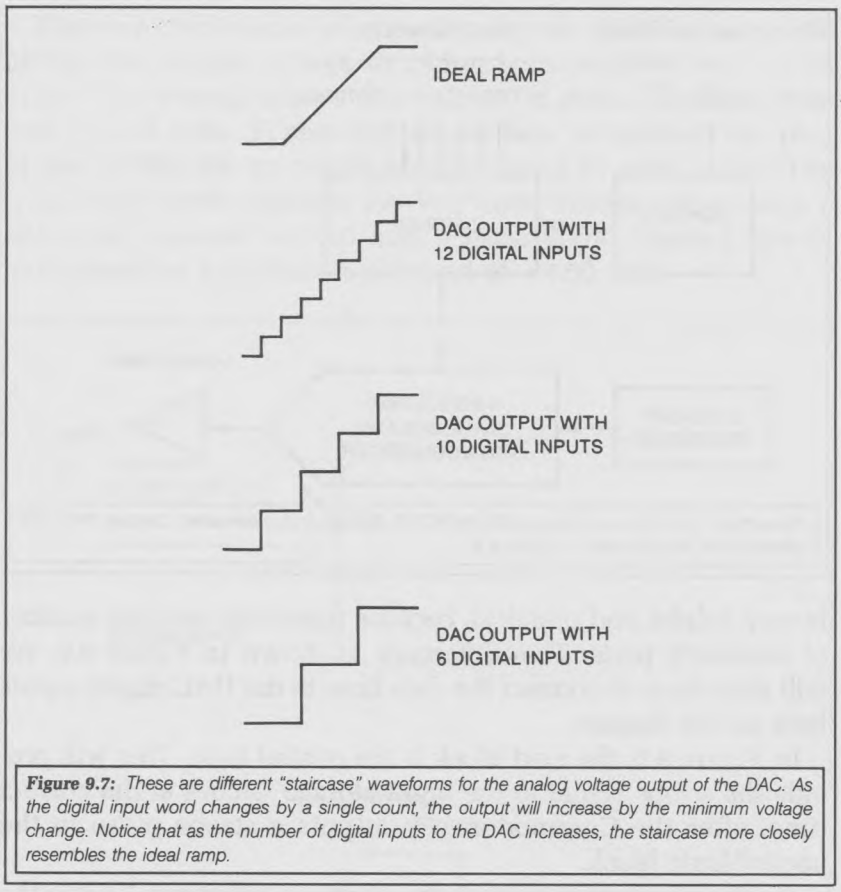


Figure 9.7: *These are different "staircase" waveforms for the analog voltage output of the DAC. As the digital input word changes by a single count, the output will increase by the minimum voltage change. Notice that as the number of digital inputs to the DAC increases, the staircase more closely resembles the ideal ramp.*

9.2 An Actual Digital-to-Analog-Converter

To illustrate the points we have covered about digital-to-analog conversion, let's examine a real, off-the-shelf DAC. This DAC is the AD558™, manufactured by Analog Devices. Appendix C gives a complete data sheet for this device.

Let's cover the details of the block diagram of the AD558 in Figure 9.8. The *digital input block*, labeled DB0–DB7, will connect to the data lines of the Commodore 64. This DAC has built-in latches that strobe and store the data output from the Commodore 64 without the use of an external latch. This feature

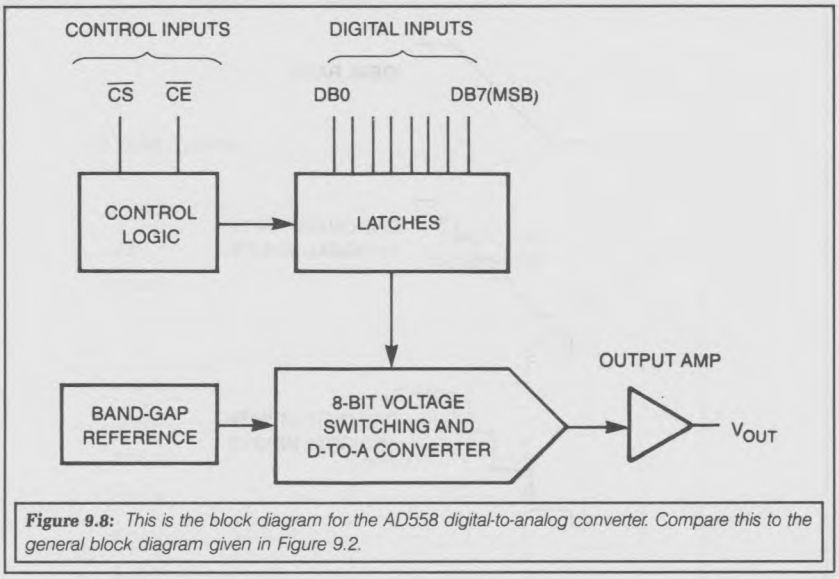


Figure 9.8: *This is the block diagram for the AD558 digital-to-analog converter. Compare this to the general block diagram given in Figure 9.2.*

is very helpful and practical, because it reduces the total number of necessary parts. This difference is shown in Figure 9.9. We will show how to connect the data lines to the DAC digital inputs later in this chapter.

In Figure 9.8, the next block is the *control logic.* This will provide the strobe signal to the input storage latches at the correct time, after the Commodore 64 outputs a strobe pulse to the control-logic block.

Another important block is the *band-gap reference.* This is the internal reference voltage for the DAC. Because the reference voltage is inside the DAC, we do not have to provide an external precision reference. This means that we can use standard power supplies to power the device. Most digital electronic systems have power supplies for this application. The Commodore 64 does not have a +12-volt supply available at its output connector. Therefore we must use an external +12-volt supply.

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 part of the reference voltage to the output. The voltage is output to the outside world from an amplifier, labeled "output amp" in Figure 9.8. Compare the block diagram of Figure 9.8 to the general block diagram in Figure 9.2.

There are two modes of operation for the AD558. One mode allows the output voltage to swing between 0.00 and +2.50 volts. The second allows the voltage to swing between 0.00 and +10.24 volts. Figure 9.10 shows how to connect the pins of the AD558 for an output voltage swing in each mode. The +10.24-volt mode requires the Vcc input voltage pin to have a potential between +11.40 and +16.50 volts. The +2.56-volt mode requires a minimum potential of +4.60 volts.

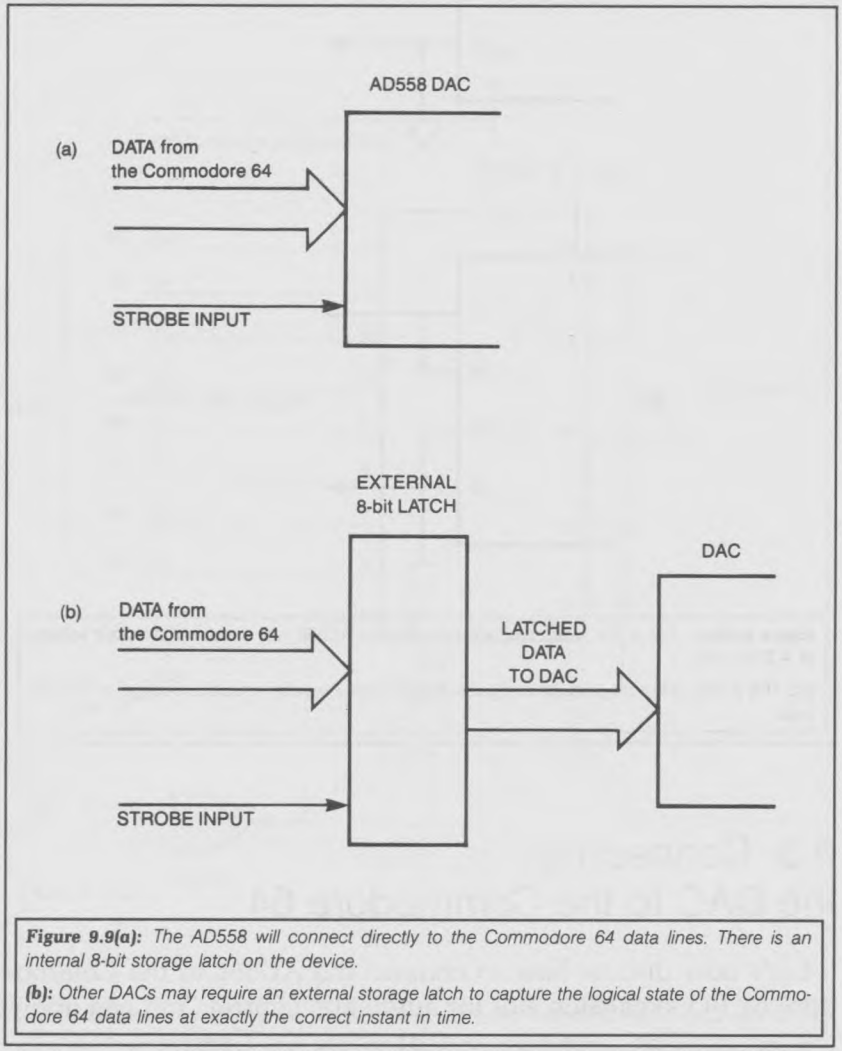


Figure 9.9(a): *The AD558 will connect directly to the Commodore 64 data lines. There is an internal 8-bit storage latch on the device.*

(b): *Other DACs may require an external storage latch to capture the logical state of the Commodore 64 data lines at exactly the correct instant in time.*

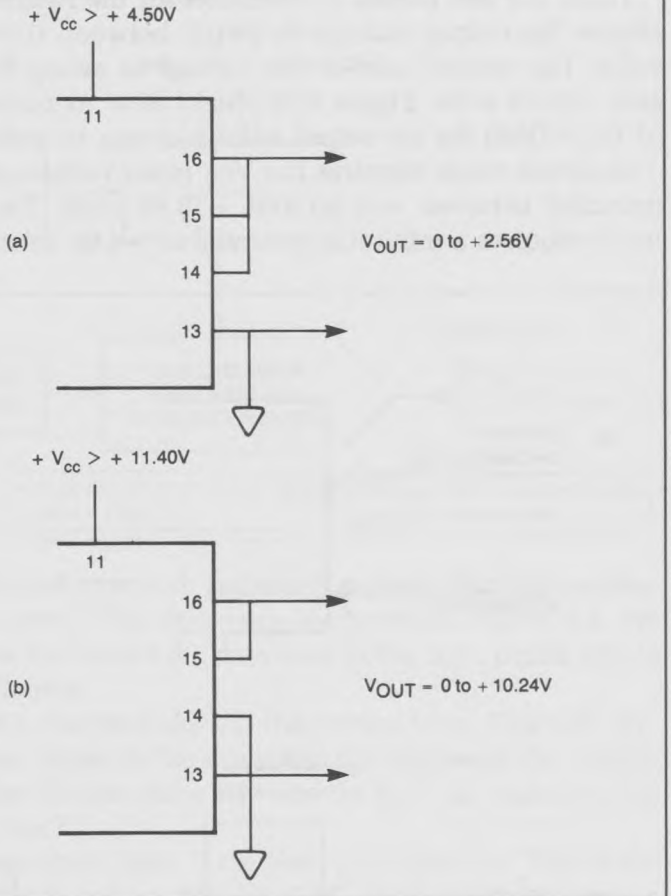


Figure 9.10(a): *This is the wiring diagram for using the AD558 with a maximum output voltage of +2.56 volts.*

(b): *This is the wiring diagram for using the AD558 with a maximum output voltage of +10.24 volts.*

9.3 Connecting the DAC to the Commodore 64

Let's now discuss how to connect the AD558 to the Commodore 64 I/O expansion slot for automatic control. You can apply the information presented to connecting the Commodore 64 to

other DACs as well. Figure 9.11 shows the complete schematic diagram for connecting the AD558 to a Commodore 64.

In Figure 9.11, data lines D0–D7 are connected directly to the data input pins of the device, because the AD558's built-in 8-bit latch will store the logical conditions of the data lines at the correct time. Other DACs may not have this feature. DACs without built-in storage latches require an external 8-bit latch, as shown in Figure 9.12.

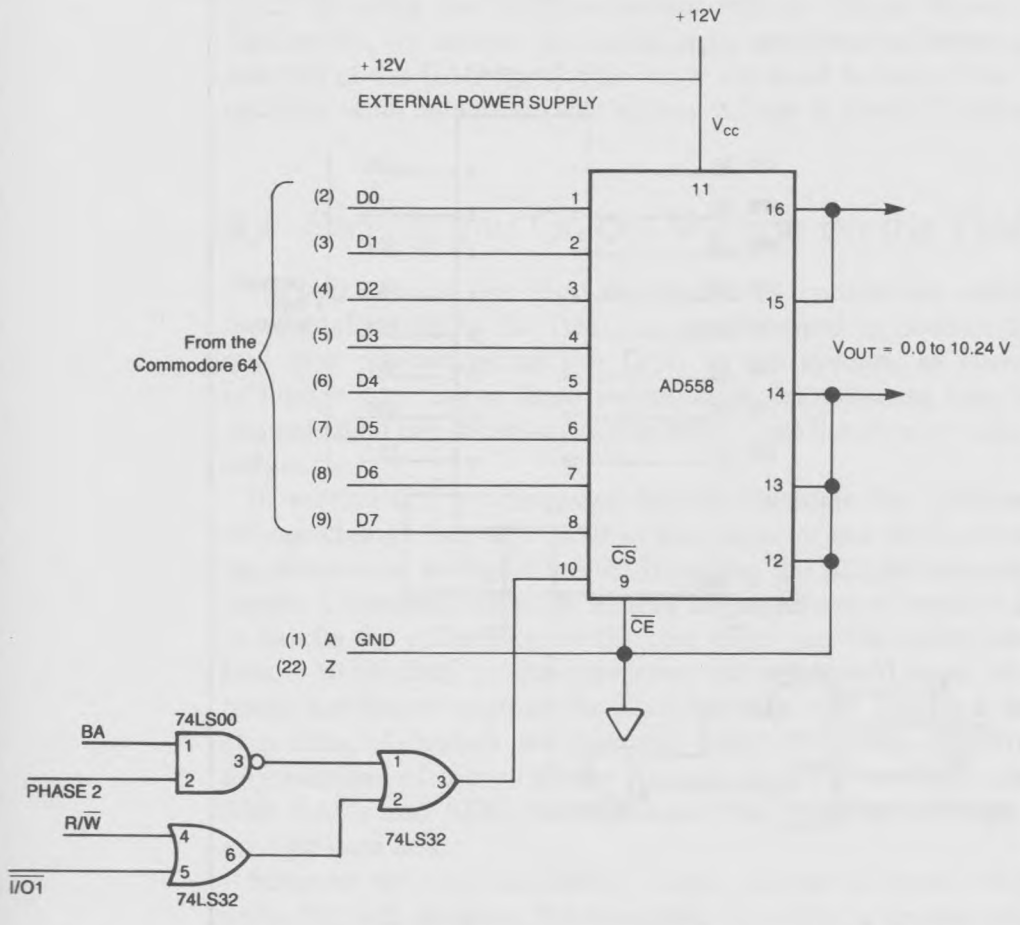


Figure 9.11: *This is the schematic diagram for connecting the AD558 to the Commodore 64 through the I/O expansion slot.*

We will use a POKE statement to strobe the data into the DAC circuit shown in Figure 9.11 or 9.12. (Remember that we described the use of the POKE statement in Chapter 2, and the timing considerations for latching the data in Chapter 4.) When the Commodore 64 executes the POKE statement, the data will be placed at the AD558 data inputs through the data lines.

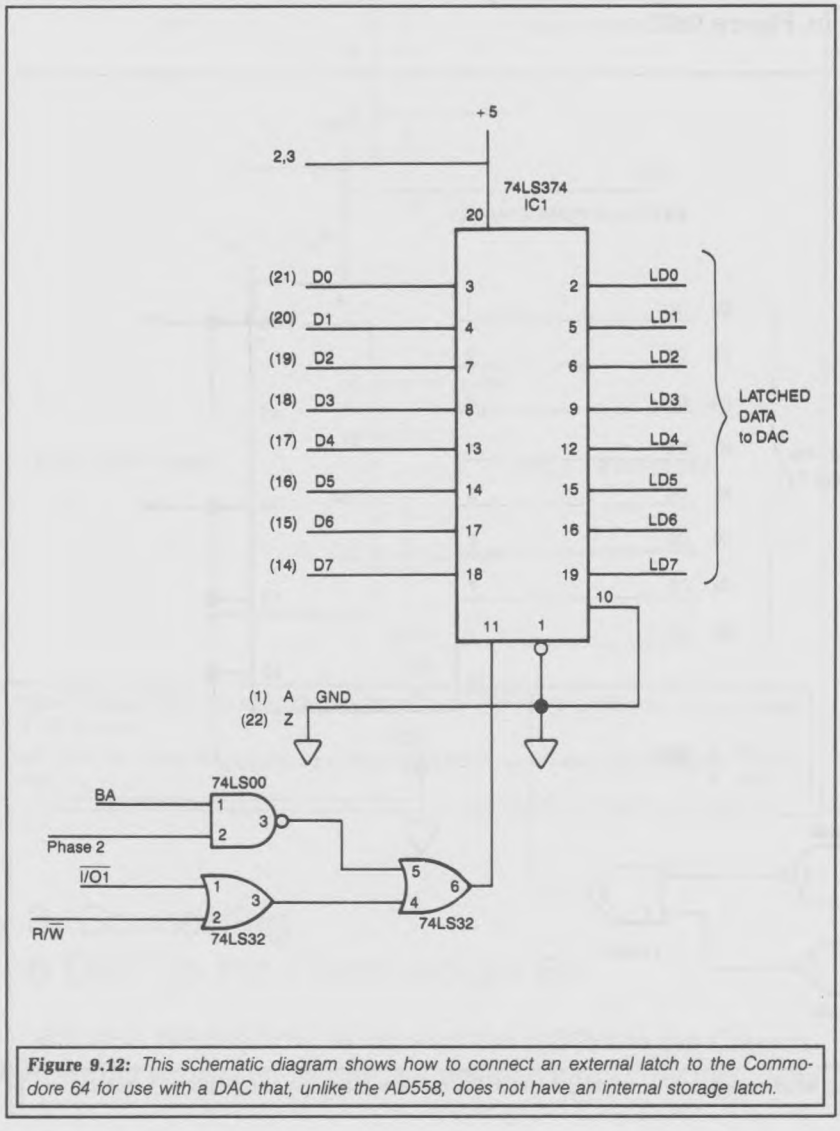


Figure 9.12: *This schematic diagram shows how to connect an external latch to the Commodore 64 for use with a DAC that, unlike the AD558, does not have an internal storage latch.*

At this time, the $\overline{I/O1}$, Phase 2, and the R/$\overline{W}$ control lines will go to a logical 0, and the BA line will be a logical 1. When this occurs, the CS input pin 10 of the AD558 will be set to a logical 0. After the POKE statement is complete, the CS input line will be set to a logical 1, because the R/$\overline{W}$, $\overline{I/O1}$, and Phase 2 lines will go back to a logical 1. We discussed the operation of these lines in Chapter 4.

