POWERFUL PROJECTS WITH YOUR TIMEX/SINCLAIR

Jim Stephens

POWERFUL PROJECTS WITH YOUR TIMEX/SINCLAIR

Jim Stephens

Scott, Foresman and Company

Glenview, Illinois

London

ISBN 0-673-18038-7

Printed in the United States of America.

Library of Congress Cataloging in Publication Data

Stephens, Jim, 1944–
Powerful projects with your Timex/Sinclair.

Includes index.
1. Computers—Amateurs' manuals. 2. Timex/Sinclair 2068 (Computer) I. Title.
TK9969.S74 1985 621.3819'58 84-14123
ISBN 0-673-18038-7

1 2 3 4 5 6—KPF—89 88 87 86 85 84

Notice of Liability

The information in this book is distributed on an "AS IS" basis, without warranty. Neither the author nor Scott, Foresman and Company shall have any liability to customer or any other person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by the programs and projects contained herein. This includes, but is not limited to, interruption of service, loss of data, loss of business or anticipatory profits, or consequential damages from the use of the programs and projects.

Preface

This book is written for the many beginners who want to learn about the fascinating world of computer electronics. It is for those who want to experiment and those who believe there is more to working with a computer than watching it balance a checking account or zap space creepers.

The purpose of this book is to convince the reader that there is nothing mysterious or threatening about the electronics of computers and that basic electronic interfacing is at least as easy as programming. This book is written to introduce these basics and to encourage the beginner to experiment, to learn, and, most important, to have the satisfaction that comes from "doing it yourself."

Chapter 1 discusses the background of the Timex/Sinclair 2068 and the system details such as connections and fine tuning for best operation. The innards of the computer are explained for those who like to know what's inside. The discussion will help you considerably to understand exactly what your computer is trying to do.

Chapter 1 also contains a necessary discussion of software, a discussion that is not overly complicated.

Chapter 2 presents various components that will be used in the projects. It also gives instructions on wiring techniques. Even if you are a seasoned electronic hobbyist, you may find tips here that will save you some headaches later.

The actual projects are detailed starting at the end of chapter 2. Once you begin building, don't be discouraged if something doesn't seem to be working correctly. Review the preceding material, and, above all, don't give up.

Chapter 3, by the way, contains a section that explains the *backplane,* or *interface connector,* in preparation for the projects that follow. This section will be of considerable help when you construct the projects. Study it carefully. When a circuit doesn't work, refer to this section.

I may give up electronics myself one day, but not before I've tried every diagram for which I can find the appropriate parts. Maybe one day a project will work perfectly the first time; maybe one day I'll completely understand this maze of knowledge. Until then, I'll keep staying up all night, wondering which connection is causing a certain problem or marveling at the behavior of some new creation of mine.

Through all of my research for this book and my years of circuit building, I've found only two types of hardware manual. There is the one that shows you the theory in a very complicated way and gives a few examples of specific applications. The other is the project-book type, which shows how to construct various circuits but gives little information on how or why the circuits work. Neither type bothers to apply concepts and projects to everyday problems.

The reader of the book on theory is left with no very concrete application for his or her new knowledge. The reader of the project book gets several hardware ideas but little understanding (of the circuits) that would allow independent work. This book hopes to avoid these two consequences by combining a little theory with several experiments and then applying the resultant knowledge in useful project circuits.

Very few hardware manuals make an effort to specify parts that are readily available from local sources. Of those that do, most must be written in the heart of the Silicon Valley. I once learned of a nice circuit that I wanted to try, only to find that the major part in the circuit was a single-source item that had to be ordered from the factory. Of course, the factory's minimum order quantity was ten items per stock number, and the factory's minimum order amount was $65! I've tried to make every effort to specify only parts locally available or readily obtainable by mail order from small-quantity suppliers. Appendix A lists a few of these suppliers that have given fast and inexpensive service and that stock the parts specified for the projects in this book.

Finally, I want to dedicate this book to my two young sons, Gregg and John, who will inherit the new world of computers and who wait so patiently while their father pursues his endless tinkering.

Contents

1

The Timex/Sinclair 2068

More and more computer-related knowledge is needed daily by those who are entering the world of work. For instance, in the past ten years, the computer and its associated electronics have changed the whole field of manufacturing and the training of individuals who work in it. Professionals in manufacturing must have some knowledge of electronics or the computer control of manufacturing processes. Those involved in manufacturing design practically have to have degrees in programming or computer control.

This book endeavors to explore concepts related to the electronic nature of computers and the computer control of peripheral devices. The Timex/Sinclair (TS) 2068 is a perfect learning tool in both areas. With it, one can learn the essentials in preparation for further study.

The original ancestor of the Timex/Sinclair 2068 was the ZX80, designed in Britain by Sinclair Research Ltd. Its sales were, unexpectedly, enormous. So impressive was the success of the ZX80 that Sinclair Research immediately redesigned it and, using the marketing techniques of Timex Corporation, introduced the ZX81. This new computer was an even greater success than its predecessor because it greatly improved upon the ZX80's capabilities.

Both machines offered an excellent BASIC language, but the BASIC lacked several important commands necessary for serious programming. The ZX81 proved so popular that Sinclair Research teamed up with the Timex Corporation and produced the Timex/Sinclair 1000. The two companies later improved the keyboard on the TS 1000 and called it the Timex/Sinclair 1500. Both machines were just restyled ZX81s. Sinclair improved upon the ZX81 with another model, the Timex/Sinclair 2000, or

Spectrum (as it was called in Britain). Commands were added to its BASIC, and color and sound were incorporated for improved game playing. But, like its predecessors, the TS 2000 suffered in quality because its small keyboard was not adequate for proper typing.

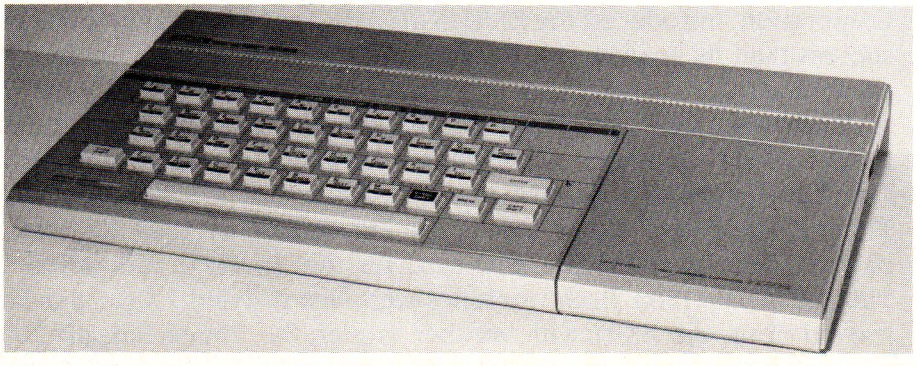
After almost two years of design, Timex introduced the enhanced version of the Spectrum, the 2068. Besides offering color, sound, and improved BASIC, the new 2068 has a full 64K (64,000) bytes of memory and a larger keyboard (figure 1.1). This new machine promises to be one of the best home computers around for a long time to come. With this machine, the average citizen has at his or her fingertips the computing power to meet all needs at a fraction of the cost of most other units with the same capabilities.

Unlike several other computers, the Timex/Sinclair 2068 has joystick connectors, a monitor jack, peripheral edge connections, and even a cartridge slot for ready-to-run software. Its sleekness and small casing make it a natural choice for the home.

The 2068 System

Now let's get familiar with the system components of the 2068 (figure 1.2) before we start our journey into the world of computer electronics. This section will cover the setup of each of the components except the printer and give details of proper placement, connection, and operation. Included here are many suggestions on how to adjust the system for proper operation. Study this section carefully. It is fundamental.

Figure 1.1
The Timex/Sinclair 2068 keyboard console.



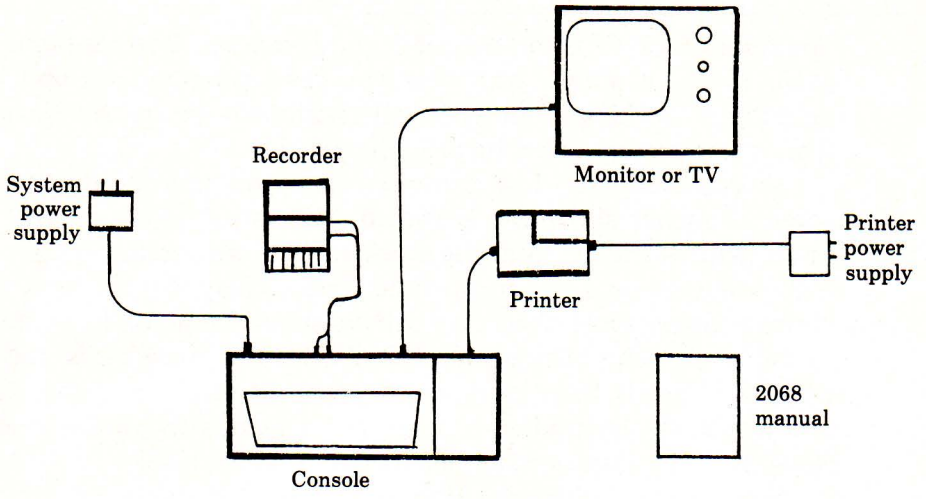


Figure 1.2
The complete Timex/Sinclair 2068 computer system.

The Console

The *console,* or *keyboard,* is the heart of the system. All of the computer's electronics, connections, and memory are housed in it. (Anything that connects to the console is called a *peripheral.* The recorder, the monitor, and even cartridges you may use are considered peripherals.) Since the console contains all of the electronics, be extremely careful in handling it and locating it for use or storage.

Remember to place the console in the most comfortable position for typing. The dining room table is not the best surface on which to work. The height of the typing work surface should be about 28 inches. Anything greater will slow your typing (and programming) speed considerably.

The Recorder

The recorder should be placed to the right of the console if you are right-handed or to the left if you are left-handed. The 2068 manual explains how the cables connect, but you should know that the idiosyncrasies of your tape recorder have a lot to do with what happens after you connect the cables. Some recorders (mine, for instance) do not save programs well when the ear cable is plugged into its jack. And if your tape recorder is one of the less expensive models, you may have to position it away from the display since the electromagnetic field of the monitor can cause improper loading and saving.

The Monitor

The monitor or TV is the output of the computer. Besides being its voice, so to speak, it shows you what you have programmed and how your program is working. Your eyes will continually be on this device. Therefore, it is necessary that it be properly placed.

Several computer desk manufacturers think that the display has to be some 24 inches above the keyboard. This is just not so. If the display is more than 14 inches from the work surface, the strain on your neck and eyes will wear you out in no time. The display should be at or slightly above your eye level. Your eyes will constantly be going from the keyboard to the screen and back again. The shorter this distance is, the less your eyes (and head) have to move. Because of this, I find that the display monitor works best when it is around 4 inches above the work surface. Never have the display to the left or right of the console.

You will probably be using a standard television set for displaying information. This is fine because the 2068 was designed mainly for this type of peripheral. However, there are several things that you should take into account when selecting the TV to use.

If your programming consists mainly of game playing, then you probably will want a color set to take advantage of the great color graphics of the 2068. This can get to be a problem if the set happens to be the family color TV. Many arguments can be avoided if the set is an extra that is used solely with the 2068. If most of your work is programming (not using color), a small, portable black and white set is your best bet. This should have good video response (clarity) and a wide range of fine tuning. Since the console outputs its signal on either channel 2 or channel 3, some fine tuning is going to be necessary for proper operation. When you connect some of your prototype circuits to the console, even more fine tuning may be necessary.

The System Power Supply

The power supply for the 2068 that is included with the system is much larger than the ones that operated the earlier models. It has a capacity of 1 amp, but this supply is operating more than twice the number of chips than the smaller supplies had to operate. Therefore, it is marginal if more than the console is connected to it. Even though there are power connections on the backplane, we will look elsewhere for power for our circuits and not rely on the 2068 power pack for our purposes.

The 2068 Manual

It will be assumed that you have studied the 2068 manual in depth and know how to use the keyboard (a challenge in itself). There are parts of

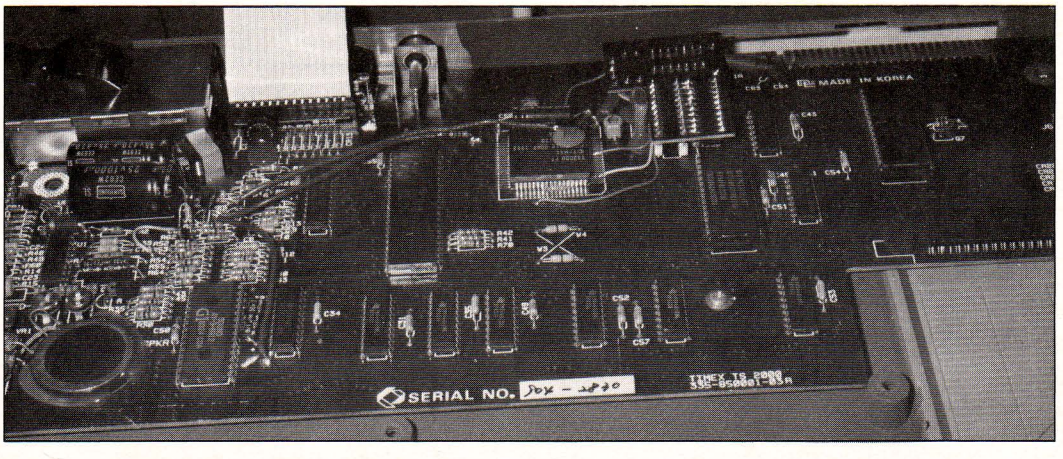
the manual that will be extremely important in our designs. These are the appendixes on pages 217 through 290. Normally, you would not need most of this information in order to use the 2068. We will need it because it pertains to the things that make the computer and its peripherals work. The memory map on page 255 of the 2068 manual is a good example of this kind of information. Moreover, we will learn how to use machine code for control purposes and learn to enter data in hexadecimal. The Character Set appendix on page 242 of the 2068 manual will be a tremendous help.

The Printed Circuits

If you are like me, you always want to take things apart to see how they operate. This is not a good idea with the 2068. Besides voiding the warranty (a good thing to have in effect these days), disassembly and reassembly of the console can damage the system and keep it from operating. The *backplane,* or *edge connector,* is the only part of the system that we need to inspect. So that you can see what's inside without actually opening the console, figure 1.3 (a photo of the open case) shows the complicated components of the 2068. Many of the chips are special designs for Timex, and only Timex understands their operation. Figure 1.4 shows the various components and their layout. Two standard components on the circuit board are the Z80 microprocessor and the AY3-8912 sound synthesizer chip.

The printed circuit board is dominated by the Z80 central processing unit (CPU) and the Timex Logic Unit. The other integrated circuits tie the

Figure 1.3
The interior of the 2068. The Timex/Sinclair is well designed, with state-of-the-art components.



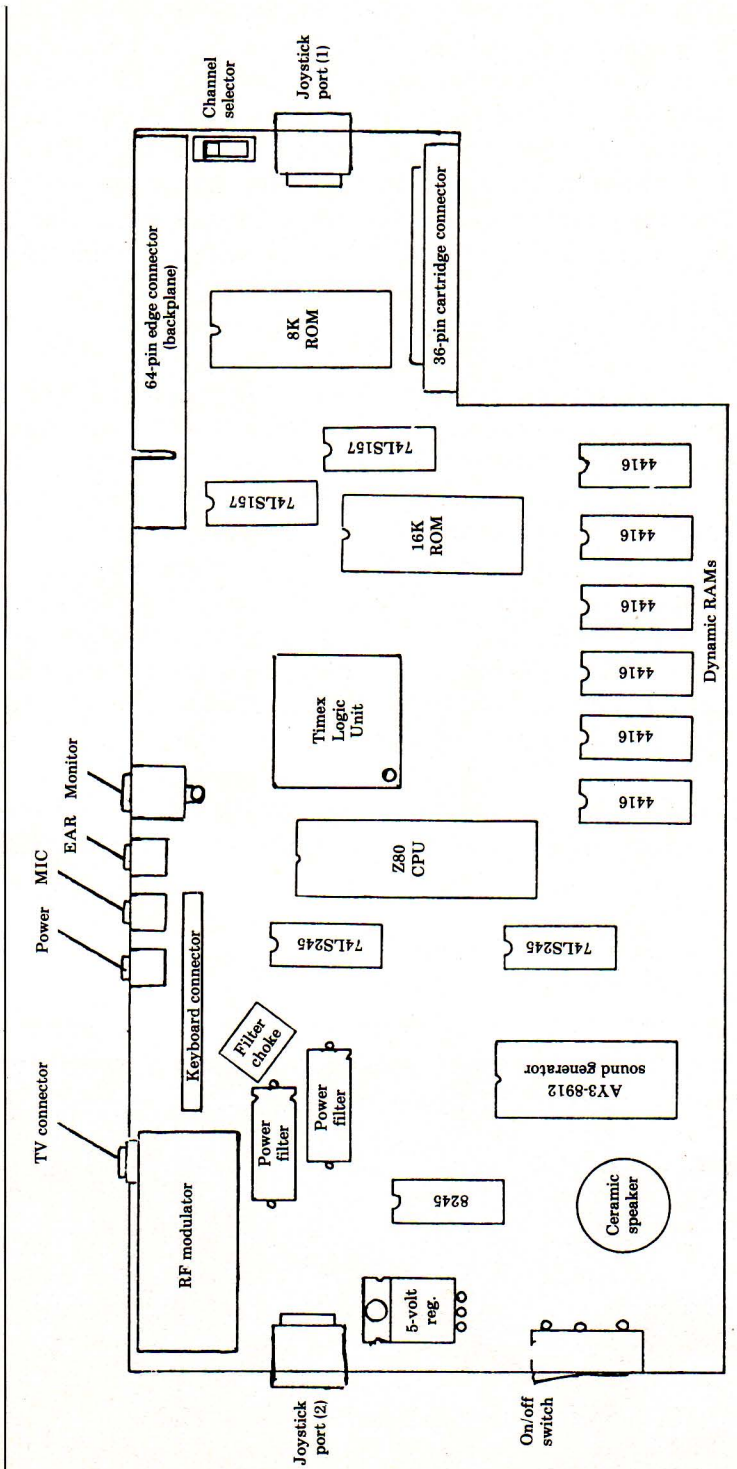


Figure 1.4
The component layout of the Timex/Sinclair 2068.

components and circuitry together. The BASIC ROMs (ROM stands for *read-only memory*) are responsible for the console's language, and the Timex Logic Unit takes the place of the older Sinclair Logic Unit, which replaced several hundred components that were necessary in older personal computers.

Most of the signals on the double-sided edge connector of the 2068 are identical to those of the older ZX81, TS 1000, and TS 1500 computers and will present no problem for proper interfacing. These connections will be covered in detail in later sections. These connections vary considerably in the manner in which the system addresses them, and, therefore, they are not compatible with earlier cartridges and peripherals. The printer for the older-model Timex computer will still work when connected to this backplane.

The Electronic System

The 2068 is very similar in construction to its parent computer, the TS 1000. The matrix lines of the keyboard are arranged in the same format. The keys themselves and, of course, the controlling software have been changed to support the additional BASIC commands. The keyboard connects to the printed circuit board of the 2068 by a thirteen-lead ribbon cable.

Power for the computer is supplied by a 1-amp wall power pack that is unregulated. The power is regulated by a standard 5-volt power tab inside the console, and the ripple is smoothed by two heavy power capacitors and a small filter choke located in the console. All overheating problems of the earlier machines appear to have been corrected. Even the wall power pack does not heat up. As I mentioned above, however, we will be looking elsewhere for power for our project circuits.

Memory consists of 48K (48,000) usable bytes, but only 38,652 bytes are immediately available. User graphics, display files 1 and 2, and reserved area for system variables take up much of the remaining space. Appendix B shows how the memory is allocated.

The Central Processing Unit

The heart of the system is the Z80 CPU, which is discussed in more detail in the next main section of this chapter. Some other systems use specially designed CPUs about which very little literature is available. In contrast, there is a lot of literature about the Z80. Moreover, all of the signal characteristics of the Z80 CPU have been retained on the backplane. This is a definite advantage because there is also an abundance of information available about the behavior of the electronic backplane. All this information is useful to the electronic hobbyist.

The codes for the display characters of the 2068 are not standard ASCII (American Standard Code for Information Interchange), but you can easily overcome this disadvantage with proper conversion software. Later we will discuss the serial transmission of data to the 2068, and conversion of ASCII codes to the Timex codes would be necessary in case of such transmission.

The AY3-8912 Sound Chip

Sound on the 2068 is handled by the standard AY3-8912 sound synthesizer, for which complete details are published and available. This integrated circuit (IC) is quite complicated to program, but its functions are independent of the software, and it allows the CPU to operate without interference from software. Thus, the BASIC is not slowed down once the chip has been initialized. This chip is the most versatile of the sound-generating chips available today, and it makes for state-of-the-art sound on the TS 2068.

The small speaker in the bottom of the case detracts somewhat from the quality of the sound produced, but Timex has provided another option that eliminates the tinny sound of the console speaker. You have a jack on the console, marked MIC, that is used with the recorder to save programs. The sound is also channeled through this connection and can be fed to an audio amplifier (such as your stereo) for spectacular effects. Therefore, if you write a small symphony using BASIC, just push down the *record* and *play* buttons on your recorder, and the melody will be saved.

The Memory Map

Memory is divided into several sections. These are shown in Appendix B, along with their distinct uses. Notice that I have listed both the decimal and hex addresses of each section so that you may call locations by the hexadecimal notations later. Notice also that the display file is located below the program area. This arrangement is an improvement over that in the earlier TS 1000 model; it makes the display memory stationary and simplifies your work with graphics.

The Z80 CPU

The central processing unit is the brain behind all personal computers. This single chip is responsible for all of the output that is seen on the screen or printed on the printer. The CPU in the Timex/Sinclair 2068 is no exception.

The central processing unit that operates the Timex/Sinclair computer is the Z80 microprocessing chip made by Zilog Corporation. This remark-

able CPU is a leader in the field of microprocessors. Many personal computers use this chip as the workhorse of the system. The Z80 is an advanced version of the once popular 8080A CPU. The instructions are very similar; in fact, many of them are identical. There are some four hundred instructions that can be used by the CPU in operation. We will look at many of the more common instructions that our CPU utilizes when we study the instruction set of the Z80.

Three main parts make up the total system of almost all personal computers: the CPU, the memory, and the peripherals such as the display, the printer, and the recorder. Of course, there are usually a few hundred parts besides each of these components, tying them together. The Timex/Sinclair drastically reduces the number of these associated parts by taking advantage of the many capabilities of the Z80 and by using software rather than hardware.

The Z80 chip is an *8-bit microprocessor.* This means that it transmits data by the sole means of routing 8 bits of information at a time to and from memory and associated devices. This simultaneous transmission of 8 bits of data is called *parallel transmission.* You'll see that the Timex/Sinclair's backplane has eight lines marked D0 through D7. These make up the *data bus.* It is by this path that the 8-bit data words, or *bytes,* are sent. Let's see how the Z80 operates and study exactly what happens inside our CPU when we turn it on.

Z80 Operation

The Z80 is a forty-pin IC (figure 1.5) that is located in the center of the Timex/Sinclair circuit board and does most of the work of the system. Through connections between these forty pins and various components, all of the operational needs of the CPU are supplied.

Z80 Power Supply

Many earlier CPUs needed up to four separate power supplies. The Z80 needs only one 5-volt supply. This is a big advantage for anyone who is designing to save cost. The power supply actually delivers 15 volts, but the voltage is reduced by the regulator inside the console to a mere 5 volts. This reduction allows the supply to maintain the 5-volt level without problems. Even though the power pack can deliver a full ampere of current, the console electronics need only about a fourth of this, or 250 milliamps.

Z80 Timing

In order to keep all of its operations in step with each other, any CPU needs a *master clock* that supplies a timed signal. The Z80 is no exception.

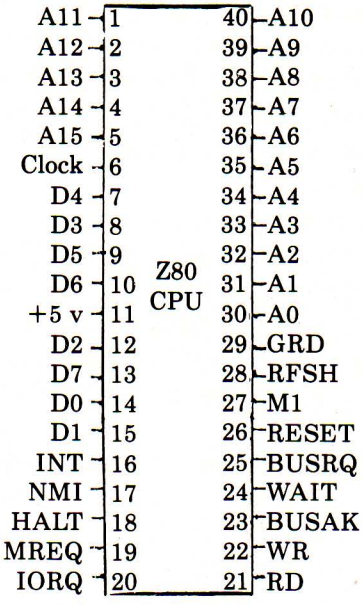


Figure 1.5
A pinout of the Z80 microprocessor.

The *master clock* is the device or circuit that trips all the other signals at the precise time necessary for proper operation. The clock oscillates between 5 volts and ground and does this at a fantastic speed. This speed of oscillation is called its *frequency*. Though the Z80 is capable of using a clock frequency of up to 10 million cycles per second (10 MHz), the Timex/Sinclair clock is set to operate at 6.5 million cycles per second. The frequency (6.5 MHz) is divided by 2 to give the system speed of 3.25 MHz. This simply means that, for every clock cycle, one instruction or part of an instruction is carried out inside the CPU.

The reason for the lower frequency of the master clock in our computer is to increase the reliability of the system and allow proper system timing for the television scanning. The slower the clock, the more time the other parts of the computer have in which to keep in step, read data, or store information without error.

Z80 Data Bus

Again, there are eight of the Z80's pins that are set aside for sending or receiving data. These pins are tied to eight lines called the *data bus*. A *bus* is a group of signal paths that serve one specific purpose. If one of these lines is at 5 volts, it is said to be *active*, or *active high*, and is read as meaning 1. The eight data lines of the Z80 are active high data lines. (If a

signal is usually high when the CPU is not using it for a purpose but goes low when it is active, it is said to be *active low.* Several of the control lines of the bus remain high until certain operations are performed by the CPU. This is an important fact to keep in mind when you are designing a control circuit. We will talk more about using the active high and active low characteristics of the buses when we discuss actual projects.)

Z80 Address Bus

Sixteen of the pins on the Z80 are marked A0 through A15. These make up the *address bus* of the system. These lines are the ones that advise the system where information is supposed to go. They tell the CPU the target *location,* or *address.* We'll study binary code in more detail a little later. For now, remember that 8 bits combine in just 256 ways, but 16 bits, the number of bits carried by the address bus, allow 65,536 combinations, or in this case addresses. Therefore, since the Z80 has 16 bits on its address bus, it can address 65K words of memory. Each of these lines is also active high. This will be an important fact to remember when we start interfacing devices to the Z80's address bus.

Z80 Control Signals

There are five lines that carry what can be called the *control signals* of the Z80. These are the ones that tell the CPU what it can and cannot do. Think of these signals as the traffic controllers of the brain of your system. We'll learn more about the use of some of these signals in our system when we discuss the operation of the computer. The signals are as follows:

BUSRQ This pin handles the line called the *bus request.* The *bus request* is an input signal that tells the CPU a peripheral device wants to put data on the bus. The CPU will turn over its data and address buses to any device by removing itself from these lines. This way, an input device can get control of the bus and transfer information through the CPU to the desired location. This line is active low.

INT When an external device wants the attention of the CPU and does not have the time to wait, it uses INT, or the *interrupt request.* This active low signal makes the CPU stop what it is doing and listen for further instructions from the device.

NMI *Nonmaskable interrupt* is similar to INT, but it has a higher ranking. That is, the effectiveness of INT depends on the states of several of the other control lines; NMI demands that it have the attention of the CPU no matter what.

Wait This is an easy signal to remember since it makes the CPU do exactly what its name implies. It makes it wait, much like an airplane that has been put into a holding pattern. The CPU stops executing instructions and just sits there and idles. The only time this active low signal is of value is when you want to slow down the CPU. If the external device is slow in sending data, several of these signals allow the device to catch up.

Reset This signal, when brought low, tells the CPU to start over. The CPU will reset all of its counters and pointers and start executing from location zero. The instructions at location zero happen to be *clear screen and start over* in the TS 2068, so *reset* has the same effect as pulling the plug and plugging it back up. This signal is not very useful to us, however—although it would make a good reset switch. Now that might make a good project!

Z80 System Controls

The CPU does not always take orders; sometimes it gives them. It does this using eight of its forty pins. These signals let the CPU tell the peripheral devices what, where, and when they can do their thing. The CPU's commands are as follows:

BUSAK This is the response signal of the CPU to a bus request from an external device. The CPU finishes the current instruction and then lets the device know that it is ready to give bus control to the device. This active low signal helps avoid traffic jams on the bus.

Halt This line goes low in response to the machine-code instruction HALT. The CPU sits and idles, waiting for an interrupt that has been programmed to happen. It is an active low line.

M1 This is a status indicator that means *machine cycle 1.* It tells the external devices where the CPU is in the instruction cycle. This line goes low when the CPU is fetching instructions from memory. We will not be using this indicator.

RFSH The *refresh* line is intended to be used with *dynamic* memory chips. These chips require that they be *refreshed,* or *strobed,* constantly, or they lose their information. Since this signal is so critical to the memory pack in maintaining its data, we won't bother it in our projects.

MREQ This abbreviation stands for *memory request,* and the line so named goes low when the CPU wants the bus available to either read or write to memory. This is a good interfacing signal since most interfacing projects treat the external device as if it were memory.

IORQ IORQ stands for *input/output request.* The Z80 has the capability to address an input/output (I/O) device directly using the lower 8 bits of the address bus. When it executes an OUT or an IN instruction, this line goes low. It is very useful in decoding when you are using these instructions for the driving routines. We will use IORQ in most of our work.

WR This *write* line goes low when the CPU is placing data on the data bus for writing to memory or sending information to an external device. This is another signal we will put to good use.

RD When the CPU wants to receive data from either an external device or memory, it puts a low on this *read* line. Again, this is a very important interfacing signal.

Now that we have discussed the pin diagram of the Z80 CPU in some detail, the pins aren't too intimidating, I hope. Each has a function, and several are more important to us than others. The backplane, or output connector, of the Timex/Sinclair makes most of these signals available for our use, as we will discuss in chapter 3. By the proper use of these signals, we can make the computer do anything we would like that is within its capabilities. However, the important thing is that these lines be used correctly. Improper use can do terrible things to the CPU, the whole system, and your pocketbook. Treat these signals with respect. Always doublecheck your connections to these points, and *never apply more than 5 volts to any of these connections.*

Most of these signals fluctuate in response to our instructions (or the instructions of the ROM). They simply go low or high depending on what our program has told the CPU to do. These instructions as a group are called the *instruction set.*

Machine Code and the Z80 Instruction Set

Machine Code

Machine code is just another name for *binary code,* a combination of ones and zeroes that the CPU can handle. Figure 1.6 shows the conversion to binary of the first sixteen decimal numbers, 0 through 15. All the decimal numbers convert to individual combinations of ones and zeroes in the binary system. The four hundred or so machine code instructions of the Z80 are represented by such combinations. Practice counting in binary to get a feel for the various binary codes.

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

Figure 1.6
A decimal-binary conversion chart.
Notice how the binary 1s alternate.

The Z80 CPU operates by using binary numbers. There are 8 bits of data on the data bus that can represent 256 combinations of information. The CPU responds to these combinations with different operations such as reading, writing, and halting. (Appendix C of this book lists many of these instructions and tells what the CPU tries to do in response to them.)

The instructions are received by the Z80 CPU in sequence. That is, the instruction is fetched from memory, the operation is identified, and then the operation is performed. Simple? Yes, it really is. Let's see how it works.

Suppose you want the CPU to read memory location 65200. In BASIC, you simply write PRINT PEEK (65200). The CPU, however, operates in binary, so our ROM program has to convert this statement to machine code, or binary code, that the CPU can understand. You can see by looking at the instruction set in Appendix C that the instructions PEEK and READ are not present. There is an instruction that does the same thing, however. This instruction is called LOAD (LD). LOAD tells the CPU to take the information and put it somewhere. This somewhere is designated by the two pieces of information that follow the LD. The LOAD instructions, then, need more than one number for proper operation. (Some instructions are one-word operations; others are two, three, and even four instruction words long.)

Instruction Groups

There are eleven groups of instructions recognized by the Z80. Each group tells the CPU what steps are necessary to perform a specific function. The groups are as follows:

1. 8-bit load group
2. 16-bit load group
3. Block move and transfer group
4. 8-bit arithmetic and logic group
5. 16-bit arithmetic and logic group
6. General control group
7. Jump group
8. Call and return group
9. Rotate and shift group
10. Set and reset group
11. Input/output group

We'll not cover each of these in detail, but it is good to know what each group does in preparation for some limited machine code use with our projects.

Load Groups These groups take information from the bus (or from memory) and put it at a location specified in part of the instruction. This instruction is very similar to the POKE command in Sinclair BASIC.

Block Move and Transfer Group This group of instructions is very similar to the LOAD instruction in that it takes a large chunk of data from memory and moves it to another location. It can even transfer it to an output device such as a disk, display terminal, or printer. This is one of the most powerful commands in the Z80 instruction set.

Arithmetic and Logic Groups All of these instructions involve either addition and subtraction or Boolean logic functions. This group is extremely useful in comparing numbers, performing mathematical operations, and programming decision-making capabilities into the computer.

General Control Group This limited set of instructions controls the operation of the CPU. It includes instructions such as HALT, NOP (no operation, idle), and IM (set interrupt enable).

Jump Group These instructions allow the code to be executed from various locations depending on changing factors. They are much like the IF . . . THEN and GOTO statements found in BASIC.

Call and Return Group The BASIC statements of GOSUB and RETURN jump to special subroutines that perform specific functions based on specific parameters. The same holds true for CALL and RETURN in machine code. This group remembers where it jumped from and returns back to this location when it finishes. (You may have noticed that machine code programming is beginning to sound a lot like BASIC programming.)

Rotate and Shift Group It's difficult to understand the rotate and shift group unless you have had some experience with shift registers. However, if you visualize eight windows, each containing a one or a zero, and if you imagine this pattern of bits marching either to the right or to the left, then you have an image of the bits *shifting* from window to window. Visualize the space in the windows from which the bits disappear as being filled with zeroes as the pattern advances. This is *bit shifting. Rotating* is the same operation except that, as the one or zero leaves the last window, it comes around and takes the available window behind the marchers. You can program the Z80 CPU to rotate or shift a specified number of times and by this method figure out what bits you have in the windows. Just rotate or shift eight times, and each bit of the 8-bit word can be examined by the computer. It is a very important operation of the computer when it makes decisions based on certain data values.

Set and Reset Group This instruction group allows the CPU to change the lowest bit in the 8-bit window to either zero or one by the use of the RESET (change to 0) or SET (change to 1) instruction. Depending on the original value of the low bit, the CPU sets or resets a flag, or marker, that it can test. This way, the CPU is able to tell if the word is odd or even without changing the contents of the window. This window is called the *accumulator,* or *A register,* in the CPU.

Input/Output Group This group contains two categories of instructions: the machine code commands that transfer data from and those that transfer data to an external device. The information can be selectively accepted by or given to various input or output ports. We'll talk more about these two commands when we build our I/O port.