When the $\overline{CS}$ input line goes to a logical 1, the data at the input pins are strobed into the internal 8-bit latch. The output voltage at pin 16 depends on the POKE data written to the DAC. By using the POKE statement 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 what data to write to set any voltage at the DAC output.

9.4 Setting any Output Voltage on the DAC

We will assume that the Commodore 64 system can control the digital inputs to the DAC, as we discussed in Section 9.3, and that we connected the DAC to the system, as shown in Figure 9.11. Using these assumptions, let's discuss how we can calculate the necessary digital byte to set the correct voltage output.

In section 9.1, we discussed how to calculate the minimum voltage change that will occur at the output of the DAC, assuming there were 10 digital inputs. However, the AD558 has only 8 inputs. Therefore, we must modify the equations of section 9.1 to handle the differences in the two examples. We could interface a 10-bit DAC to the computer, but we would need additional hardware because the data bus can only handle 8 bits at a time. Although we can use DACs or ADCs handling any number of bits with the Commodore 64, we have used 8-bit DACs and ADCs here because they interface directly to an 8-bit data bus.

Suppose we wish the DAC's output voltage to equal +4.80 volts. We will program the computer 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 equal 120.

Let's cover the necessary steps for this calculation. We will use the AD558 DAC in our example, although you can modify the equation to fit any DAC.

We must first calculate the minimum voltage change at the DAC output, by using the procedure outlined in Section 9.1 and the appropriate specifications for the AD558.

$$\text{Minimum output voltage swing} = \frac{\text{maximum output voltage change}}{(\text{maximum number of states}) - 1}$$

The maximum voltage swing on the AD558 will equal 10.24 volts. Because there are eight digital inputs to the AD558, 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.

$$\text{Minimum output voltage swing} = \frac{10.24}{255}$$

$$= 0.040 \text{ volts (40 millivolts)}$$

This means that if the digital input word to the DAC were .00000001, the output voltage would be 40 millivolts.

Given the digital input word, we can calculate the DAC output voltage with this equation:

$$V_{out} = (\text{digital input word}) \times 0.040 \text{ volts}$$

Suppose we have a digital input word equal to 65. We calculate the output voltage as $65 \times 0.040 = 2.60$ volts.

However, if we are given a voltage, 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 0.040 volts for each digital count. Therefore, if we divide the voltage needed at the DAC output by 0.040, the result is the digital value needed to produce the output voltage.

For example, let's suppose that we wished the DAC output voltage to equal 8.00 volts. To calculate the digital input word,

we set up the relationship:

$$\text{Digital word} = \frac{V_{out} \text{ wanted}}{0.040}$$

In our example, the equation would be:

$$\text{Digital word} = \frac{8.00}{0.040}$$
$$= 200$$

If we input the digital weight of 200 to the DAC data input lines, the DAC output voltage will equal 8.00 volts. This calculation is easy to do with the Commodore 64. In the next section, we will present a program to perform this calculation.

Let's suppose that we wanted the DAC to output a voltage equal to 5.25 volts. To calculate the digital word, we would use the equation:

$$\text{Digital word} = \frac{5.25}{0.040}$$
$$= 131.25$$

Notice that the digital word is a real number, not an integer. (Remember that a real number has digits after the decimal point that do not equal zero.) However, to control the DAC with the Commodore 64, we must output digital words as integers such as 1, 2, or 234, or else we cannot obtain the exact voltage we require at the output of the DAC. We must decide 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.)

We must decide between a digital word of 131 and 132. Neither will give the exact output voltage of 5.25 volts. The output voltage of 131 will equal $131 \times 0.040 = 5.24$ volts. The output voltage of 132 will equal $132 \times 0.040 = 5.28$ volts. It is clear that 131 gives an output voltage value closer to 5.25 volts than 132.

Generally, we may round off to the closest integer, and use it as the digital word we need to output. If the decimal part is less than 0.50, use the smaller integer. If the decimal part is greater than or equal to 0.50, use the larger integer as the output word.

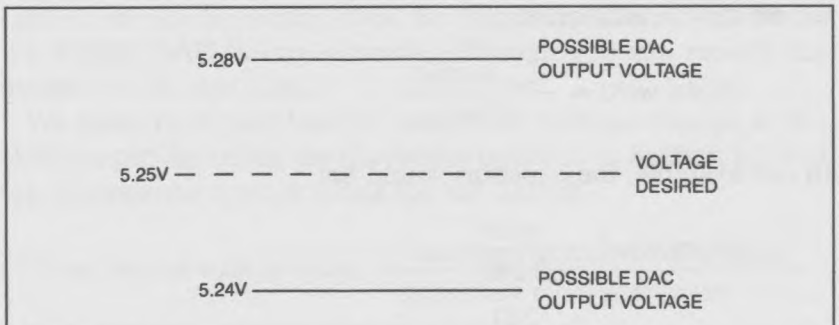


Figure 9.13: *The desired output voltage is 5.25 volts. The DAC will output a voltage of either 5.24 volts or 5.28 volts. We must choose which of these two output voltages is the most desirable for the application.*

To illustrate, let's suppose the required digital word equaled 156.34. Since the decimal part is 0.34, we will use the smaller integer, 156, as the output word. If the required digital word equaled 156.67, we would use 157 as the output word. We can use this process—testing the decimal value and outputting the nearest integer— quite easily with the Commodore 64, if we use the INT function in BASIC. The program in the next section, which calculates the digital value needed, uses this function.

9.5 Controlling the DAC with a BASIC Program

This section presents a BASIC program to program the AD558 to any output voltage between 0.00 and 10.24 volts. The program will perform calculations using the information presented in Section 9.5. The program is shown in Figure 9.14.

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 of the device. (See Figure 9.15.) The AD558 can output 5 milliamperes with no electrical problems. In this section, we will

```
 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
 40 REM
 50 REM
 60 PRINT"INPUT THE VOLTAGE FOR THE DAC"
 70 PRINT"IT MUST BE BETWEEN 0 AND 10.24 VOLTS"
 80 PRINT
 90 INPUT V1
100 REM
110 REM NOW TO CHECK FOR A VALID INPUT VOLTAGE
120 REM
130 IF V1 < 0 OR V1 >10.24 GOTO 1000
140 REM
150 REM NOW TO CALCULATE THE DIGITAL WORD FOR THE DAC
160 REM
170 X = V1/.040
180 REM
190 REM NOW TO ROUND OFF THE NUMBER
200 REM
210 X = INT(X + .5)
220 REM
230 REM NOW TO OUTPUT THE WORD TO THE DAC
240 REM 56832,X
250 POKE 56832,X
260 REM
270 REM THE DAC VOLTAGE IS NOW SET. LOOP BACK TO START
280 REM
290 GOTO 60
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
```

Figure 9.14: *This program will calculate and output the digital value necessary for the DAC to produce a given voltage.*

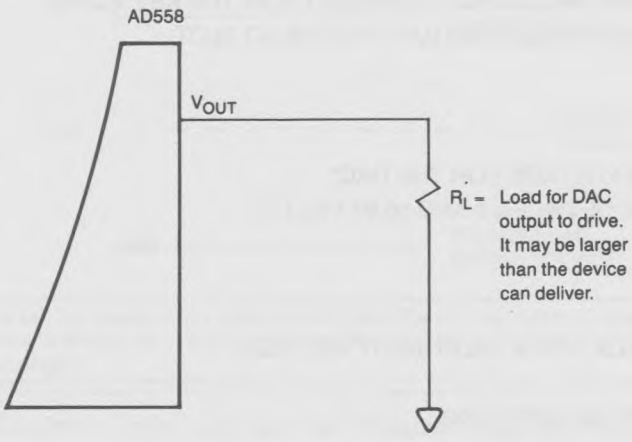


Figure 9.15: *In some applications, the DAC output voltage will be required to drive a load that needs more output current than the DAC output pin can deliver. Usually, DACs will output only a few milliamperes of current to a load.*

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 ranges from 100 to several hundred milliamperes.

We can use the circuit in Figure 9.16 to increase the output-current-drive capability. In this circuit, we take advantage of the current gain of a transistor, the 2N2222, manufactured by many companies, as shown in Figure 9.17. However, we do not know what the exact output voltage will be, because some voltage will drop across the base-emitter junction of the transistor.

Figure 9.18 shows another technique we can use to increase the output-current-drive capability of the AD558. This circuit uses an *operational amplifier* (or op-amp), in addition to the transistor, to compensate for any voltage drop across the base-emitter junction. The output of the op amp will automatically adjust so the two inputs are at the same voltage. This puts the emitter of the transistor at the desired voltage potential, and lets the transistor handle the heavy current. This circuit is a current amplifier with a voltage gain of 1. In Figure 9.17, the voltage at the load will equal the output voltage of the DAC.

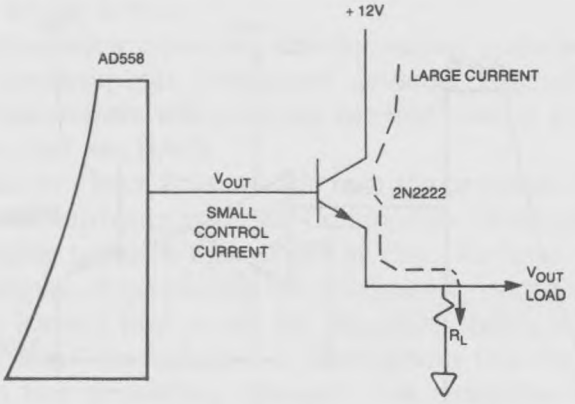


Figure 9.16: *One way to increase the output drive current of the DAC is by using the circuit shown above. The heavy current is controlled by the smaller current in the base of the transistor. This base current is output by the DAC.*

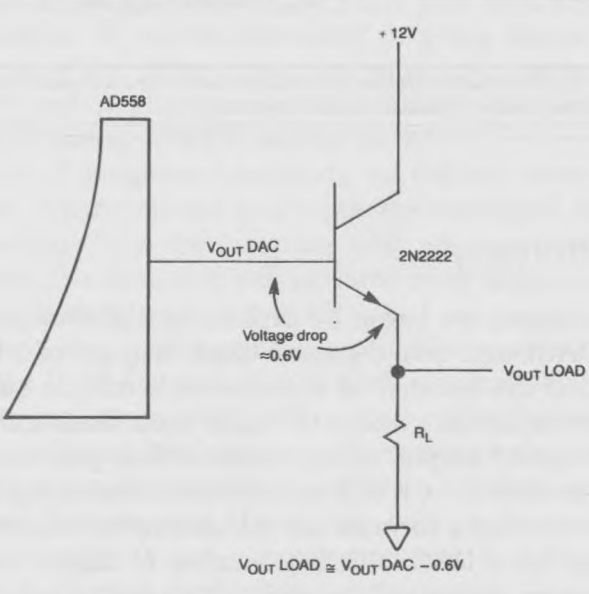


Figure 9.17: *In the circuit of Figure 9.15, we do not know exactly what voltage will be delivered to the load. This is due to the voltage drop across the base-emitter junction of the transistor. The voltage drop is approximately 0.6 volts.*

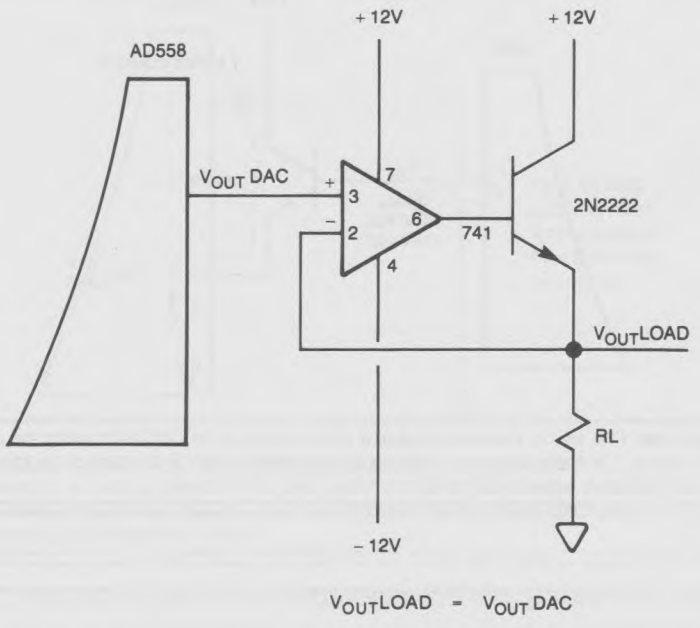


Figure 9.18: *This is a schematic diagram of a circuit that will increase the output current drive of the DAC and compensate for the base-emitter voltage drop.*

9.7 Summary

In this chapter, we began by explaining digital-to-analog conversion (DAC), and gave a general block diagram of a DAC. We showed that the number of a converter's unique output voltages depends on the number of digital input lines and the converter's range of output voltage levels. When you encounter a DAC for the first time, it will be helpful to refer to this diagram.

We then derived a formula for calculating the minimum voltage change for a DAC with any number of digital inputs and any maximum output voltage swing. This led to how to determine what digital word to POKE to the DAC in order to set a desired output voltage. We ended the chapter by showing a sample program for controlling a DAC with the Commodore 64,

and giving general schematics for increasing the output drive capability of the device.

Digital-to-analog coverters are becoming quite popular in many home-computer peripheral devices. The information given in this chapter will help you use and control the peripheral devices that use DACs.

Now that you have finished this text, the prospect of controlling external hardware with the Commodore 64 should seem a much simpler problem than it did at first. We have presented many examples of interfacing the computer to peripheral equipment. You learned how to use the important timing and control signals of the Commodore 64. Throughout this text, we followed the two prevailing concepts that 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.

By now you know enough about controlling peripheral equipment with the Commodore 64 to make your own Commodore 64 connection. If you are interested in going deeper into this subject, there are other important aspects to study. (A good source to read is *Microprocessor Interfacing Techniques*, by R. Zaks and A. Lesea, SYBEX, 3rd ed. 1979.)

One area of computer interfacing we did not cover is that of interrupts. When you use 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 peripheral device at all.

We should also mention the use of different standard buses, such as the IEEE-488 and RS-232, with external devices. These buses allow you to connect your computer directly to a peripheral device with no hardware modifications or special designs. This frees you to concentrate on software development.

We hope that we have opened the door to the essential elements of interfacing and computer control, and that you will use them to develop applications of your own. Good luck and have fun making the Commodore 64 connection.

APPENDIX: GLOSSARY A

AC Appliance

AC is the abbreviation for Alternating Current, a description given to any appliance that can be plugged into a wall socket. A toaster, desk lamp, electric can opener, and coffee maker are examples of AC appliances.

Analog Event

This is 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)

This electrical process converts an analog voltage into its digital equivalent for inputting to any type of digital computer.

Analog Voltage

This is a voltage output from any source that can take on any numeric value. For example, 15.2345V is an analog voltage.

BASIC

(Beginner's All-purpose Symbolic Instruction Code) This is a computer-programming language used by the Commodore 64 and most other home computers.

Binary

This description is given to a set of values that may have only two possible outcomes. For example, the binary number system uses only two digits, 1 and 0.

Bit

A bit is a single binary digit in a computer word. It may have a possible value of 1 or 0.

Byte

A byte is a group of 8 bits. An example would be 00101100. *Byte* and *data word* are synonymous.

Computer Control

This term describes the direction of an external device by a computer.

Data

This is information output or input to the computer.

Digital-to-Analog Conversion

This is 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.

Floppy Disk

This flexible, mylar disk has one or both sides coated to allow magnetic recording of computer information. A floppy disk may be either 8 or 5 ¼ inches in diameter.

Flowchart

This is a visual representation of the logical paths a computer program will follow as the instructions are executed.

4.7K

This is an abbreviation for 4700. The letter K is the metric symbol for 1000. For example, the number 3.6K equals 3.6 × 1000 = 3600.

I/O

I/O is the abbreviation for Input/Output.

I/O Address

This is a logical memory address that is assigned to a specific peripheral device.

I/O Expansion Slot

This slot, located at the back of the Commodore 64, allows for connections with external devices.

Input

During an I/O operation, the computer can *input* or accept data.

Integrated Circuit (IC)

This describes an electronic component that is usually packaged in a Dual In-line Package (DIP).

Latch

This electronic device is capable of storing a single bit of binary information after it has been electrically instructed to do so. An 8-bit latch will store 8 bits of binary information simultaneously.

LED

This stands for Light-Emitting Diode, which is an electronic device that has the physical property of emitting light when current passes through it in the forward direction.

Logical 0

This is one of the two possible binary states that a digital logic circuit can reside in. For the Commodore 64, a logical 0 is when the voltage level of the digital signal is less than or equal to 0.8 volts.

Logical 1

This is the other of the two possible binary states that a digital logic circuit can reside in. In the Commodore 64, a logical 1 is when the voltage output is greater than 2.0 volts.

Nibble

A nibble is a group of 4 bits, such as 0011. A byte consists of 2 nibbles.

Output

A computer *outputs* when it sends data to a peripheral device to control it.

Output Port

This is any output address to which the computer can output data.

PEEK Address

This is the address used in a PEEK statement.

PEEK Statement

This statement in BASIC allows data to be input from a valid memory address.

Phase 2

This is a clock line that times the writing of data to the peripheral circuits connected to the Commodore 64 computer.

POKE Address

This is the memory address that is specified during the execution of the POKE statement.

POKE Statement

This is an instruction in BASIC that allows data to be output to a valid memory address.

R/$\overline{W}$ Line

This signal line is used to electrically inform the I/O circuits if the computer is reading or writing during a memory cycle.

*74LS_*

74LS_ is a numeric label given to integrated circuits. LS is an abbreviation for Low-power Schottky, a type of design technology. The _ in the label can be any 2- or 3-digit number, which will allow you to look the integrated circuit up in a data book. An integrated circuit used in this text is a 74LS32.

Transducer

A transducer is an electronic device that will change a physical action into an electrical equivalent. For example, a temperature transducer will change temperature into an equivalent electrical value.

TTL (Transistor Transistor Logic)

This logic family is used in the circuits of the Commodore 64 computer.

APPENDIX: TIPS ON READING A SCHEMATIC

B

In this text, we use schematic diagrams to illustrate certain aspects of connecting the Commodore 64 to the outside world. We intended the following tips for beginners to help them understand how to read a schematic.

Each symbol used in the schematic diagram represents some physical device. The lines that connect the symbols represent physical wires. Figure B.1 represents a transistor connected to a resistor. Each physical device has a companion schematic symbol.

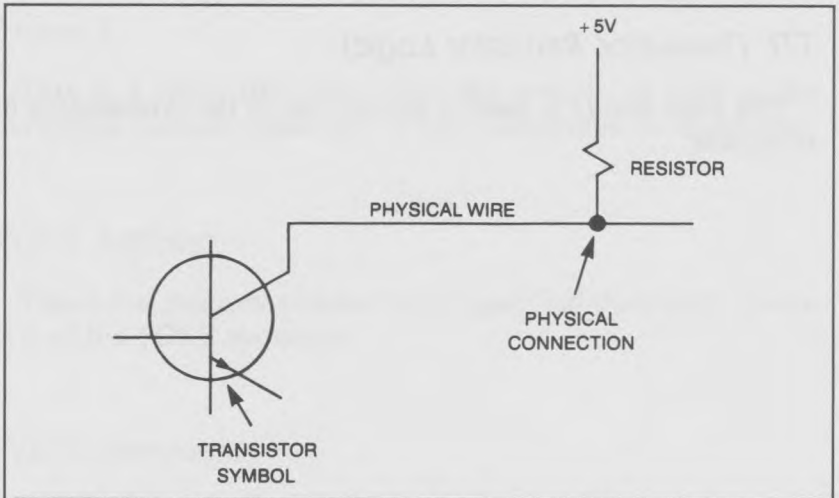


Figure B.1: *This schematic diagram represents a transistor connected to a resistor.*

In digital logic, we use many different symbols to represent the components of a circuit. We will discuss only those symbols that will help you read the schematics presented in this text. However, if you understand these symbols, it will be much easier to read schematics in other texts. Let's discuss all the important parts of the schematic in Figure B.2.

In Figure B.2, the schematic will flow from left to right just as the words do on a page. This is true of most schematics. On the left side of Figure B.2 are the inputs to the circuit. These are given a signal name so that they may be identified wherever they are in the schematic.

Each signal input in Figure B.2 also has a point of origin that indicates where the signal starts from. The inputs will start from the I/O connector pin of the I/O expansion slot. The pin numbers shown next to the signal names are also listed in the Commodore 64 documentation. For example, the data lines D0–D7 connect to a rectangle that represents a single integrated circuit, 74LS374. Each data line is connected to a specific number that corresponds to the pin number of the integrated circuit. This is shown in Figure B.3.

In Figure B.3, we see that the pin numbers of the integrated circuit are labeled in a "U" arrangement. Pin 1 is always at the upper-left corner. Notice that the pin numbers in Figure B.2 do not correspond to Figure B.3. We obtained the pin numbers in Figure B.2 from Figure B.3, but we placed them so that we could draw the schematic easily.

Let's now discuss another input line, $\overline{\text{DEVICE SELECT}}$, to the schematic diagram B. The heavy line over the signal name indicates that this signal will perform its specific function only when it is a logical 0. In this case, the I/O expansion slot is selected when the $\overline{\text{DEVICE SELECT}}$ signal is a logical 0. When the $\overline{\text{DEVICE SELECT}}$ signal is a logical 1, the I/O expansion slot is ignored.

The $\overline{\text{DEVICE SELECT}}$ line originates from the edge connector pin 41. It then connects to the symbol for a logical OR gate shown in Figure B.2. Pin 1 of the OR gate is the physical pin 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.

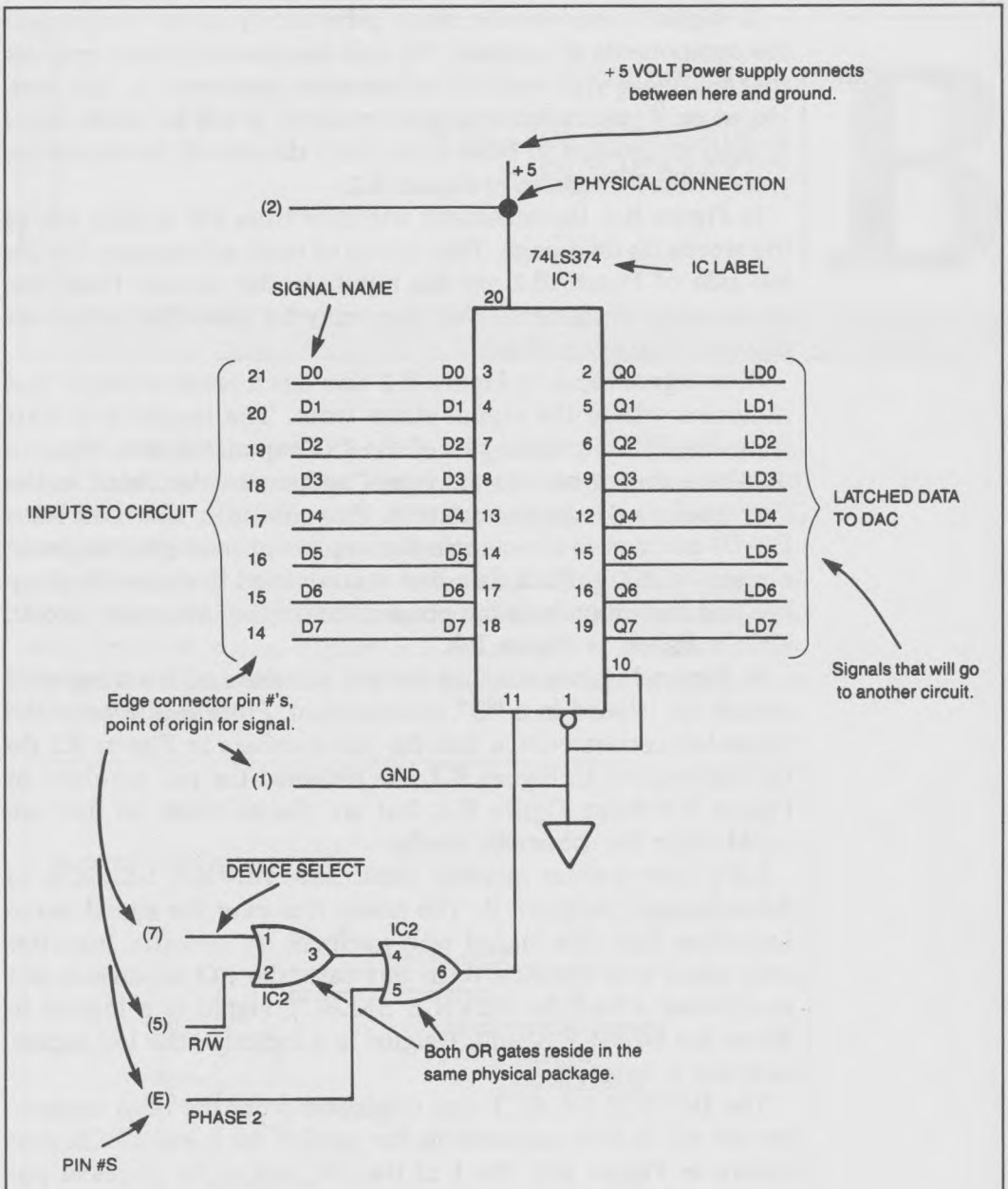


Figure B.2: *This schematic shows the details of an integrated ciruit and its connections.*

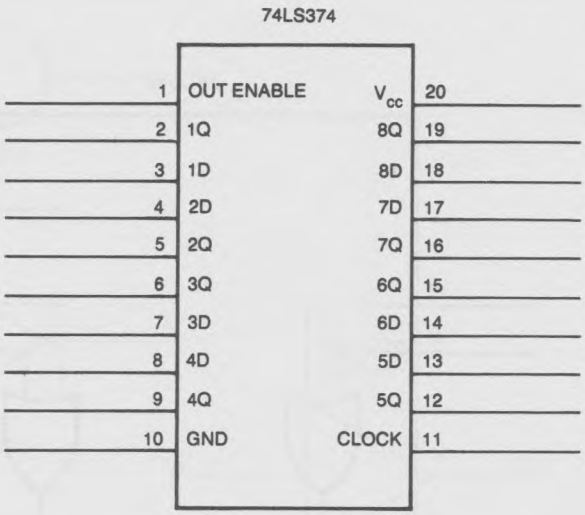


Figure B.3: *The pins on this integrated circuit are shown in their correct numerical order. In Figure B.2, the layout was altered to simplify the drawing and make it more legible.*

Output pin 3 of the OR gate connects to pin 4 of another OR gate. We know that both OR gates are contained in the same integrated circuit package, because both are labeled IC2. "IC" stands for Integrated Circuit. This type of labeling denotes gates that reside in the same physical package on the circuit board.

Output pin 6 of the OR gate is connected to input pin 11 of the 74LS374 integrated circuit that we discussed earlier. We can see that the outputs of the 74LS374 integrated circuit are not connected to anything in schematic B.2, because the lines are not connected to other symbols. Notice that the flow of the schematic is from left to right, input to output.

When we use this information, we can read the schematic diagrams shown in this text. We discussed all of the special symbols—like the symbol for a light emitting diode—in the section of the text they are used. We have used only a few of the many logic symbols available, which are shown in Figure B.5. Appendix C gives all of the pinouts for each of the physical integrated circuits used in this text.

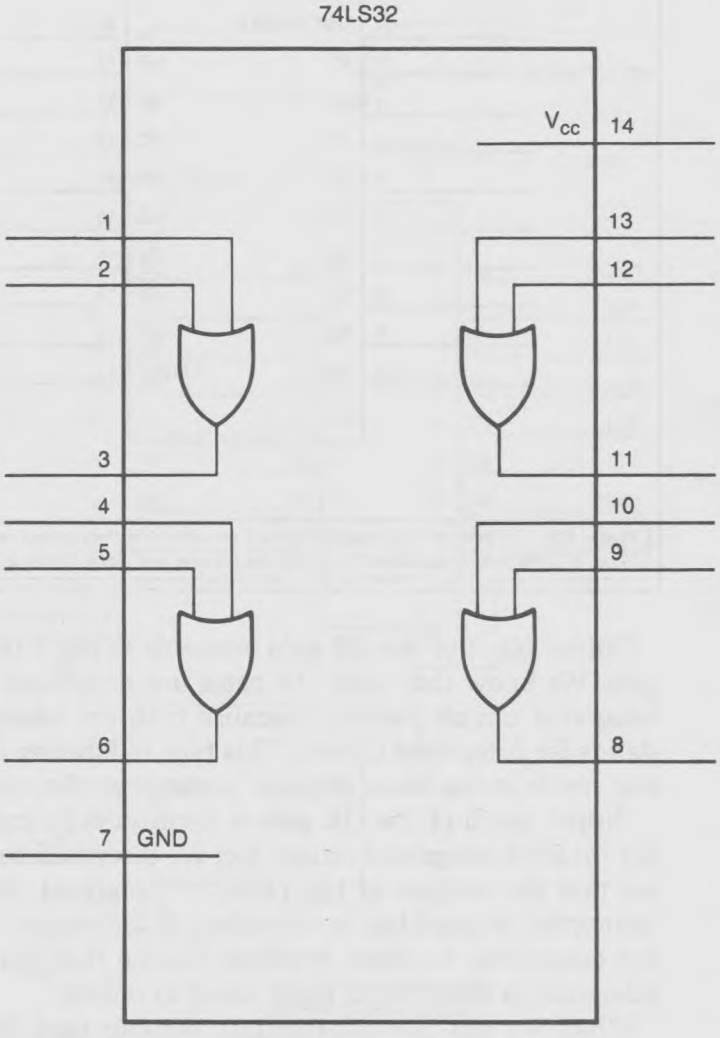


Figure B.4: *All four OR gates are located in the same device package.*

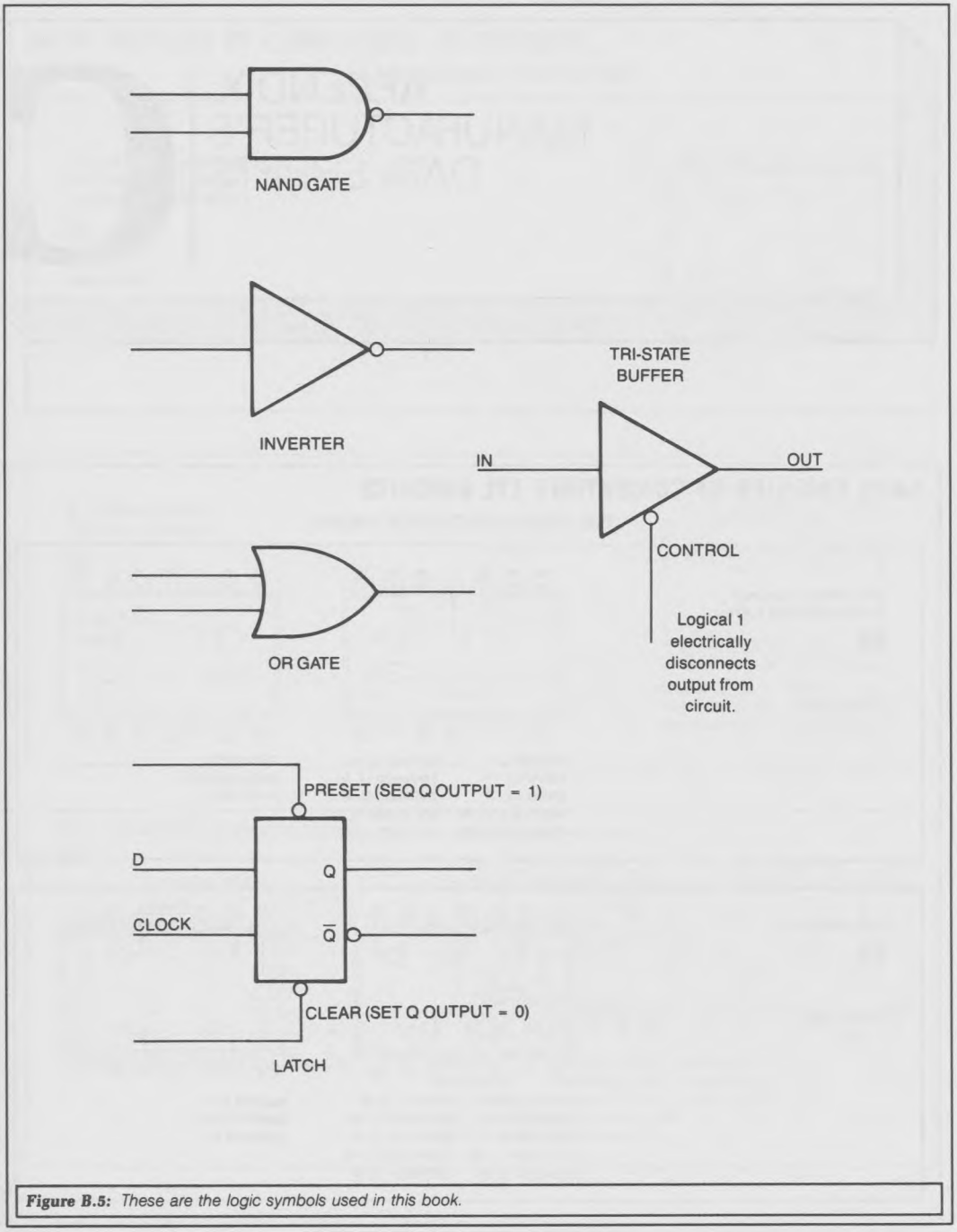


Figure B.5: *These are the logic symbols used in this book.*

APPENDIX: MANUFACTURER'S DATA SHEETS

C

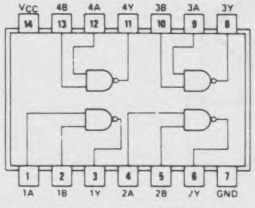
54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

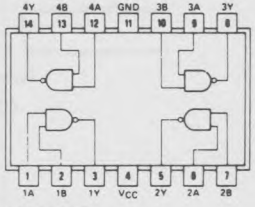
PIN ASSIGNMENTS (TOP VIEWS)

QUADRUPLE 2-INPUT POSITIVE-NAND GATES

00

positive logic:
$Y = \overline{AB}$





SN5400 (J)
SN54H00 (J)
SN54L00 (J)
SN54LS00 (J, W)
SN54S00 (J, W)

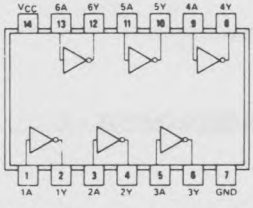
SN7400 (J, N)
SN74H00 (J, N)
SN74L00 (J, N)
SN74LS00 (J, N)
SN74S00 (J, N)

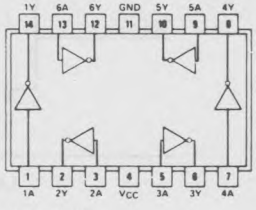
SN5400 (W)
SN54H00 (W)
SN54L00 (T)

HEX INVERTERS

04

positive logic:
$Y = \overline{A}$





SN5404 (J)
SN54H04 (J)
SN54L04 (J)
SN54LS04 (J, W)
SN54S04 (J, W)

SN7404 (J, N)
SN74H04 (J, N)
SN74L04 (J, N)
SN74LS04 (J, N)
SN74S04 (J, N)

SN5404 (W)
SN54H04 (W)
SN54L04 (T)

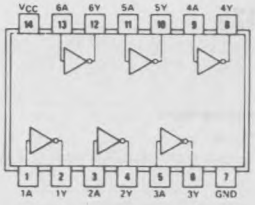
54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

PIN ASSIGNMENTS (TOP VIEWS)

HEX INVERTER BUFFERS/DRIVERS
WITH OPEN-COLLECTOR
HIGH-VOLTAGE OUTPUTS

06

positive logic:
$Y = \overline{A}$

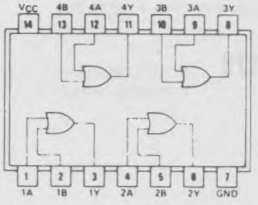


SN5406 (J, W) SN7406 (J, N)

QUADRUPLE 2-INPUT
POSITIVE-OR GATES

32

positive logic:
$Y = A+B$



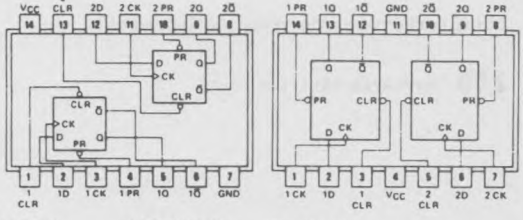
SN5432 (J, W) SN7432 (J, N)
SN54LS32 (J, W) SN74LS32 (J, N)
SN54S32 (J, W) SN74S32 (J, N)

DUAL D-TYPE POSITIVE-EDGE-TRIGGERED FLIP-FLOPS WITH PRESET AND CLEAR

74

FUNCTION TABLE

INPUTS				OUTPUTS	
PRESET	CLEAR	CLOCK	D	Q	$\overline{Q}$
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	$\overline{Q}_0$



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)

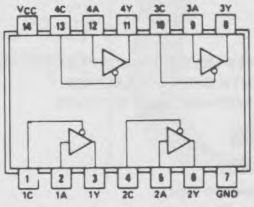
54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

PIN ASSIGNMENTS (TOP VIEWS)

QUADRUPLE BUS BUFFER GATES WITH THREE-STATE OUTPUTS

125

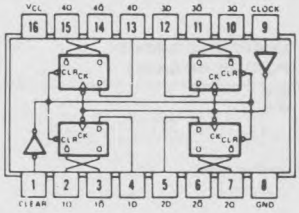
positive logic:
Y = A
Output is off (disabled) when C is high.