This description of the instruction set of the powerful Z80 CPU was very brief. The task of machine code programming is, in fact, fairly complicated. A whole book could be written just on using machine code for the Z80. When I discuss the actual projects, I will say more about the use of some of these instructions.

Serial and Parallel Data Handling

The process of transferring data from either the memory, the CPU, or external devices is called *data transmission.* If the bits are transferred in 8-bit chunks, then the transmission is called *parallel.* If the same information is transferred 1 bit at a time, the transmission is said to be *serial.* There is a need for both types of data handling in the modern computer system.

Parallel Data Transmission

Most data is transferred, stored, and read in the parallel fashion. Our Timex/Sinclair handles all of its internal data in this mode. That is, each *word* (8 bits) is put on the data bus one whole word at a time. Each bit of information, either 1 or 0, has its own individual data line. A look at the pin assignments of the Z80 chip and those of the output connector shows these eight individual lines. This is our computer's parallel data bus. If the data stream could be frozen at any single instant, all eight lines would contain either 1 or 0. Therefore, to read the information, the computer must transfer whole 8-bit words rather than 1 bit at a time.

If the computer *latches,* or *captures,* 8 bits of data and presents them to an external device for use, the data-capturing electronics in the computer are called a *parallel output port.* A *port* is nothing but a terminal or connection for incoming or outgoing data.

If the computer latches 8 bits of incoming data with an electronic circuit, then the computer is using its *parallel input port.* We will build a parallel I/O port as one of the upcoming projects in this book. The I/O port is an important device for proper interfacing of the computer to the outside world. Without an I/O port, a computer is almost useless.

Our computer does not handle data solely by means of the 8-bit parallel data bus, however. Ever notice that information comes from your recorder via only two lines? Actually, it comes via only one line: one wire of the recorder's cable is a ground wire and does not carry information. How are the 8-bit words that are stored on tape transmitted by only one line? They are transmitted 1 bit at a time.

Serial Data Transmission

When the words in the computer are separated into 8 different bits, and 1 bit is transferred at a time until all 8 have been sent, this process is called *serial transmission.* Much like our marching bits in the window, the bits

are shifted out and sent over one data line. Serial transmission is used by our computer to transfer data to and from our tape recorder. It is a method that is very handy when you have only one line, or *channel,* on which to send information. Instances of this one-line type of transmission are transmission by telephone line or radio signal. Sometimes it might be inconvenient to string eight separate lines—if you are connecting the computer to a separate terminal down the hall, for instance. You can use only one line, send the information 1 bit at a time, and tell the receiving device to put the word back together again once all 8 bits have been received.

Many computers have the electronic circuits (and software) necessary to break words into separate bits and transmit these bits to a peripheral device. The point on the computer from which these bits are sent is, again, the serial port. Many computers have both a parallel and a serial port available on the back of their console. Unless you call the Timex/Sinclair's recorder port a serial port (and, technically, it is), we don't have a real choice of ports. However, a little hardware and a few lines of software can take the data on our parallel data bus and transmit it rather easily in a serial manner.

The device, or component, that accomplishes this task is a *universal asynchronous receiver/transmitter* (UART). This little chip has all the necessary electronics to serialize, receive, transmit, and reconstruct our data using only one line. Actually, you need two UARTs: one to send the information from the computer and one to be used by the device at the end of the line as the receiver. Fortunately, many printers and data terminals have a UART as part of their electronics.

Figure 1.7 shows how serial data is sent along a line. The bits are read at the rate at which they are transmitted. If no bits are missed, the received word is an exact copy of the word that was sent. The devices have to agree on the rate of transmission, however; otherwise, the sampling would take place at the wrong time, and chaos would result. The output would be garbage. This rate of transmission is called the *baud rate. Baud* means *bits per second.* Most UARTs send data at a rate of around 300 baud, but many have methods by which the rate can be varied among 110, 250, 300, 400, and even as high as 1200 baud. Teletype bits are usually sent at 110 baud.

A UART transmits a single 1 to signal the start of the incoming word. Then the 8 bits are sent and read. The transmitter then sends two 1s (see figure 1.7) as the signal to stop reading and to transfer the word into the device. This pattern allows the receiver and transmitter to stay synchronized, or in step.

Timing Diagrams

It is very important that you understand the function of the constantly changing signals on the control, address, and data buses. If you understand

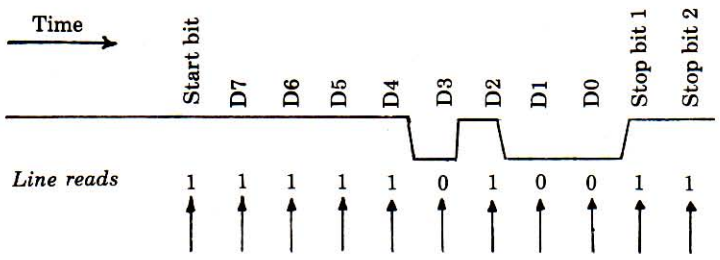


Figure 1.7
A diagram of serial data on a single line. The line is sampled by the electronics at the points indicated by the arrows. Notice that the bit is read as a level at the time of sampling.

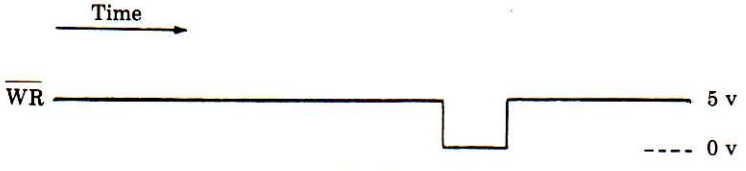
the relation of the signals to the operation of the CPU, your projects stand a good chance of working properly once the interfacing circuits are complete.

A computer bus or control line carries either a high or a low voltage level depending on the command given to the CPU by the program. For instance, if the command LOAD is executed by the CPU, a very fast, but brief, negative pulse is generated on the WR line. Otherwise, the line remains high, or positive. Figure 1.8 illustrates how the level of the control line changes in response to the LOAD command. The bar over the WR means that the line is low when it is active.

Remember, there are also many other control lines and two buses that have changing signals on them. These lines work together to produce intelligent output from the computer system.

The changing of states in one line is not important. But if several lines fluctuate in response to a command, we can use these combinations in conjunction with logic gates to route information where we want it to go. In this way, we are able to control the flow and direction of the data. This is the most important factor in the proper interfacing of any device to the CPU.

Figure 1.8
A diagram of the write pulse.

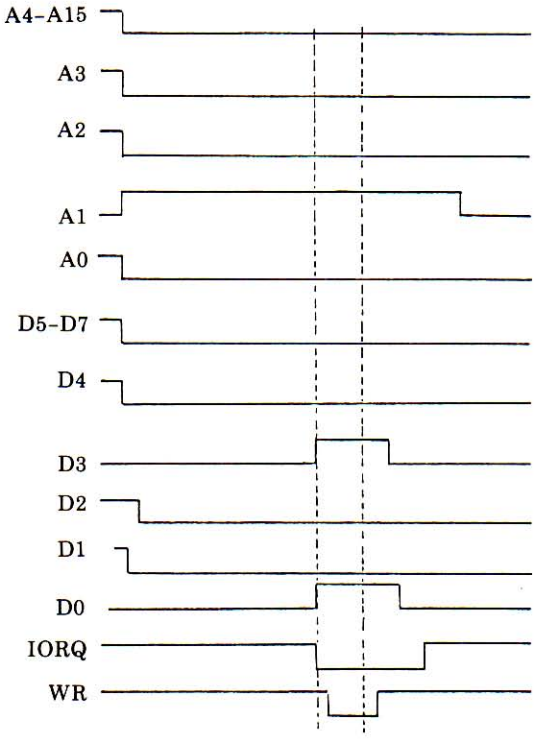


An example of several of the control lines and their logic levels is shown in figure 1.9. The left dotted line shows the change in the line signals at the execution of the command LOAD. The right dotted line shows the point at which the actual operation (that is, loading data) occurs. Notice that several of the CPU's lines respond to the command. Since we are using all address and all data lines in the example, A0 through A15 and D0 through D7, can you tell to what location in memory the data will be loaded? Can you tell what that data is?

If you guessed the memory location as 0002 and the data word as 09, then you were correct. If the data is transferred where the right dotted line intersects the signals, the number on the data bus is transferred to the location determined by the number on the address bus at that point in time. Note that, if the full 8 bits of data were high in our example, the number would be 255. If all sixteen lines of the address bus had been high, then the address would be 65,536.

A diagram of the entire output of the CPU is called a *timing diagram.* The ability to read a timing diagram for a CPU is especially important in

Figure 1.9
Signal fluctuations on the bus in response to a write instruction.



circuit design or in troubleshooting a problem with an interface circuit. All manufacturers publish a timing diagram with their digital components. Most contain complicated engineering notations, but you can usually make them out nevertheless.

Strobe Signals and Bus Sampling

Unless you can capture, or latch, data and address information at one precise instant when all of the necessary information is present on the lines, the information might as well be garbage. The levels change too rapidly to read. But, as in our example, a *strobe flash,* so to speak, of the lines at the right time can make the information quite meaningful. In order to read (or write) data in this instantaneous way, most external devices use what is called a *strobe pulse.*

The so-called strobe can be created in many ways by the CPU, and several of the CPU's lines are strictly for strobe operation. The IORQ is a strobe pulse that signals that an address is ready on the low 8 bits of the address bus. The strobe, then, is a signal that conveys that everything is ready. So, if we want to get valid information off the data bus, we generate a strobe that fires when all of the data is present with the correct address information. Since the Z80 CPU has so many control lines, this isn't difficult. With the proper use of logic gates and decoders, the necessary control strobe can be generated to capture the data. This is explained fully when we construct our first data latch.

The Control Software

Since the Industrial Revolution in the early 1800s, most machines have been independent, task-oriented mechanisms that operated as soon as they were plugged in and turned on (or were otherwise set up). Only recently has this idea of task orientation given way. More and more machines are being built that are *programmable.* The ability of a mechanism to change its task according to a user's program has prompted manufacturers to take a different approach. Program control of heating systems, precision metalworking machines, and even stenographic equipment has brought the use of *programmable equipment* closer and closer to the average consumer. The person preparing dinner who enters several codes into the keyboard of a microwave oven is actually programming a small microprocessor, or microcomputer, if you will, though on a very limited scale. The child who instructs his or her programmable toy to make several turns is, again, using a keyboard to program a microcomputer. The idea of computer programming still intimidates many who do not understand that programming is

merely a form of control that lets them turn on a machine in such a way as to make that mechanism perform certain desired tasks. This is not very different from what we've always done with machines.

Unlike hardware produced during the first fifty years of the 1900s, the computer is a "dead" machine until it gets programmed instructions. Without programs, or *software,* it could not and would not perform a single task.

Early computers were even more dependent on the user than those of today. Many had no instructions with which to work when they were first powered on. Several small programs had to be keyed in manually just to get a computer ready to respond to the user. These small programs were called *bootstraps,* or *boots,* and were entered in machine language. Many times, these simple programs were entered in binary form by means of small single-pole switches on the front of the console. These boots allowed the computer to accept larger programs from paper or magnetic tape. These larger programs were the ones that actually ran the computer, much as do our BASIC programs of today. We've come a long way since then.

The BASIC Language

Most modern-day computers have built-in programs that start operating the instant the machine receives power. These programs (usually on ROM) tell the computer to clear memory, reset itself, print READY on the screen, and perform many other start-up tasks handled in the past by the user.

More important, the computer's ROM also includes a high-level language that makes communicating with the computer easier for the human operator. The most popular high-level language is BASIC. BASIC stands for *beginner's all-purpose symbolic instruction code.* It allows the user to type in English-like or purely English commands such as PRINT and LIST and lets the computer respond with the appropriate actions. The fact that users can talk to computers in something close to English has done more to make the computer a household product than any other single factor.

However, BASIC has at least one flaw: it gets results so slowly at times that you might think your computer has quit. Of course, if you had to follow some assignment using a language other than your own, you probably would slow down somewhat yourself.

Let's say that your employer wants you to take an in-depth printout of figures and data, total them, study them, and format a report for the stockholders of your company. It might take you a day or two. But what if the data and figures were in Chinese (assuming you don't understand Chinese) and all you had to work with was a five-hundred-page translation guide? You would need weeks. Now you know how the poor little computer feels when it has to handle BASIC.

BASIC is not the computer's native language. Remember, it speaks only in 1s and 0s. It has to have an interpreter program to read your instructions, figure out the results, and then reinterpret those results into an output that you can read. No wonder BASIC gets slow sometimes.

Machine Language

Most computers, including the Timex/Sinclair 2068, can bypass BASIC if they are addressed in their own language, machine code. Most BASICs allow you to enter machine code directly into memory and then run that code without having the machine interpret even one BASIC command. Timex BASIC is not that versatile. To use machine code, you have to use a few Timex BASIC commands along the way. But the direct use of machine code (the 1s and 0s) lets the computer perform the programmed actions as much as one hundred times faster than standard BASIC would allow. This fact alone makes worthwhile the study of the difficult art of machine code programming.

The Z80 machine code instruction set is not one of the easiest sets to master. There are more than four hundred separate instructions. Many less sophisticated CPUs have only a hundred or so instructions. But, with such a large array of instructions, the Z80 may easily win the prize for being the most versatile CPU on the market. This is one reason that it was chosen by the manufacturers of the Timex/Sinclair 2068 as the heart of their machine. While it may take some CPUs twenty instruction codes to perform a task such as transferring a block of memory, the Z80 can do the task using only three instructions.

Registers in the CPU

It is very difficult to discuss machine code instructions without first discussing the fact that the CPU utilizes several blocks of storage areas within the chip. These blocks are called *registers.* They are similar to memory locations since they can hold a binary number and transfer it when needed. You might want to think of the registers as little baskets holding the 8-bit binary numbers. The CPU can use these baskets to store, add and subtract, compare, and manipulate these numbers in order to perform a designated task. Figure 1.10 is a block diagram of the registers of the Z80 CPU. Notice that there are twenty-two of them. Several have letter names. Notice that there is one register called A. This is sometimes referred to as the *accumulator* because this is where most of the answers go when the computer figures them out. It is in this register that the CPU stores results, compares numbers, and performs tests on the binary word. It even uses

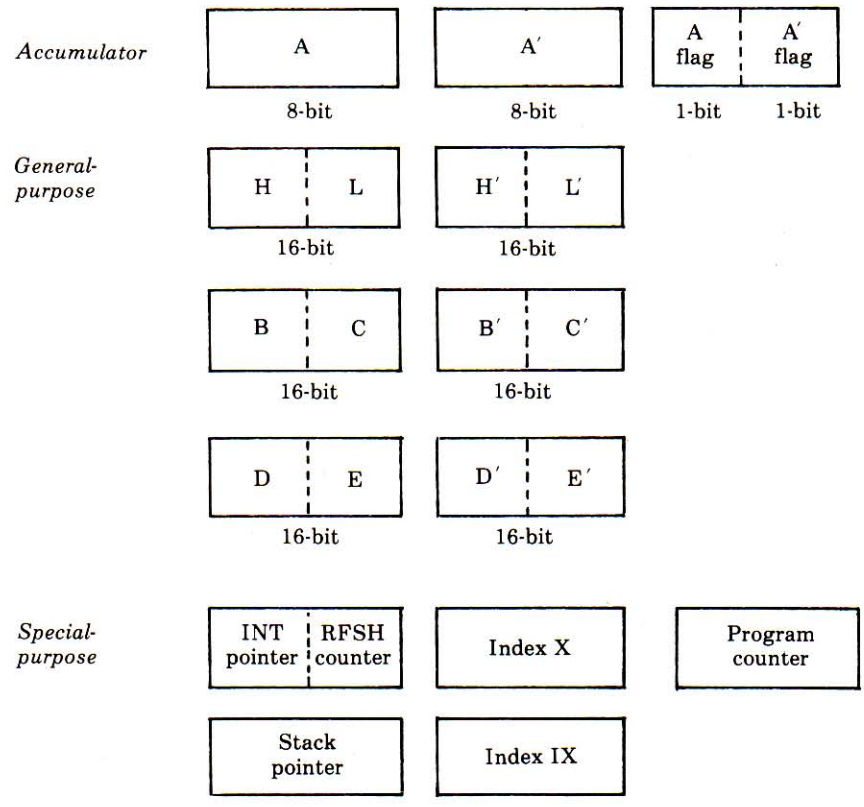


Figure 1.10
A diagram of the Z80 registers. All 16-bit registers can be addressed as single 8-bit registers.

this block when it shifts information in and out of the CPU. The accumulator, or A register, is the workhorse of the CPU. Most of the Z80 instruction set uses the A register in some fashion. Notice that there is also an A′ (*A prime*) register; this is the second half of the A register. A and A′ put together can hold a full 16-bit number. In a similar way, general-purpose registers B, C, D, E, H, and L and their prime versions can hold 16 bits when combined as BC, B′C′, DE, D′E′, HL, and H′L′. The other, special-purpose registers vary in the number of bits they can hold, and each has a special function that will not be of much relevance to our discussion of machine code programming.

The Instruction Set

Appendix C lists many of the more frequently used instructions of the Z80. This table shows the name of the instruction, its abbreviation, both the

decimal and hexadecimal representations, and a comment as to what the instruction does when it is put into action. Several of these instructions are called *single-word* instructions (for example, RETURN). Most of the other instructions include one or more letters that stand for variables. For example, there is an instruction LOAD (NN),A, which means, "Load the contents of memory location address NN to register A." The instruction LOAD A,(HL), on the other hand, means, "Load the accumulator with the data from the memory address specified by the contents of register HL." As you can probably tell, the parentheses mean "the address specified by." If the instruction read LD HL,(NN), the 16-bit number in memory location NN would be placed in the register pair HL. The H register would hold the 8 high-order bits of the number, and register L would hold the low 8 bits.

I strongly recommend that you purchase a good Z80 instruction manual if you plan to do extensive machine code programming. If you plan to use machine code only for control purposes for the projects in this book, you may find the information given here enough.

Using BASIC for Control

Timex BASIC offers three kinds of commands that can be used to operate or control interfaced components: (1) PEEK and POKE, (2) USR, and (3) IN and OUT. Each of these will be discussed in detail in preparation for the upcoming interface projects.

PEEK and POKE

If the control application does not require speed, the use of BASIC's POKE will serve quite well. The POKE command takes a specified number and loads it to a specified location in memory. If you recall, loading does several things to the control bus. Signals on the control lines fluctuate, and these fluctuations can be used to generate a strobe pulse for an I/O device. The POKE command could be a control command for an output port. PEEK works in a similar manner, except that it takes a number from a specific memory location and places it on the bus. The PEEK command, then, could be a control for an input port.

If we had already built our I/O port, we could now see these commands operate. But there is another way that their operation can be demonstrated. This is by the use of the TS 2068's existing I/O device, the display.

Type in and enter the following BASIC command directly, without a line number:

```
POKE 16399,253
```

You may not have seen anything happen, but if you look up at the top of the screen, you will notice something new—a small dash. This is your number as displayed by the 2068. It appears as a dash because of the way the 2068 handles display information. That is, it does not recall the information for display in sequence. The number at location 16399 is now 253, however. Prove that it is by typing in and entering the following BASIC command:

```
PRINT PEEK 16399
```

The response on the display is 253. This is what happened: the POKE command loaded 253 on the data bus and 16399 on the address bus. The CPU then told the memory that there was data available for it. The memory looked at the address, accepted the number 253, and stored it in location 16399. The PEEK command reversed this process by placing 16399 on the address bus and telling the memory it needed data. The memory looked at the address and placed the contents of that address on the data bus, which contents the computer promptly printed on the display.

USR

Each time the Timex/Sinclair machine is powered up, the memory content is read starting at memory location 0000. If you want instructions located in some other part of memory to be read by the CPU, you must tell the machine to go to this other location and start execution of the machine code program located there.

Timex/Sinclair BASIC has an instruction that tells the computer to start reading the program at any location you choose. This is the USR (USER) command. Both of the following direct command lines tell the computer to start reading at memory location 65268:

```
PRINT USR 65268
LET A = USR 65268
```

BASIC IN and OUT

Timex/Sinclair BASIC will allow the user to place a peripheral address number in a command and to command the machine to either bring information into the computer from the external device or send information to a selected external device. If a printer such as the Timex 2040 were connected to the bus, we could send information to it by way of the OUT command. The device number of the printer is 251. If we wanted to send a

number to it for printer activation, we might enter the following command in BASIC:

```
OUT 251,168
```

The printer will respond with a definite jerk when this command is executed.

The same procedure can be used with the IN command. If we changed the OUT to an IN in the above command, any input device that was selected as device number 251 could send its data to the computer with the use of this command. The BASIC form of this command is LET X = IN 194. This allows an external device whose number is 194 to place its data on the bus. The BASIC program will give the value of this data to the variable X. The variable can then be evaluated by the program.

We'll discuss the IN and OUT functions in greater detail later in the chapter.

So the Timex/Sinclair 2068 has several BASIC commands that generate input/output signals: PEEK, POKE, USR, IN, and OUT. Machine language has several instructions that also generate I/O signals. These are LOAD, IN, and OUT. (Even though the names of IN and OUT in machine language are the same as in BASIC, they are handled quite differently in machine language by the programmer. The BASIC commands are much easier to handle than the machine code instructions but are quite a bit slower in operation.)

The above-mentioned commands (in conjunction with many others) generate tiny pulses that tell electronic circuits when to operate. If these are timed and sequenced correctly, just about any electronic circuit can be made to operate. Computer interfacing is a two-part task. First you have to get the circuit operating, and then you have to get the program to operate the circuit correctly. The complicated circuits that operate printers and robots and so on are made up of smaller and much simpler circuits strung together to produce a desired effect. We will cover many of the simple I/O circuits and then string a few together to demonstrate the concept (chapter 4).

Using Machine Code for Control

Several of our projects will need some form of control from software. Most of the examples are given in BASIC. However, there may be a time when the speed of BASIC is not sufficient to handle all of the requirements of

checking all of the conditions necessary for proper operation. If the inputs have to be scanned for a certain condition, and the action has to be initiated fast, your best bet is to use machine language as much as possible. Using machine code may make an application work when BASIC fails. Let's look at the way machine code routines are handled in TS 2068 BASIC.

In the memory areas of the Timex/Sinclair machine, the memory can be overwritten as more and more BASIC lines are added. You can see that inserted machine code routines might get in the way of BASIC and, of course, be similarly wiped out. But there is a way to set up an area in memory that will be protected from overwriting.

The easiest way to protect a block of memory is to set aside a chunk of memory above the random-access memory (RAM). Simply make your first statement in the BASIC program read like this:

```
10 CLEAR 65267
```

This effectively sets aside one hundred spaces that can hold machine language and not get written over. You can make the space as long as it needs to be. A hundred spaces should be enough to practice with for now. Remember, though, your machine codes will start with address 65268, or 1 above the number you cleared.

Now that we have a place to POKE our code, let's look at ways to get our machine code into this area. We'll use the simple method first, of course, which is simply to POKE decimal numbers in until we get enough of a machine routine with which to work. Rather than doing this 1 byte at a time, though, we'll use a little BASIC program to do it for us. Type in the following *decimal loader* program:

```
 1 REM "DECIMAL LOADER"
10 CLEAR 65267
20 LET X = 65268
30 PRINT "ENTER DECIMAL INSTRUCTION"
40 INPUT Y
50 POKE X,Y
60 LET X = X + 1
70 PRINT Y
80 GOTO 30
```

The address 65268 is simply the starting address at which we insert our first machine code instruction. This little BASIC program will keep inserting as long as we input numbers. Don't insert more than one hundred since the computer would then start running out of space, and the program would probably crash or lock up as a result.

Now we need a machine code program to enter. (This may seem complicated, but a little perseverance will get us through it.) Since our only

output device at present is the display, let's use it as the device to which we communicate with our machine language routine. We'll use one of our own I/O ports later.

The display can be thought of as a block of memory that can hold as many as 768 bytes. It usually contains fewer since, most of the time, the screen is not completely full. The location (or starting address) of this block is 16384.

Now that we know where our output is going to be located, we can develop a small machine language program to output to this area.

The display is a form of memory, so let's write a short routine to load a number to the display area. By looking at Appendix C, we see that we can load a number into register B (LOAD B,N) and then load this number from the register to memory. The LOAD immediate instruction, 06 decimal, means to take the number and load it into the B register. We can also see from Appendix C that we can load the B register contents to the location pointed to by the number in the HL register. We do this with decimal 112. We point HL to our memory location with the instruction LD HL,N, or 33 decimal.

We know that the display starts at location 16384. Since we don't want to place our displayed number in the first location of the screen (it's difficult to read there and has a tendency to foul up the display), let's skip over on the screen some ten spaces to location 16394. Because each memory location holds only 8 bits, or binary digits, and since 8 bits can represent decimal numbers up to only 256, we must break up the address (16394) into smaller numbers that the computer can read. We can do so by a technique established for just such a purpose: dividing the address by 256, which will give us a quotient and a remainder, both small enough for the CPU to handle. Now, if we divide 16394 by 256, we get a quotient of 64, with a remainder of 10. The Z80 will understand these two numbers as address 16394 if we enter them in the order *remainder, quotient* (10 and then 256).

Now we need something to place on the screen. The character code 253 is a CLEAR command. Let's use it as the number we'll place in memory and see on the display. We are using 253 since it gives a good indication on the display (a small dash). Then, by using the instruction 112, we can place the contents of the B register into the address in HL. Our code, then, should read:

```
06,253,33,10,64,112,201
```

What, you may ask, is that 201 on the end? This is the instruction for RETURN. It is important that it always be placed on the end of a machine code routine since it tells the computer to return to BASIC. Otherwise, the machine would try to execute the next character it saw as an instruction

and would probably go haywire. *Remember always to include a return (201) on the end of any coded routine.*

Run the small decimal loader program that we input above, and enter the decimal numbers as they are listed. Once all seven numbers have been entered, input a letter in response to the ENTER DECIMAL INSTRUCTION prompt, and the program will stop with an error code. We are now out of the program.

The only task remaining is to get the machine code to run. In Timex BASIC, you accomplish this task with the USR command. Clear the screen by entering CLS directly, and get ready to run the machine code.

Since the starting location of our code is 65268, type in the following command, and enter it directly without a line number:

```
LET A = USR 65268
```

When the ENTER key is pressed, the machine executes the code at location 65268, and, lo and behold, you print a dash on the screen without using BASIC (other than to get the routine started). The reason that the character prints as a dash is that the information for the character on the display is read out to the screen from nonconcurrent locations. That is, the character is broken up and stored in fragments in various locations. When only one location is poked, only part of the character is displayed. But don't worry about this for now. What's important is that the machine code program worked. By using a few more instructions, we can see more clearly the speed of machine code operation.

If we want to print ten dashes on the display, we need to set up a counter with the number 10 in it, decrement it each time a dash is printed, increment the memory location pointer, compare the counter with the number 0, and, if all ten dashes have not been loaded to the display, jump back and do the load routine again. If you look at the partial instruction set in Appendix C, you can see that there are all the codes we need for incrementing, comparing, and jumping.

We first want to point HL to our memory location. Do this as in the last example by entering 33,10,64. To set up the print counter, use register C and load it with 10. This is done with the LD C,N instruction, or 14,10.

After printing one dash on the display, we need to decrement the C register so that it will read 9. This is done with the DEC C instruction, 13.

If C is not equal to 0, we must JUMP back to the load instruction again. However, JUMP needs to know where to go if the C register is not 0. In this situation, we use a relative jump instruction, JR NZ. This is followed by a negative number that tells how many spaces to go back including the space taken by the negative number. Figure 1.11 shows the decimal numbers that are given after a JR instruction to show negative direction. That is, if the instruction is to jump back nine instructions, the number following

Decimal	*Reverse jump*
255	−1
254	−2
253	−3
252	−4
251	−5
250	−6
249	−7
248	−8
247	−9
246	−10
. . .	. . .

Figure 1.11
Reverse jumps. Backward jump instructions in machine code are made up of JR and one number. JR,253 means to jump back three memory locations. You can get the program to jump backward 128 locations using this convention.

JR NZ is 246. In our counting, we must count the second digit in the JP instruction as zero.

If all of our ten dashes have been entered into the display memory, this relative jump instruction is skipped by the program. The final instruction, RETURN, is executed. Now our routine reads as follows:

```
33      Load HL
10,64   with the display location
14,10   Load register C with 10
06,253  Load register B with 253 (dash)
112     Load register B to memory (HL)
35      Increment HL
13      Decrement register C
32,251  Jump back if decrement did not give 0
201     Return
```

Enter the complete routine using the decimal loader, and run it as we did before. Note the incredible speed of operation. If this routine had been in BASIC, you might still be waiting for it to finish. Since the ten dashes are strung together, the resulting display is a continuous line. You just learned to use machine code to do graphics fast.

You can see how a simple machine code routine could be used to send a number to an output port that is wired to respond as a memory location. A machine code routine could even check to see if a number at

an input port is of a certain value (you would need to specify the address of the port, of course).

Hexadecimal Code

Up to now, we have been using decimal values for our code; they work with our BASIC. Most reference manuals on the Z80 instruction set, however, use a form of coding called *hexadecimal,* or just *hex.* It is much easier to use than the decimal form. Since hex uses, at most, two characters to represent decimal numbers as large as 255, it is easier to enter and appears more readable on a page. Since most of the charts and tables in the manuals list each code by its hex value, looking up a certain code is almost impossible unless you understand hexadecimal notation.

If we (like the computer) could count with only two digits, we could count only four one- and two-place numbers (0, 1, 10, and 11). This is counting in the *binary* number system. Beyond 11 (binary) we would have to count with numbers containing more than two placeholders (100, 101, 110, etc.).

In the *decimal* system, of course, with its ten digits, we can represent one hundred numbers (0 through 99) as one- and two-place numbers. Beyond 99 (decimal) we have to count with numbers containing more than two placeholders (100, 101, 102, etc.).

The beauty of the *hexadecimal* system is that it allows us to represent 256 numbers as one- or two-place numbers. It does this by using the letters A through F as single digits to replace the decimal numbers 10 through 15. Thus, we can translate 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (decimal) to 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (hexadecimal). The fact that hexadecimal allows us to count 256 numbers using, at most, two placeholders can save us a lot of keystrokes and several electronic components that would have to be used to process additional columns (places). The chart shown in figure 1.12 gives the decimal number for each hexadecimal number 0 through FF and allows you to obtain the hexadecimal value of any decimal number 0 through 255. The left column is the first hex digit and the top row is the second hex digit. The chart can be read for finding the hex value or the decimal.

We could enter all of our previous codes using hex if we could get BASIC to accept them. Since our BASIC works only with decimal numbers, we would have to convert hexadecimal values to decimal values to make them acceptable.

There are several standard ways in which hex can be converted to decimal, and BASIC is capable of making such a conversion before the final decimal numbers are poked to memory.

Since we will be entering both numbers and letters when using hex, the numerical INPUT statement of the decimal loader program we have

LEFT HEX DIGIT	RIGHT MOST HEX DIGIT 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Figure 1.12
A decimal-hexadecimal conversion chart.

been using will not work. We can, however, input the hex number as a string and then evaluate that string to get the required value in decimal form. Consider the hex number 2A: the hex 2 stands for 2 × 16 (decimal), and the A stands for 10 (decimal). These two decimal numbers add together to equal 42 (decimal): 2A (hex) = 42 (decimal). A short BASIC routine that makes such a conversion is as follows:

```
 1  REM "HEX LOADER"
 2  CLEAR 65267
 5  PRINT "ENTER START ADDRESS IN DECIMAL"
10  INPUT A
15  PRINT "ENTER HEX INSTRUCTIONS"
20  INPUT X$
22  IF CODE X$ = 113 THEN GOTO 85
25  LET X = CODE X$(1) * 16
30  IF X > 912 THEN GOTO 40
35  LET X = X - 768: GOTO 45
40  LET X = X - 1392
45  LET Y = CODE X$(2)
50  IF Y > 57 THEN GOTO 60
55  LET Y = Y - 48: GOTO 65
60  LET Y = Y - 87
65  LET Z = X + Y
70  PRINT A; " ";Z
73  IF Z > 255 THEN BEEP 1,0
75  POKE A,Z
```

```
76  LET A = A + 1
80  GOTO 20
85  STOP
```

Consider again the hex number 2A. Our first digit is a 2, which has a character code of 50 as shown on page 241 of the 2068 user manual. The code of X$(1) is determined and multiplied by 16 in line 25. The result is decimal 800 (50 × 16). Now, the 2 in the hex number 2A has a value of 32 (decimal), and indeed the program subtracts 768 (line 35) from the 800 to yield the correct decimal value. When the character code of the A in hex number 2A is determined in line 45, the result is 97. The program then subtracts 87 from the 97 to yield the actual decimal value (10) of the hex A. Line 65 adds the decimal numbers 32 and 10, and the result is 42 (decimal). Lines 30 and 50 check to see if the character in the hex digit is alphabetic or numeric. This little program will work for all two-digit hex numbers. Remember this routine since all of the machine code we enter will be in hex from now on.

The OUT Instruction

The OUT command is the machine code instruction that generates device-select pulses directly from machine code. If the OUT immediate instruction, D3, is executed as OUT (N),A, the address N appears on the low eight address lines, and IORQ goes low. Since the address lines total only eight, there can be no more than 255 devices that the CPU can address with this type of select. This method allows the peripheral device to be totally separate from memory. That is, the peripheral does not take up any memory area as with the memory-map technique. Since the 2068 uses all of its memory address lines, we will make extensive use of the OUT instruction in BASIC.

The following BASIC direct command generates a specific device-select pulse that can be connected to an output port:

```
OUT 194,1
```

Figure 1.13 is a general diagram that shows how this instruction might be decoded using a multi-input NAND gate.

The Timex BASIC OUT command generates the same sequence of events that machine code OUT initiates, but in a much slower fashion. The 194 is the *device number* but could as easily be called the *device address* since it is the number that is placed on the lower eight address lines when the command is executed. If we had made the device number 1218, the result would have been the same since the lower eight address lines

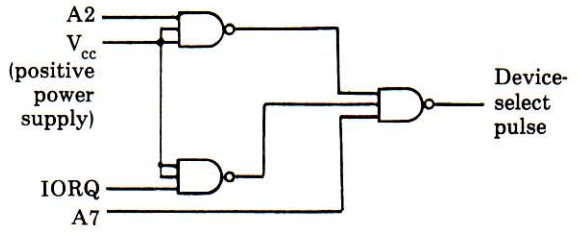


Figure 1.13
A diagram of a simple device-select decoder using three sections of a 74LS10 three-input NAND gate. This type of decoder can have overlapping select numbers that are not compatible with the entire system.

would have contained only the decimal 194. The higher part of the address, A9 through A16, would have been lost.

The IN command is similar in nature to the OUT instruction since it too sends the IORQ line low and places a device-select address on the eight low address lines. The following BASIC direct command shows the form that this instruction takes when executed in Timex BASIC:

```
LET X = IN 194
```

Since IN is used as a function in Timex BASIC, it has to take the above form. But this form means simply that a device pulse is generated for device number 194, and the data that is placed on the bus by this device is put into variable X. This is a neat way to read information into the program for use from an external source.