SN54125 (J, W) **SN74125 (J, N)**
SN54LS125A (J, W) **SN74LS125A (J, N)**

QUAD D-TYPE FLIP-FLOPS

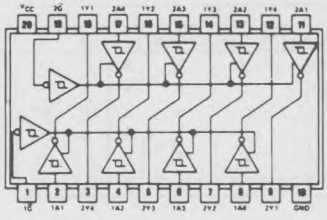
175 COMPLEMENTARY OUTPUTS
COMMON DIRECT CLEAR



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



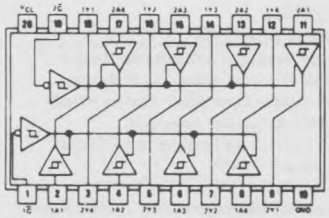
SN54LS240 (J) **SN74LS240 (J, N)**
SN54S240 (J) **SN74S240 (J, N)**

54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

PIN ASSIGNMENTS (TOP VIEWS)

OCTAL BUFFERS/LINE DRIVERS/LINE RECEIVERS

244 NONINVERTED 3-STATE OUTPUTS



SN54LS244 (J) SN74LS244 (J, N)

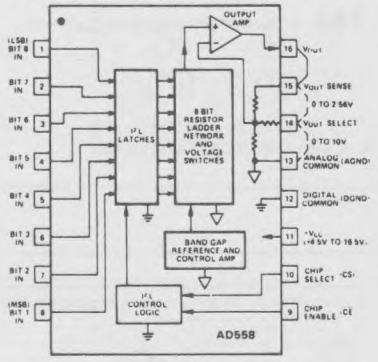
ANALOG DEVICES

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	•••

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

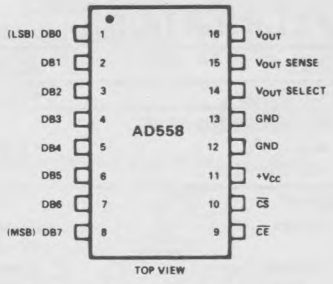


Figure 1. AD558 Pin Configuration

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.

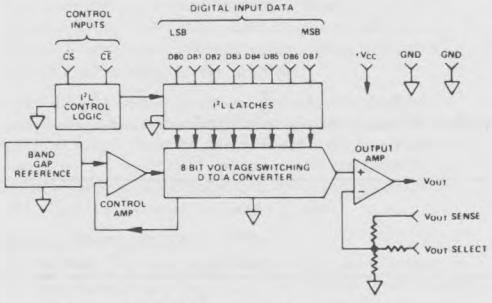


Figure 2. AD558 Functional Block Diagram

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

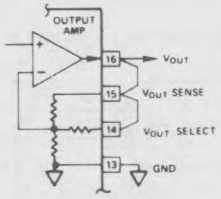
The AD558 has been configured for ease of application. All reference, output amplifier and logic connections are made internally. In addition, all calibration trims are performed at the factory assuring specified accuracy without user trims. The only connection decision that must be made by the user 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.

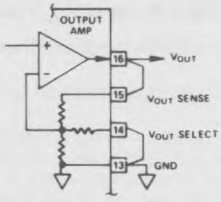
Because of its precise factory calibration, the AD558 is intended to be operated without user trims for gain and offset; therefore no provisions have been made for such user-trims. If a small increase in scale is required, however, it may be accomplished by slightly altering the effective gain of the 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

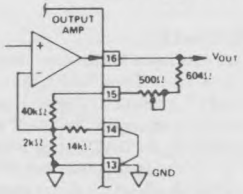


Figure 4. 10.24V Full-Scale Connection

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.

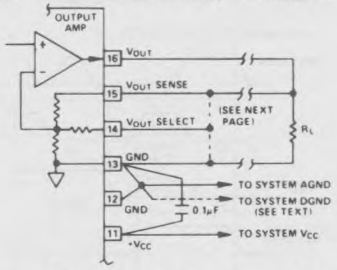


Figure 5. Recommended Grounding and Bypassing

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

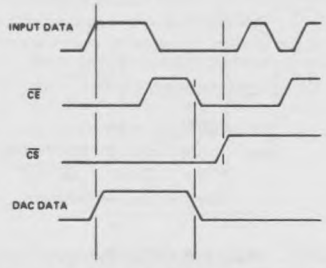


Figure 6. AD558 Control Logic Function

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.

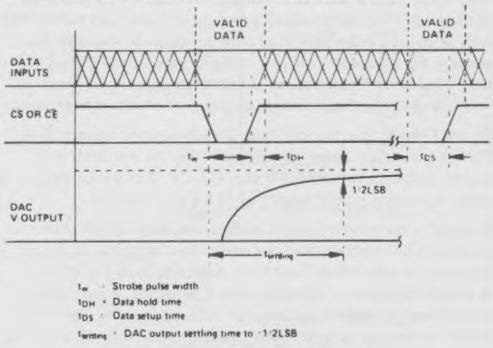
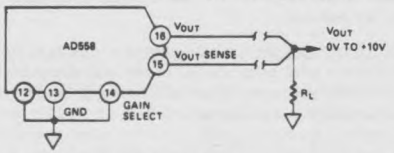


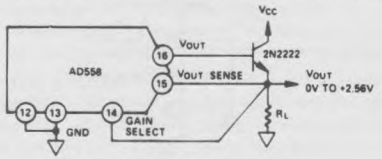
Figure 7. AD558 Timing

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.

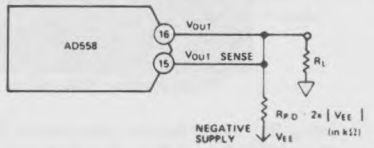


Figure 9. Improved Settling Time

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.

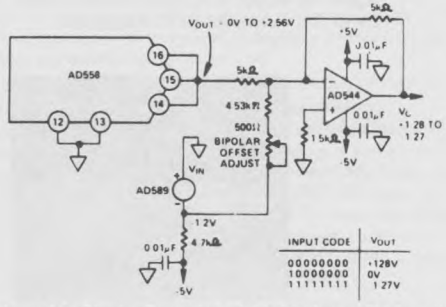
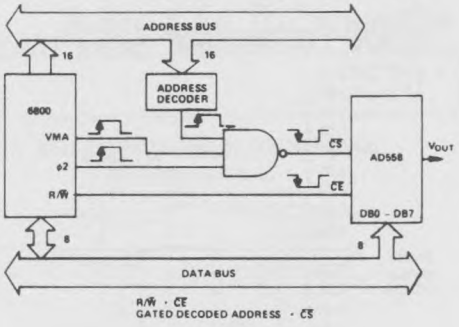


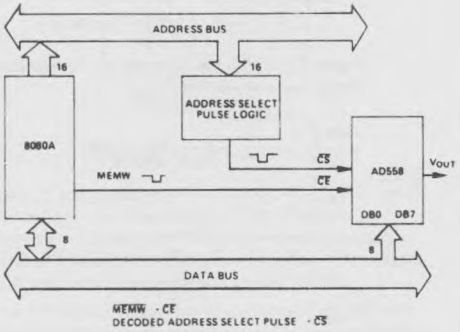
Figure 10. Bipolar Operation of AD558 from ±5V Supplies

INTERFACING THE AD558 TO MICROPROCESSOR DATA BUSES*

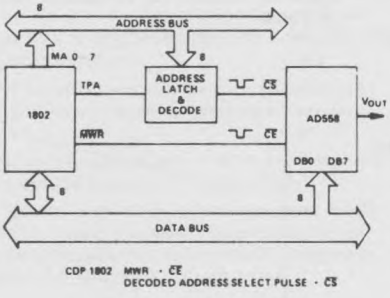
The AD558 is configured to act like a "write only" location in memory that may be made to coincide with a read only memory location or with a RAM location. The latter case allows data previously written into the DAC to be read back later via the RAM. Address decoding is partially complete for either 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)

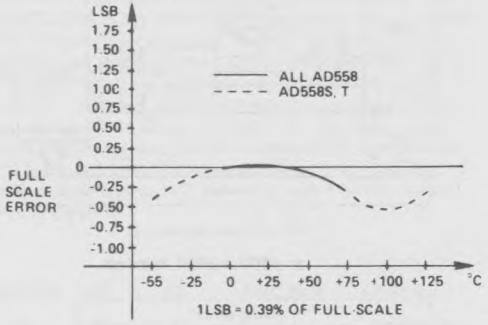


Figure 12. Full Scale Accuracy vs. Temperature Performance of AD558

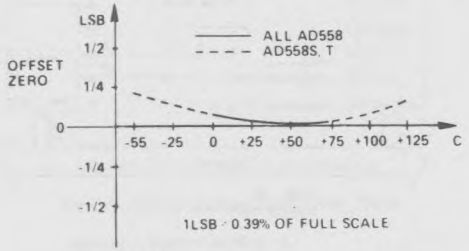


Figure 13. Zero Drift vs. Temperature Performance of AD558

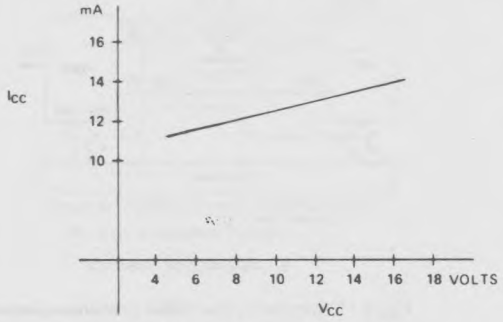


Figure 14. Quiescent Current vs. Power Supply Voltage for AD558

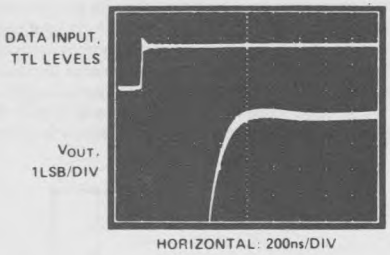


Figure 15. AD558 Settling Characteristic Detai 0V to 2.56V Output Range Full-Scale Step

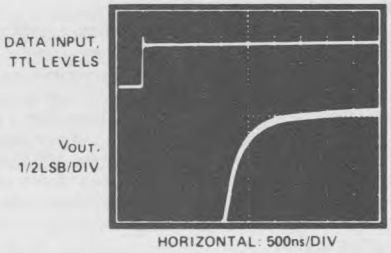


Figure 16. AD558 Settling Chracteristic Detail 0V to 10V Output Range Full-Scale Step

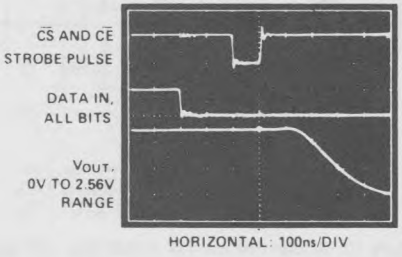


Figure 17. AD558 Logic Timing

ANALOG DEVICES

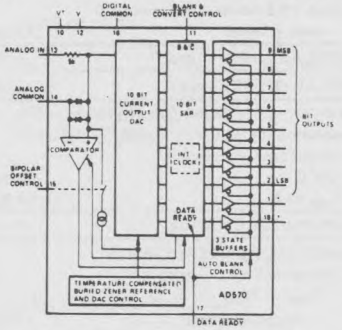
Low Cost, Complete IC 8-Bit A to D Converter

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≤V_{in}≤V+	±40μA max	•
Blank – Logic "1"	2.0V min	•
Convert – Logic "0"	0.8V max	•
CONVERSION TIME	15μs min	•
	25μs typ	•
	40μs max	•

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.

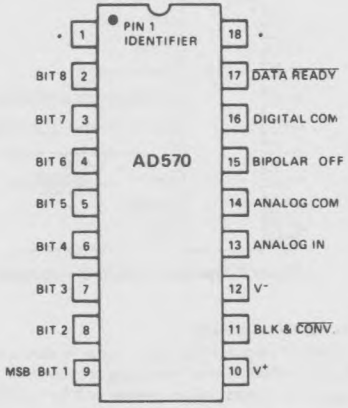


Figure 1. AD570 Pin Connections

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.

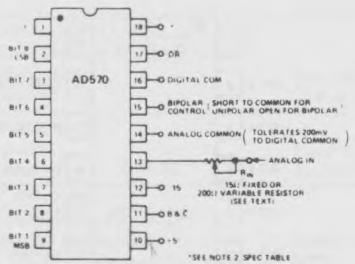


Figure 2. Standard AD570 Connections

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.

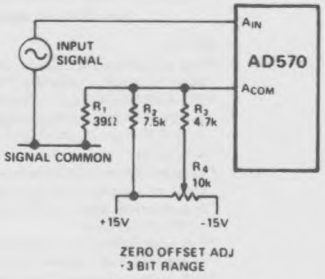


Figure 3A.

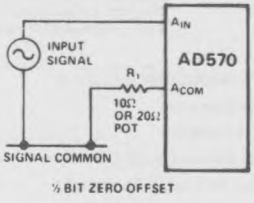


Figure 3B.

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.

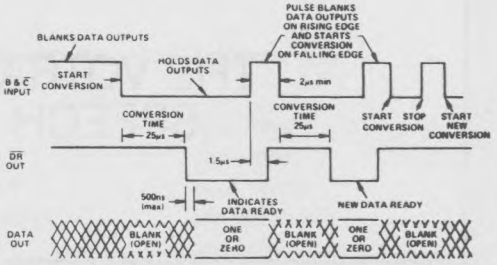
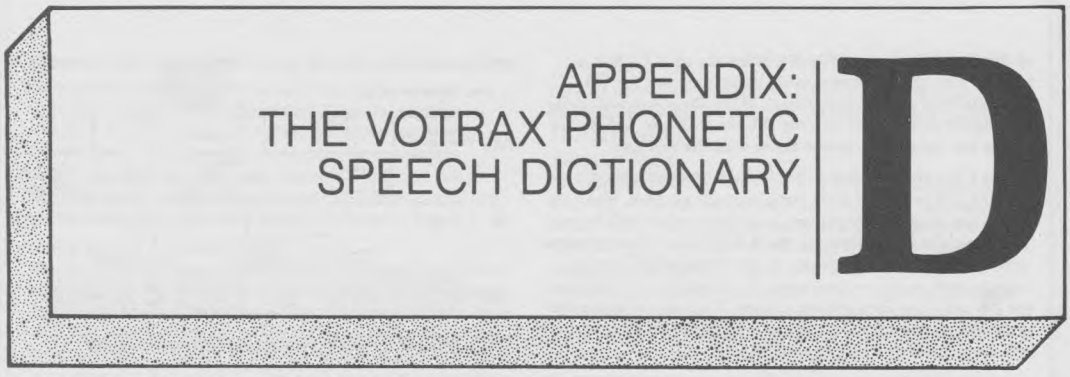


Figure 4. AD570 Timing and Control Sequence

APPENDIX: THE VOTRAX PHONETIC SPEECH DICTIONARY



General Description of the SC-01 Chip

The SC-01 Speech Synthesizer is a completely self-contained, solid-state device. This single chip phonetically synthesizes continuous speech of unlimited vocabulary from low data-rate inputs.

Speech is synthesized by combining phonemes (the building blocks of speech) in the appropriate sequence. The SC-01 contains 64 different phonemes that are accessed by a 6-bit code. The proper sequential combination of these phoneme codes creates continuous speech.

Phoneme Description

Figure 6.1 lists the 64 phonemes produced by the SC-01. Each sound is represented by its Votrax phoneme code and is accompanied by its phoneme symbol and an example. The underlined segment of the example word demonstrates the sound of the phoneme.

Table 1 provides the phoneme sequences used to produce vowel groups called dipthongs (2 vowel sounds in sequence with a single sound).

Table 2 lists approximately 1400 words and their phoneme sequences.

TABLE 1: DIPHTHONG CHART

Phoneme Combination	Key Words
A1-AY-Y	fate, maid
AH1-EH3-Y	find, wide
UH3-AH2-Y	fight, white
AH1-I3-UH3-L	file, smile
01-UH3-Y	foy, boy
01-I3-UH3-L	foil, spoil
AH1-02-U1	found, cow
UH3-AH2-U1	foust, house
O1-U1	float, note
Y1-IU-U1	few, you, music
AY-I1	fear, beer

Word	Phonetic Program
Prefixes	
con...	K, UH1, N
dis...	D, I1, S
en...	EH1, N
in...	I1, N
non...	N, AH1, UH3, N
pre...	P, R, E1
re...	R, E1
un...	UH1, N
Suffixes	
...d	D
...ed	I2, D
...er	ER
...es	I2, Z
...ful	F, UH3, L
...ing	I2, NG
...less	L, EH2, S
...ly	L, Y
...ment	M, EH3, N, T
...ness	N, EH3, S
...s	S
...t (...ed)	T
...tion (...sion)	SH, UH3, N
...teen	T, E1, Y, N
...ward	W, ER, D
...y	Y
...z (...es)	Z

Word	Phonetic Program
A	A1, AY, Y
a-2	UH2, UH3
able	A1, Y, B, UH3, L
abort	UH1, B, O2, O2, R, T
about	UH1, B, UH2, AH2, U1, T
above	UH1, B, UH1, UH3, V
accept	EH1, K, PAO, S, EH1, EH3, P, T
access	AE1, EH3, K, PAO, S, EH1, EH3, S
account	UH1, K, AH1, UH3, W, N, T
acid	AE1, EH3, S, I1 D
act	AE1, EH3, K, T
active	AE1, EH3, K, T, I1, V
actual	AE1, EH3, K, T, CH, U1, UH3, L
add	AE1, EH3, D
address	AE1, EH3, D, R, EH1, EH3, S
ade	(use "aid" program)
adjust	UH1, D, J, UH1, UH3, S, T
adjucent	UH1, D, J, A1, AY, S, EH3, N, T
advance	AE1, EH3, D, V, AE1, EH3, N, T, S
advise	AE1, EH3, D, V, AH1, EH3, Y, Z
affect	UH1, F, EH1, EH3, K, T
after	AE1, EH3, F, T, ER
again	UH1, G, A2, EH1, N
age	A1, AY, Y, D, J
agent	A1, Y, D, J, EH3, N, T
ahead	UH1, H, EH1, EH3, D
aid	A1, AY, Y, D
air	EH2, EH2, R
alarm	UH1, L, AH1, R, M
alert	UH1, L, ER, R, T
all	AW, L
allocate	AE1, UH3, L, UH2, K, A1, Y, T
allow	UH1, L, AH1, UH3, U1
alpha	AE1, AW2, L, F, UH1
already	AW, L, R, EH1, EH3, D, Y
also	AW, L, S, O1, U1
altitude	AE1, UH3, L, T, I2, T, IU, U1, U1, D
aluminum	UH1, L, IU, U1, M, I3, N, UH1, M
am	AE1, EH3, M
america	UH1, M, EH1, R, I3, K, UH2, UH3
amount	UH1, M, AH1, UH3, W, N, T
amp	AE1, EH3, M, P
amplify	AE1, EH3, M, P, L, I3, F, AH1, EH3, AY
an	AE1, EH3, N
and	AE1, EH3, N, D
angle	AE1, EH3, NG, G, UH3, L
another	UH1, N, UH1, UH3, THV, ER
answer	AE1, EH3, N, S, ER
any	EH2, EH2, N, Y
apostrophe	UH1, P, AH1, UH3, S, T, R, UH3, F, Y
approach	UH1, P, R, O1, U1, T, CH
approve	UH1, P, R, IU, U1, U1, V
approximate	UH1, P, R, AH1, K, PAO, S, EH3, M, I3, T
approximate-2	UH1, P, R, AH1, K, PAO, S, EH3, M, A2, Y, T
april	A1, Y, P, R, UH2, L
architect	AH1, R, K, UH2, T, EH3, EH2, K, T
are	(see "R" program)
area	EH1, EH3, R, Y, UH1
arrive	UH1, R, AH1, EH3, Y, V
arrow	EH1, EH3, R, O1, U1
article	AH1, R, T, EH3, K, UH3, L
as	AE1, EH3, Z
ASCII	AE1, EH3, S, K, Y
ask	AE1, EH3, S, K
assemble	UH1, S, EH1, EH3, M, B, UH3, L
asset	AE1, EH3, S, EH1, T
assign	UH1, S, AH1, EH3, Y, N
assist	UH1, S, I1, I3, S, T
associate	UH1, S, O1, SH, Y, A1, Y, T
associate-2	UH1, S, O1, SH, Y, I2, T
assume	UH1, S, IU, U1, M
at	AE1, EH3, T
ate	(see "eight" program)
attach	UH1, T, AE1, EH3, T, CH
attempt	UH1, T, EH1, EH3, M, P, T
attend	UH1, T, EH1, EH3, N, D
audio	AW, D, Y, O1, U1
august	AW2, AW2, G, EH2, S, T
authorize	AW2, AW2, TH, ER, AH1, Y, Z
automatic	AW2, AW2, DT, UH3, M, AE1, EH3, DT, I3, K
available	UH1, V, A1, Y, L, UH3, B, UH3, L
average	AE1, EH3, V, R, I1, D, J
avoid	UH1, V, O1, UH3, I3, AY, D
B	B, E1, Y
back	B, AE1, AE1, K
bad	B, AE1, AE1, D
badge	B, AE1, AE1, D, J
bag	B, AE1, AE1, G

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program
balance	B, AE1, AH2, L, I3, N, DT, S
ball	B, AW2, AW1, L
band	B, AE1, EH3, N, D
bank	B, AE1, I3, NG, K
bar	B, AH1, UH3, R
base	B, A1, AY, Y, S
basic	B, A1, Y, S, I2, K
bat	B, AE1, EH3, T
batch	B, AE1, EH3, T, CH
bath	B, AE1, AE1, EH3, TH
battery	B, AE1, EH3, T, ER, Y
be	(use "B" program)
bed	B, EH1, EH3, D
been	B, EH1, EH3, N
beep	B, E1, Y, P
before	B, Y, F, O2, O2, R
begin	B, Y, G, I1, I3, N
bell	B, EH1, UH3, L
below	B, Y, L, UH3, O2, U1
bend	B, EH1, EH3, N, D
best	B, EH1, EH3, S, T
beta	B, A2, A2, AY, T, UH2
better	B, EH1, EH3, T, ER
between	B, Y, T, W, E1, Y, N
bid	B, I1, I3, D
big	B, I1, I3, G
bill	B, I1, I3, L
billion	B, I1, I3, L, Y, UH3, N
bin	B, I1, I3, N
binary	B, AH1, Y, N, EH3, EH3, ER, Y
birthday	B, ER, R, TH, D, A1, I3, Y
bit	B, I1, I3, T
bite	B, UH3, AH2, Y, T
black	B, L, AE1, EH3, K
blank	B, L, AE1, EH3, NG, K
blew	(use "blue" program)
blind	B, L, AH1, EH3, Y, N, D
block	B, L, AH1, UH3, K
blown	B, L, O1, U1, N
blue	B, L, IU, U1, U1
blur	B, L, ER, R
board	B, O1, O2, R, D
bolt	B, O2, O2, L, T
bond	B, AH1, UH3, N, D
book	B, OO1, OO1, K
bored	(use "board" program)
boss	B, AW1, AW2, S
bother	B, AH1, UH3, THV, ER
bottom	B, AH1, UH3, T, UH1, M
bought	B, AW1, AW2, T
box	B, AH1, UH3, K, PAO, S
brace	B, R, A1, Y, S
brain	B, R, A1, Y, N
brake	B, R, A1, Y, K
branch	B, R, AE1, EH3, N, T, CH
bravo	B, R, AH1, UH3, V, O1, U1
break	(use "brake" program)
bridge	B, R, I1, I3, D, J
brief	B, R, AY, Y, F
bright	B, R, UH3, AH2, Y, T
bring	B, R, I1, I3, NG
broke	B, R, O1, U1, K
brought	B, R, AW, T
brown	B, R, AH1, UH3, U1, N
bubble	B, UH1, UH2, B, UH3, L
budget	B, UH1, UH3, D, J, I2, T
bug	B, UH1, UH2, G
build	B, I2, I2, L, D
bus	B, UH1, UH2, S
business	B, I3, I3, Z, N, EH2, S
busy	B, I3, I2, Z, Y
but	B, UH1, UH2, T
button	B, UH1, UH3, T, EH3, N
buy	B, AH1, EH3, I3, Y
by	B, AH1, EH3, I3, Y
bye	B, AH1, EH3, I3, Y
byte	(use "bite" program)
C	S, E1, Y
cable	K, A1, Y, B, UH3, L
calendar	K, AE1, UH3, L, I3, N, D, ER
calibrate	K, AE1, UH3, L, UH3, B, R, A1, Y, T
call	K, AW2, AW1, L
came	K, A1, AY, Y, M
can	K, AE1, EH3, N
cancel	K, AE1, EH3, N, S, UH3, L
capable	K, A1, Y, P, UH3, B, UH3, L
capacitor	K, UH2, P, AE1, EH3, S, EH3, T, ER
capacity	K, UH2, P, AE1, EH3, S, I3, DT, Y
car	K, AH1, UH3, R
card	K, AH1, R, D
care	K, EH3, EH3, ER
carpenter	K, AH1, R, P, I3, N, D, ER
carriage	K, EH2, EH3, R, I1, D, J
carry	K, EH2, EH3, R, Y
carton	K, AH1, R, T, I3, N
case	K, A1, AY, Y, S

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program
cash	K, AE1, EH3, SH
cassette	K, UH1, S, EH1, EH3, T
cassette-2	K, A2, AY, S, EH1, EH2, T
category	K, AE1, EH3, DT, UH3, G, O1, R, Y
catalog	K, AE1, EH3, DT, UH3, L, AW2, AW2, G
caution	K, AW2, AW1, SH, UH3, N
cent	S, EH1, EH3, N, T
center	S, EH1, EH3, N, T, ER
centi	S, EH1. EH3, N, T, I1, I3
centigrade	S, EH1, N, T, I3, G, R, A1, Y, D
certify	S, R, R, T, I3, F, AH1, Y
change	T, CH, A1, AY, Y, N, D, J
character	K, EH1, R, EH1, K, T, ER
charge	T, CH, AH1, R. D, J
charlie	T, CH, AH1, R, L, Y
chart	T, CH, AH1, R, T
check	T, CH, EH1, EH3, K
cheer	T, CH, AY, I2, R
chip	T, CH, I1, I3, P
choice	T, CH, O1, UH3, I3, AY, S
circle	S, ER, R, K, UH3, L
circuit	S, R, R, K, I2, T
city	S, I1, T, Y
claim	K, L, A1, AY, Y, M
class	K, L, AE1, EH3, S
clean	K, L, E1, AY, N
clear	K, L, AY, I3, R
clerk	K, L, ER, K
clip	K, L, I1, I3, P
clock	K, L, AH1, UH3, K
close	K, L, UH3, O1, U1, Z
close-2	K, L, UH3, O2, U1, S
cloud	K, L, AH1, UH3, W, D
coarse	K, O1, O2, R, S
code	K, OO1, O2, U1, D
coin	K, O1, UH3, I3, AY, N
collar	K, AH1, UH3, L, ER
collect	K, UH1, L, EH1, K, T
colon	K, OO1, O2, U1, L, I2, N
color	K, UH2, UH2, L, ER
column	K, AH1, UH3, L, UH3, M
combine	K, UH2, M, B, AH1, EH3, Y, N
comma	K, AH1, UH3, M, UH1
command	K, UH2, M, AE, EH3, N, D
commerce	K, AH1, UH3, M, ER, S
commercial	K, UH1, UH3, M, ER, SH, UH3, L
communicate	K, UH2, M, Y1, IU, U1, N, I3, K, A1, Y, T
company	K, UH1, UH3, M, P, EH3, N, Y
compare	K, UH1, UH3, M, P, EH3, EH3, ER
compile	K, UH1, UH3, M, P, AH1, EH3, I3, UH3, L
complete	K, UH1, UH3, M, P, L, AY, Y, T
comply	K, UH1, UH3, M, P, L, AH1, EH3, Y
component	K, UH2, M, P, O2, O1, N, EH2, N, T
computer	K, UH1, M, P, Y1, IU, U1, T, ER
conceal	K, UH1, N, S, E1, AY, L
condense	K, UH1, N, D, EH1, EH3, N, S
condition	K, UH1, N, D, I1, I3, SH, UH3, N
confirm	K, UH1, N, F, ER, R, M
confuse	K, UH1, N, F, Y1, IU, U1, U1, Z
confusion	K, UH1, N, F, Y1, IU, U1, U1, ZH, UH3, N
congratulations	K, UH1, N, G, R, AE1, D, J, UH3, L, A1, AY, SH, UH3, N, Z
connect	K, UH1, N, EH1, EH3, K, T
console	K, AH1, UH3, N, S, O1, U1, L
console-2	K, UH1, N, S, O1, O2, L
consult	K, UH1, N, S, UH1, UH2, L, T
consume	K, UH1, N, S, IU, U1, U1, M
contain	K, UH3, UH3, N, T, A1, AY, Y, N
continue	K, UH1, N, T, I1, I3, N, Y1, IU, U1
contract	K, AH1, UH3, N, T, R, AE1, EH3, K, T
contrast	K, AH1, UH3, N, T, R, AE1, EH3, S, T
control	K, UH1, N, T, R, O1, O2, L
convenient	K, UH2, N, V, E1, N, AY, EH3, N, T
copper	K, AH1, UH3, P, ER
copy	K, AH1, UH3, P, Y
correct	K, O2, O2, R, EH1, EH3, K, T
correspond	K, O1, R, I3, S, P, AH1, AH2, N, D
cosine	K, O1, U1, S, AH1, Y, N
cost	K, AW2, AW1, S, T
could	K, IU, IU, OO1, D
count	K, AH1, UH3, W, N, T
country	K, UH1, N, T, R, Y
couple	K, UH3, UH1, P, UH3, L
courage	K, ER, R, I3, D, J
course	K, O1, O2, R, S
court	K, O1, O2, R, T
cover	K, UH1, UH3, V, ER
crane	K, R, A1, AY, Y, N

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program
crash	K, R, AE1, EH3, SH
crease	K, R, E1, Y, S
create	K, R, Y, A1, Y, T
creation	K, R, Y, A1, Y, SH, UH3, N
credit	K, R, EH1, EH3, D, I1, T
crew	K, R, IU, U1, U1
critical	K, R, I1, T, I3, K, UH3, L
cross	K, R, AW, S
crowd	K, R, AH1, UH3, U1, D
cry	K, R, AH1, EH3, I3, Y
cue	(use "Q" program)
cup	K, UH1, UH2, P
curious	K, Y, ER, Y, UH1, S
current	K, ER, R, EH3, N, T
currency	K, ER, R, I2, N, DT, S, Y
curse	K, ER, R, S
curve	K, ER, R, V
customer	K, UH1, UH2, S, T, UH1, M, ER
cut	K, UH1, UH2, T
cycle	S, UH3, AH2, Y, K, UH3, L
D	D, E1, Y
daily	D, A1, AY, Y, L, Y
damage	D, AE1, EH3, M, I1, D, J
danger	D, A1, AY, Y, N, D, J, ER
dark	D, AH1, R, K
dash	D, AE1, EH3, SH
data	D, A1, Y, DT, UH1
date	D, A1, AY, Y, T
day	D, A1, I3, Y
dead	D, EH1, EH3, F
dealer	D, E1, AY, L, ER
dear	D, AY, I3, R
debit	D, EH1, EH3, B, I2, T
debt	D, EH1, EH3, T
december	D, Y, S, EH1, EH3, M, B, ER
decide	D, Y, S, AH1, EH3, Y, D
decimal	D, EH1, S, M, UH3, L
decision	D, Y, S, I1, ZH, UH3, N
decline	D, Y, K, L, AH1, EH3, Y, N
decrease	D, Y, K, R, E1, Y, S
deduct	D, Y, D, UH1, UH2, K, T
deep	D, E1, Y, P
deer	(use "dear" program)
defeat	D, Y, F, E1, AY, T
defend	D, Y, F, EH1, EH3, N, D
defensive	D, Y, F, EH1, EH3, N, S, I1, V
defer	D, E1, F, ER, R
deficit	D, EH1, F, I3, S, I1, T
degree	D, Y, G, R, E1, Y
delay	D, I1, L, EH3, A1, Y
delete	D, E1, L, E1, Y, T
deliver	D, Y, L, I1, V, ER
delta	D, EH2, EH3, L, T, UH1
demand	D, Y, M, AE1, EH3, N, D
demonstrate	D, EH1, M, UH3, N, S, T, R, A1, Y, T
deny	D, Y, N, AH1, EH3, Y
destroy	D, Y, S, T, R, O1, UH3, I3, AY
detail	D, E, T, EH3, A1, I3, UH3, L
determine	D, Y, T, ER, M, I1, N
device	D, Y, V, UH3, AH2, Y, S
dew	(use "do" program)
diagnostic	D, AH1, AY, I3, G, N, AH1, UH3, S, T, I3, K
dial	D, AH1, EH3, I3, UH3, L
dictionary	D, I1, I3, K, SH, UH3, N, EH3, EH3, ER, Y
did	D, I1, I3, D
die	D, AH1, EH3, Y
diet	D, AH1, EH3, AY, I2, T
differ	D, I1, I3, F, ER
difference	D, I1, F, R, EH3, N, DT, S
different	D, I1, F, R, EH3, N, T
digit	D, I1, D, J, I1, T
digital	D, I1, D, J, I3, T, UH3, L
dime	D, AH1, EH3, Y, M
diode	D, AH1, EH3, AY, O1, U1, D
direct	D, ER, EH1, EH3, K, T
directory	D, ER, EH1, EH3, K, T, ER, Y
dirt	D, ER, R, T
disagree	D, I1, S, UH1, G, R, E1, Y
disappear	D, I1, S, UH1, P, AY, I3, R
disconnect	D, I1, S, K, UH1, N, EH1, EH3, K, T
discuss	D, I1, I3, S, K, UH1, UH2, S
disk	D, I1, I3, S, K
display	D, I1, I3, S, P, L, A1, I3, Y
distance	D, I1, S, T, EH3, N, T, S
divide	D, I1, V, AH1, EH3, Y, D
dividend	D, I1, V, I1, D, EH1, EH3, N, D
division	D, I1, V, I1, ZH, UH3, N
do	D, IU, U1, U1
dock	D, AH1, UH3, K
doctor	D, AH1, UH3, K, T, ER
document	D, AH1, K, Y1, UH3, M, EH3, N, T
does	D, UH2, UH1, Z
dollar	D, AH1, UH3, L, ER
done	D, UH1, UH3, N