The only difference in these two instructions is that OUT drives the write line low, and IN drives the read line low. By using the combinations of the address lines, the I/O request line, and either RD or WR, you can fashion a device-select circuit that will give a pulse when all of the conditions are right.

Even though the circuit shown in figure 1.13 decodes the conditions of both IN and OUT, it has several flaws that make it unusable for many purposes. It does work for many applications, however. The TS 2068 controls the printer with a circuit very similar to this. If you have a 2040 printer connected to the backplane, try the following BASIC routine:

```
10  OUT 251,168
20  GOTO 1
```

This little routine drives the printer crazy. Don't let it run too long since it has a tendency to eat up your paper supply.

Ambiguous Decoding

The circuit shown in figure 1.13 will decode several different device numbers as the same one. This make it *ambiguous*. That is, you get pulses consistent with several different addresses even though you entered only one address, 194. If we examine how this circuit generates its pulse, you can see why.

If you know how NAND gates work, you can see that if pin 13 had a low, and pins 1 and 2 had a high due to being connected to a positive power supply (V_{cc}), the output on pin 12 would be high. The same would be true of the signals on pins 9, 10, and 11. Pin 8 would be high.

Since pins 8 and 12 are high, the next gate has two highs and whatever is on address line A7 (a high in the case of the printer). Now that all the pins have a high, the output of the gate (pin 6) goes low. Since it takes two lows on the NOR gate to produce a high strobe pulse, the RD or WR line has to go low. It does when an IN or OUT instruction is given, and the strobe is generated. Otherwise, it always remains low. This pulse tells the printer to turn on. The other bits in the data word tell it to print dots. Now let's see why more than one address can do the same thing.

We said that A2 had to be low and A7 had to be high. Let's look at the data bits on the address lines when this is true. They would look like this for address 251:

Address line	A7	A6	A5	A4	A3	A2	A1	A0
Binary	1	1	1	1	1	0	1	1
Value	128	64	32	16	8	4	2	1

You add the values under any 1s that appear in the binary row to get the device number. In this example, the device number is 251 (the sum of all the values except A2's value of 4 since we had to leave A2 low to make the circuit work). But notice that A1 and A0 could also change without affecting the circuit. So the printer should also operate if the device number is 250, 249, or 248. It does, in fact, do exactly this. Try it with the above BASIC routine and see.

This ambiguous decoding is fine for most purposes. There might be times when it could cause problems. It does cut down drastically on your device-selection numbers, as you can see. Therefore, some other, more sophisticated decoding circuit needs to be implemented that is not so unreliable. The 74LS154 decoder (chapter 3) is just such a circuit.

2

Interfacing to the Real World

As you have seen in the preceding chapter, the personal computer is a natural choice for controlling external devices. All of the necessary signals are present for operating various kinds of devices and doing tasks that were once impossible for any kind of electronic apparatus.

Although you may not know it, the same kinds of circuits as make up your personal computer control many things in everyday life. Many heating systems of large office buildings are controlled and monitored by computer around the clock. Traffic lights are programmed and operated from a central location. The holiday turkey may have been cooked in a microwave oven that a CPU controlled. Your auto probably was painted by a mechanical arm that was positioned by a small computer. Why can this type of control not be implemented with respect to many other items in our everyday life for easier, faster, and more inexpensive operation? The truth is, it can. All that is needed is a small personal computer such as the Timex/Sinclair and the proper interfacing components.

Proper computer interfacing is joining the computer and an external device in a compatible manner so as to achieve a usable and coordinated action on the part of the device.

If a computer is attached to an electric typewriter, but you cannot get the typewriter to type in any meaningful fashion, the computer is not interfaced—or, at least, not interfaced correctly. The rest of this book deals with the proper interfacing of your computer to items that you might wish to control. Although it is not yet realized, my dream is ultimately to connect my computer to my metal lathe, insert the metal stock, type in the dimensions, and let the computer do the rest. I detest having to stand at the lathe, waiting, calibrating, waiting, and waiting some more.

There is, believe it or not, enough power in the Timex/Sinclair, if properly interfaced, to handle any of the following tasks:

1. Making coffee
2. Monitoring the heating and cooling of your home
3. Monitoring a security system
4. Serving coffee
5. Programming your television viewing
6. Starting and warming your car in the morning
7. Getting the morning paper (or mail)
8. Typing your personal letters
9. Washing your coffee cups (I like coffee)
10. Answering the phone
11. Mowing the lawn
12. Vacuuming the carpet
13. Setting the lighting of rooms
14. Running the power equipment in the shop (including the lathe!)
15. Watering the lawn
16. Teaching computer control

The list is infinite. The personal computer (with the right interfacing) can be made to do almost any task to which you set your mind. The interfacing is the important factor, however. Without it, the computer is limited to printing its power away on the screen or printer. That limits you to mainly number crunching and game playing.

The Control Devices

Appendix D shows pinouts of some of the components that are used for interfacing computers to other devices. These ICs are but a few of the many available that convert those tiny computer signals to usable output. (Other interfacing chips are available that are sometimes as complex as the computer itself. These chips serve their purpose and many times make life easier for the designer of interface circuits. But they are difficult to find, many times, and they have a tendency to discourage rather than help the beginning hobbyist.) One of the more popular ICs of the large-scale integration (LSI) interfacing group, the Intel 8212 I/O port (represented in Appendix D), is used as a project component in this book. You will see how the component count can be reduced by its use.

As you look at the pinouts of the ICs in Appendix D, do not be discouraged by their apparent complexity. The ICs are really quite simple. The operation of each will be explained as it is used in a circuit, and once

the circuit is complete, you will undertand its ICs completely. You may even be tempted to try some of the more complicated LSI devices as the world of interfacing becomes less mysterious to you.

Backward Designing

Any new peripheral that is connected to your computer must first be designed. This designing is usually done on paper unless you are one of those experimenters who designs by what is called the *smoke test* method. With this technique, you waste no time with schematics or theory. You simply start plugging in ICs until something works. This method is not recommended due to the high cost of burned out components or your burned out computer. There is a time for smoke testing, but not until some preliminary paperwork is done, at least. While you are designing on paper is the time to get the theory of the components, their operation, and the chip requirements down in some workable order. Designing on paper also gives you a blueprint from which to work and, in most cases, cuts down on the smoke when the circuit is finally powered up. You must make the drawing look as though the circuits will work, according to the literature. This won't guarantee you success, however. You must actually connect the wires to determine if, in fact, the book contained all the details and you understood them correctly.

Before you start a design, you must clearly think out the major task to be accomplished. That is, you must establish exactly what it is that you want your computer to do. This becomes your starting point for designing an interface that will perform the job. Once you determine the goal, you are then ready to take the necessary steps backward in the design process. You determine the control requirements, the power needs, the signals required, and the electronics. Finally, the actual control programming can be tackled. Let's take a simple task as an example and see how this procedure works.

Suppose that you want to have the computer flash a small light-emitting diode (LED) each time you press a key. This simple operation by the computer would be the *major task* that you determine. Your next step is to determine the control requirements. In this example, only a 5-volt signal to the LED is required. The computer operates at this voltage level, so this should present no problem. Then you need to select interfacing components that provide a signal of the necessary type. Transistor-transistor logic (TTL) is a good choice here. Your next step is to design a circuit that will respond to the key-press signals on the bus. Finally, you write a short program that will monitor the keys and activate the LED each time a key press

is detected (unless the hardware handles this). You'll see how this method is used when we get to chapter 4.

The Interfacing Components

Now that you know the design method, let me detail several interfacing ICs for study and future reference. These brief descriptions and the corresponding pinouts in Appendix D do not show all the capabilities of these few ICs. If you need further details or want to explore other interfacing components, several good reference manuals that are available would make excellent additions to your bookshelf. These books show the pinouts, voltages, and operating conditions and even give example circuits.

This section explains those components that will be utilized in our interfacing projects as well as other projects you may design.

Figure 2.1 lists both a TTL component number and also the number of the matching complementary metal oxide on sapphire (CMOS) component for a number of ICs, all of them digital logic gates.

Pinout charts are shown in Appendix D for the ICs listed in figure 2.1. These charts can be very helpful to you in designing and troubleshooting. The pins have been numbered and marked in the charts according to their function. I have tried not to use abbreviations in the function labels, except for V_{cc} for the positive power supply connection and GRD (ground) for the negative connection (two common abbreviations with which you should be familiar). Some manuals denote ground as V_{dd} and the positive terminal as V_{ss} (for *drain* and *primary source,* respectively).

Many of the descriptions given here are very brief, and a whole book could be written on each of several of the components. I will expand on each component's function when it is used in a project.

We will be using *positive logic,* which means that 1 will represent a positive level of voltage, and 0 will represent negative, or ground. Most popular logic manuals take this approach. The diagrams of the circuits in Appendix D are viewed from the top so that pin 1 always appears at the top left of the IC next to the IC's *mark.* This mark is usually a slot cut in the top of the chip or a raised dot near pin 1. When you wire-wrap, remember that this order is reversed.

Most of the ICs are multidevice chips. That is, each chip contains more than one gate. The diagrams in Appendix D show all of the devices within the chip, but in a schematic drawing only one of the gates is depicted, for clarity. Most of the time, the pin number in a schematic is given on this single gate.

You'll notice that many of the ICs are called *quad.* This means there are four of these gates within the chip. The designation *hex,* in this con-

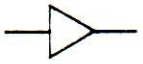
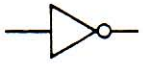
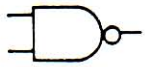
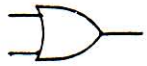
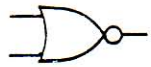
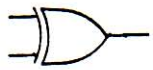
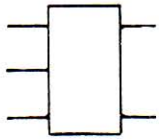
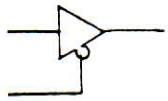
Symbol	Name	TTL number	CMOS number
	Hex noninverting buffer	7407	4050
	Hex inverter	7404	4049
	Quad two-input AND gate	7408	74C08
	Quad two-input NAND gate	7400	4011
	Quad two-input OR gate	7432	74C32
	Quad two-input NOR gate	7402	4001
	Quad two-input exclusive OR gate	7486	74C86
	Dual JK flip-flop	7473	4027
	Quad Tri-State buffer/driver	74125	None

Figure 2.1
Digital logic gates.

text, means that there are six separate circuits within the chip. Many of the 7400-series ICs have been designed to draw very small amounts of current. These low-power ICs carry the designation LS within their original TTL number.

Quad Two-Input OR Gate

The OR gate takes two (or more) signals, and outputs either one or the other. If either input is high, the output will be high. Only if both inputs are low will the output be low. This is an excellent component for use

with active low control lines. If a peripheral device needs a low only when two control lines are low, use the OR gate to generate the device-select pulse by tying the two control signal lines to it.

Quad Two-Input NOR Gate

The NOR gate is basically an inverted OR gate. That is, it will generate an output if either of its inputs is active. Since the NOR gate inverts the output signal, the output will go low any time one of its inputs goes positive. If both inputs are negative, it will generate a high output. This is a very important component in the proper interfacing of components that rely on the timing of other signals. It is a good component to use if two signals have to be low before a high is sent to another component (for instance, if an active low on the WR line must be coincidental with a low on the device-select pulse). If an inverter (74LS04) is tied to the output of an OR gate, you effectively have made a NOR gate. This arrangement is slightly slower in operation, but it may save on components if the response time is not critical.

Hex Inverter

The inverter is the IC component whose only task is to take a signal and reverse its polarity. That is, if the signal is low, the inverter makes the signal high, and vice versa. The chip contains six inverting circuits. Most interfacing circuits contain at least one of these components since at least one is usually necessary for component signal matching. Each section of the IC has only one input and one output. The inverter is probably the least complicated (and most used) of all of the interfacing chips we will cover.

Remember that an absence of any signal on the input pin will make the output float high (or at least fluctuate to high levels). If any input of the chip is not used in the circuit, it is a good idea to tie these unused inputs to ground. The pins that are being used can get noisy if unused inputs are left unconnected. You will mainly see the CMOS version of this chip in the diagrams in this book, but the 7400 series will work too.

Quad Two-Input AND Gate

The AND gate is an all-or-nothing gate. In order for the output pin to be positive, all of the inputs have to be positive also. It is a gate that is similar in operation to a series of switches that are strung together, with each switch representing an input. If one switch is open, the output signal just doesn't get through.

It is truly a gate. If one of the inputs is used as a control pin, it can cut the signal off (or let it through).

Quad Two-Input NAND Gate

If an inverter were placed on the output of an AND gate, the result would be a NAND gate. In the NAND gate, if all of the signals are positive, the output is negative. Otherwise the output remains positive.

Again, this is a good control gate. And it inverts the signal at the same time!

Quad Two-Input Exclusive OR Gate

There are times when you want a negative pulse when two inputs are both either low or high but never when the inputs are different from each other. This can be accomplished with the exclusive OR gate, or XOR (as most manuals call it). It produces a low only when both of its inputs have the same polarity. No other gate has this kind of exclusivity. This is a good component to use for inverting a signal from one polarity to another. Simply tie one of the inputs to a double-throw, single-pole switch, and choose the polarity of the signal at the output by placing either a plus or a minus on the other input line.

Quad Tri-State Buffer/Driver

Most TTL devices generate a low when the input pin is not receiving a signal. Essentially, there is a logic 0 on the line at all times when it's not in use. If the device is connected to the data bus, complete havoc results. What can be done? Don't all TTL devices do this? No, the Tri-State buffer doesn't because it has three states. It has a regular logic 1, a logic 0, and, finally, a disconnected, or open, state called *Tri-State.* When the input pin is not receiving a signal, the device acts as if it is completely disconnected from the bus. Thank goodness for a device that can do this. I don't know what we did before its invention.

The 74LS125 quad Tri-State buffer/driver is one of the most popular interfacing chips in the TTL category because of this special property. There are now many other three-state buffers and similar devices, but the 125 is probably the leader with hobbyists and experimenters since its cost is very low.

There are four buffers in the device that you can separately three-state by placing a logic 1 on the control pin. The whole 8-bit data bus can be handled with just two of these ICs. However, they are not bidirectional:

they cannot send information in two directions. But we can use four and just turn the inputs in the other direction on two of them. Just remember to keep the control pins opposite in polarity since these chips self-destruct immediately if their outputs are tied together and are of different polarity!

Dual JK Flip-flop

A very handy component in digital circuits is the *flip-flop,* or *bistable latch.* This little circuit can capture data or divide a frequency by 2.

The JK flip-flop has two outputs, Q and not-Q (represented by Q with a bar over it). Each time the clock input pin receives a negative transition in a clock pulse, it will flip, or *toggle,* from one state to the other. This, of course, assumes that both the J and the K control pins are tied to a plus voltage.

Hex Noninverting Buffer

There is one CMOS device that deserves special mention. This is the hex buffer (4050). The 4050 is faster and consumes much less power than its 7407 TTL cousin. The buffer is an important device in digital circuits. It does several things that are necessary and beneficial. When a line is buffered with the 4050, the signal gains strength. It is able to drive more outputs. The device's capability to drive inputs of a number of other circuits is called its *fan-out.* The fan-out of a TTL device is low; it has a difficult time driving more than three TTL inputs (in fact, three might stretch its capability). If you add a noninverting buffer, the signal might drive as many as ten TTL inputs before starting to disintegrate.

More important, the buffer offers protection to the CPU, which might otherwise suffer serious damage if a wiring error is made. It is even possible that an input could short and, without a buffer, feed a devastating signal or current directly to the fragile outputs of the CPU. The protection is well worth the effort required to buffer lines coming straight from the CPU and even other LSI chips of the microsystem. Therefore, one of our first projects will be to buffer the address lines.

Quad Bilateral Switch

The 4016 CMOS and the 4066 CMOS are identical in pinouts (see Appendix D). They differ in that the 4066 is faster and has less resistance when on. That means it works better than the 4016, so we will use the 4066 if at all possible.

The quad bilateral switch is similar to the 74LS125 Tri-State buffer in that it can connect and disconnect a line from the bus. It serves to buffer the line, also. It acts as a connection when the control pin is high and acts as if it's disconnected when the control pin is low. However, it goes further: it is bidirectional (able to feed a signal in two directions). The 4066 is a perfect component to use on the data bus since the data bus is also bidirectional.

Seven-Segment Decoder-Driver

There is an integrated circuit (not shown in Appendix D) that will take 4 binary bits and change them to the proper combinations for driving a seven-segment display. The 74LS48 will decode the proper segments to read the display in decimal. There are other ICs that will decode the 4 bits to make the display read hexadecimal. With the 74LS48, you no longer have to make the binary conversion mentally. All of the decoding is done within the chip. A binary number of 1001 appears on the display as a 9. This decoding of 4 bits at a time is called the *binary-coded decimal* (BCD) form of writing binary numbers.

The outputs (pins 9 through 15) are active high. You might find that some seven-segment readouts require an active low. Simply invert all of the signals with hex buffers to obtain proper operation.

One-of-Eight Decoder-Multiplexer

When using memory-mapped techniques to generate device-select pulses, you decode the address lines so that the decoder responds with a select pulse each time a certain address is called or activated. This would be an almost impossible task with just individual gates. But there is a readily available component that will do this very thing for you. The 74LS138 (see Appendix D) will take 3 binary bits and output eight separate combinations depending on the pins activated by the address lines. The device is enabled when its pin 6 is made high and its pins 5 and 4 are made low. Then the combinations of addresses on pins 1 through 3 cause a low to appear on one of the eight output lines.

The 74LS138 is highly recommended as the decoding component in memory-mapped techniques since the select line needs a high for enabling. Since the Z80's address lines are active high, this chip is a natural choice. Most memory chips need an active low for enabling. The outputs of the 138 fit well here because, when selected, these are low select signals.

Four-to-Sixteen-Line Decoder-Multiplexer

The 74LS154 (see Appendix D) is very similar to the 74LS138 in that it, too, decodes bits of the address bus to generate a special select pulse on its output pins. This will be our choice for the main device-select generator.

There are times when more than eight different selections are required for a specific application. If this is true, a larger chip will be needed to accomplish this task. The 74LS154 is a twenty-four-pin device because the outputs alone take up sixteen of the pins on the chip. This IC decodes 4 address bits, whereas the 74LS138 decodes 3. We now have sixteen different combinations due to the fact that we now have 4 binary bits. However, the enable pins (18 and 19) must be low for the chip to decode properly. The 74LS138, in contrast, needs both a high and a low on its enables. The 74LS154 will output a high on all of its outputs if either of its enables is low.

Quad Two-to-One-Line Decoder-Multiplexer

It is often the case that two signal paths must be controlled and routed to one device (but never two at the same time). The switching of a signal from two different sources is called *multiplexing*. The 74LS157 (not shown in Appendix D) is a component that will switch a signal from two sources to a single path. The device contains four sets of data path selectors that are controlled by one pin. Placing a low on the enable pin (15) allows the chip to function. A high on the select line routes the signal from device A to the output, while a low on the select pin routes the signal coming from device B.

Other Useful Interfacing Integrated Circuits

This list of ICs could be very much longer. However, there are several other components that warrant special mention here. These special ICs are used in our interfacing projects, and the operation of each will be explained more fully as it is used.

8212 I/O Port

We will build an output port for the Timex/Sinclair 2068, but you will see that our output port has inherent problems. Since it only latches data from the data bus and holds it for use, it could cause bus problems if it were

connected to the data bus as an input device. Because it is not a three-state device, it would need three-state buffers if the same circuit were used as an input port. The reason that this is true is that the latches would be reading either high or low all the time.

The 8212 (see Appendix D) has gated three-state buffers, eight parallel data latches, and all the necessary control logic to perform as either an input or an output port. It has the necessary logic to generate an interrupt pulse to tell the microprocessor that it has data ready. This special signal can give the 8212 top priority with incoming data.

Analog-to-Digital Converter

One of the most important aspects of computer control is allowing the computer some knowledge of its environment. Then, by having the computer make decisions using this knowledge, you have taken the first real step in creating artificial intelligence. The knowledge the computer gains because of its ability to access its surroundings is called *feedback.*

However, a small problem exists when environmental feedback is implemented. The world is mainly an environment of *analog* signals. (This means that the signals or stimuli are not black or white, but various shades of grey, so to speak, as found in light, touchable surfaces, smells, and sound.) The computer has to have a way to change these analog signals to digital codes that it can recognize. It can do this neatly with the ADC0801 analog-to-digital converter.

The ADC0801 (figure 6.6, chapter 6) can take a varying voltage and output it to the data bus as an 8-bit word that can be interpreted by the CPU (and, of course, your program). The ADC can be read as if it were a memory location.

The ADC0801 can read a varying voltage between 0 and 5 volts. It, then, is perfect for the TS 2068 system. The reference voltage (V_{ref}) pins are tied to the positive and negative supply, which sets the voltage range. The chip has its own sampling clock and the electronics necessary for proper timing. A complete description and pinout diagram is shown in the section dealing with analog-to-digital conversion (chapter 6).

AY3-1015 UART

In our earlier discussion on the CPU's signals, we saw that there is an advantage in converting 8 bits of data from parallel to serial. Then this serial data, which takes the form of 8 consecutive single bits, can be transmitted (or received) on a single line. The device that accomplishes this conversion is the universal asynchronous receiver/transmitter.

The AY3-1015 (see Appendix D) is such a device. It is a high-speed serial receiver or transmitter whose modes can be switched by logic cir-

cuits. A high-to-low transition on the data strobe (DS) line transfers data from the data bus into the transmitter holding buffer. The low-to-high data transition starts a bit-by-bit transmission of the word, including the start and stop bits.

This device is a three-state device, so very little is needed in the form of interfacing hardware.

Optocoupler—Triac Output

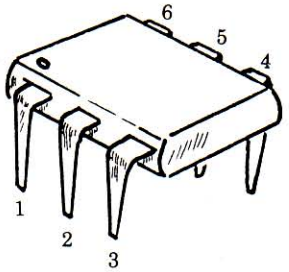
There are instances when you should not directly connect your computer to the controlled device. If the controlled device operates with 115 volts AC, you would have good reason to worry if it were directly connected. If there were a short in the equipment, that voltage could be fed directly into your little 5-volt computer. Needless to say, this would be the end of most of the components and microchips. They all would probably go up in a puff of smoke. The MOC3010 optocoupler, as shown in figure 2.2, is a six-pin device that lets you control a high-voltage, high-current line without actually connecting to it. It does this with light.

If your computer turned on a small LED, and the light from the diode activated a photosensitive silicon-controlled rectifier, the interaction would be an instance of what is called *optical coupling*. The MOC3010 has enough drive to turn a high-current triac on and off. Using the MOC3010 can be a great way to control home appliances.

Discrete Components

Interfacing circuits almost always need a few individual, or *discrete*, components such as transistors and capacitors. More and more of these components are being packaged in integrated-circuit form. One day soon, no individual components will need to be used whatsoever. We are almost to that stage now. Even transistors are being produced in arrays inside sixteen-

Figure 2.2
The MOC3010 optocoupler. This optocoupler IC protects the TS 2068 circuits.



pin integrated circuits. The mechanical relays of yesterday are now *solid-state.* Resistors and diodes are headed in the same direction. You may find that you will still need a few of these single components in your projects from time to time, however.

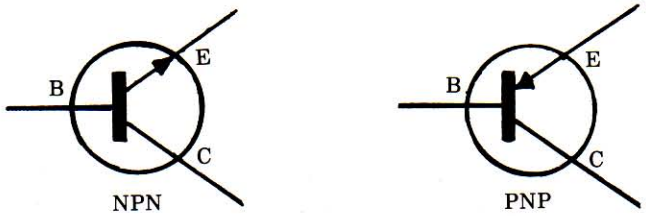
Transistor

The lowly transistor is probably the most versatile of all the discrete components. Figure 2.3 shows the standard schematic diagram of the two types of transistor, the NPN and the PNP. The letters N and P stand for *negative* and *positive,* respectively, and refer to the way the silicon layers are arranged inside the case. The NPN silicon type is the type most used in digital circuits. This type switches off and on rapidly and is not as susceptible as other types to damage from being highly saturated or heavily conducting. I never could quite remember which type I was looking at on the diagrams. One electronics instructor suggested I read the NPN as meaning "Not Pointing iN." Sure enough, the arrow on the emitter was not pointing in, and I've remembered it to this day.

A transistor is only an electronic switch very similar to a relay. When a small signal is applied to its base connection (B), the transistor will conduct a much larger signal across its emitter (E) and collector (C). It acts very much as an amplifier does, in this respect. Since digital circuits switch a transistor from completely on to completely off, I think of transistors as switches.

How do you tell which is the emitter, which the base, and which the collector? Appendix E shows the standard configuration of most of the various case styles. The leads are marked, and their placement is fairly standard, also. Most schematics you see will have the cases shown and the leads marked for you. Case styles differ in relation to the heat that the transistor is supposed to dissipate. Usually, the larger the case, the more current (and heat) the transistor can handle.

Figure 2.3
Two types of transistor.



Resistor

A resistor is a small, two-lead component that *resists* current. This resistance is measured in units called *ohms*. The higher the resistance, the greater the number of ohms. A resistance of 10 ohms would be the near equivalent of a direct connection, and a resistance of 100,000 ohms would be the near equivalent of an open circuit. Some resistors are made with resistances as high as 10 million ohms. Most of the ones that are used in digital circuits have low resistances (200 to 300 ohms). These are used mainly as current limiters to protect transistors and light-emitting diodes. There are higher resistances used for *pull-up* or *pull-down* resistors. These resistances are usually around 10,000 ohms, and the resistors connect between the positive or negative side of the power supply and the data or control lines. With this arrangement, the voltage can be raised or lowered to get the level closer to a value that the device can read as a logic level. Appendix F shows the schematic diagram of a resistor. (Appendix G shows resistor color codes.) Most resistors used in digital circuits are the small, ⅛-watt size. You may find larger ones in the power supply circuits, however.

Capacitor

A capacitor is a two-lead device that blocks direct current. The device stores a charge and will release it slowly if power is removed. If the polarity is quickly reversed, it will release the charge rather quickly. Because of this unique property, a capacitor can duplicate a signal on its opposite surface. Therefore, a capacitor will, in effect, transmit a rapidly changing signal but will block an unchanging signal such as direct current. Most capacitors that are found in digital circuits are used mainly in the power supply to smooth out the alternating current. Many times, small ceramic-wafer capacitors are used along the power leads to cut down on power spikes that occur in the power leads due to high-frequency switching components. Appendix F shows the standard symbol of capacitors used in electronic circuits.

Diode

A diode is a small piece of semiconductor that lets current flow in one direction but not in the other. The two leads, as shown in Appendix F, are called the *anode* and the *cathode*. The anode is the lead shown with the arrowhead, and the cathode is the one shown with the blocking line. Current flows through the diode if the anode is connected to a positive voltage and the cathode is connected to a negative. The characteristic of the diode thus illustrated is called its *forward bias*. If the leads were reversed, no current could get through the diode. A diode can be used for decoding

circuits and for *rectifying* a signal (that is, letting the positive pulse through but blocking the negative pulses).

Light-Emitting Diode

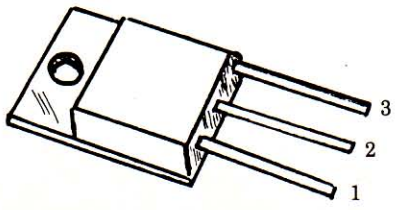
It was discovered years ago that a conducting diode actually glows when current passes through the diode if there are certain impurities in the diode's silicon. If a lens of glass or epoxy is placed over this glow, the diode can be used as an indicator. These diodes have become known as light-emitting diodes (LEDs). Since they draw very little current and produce a great deal of light, they are used as indicators of current flow and logic states and have even been made into seven-segment numerals for calculators and watches. It is necessary to place a resistor of around 300 ohms on the anode or cathode of the LED to keep it from drawing too much current and burning itself out. Appendix F shows the schematic representation of an LED with the current-limiting resistor attached. Notice that the only difference between its symbol and the symbol of a regular diode is the circle and arrow. The circle represents the lens, and the arrow represents the emitted light. Since these indicators can be driven directly from logic gates, we will use them as our indicators.

Triac

Most home appliances such as lamps, coffee makers, and radios pull less than 6 amps. Their voltage runs around 115 to 120 volts alternating current (AC). If an output of the computer were connected to this type of power, it probably would go up in smoke. What is needed is a device that will take a very small signal and control the power of one of these appliances. A triac is such a device.

The triac, as shown in figure 2.4, is similar to a power transistor or silicon-controlled rectifier. A small voltage applied to the gate of a triac

Figure 2.4
The 400-volt, 6-amp triac. This component handles 120 volts.



will allow its other two leads to conduct current up to 6 amps at a voltage up to 400 volts AC. *Caution: This amount of current is deadly! Do not touch any live circuit with this amount of voltage and current.* See complete details on circuit handling given in the triac project (the three-channel home-appliance controller, chapter 4).

Optocoupler

An optocoupler, as we have discussed, is a device that uses light to control another device. It is made up of an LED and a light-sensitive transistor. The transistor does not have a base connection, but, when activated by light, it starts conducting just as would an ordinary transistor. These two components placed end to end create a light-controlled junction that cannot be damaged from a component failure. What's even better, this arrangement prevents a component failure (such as a short circuit) from damaging your computer. Appendix F shows the schematic representation of an optocoupler.

Prototyping

Prototyping is the actual wiring of the circuit after the design has been committed to paper. The reasons for this type of nonpermanent construction will become quite clear once a circuit is tested. Most prototyping takes the form of either *breadboarding* or *wire wrapping.*

Breadboarding

I recommend that most of the projects in this book be *breadboarded* as a first step in their construction. This method is nonpermanent and, in most cases, nondamaging. It gives you an opportunity to try the project out before you make it a permanent piece of work.

You will need the following items for proper breadboarding:

1. *The breadboard.* The Radio Shack modular breadboard socket 275-174 is recommended, although there are other prototyping boards that are just as reliable. The modular breadboard comes with its own power rails along the sides, and ICs are simply inserted down the middle of the board. Four connection points are left for each IC pin. There are more than enough pin connections for most projects. You can obtain additional connections by properly using the extra rows of pin sockets.

2 *Hookup wire.* This can be almost any wire that will fit into the socket connectors, but the wire should be one solid strand of about 24 gauge. Our breadboard will accept any gauge size from 30 to 22. I've found that the best hookup wire is standard color-coded telephone cable wire that has been precut into varying lengths. Most phone repairpeople throw enough away on each job to last an experimenter a lifetime. I find that many will give you wire scraps just for the asking. More on the wire later.

3 *Color-coded test leads.* These handy little jumper cables are sold in packs of ten by most electronic stores and are invaluable for most electronic prototyping. Since breadboarding necessitates power and other outside connections that don't work well with hookup wire, the jumpers with small alligator clips are a must.

That's not a long list, is it? That's the beauty of breadboarding. The circuit can be made operational without all the fuss and bother of the more permanent methods.

Wire Wrapping

The method of permanent wiring that is preferred by most experimenters is *wire wrapping.* Wire wrapping is not new, but this method has seen a surge in popularity in the past ten years due to the increased availability of wire-wrap material on hobby supply shelves.

Wire-wrap is a wiring method that is more expensive than breadboarding but takes many of the headaches out of rewiring and debugging a permanent circuit. The ease with which wire-wrap is done and undone makes the extra expense well worth it. Wire wrapping is recommended for several of the projects listed later. These projects are the ones that will be permanently built and put to useful work. You'll need the following material for proper wire wrapping:

1 *Wire-wrap board.* This special board (figure 2.5) is sold by most electronic stores, such as Radio Shack. The Tandy part number is 276-152. This board is drilled to accept standard wire-wrap sockets and is terminated with a forty-four-pin male edge connector.

2 *Wire-wrapping tool.* There are several different versions of this little gadget. The best is the one with the wrap connection on one end and the unwrap connection on the other. Most have a handy wire-stripping slot in the center of the handle that eases the chore of stripping the ends of each strand of wire.

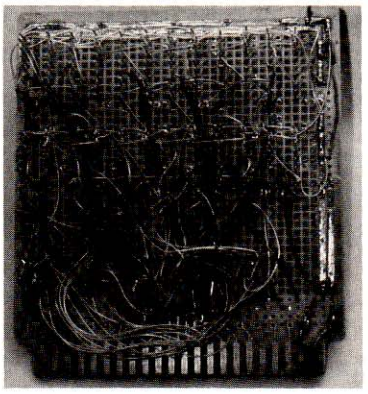


Figure 2.5
A wire-wrap board for prototyping.

3 *Wire-wrap wire.* You'll need at least three spools of 30-gauge Kynar wire. This is sold in 50-foot rolls. It is a good idea to buy three different colors. The three colors make circuit tracing easy if a problem arises. Some outlets sell this wire in packages of precut lengths. Though a bit expensive, this precut wire saves hours in construction time.

4 *Wire-wrap sockets.* IC sockets come in either solder-tail (st) or wire-wrap (ww) versions. Solder-tail sockets can be used only with printed circuit boards. Wire-wrap sockets are supplied with extra-long leads that are made especially for wrapping. You can get either three- or five-level types. This means that the posts are long enough to take either three or five wire-wrap connections. I recommend that all of your sockets be five-level, if possible. The post should be at least 0.025 inch in size, and it does not have to be gold-plated.

There are several advantages to wire wrapping. The most important is that wire-wrapped parts are not soldered permanently, but the connections have the same characteristics as permanent solders. If a mistake is made, the connection can be unwound, and a new connection can be made. The wire-wrapped board is not as neat as the printed circuit but sometimes works better due to the fact that the scrambled wiring eliminates cross talk between wiring paths. Cross talk is the picking up of the signal on a wire by an adjacent wire. When the wires are scrambled, the tendency toward cross talk is canceled out by the many crossovers of the leads. (Printed circuit wiring is notorious for cross talk unless it is specially designed.)

The layout of the sockets on the board is important. The sockets should be close together, but not so close that the posts can be confused with

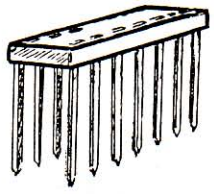
their neighbors. A good separation is around ¼ inch between sockets for beginning projects. The chip sockets should all be oriented in the same direction, either vertically or horizontally. The number 1 pin should always be at the top or on the right. Improper layout leads to mistakes. Select a layout method and use it consistently.

Most hobbyists cut small strips of white paper, write the IC numbers on them, and glue these strips between the two rows of pins. (I wish I had a nickel for every time I wired up the wrong chip before I learned this trick.) To ensure that you do not select the wrong pin of the socket, it is a good idea to mark the number 1 pin with a red marker on the pin side of the board, especially if you have not worked with wire-wrap before. Most diagrams show the number 1 pin on the chip's left side at the top. This arrangement is reversed if you work from the bottom of the chip, as you will when you are wire wrapping. On very large chips (those with twenty-four or more pins) it is a good idea to number every other pin to facilitate the pin counting that you will have to do each time a connection is made.