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program
door	D, O1, O2, R
double	D, UH3, UH1, B, UH3, L
doubt	D, UH3, AH2, U1, T
down	D, AH1, UH3, U1, N
draft	D, R, AE1, EH3, F, T
draw	D, R, AW
drill	D, R, I1, I3, L
drink	D, R, I1, I3, NG, K
drive	D, R, AH1, EH3, Y, V
drop	D, R, AH1, UH3, P
drum	D, R, UH1, UH2, M
dry	D, R, AH1, EH3, I3, Y
due	(use "do" program)
dump	D, UH1, UH2, M, P
duration	D, ER, R, A1, Y, SH, UH3, N
during	D, ER, R, I1, NG
duty	D, IU, U1, U1, T, Y
dwell	D, W, EH1, EH3, L
E	E1, Y
each	E1, AY, T, CH
ear	E1, I2, R
early	ER, R, L, Y
earn	ER, R, N
east	E1, AY, S, T
easy	E1, AY, Z, Y
echo	EH1, EH3, K, O1, U1
edge	EH1, EH3, D, J
edit	EH1, EH3, D, I2, T
educate	EH1, D, J, U1, K, A1, Y, T
effect	UH1, F, EH1, EH3, K, T
efficient	E1, F, I1, SH, EH3, N, T
effort	EH2, EH3, F, ER, T
eight	A2, A2, Y, T
eighth	A2, A2, Y, DT, DT, TH
eighty	A2, A2, Y, T, Y
either	E1, Y, THV, ER
electric	EH3, L, EH1, K, T, R, I2, K
electrician	EH3, L, EH1, K, PAO, T, R, I1, SH, UH3, N
electronic	EH3, L, EH1, K, T, R, AH1, N, I2, K
elevator	EH1, L, UH3, V, A2, AY, D, ER
eleven	EH1, L, EH1, EH3, V, I2, N
eligible	EH1, L, UH3, D, J, EH3, B, UH3, L
eliminate	EH1, L, I1, M, I1, N, A1, Y, T
else	EH1, EH3, L, S
emit	Y, M, I1, I3, T
employ	EH1, EH3, M, P, L, O1, UH3, I3, AY
empty	EH1, EH3, M, P, T, Y
enable	EH1, N, A1, Y, B, UH3, L
enclose	EH1, EH3, N, K, L, O1, U1, Z
end	EH1, EH3, N, D
engine	EH1, EH3, N, D, J, I1, N
engineer	EH1, N, D, J, I2, N, AY, I1, R
endorse	EH1, EH3, N, D, O2, O2, R, S
english	I1, NG, G, L, I2, SH
enter	EH1, EH3, N, T, ER
entry	EH1, EH3, N, T, R, Y
epsilon	EH1, P, S, UH3, L, AH1, UH3, N
equal	Y, K, W, UH3, L
equipment	E1, K, W, IL, P, M, EH3, N, T
erase	E1, R, A1, Y, S
error	EH3, EH3, EH3, R, ER
escape	EH1, EH3, S, K, A1, AY, Y, P
escrow	EH1, EH3, S, K, R, O1, U1
establish	UH1, S, T, AE1, EH3, B, L, I2, SH
estate	EH1, EH3, S, T, A1, AY, Y, T
estimate	EH1, S, T, EH3, M, I3, T
exact	EH1, EH3, G, PAO, Z, AE1, EH3, K, T
examine	EH1, EH3, G, PAO, Z, AE1, EH3, M, I1, N
exceed	EH1, EH3, K, PAO, S, E1, Y, D
except	EH1, EH3, K, PAO, S, EH1, EH3, P, T
exchange	EH1, EH3, K, PAO, S, T, CH, A1, AY, Y, N, D, J
execute	EH1, EH3, K, PAO, S, UH3, K, Y1, IU, U1, T
exempt	EH1, EH3, G, PAO, Z, EH1, EH3, M, P, T
exit	EH1, EH3, G, PAO, Z, I1, I3, T
expect	EH1, EH3, K, PAO, S, P, EH1, EH3, K, T
expedite	EH1, EH3, K, PAO, S, P, EH1, EH3, D, UH3, AH2, Y, T
expend	EH1, EH3, K, PAO, S, P, EH1, EH3, N, D
experiment	EH1, K, PAO, S, P, EH1, R, UH3, M, EH3, N, T
exponent	EH1, K, PAO, S, P, O2, O2, N, EH3, N, T
express	EH1, EH3, K, PAO, S, P, R, EH1, S
extension	EH1, EH3, K, PAO, S, T, EH1, EH3, N, SH, UH3, N
F	EH1, EH2, F

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program
face	F, A1, AY, Y, S
facility	F, UH2, S, I1, L, I3, T, Y
fact	F, AE1, EH3, K, T
fahrenheit	F, EH1, R, I2, N, H, UH3, AH2, Y, T
fail	F, A1, AY, I3, UH3, L
fall	F, AW, L
false	F, AW, L, S
familiar	F, UH1, M, I1, L, Y1, ER
far	F, AH1, UH3, R
farad	F, EH3, EH3, ER, AE1, EH3, D
fast	F, AE1, EH3, S, T
fault	F, AW, L, T
feat	(use "feet" program)
feature	F, E1, AY, T, CH, ER
february	F, EH1, B, Y1, IU, W, EH1, R, Y
federal	F, EH1, EH3, D, R, UH3, L
fee	F, E1, Y
feed	F, E1, Y, D
feet	F, E1, Y, T
female	F, AY, Y, M, A1, AY, UH3, L
field	F, E1, AY, UH3, L, D
fifteen	F, I1, I3, F, T, E1, Y, N
fifth	F, I1, I3, F, TH
fifty	F, I1, I3, F, T, Y
file	F, AH1, EH3, I3, UH3, L
fill	F, I1, I3, L
final	F, AH1, Y, N, UH3, L
finance	F, AH1, EH3, Y, N, AE1, EH3, N, S
find	F, AH1, EH3, Y, N, D
finger	F, I1, I3, NG, G, ER
finish	F, I1, N, I1, SH
fire	F, AH1, EH3, AY, R
first	F, ER, R, S, T
fit	F, I1, I3, T
five	F, AH1, EH3, Y, V
fix	F, I1, I3, K, PAO, S
fixture	F, I1, I3, K, PAO, S, T, CH, ER
flash	F, L, AE1, EH3, SH
flat	F, L, AE1, EH3, T
flight	F, L, UH3, AH2, Y, T
flip	F, L, I1, I3, P
floor	F, L, O1, O2, R
flop	F, L, AH1, UH3, P
flow	F, L, O1, U1
fly	F, L, AH1, EH3, Y
fold	F, O2, O2, L, L, D
follow	F, AH1, AW2, L, O1, U1
food	F, U1, U1, D
foot	F, OO1, OO1, T
for	(use "four" program)
fore	(use "four" program)
force	F, O2, O2, R, S
foreman	F, O2, O2, R, M, EH2, N
forget	F, O2, O2, R, G, EH1, EH3, T
forgive	F, O2, O2, R, G, I1, I3, V
form	F, O2, O2, R, M
format	F, O2, O2, R, M, AE1, EH3, T
forty	F, O2, O2, R, T, Y
forward	F, O2, O2, R. W, ER, D
found	F, AH1, UH3, W, N, D
four	F, O1, O2, R
fourth	F, O1, O2, R, TH
fox trot	F, AH1, UH3, K, PAO, S, T, R, AH1, UH3, T
frame	F, R, A1, AY, Y, M
fraud	F, R, AW, D
free	F, R, E1, Y
french	F, R, EH1, EH3, N, T, CH
frequency	F, R, E1, K, W, EH3, N, DT, S, Y
frequent	F, R, E1, K, W, EH3, N, T
friday	F, R, AH1, EH3, Y, D, A1, I3, Y
fright	F, R, UH3, AH2, Y, T
from	F, R, UH1, UH3, M
front	F, R, UH3, UH1, N, T
fruit	F, R, IU, U1, T
fuel	F, Y1, IU, U1, UH3, L
full	F, OO1, L
function	F, UH1, UH2, N, K, SH, UH3, N
fund	F, UH1, UH2, N, D
furnace	F, ER, R, N, EH3, S
further	F, ER, R, THV, ER
future	F, Y1, IU, U1, T, CH, ER
G	D, J, E1, Y
gage	(use "gauge" program)
gain	G, A1, AY, Y, N
gait	(use "gate" program)
gallon	G, AE1, AH2, L, UH3, N
game	G, A1, AY, Y, M
gamma	G, AE1, EH3, M, UH2, UH3
gap	G, AE1, EH3, P
garage	G, UH1, R, AH1, UH3, ZH
gas	G, AE1, EH3, S
gate	G, A1, AY, Y, T
gauge	G, A1, AY, Y, D, J
general	D, J, EH1, EH3, N, ER, UH3, L
generate	D, J, EH1, N, ER, A1, Y, T
gentlemen	D, J, EH1, EH3, N, T, L, M, I2, N

Word	Phonetic Program
german	D, J, ER, R, M, EH2, N
get	G, EH1, EH3, T
girl	G, ER, R, L
give	G, I1, I3, V
glass	G, L, AE1, EH3, S
glitch	G, L, I1, I3, T, CH
globe	G, L, O1, U1, B
go	G, OO1, O1, U1
golf	G, AW2, AW2, UH3, L, F
good	G, OO1, OO1, D
govern	G, UH1, UH3, V, ER, N
grade	G, R, A1, AY, Y, D
gram	G, R, AE1, EH3, M
grand	G, R, AE1, EH3, N, D
graph	G, R, AE1, EH3, F
grate	(use "great" program)
gray	(use "grey" program)
great	G, R, A1, Y, T
green	G, R, E1, Y, N
greet	G, R, E1, Y, T
grey	G, R, A1, AY, Y
grind	G, R, AH1, EH3, Y, N, D
grocery	G, R, O1, U1, S, ER, Y
ground	G, R, AH1, UH3, W, N, D
group	G, R, U1, U1, P
grow	G, R, O1, U1
guard	G, AH1, R, D
guarantee	G, EH1, R, I3, N, T, E1, Y
guess	G, EH1, EH3, S
H	A1, AY, Y, T, CH
had	H, AE1, EH3, D
half	H, AE1, EH3, F
halt	H, AW, L, T
hammer	H, AE1, EH3, M, ER
hand	H, AE1, EH3, N, D
handle	H, AE1, EH3, N, D, UH3, L
hang	H, AE1, I3, NG
happy	H, AE1, EH3, P, Y
hard	H, AH1, R, D
has	H, AE1, EH3, Z
have	H, AE1, EH3, V
he	H, E1, Y
head	H, EH1, EH3, D
hear	H, AY, I3, R
heart	H, AH1, UH3, R, T
heat	H, E1, AY, T
heavy	H, EH1, V, Y
height	H, UH3, AH2, Y, T
held	H, EH1, UH3, L, D
hello	H, EH1, UH3, L, UH3, O1, U1
help	H, EH1, EH3, L, P
henry	H, EH1, EH3, N, R, Y
her	H, ER
here	(use "hear" program)
hertz	H, R, R, T, S
hex	H, EH1, EH3, K, PAO, S
high	H, AH1, EH3, Y
his	H, I1, I3, Z
hold	H, O2, O2, L, L, D
hole	H, O1, U1, L
home	H, O1, U1, M
hook	H, OO1, OO1, K
host	H, O1, U1, S, T
hot	H, AH1, UH3, T
hotel	H, O1, U1, T, EH2, EH2, L
hour	AH1, UH3, W, ER
house	H, UH3, AH2, U1, S
how	H, AH1, O2, U1
human	H, Y1, IU, U1, U1, M, EH2, N
hundred	H, UH1, UH2, N, D, R, I3, D
hungry	H, UH1, UH2, NG, G, R, Y
I	AH1, EH3, I3, Y
idle	AH1, Y, D, UH3, L
idol	(use "idle" program)
if	I1, I3, F
immediate	I1, I3, M, E1, D, Y, EH3, T
important	I1, I3, M, P, O2, O2, R, T, EH3, N, T
improper	I1, I3, M, P, R, AH1, UH3, P, ER
improve	I1, I3, M, P, R, IU, U1, U1, V
in	I1, I3, N
inch	I1, I3, N, T, CH
include	I1, I3, N, K, L, IU, U1, U1, D
income	I1, I3, N, K, UH1, UH3, M
independent	I1, N, D, E1, P, EH2, EH3, N, D, EH3, N, T
index	I1, I3, N, D, EH1, EH3, K, PAO, S
india	I2, I3, N, D, Y, UH2
indicate	I1, N, D, I3, K, A1, Y, T
industrial	I1, I3, N, D, UH1, UH2, S, T, R, AY, UH3, L
inform	I1, I3, N, F, O2, O2, R, M
initial	I1, I3, N, I1, SH, UH3, L
inn	(use "in" program)
input	I1, I3, N, P, OO1, OO1, T
inquire	I1, I3, N, K, W, AH1, EH3, AY, R
insert	I1, N, S, R, R, T
inspect	I1, I3, N, S, P, EH1, EH3, K, T

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program
install	I1, I3, N, S, T, AW, L
instead	I1, I3, N, S, T, EH1, EH3, D
instruct	I1, I3, N, S, T, R, UH1, UH2, K, T
instrument	I1, I3, N, S, T, R, UH1, M, EH1, EH3, N, T
insufficient	I1, N, S, UH2, F, I1, SH, EH3, N, T
insurance	I1, I3, N, SH, ER, R, EH3, N, T, S
interest	I1, N, T, R, EH1, S, T
interface	I1, I3, N, T, ER, F, A1, AY, Y, S
interpret	I1, I3, N, T, ER, P, R, EH3, T
interupt	I1, N, T, ER, UH3, UH1, P, T
intrude	I1, I3, N, T, R, IU, U1, U1, D
invalid	I1, I3, N, V, AE1, AW2, L, I1, D
invent	I1, I3, N, V, EH1, EH3, N, T
inventory	I1, N, V, EH1, N, T, O1, R, Y
invest	I1, I3, N, V, EH1, EH3, S, T
invoice	I1, I3, N, V, O1, UH3, I3, AY, S
irregular	I1, R, EH1, G, Y1, UH3, L, ER
is	I1, I3, Z
it	I1, I3, T
item	AH2, UH3, Y, D, UH3, M
J	D, J, EH3, A1, AY, Y
jack	D, J, AE1, EH3, K
january	D, J, AE1, EH3, N, Y1, UI, EH3, EH3, ER, Y
job	D, J, AH1, UH3, B
join	D, J, O1, UH3, I3, AY, N
jolt	D, J, O2, O2, L, T
joy	D, J, O1, UH3, I3, AY
judge	D, J, UH1, UH2, D, J
juliet	D, J, IU, U1, L, Y, EH2, EH3, T
july	D, J, UH1, L, AH1, EH3, Y
jump	D, J, UH1, UH2, M, P
june	D, J, IU, U1. U1, N
K	K, EH3, A1, AY, Y
keep	K, E1, Y, P
key	K, E1, Y
keyboard	K, AY, Y, B, O1, O2, R, D
kill	K, I1, I3, L
kilo	K, E1, AY, L, UH3, O2, U1
knew	(use "new" program)
knot	(use "not" program)
know	(use "no" program)
knowledge	N, AH1, UH3, L, I3, D, J
L	EH1, EH3, UH3, L
lab	L, AE1, EH3, B
labor	L, A1, Y, B, ER
language	L, AE1, EH3, NG, G, W, I1, D, J
lapse	L, AE1, EH3, P, S
large	L, AH1, R, D, J
last	L, AE1, EH3, S, T
late	L, A1, AY, Y, T
law	L, AW
lead	L, E1, Y, D
led	L, EH1, EH3, D
left	L, EH1, EH3, F, T
leg	L, EH1, EH3, G
legal	L, E1, G, UH3, L
lend	L, EH1, EH3, N, D
length	L, EH1, EH3, NG, TH
less	L, EH1, EH3, S
let	L, EH1, EH3, T
letter	L, EH1, EH3, T, ER
level	L, EH1, EH3, V, UH3, L
life	L, UH3, AH2, Y, F
light	L, UH3, AH2, Y, T
like	L, UH3, AH2, Y, K
lima	L, AY, Y, M, UH1
limit	L, I1, M I1, T
line	L, AH1, EH3, Y, N
linear	L, I2, I3, N, AY, Y, ER
link	L, I1, I3, NG, K
lip	L, I1, I3, P
liquid	L, I1, K, W, I1, D
list	L, I1, I3, S, T
listen	L, I1, I3, S, I2, N
little	L, I1, I3, T, UH3, L
load	L, UH3, 01, U1, D
loan	L, UH3, 01, U1, N
local	L, 02, 02, K, UH3, L
lock	L, AH1, UH3, K
log	L, AW, G
long	L, AW, NG
look	L, 001, 001, K
loss	L, AW, S
lost	L, AW, S, T
lot	L, AH1, UH3, T
low	L, 01, U1
M	EH1, EH2, M
machine	M, UH2, SH, E1, Y, N
mail	(use "male" program)
maintenance	M, A1, Y, N, T, EH2, N, EH3, N, DT, S
make	M, A1, AY, Y, K
male	M, A2, A2, AY UH3, L
man	M, AE1, EH3, N