A typical wire-wrap socket is depicted in figure 2.6. When you make a connection to a post, you must wrap the wire at least five turns around the post. This usually uses about ¾ inch of wire. I have gotten by with fewer wraps, but the chances of getting a bad connection increase dramatically as the number of turns decreases. Figure 2.7 illustrates the proper method of wrapping a post with wire. The resulting connection is as good as or better than a soldered one.

Of course, not all of your components will be ICs. Some will be discrete components such as capacitors and resistors. These components do not lend themselves well to wire wrapping since their leads are not conducive to being wrapped with wire. One way to handle discrete components is to use push-in terminals like the one shown in figure 2.8. They are nothing more than individual wire-wrap posts that you insert into the board and to which you can solder components. It is possible to wrap directly to a component's leads, but the chances of a resultant bad connection are very high. Even if the connection is good at first, it not likely to last very long because the lead will oxidize.

Figure 2.6
A typical wire-wrap socket. Many sockets have extra-long posts for increased wrapping area.



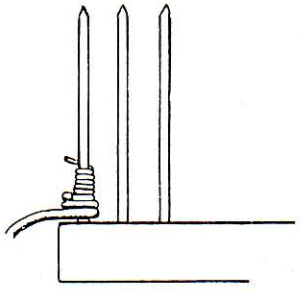
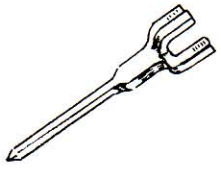


Figure 2.7
A detail of a proper wire-wrap connection. Notice that the insulation is drawn onto the post.

Figure 2.8
A discrete-component wire-wrapping post. This push-in terminal is inserted into the perforated board, and the component's leads are soldered to it.



In summary, breadboarding and wire wrapping are excellent ways to test and construct digital projects. However, there are certain types of projects that do not lend themselves to wiring by these two methods. For such special projects, there are two other wiring methods, point-to-point wiring and printed circuit wiring.

Point-to-Point Wiring

Point-to-point wiring (sometimes called *hardwiring,* also known as *soldering*) is reserved for high-current projects such as those involving power supplies and lighting. These types of projects are usually wired with heavy-duty wire of the stranded and insulated type (16-gauge or larger). One of our upcoming projects will use this method, so don't throw away your soldering iron yet.

Printed Circuit Wiring

Printed circuit wiring is the method that is used by manufacturers in products such as your TV and radio. It is neat and sophisticated and is highly

recommended for professional projects. The money and time required to prepare an acceptable circuit board, however, dictate that we not attempt printed circuit wiring in this book. (I once took about a week to prepare four etched boards for a memory expansion. When I finished, none of the four worked because of a flaw in my diagram. I learned my lesson on that one.)

General Prototyping Construction Techniques

No matter which method you use, certain practices are mandatory. These are mostly commonsense practices, but their use may save hours of frustration, and chip failure once the project is powered up.

Always have enough power available to run the system and the computer. Your computer's power supply can adequately handle the one- to four-IC projects presented later. The larger projects may need a separate supply. A small power supply is detailed in chapter 3. It might be a good idea to construct it first, in the event it is needed. A standard 6-volt lantern battery can be used for most of the experimental circuits.

Operating your system with low voltage (too much current drain) will shorten the life of the ICs and maybe your computer. The components will overheat, and overheating causes them to have what we can call heatstroke (usually fatal). One way to determine if the system is operating at low voltage is to connect your multimeter, or volt ohm meter (VOM), to the power leads when the system is operating. Another way is to look at the display: if there are wavy lines or dark bands moving up or down the screen, the power supply probably is stretched to the limit. (Some Timex power supplies provide more current than others. Many of the earlier models had power packs that were marginal even with just the basic system operating.)

Make certain that the polarity of the power supply connection is correct. Never connect the power leads backward. A small polarized power plug will help. Most ICs can take a momentary reversed-polarity connection, but many newer LSI chips and CMOS types go to dreamland rather quickly.

Try to add at least one despiking capacitor for every IC installed. Specifically, a 0.05-microfarad (0.05-μf) ceramic capacitor should be added across the power connections of the IC. It should be placed as close as possible to the IC itself. Despiking capacitors diminish the spikes in the power leads that are caused by switching of the latches, multiplexers, and so forth. These spikes not only are harmful to the ICs, they also can cause them to present errors in signals on the address and data buses.

Using the Breadboard

The ease with which you can attach the components to each other on the breadboard is amazing. Unlike printed circuit connections, breadboard connections usually follow the schematic closely. If certain procedures and techniques are used, the resemblance of the wiring on the board to the schematic drawing is striking. This resemblance decreases the likelihood of the wiring errors that other wiring methods seem to invite.

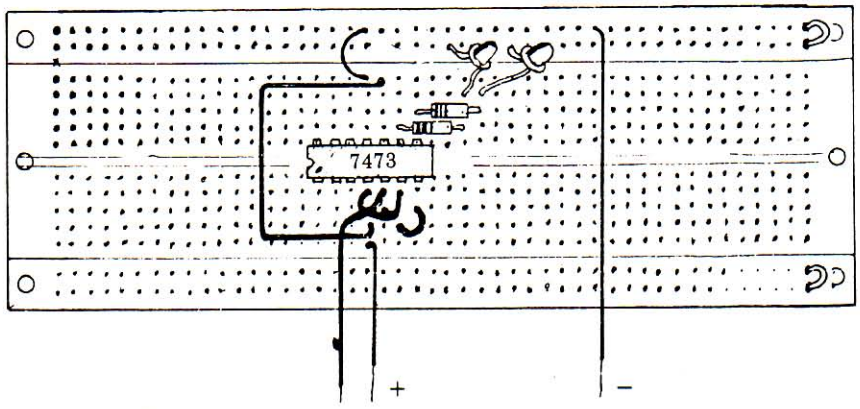
You should convert the wiring diagram to breadboard construction in an orderly fashion. That is, follow the same sequence of steps for every wiring project in order to minimize errors and omissions.

I've found that the best sequence of breadboarding includes several of the following steps. (Certain types of components may require slightly different techniques from those discussed here.)

The Radio Shack breadboard has two sections of common pins, arranged as shown in figure 2.9. The integrated circuits are mounted across the center of the board. The common connections for power are small rails that attach to either side of the breadboard for each of the two power connections.

I usually start by placing all of the components on the board in the same relative locations as they occupy on the schematic. Since the integrated circuits must be placed down the center of the breadboard, though, their locations cannot mimic the schematic in every instance. If a discrete component is shown above a certain IC on the drawing, however, I always try to place the item above the IC on the breadboard. This allows me to locate the component easily when I'm making connections.

Figure 2.9
Radio Shack's prototyping board, with wiring for the pulse detector.



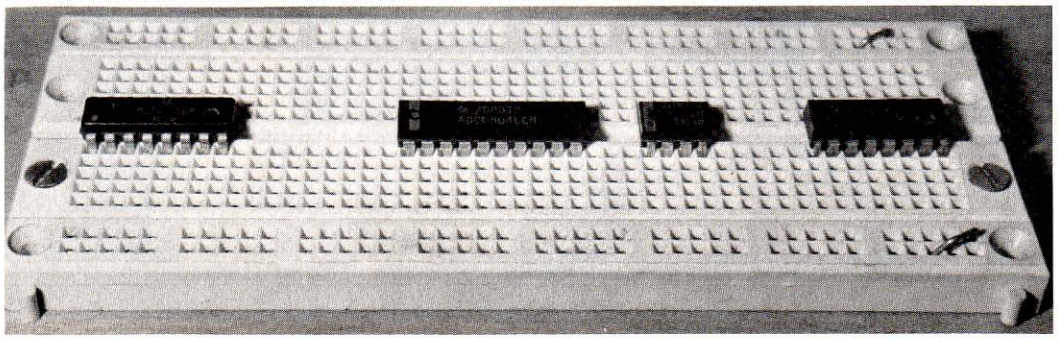
Insert the ICs first. Be extremely careful when you apply pressure to a chip as you insert a pin in the board. You may have to center, wiggle, or adjust a pin to avoid bending it under the IC. The pins of the IC usually come bent slightly outward, and you may need to straighten them for proper insertion. If you insert CMOS ICs first, be especially careful not to let static build up. Figure 2.10 shows several ICs properly inserted into a breadboard.

Most discrete components, such as capacitors and resistors, have leads that will insert directly into the breadboard. Others, such as relays, may need to be temporarily wired in with your jumper cables. The leads of the resistors should be bent 90 degrees at the appropriate point to match the correct pin insertion point. These leads should not be cut off or shortened at this time, however.

You might find it helpful to sketch the breadboard's layout on paper and mark each component as it relates to the wiring diagram. Most wiring diagrams list the components by an alphabetic prefix and a number. Resistors are shown as R1, R2, etc. Capacitors carry the numbering C1, C2, etc. The whole point of these precautions is to eliminate as many wiring errors as possible. (Some errors may creep in no matter how careful you are or how many precautions you take. I speak from experience.)

The final construction step is the actual wiring of the breadboard project. There are many sorts of wire you can use, but you'll find that multicolored telephone hookup wire is quite easy to handle. You can precut this wire in lengths that range from 2 to 6 inches and then strip about ¼ inch of the plastic insulation from each end of every piece. These lengths of wire can then be used over and over again for various prototyping projects. I've found that, if I reserve all of my black and red leads for power

Figure 2.10
The Radio Shack breadboard. Notice how the ICs are mounted down the center. The spaces between the ICs can be used for discrete components.

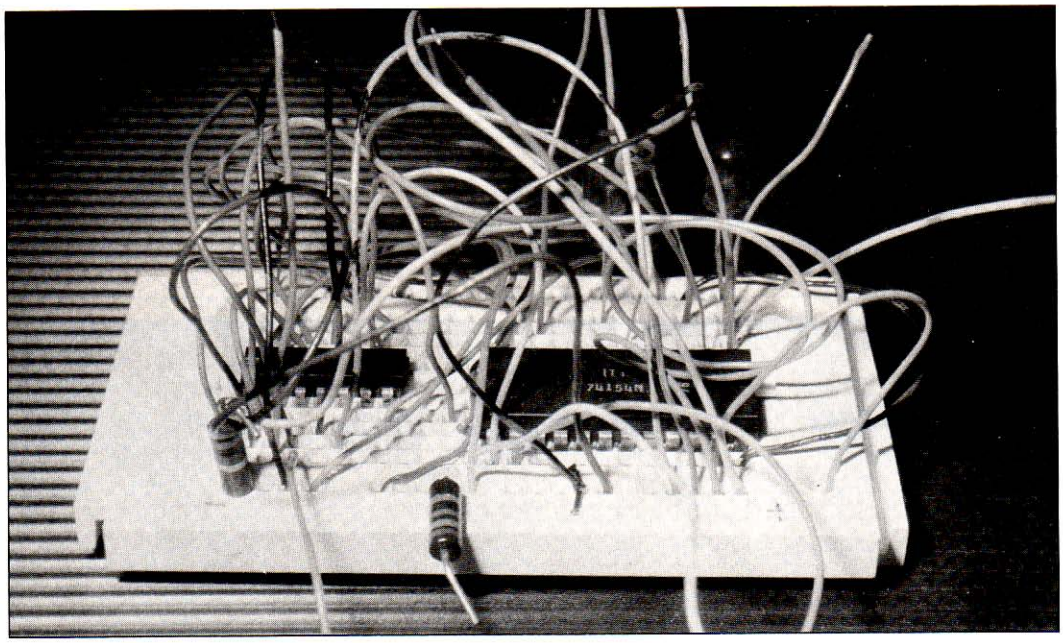


connections only, troubleshooting the circuit is much easier than otherwise. Figures 2.11 and 2.12 show breadboards wired incorrectly and correctly, respectively. Both may work, but the one shown in figure 2.11 may drive you crazy when you have to recheck your wiring and trace signals. The incorrect method also is highly error-prone, though initially faster than the neater method. Remember, when you are building a circuit, accuracy is the most important consideration, not speed.

Make it a habit to mark off each connection on the schematic as the corresponding new connection is added on the board (again, to avoid omissions). Use a small red marker to draw a line across the connection on the drawing each time a wire is inserted. I've found that, when I don't use this method, about halfway through the wiring, I forget which connections I have and have not completed. Make a photocopy of the diagram for this purpose. Marking the original schematic is not recommended.

The first connections made should be the power connections, but *not to the power supply, just to the power rails. Always apply power only as the last step.* Always make sure the ICs have the proper power connection by using your ohmmeter. Simply place one lead into the negative rail, and touch the other to each associated connection point. With your meter on

Figure 2.11
An incorrectly wired breadboard. The circuit may work, but finding errors in the wiring is almost impossible.



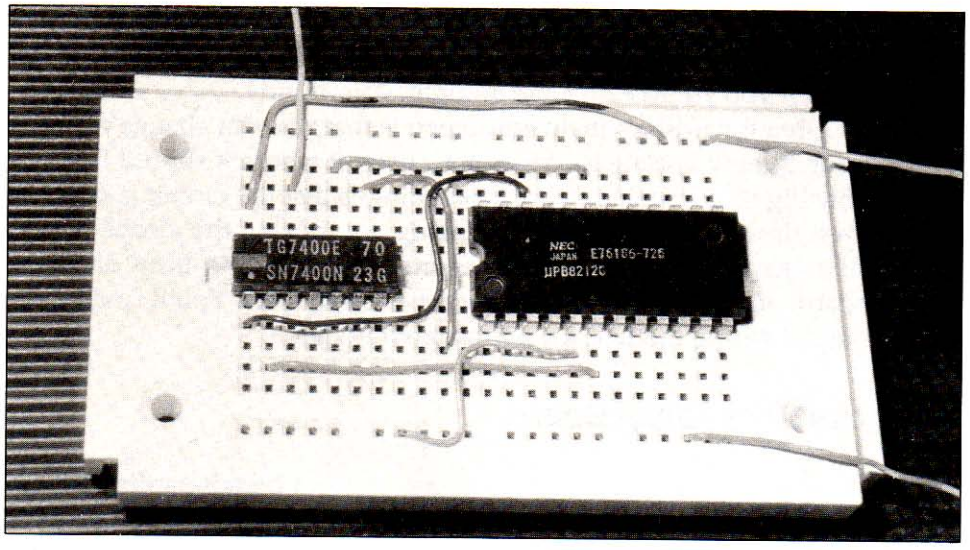


Figure 2.12
A correctly wired breadboard. This circuit takes time to wire but is a snap to correct.

R × 1, the resistance should be low (no resistance = 0) if the proper connection has been made correctly. Do the same for the other power rail and its associated connections. Now comes a very important step. With the rails disconnected from the power supply and with your meter set on the R × 1 scale, place the point of the red lead into the positive rail and the black lead into the negative rail. If all the ICs have been inserted and all the other components have all of their direct power connections, the resistance should be at least 10 ohms. If the resistance is zero, there is a problem. You probably have a short, and you should recheck your power wiring closely at this point. Make certain that your plus connections and your minus connections are going to the right IC pins, as shown on the diagram. If an IC is wired backward, it usually will heat up and will more than likely be ruined.

Once you have wired all of the breadboard's power connections (except to the power supply), you can start the general wiring. I find it easier to wire by sections of the diagram. Most diagrams group the components by circuit function, such as the oscillator, the amplifier, and the input sections. Wiring each of these groups helps you keep track of your connections. This is very much better than wiring all the resistors, then all the capacitors, and so on.

Since we will be breadboarding digital circuits, you'll have to wire buses quite frequently. Again, a bus is a multilead path that data and address signals take to and from an interfaced device. The data bus usually

needs eight connections, and the address bus can require up to sixteen. Always wire these connections as a group. Missing a connection on the address or data bus can do funny things to the operation of the circuit (and, if you are working late at night, your mind).

Breadboarding's main advantage is that you can change it if you find an error. Simply unplug the wire, and insert it where it should have gone. The only big disadvantage to breadboarding is that the circuit is nonpermanent. Even though it works fine for testing, try moving the circuit around and it falls apart. Wires come loose, parts start dangling from all sides of the board, and complete disintegration soon follows. You'll need a more substantial wiring method for permanent circuits.

Using the Wire-Wrap Board

Many of the construction techniques used in breadboarding are used with wire wrapping also. The sequence of wiring the power components and groups is the same. The checkoff of the connections is also similar. The layout of the components is somewhat different, however.

Lay out the components for the wire-wrap board by function group. Place the components that connect directly to the computer's backplane, such as the address buffers, close to the copper fingers of the board's edge connector. This layout method not only saves long runs of expensive wire but also lessens the chances of cross talk and *line loading. Line loading* is the capacitive effect long runs of wire tend to have on the signal due to several factors, such as stray radio-frequency waves and AC hum in the area. These things have a tendency to dampen the signal or cause signals to be misread.

Remember, keep the leads as short as possible in all of your construction, to promote the circuit's operation.

The trick to wire wrapping is in the technique of connecting the wire to the post. Tight and even turns are important. They ensure a good connection that will not loosen, oxidize, and lose signal. Your wrapping pressure must be steady, and you need a lot of practice before you can consistently achieve a good wrap. Again, figure 2.7 shows a properly wrapped post.

You will want to be neat, but don't be! Scramble your runs as much as possible. This is especially important for circuits that use high-frequency switching (address lines, for example). Scrambled wiring is called a *rat's nest* since it looks like something a rat built. Don't let its looks bother you. It eliminates cross talk, and it works. That's what's important. It is a nightmare to troubleshoot, however. It is highly recommended that three different colors of wire be used for the various functions. Red should be used for positive voltage and all address information lines. Blue should be used

for all negative power connections and all data-related lines. And white should be used for all control lines, such as those coming from the control bus. This will help you tremendously in tracing the connections if the circuit fails to operate perfectly the first time. (Each of our project discussions will contain a short troubleshooting section that will also help.)

Radio Shack wire-wrap board 276-152 is shown in figure 2.5, but there are others that are just as good. You might want to check around for the best bargain. I chose to use a board that has a forty-four-pin male edge connector since these are easier to obtain. Others might be used. Most wire-wrap boards have power traces that run around the outside of the board. The traces carry the power to all parts of the board for easy (and short) connections. Avoid these traces when you insert the wire-wrap sockets. The forty-four-pin edge connector's fingers terminate on the same end of the board by small copper lands that need to have wiring posts inserted. Figure 2.8 shows one of these posts that insert into the traces (the same post that is sometimes used for mounting a discrete component).

The Wire-Wrap Socket

Wire wrapping to the pins of integrated circuits is possible but not recommended. The connections have a tendency to fall off. The use of wire-wrap sockets is virtually a must. These make the expense of wire wrapping even higher, but they are worth it. You can even use them to mount discrete components with success, as I have said. Figure 2.13 shows a wire-wrap socket so employed.

We discussed some wire-wrapping techniques earlier, but there are a few standard items that you might want to try when you are making permanent connections with the wire-wrap board.

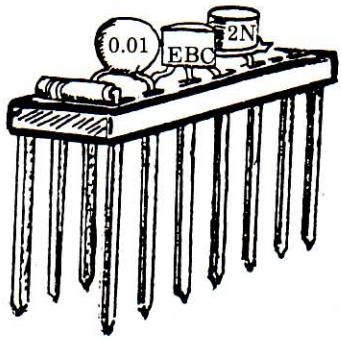


Figure 2.13
A wire-wrap socket used as a component carrier. Make sure that pins of all discrete components make contact in the socket.

The Wire-Wrap Tool

In order to wrap the wire correctly on the socket post, you need a special tool. There are several models available, two of which are shown in figure 2.14. I prefer the type that contains all that is necessary for handling standard wraps. One end has a slot that is specially designed to wrap good, tight connections. The other end has a special slot that unwraps. This will be an important end of the tool since you are apt to unwrap as much as you wrap in the beginning.

Daisy Chaining with Just Wrap

There is a childhood pastime called *daisy chaining*, the stringing together of daisies (in my case, actually, it was clover) to make necklaces. A similar method is used in wire wrapping.

OK Machine and Tool Corporation makes a special wire-wrapping tool called *Just Wrap* (figure 2.15). It can split the insulation on the wire as the wrap is being made. Using this tool eliminates the drudgery of stripping

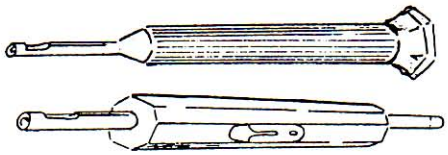
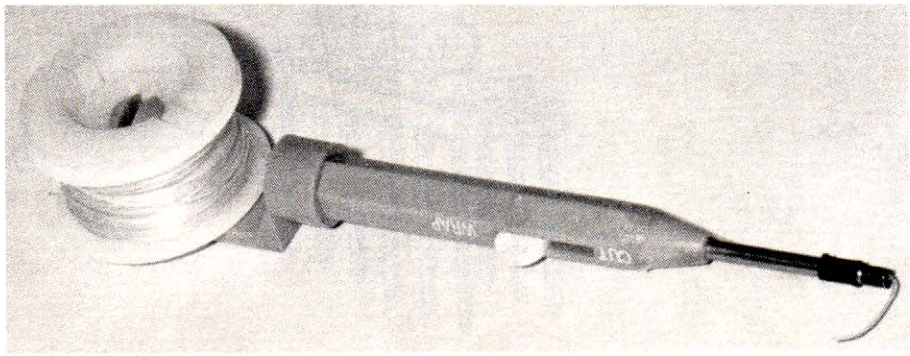


Figure 2.14
Two examples of wire-wrapping tools. Both are sold by Radio Shack and local electronic suppliers.

Figure 2.15
The Just Wrap tool for fast and easy wire wrapping.



each small piece. In fact, the wire comes off of the tool in a continuous fashion. With the Just Wrap tool you can daisy-chain the wire so that the wraps are continuous as long as the wire continues on the schematic. Figure 2.16 shows daisy-chained wire down one side of a wire-wrap socket.

You should avoid daisy-chaining any power leads, however. The small wires do not carry a great deal of current. Most of the time a lot of current is not necessary. But if you daisy-chained a power lead (and power leads can run all over the board), the small wire might be trying to supply power to as many as ten or twelve power-hungry ICs. They might not be able to get the current they require if the power leads have been daisy-chained. Do these connections with separate wires. Make a connection back to the power trace every time power leads are wired to a new chip.

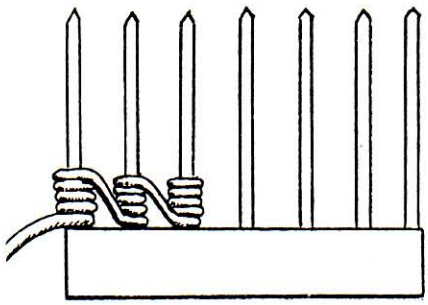
More Labeling

Besides labeling the ICs and pins on the bottom of the board, you also need to label each socket on the IC side of the board. You might think this is too much labeling, but, again, later circuit tracing will be eased considerably as a result. These labels can be nothing more than small white strips of paper glued beside each socket that indicate IC number. If there is more than one socket per IC number, each of the sockets should also be designated A, B, or whatever on both the label and the schematic. I feel a circuit can't have too many labels unless they start covering up the ICs.

The Motherboard

If you wire-wrap and you use a forty-four-pin board, you need a means of connecting the board to the 2068. The *motherboard* provides such a means. It is simply a rack with several forty-four-pin female connectors wired in

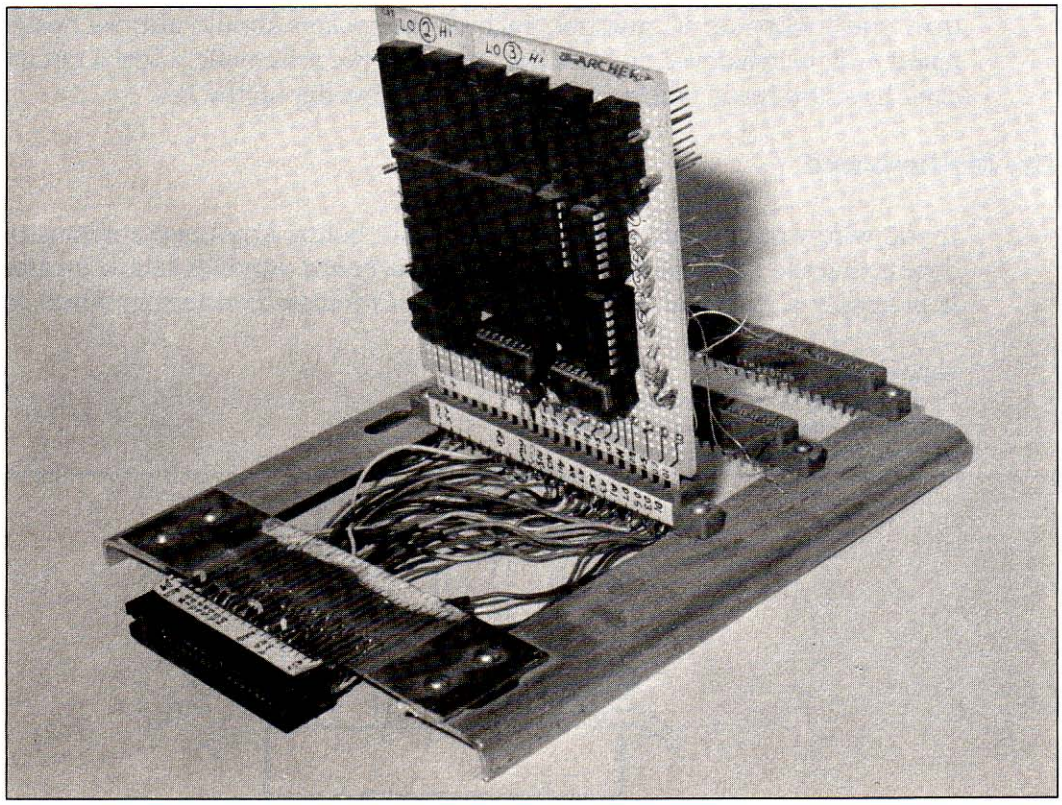
Figure 2.16
Daisy chaining done by the Just Wrap method. At least five turns per post are required for proper connections.



parallel. The connectors are wired directly to an interface connector similar to the one we will construct in chapter 3. This rack can contain several of these board connectors, but three should be enough for our purposes. Figure 2.17 shows a completed motherboard.

Now that you have read about all of the construction techniques, I think it's time we built a small but very useful circuit so that you can get some experience in using the breadboard and in the soldering technique. We will first test the circuit on the breadboard and then hardwire it for permanent use. This project is the *digital logic probe,* or *pulse detector.*

Figure 2.17
A completed motherboard, by means of which wire-wrapped circuits are attached to the TS 2068. This one has slots for three boards and a place to attach an interface cable.

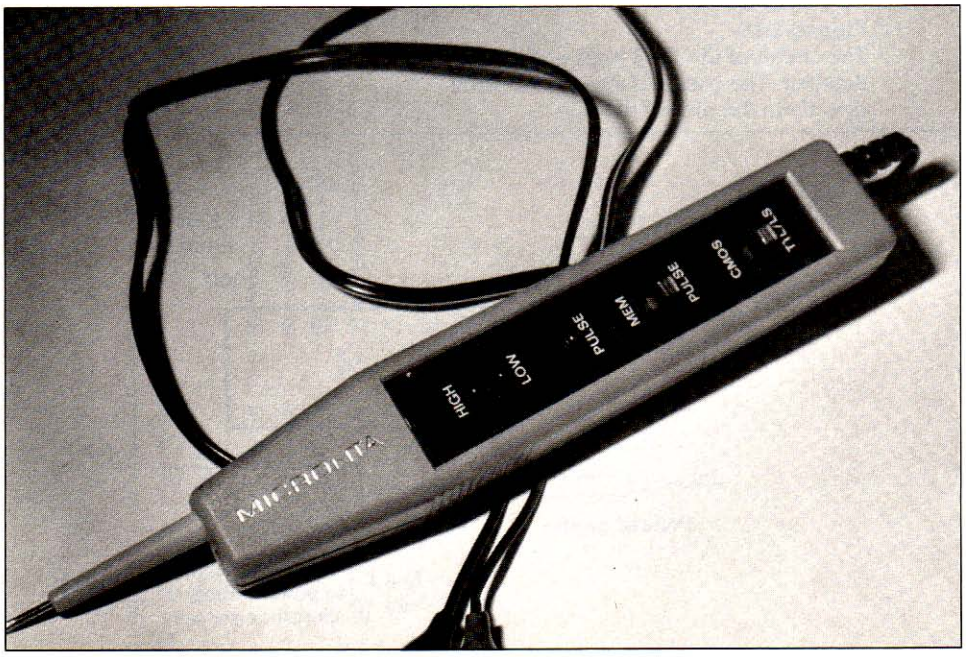


Constructing the Pulse Detector

Many of the signals that the computer uses are rapidly changing from high to low. The signals' highs and lows represent bits of information that are traveling on the lines and telling the components what to do. We will be using many of these signals for control later in the book. By now you should have learned to use your volt ohm meter. The manual for your tester probably told you that it was not a good instrument for testing rapidly changing voltage levels. If the manual didn't say it, it is still quite true. A VOM measures voltage, but it does so at only a partial level when the signal is changing.

What we really need for proper signal testing and verifying is a *logic probe*. This is a device that has indicators that tell you what is happening on the bus or control line at any time. It can tell if your signal is pulsing, if it is positive or negative, if it is a stationary logic 1 or 0, and even something about the duration (time length) of the pulse. This is a handy little gadget to have. Figure 2.18 shows a photograph of one of these units. You

Figure 2.18
A logic probe. This one is sold by Radio Shack, but almost any will work for our purposes.

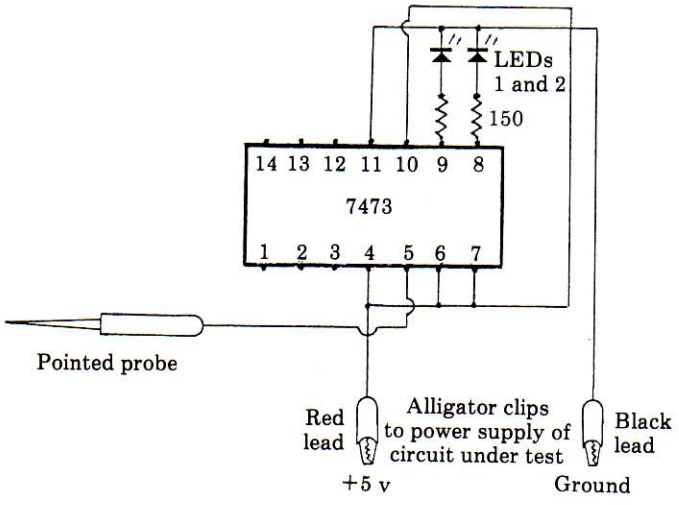


simply touch the probe to the point that is of interest. The LEDs do the rest. You attach the probe's two clips to the device's power points to give the probe voltage. You might check around to see if there is a probe on sale, but you may still find its cost quite high. With your VOM and a little ingenuity, however, you can easily construct a probe that is inexpensive and simple to operate. Now, in the diagrams of the interfacing chips in Appendix D, there is a chip numbered 7473. This inexpensive little IC is called a *flip-flop*. It gets its name from the fact that it can flash from a 1 to a 0 (high to low) if a pulse is applied at the correct pin. (This kind of switching is called *toggling*.) Why not use one of these (along with your VOM) to give yourself an indication of pulses that are present and even their duration? You can do it very easily and with very few parts. Once it is built, you'll have a little tester that can tell you a great deal about what is happening in your circuit while it is operating. Several of the project writeups that follow tell you to check certain points with your *logic probe*. You can use a high-priced probe if you have one, but this little circuit does just as well.

Wiring the Pulse Detector Circuit

Figure 2.19 is a schematic diagram of the connections you must make to build your pulse detector. Remember the discussion on ICs and discrete components. You'll notice that the only other components you need are two LEDs and two low-ohm resistors. Now is this simple, or what?

Figure 2.19
The circuit of the 7473 logic probe, or pulse detector. The connections may be made directly to the pins of the ICs.



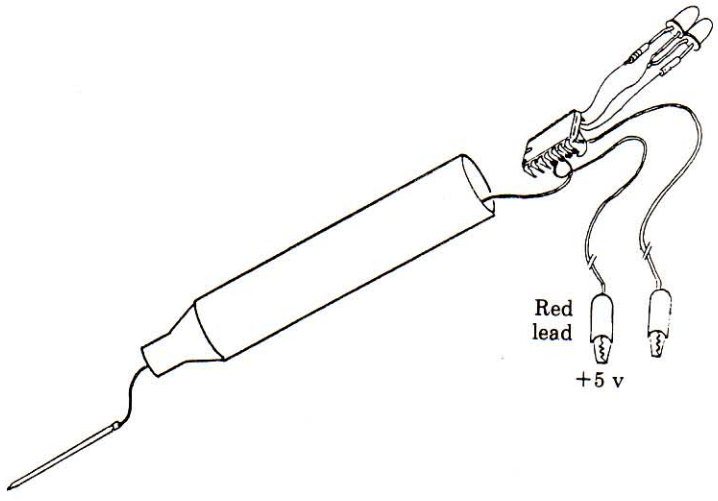
Let's first use the breadboard to construct, test, and analyze the operation of our probe. We'll build the circuit in a more permanent manner later. Figure 2.9 shows how the components are laid out on the board. The connections are shown going from each pin insertion point. Notice how the layout and connections (figure 2.9) follow the diagram (figure 2.19). Not all schematics and breadboard projects will be as similar, however. Now, looking at the schematic (figure 2.19) and the assembly drawing (figure 2.20), you can see two wires that have alligator clips attached to them. These are power leads. The black clip connects to the negative power supply of the device to be tested, and the red connects to the positive power supply lead of that same device.

Actually, the 7473 is a *dual* flip-flop; it contains two little flip-flop devices. We'll be using only one for our probe. In figure 2.19 you can see how each LED is connected to an output pin of the chip.

Wire each pin as shown in figures 2.9 and 2.19. The probe's point can be just a 12-inch strand of wire for now. Remember to check off your connections on the schematic as you go. You might not think this step is necessary with so few pins, but it's a good habit to establish.

Don't connect the breadboard to your power supply yet. We want to check a few things before we do our smoke test. First, are all connections made? Second, are the plus and minus leads going to the correct power rails? If both are correct, take a 6-volt battery, like those in square handheld flashlights, and connect the power rails of your breadboard to the

Figure 2.20
The assembly of the digital logic probe. The parts are inserted into the marker housing and held in place with silicon compound.



battery with alligator clips. Make certain that the clips are correctly connected to the battery's terminals, red to plus and black to minus.

You should now have a working circuit. One of the LEDs should have come on; the other should be off. If the situation is not thus, then disconnect the power, and recheck your wiring. Now check its operation. It would be good to place the probe on a point that is rapidly changing or pulsing. Since you don't yet have such a point handy, you can improvise. Touch the probe lead to the minus connection of the battery's terminal. The LEDs should flip; that is, one LED should go off, and the other should come on. Any time the chip undergoes a negative transition, it flips. Now touch the positive terminal. It might flip, but, more than likely, it will stay the same. Do this back and forth several times to verify that the flip-flop is working. If the probe were connected to an address line on the computer that was rapidly changing states (switching from positive to negative), both LEDs would glow. If only one glowed when a point was touched, then you would have no pulse. If one LED glowed, you would have either no voltage or stationary (either positive or negative) voltage. (You can use your VOM to check which it is.)

Constructing the Permanent Probe

Now that you have the circuit properly operating, you can make it a little more permanent. Solder short lengths of wire to each pin as shown in figure 2.20. Use the leads from two alligator clips for the power connectors. You can make the probe housing and point with a small finishing nail and a large plastic marking pen. Glue the finishing nail neatly into the end of the pen after soldering a 6-inch wire to the nail's head (figure 2.20). The LEDs simply stick out the back of the probe housing. Glue the 7473 into the marker's case with either silicon rubber sealant or white glue. Let it dry, and take a break.

3

Simple Interfacing Hardware

In the preceding chapters, you have studied all of the components and techniques that are required to interface your computer to the real world. In this section we will discuss the actual construction of several different projects that will make the Timex/Sinclair 2068 computer a real-world control unit. These projects will take the concepts and components and turn them into actual working circuits.

Once the construction of the circuits has been discussed, the programming needed to run them is presented. The routines will be short and simple and will only show how the circuits depend on the computer for proper operation. The circuits are complete enough, however, to respond to more advanced 2068 programming, and you are encouraged to add to my software routines whenever possible. Remember, in computer control, the circuit is minor—it is the software that actually controls the mechanisms. The more intricate the software, the more intricate will be the operation. Remember, too, that, with the proper software, these circuits are capable of performing most real-world control tasks no matter how complicated those tasks might be.

The circuits are presented in order of complexity, as a general rule. There are a few simple (and not so simple) circuits that are sequenced to show that the major circuit will be incomplete without others.

Electronic Hobby Skills

If you have not done much hardware work with digital circuits, you may be worried that you will blow up your computer once you start hardware

work in earnest. This is a remote possibility. But please understand that I work on most of my circuits late at night (I mean, really late!). Sometimes I'm so tired that I can't think, see, or even find components right in front of me. In this condition, I make mistakes. I hook things up backward. I even burn out a few interfacing chips. I did occasionally make wiring errors when I was constructing some of the projects I describe. It's not unusual to goof up, but the goofs rarely harm the computer. If you take your time, do the projects in the order in which they are presented, buffer the lines (see Buffering the Signals, later in this chapter), and always try to understand clearly what you are doing before you do it, your chances of hurting your 2068 are very slim indeed.

I once purchased several used-kit rejects of the older ZX81. These were some that had developed problems and had been returned to Timex. Most of the kits' problems were caused by the hobbyists who had attempted to put the kits together without either the proper instructions or the proper experience in kit building. Just as it takes practice to do anything else well, it takes practice to solder well. Most of these rejects looked as though they had been assembled with a blowtorch. One kit had so much solder on the leads that almost every trace was shorted out. I don't recommend soldering as a first step in project construction. I do believe that a kit can be properly assembled (even by a beginner) if the instructions are clear and the hobbyist takes his or her time.

If you don't have a lot of experience in kit building, then you might want to find someone who does. I've found that most electronic hobbyists are more than eager to help. You may feel silly asking questions, but the reward is well worth it. Once, while working on an old shortwave set, a friend whom I had asked for help reached into the set and cut several resistor leads. I almost fainted: as far as I could tell, he was tearing the thing up! He later explained that the easiest way to check resistors is to disconnect them from the board. The best way to do this on a printed circuit board is to clip their leads, check them with an ohmmeter or VOM, and solder the leads back together. It was fascinating to watch an experienced person work. There have been many others who have helped more than they will ever know by eagerly answering my many naive questions.

This book assumes, however, that you will try most if not all of these projects on your own. The directions are written for beginners. I feel that the directions will make up for your inexperience, and the explanations will ensure that you understand what you are doing before you actually do it. If you don't feel at ease with your understanding of a circuit, go back and read over the explanations until you do. Above all, don't just start connecting wires without knowing what you are connecting. This type of construction can cause problems. And always doublecheck each connection before you leave it. Once a hobbyist starts doublechecking his or her

work, the hobbyist is no longer a true beginner but has advanced to the status of novice.

The first projects, the power supply, the backplane connector, and the buffers, are not actually interfacing circuits. They are necessary projects for those that follow, however. If they are constructed correctly, there is hardly any danger to your 2068; the buffers, especially, will protect the computer from the rest of the circuits. The buffers are simple, but they are very important. They let you make a few errors. They are your failsafe devices. When other circuits are added, they will also have buffers in them that help keep you from harming the computer.

You may be wondering, of course, why you should go to all the trouble of building an interfacing device when it can be purchased already constructed. The answer is not that building it yourself will save you dollars, although it does cost less to build your own. The answer is not even that there is a special joy—a feeling of accomplishment—in doing it yourself, although there is. The real reason, I think, is that there is an even greater joy in knowing that you have acquired an understanding that very few people possess. It is the understanding that makes it all worthwhile. One of the greatest feelings I have ever had is (after several failures) seeing a project work and finally understanding why it works and why it didn't work the first time! This knowledge can't be purchased. Gaining it is like collecting rare coins. And the more of this knowledge you collect, the easier it gets to augment your collection.

The purpose of this book, again, is to help you understand the fascinating world of computer electronics and to help you find that there is much more to computers than number crunching.

The Power Supply

The small, do-it-yourself pulse detector described in chapter 2 got its power directly from the circuit under test. That is, it was powered by the same power supply that was providing current and voltage to the rest of the circuit. This is usually the best way to provide power. Most electronic circuits have one supply that provides both voltage and current to all parts of the circuit at the same time. If circuits are added, there might be a need to add a separate supply for the extra circuits only. Let's discuss voltage and current first.

Most ICs (those in your 2068 included) require 5 volts. You can think of the *voltage* as the *pressure* required to push the electrons through the circuits. Actually, voltage is the force exerted on electrons by a magnetic field. A force of 1 volt pushes 1 ampere of current through a resistance of 1 ohm (1 ohm is practically no resistance at all). Your 2068 has an overall

resistance that is low. So, if you had more resistance, you would need more voltage, or you would have to get by with less current.

Current is the amount of electrons passing through a circuit. Or, to put is another way, current is the *flow* of electrons. Electrons (small, charged subunits of atoms) are generated by a power supply and pushed through the circuit by the voltage. Therefore, the more volts you have, the more current you have. The more resistance you have, the less current.

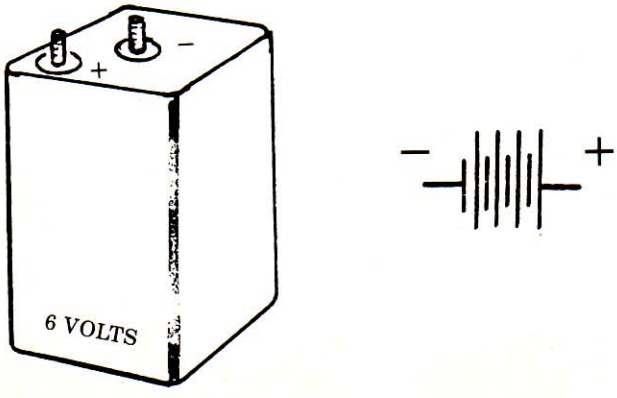
Now, the power supply of the 2068 is 5 volts, and this cannot be changed. If more than the console is operated from the supply, you will need more flow from the supply. If you decrease the resistance (by adding more circuits), more current will try to flow, and the pressure (voltage) will drop. If the voltage drops below a predetermined level (5 volts), the circuits that were designed to operate at 5 volts will go dead. You will need another supply to supplement the power supply of the Timex/Sinclair 2068.

A Battery Supply

Most small, two-IC circuits can operate just fine if the power is taken from the backplane of the console. The manufacturers did build some margin into the 2068. However, as more ICs are added, the regulator starts to overheat as it tries to supply the needed current. This is not good for the regulator or the other ICs that are starving for current (they overheat, too). But a small battery for the additional circuit can solve most of the problem.

Figure 3.1 shows (in addition to a sketch of a battery) the schematic symbol for a battery power supply. Notice that one lead is marked + (*positive,* or *plus*) and that the other is marked − (*negative,* or *minus*). In our

Figure 3.1
A 6-volt dry cell that can be used as a power supply. The schematic symbol shows that it contains four separate cells.



circuits, the negative lead will be called *ground* (it is also sometimes called *common*). *Ground* is an important power-supply concept.

In order for a voltage to be measured, there must be some reference point. That reference is *ground,* or 0 volts. Most electronic control signals swing from 0 volts (ground) to 5 volts (positive). Ground, then, is the reference point for the signals. *Five volts* means five units above the potential of ground. Without ground, the components would have no way to determine what voltage was being fed to them. You'll see why this is important in a minute.

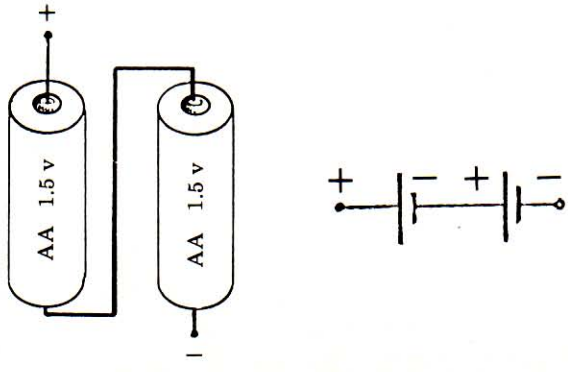
Since our circuits can operate fine with a voltage of between 4.5 and 6.5, a small, 6-volt battery can be used to power most smaller circuits. It has the necessary voltage, and, if it is a 6-volt lantern battery, it can supply a very large current.

Figure 3.2 depicts two small batteries connected in series. Each battery supplies 1.5 volts. If you connect the plus (+) of one to the minus (−) of the other, the voltages of the batteries are added to each other. The separate transformer supply, which may be constructed later, will need the minus connection tied in common with the ground of the 2068 for a common reference point. Batteries have a tendency to run down and can make you think your circuit is wired wrong. It is for this reason (and to save money) that you probably will want to construct your own supply.

A Separate Power Supply to Build

The separate supply detailed here is a small supply that can deliver only about ½ amp (500 milliamps) at 5 volts. The components are few, and they

Figure 3.2
Two small batteries connected in series. This arrangement gives a total of 3 volts. Note the symbol used to depict this type of power connection.



must be hardwired (soldered) and put into a small enclosure for protection.

All components (as with most of the projects) can be obtained from Radio Shack or local supply outlets. Power supplies can be simple or extremely complicated. We'll stick with the simple. Only seven parts are needed to construct this project, but, in addition, a small perf board, an on/off switch, and a metal enclosure are highly recommended.

Circuit Operation

Let's look briefly at how a power supply works. A power source from an electronic circuit is a simple and economical means of powering digital ICs. Since most ICs require very little current, a supply for them does not have to be big and complicated. The output (the voltage and current) does have to be appropriate. Digital circuits are very intolerant of a bad supply. Therefore, if at any time a circuit fails to operate, always check the power supply to that circuit first. Without proper power, nothing works.

The Transformer The *transformer* provides the necessary force to get the electrons moving. The size of the transformer usually gives you some idea of its *capacity,* or how much power it will produce in amps. Very small power transformers, those measuring less than 2 inches square, deliver less than 1 amp. Larger ones (up to around 6 inches square) can provide as much as 4 amps. Since most of our circuits are small, we will use a ¼-amp (250-milliamp) miniature power transformer. We will use a 6.5-volt transformer. This voltage will later be reduced to 5 volts for use by the circuits.

Since the wall plug power and the output of the transformer are alternating (changing back and forth), the electrons are of little use to us: they move first in one direction and then in the other. We need electrons that move in one direction only. Our transformer will provide an alternating current that must be changed to something approaching continuous current, or direct current (DC), by our next supply component, the bridge rectifier.

The Bridge Rectifier We said in our discussion of diodes (chapter 2) that a diode lets electrons move in one direction but makes them stop if they try to go in the other. We'll need to add a special diode circuit to change AC to DC for us. This diode array is called a *bridge rectifier.* It keeps the electrons moving from plus to minus. The flow is very choppy, however, because half the time the transformer is trying to move the flow in the other direction. We'll need a device that will smooth this current out. The device that does this is the filter capacitor. Figure 3.3 shows how our voltage looks at different points.

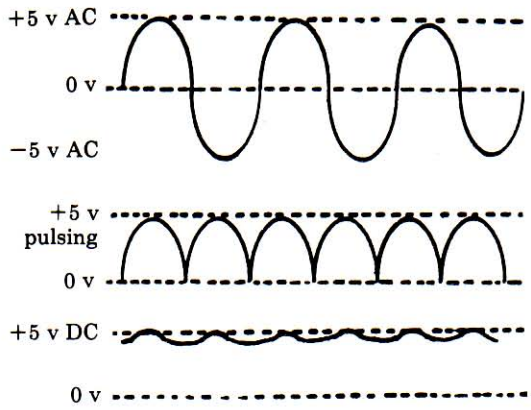


Figure 3.3
Power-supply waveforms. A rectifier changes AC to pulsing current. A large capacitor fills in the gaps. The ripple that is still present must be slight.

The Filter Capacitor A *capacitor* charges up and then slowly releases its charge when voltage is removed. A large-value (1,000- to 2,000-millifarad [mf]) capacitor placed across the output leads of the rectifier can release the capacitor's accumulated electrons during the time that the voltage is falling. The waveforms shown by figure 3.3 will be smoother. (Figure 3.3 shows the changes in voltage as the current is rectified, then *filtered,* or smoothed.) Next to the capacitor we will place a resistor. This resistor is called the *bleeder resistor* since it bleeds the charge off of the capacitor once the supply is turned off. Without this component, the capacitor could hold the charge for many hours.

The filtered current is very nearly direct current, or current without ripple, such as that supplied by a battery. Notice that a little ripple still exists, however. Too much ripple causes digital circuits to fail, but our ripple should be within acceptable limits. The ICs of our circuits operate on direct current only, so we are very near having the required power for our devices. We need to add only one other part to make our power supply functional.

The Regulator Electronic circuits that are operating have a tendency to make a power supply's voltage fluctuate. As more circuits are added, the voltage tends to drop. As circuits warm up, the voltage drops even more as the resistance decreases. Some means of regulation is required to keep the level steady as the circuits draw more or less current. A means of regulation is required for the additional reason that a diode can even break down

completely (and allow raw alternating current to be fed directly to your circuits, unless there is a regulator). The voltage regulator is a must for any transformer power supply.

The 5-volt 7805 voltage regulator will be used in our small supply. This device actually operates like a power transistor: it monitors the voltage and increases or decreases the current as demand fluctuates. It is a device that is specially designed to deliver a constant 5 volts at all times. Since it will heat up when it is regulating heavily, it should be mounted on a large piece of metal or to the side of the metal case of the power-supply enclosure. This arrangement allows the transistor's heat to dissipate.

Multiple Voltages

Circuit diagrams that require three or even four separate outputs of voltage are uncommon. A few years ago, many ICs required three different voltages to operate, but no longer. Now, with advancing technology, most of the new ICs (even LSI chips) have gone to one voltage level (+5 volts). It is for this reason that your supply can stay simple and inexpensive.

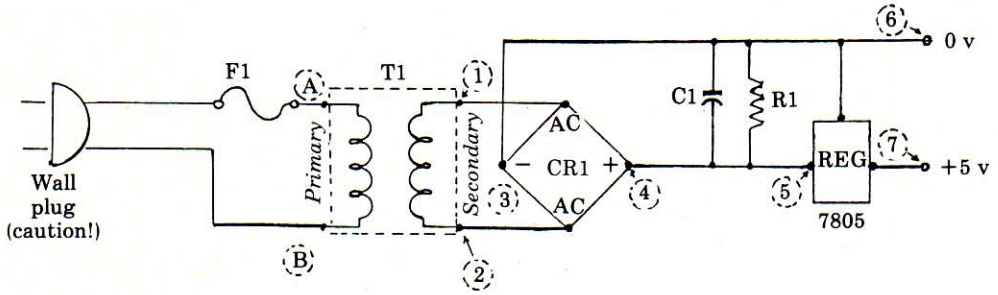
Building the Power Supply

Materials Figure 3.4 shows the complete circuit you will be building. The power supply is the only circuit that you will need to hardwire. Since it carries much greater currents than most of the other circuits you will build, the wire for the leads will have to be heavy. Regular hookup wire for this circuit can be multiple- or single-strand. It must be medium-heavy and insulated. Bare wire could carry the current, but it shorts out easily and can ruin you and your computer. *Remember, until you run the leads through the tranformer, you will be working with 110 volts. This high voltage is dangerous!*

Most of the circuit can be wired on a piece of heavy-duty perf board. The wiring is then run under the board and appropriately soldered. So that the solder connections cannot touch the metal enclosure, I use a terminal strip to hold all of the components suspended. Figure 3.5 shows the power supply components mounted on the strip and soldered into place.

Soldering the Components The most difficult part of hardwiring a circuit is soldering. It is almost an art. With a little practice and the right tools, however, you can solder as easily as you can breadboard.

The right soldering iron is a must (figure 3.6). You must use a small pencil iron (40-watt), and you must have the right solder. Use printed circuit solder that is very fine (1/16 inch) and has a ratio of tin to lead of about 60/40. It must be rosin-core. Never use acid-core solder on copper wire. It makes the wire grow a coat of green crud.



F1 270-1224 1.5-amp fuse
T1 273-1384 6.5-volt 300-ma miniature transformer
CR1 276-1151 50-PIV 1.4-amp bridge rectifier
C1 2,000-mf electrolytic capacitor
R1 1-watt 1,000-ohm resistor
REG 276-1770 voltage regulator

Miscellaneous
276-1151 small metal case (see text)
Single-strand 18-gauge insulated wire
Terminal strips

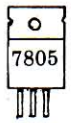


Figure 3.4
A complete circuit diagram of the separate power supply. The parts shown are readily available. *Caution: be extremely careful when handing 110 volts AC.*

Another important item is soldering paste. I've found that, no matter how clean your connection is, there is always some oxidation present that will not let the solder stick. Each connection should be coated lightly with soldering paste, a greaselike compound that removes all oxidation and makes soldering a breeze.

Connect your two points of wire tightly with needle-nose pliers, and coat the connection with soldering paste. Make certain your pencil iron is hot, and touch the connection with the iron. Let the connection heat (not too much!), and touch the connection and the pencil with the solder at the same time. (It sounds as though you need four hands, I know.) The solder should flow smoothly into all parts of the joint. Make sure the parts do not move while the solder is cooling. If the solder is not shiny and bright when cool, you will want to reheat the joint. A dull grey joint indicates a bad solder connection.

Once the circuit is completely wired, use your VOM (figure 3.7) to test the output. Make sure the VOM is set on the 5-volt DC scale. The output of the power supply with no circuit attached should be exactly 5 volts. If no output is showing or the voltage is low on the meter, we will have to look for the trouble, or *troubleshoot.*

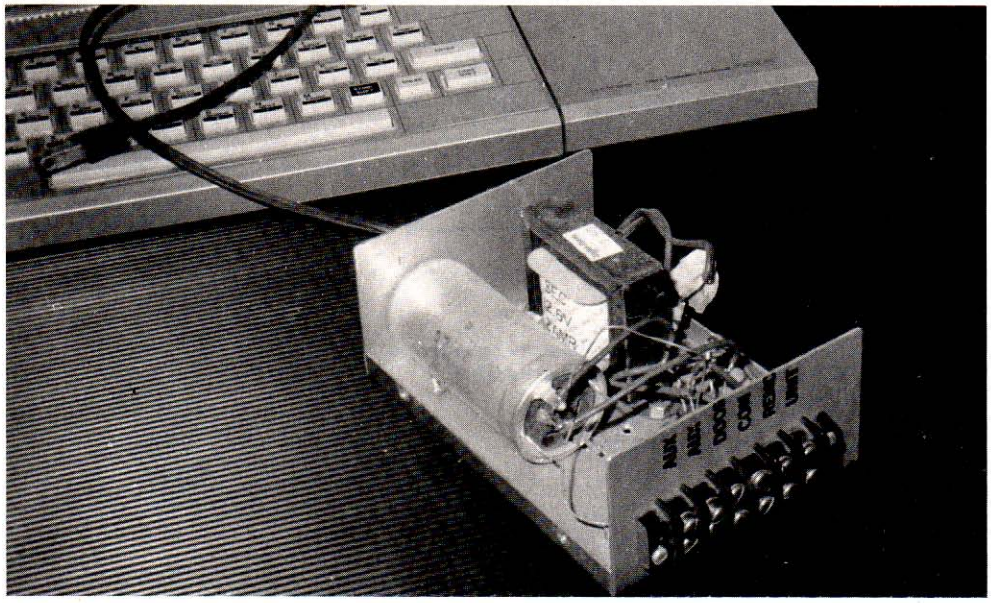
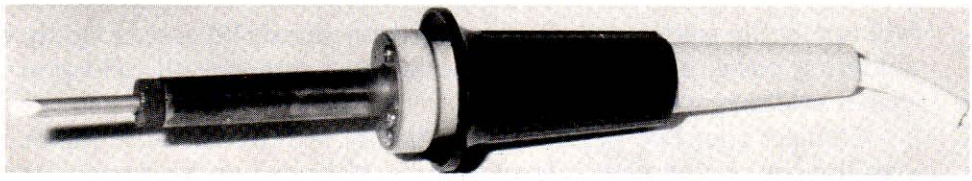


Figure 3.5
The interior of the completed power supply. The large can capacitor can be as small as 5,000 mf.

Figure 3.6
A good low-wattage soldering pencil. This tool is a must for close work.



Troubleshooting the Power Supply

Notice in figure 3.4 that various points on the schematic are marked with letters and numbers in dashed circles. The points indicated are called *test points.* It is at these points that we will connect the VOM to see what voltage we are getting. Since we are assuming that we are getting no output at points 7 and 6, we will move the test farther back down the line until we do.

With the black lead of the VOM on the point marked 0 volts in figure 3.4, move the red lead to point 5. If the voltage increases, the regulator is

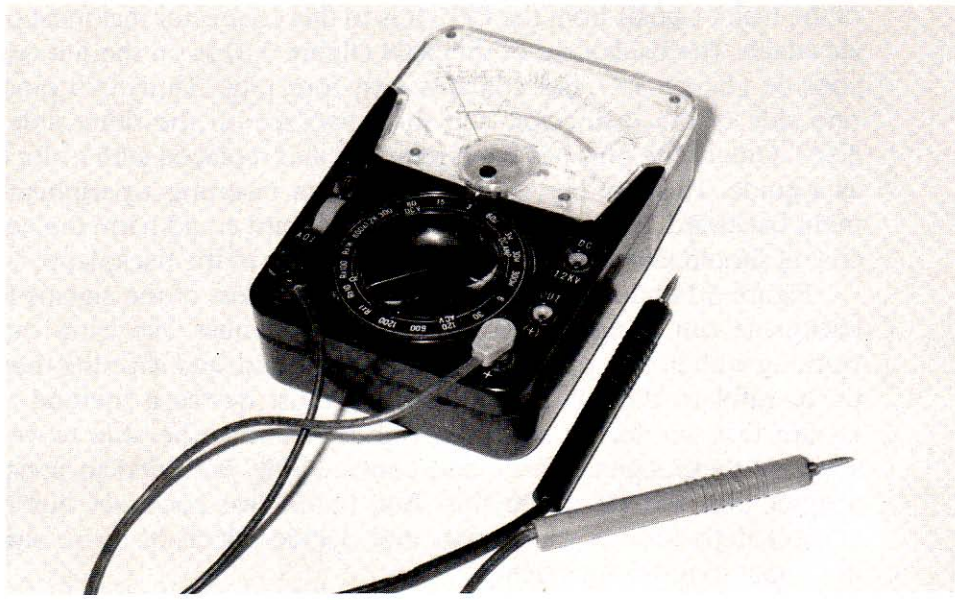


Figure 3.7
A volt ohm meter.

wired wrong or shorted out. Now change the VOM scale to 6 volts AC, and place the leads on points 1 and 2. The voltage should be around 6 volts. If not, the diode array is not working. You should check the wiring. If no voltage is showing at points 1 and 2, the transformer or fuse is not producing, so check the wiring to the transformer. Change the VOM to the 110-volt AC scale, and place the leads carefully on points A and B. It should be around 110 volts. If not, the fuse is blown, or the power cord is not plugged in. Probably it's the latter.

The Backplane Connector

The Backplane

Most home computers have what is commonly called a *rear edge connector.* Since more and more specialized connectors, many of them multi-fingered, are being added to the back of the computer console, this terminology is no longer the best. We will call our peripheral connector the *backplane.*

The *backplane* is the point on the computer that has most (if not all) of the major signals from the CPU. It is to this connector that most peripherals attach. The backplane of the 2068 (figure 3.8) is on the left rear of the console (figure 3.9) and contains sixty-four pins. Thirty-two pins are on one side of the connector, and thirty-two are on the other side (figure 3.10). One of the pins has been removed and replaced with a slot that acts as a guide. This slot prevents the user from inserting a peripheral attachment backward. Inserting a connector backward could harm the computer, so you should take extra care when connecting to the backplane.

Figure 3.11 shows the adapter connector. Most of the signals from the backplane run straight from the CPU, so we must take extra care when working with it. The pins are very closely spaced, and shorting them could be harmful to the CPU. Therefore, we must devise a method that will ensure that we do not short these pins. We must be able to work with some of these signals easily and conveniently. We need to construct an adapter that will let us do this. But, before we construct our interface adapter, let's look at the signals and decide which of those signals are important to us for interfacing.

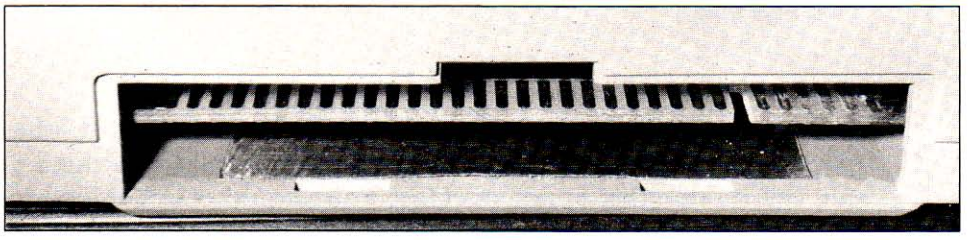
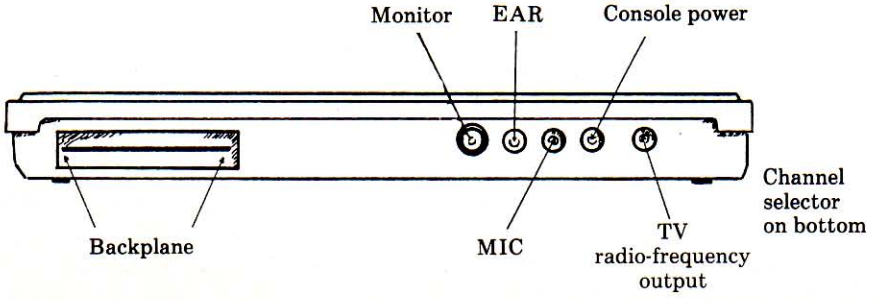


Figure 3.8
The edge connector, or backplane, of the TS 2068.

Figure 3.9
A diagram of the rear of the console showing component connections.



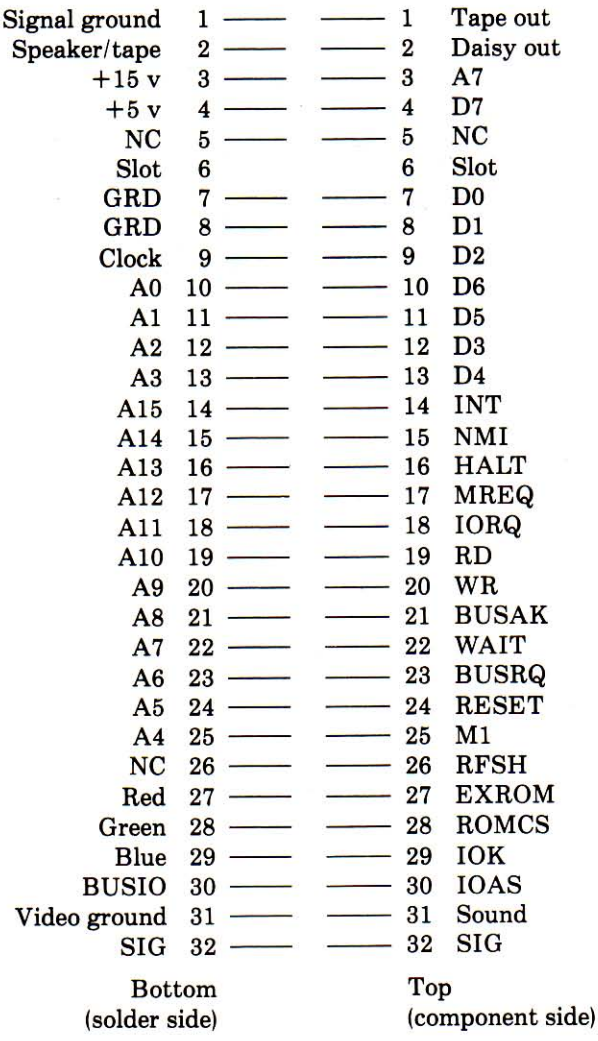


Figure 3.10
A diagram of all signals on the backplane. Only the I/O signals will be important to our projects.

Most of the signals on the backplane are special signals to be used in attaching special Timex peripherals. We will be concerned with only those signals that are directly related to the CPU (figure 3.12). These include most of the address lines (A0–A15) and all of the data lines (D0–D7). Since we will be addressing our devices using the IN and OUT commands and instructions, we also will need the IORQ, RD, and WR signals. Also, we want to take advantage of the power supply lines (especially ground); these are marked +5 v and GRD. The rest will be left for more advanced work with the 2068.

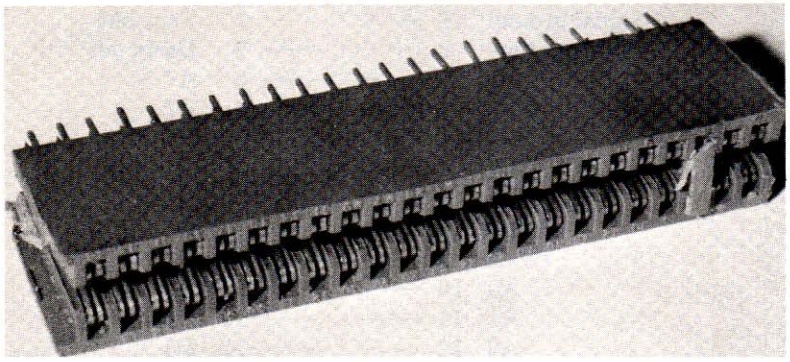


Figure 3.11
The adapter connector. Notice the small insert that fits the slot in the backplane edge connector.

Figure 3.12
The only area of the backplane outlet that the adapter needs to fit. Signals outside this area will not be used.

+5 v	4		4	D7
NC	5		5	NC
Slot	6		6	Slot
GRD	7		7	D0
GRD	8		8	D1
Clock	9		9	D2
A0	10		10	D6
A1	11		11	D5
A2	12		12	D3
A3	13		13	D4
A15	14		14	INT
A14	15		15	NMI
A13	16		16	HALT
A12	17		17	MREQ
A11	18		18	IORQ
A10	19		19	RD
A9	20		20	WR
A8	21		21	BUSAK
A7	22		22	WAIT
A6	23		23	BUSRQ
A5	24		24	RESET
A4	25		25	M1

All the signals of interest to us are in the middle of the backplane. This is because the manufacturers wanted to encourage the use of their old 2040 printer with the new console and so left the arrangement the same as

with the earlier TS 1000 model. Therefore, we do not need an adapter that covers all of the connector but only twenty-two pins (forty-four pins if you count both sides). This should save some expense when you buy a connector for your interface projects.

Building the Backplane Adapter

The optimal way to connect a peripheral device (or circuit) to the backplane would be to build the circuit on a card and incorporate a sixty-four-pin connector in the circuit. That way, the circuit could be attached directly to the edge connector when we wanted to operate the circuit. This is fine for a completely debugged circuit (such as a parallel printer interface) that operates for a special purpose. However, we will be experimenting with wire wrapping, breadboarding, and other nonpermanent work, and we will need an adapter that is not so permanent.

First we need to obtain a 0.1-inch twenty-six- to fifty-pin double-sided connector and cut off each end in order for it to slide onto the middle of the backplane. This connector can be obtained from most electronic supply outlets and should be one with pin spacing of 0.1 inch (with a pin centered every tenth of an inch). Any other spacing will not work.

Two pins on the adapter (one on the top and one on the bottom) must be removed from the pin 3 position. This leaves a slot for our guide. The guide can be made of any small piece of plastic glued into this slot. The guide will prevent us from inserting the adapter upside down. The pins that have the signals shown in figure 3.12 should be clearly marked with a strip of white paper as in figure 3.13 so that we can easily locate the points for soldering and later testing. Flexible wire should then be soldered to each of the pins mentioned above. We will not wire up the rest of the pins at this time. This flexible wire should be multicolored for easier lead locating, and the easiest way to get multicolored wire for this purpose is to use what is known as *ribbon cable.* This multicolored cable is made up of twenty to fifty multistrand leads that will connect our devices to the computer. The length of these leads should be around 10 inches since longer lengths will degrade the signals and cause *glitches. Glitches* are misreads of signals caused by interference and line loading.

When the leads are soldered to the pins of the adapter, they should be insulated with ⅛-inch heat-shrink tubing. This keeps the connections from shorting together and causing the circuit to fail. About ¼ inch of the other end of these leads should be stripped of insulation, and the strands should be twisted to form a solid bare end. This bare end should be coated lightly with soldering paste, and a small amount of solder should be applied. This makes the end of each lead rigid and will allow us to insert it into a breadboard socket easily. Figure 3.14 is a photo that shows a ribbon cable thus prepared.