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program
manage	M, AE1, EH3, N, I1, D, J
manual	M, AE1, EH3, N, Y1, U1, UH3, L
manufacture	M, AE1, EH3, N, Y1, U1, F, AE1, EH3, K, T, CH, ER
many	M, EH2, EH2, N, Y
map	M, AE1, EH3, P
march	M, AH1, R, T, CH
margin	M, AH1, UH3, R, D, J, I2, N
mark	M, AH1, R, K
market	M, AH1, R, K, EH3, T
match	M, AE1, EH3, T, CH
mature	M, UH1, T, CH, IU, ER
maximum	M, AE1, EH3, K, PAO, S, EH3, M, UH2, M
may	M, A1, I3, Y
me	M, E1, Y
measure	M, EH3, EH1, ZH, ER
meat	M, E1, AY, T
mechanical	M, UH1, K, AE1, EH3, N, I3, K, UH3, L
media	M, E1, AY, D, Y, UH1
medicine	M, EH2, EH3, D, I3, S, I1, N
medium	M, E1, D, AY, UH1, M
meet	(use "meat" program)
mega	M, EH1, EH3, G, UH2, UH3
member	M, EH1, EH3, M, B, ER
memory	M, EH1, EH3, M, ER, Y
men	M, EH1, EH3, N
merchandise	M, ER, T, CH, EH3, N, D, AH1, EH3, Y, Z
merge	M, ER, R, D, J
message	M, EH1, EH3, S, I2, D, J
metal	M, EH1, EH3, T, UH3, L
meter	M, E1, Y, T, ER
micro	M, UH3, AH2, AY, K, R, 01, U1
middle	M, I1, I3, D, UH3, L
mike	M, UH3, AH2, Y, K
mile	M, AH1, EH3, I3, UH3, L
mill	M, I1, I3, L
milli	M, I1, I3, L, UH3
million	M, I1, I3, L, Y, UH3, N
mini	M, I2, I2, N, Y
minus	M, AH1, Y, N, EH3, S
minute	M, I1, N, EH3, T
miscellaneous	M, I1, S, UH3, L, A1, AY, N, Y, UH3, S
miss	M, I1, I3, S
mistake	M, I1, I3, S, T, A1, AY, Y, K
mode	M, 01, U1, D
model	M, AH1, UH3, D, UH3, L
module	M, AH1, UH3, D, J, IU, U1, UH3, L
monday	M, UH3, UH1, N, D, A1, I3, Y
money	M, UH3, UH1, N, AY, Y
month	M, UH3, UH1, N, TH
more	M, 02, 02, R
morning	M, 02, 02, R, N, I1, I3, NG
most	M, 01, U1, S, T
motor	M, 01, U1, T, ER
mount	M, AH1, UH3, W, N, T
move	M, U1, U1, V
Mr.	M, I1, S, T, ER
Mrs.	M, I1, S, I2, Z
Ms.	M, I1, I3, Z
much	M, UH1, UH2, T, CH
multi	M, UH2, UH3, L, T, Y
multiple	M, UH1, L, T, EH3, P, UH3, L
multiply	M, UH1, L, T, I3, P, L, AH1, Y
N	EH1, EH2, N
name	N, A1, AY, Y, M
nano	N, AE1, EH3, N, 01, U1
national	N, AE1, EH3, SH, UH3, N, UH3, L
native	N, A1, Y, T, I1, V
near	N, AY, I1, R
neat	N, E1, AY, T
neck	N, EH1, EH3, K
need	N, E1, Y, D
negative	N, EH1, G, EH3, T, I1, V
net	N, EH1, EH3, T
neutral	N, IU, U1, T, R, UH2, L
new	N, IU, U1, U1
next	N, EH1, EH3, K, PAO, S, T
nice	N, UH3, AH2, Y, S
nickel	N, I1, I3, K, UH3, L
night	N, UH3, AH2, Y, T
nine	N, AH1, EH3, Y, N
ninety	N, AH1, EH3, Y, N, T, Y
nineth	N, AH1, Y, N, DT, TH
no	N, 001, 01, U1
noise	N, 01, UH3, I3, AY, Z
none	N, UH1, UH3, N
noon	N, IU, U1, U1, N
normal	N, 02, 02, R, M, UH3, L
north	N, 02, 02, R, TH
not	N, AH1, UH3, T
note	N, 01, U1, T
nothing	N, UH1, TH, I1, I3, NG
notice	N, 01, U1, T, I1, S

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program
notify	N, 01, U1, T, I1, F, AH1, EH3, Y
november	N, 01, U1, V, EH1, EH3, M, B, ER
now	N, AH1, UH3, U1
number	N, UH1, UH2, M, B, ER
nurse	N, ER, R, S
nut	N, UH1, UH2, T
O	02, 01, U1
oar	(use "or" program)
object	UH1, B, D, J, EH1, EH3, K, T
object-2	AH1, UH3, B, D, J, EH2, EH2, K, T
obligation	AH1, B, L, I3, G, A1, Y, SH, UH3, N
obsolete	AH1, UH3, B, S, UH3, L, AY, Y, T
october	AH1, UH3, K, T, O1, U1, B, ER
odd	AH1, UH3, D
of	UH1, UH3, V
off	AW, F
office	AW, F, I1, S
official	UH1, F, I1, SH, UH3, L
often	AW2, AW2, F, I3, N
ohm	O2, O2, U1, M
oil	O1, EH3, I3, UH3, L
old	O2, O2, L, L, D
omega	O1, U1, M, A1, Y, G, UH2
omit	O1, U1, M, I1, I3, T
on	AH1, UH3, N
once	W, UH1, N, T, S
one	W, UH1, UH2, N
only	O1, O2, N, L, Y
open	O1, P, I2, N
operable	AH1, UH3, P, ER, UH3, B, UH3, L
operate	AH1, UH3, P, ER, A1, Y, T
operator	AH1, UH3, P, ER, A1, Y, T, ER
option	AH1, UH3, P, SH, UH3, N
or	O2, O2, R
orange	O2, O2, R, I1, N, D, J
order	O2, O2, R, D, ER
ore	(use "or" program)
original	O2, R, I2, I3, D, J, I3, N, UH3, L
oscar	AH1, UH3, S, K, ER
other	UH1, UH3, THV, ER
ounce	AH1, UH3, W, N, S
out	UH3, AH2, U1, T
oven	UH1, V, I2, N
over	O1, O2, V, ER
oxygen	AH1, UH3, K, PAO, S, I3, D, J, I2, N
own	O1, U1, N
P	P, E1, Y
pack	P, AE1, EH3, K
package	P, AE1, EH3, K, I1, D, J
paid	P, A1, AY, Y, D
pain	P, A1, AY, Y, N
pane	(use "pain" program)
panel	P, AE1, EH3, N, UH3, L
papa	P, AH1, UH3, P, UH3, UH3
paper	P, A1, Y, P, ER
parcel	P, AH1, R, S, UH3, L
paren	P, EH3, EH3, ER, I2, N
part	P, AH1, R, T
partial	P, AH1, R, SH, UH2, L
pass	P, AE1, EH3, S
passed	(use "past" program)
past	P, AE1, EH3, S, T
pat	P, AE1, EH3, T
pattern	P, AE1, EH3, T, ER, N
pause	P, AW, Z
pay	P, A2, A2, AY, Y
pea	(use "P" program)
peace	(use "piece" program)
peak	P, E1, AY, K
peek	(use "peak" program)
percent	P, ER, S, EH1, EH3, N, T
period	P, I1, R, Y, UH2, D
permanent	P, ER, M, EH2, N, EH1, N, T
person	P, ER, S, UH1, N
personal	P, ER, S, UH3, N, UH2, L
personality	P, ER, S, UH3, N, AE1, UH3, L, I3, T, Y
phase	F, A1, AY, Y, Z
phone	F, O1, U1, N
pick	P, I1, I3, K
pico	P, E1, Y, K, O2, U1
piece	P, E1, Y, S
pint	P, AH1, Y, N, T
pipe	P, UH3, AH2, Y, P
place	P, L, A1, AY, Y, S
plain	(use "plane" program)
plan	P, L, AE1, EH3, N
plane	P, L, A1, AY, Y, N
plant	P, L, AE1, EH3, N, T
play	P, L, A1, I3, Y
please	P, L, E1, Y, Z
plot	P, L, AH1, UH3, T
plus	P, L, UH1, UH2, S
pocket	P, AH1, UH3, K, EH3, T
point	P, O1, UH3, I3, AY, N, T

Word	Phonetic Program
poke	P, O1, U1, K
police	P, UH1, L, AY, Y, S
plain	(use "plane" program)
plan	P, L, AE1, EH3, N
plane	P, L, A1, AY, Y, N
plant	P, L, AE1, EH3, N, T
play	P, L, A1, I3, Y
please	P, L, E1, Y, Z
plot	P, L, AH1, UH3, T
plus	P, L, UH1, UH2, S
pocket	P, AH1, UH3, K, EH3, T
point	P, O1, UH3, I3, AY, N, T
poke	P, O1, U1, K
police	P, UH1, L, AY, Y, S
policy	P, AH1, UH3, L, I3, S, Y
poor	(use "pour" program)
pop	P, AH1, UH3, P
port	P, 02, 02, R, T
position	P, UH1, Z, I1, SH, UH3, N
positive	P, AH1, UH3, Z, I1, T, I1, V
possible	P, AH1, UH3, S, UH3, B, UH2, L
post	P, 01, U1, S, T
potential	P, 01, T, EH1, EH3, N, T, CH, UH3, L
pound	P, AH1, UH3, W, N, D
pour	P, 01, 02, R
power	P, AH1, UH3, W, ER
practice	P, R, AE1, EH3, K, T, I1, S
premium	P, R, AY, Y, M, Y, UH1, M
prepare	P, R, E1, P, EH1, EH3, R
press	P, R, EH1, EH3, S
pressure	P, R, EH1, SH, ER
prevent	P, R, Y, V, EH1, EH3, N, T
previous	P, R, Y, V, Y, UH1, S
price	P, R, UH3, AH2, Y, S
principal	(use "principle" program)
principle	P, R, I1, N, DT, S, UH3, P, UH3, L
print	P, R, I1, I3, N, T
prior	P, R, AH1, Y, ER
priority	P, R, AH1, Y, 01, R, I3, DT, Y
private	P, R, AH1, EH3, Y, V, I3, T
probe	P, R, 01, U1, B
problem	P, R, AH1, UH3, B, L, UH3, M
procedure	P, R, UH1, S, E1, D, J, ER
proceed	P, R, 01, S, E1, Y, D
process	P, R, AH1, UH3, S, EH1, EH3, S
produce	P, R, UH1, D, IU, U1, U1, S
product	P, R, AH1, UH3, D, UH1, UH2, K, T
progress	P, R, AH1, UH3, G, R, EH1, S
profession	P, R, UH1, F, EH1, EH3, SH, UH3, N
profit	P, R, AH1, UH3, F, I1, T
program	P, R, 01, G, R, AE1, EH3, M
project	P, R, AH1, UH3, D, J, EH2, EH2, K, T
PROM	P, R, AH1, UH3, M
promote	P, R, UH1, M, 01, U1, T
propose	P, R, UH1, P, 01, U1, Z
protect	P, R, UH1, T, EH1, EH3, K, T
public	P, UH1, UH3, B, L, I3, K
pull	P, 001, 001, L
pulse	P, UH1, UH2, L, S
punch	P, UH1, UH2, N, T, CH
purpose	P, R, R, P, EH2, S
purchase	P, R, R, DT, CH, I2, S
pure	P, Y1, IU, ER
push	P, 001, IU, SH
put	P, 001, 001, T
Q	K, Y1, IU, U1, U1
qualify	K, W, AW1, L, I1, F, AH1, EH3, Y
quantity	K, W, AH1, N, T, I3, T, Y
quart	K, W, 01, R, T
quarter	K, W, 01, R, T, ER
quebec	K, W, I1, B, EH1, EH3, K
question	K, W, EH1, EH3, S, T, CH, UH3, N
quick	K, W, I1, I3, K
quiet	K, W, AH1, EH3, AY, I2, T
quit	K, W, I1, I3, T
quiz	K, W, I1, I3, Z
quota	K, W, 01, 02, T, UH1
quote	K, W, 01, U1, T
R	AH1, UH2, ER
rail	R, A1, AY, I3, UH3, L
rain	R, A1, AY, Y, N
raise	R, A1, AY, Y, Z
range	R, A1, AY, Y, N, D, J
radio	R, A1, Y, D, Y, 01, U1
rate	R, A1, AY, Y, T
ratio	R, A1, Y, SH, Y, 01, U1
reach	R, E1, Y, T, CH
read	R, E1, Y, D
ready	R, EH1, EH3, D, Y
real	R, E1, AY, L
reason	R, E1, Y, Z, UH1, N
rebate	R, E1, B, A1, Y, T

Word	Phonetic Program	Word	Phonetic Program
recall	R, E1, K, AW2, AW1, L	root	R, U1, U1, T
receipt	R, E1, S, AY, Y, T	round	R, AH1, UH3, W, N, D
receive	R, E1, S, E1, Y, V	route	R, UH2, AH2, U1, T
record	R, E1, K, 02, 02, R, D	row	R, 01, U1
record-2	R, EH1, EH3, K, ER, D	run	R, UH1, UH3, N
red	R, EH1, EH3, D	rush	R, UH1, UH2, SH
reel	(use "real" program)		
refer	R, E1, F, UH1, UH2, N, D	S	EH1, EH2, S
refuse	R, E1, F, Y1, IU, U1 U1, Z	safe	S, A1, AY, Y, F
register	R, EH1, D, J, I1, S, T, ER	sail	(use "sale" program)
regular	R, EH1, G, Y1, IU, L, ER	salary	S, AE1, AH2, L, UH3, R, Y
rein	(use "rain" program)	sale	S, A1, A2, AY, UH3, L
reject	R, E1, D, J, EH1, EH3, K, T	same	S, A1, AY, Y, M
relay	R, E1, L, A1, I3, Y	saturday	S, AE1, EH3, T, ER, D, A1, Y
release	R, E1, L, E1, AY, S	save	S, A1, AY, Y, V
remain	R, E1, M, A1, AY, Y, N	say	S, A1, I3, Y
remove	R, E1, M, U1, U1, V	scan	S, K, AE1, EH3, N
repair	R, E1, P, EH2, EH2, R	scent	(use "cent" program)
repeat	R, E1, P, E1, AY, T	schedule	S, K, EH1, EH3, D, J, IU, U1, L
replace	R, E1, P, L, A1, AY, Y, S	school	S, K, U1, U1, L
report	R, E1, P, 02, 02, R, T	science	S, AH1, I3, Y, EH3, N, DT, S
represent	R, EH1, P, R, I2, Z, EH1, EH3, N, T	score	S, K, O2, O2, R
		scrap	S, K, R, AE1, EH3, P
request	R, E1, K, W, EH1, EH3, S, T	screw	S, K, R, IU, U1, U1
require	R, E1, K, W, AH1, EH3, AY, R	sea	(use "C" program)
requisition	R, EH1, K, W, I2, Z, I1, SH, UH3, N	seat	S, E1, AY, T
		second	S, EH1, EH3, K, UH1, N, D
rescue	R, EH1, EH3, S, K, Y1, IU, U1	secret	S, E1, K, R, I3, T
resemble	R, E1, Z, EH1, EH3, M, B, UH3, L	section	S, EH1, EH3, K, SH, UH3, N
		security	S, EH1, EH3, K, Y, ER, I1, T, Y
reset	R, E1, S, EH1, EH3, T	see	(use "C" program)
resistor	R, E1, Z, I1, S, T, ER	seize	S, E1, Y, Z
respect	R, E1, S, P, EH1, EH3, K, T	select	S, UH1, L, EH1, EH2, K, T
respond	R, E1, S, P, AH1, UH3, N, D	sell	S, EH1, EH3, L
responsible	R, I2, S, P, AH1, UH3, N, DT, S, UH3, B, UH3, L	semi	S, EH1, M, AH1, Y
		semicolon	S, EH1, M, AH1, Y, K, OO1, O1, L, I2, N
rest	R, EH1, EH3, S, T		
restrict	R, E1, S, T, R, I1, I3, K, T	send	S, EH1, EH3, N, D
result	R, E1, Z, UH1, UH2, L, T	sent	(use "cent" program)
resume	R, E1, Z, IU, U1, U1, M	sentence	S, EH1, N, T, I2, N, DT, S
retail	R, AY, E1, T, EH3, A1, I3, UH3, L	separate	S, EH1, EH3, P, UH1, R, A1, AY, T
retain	R, E1, T, A1, AY, Y, N	separate-2	S, EH1, EH3, P, R, I2, T
return	R, E1, T, ER, R, N	september	S, EH1, EH3, P, T, EH1, EH3, M, B, ER
revision	R, E1, V, I1, ZH, UH3, N		
revolve	R, E1, V, AH1, UH3, L, V	sequence	S, E1, K, W, EH1, EH3, N, S
ribbon	R, I2, I3, B, UH3, N	serial	S, I1, R, Y, UH3, L
right	R, UH3, AH2, Y, T	series	S, I1, R, Y, Z
romeo	R, 01, U1, M, Y, 01, U1	service	S, ER, V, I1, S
room	R, U1, U1, M	set	S, EH1, EH3, T

Adapted from *Phonetic Speech Dictionary for the SC-01 Speech Synthesizer,* copyright Votrax® 1981. Reproduced by permission.