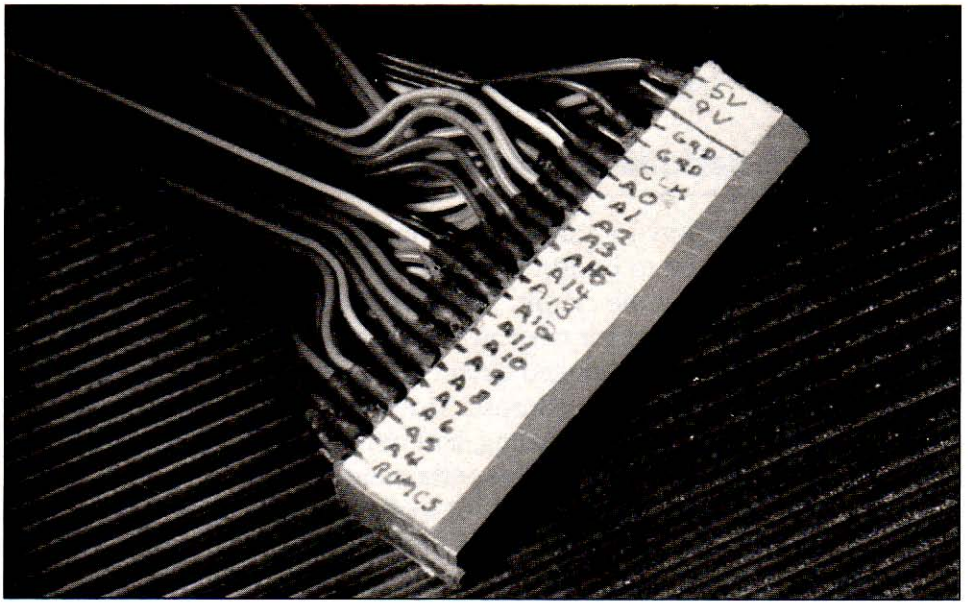
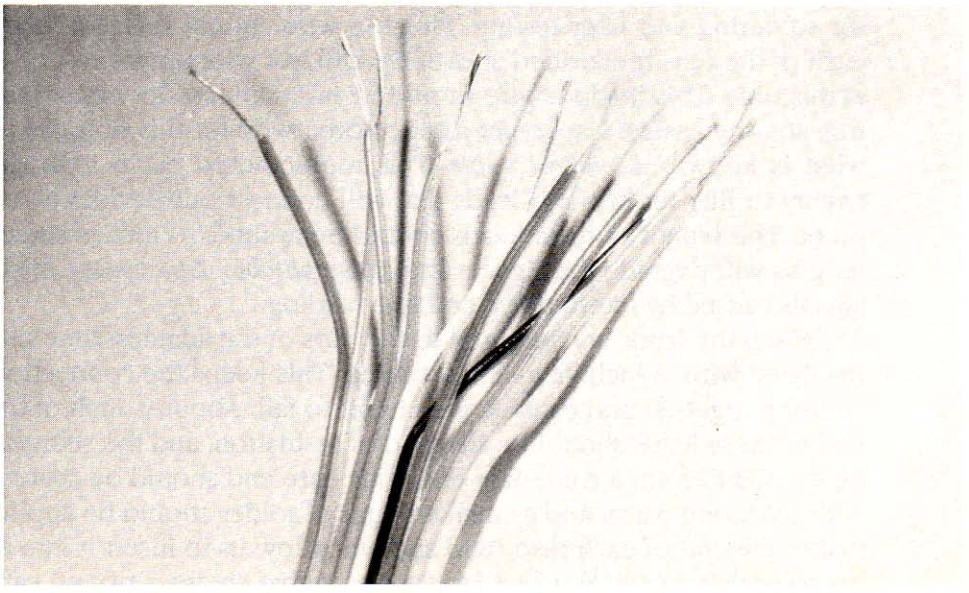


Figure 3.13
The completed backplane adapter.

Figure 3.14
The bare leads of the backplane adapter cable. These insert into the breadboard.



Once the wiring of the adapter is complete, it is ready for terminating into a special breadboard section that will allow us to breadboard into the signals. This small breadboard is placed to the rear of the 2068, and the ends of the multicolored leads are inserted into this small breadboard. Each of the signals should be labeled again on the breadboard with strips of white paper so that we do not have to trace the leads to the backplane each time we make a connection. Once this is done, the backplane adapter is complete.

Buffering the Signals

Buffering the Address Bus

Most of the signals on the backplane are stretched to the limit. That is, they are connected to several other components for the proper operation of the system. The address lines are especially loaded. This is no problem as long as no other components are added. If too many components are connected to a lead, the signal starts to fade. If this happens, the system cannot operate the other components reliably.

The 4050 CMOS Hex Noninverting Buffer

Most Z80 signals have a fan-out of five. That means that five components can be connected to a line without problems. There is very little leeway in fan-out for most of the signals of the 2068. Therefore, we want to buffer the lines before we attempt to use them. Buffering is simply amplifying the signal through a buffer chip so that it can drive several more chips without problems. If you remember, one of the components we have discussed is the hex noninverting buffer (4050 CMOS) (Appendix D). This six-element chip can take six lines and add drive to the signal, improving the signal's power so that it can drive five to seven more chips easily. This buffer will also protect a line, especially any of the lines going to the CPU (shorts in these lines could harm the CPU chip and make the whole system nonfunctional—break it, in other words).

If, for example, you hook up 5 volts to the address line, A3, accidentally, and that line already has a low on it, the address line could pull enough current to injure that connection inside the CPU. But, if the line is buffered, the CMOS chip will absorb the damage, if any, and the CPU will not be harmed. So it is a really good idea to put buffers on all the address lines that we will use. Figure 3.15 shows a diagram of such an arrangement.

On your larger breadboard, insert two 4050 CMOSs for buffering the address lines that you will need. These should never have to be changed,

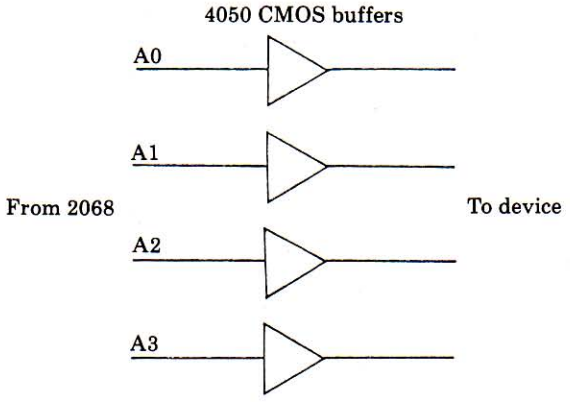


Figure 3.15
CMOS buffers protecting address lines and giving them more drive.

so place them to one end of the board, and make the proper connections to the chips from the power rails. Then, looking at the diagram of the 4050 in Appendix D and at figure 3.15, insert leads from each of the address lines shown. Since address lines are active in only one direction, the buffers should be pointed away from the computer. The computer always sends information to the device on the address lines. It never reads these lines for information. Before appyling power to these buffers, make doubly sure that all of your connections are correct. These buffers are your protection chips. Remember that there is very little protection if they are incorrectly wired. Once the connections have been completed, you are ready to test them with your logic probe or pulse detector.

Testing the Address Line Buffers

Turn on the computer, and the copyright mesage should appear within a second. If it doesn't, turn the console off, and doublecheck all of your connections. If one of the buffers is backward, it will cause the system to misread that buffer's line and operate incorrectly. If the copyright message appears, tell the computer to print something. If it does, everything is fine. Get out your logic tester, and connect it as detailed earlier.

You should have an indication that the logic probe is working by the glow of one of the LEDs. Touch the probe to one of the buffer's outputs (say, A6) and both LEDs should come on dimly. This means that they are toggling as they should.

The system is constantly changing the level on all of the address lines as it reads information from the ROMs and stores other information in RAM. If the probe indicates that the address line is not changing, then carefully place the probe on the input side of the same buffer. It should

definitely work there if the system is working. If it does not, the probe is not toggling. Recheck it by placing it alternately on the plus and minus rails. The probe's LEDs should toggle back and forth.

Buffering the Data Lines

Most large chips contain buffers already, so, many times, a separate buffer for the data lines is not necessary. However, there are times when you need to buffer these lines. You can do it in several ways. If you are going to route information along the data line in only one direction (away from the CPU), then you could use 4050s again. But, most of the time, information will need to be sent back and forth across the data lines, and 4050s will not work. But there are two special data buffers that will. These are the 74LS125 and the 4016. Even though the two carry out similar functions, they operate quite differently.

The 74LS125 Quad Tri-State Buffer/Driver

Appendix D shows a schematic diagram of the 125. Notice that it is very similar to the 4050 CMOS. The 125 has a control line that takes the data off the bus, or *Tri-States* it. It would take four 125s to fully control the data bus in both directions.

The 4016 CMOS Quad Bilateral Switch

Appendix D also shows the 4016 CMOS. It has basically the same bidirectional ability as the 74LS125 and also has a control line to open and close the buffer. It is basically a double throw switch that allows data to pass in either direction. If a low is on the 4016's control pin, the data line is disconnected from the bus. If the pin is high, the path is open to data information flowing through it in either direction. Though not actually a true buffer, it serves the same purpose.

We will not connect the data buffer at this time since it will change depending on our need. Remember, however, that the data lines are loaded and that some form of buffering is necessary for proper operation.

Decoding

The Device-Select Pulse

Probably the concept of computer interfacing most important for you to understand is that the computer can do only one thing at a time. Most people believe that the computer can do thousands of things at once. The

truth is, the computer can do only one thing at a time at a speed of thousands of times a second. This speed makes the computer appear to be doing thousands of things simultaneously. Likewise, when the computer talks to (or *addresses*) a peripheral device, it addresses only one device at a time. The other peripherals must stay off the line when the computer is talking to one peripheral. Otherwise, chaos results. The computer switches control to various devices at different times and keeps them in an orderly line, each waiting to speak in turn to the CPU. The computer maintains this order by using *device-select pulses.*

The CPU has several methods of generating a device-select pulse, but it makes sure that it sends out only one at a time. If it did not, two or more devices would try to send data to the CPU at once, and the whole system would crash. The select pulse is sent from the CPU through a decoding circuit, and the device responds with data when its number is called. Let's see how this works.

The 74LS154 Four-to-Sixteen-Line Decoder-Multiplexer

In order to understand fully how a device-select pulse is generated, you almost have to build a decoder and generate a select pulse. Until you do, the device-select concept will be very difficult to grasp. We will use the 74LS154 decoder (Appendix D) to decode a select pulse, and then we will experiment with some of the addresses. We will not dismantle this decoder but instead go on to build another device to attach to it to demonstrate the operation of both the device-select and also the newly built peripheral circuit.

You already have two address buffers in place on the breadboard, so simply skip one set of pin sockets, and insert a 74LS154 IC. Figure 3.16 shows a complete diagram of the decoder. Notice that you will need two more gates, a 7400 two-input NAND gate and a 7402 two-input NOR gate (Appendix D). These should all fit nicely on the large breadboard.

Using the techniques described in chapter 2, breadboard the circuit shown in figure 3.16. It will not be necessary to buffer IORQ and WR at this time. Later, we might want to buffer these lines if they are driving several chips. Remember to wire the power leads from the ICs to the power rails first and to check off your connections as you go.

How the Decoder Works

We saw in chapter 1 that a string of several NAND gates can decode a combination of addresses. We also saw that the CPU responds with an address on the lower eight address lines when an IN or OUT command is

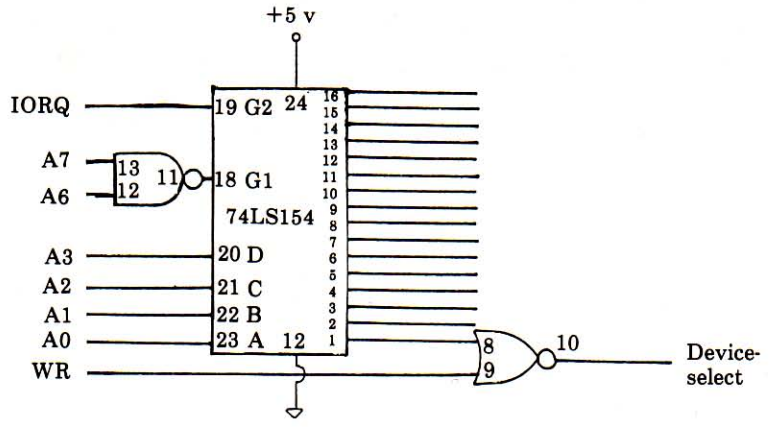


Figure 3.16
A complete schematic of the 74LS154 device-select decoder circuit. This decoder gives sixteen separate selections for use with all of the circuits shown later.

given. The decoder shown in figure 3.16 is nothing more than many NAND gates that do this very thing.

Let's again review what happens on the bus when the command OUT is executed. The CPU drives IORQ low, makes WR go low, and (depending on the number following OUT) drives the address lines high for the unique combination. The 74LS154 will decode inputs A and B if both G1 and G2 are low. If we connect IORQ to G2, we have met one of the conditions. The other control, G1, is driven low by a combination of two addresses (in this example, A6 and A7). Notice that we ran these two address lines through a NAND gate first. If we wanted G1 to be low, it would be necessary to have both of the addresses high. The address range to do this using only the eight bottom address lines would be from 192 through 255, since both A6 and A7 have to be high for these numbers. Any lower number would cause one of these lines to go low. Since we have the lowest four address lines decoded (A0 through A3) the range is even shorter—192 through 207. This represents sixteen different device-select pulses, and this should be enough for our purposes. If you do not see where we got the different ranges, go back to the discussion of address decoding and counting in binary in chapter 1.

When all of the requirements have been met, the 74LS154 will output a low at one of the output pins of the decoder. The problem is that all of the requirements are met constantly by the system as it performs its other operations. Therefore, the decoder is constantly outputting a stream of pulses on the output pins of the decoder. What do we do?

Well, for starters, let's remember that there is one other signal that was also brought low as a result of the OUT command, the write signal, WR. If we route this signal to the NOR gate along with one of the device pulses, we should get a high pulse any time that both the decoder's signal and the WR signal are low. This, of course, is when the OUT command is executed. Breadboard the circuit, and let's do some experimenting to see if it will work. If it does, we will get a pulse only when we address the specific device number with an OUT instruction and at no other time. The same device number will generate a pulse if we connect the RD line in place of the WR and use an IN instruction. Remember that IN drives the RD line low.

Checking Out the Decoder

Doublecheck all of your wiring. This is especially necessary for your power wires: they must be correct. Check them first. If they are wrong, the chip could heat up and burn out. If the wiring appears to be correct, then plug in your backplane adapter, and turn on the computer. If the copyright notice does not appear within a second, turn the computer off immediately. Do not let it operate if the system is not working. Touch each chip on the breadboard to see if it has heated up. If one has, your power leads are wrong. Make sure that each chip has both a negative and a positive supply lead and that it is in the right socket.

Take your logic probe or pulse detector, and power it from the power rails. Make sure your leads go to the correct rails. Place the tip of the probe on pin 19 of the 74LS154. You should have two dim LEDs. This means the signal is there. That's good. Place the probe on pin 11 of the 7400 NAND gate. Again, you should detect a signal. Place the probe tip on pin 18 of the 74LS154. The toggle should be the same. Do the same for pins 20 through 23 of the 74LS154. If there is a signal on pins 8 and 9 of the 7402 NOR gate, everything should operate correctly. Now let's look at the necessary software to generate our first computer-controlled device-select pulse.

Generating Device-Select Pulses with Software

If you place the probe tip on pin 10 of the 7402, you should see one LED lit. This means that the signal is low. It will remain in this state until it is activated by a software routine. The programming to check the circuit is very simple. Type (but do not enter) the following direct command on the computer keyboard, so that the screen reads:

```
OUT 192,1
```

Before you enter the command by pressing the ENTER key, let me explain what you are telling the computer to do. The OUT command instructs the computer to send the indicated data (the 1) to the device selected by the number following the OUT command. That is, OUT 192,1 tells the computer to turn device 192 on, open the data bus to device 192, and place a 1 on the eight data lines for receipt by device 192. Actually, that's what the computer will *try* to do, but our circuit is built only to generate the device turn-on signal. Let's see if it works.

Place the tip of your logic probe on pin 10 of the 7402. While watching the LEDs, press the ENTER key to execute the above command. The logic probe's LEDs should switch immediately. That means that the signal did, in fact, get through, and your circuit is a success.

Try entering the following direct command and making the same test:

```
OUT 193,1
```

You say nothing happened? That's right—the decoder is wired to generate a pulse for only device 192. But, if you change the lead on pin 8 of the 7402 from pin 1 on the 74LS154 over to pin 2, the new decoded address for this pin is 193. Likewise, pin 3's address on the 74LS154 is 194. Pin 4 has the signal for device 195. See how it works?

There is one problem in all of this, however. Try connecting the lead to pin 6 of the 74LS154 and generating the select pulse (the address on pin 6 is 197). You have a problem, right? You are getting pulses even when you are not executing the OUT command! That's because the system is using this device number for its own operations. Several of the numbers in our sixteen addresses are being used. Figure 3.17 shows the addresses that are not available. Do not use these addresses for I/O purposes.

Generating Device-Select Pulses under Program Control

All of our commands have been executed directly up to this point. Let's write a small program that addresses devices in sequence. To do this, enter the following short routine:

```
10  FOR X = 192 TO 196
20  OUT X,1
30  NEXT X
40  GOTO 10
```

This little loop generates a sequenced device-select pulse for each address from 192 to 196. We could do this for the whole range of devices from 1 to

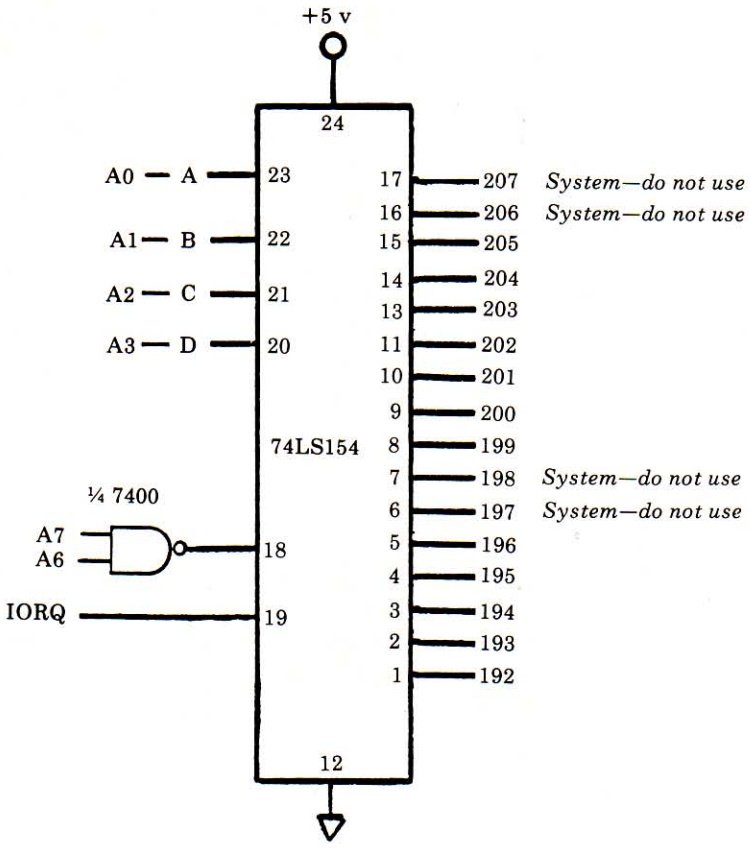


Figure 3.17
The device-select decoder. Those device numbers marked *System* are used by the computer for other purposes. If you try to use them, fouled displays and memory errors will result.

255, but some of those addresses are being used by the system, and your program might crash if a pulse to any of these addresses in use is generated at the wrong time. To check the operation of the program and circuit, place the tip of your probe on pin 10 of the 7402. With the program running, change the lead on the 74LS154 from pin 1 to pin 2, then to pin 3, and so on. You should get a pulse on each of the five pins. Remember that these pulses are not occurring at the same time, however. The computer can address only one device at a time. The pulses will be sequenced from pin 1 through pin 5 of the 74LS154.

Notice that pin 9 of the 7402 is connected to the write signal. The write line is brought low in response to the OUT command. This means that the

computer is trying to write data into an external device. But there are times when the computer needs to get data from an external device. It does this with the IN command.

Generating Device-Select Pulses with the IN Command

The IN command generates a device-select pulse each time information is needed from an external source. This source could be an input port, a disk drive, or a simple sensing device. There must be a way for the computer to select a device number and write the data contained in that device into its memory.

Figure 3.18 shows how an input port device is selected by the IN command. Notice that the only difference in the wiring from our previous circuit is that the lead to pin 9 of the 7402 is now connected to the read signal, RD, on the backplane. Therefore, if we change our lead to this signal, we can generate pulses that control input to the computer.

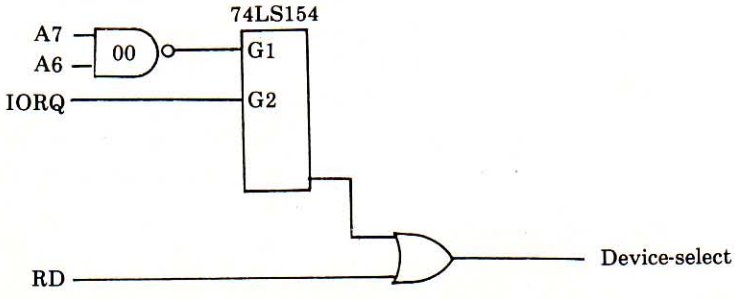
Simply remove WR from the backplane connection and replace it with RD. Connect pin 8 of the 7402 to pin 1 again on the 74LS154. We are now ready to generate input device-select pulses. With your probe again on pin 10 of the NOR gate, execute the following direct command:

```
LET A = 8
```

Now print the value in A by entering:

```
PRINT A
```

Figure 3.18
An illustration of how the select pulse is generated by the IN command. Note the connection of the RD line.



The value of A is, of course, 8. But now enter the following direct command:

```
LET A = IN 192
```

Again, we get our select pulse. There is one difference, though. This time, we told the computer to open device 192 (which we did not have), take its data from the bus, and place that value into variable A. To see if the computer did in fact do this, give the computer this instruction:

```
PRINT A
```

The value in A, you are probably surprised to see, is now 255. It might seem that it should have been 0 since we did not have a device that would put its data into A. But here's what happened. The computer told the device to place its data on the bus. The computer reads data in the inverted mode. That is, if the device's data line is low, the computer takes that value to be 1. Since there were all lows, or 0s, on the bus, the computer accepted these as all 1s, or the decimal number 255. It placed this value into A.

This is an important experiment to remember when we build our I/O ports in the next section. Incoming data is read by the computer in an inverse fashion. This is a characteristic of the TS 2068 system. Most input-port devices generate data in the active low fashion, so this works to our advantage. Most data signals would have to be inverted if the system did not have this characteristic.

Now that we have a device strobe, as the device-select decoder pulse can be called, we need a device to which the computer can talk. The simplest device is the *data latch.* We discussed capturing, or latching, data in chapter 1. We now want to construct a small data latch that will read data from the computer.

Constructing the 7475 Output Port

Our control pulses and the outgoing data pulses are much too fast to be put to good use by either us or an external device. What is needed is a way to capture that data for use by a device. If we could catch the data at the exact instant it is available on the bus, we could then either see it or use it for control purposes. Let's build an output port that will do this for us.

The 7475 Quad Latch

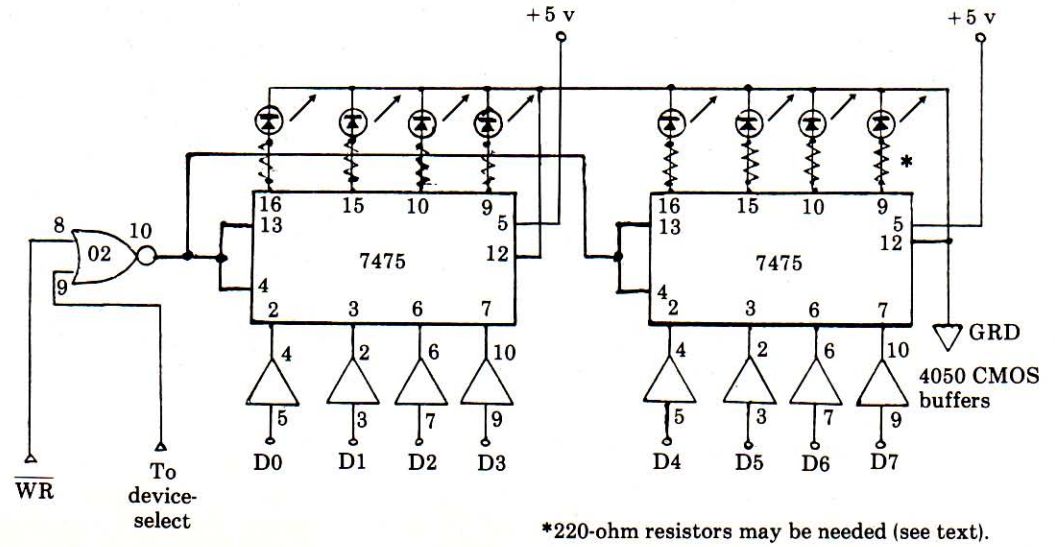
The 7475 quad latch, or 4-bit latch, is very similar to the 7473 dual JK flip-flop. It is capable of latching, or capturing, 4 bits of data. This is a

handy little device for anyone who is building output ports. Appendix D shows the pinout of this component. It has four internal gates that set themselves to the value that is contained on the data input leads at the instant the enable pins receive an active-low strobe pulse. The data is presented on the Q pins of the latch, and the exact inverse of the data is presented on the not-Q, or $\overline{Q}$, pins of the latch. This way, almost any combination of highs or lows can be obtained as needed.

Figure 3.19 shows the wiring of the data latch. This circuit could also be called a parallel output port since the entire 8-bit data bus is captured at one time.

We could use our VOM or logic probe to test each of the outputs to determine the data that is present, but this would be a cumbersome procedure. Therefore, we will connect each output of the latch to a small LED, which will indicate the value (either 1 or 0) without using the VOM. When these LEDs are placed on the breadboard, the leftmost LED becomes the least significant bit (LSB). When on, it is read as 1. The next LED from the left would be read as meaning 2. If you remember our section on binary counting, you know that the next LED would read 4 when on, and so on (through the value 128).

Figure 3.19
A diagram of the complete 7475 output port. The data buffers will require two 4050s since each IC contains only six separate buffers. Only one section of the 7402 NOR gate IC is needed.



Breadboarding the Parallel Output Port

Notice in figure 3.19 that there are eight 4050 CMOS buffers on the data lines. There is a reason that they have been included. The TTL 7475 places too much load on the data lines and causes the system to misread its own data (it usually crashes). Since there are now two more components, it may be necessary to add another small breadboard. This, of course, depends on the size of the breadboards you purchased in the beginning.

Pins 13 and 4 for both latches are wired in parallel. This means that, if one of these pins gets a signal, they all do. These are the *enable pins.* It is these pins that enable the latches and transfer the data to the output pins. The strobe (or device-select) connects to these from our 74LS154 decoder. Wire these pins initially through the NOR gate to pin 1 on the decoder to give us a device number of 192.

The power for the circuit can be obtained from either a 6-volt battery or your small power supply. Make certain that the polarity is correct. Also, make sure that the ground connection of this circuit is connected to the ground connection on the backplane of the computer. This allows the computer to keep the voltage referenced to its own ground. Notice that the power leads that connect to the 4050 CMOS buffers and the 7402 NOR gate have been omitted from figure 3.19 so as not to clutter it. Be sure that you don't miss the power connections for these components when you wire up the circuit. The cathodes (the short leads) of the LEDs connect to the ground power rail on the breadboard. Most literature shows that these LEDs need a small-value resistor on their power leads to cut down on the current and keep them from burning out. Since you may be using small, sensitive LEDs, these resistors are shown in the diagram. You may find that the brightness is too low, and the resistors may have to be reduced in value. The LEDs I have used do not require a resistor at all. You need to experiment with this component to find the best value of this part of the circuit.

Following figure 3.19, connect data lines 0 through 7 and the WR line to the outputs on the backplane. Use figure 3.12 as a guide to connecting the required addresses to the decoder circuit we built in the last section.

Now that you have the parallel output port correctly wired (double-check all leads, especially power), we can write a few short routines that test the circuit and show its proper operation.

Routines for Output Control

We now have a circuit that will capture the data we output. We also have a circuit that will activate this data latch on command (our decoder). With

the two together, we can transmit data to the output control device and hold it for almost any purpose. (Some purposes are detailed later, but, for now, let's concentrate on how this data is controlled.) Apply power to the latch and decoder circuits first, and then turn on the 2068. If the copyright notice does not appear, turn off the power to both the computer and the latch, and check for a wiring error.

When power is first applied to our data latch, the LEDs might be randomly either on or off. This is normal. Most sophisticated ports have additional circuits that set all of the outputs to 0. We will forgo that luxury and initialize (set to 0) our outputs with software.

To test and initialize our new parallel output port, enter the following command directly:

```
OUT 192,0
```

This should turn off any LEDs that randomly were on. Now all LEDs should be off. If any are on, you have a problem (probably a wiring error). If they responded correctly, enter this command:

```
OUT 192,3
```

The two leftmost LEDs should come on. The first is read as a 1 and the second is read as a 2. Added together, they represent your data in the command, 3. Try the same command with other data values. Using 255 as a data value will turn all of the LEDs on since this is the highest number that can be carried on the data bus.

Letting the Computer Do the Talking

The following short program will let the computer count in binary and will place the ascending values into the outport port. Since the TS 2068 executes these few instructions very rapidly, we will need to put in a short FOR . . . NEXT loop to slow the counting down so that you can see the sequence of numbers (in binary). Enter and run the following program:

```
10  LET A = 0
20  OUT 192,A
30  FOR N = 1 TO 100
40  LET T = 100
50  NEXT N
55  PRINT A;
60  LET A = A + 1
70  GOTO 20
```

Lines 30 through 50 form a loop that slows the counting down. Line 55 was inserted to show you the decimal value on the screen as the binary is output on the port. This little routine will count up to 255, at which time it will stop with an error message that says it cannot output a number greater than 255. You can restart it again by entering RUN.

Let's try one more routine to control the port with software. Suppose we want the LEDs to light up in turn from left to right across the eight outputs. We can write a routine that can cause them to behave in this manner. The first LED would be a 1. The second would have to be a 2, and the third would be a 4. (Each LED represents the previous value times 2.) Try this program:

```
10  LET A = 1
20  OUT 192,A
30  FOR N = 1 TO 100
40  LET T = 100
50  NEXT N
55  PRINT A;
60  LET A = A * 2
70  GOTO 20
```

If you understand the binary numbers, it should be very easy for you to get the LEDs to light up in turn from right to left and then back again. Try your hand at this problem. It's not as hard as you at first might think.

Changing the Output Port's Address

You have been addressing the port as device 192. If the lead presently in pin 1 of the 74LS154 were changed to pin 2, the new device number would be 193. This might not seem important now, but there might be a time when you require a change in device numbers (when two separate devices are connected to the same port address, for instance). Since the computer can talk to only one device at a time, it would be totally confused if two devices were opened simultaneously.

Constructing the 8212 Input Port

Not all computer information is sent from the CPU. Some data must be fed into the computer from an external source. In order to get information to the CPU, there must exist a port that places data on the bus in reply to a request by the CPU. The task of taking information in is slightly more complicated than that of sending information out.

Someone once said that computers are stupid since they do only what they are told. This person maintained that, for example, if a window were open, and you told the computer to open it, the computer would burn itself up trying to open an already open window. In a sense, the person was right. But the example failed to take into account the fact that a computer rarely operates without *feedback. Feedback* is the incoming information that lets the computer know what the conditions are before it tries to execute an instruction. Without proper feedback, a computer is indeed stupid.

Feedback can be obtained by the computer through an *input port.* This device is nothing more than an inverted output port like the one we just constructed. However, if we used a 7475 JK flip-flop (and we could), several additional circuits would be required. First we would have to turn all of the 7475's inputs around and then use three-state buffers to keep the data off the bus when it wasn't needed, and this could get complicated. Thank goodness, there is another component that can simplify things considerably. This component is the Intel 8212 8-bit I/O port.

The 8212 Component

Generally, I dislike using complicated components due to the fact that I burn out more of them than I like to admit. The 8212 is an exception. I feel that your knowing how much simpler interfacing can be with large-scale ICs is worth our taking a chance.

The 8212 8-bit I/O port is a twenty-four-pin device that lets you construct an input port with very few hassles. It has all of the circuitry that the port needs for proper operation. Best of all, it has three-state data buffers, which means that it doesn't interfere with your computer's data bus unless you tell it to. Also, it has a reset that sets the data outputs back to 0, which helps avoid a fouled-up input stream. If you look at the internal diagram of the 8212 (figure 3.20), you can see that it has our 8-bit latch and the three-state buffers. What more do we need?

Since the 8212 has many pins, it appears complicated (figure 3.21). Actually, it isn't. It has eight input pins (DI1 through DI8) and eight output pins (DO1 through DO8). It has a pin marked STB (strobe), and we know that this is our device-select pulse. So far, so good. The only confusing pins are DS1, DS2, MD, and INT. An explanation of these four pins should clear up any confusion.

The DS1 and DS2 (device-select 1 and device-select 2) pins are inputs to the 8212. The bar over DS1 (figure 3.21) means that it is active low. If a low is applied to DS1 and a high is applied to DS2, the device is activated. This means that the data latch (contained in the 8212) will capture the data. (Appendix D shows this component a little more clearly.)

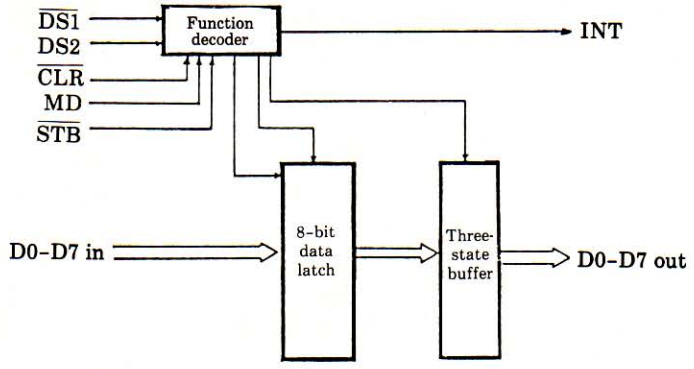


Figure 3.20
A block diagram of the 8212 input/output port.

MD is the mode-select input. The 8212 can be an input or output port. A plus, or positive, voltage on the MD pin makes it an output port. A minus, or negative, on the MD pin makes the 8212 an input port. We will hardwire the MD pin to ground since we will be using the 8212 for input purposes.

The INT pin is used by fancy interrupting I/O devices that like to stop the operation of the CPU and have the CPU take its data off the bus immediately. I don't like interrupts because they are unnecessarily complicated. So as not to rudely interrupt our CPU (or tax our brains), we'll just leave old INT hanging.

Building the Input Port

Figure 3.21 is a complete diagram of an 8212 input port for the TS 2068. The port wiring is quite simple but must be connected to the strobe of our decoder circuit. If the strobe from the decoder is connected to pin 1 of the 74LS154, then the input port's address, or device number, is 192.

Wire DS1 and MD permanently to ground. DS2 will be brought high with the select pulse from our decoder's 7402 NOR gate. Hold CLR high by tying it permanently high through a 4,700-ohm resistor. This lets the buffers take any information presented to them at any time. Since we will be reading data from the device, we'll use the RD signal to strobe the port, and we'll activate the port with the device-select pulse from our decoder.

The load of the line of the 8212 is so light (because of the three-state buffers) that it will not be necessary to buffer the incoming data lines. This further simplifies the wiring.

The one flaw in this device is that the 8212 is sensitive to static and to being wired backward, so be especially careful about both. This chip heats

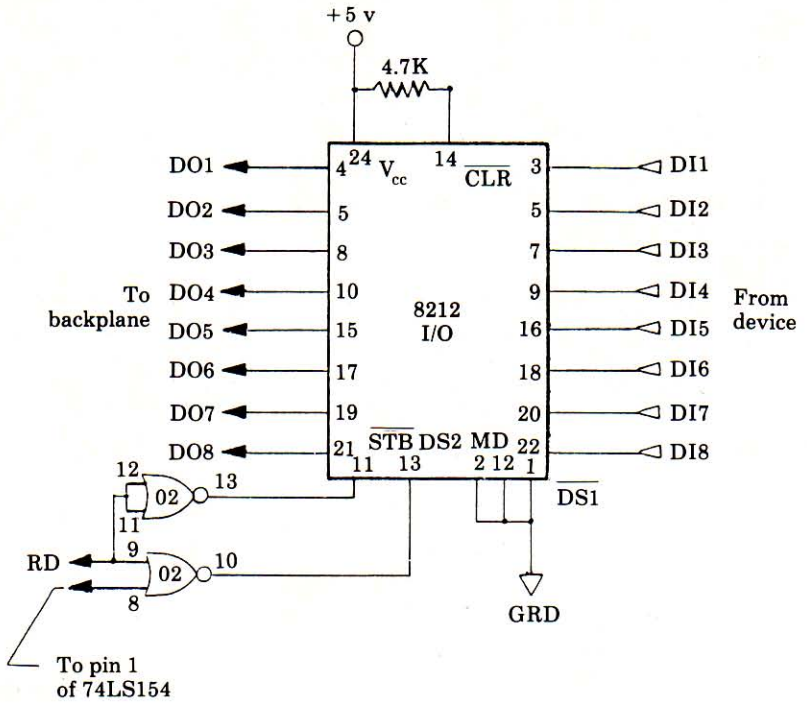


Figure 3.21
A complete diagram of the 8212 input port. This IC is a three-state device, and no data buffers are needed. Only two sections of the 7402 NOR gate IC are shown. This is the port used for all the input projects to come.

up very quickly if the power is backward. Doublecheck it until you are certain it is right!

Power the 8212 from your separate supply or from the 6-volt battery since the 8212 consumes up to 130 milliamps by itself. Turn on its power first, and then turn on the computer. If the copyright notice does not appear, or the display is wavy or fouled up, turn the power off quickly since the port's data may be getting onto the bus and could harm the CPU. Doublecheck all of the connections. Once both the computer and the port are powered and the computer appears to be operating normally, we can run some software to test our new input port.

Sensing the Real World

Now that our computer has a sensing device (the input port), we can write routines that will let it sample incoming data and make evaluations of stimuli. (It sounds as though the computer is going to get smarter, and indeed it is.)

Unless the 2068 can tell what conditions exist around it, the chances of its making intelligent choices are slim. With the input port, we can now provide information to it (other than through the keyboard) that it can interpret. This information is the feedback by the use of which the computer can exhibit some form of intelligence. Most of the time, the computer's feedback is provided by the user through the keyboard. Now the computer can get data from other devices and, in a way, become a device with eyes and ears.

Testing the Input Port

Once the 8212 I/O port is wired and ready, we can test it with a very simple routine. Enter the following two commands directly without line numbers:

```
LET A = 0
PRINT A
```

Notice that A is, in fact, 0. The computer would give an error message if we had not told it that A was some value. From now on, the 2068 will assume that A is 0 unless we tell it differently. Now test the port with the following commands:

```
LET A = IN 192
PRINT A
```

The variable A as printed on the screen is now 255. Here's why. The command told the computer to take the information from device 192 and place that value into variable A. Since no data lines were connected to our port, all lines read active (all 1s). But take line DI1 on the input port and insert it into the ground rail, and only the last seven lines will read high.

Now repeat the above two commands and see what the value of variable A becomes. Notice that it is 1 less than that previously shown. If you grounded all of the DI lines, the value would be 0, right? Try it and see. Remember that the data lines are read as active low. So, if you want to make the lines read 1, all lines except DI1 would need to be grounded.

An Intelligent Routine

Let's impart some degree of intelligence to the computer, now that it is able to sense what is happening on our end of the line. Enter and run the following small routine:

```
10  LET A = 0: LET B = 0
20  LET A = IN 192: IF B = A THEN GOTO 20
30  IF A = 255 THEN GOTO 50
40  PRINT "You just connected a line with a value of ";255 - A
50  LET B = A: IF A < 255 THEN GOTO 20
60  PRINT "There are no lines connected": GOTO 20
```

With the input port operating and the above routine running, each time you connect one of the input data lines to ground, the computer will tell you which line you connected, as though it could see your actions. The computer is using feedback.

Figure 3.22 shows a simple sense switch that can be attached to a door and connected to the computer. If the door is opened, the computer can recognize this condition and take the appropriate action if we make a small addition to our feedback routine.

Replace line 40 with the following line:

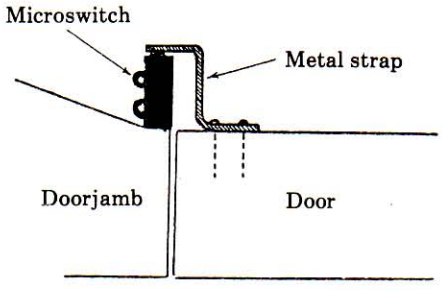
```
40  GOTO 70
```

and add these lines to the routine:

```
25  FLASH 0
70  FLASH 1: PRINT "INTRUDER ALERT"
80  FOR N = 1 TO 3: BEEP 1,0: NEXT N: GOTO 50
```

Instead of connecting one of the input data lines to the door at this time, you may simply want to run the program and ground a line to see the program's operation. This little burglar alarm shows just one small computer-feedback application. There are many others, as you will see.

Figure 3.22
A detail showing a microswitch connected to a doorway.



4

Advanced Computer Control

Computer interfacing is a never-ending challenge. That may be why I find it so fascinating. It seems that, no matter how much you learn, there is always another area or application to be tried. Each time you try something, you gain something new in the way of knowledge or technique. As technology advances, more and more applications present themselves to the experimenter. Each new advance leads to other experiments. One could not foresee a possible computer application to laser technology a few years ago. The field of fiber optics is just beginning and suggests a wide range of computer applications to the experimenter. Basically, computer interfacing centers around control. Anything that can be controlled is open for computer applications. This is especially true of electronic and mechanical devices. We'll try a few experiments in these areas, and, if what I've said holds true, you'll see many more areas that invite computer applications. Do not hesitate to try them.

Home Electrical Control

You saw in our last small experiment (the burglar arlam) that the computer can be adapted easily to the home environment. The possibilities of control applications in the home have only begun to be explored. Electronics manufacturers are trying desperately to come up with products that take advantage of the new computer mania. One enterprising home builder has successfully computerized a whole house. His applications range all the way from control of the heating and cooling to elaborate lighting and burglary protection. His garage-door opener alone is worth the price of the system. The fascinating thing is that the programming is all disk-based and open to change by the customer.

Most computer owners have contented themselves with using the keyboard for running prepared software for either games or number-crunching purposes. As you have seen, the use of the computer (especially the 2068) does not have to end there. The ease with which the 2068 can be used for electronic and mechanical applications in the home is apparent. Let's take an example and see if we can connect our 2068 to something besides a joystick.

Home Lighting Control

Most of us depend on electricity for most of the conveniences in the home. The major exception might be the commode, but I think I read somewhere that even that is available in an electrical model. (It may have a few flaws, but they are working them out, I hear. The idea of an electrically heated model is not bad.) In any event, there is a whole range of electrical appliances by which we live. Perhaps the most important electrical convenience since 1900 has been the light bulb. Without it, we would tend to revert to the Dark Ages, almost literally. Ever since its harnessing, light has been people's most favored convenience. If we could control light, we could control the world! Let's start simple and work our way up, however.

Now that we have focused in on the type of appliance we want to control, there are a number of problems we must consider. First, there is the problem of the 110 volts. This is not a good voltage for a computer to fool around with. And, even if the computer can be made to handle it, *we* can get zapped if we are not careful.

In fact, the computer, using our input and output ports, *can* handle the voltage and current required for home lighting. Knowing this, then, you can see that, if the little burglar alarm in the last section were appropriately programmed, the computer could also turn on the outside lights when it sensed something wrong. But let's not stop there: it could also turn on our bedroom lights, start a prerecorded antiburglar message, and call the dog. The electronics to control the lights and the recorder, at least, are shown below.

The Three-Channel Home-Appliance Controller

In chapter 2 you read about two devices that could be used to control the voltage and current used in the home and in home appliances, the optocoupler and the triac. We can use these with our I/O ports to activate (and deactivate) most home appliances. The ordinary table lamp will be the first appliance we will try to control.

A table lamp uses 110 volts alternating current. It also pulls around 3 amps. Watch yourself when working with this amount of juice. A circuit such as we are about to construct is not dangerous as long as it is *unplugged,* so leave it unplugged as long as you are working on it. *Plug it in only when you are testing it, and test it only when it is completed and properly insulated and cased.*

Components

The two components that will be necessary in addition to our port electronics are the MOC3010 optocoupler and the 400-volt, 6-amp triac sold by Radio Shack and other supply outlets. With three each of these components, we can build a three-channel appliance controller. Then, with the proper software, we can control at least three devices in the home simultaneously.

The Optocoupler

The MOC3010 is nothing more than an LED and a light-sensitive transistor connected together. When our data port turns on the LED in our optocoupler, the LED's light activates a transistor that controls an electronic relay (the triac). In this way, our valuable little 5-volt computer is never in danger of being blown up since it is only connected to the appliance controller by a beam of light.

The Triac

The 400-volt, 6-amp triac is a semiconductor (transistor) that can handle enough voltage and current to run large motors, home lights, toasters, and, if necessary, even hedge trimmers. Radio Shack's part number for the triac is 276-1000. By the use of this semiconductor, we can safely use our 2068 to handle most 110-volt devices in the home.

Building the Three-Channel Home-Appliance Controller

Figure 4.1 shows the complete schematic for the controller. You must remember that the control signals are generated by the computer, however. The output of our 7475 port tells the triacs when to turn on and off. The diagram of our device-select pulse decoder is not shown. The device-select lines that control our port are shown with their proper connections noted. The NOR gate that is part of the decoder is shown for clarity.

Since the controller uses 110 volts, *this circuit is not to be breadboarded.* Its construction is similar to that of our power supply. The connections are soldered with heavy, stranded wire, and the parts are mounted

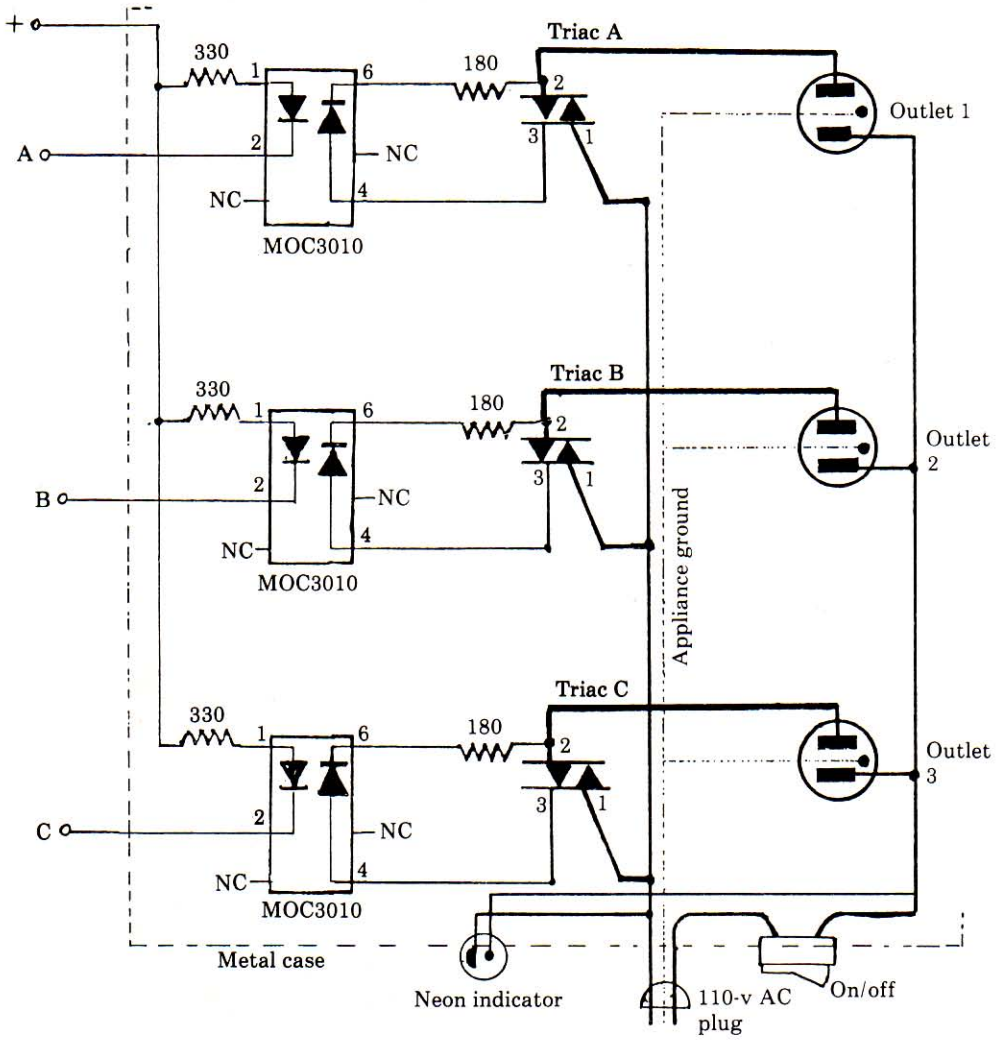


Figure 4.1
A complete schematic of the three-channel appliance controller.

on a heavy perf board and insulated from the case. Take extra care to see that no shorts occur in any part of the circuit. The case (shown by the dashed line) protects you from high voltage when the controller is operating. The control signals from the computer enter the case through plugs on the back of the controller marked A through C. Since these signal lines connect to optocouplers within the case, the 110-volt lines present very little hazard. Still, be extra careful when constructing the circuit.

The supply line that provides current to the controller should be a heavy-duty cord similar to those on drills and other power equipment. This

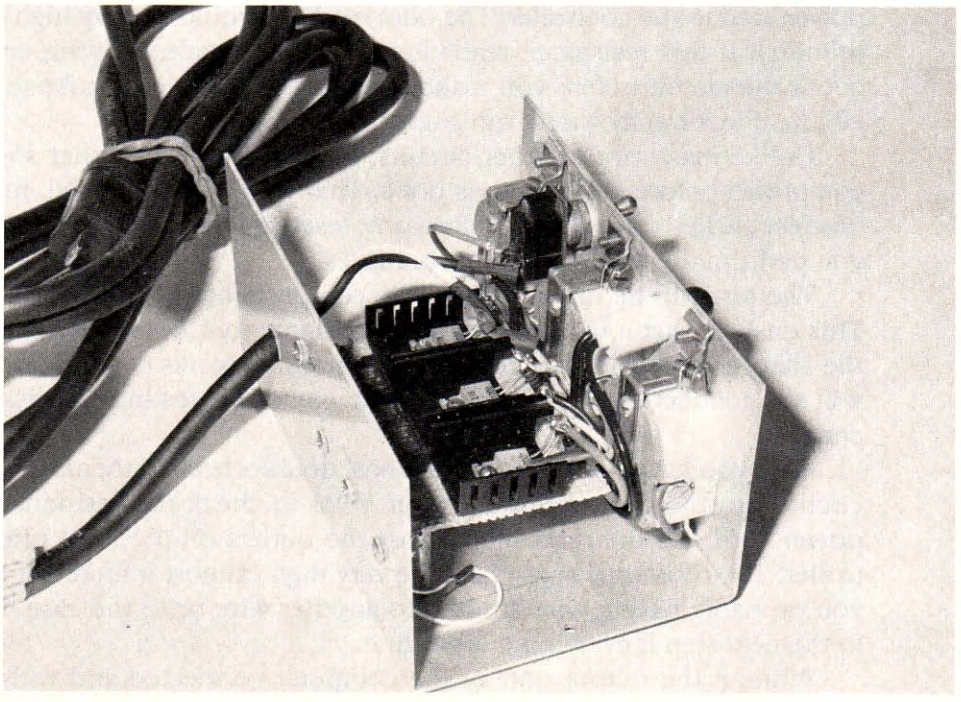
item can be purchased at almost any hardware outlet. The receptacles are also standard hardware items. The neon pilot light should be installed to indicate that the triac circuit is under power. Never open the case when the light is on or when the controller is plugged in. Use insulation such as silicon rubber compound on all connections within the controller's case.

The small, six-pin optocouplers can be inserted into two regular fourteen-pin wire-wrap sockets. The lines that connect to these components should be soldered to the posts, however. This part of the wiring can be single-strand, insulated hook-up wire. The heavy-duty multistrand wire for the house current is indicated by heavy black lines in figure 4.1. Your construction should follow the schematic closely. The photo in figure 4.2 shows the interior of the completed controller. Figure 4.3 is a photo of the outside that shows the face of the controller.

Testing the Three-Channel Home-Appliance Controller

Before the top of the case is installed and before you plug the power cord into the wall outlet, take your VOM and check the resistance through the

Figure 4.2
An interior view of the completed three-channel appliance controller.



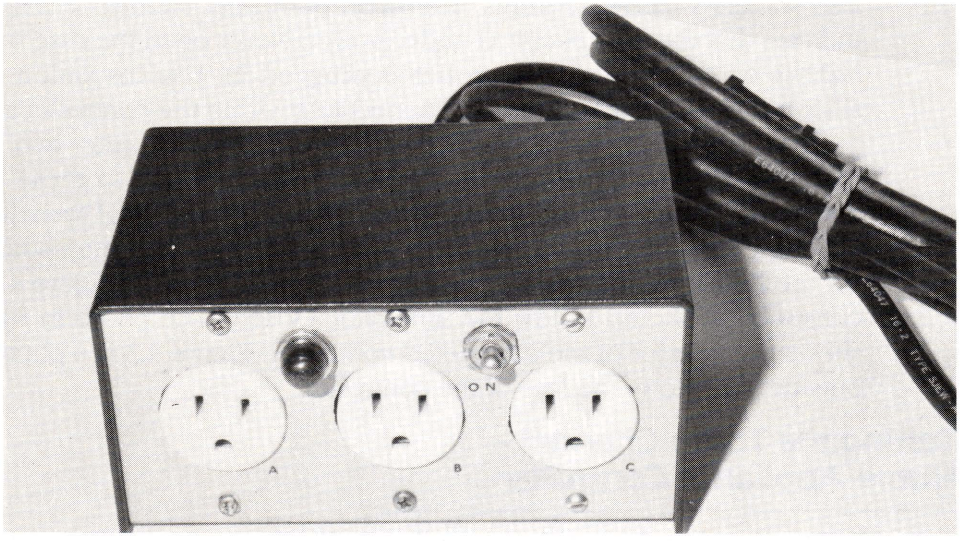


Figure 4.3
The completed three-channel controller. Notice that each channel is clearly marked. Inputs to control the channels are located on the rear of the case.

power cord to the controller. The ohm reading should be very high (almost infinite). If this resistance reads low, you have made a wiring error. Recheck the circuit before you make your smoke test (yes, smoke test—this circuit, if incorrectly wired, can smoke!).

Unlike many of our other circuits, the appliance controller should be completed before any testing is done. That is, it must be wired, insulated, doublechecked, and cased before any tests are made. Only then should you perform the following test procedures.

The test will be made *without* the computer hooked up to the device. This ensures that a wiring error will not harm your output port and slims the chances of hurting the console. If everything works correctly, only then will we connect our output port to the circuit for testing under software control.

After you have made all connections, doublechecked them, and put the circuit into the case, again use your VOM to check the resistance of the power cord and the resistance across the outlets on the front of the controller. The resistance should still be very high (almost infinite). If it is not, you probably have a wire shorted to another wire or to the case. Proceed to the next step if everything looks fine.

Without the output port of the computer connected and without any devices plugged into the outlets, turn the controller's power switch on.

The controller shouldn't do a thing since it is not plugged into the wall outlet. Now plug the controller's plug into a wall outlet while watching the components in the case. The neon light indicator should glow. If you hear a pop or see smoke, unplug the controller and check for a problem.

If the only thing that happens is that the neon light comes on, everything is probably okay. Disconnect the controller's power cord from the wall socket, and attach a positive lead from the small power supply or battery to the socket marked + on the back of the controller. Insert a signal lead into the jack marked A on the back of the controller. Do not connect these leads to the power supply or computer yet. Take a small table lamp with its switch set to on, and connect it to the outlet marked A on the front of the controller. Now plug the controller's power cord into the wall outlet. The neon indicator should come on, but the table lamp should remain off. If the table lamp comes on, there is an error or bad part, and the controller should be disconnected. Most likely, the lamp remained off. If so, take the small signal lead that you inserted into the input jack marked A, and touch the negative post of the battery or power supply. The table lamp should come on. Hold the connection for several seconds to see that nothing smokes. If everything works (without smoke), the circuit of channel A is a success! Now, repeat the above checkout procedure on channels B and C. These should perform correctly also if all the connections are right. You are now ready to run some software routines for controlling the lights.

Test Routines

You should have the output port wired and operating before the following routines are run to test the appliance controller. It might be a good idea at this time to wire-wrap both the output and input ports (the 7475s and the 8212) permanently on a wire-wrap board. We will be using these ports for further applications with other circuits on the breadboard, so things could get a little crowded. Also, the permanent construction of the motherboard or expansion board will ease future hookup problems.

With the output port operating correctly as in previous experiments, connect the appliance controller to the output port. By this I mean, connect the signal lead in input jacks A and B to data bits 0 and 1 of the 7475 output port. Since the appliance controller uses negative signals, the control-signal leads from the 7475s must come from the not-Q connections, pins 1, 11, and 14. This port will give control directions to the appliance controller. We'll not connect channel C at this time. Our first test will test only the first two channels, and then we'll test channel C by inserting the B-channel lead into C and connecting a lamp into the controller's C outlet.

Our first test is mainly to check if the channels operate on command. With the appliance controller on and the output port on, execute the following direct command:

```
OUT 192,0
```

This should have turned any lamp that was randomly on to off. Now enter this command:

```
OUT 192,1
```

The lamp connected to channel A should have come on. If it did, great! Let's try channel B:

```
OUT 192,2
```

The lamp in channel A should have gone off, and lamp B should have come on. We are making real progress. Now see if this command turns them both on:

```
OUT 192,3
```

If they are both on, the first two channels are working fine. Now turn the controller off, and disconnect lamp B from its outlet, and place lamp B's plug in outlet C. Enter this command:

```
OUT 192,4
```

It works just as our LEDs did, right? Now, that wasn't too difficult, was it? Let's try the following routine for some really spectacular results.

The appliance controller's signal leads are connected to pins 1, 11, and 14 of the 7475 output port. Three lamps can be connected at the same time to the controller. I had some trouble in getting three lamps, period, not to mention getting them into my small shop. But the result when this test routine was run was well worth the effort. Enter this routine, turn out the lights, enter RUN, and get set for a very psychedelic experience:

```
10  LET A = INT (RND * 4) + 1
20  OUT 192,A
25  FOR N = 1 TO 20
30  LET Z = 100
35  NEXT N
40  GOTO 10
```

The small FOR . . . NEXT loop in the program slows down the execution. Otherwise, the action would be too fast to be impressive. If the lamps are located at some distance from each other in the work area, their behavior as a result of this routine has a tendency to make you dizzy. If you really want to feel dizzy, change line 10 to read:

```
10  LET A = INT (RND * 8) + 1
```

Since our control lines are hooked up to only binary bits 0 through 2, our largest number is 7. By adding 1 to the range of random combinations in line 10, we make possible a second of darkness when the number is 8 and our port does not generate a signal to the controller.

Other small programs could be written that would make the three lamps do various things such as illuminate in rotation, count in binary, or even flicker at the SOUND command (really psychedelic!). You could even place the lamps in different rooms and have them come on randomly when you are away over long periods. This would give would-be intruders the idea that someone is at home who loves to play with lights. I personally hesitate to leave my little pride and joy running while I'm gone, however.

Your three-channel controller makes a good afternoon alarm clock if you connect a radio to port A and enter the following small routine:

```
 1  OUT 192,0
 5  PRINT "ENTER NAP TIME IN MINUTES ": INPUT X
10  FOR N = 1 TO X
20  PAUSE 3600
30  NEXT N
40  OUT 192,1
50  STOP
```

Okay, so you can do the same thing with the BEEP command. But the BEEP command can't start the percolator at a given time! By using your imagination, you can do a multitude of things with this little timer program. It's possible even to have the computer turn itself off when it has completed its task. Now, that's impressive.

Feedback Control

Remember our person who maintained that computers are stupid because they would try to close a window that is already shut? The factor that makes the computer intelligent, we said, is feedback, the incoming signals that allow a machine to regulate itself. Without feedback, a machine could

theoretically tear itself apart, and this is actually the most important reason that feedback is necessary. With our appliance controller, though, there is another reason for some form of feedback: to let the computer know existing external conditions in order to make proper, or intelligent, decisions.

Not all one-purpose circuits need feedback. But the computer's myriad signals and controls must work together as a coordinated unit. The addition of a simple switch sometimes is enough to let the computer know that something is wrong and that action must be taken. The small burglar-alarm switch we used with our input port (chapter 3) is a good example of a feedback device. (You can see why an input port is so important to the computer.)

An action by the computer should always result in a reaction on the part of the device. This reaction (its movement, for example) needs to be sensed by the computer. Otherwise, if the machine malfunctioned, the computer would never know. The sensing device to register the movement is the feedback circuit. The light-emitting diode and phototransistors are popular choices for feedback circuits, as is the simple microswitch. In the optocoupler, the light from the diode is registered by a light-sensitive transistor. Any interruption of the light beam will cause a signal to be produced on the output of the transistor. This signal is sent to the computer to be read as data. This data is then interpreted by the program as the condition for appropriate action.

Our appliance controller is an excellent example of a device that needs feedback circuits. In executing the next-to-last program, the computer randomly turned lamps on and off with no thought as to why or even if the lamps were already on. The computer simply ran through the program without thinking. This is certainly not a good way for the computer to act. It should at least find out first if the lights are off or on. It could determine if the room is dark or even if it is day or night. It could do all of this by means of a small light-sensitive transistor or even a photocell. Figure 4.4 is a circuit that we could add to the appliance controller that would allow it to make choices before taking action. The photocell can be placed near the lightbulb with its lens facing the light source. This simple circuit can be quickly breadboarded and tested with the following test routine:

```
 5  PRINT "PRESS O OR F"
10  IF INKEY$ = "" THEN GOTO 10
20  IF INKEY$ = "O" THEN GOTO 50
30  IF INKEY$ = "F" THEN GOTO 120
40  GOTO 5
50  LET A = IN 192: IF A = 254 THEN GOTO 100
60  OUT 193,1
65  IF INKEY$ = "O" THEN GOTO 65
```

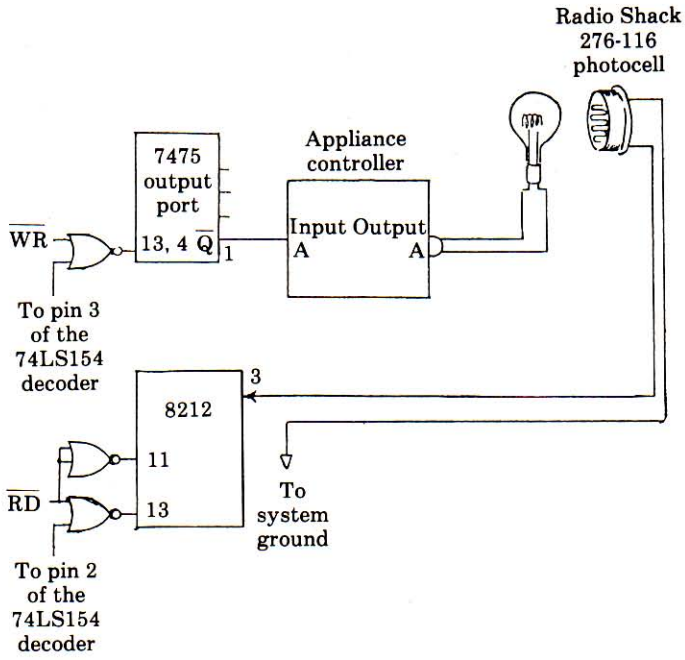


Figure 4.4
A circuit diagram for a light control with feedback provided by a photocell. Note that only the control connections are shown on both the input and output ports. The other connections, such as the data and address lines to the computer, are as shown in the schematics for the two ports.

```
 70  GOTO 5
100  PRINT "LIGHT IS ALREADY ON"
105  IF INKEY$ > "" THEN GOTO 65
110  GOTO 5
120  LET A = IN 192: IF A = 255 THEN PRINT "LIGHT IS
     ALREADY OFF"
125  OUT 193,0
130  IF INKEY$ = "F" THEN GOTO 130
135  GOTO 5
```

Now, with the appliance controller circuit connected and the feedback circuit attached, touch the O key to turn the light on and the F key to turn the light off. The program checks to see if the light is either on or off before taking action. Several lines in the program check the keys to make sure that the key is released before continuing. These lines form a software *debounce* procedure. It eliminates the repeat printing that the program

would otherwise cause since the computer can run through the program three or four times before you can release the key.

The table lamp is connected to our appliance controller's output A (pin 1, or not-Q), and the feedback is taken from our 8212 input port. If the signal level is high, the computer knows that the lamp is already on. Try touching the O key with the lamp already on. The screen should give you the message that indicates that the lamp is active and that no further action is required. The program works similarly if the F key is pressed with the lamp off. In a simplified way, this one photocell acts as do our eyes. With a little imagination, you can implement other feedback circuits that give the computer other kinds of visual information. But, despite the fact that technology has come a long way in the last few years, one goal yet to be achieved is that of providing really good visual information to a computer. We can keep a computer regulated by using simple circuits such as that shown in figure 4.4, but providing visual information comparable to that gathered by human vision has not been perfected.

The Mail Indicator

Data input to the computer is probably much more important than output. On the basis of input, the computer can make choices that eliminate much of the drudgery involved in our lives. The computer can take data such as the information in your checkbook and formulate a home budget for you in a matter of seconds. It can balance your checkbook, chart your horoscope, and maybe even predict your future. In addition, if it receives the right data through its input port, it can monitor many things that we ordinarily spend hours monitoring with our senses. Let's take a small example involving only a photocell.

How many times have you gone to the mailbox only to find that the mail carrier had not yet arrived? Or how many times have you watched the street, waiting for that letter you needed so badly to see? Why not let the computer watch for you and tell you when the mail comes? The following circuit, the mail indicator, is a simple feedback loop that will do just that. When the mailbox door is opened, the computer senses the light and tells you that the mail has come. There is no need to hook anything to the mailbox. Simply place the photocell in the back of the box, and close the door. The absence of light in the box make the bus read 1. When the door is opened, the photocell places a negative voltage on the bus, causing it to be read as 254. You might have to add a resistor of around 500 ohms to one end of the photocell (if your box lets in too much light). Of course, you could rig a simpler, mechanical mail indicator by just attaching a bell to your mailbox door, but the thought of having the little computer do

your looking is very satisfying. The arrival of the message on the screen is especially nice. The monitoring program itself has a built-in bell that sounds when the mail arrives to remind you to look at the screen. Therefore, it is not even necessary to watch the display.

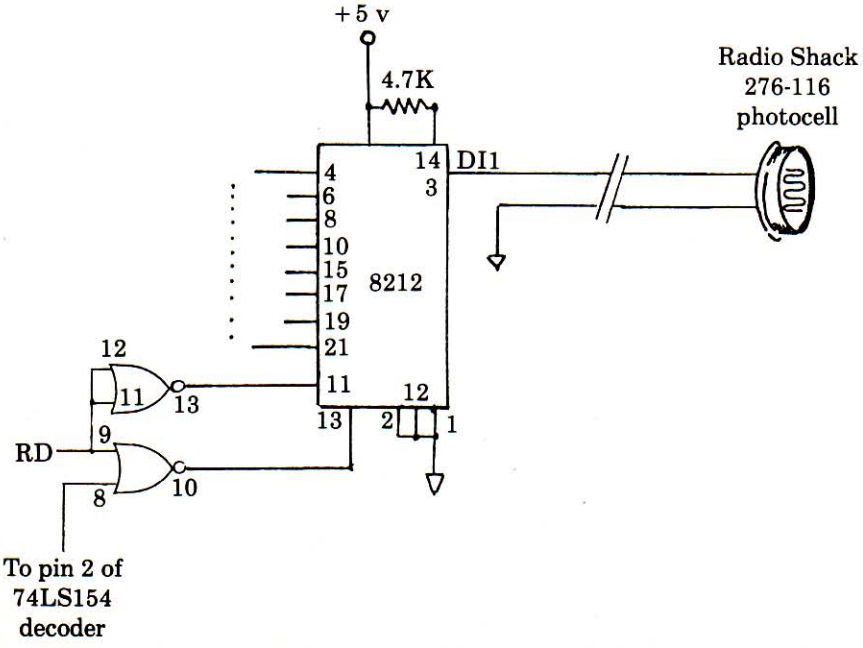
The two wires for the indicator eventually have to be strung out to your mailbox. I wanted to keep my indicator lines in place, so I buried a small two-conductor wire next to the driveway that runs out to my box. If your box is on the front porch, you'll have less trouble. If you live in an apartment building, you had better check with the landlady or landlord. (If your box is a post office box, then I guess you're out of luck.)

There are many other things this little circuit can do. For example, it could monitor your driveway. It could even count the times an object passes between its sensor and a light source (as on an assembly line).

Construct the mail indicator circuit (figure 4.5), and run the small program below. Remember not to start it running until around the time the mail is due since it will tie up your computer's time too much otherwise.

```
10  PRINT "I'M WAITING FOR THE MAIL CARRIER"
15  LET A = IN 192
```

Figure 4.5
The circuit of the mail indicator. This circuit can be used for any application that monitors light intensity.



```
20  IF A = 254 THEN GOTO 40
30  GOTO 15
40  FOR N = 1 TO 15
50  BEEP 1,1
60  NEXT N
70  PRINT "THE MAIL IS HERE!"
```

The Automatic Tape-Recorder Controller

Let's say that you didn't choose to tackle one of the previous projects. That's fine; I don't blame you for wanting to start with something less complicated than, for instance, the appliance controller. There are other things that can be done with the 2068 that are just as interesting and helpful as the projects previously discussed. Before you start any of them, however, you must read all the material on electronic controls that has already been presented. You must do this in order to understand some of the things that are presented here. Many of the relevant component diagrams also are shown in previous sections. It is especially necessary that you understand and build both the 7475 output port and the 8212 input port (chapter 3) for the upcoming projects.

There are many things that you could try. After reading some of the control details, you probably can think of projects that are more exciting than the projects that are listed here. By all means, be as creative as you wish. With the 7475 output port and the 8212 input port, you can do almost anything. Take a shot at your wildest idea. It might be just the thing that we've been looking for since the invention of the transistor.