Word	Phonetic Program	Word	Phonetic Program
seven	S, EH1, EH3, V, I2, N	speed	S, P, E1, Y, D
seventh	S, EH1, EH3, V, I2, N, DT, TH	speech	S, P, E1, Y, T, CH
seventy	S, EH1, V, I2, N, D, Y	spell	S, P, EH1, EH3, L
several	S, EH1, V, ER, UH3, L	spend	S, P, EH1, EH3, N, D
sew	(use "so" program)	split	S, P, L, I1, I3, T
share	SH, EH3, EH3, ER	spoon	S, P, U1, U1, N
sharp	SH, AH1, R, P	spring	S, P, R, I1, I3, NG
shift	SH, I1, I3, F, T	square	S, K, W, EH1, R
ship	SH, I1, I3, P	stack	S, T, AE1, EH3, K
shop	SH, AH1, UH3, P	stair	(use "stare" program)
short	SH, O2, O2, R, T	stand	S, T, AE1, EH3, N, D
should	SH, IU, IU, D	standard	S, T, AE1, EH3, N, D, ER, D
shunt	SH, UH1, UH2, N, T	star	S, T, AH1, UH3, R
shut	SH, UH1, UH2, T	stare	S, T, EH3, EH3, ER
side	S, AH1, EH3, Y, D	start	S, T, AH1, R, T
sierra	S, E1, I3, EH1, R, UH1	state	S, T, A1, AY, Y, T
signal	S, I1, I3, G, N, UH3, L	station	S, T, A1, Y, SH, UH3, N
silver	S, I1, I3, L, V, ER	status	S, T, AE1, EH3, T, I2, S
single	S, I1, I3, NG, G, UH3, L	steal	(use "steel" program)
six	S, I1, I3, K, PAO, S	steel	S, T, E1, Y, L
sixth	S, I1, I3, K, PAO, S, TH	step	S, T, EH1, EH3, P
sixty	S, I1, I3, K, PAO, T, Y	stick	S, T, I1, I3, K
size	S, AH1, EH3, Y, Z	stock	S, T, AH1, UH3, K
skin	S, K, I1, I3, N	stop	S, T, AH1, UH3, P
sky	S, K, AH1, EH3, I3, Y	store	S, T, O2, O2, R
slang	S, L, AE1, EH3, NG	strait	(use "straight" program)
slash	S, L, AE1, EH3, SH	straight	S, T, R, A1, AY, Y, T
slave	S, L, A1, AY, Y, V	street	S, T, R, E1, Y, T
slip	S, L, I1, I3, P	stress	S, T, R, EH1, EH3, S
slow	S, L, O1, U1	string	S, T, R, I1, I3, NG
small	S, M, AW, L	structure	S, T, R, UH1, K, T, CH, ER
smell	S, M, EH1, EH3, L	style	S, T, AH1, EH3, AY, UH3, L
smile	S, M, AH1, EH3, I3, UH3, L	subject	S, UH1, UH2, B, D, J, EH1, EH3, K, T
smoke	S, M, O1, U1, K	substitute	S, UH1, UH3, B, S, T, I3, T, IU, U1, T
snow	S, N, OO1, O2, U1	subtract	S, UH1, UH2, B, T, R, AE1, EH3, K, T
so	S, OO1, O2, U1	sufficient	S, UH1, F, I1, SH, EH3, N, T
soft	S, AW, F, T	suggest	S, UH1, UH2, G, D, J, EH1, EH3, S, T
sold	S, O2, O2, L, L, D	suit	S, IU, U1, T
solid	S, AH1, UH3, L, I1, D	suite	S, W, AY, Y, T
son	(use "sun" program)	sum	S, UH1, UH2, M
some	(use "sum" program)	summary	S, UH2, UH2, M, ER, Y
sorry	S, AW, R, Y	summer	S, UH1, UH2, M, ER
sort	S, O2, O2, R, T	sun	S, UH1, UH2, N
sound	S, AH1, UH3, W, N, D	sunday	S, UH1, UH2, N, D, A1, I3, Y
source	S, O1, O2, R, S	super	S, IU, U1, P, ER
south	S, AH1, UH3, U1, TH	supply	S, UH2, P, L, AH1, Y
space	S, P, A1, AY, Y, S		
spark	S, P, AH1, R, K		
speak	S, P, E1, AY, K		
special	S, P, EH1, EH3, SH, UH3, L		

Word	Phonetic Program
surface	S, ER, F, I2, S
surge	S, ER, R, D, J
surgery	S, ER, D, J, ER, Y
surgical	S, ER, D, J, UH3, K, UH3, L
surplus	S, ER, P, L, UH1, S
suspend	S, UH1, S, P, EH1, EH3, N, D
sweep	S, W, E1, Y, P
sweet	(use "suite" program)
switch	S, W, I1, I3, T, CH
syntax	S, I1, N, T, AE1, EH3, K, PAO, S
system	S, I1, S, T, UH3, M
T	T, E1, AY, Y
table	T, A1, Y, B, UH3, L
tail	(use "tale" program)
tale	T, A1, Y, UH3, L
talk	T, AW, K
tangent	T, AE1, EH3, N, D, J, EH3, N, T
target	T, AH1, UH3, R, G, I2, T
tea	(use "T" program
team	T. E1, Y, M
technical	T, EH1, EH3, K, N, I3, K, UH3, L
tee	(use "T" program)
temperature	T, EH1, EH3, M, P, ER, UH1, T, CH, ER
ten	T, EH1, EH3, N
terminal	T, ER, M, EH3, N, UH2, L
test	T, EH1, EH3, S, T
than	THV, EH1, EH3, N
the	THV, UH1, UH3
then	(use "than" program)
theory	TH, AY, I2, R, Y
thin	TH, I1, I3, N
thing	TH, I1, I3, NG
think	TH, I1, I3, NG, K
third	TH, ER, R, D
thirteen	TH, ER, T, T, E1, Y, N
thirty	TH, ER, R, D, Y
thousand	TH, AH1, UH3, U1, Z, EH3, N, D
three	TH, R, E1, Y
threw	(use "through" program)
through	TH, R, IU, U1
thursday	TH, ER, R, Z, D, A1, I3, Y
ticket	T, I1, I3, K, EH3, T
till	T, I1, I3, L
time	T, AH1, EH3, Y, M
tire	T, AH1, EH3, AY, R
title	T, UH3, AH2, Y, T, UH3, L
to	(use "two" program)
today	T, U1, D, A1, I3, Y
toilet	T, O1, EH3, I3, L, I3, T
toll	T, O2, O2, OO1, L
tomorrow	T, U1, M, AH1, R, O1, U1
ton	T, UH1, UH2, N, N
tone	T, O1, U1, N
too	(use "two" program)
tool	T, U1, U1, L
total	T, O1, U1, T, UH3, L
touch	T, UH1, UH3, T, CH
towel	T, AH1, W, UH3, L
trace	T, R, A1, AY, Y, S
trade	T, R, A1, AY, Y, D
train	T, R, A1, AY, Y, N
transact	T, R, AE1, EH3, N, S, AE1, EH3, K, T
transfer	T, R, AE1, EH3, N, S, F, ER
transistor	T, R, AE1, N, Z, I1, S, T, ER
transmit	T, R, AE1, EH3, N, Z, M, I1, I3, T
transport	T, R, AE1, EH3, N, S, P, O2, O2, R, T
transportation	T, R, AE1, N, S, P, ER, T, A1, AY, SH, UH3, N
travel	T, R, AE1, EH3, V, UH3, L
triangle	T, R, AH1, I3, AE1, EH3, NG, G, UH3, L
trouble	T, R, UH3, UH1, B, UH3, L
truck	T, R, UH1, UH2, K
true	T, R, IU, U1, U1
trust	T, R, UH1, UH2, S, T
try	T, R, AH1, EH3, I3, Y
tuesday	T, IU, U1, U1, Z, D, A1, Y
tune	T, IU, U1, U1, N
turn	T, ER, R, N
twelve	T, W, EH1, EH3, UH3, L, V
twenty	T, W, EH1, EH3, N, T, Y
two	T, IU, U1, U1
type	T, UH3, AH2, Y, P
U	Y1, IU, U1, U1
ultra	UH3, UH2, L, T, R, UH1
under	UH2, UH2, N, D, ER
uniform	Y1, IU, U1, N, I3, F, O1, R, M
until	UH2, UH2, N, T, I1, I3, L
up	UH1, UH2, P
urgent	R, R, D, J, I3, N, T
us	UH1, UH2, S
use	Y1, IU, U1, U1, Z
use-2	Y1, IU, U1, S
V	V, E1, AY, Y

Word	Phonetic Program
vacant	V, A1, Y, K, EH3, N, T
valid	V, AE1, UH3, L, I1, D
vary	(use "very" program)
value	V, AE1, EH3, L, Y1, IU, U1
vendor	V, EH1, EH3, N, D, ER
vent	V, EH1, EH3, N, T
verify	V, EH1, R, I3, F, AH1, EH3, Y
very	V, EH1, R, Y
via	V, E1, AY, UH2, UH3
victor	V, I1, I3, K, T, ER
voice	V, O1, UH3, I3, AY, S
void	V, O1, UH3, I3, AY, D
volt	V, O2, O2, L, T
volume	V, AH1, UH3, L, Y1, IU, U1, M
W	D, UH1, B, UH3, L, Y1, IU, U1
wage	W, A1, AY, Y, D, J
wait	W, A1, AY, Y, T
want	W, AH1, UH3, N, T
was	W, UH1, UH3, Z
wash	W, AW, SH
water	W, AH1, UH3, T, ER
watt	W, AH1, UH3, T
wave	W, A1, AY, Y, V
way	(use "weigh" program)
we	W, E1, Y
weak	(use "week" program)
weapon	W, EH2, EH2, P, UH1, N
wear	(use "where" program)
wednesday	W, EH1, N, Z, D, A1, I3, Y
week	W, E1, Y, K
weigh	W, A2, A2, Y
weight	(use "wait" program)
went	W, EH1, EH3, N, T
west	W, EH1, EH3, S, T
wet	W, EH1, EH3, T
what	W, UH3, UH1, T
wheel	W, E1, Y, L
when	W, EH1, EH3, N
where	W, EH3, A2, EH3, R
which	W, I1, I3, T, CH
while	W, AH1, EH3, I1, UH3, L
whiskey	W, I1, I3, S, K, AY, Y
white	W, UH3, AH2, Y, T
who	H, IU, U1, U1
whole	(use "hole" program)
why	(use "Y" program)
will	W, I1, I3, L
window	W, I1, N, D, O1, U1
winter	W, I1, I3, N, T, ER
wire	W, AH1, EH3, AY, R
with	W, I1, I3, TH
withdraw	W, I1, I3, TH, D, R, AW
without	W, I1, I3, TH, UH2, AH2, U1, T
won	(use "one" program)
word	W, ER, R, D
work	W, ER, R, K
write	(use "right" program)
wrong	R, AW, NG
X	EH1, EH2, K, PAO, S
x-ray	EH1, EH2, K, PAO, S, R, A1, I3, Y
Y	W, AH1, EH3, I3, Y
yankee	Y1, AE1, EH3, NG, K, E1, Y
yard	Y1, AH1, R, D
year	Y1, AY, I3, R
yellow	Y1, EH1, EH3, L, O1, U1
yes	Y1, EH3, EH1, S
yesterday	Y1, EH3, EH1, S, T, ER, D, A1, I3, Y
yet	Y1, EH1, EH3, T
you	(use "U" program)
your	Y, O2. O2, R
you're	(use "your" program)
Z	Z, E1, Y
zap	Z, AE1, EH3, P
zero	Z, AY, I1, R, O1, U1
zone	Z, O1, U1, N
zulu	Z, IU, U1, L, IU, U1

APPENDIX: LIST OF VENDORS

E

You can receive information concerning the prices and availability of the various hardware devices described in this book 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

CMS I/O board for the Commodore 64:

Creative Microprocessor Systems, Inc.
PO Box 1538
Los Gatos, CA 95030

INDEX

Selections from The SYBEX Library

Introduction to Computers

OVERCOMING COMPUTER FEAR

by Jeff Berner

112 pp., illustr., Ref. 0-145

This easy-going introduction to computers helps you separate the facts from the myths.

COMPUTER ABC'S

by Daniel Le Noury and Rodnay Zaks

64 pp., illustr., Ref. 0-167

This beautifully illustrated, colorful book for parents and children takes you alphabetically through the world of computers, explaining each concept in simple language.

PARENTS, KIDS, AND COMPUTERS

by Lynne Alper and Meg Holmberg

208 pp., illustr., Ref. 0-151

This book answers your questions about the educational possibilities of home computers.

THE COLLEGE STUDENT'S COMPUTER HANDBOOK

by Bryan Pfaffenberger

350 pp., illustr., Ref. 0-170

This friendly guide will aid students in selecting a computer system for college study, managing information in a college course, and writing research papers.

COMPUTER CRAZY

by Daniel Le Noury

100 pp., illustr., Ref. 0-173

No matter how you feel about computers, these cartoons will have you laughing about them.

PROTECTING YOUR COMPUTER

by Rodnay Zaks

214pp., 100 illustr., Ref. 0-239

The correct way to handle and care for all elements of a computer system, including what to do when something doesn't work.

YOUR FIRST COMPUTER

by Rodnay Zaks

258 pp., 150 illustr., Ref. 0-045

The most popular introduction to small computers and their peripherals: what they do and how to buy one.

SYBEX PERSONAL COMPUTER DICTIONARY

120 pp., Ref. 0-067

All the definitions and acronyms of microcomputer jargon defined in a handy pocket-sized edition. Includes translations of the most popular terms into ten languages.

FROM CHIPS TO SYSTEMS: AN INTRODUCTION TO MICROPROCESSORS

by Rodnay Zaks

552 pp., 400 illustr., Ref. 0-063

A simple and comprehensive introduction to microprocessors from both a hardware and software standpoint: what they are, how they operate, how to assemble them into a complete system.

Personal Computers

Commodore 64/VIC-20

THE COMMODORE 64™/VIC-20™ BASIC HANDBOOK

by Douglas Hergert

144 pp., illustr., Ref. 0-116

A complete listing with descriptions and instructive examples of each of the Commodore 64 BASIC keywords and functions. A handy reference guide, organized like a dictionary.

THE EASY GUIDE TO YOUR COMMODORE 64™

by Joseph Kascmer

160 pp., illustr., Ref. 0-129

A friendly introduction to using the Commodore 64.

THE BEST OF COMMODORE 64™ SOFTWARE

by Thomas Blackadar

150pp., illustr., Ref. 0-194

Save yourself time and frustration with this buyer's guide to Commodore 64 software. Find the best game, music, education, and home management programs on the market today.

YOUR FIRST COMMODORE 64™ PROGRAM

by Rodnay Zaks

182 pp., illustr., Ref. 0-172

You can learn to write simple programs without any prior knowledge of mathematics or computers! Guided by colorful illustrations and step-by-step instructions, you'll be constructing programs within an hour or two.

YOUR SECOND COMMODORE 64™ PROGRAM

by Gary Lippman

250 pp., illustr., Ref. 0-152

A sequel to *Your First Commodore 64 Program*, this book follows the same patient, detailed approach and brings you to the next level of programming skill.

COMMODORE 64™ BASIC PROGRAMS IN MINUTES

by Stanley R. Trost

170 pp., illustr., Ref. 0-154

Here is a practical set of programs for business, finance, real estate, data analysis, record keeping and educational applications.

GRAPHICS GUIDE TO THE COMMODORE 64™

by Charles Platt

192 pp., illustr., Ref. 0-138

This easy-to-understand book will appeal to anyone who wants to master the Commodore 64's powerful graphics features.

Software and Applications

Operating Systems

THE CP/M® HANDBOOK

by Rodnay Zaks

320 pp., 100 illustr., Ref 0-048

An indispensable reference and guide to CP/M—the most widely-used operating system for small computers.

MASTERING CP/M®

by Alan R. Miller

398 pp., illustr., Ref. 0-068

For advanced CP/M users or systems programmers who want maximum use of the CP/M operating system . . . takes up where our *CP/M Handbook* leaves off.

THE BEST OF CP/M® SOFTWARE

by John D. Halamka

250 pp., illustr., Ref. 0-100

This book reviews tried-and-tested, commercially available software for your CP/M system.

REAL WORLD UNIX™
by John D. Halamka
250 pp., illustr., Ref. 0-093
This book is written for the beginning and intermediate UNIX user in a practical, straightforward manner, with specific instructions given for many special applications.

Business Software

INTRODUCTION TO WORDSTAR™
by Arthur Naiman
202 pp., 30 illustr., Ref. 0-077
Makes it easy to learn how to use WordStar, a powerful word processing program for personal computers.

PRACTICAL WORDSTAR™ USES
by Julie Anne Arca
200 pp., illustr., Ref. 0-107
Pick your most time-consuming office tasks and this book will show you how to streamline them with WordStar.

THE ABC'S OF 1-2-3™
by Chris Gilbert
225 pp., illustr., Ref. 0-168
For those new to the LOTUS 1-2-3 program, this book offers step-by-step instructions in mastering its spreadsheet, data base, and graphing capabilities.

UNDERSTANDING dBASE II™
by Alan Simpson
220 pp., illustr., Ref. 0-147
Learn programming techniques for mailing label systems, bookkeeping and data base management, as well as ways to interface dBASE II with other software systems.

Business Applications

INTRODUCTION TO WORD PROCESSING
by Hal Glatzer
205 pp., 140 illustr., Ref. 0-076
Explains in plain language what a word processor can do, how it improves productivity, how to use a word processor and how to buy one wisely.

Languages

BASIC

CELESTIAL BASIC
by Eric Burgess
300 pp., 65 illustr., Ref. 0-087
A collection of BASIC programs that rapidly complete the chores of typical astronomical computations. It's like having a planetarium in your own home! Displays apparent movement of stars, planets and meteor showers.

Assembly Language Programming

PROGRAMMING THE 6502
by Rodnay Zaks
386 pp., 160 illustr., Ref. 0-046
Assembly language programming for the 6502, from basic concepts to advanced data structures.

6502 APPLICATIONS
by Rodnay Zaks
278 pp., 200 illustr., Ref. 0-015
Real-life application techniques: the input/output book for the 6502.

Hardware and Peripherals

THE RS-232 SOLUTION
by Joe Campbell
225 pp., illustr., Ref. 0-140
Finally, a book that will show you how to correctly interface your computer to any RS-232-C peripheral.

USING CASSETTE RECORDERS WITH COMPUTERS
by James Richard Cook
175 pp., illustr., Ref. 0-169
Whatever your computer or application, you will find this book helpful in explaining details of cassette care and maintenance.

are different.

Here is why . . .

At SYBEX, each book is designed with you in mind. Every manuscript is carefully selected and supervised by our editors, who are themselves computer experts. We publish the best authors, whose technical expertise is matched by an ability to write clearly and to communicate effectively. Programs are thoroughly tested for accuracy by our technical staff. Our computerized production department goes to great lengths to make sure that each book is well-designed.

In the pursuit of timeliness, SYBEX has achieved many publishing firsts. SYBEX was among the first to integrate personal computers used by authors and staff into the publishing process. SYBEX was the first to publish books on the CP/M operating system, microprocessor interfacing techniques, word processing, and many more topics.

Expertise in computers and dedication to the highest quality product have made SYBEX a world leader in computer book publishing. Translated into fourteen languages, SYBEX books have helped millions of people around the world to get the most from their computers. We hope we have helped you, too.

For a complete catalog of our publications please contact:

U.S.A.
SYBEX, Inc.
2344 Sixth Street
Berkeley,
California 94710
Tel: (800) 227-2346
(415) 848-8233
Telex: 336311

FRANCE
SYBEX
6–8 Impasse du Curé
75018 Paris
France
Tel: 01/203–9595
Telex: 211801

GERMANY
SYBEX-Verlag GmbH
Vogelsanger Weg 111
4000 Düsseldorf 30
West Germany
Tel: (0211) 626411
Telex: 8588163

THE COMMODORE 64 CONNECTION

NOW YOU CAN DO MORE THAN PLAY GAMES WITH YOUR COMMODORE 64!

The Commodore 64 Connection will show you how easy it is to use your computer together with household devices. You will quickly learn the simple techniques for using your Commodore 64 to control:

- a home security system
- a home temperature control system
- a voice synthesizer to make your computer talk
- and many more home appliances

The Commodore 64 is well suited for connecting to non-computer devices. With this straightforward guide and a little imagination, there is no limit to the exciting and useful things you can do with your Commodore 64 personal computer.

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, including *The IBM PC Connection, The Apple Connection, The VIC-20 Connection, Programming the 8086/8088,* and *Z80 Applications.* He holds an MSEE degree from Santa Clara University.

ISBN 0-89588-192-6