The projects presented in the rest of the chapter are mainly one-purpose designs, but it's quite possible to combine some for several purposes. The electronics aren't simple, though; they involve state-of-the-art components. Again, study the previous discussions of control before you build any of the circuits presented from here on.

The Timex/Sinclair 2068 is blessed with the ability to load and save data files separately, but not automatically. You must be there in person to push the tape recorder's *play* and *record* buttons to turn the recorder off or on at the exact second that it has finished or needs to start its task.

You can use the output port to have the recorder turn itself on and off automatically to save and load data files. If you don't want to wait around to cut the recorder off when the file of your choice has been found and loaded, simply insert a small program like the one at the end of this chapter, and go take a break. The recorder will shut itself off when the load (or save) is complete.

Turning things on and off is a specialty of the 2068. Most recorders can be hooked up to the 7475 output port to obtain this on-and-off switching capability. A 5-volt relay (figure 4.6) is a good device to use for this purpose since we will be switching the recorder motor on and off. The motor that drives the recorder's spindles is not small, so a relay is used to switch the power. Figure 4.7 is a complete detail of the components and connections you'll need in order to build the recorder's control.

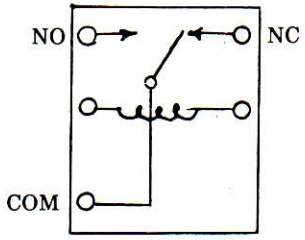
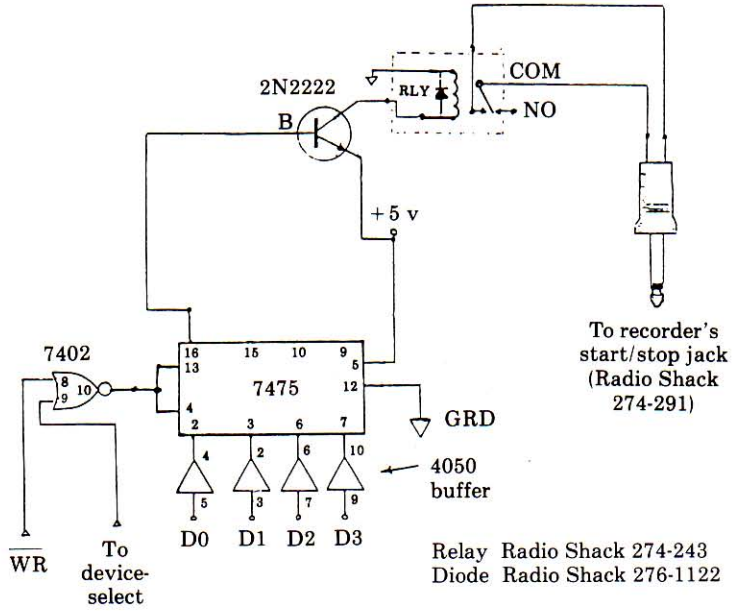


Figure 4.6
A pinout of the Radio Shack 275-243 5-volt relay. Other relays have different pinouts.

Figure 4.7
The complete circuit of the automatic tape-recorder controller. The start/stop plug should be selected to match the recorder's control jack.



First, you need a recorder that has the small extra jack on the MIC connection that allows you to use the mike as a pause button. Most recorders have this. But, if yours doesn't, then you might want to remove the recorder's case and add a pause jack to the red lead of the motor wiring. It's not too complicated to do this with most units.

The control diagram (figure 4.7) shows that the relay is driven by a 2N2222 transistor that is turned on and off by the data presented on pin 16 of the 7475 output port. If the data reads 1, the recorder's motor is engaged; if the data reads 0, the recorder stops. Of course, you will need a small jack plug that fits the pause jack. This plug can be easily obtained from most local supply outlets such as Radio Shack. The Radio Shack number is listed in figure 4.7.

Notice that the common connection of the relay is resting on the NO (normally off) post (figure 4.7). The activation of the relay simply connects the COM (common) and NO points, and the recorder's motor starts. Deactivation (giving the data word 0 to the port) stops the recorder. If you need to operate the recorder without giving a command to the port, just unplug the pause plug, and you are back to normal operation.

Wire the circuit as shown in figure 4.7. Since there are so few components and connections, this is a good circuit to breadboard for testing. You may notice that there is a small diode connected between the coil leads on the relay (figure 4.7). The diode is there to cancel any spikes that might develop when the coil is de-energized. You really should insert one to be on the safe side. I didn't, and there were no problems, but, if the computer had been driving the relay directly, you can bet that there would have been. If you want to be safe and not take a chance on ruining your 2N2222 transistor or the 7475 IC, insert the diode as shown. The Radio Shack number for it is 276-1122. Once the complete circuit is working, you might want to wire it more permanently. As shown, the 7475 port is connected to device-select 192. You can change this to 193, 194, and so on by selecting various pins on the 74LS154 decoder (see figure 3.17 in chapter 3).

Once you have the controller wired, make sure the port is cleared by entering:

```
OUT 192,0
```

Then test the controller's operation by inserting the pause plug into the pause jack, depressing the *play* button, and entering the following direct command:

```
OUT 192,1
```

The recorder should start immediately. To stop the recorder, enter:

```
OUT 192,0
```

Now you are ready for the big time—automatic data loading and saving, just as the Wangs can do! Enter and run the following program with the recorder connected. The response of the program will amaze you:

```
  1 LET X = 0
  5 DIM A(50)
 10 FOR N = 1 TO 50
 15 LET A(N) = X
 20 LET X = X + 1
 30 NEXT N
 40 OUT 192,1
 45 PRINT "PRESS RECORD AND PLAY BUTTON"
 50 SAVE "LIST" DATA A( )
 55 OUT 192,0
 60 PRINT "IF YOU WISH TO RECALL DATA FILE, PRESS PLAY
    AND ENTER Y"
 70 INPUT Z$: IF Z$ = "N" THEN GOTO 100
 75 PRINT "PRESS REWIND": OUT 192,1
 80 LOAD "LIST" DATA A( )
 85 OUT 192,0
 90 FOR N = 1 TO 50: PRINT A(N): NEXT N
100 STOP
```

Lines 1 through 20 of the program set up a numeric array and fill it with numbers from 1 through 49. This array could be file data that has been entered with an accounting program. Line 40 activates the recorder, and line 50 saves the data on tape. Line 55 stops the recorder.

To play the data back into memory from tape, press *rewind* and enter Y in response to the input command in line 70. The 2068 will prompt you to press the *play* button and will automatically load the data and display it on the screen. This program only shows the capability of the controller, but it could be effectively used as a routine in a much larger, multitask program (such as a data-base or spreadsheet program) that needs to store and load several different files. With a little ingenuity and a couple of electromagnetic solenoids hooked up to a couple of relays from the port, you could even automate the pressing of the *record, play,* and *rewind* buttons. I can't tell you exactly how to experiment with that idea because recorder mechanisms vary considerably. But, generally, just connect a relay to the solenoid, connect the solenoid shaft to the appropriate lever(s) on the recorder, and you will have a completely automated unit. (Radio Shack's part number for the solenoid is 273-251.)

5

Microbotics

Robotics

You have seen that the Timex/Sinclair 2068 is well suited for control applications around the home. What is even more interesting is that the 2068 offers all of the signals necessary for serious application to the field of robotics. Robotics is the most challenging aspect of computer control that can be undertaken.

The area of *microbotics* (control of robotic mechanisms through microcomputers) is fascinating but complicated. The experimenter in microbotics must understand not only the electronics necessary for control applications but also mechanics, programming, and the nature of feedback. That is, you need four kinds of knowledge in order to apply the microcomputer to this interesting field. This chapter will attempt to cover all of these aspects so that you can construct a demonstration model of a working robotic platform. I avoided using the word *robot* just now since this term implies humanlike appearance or at least behavior (not unlike that displayed by the *Star Wars* androids). Most robots do not in fact look like people.

We will cover simple electronic control of actuating mechanisms, environmental sensors, and robotic programming for demonstration purposes. Although all of the basic knowledge and hardware necessary for the construction of a mechanical creature are presented in this chapter, the chapter is not meant to be a step-by-step assembly guide. The general mechanical principles are shown, and particular designs are discussed, all so that you can apply the principles in your own similar design. Mainly, this chapter deals with the electronic control of any mechanics that you construct.

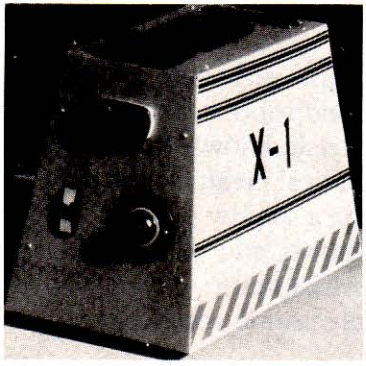
This control and the associated programming should enable you to connect components similar to the ones I discuss for very impressive robotic demonstrations.

Thus, the design of the particular robot shown in this chapter should not necessarily be copied in detail. Robotic designs can be as varied as the human hands that create them. Most are developed to fulfill a specific purpose, and many do not even resemble their human counterparts (see figures 5.1 and 5.2, for example). A robotic platform that drills a piece of aluminum day after day might look more like an assembly-line frame than a human operator. Most designs are determined, too, by the parts that you, the designer, have on hand.

One thing's for certain. Robots are here, and here to stay. They will become more and more sophisticated as technology advances. Not too far in the future is the working android that will seem to have most human capabilities. The advancements that have been made in robotics in the last ten years are astounding. The advancements that will be made in the next five years will make today's robots look primitive.

In order to study robotic control, we must study its various aspects. The most important, of course, is the electronic. More fundamentally, however, we need to know something of the mechanics of the things to which the electronic controls will be attached. In most of our experiments, we will use the small DC motor as the activating force. There are other actuators that can be used effectively. These include compressed-air actuators, hydraulic devices, and even such radical activators as thermocouplers. Each sort has it own advantages and disadvantages. We will use the DC motor since its parts are easy to either obtain from local and mail-order supply

Figure 5.1
Robot Shack Inc.'s X-1. This unit has an object avoidance sensor and can follow a moving target such as a person.



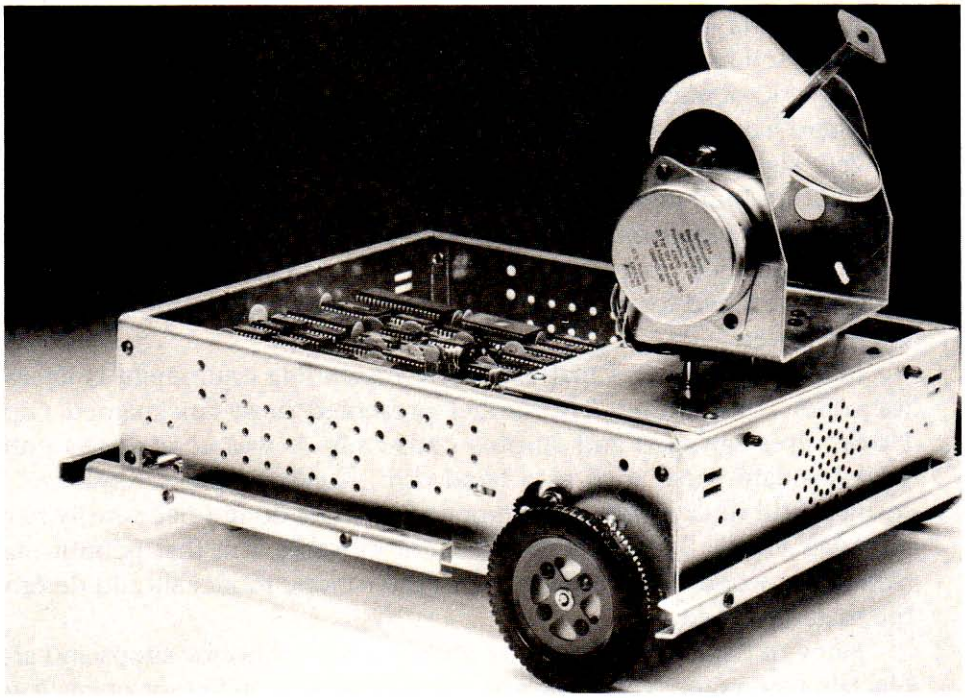


Figure 5.2
The Scorpion 1 sold by Rhino Robots. This unit is quite compact and carries most of its electronics with it.

outlets or salvage from mechanical items around the home such as toys and even old autos. Above all, the design presented here is meant to show concepts of robotics all the way from the mechanics to the programming. The robotic device shown is only meant as a test bed for the application of these concepts. The low-resolution platform presented here would make a good demonstrator for classroom use, an excellent science-fair project, and a great device on which to practice the skills necessary to build that android that you or someone like you will soon create.

Robotic Parts

The most difficult part of constructing a mechanical robot is obtaining the necessary parts. Looking through robotic magazines will convince you that only eccentric millionaire geniuses can ever attempt such a venture. The special parts alone could cost more than the highest-priced personal computer system you could use to run the robotic device. Fortunately, special robotic parts do not necessarily have to be used to construct a working

model. Most of the necessary parts can be obtained quickly and inexpensively. A parts supplier list is included as Appendix A of this book. The suppliers sell surplus and new components (figure 5.3) that can be adapted to the world of microbotics.

There have been ingenious designs that have incorporated toy mechanics very successfully. Many toys sold today not only use microprocessors but also are built along the same lines and use the same mechanical principles as some multimillion-dollar robots. The Armatron arm sold by Radio Shack (figure 5.4) is a prime example. This so-called toy would make an excellent test bed for most of the electronic and programming applications we will cover.

Most beginners feel that expensive machining equipment is necessary for robot construction. The model presented here (nicknamed Captain Hook) uses salvaged and surplus parts only. It was constructed entirely with standard tools such as a hand drill, hacksaw, and screwdriver. The result is still quite good. Even expensive metals do not necessarily have to be used. Figure 5.5 shows an excellent movable arm that is built mainly from balsa wood. The ultimate use of the robotic model should determine the materials used.

Many of Captain Hook's parts came from local hobby shops and an old electric typewriter. The junkyard has almost all you'll ever need. It gives you a special sense of accomplishment to see your old typewriter scuttle around the room, picking up objects and dropping them into your hand.

The motors are surplus items that cost less than $3 each. Several old model-airplane parts were also incorporated. The gears (for the most part) came from miniature gas-powered cars. Parts such as these can be easily obtained from a local hobby shop.

Now that I've convinced you that robotic experimenting is not only easy but inexpensive, let's look at some of the necessary mechanics and a little of the electronics for a robotic platform.

The Robotic Platform

Designing the Platform

Most self-respecting robots need some form of mobility. In order fully to grasp the intricacies of robotic mobility, we will need to look at several basic factors.

First, we must consider what means of power to use. Most hobby work that is being done in robotics today relies on direct current supplied by batteries. These can range all the way from D cells through 6-volt motorcycle batteries on up to 12-volt, 26-pound car batteries. I chose to use a

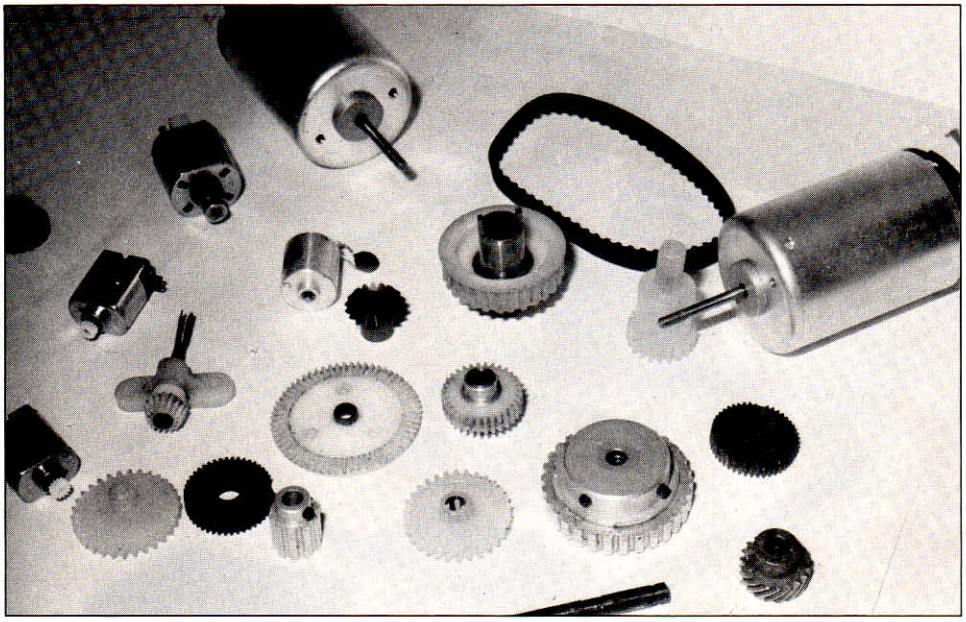


Figure 5.3
Gears and motors. You must have a good supply of them before you start to build your robotic device.

6-volt motorcycle battery in my project but later changed to a lantern-type battery because it weighed less.

You might want to explore what is available in DC motors before you choose a power supply. Several big robot designs have used auto starters as drive motors and thus needed 12-volt car batteries. I myself wanted to keep Captain Hook small and portable; I didn't intend him to do things like picking up and moving the refrigerator, after all. Once you choose the motors and power supply, you have an idea of how much your robotic device itself will weigh. This is an especially critical design consideration. I once built a small radio-controlled platform only to find that I had so much weight in and on it that it refused to move an inch.

Now you need a good *platform.* This is the base to which all the rest of the components are attached. It should be strong, yet light in weight. Aluminum is the best choice for the metal framework and base of a robotic mechanism, because aluminum is lightweight, strong, easy to handle, and easy to form (drill, bend, cut, and so on).

As when assemblying electronic circuits, you must commit a design to paper beforehand. You need to know where the parts will go and if there will be enough room for them when the time comes to attach all of the

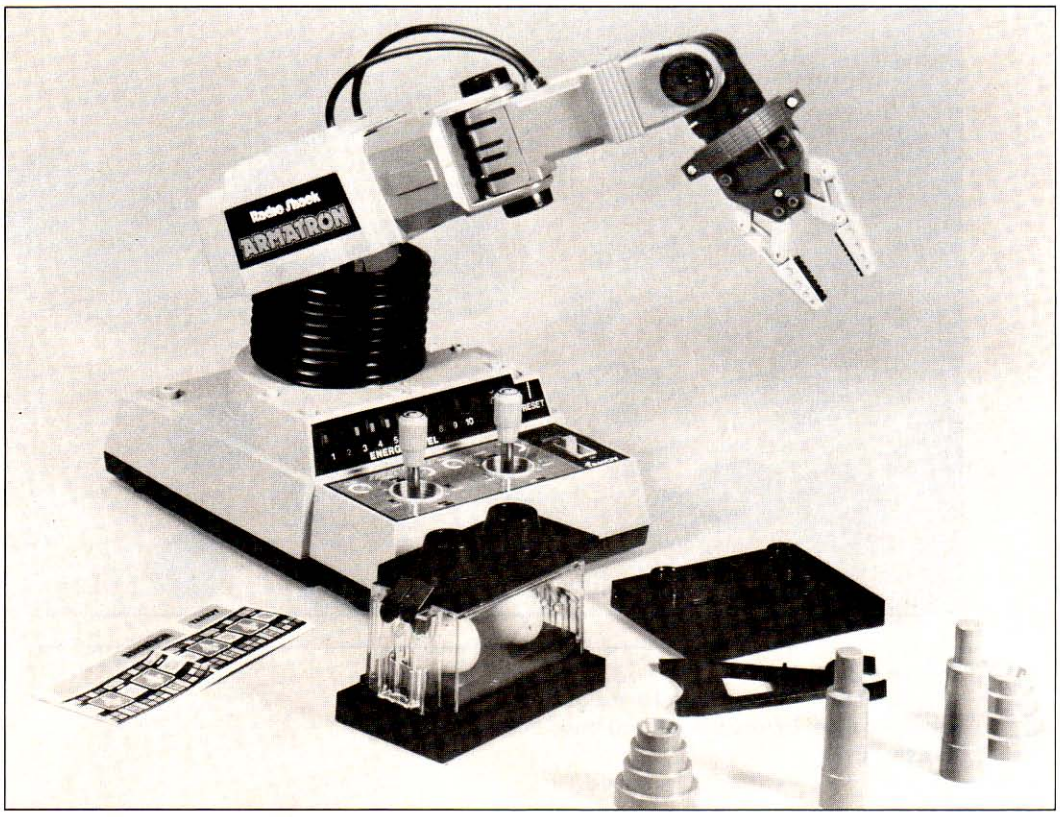


Figure 5.4
The movable Armatron arm sold by Radio Shack.

pieces. A drawing of a small motion platform is shown in figure 5.6, but this, again, is only an example. Your design will be determined by several factors, including the size of your motors, wheels, and other components. Obviously, you should gather most (if not all) of your parts before you begin construction.

Captain Hook's platform is a sheet of 3/16-inch rigid aluminum. It could have easily been a lattice of aluminum pieces. Whatever you use, make sure it is large enough for the ultimate design. Captain Hook ran a little short on platform space. In any event, it should be strong enough to hold a good deal of weight. I estimated that my model would weigh around 3 pounds when finished. It actually weighs three times that much!

Motorized Movement

All of the mechanical motion in our design is a result of drive provided by one of several small DC motors. Even small hobby motors can be used

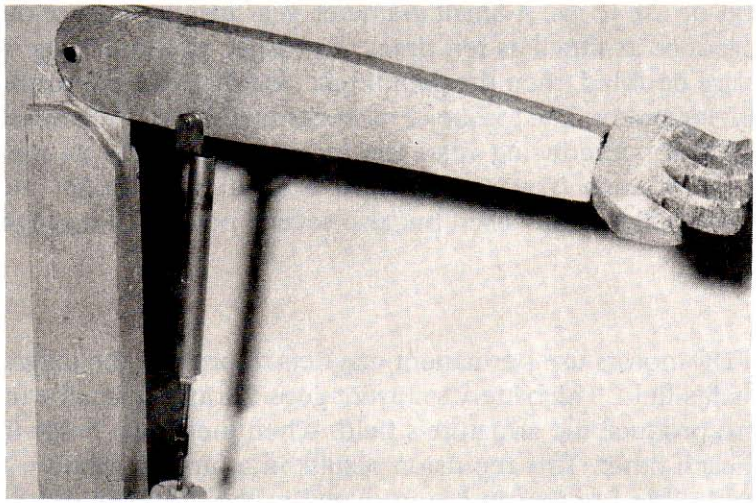
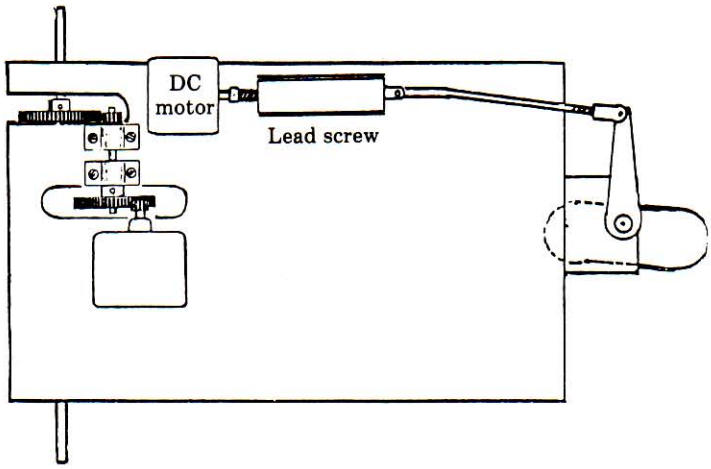


Figure 5.5
A movable demonstration arm made of balsa wood.

Figure 5.6
A top view of a robotic platform.



successfully in small robotic models. But most small DC motors have to be geared to provide the necessary turning power, whatever the size of the design. This turning power is called *torque.* Most small, hobby-type motors don't possess enough torque even to turn a small freewheeling propeller, much less drag a 10-pound robot around the room. To compensate for the lack of torque, we will need to use gears to change speed just as in an automobile's transmission. If a gear increases turning speed, it loses torque.

If a gear decreases turning speed, it gains torque. It is this latter fact that will be of use to us. A small-diameter gear turning a large-diameter gear can produce as much as ten times the torque of the driving motor. If the gears are doubled, then the gain might be fifty times the turning power of the driving motor. The torque of most hobby motors is measured in ounces. A double set of reducing gears can increase this to the torque required to move 12 pounds. You can see that a small, lightweight motor can be geared to pull not only itself but also several pounds of luggage.

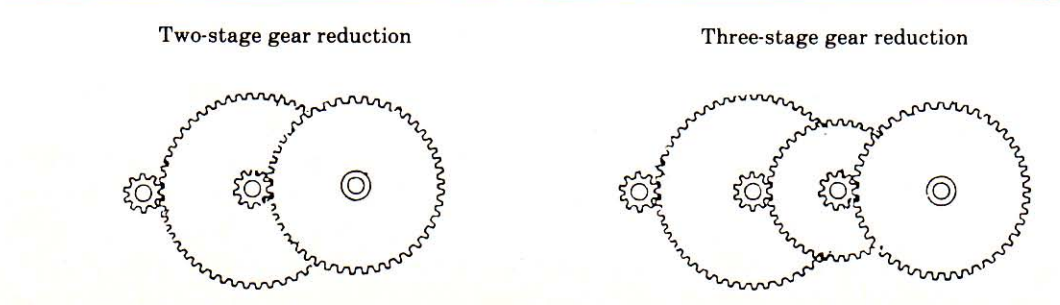
DC Motors

Small DC motors use permanent magnets to provide the magnetic force of the shaft's field. The battery's current goes through a small winding on the shaft to produce the armature's field. When these two fields interact, they repel each other. This repulsion results in a turning motion. You can see that very little torque can be produced in this way, but what can be produced is speed. A small DC motor may revolve 2,500 times a minute. Gears are needed to reduce this speed and increase the motor's torque (figure 5.7). It all works out very nicely.

If you use a small, toy motorized platform, you can save yourself many headaches. Figure 5.8 shows a platform made from the base of a small motorized dump truck. The torque that is produced by this model is amazing. What is more amazing is that the motor producing this huge torque is tiny! So, remember, your motor doesn't have to be huge if the gearing is appropriate.

DC motors usually run on 6 volts. You can increase this a bit, but a voltage increase usually increases the speed. If there is too much voltage, the DC motor will overheat. There is one electrical characteristic of DC motors that works to our advantage, though. If the leads of the current source are reversed, the motor drive direction is also reversed. This lets us use one motor for driving our mechanism in two directions. This character-

Figure 5.7
Reduction by gearing. Both arrangements increase torque and reduce speed.



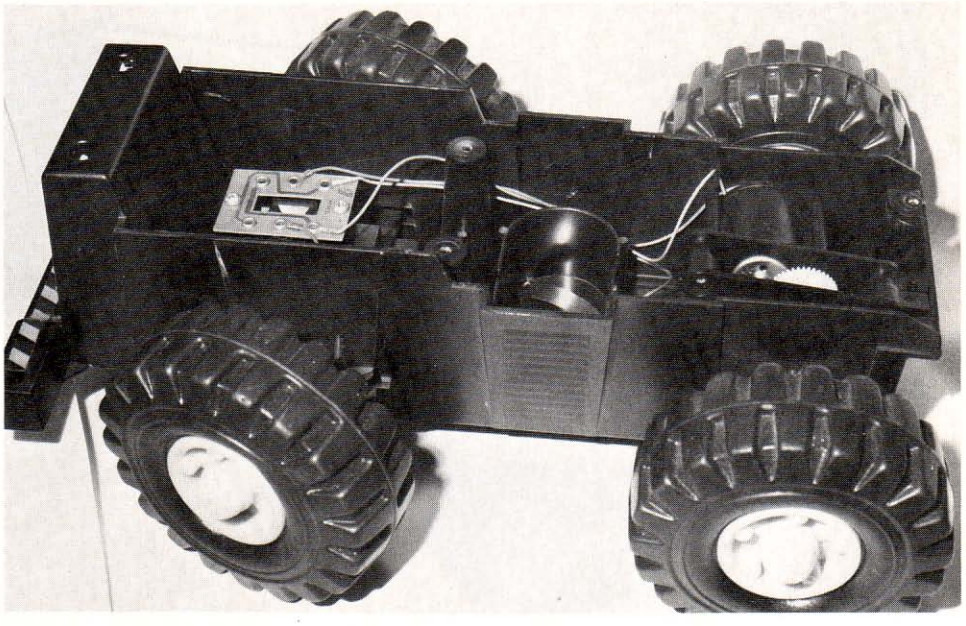


Figure 5.8
The base of a small motorized dump truck, which can be used as a robotic platform.

istic of the DC motor eliminates many of the components and much of the weight that would otherwise be necessary. The circuits that drive the platform forward can also drive the platform backward by simply reversing the currents.

Building the Platform

After you have estimated your model's size and weight, you need to obtain the motors, if you have not already done so. You may need to experiment with several. It could be that these are readily available locally, as I have said. Once you get these motors, find appropriate gears. Again, many gear sizes are available in used pieces of equipment, including toys. Then, using the general layout in figure 5.6, you can construct the robot's movable base. Notice that the wheels on Captain Hook are small (figure 5.9). These also serve to increase the platform's pulling power but make for difficult travel over such rough terrain as rugs, carpets, and doorjambs. I chose to use double gearing, but now wish I had used triple gears, because of the weight.

Rather than use screws and taps on all connections, you might consider using pop rivets or tapping and threading the base to take machine screws. These strategies save some weight, and every little bit counts.

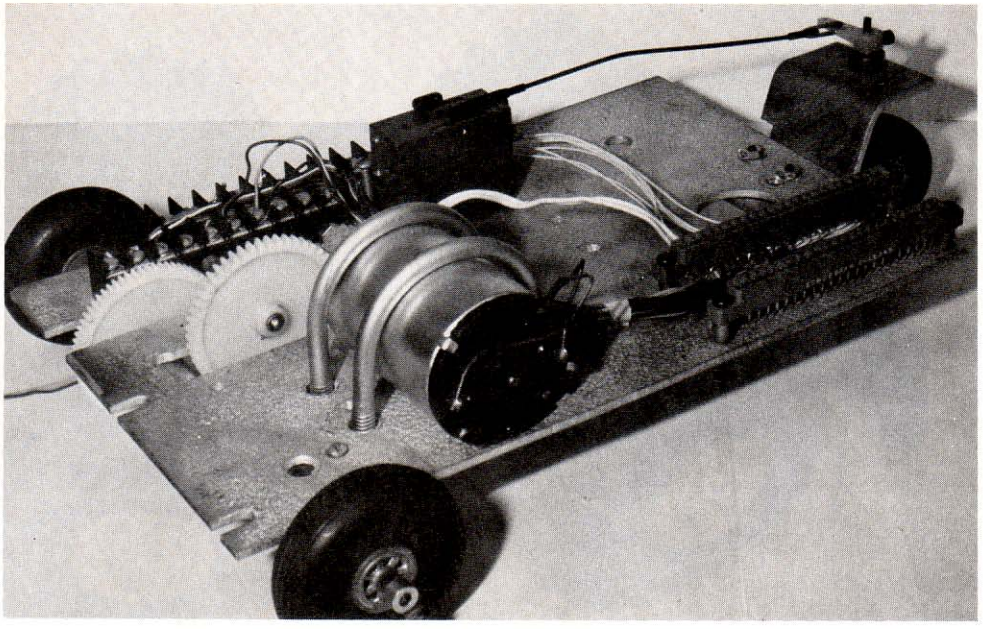


Figure 5.9
The motorized robotic platform.

You will probably elect to build your own design around the parts that are available and your own special needs. You might want your creation to be an original, and that's just great! What is important is that your mechanics be neat and correct. Otherwise, your electronics and subsequent programming will refuse to work. (Once, while testing a new arm design that incorporated switches for feedback, my program seemed to be skipping a whole feedback section. In reality, the switches were so sloppily done, they were not even contacting.) I will therefore explain the techniques of construction throughout the sections that follow. These techniques will then let you create your own version of that special robot.

Figures 5.10 and 5.11 detail how Captain Hook's drive mechanics looked on paper. Naturally, the final design was not as originally conceived, but, for the most part, the Captain came out surprisingly like the paper model. Improvements to the first drafts have been incorporated into the drawings you now see. What is important is that the first sketches gave me the outline I needed. They showed me some design problems I had not foreseen. They allowed me to correct some errors before they manifested themselves as incompatible parts!

I made the drive gear ten times the size of the smaller gear to increase the motor's torque. The base should be strong and light, and the gears should be nylon or plastic. Small bushings should be installed on gear shafts that

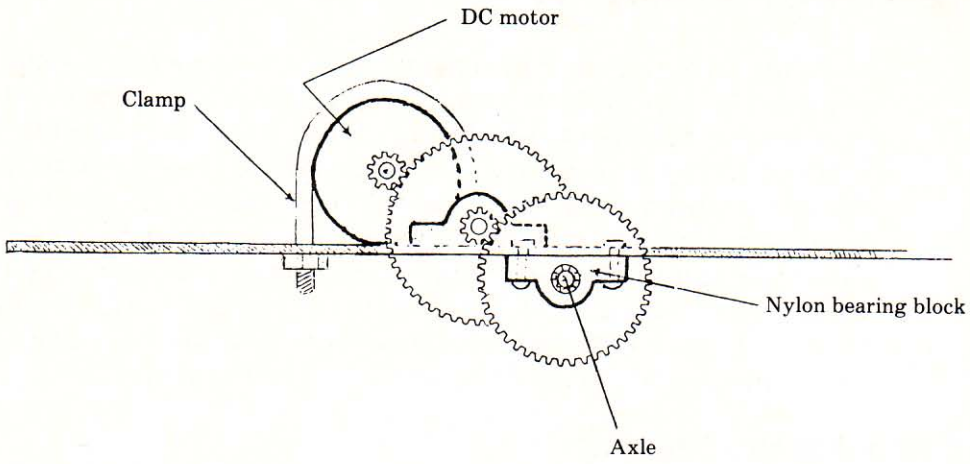
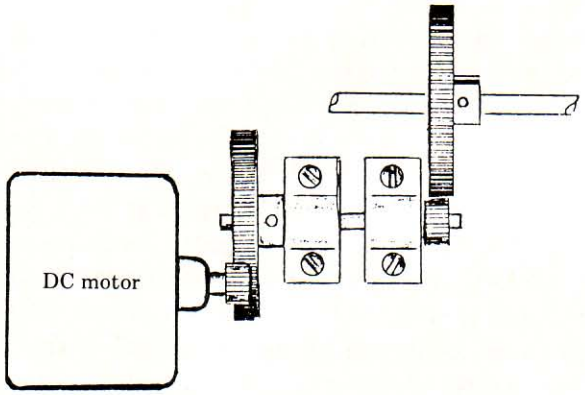


Figure 5.10
A side view of the drive assembly of the robotic platform.

Figure 5.11
A top view of the drive assembly of the robotic platform.



are too small for the shaft hole of the gears. A pin through the gear hub and its shaft will keep the two units operating as one and will also keep the gear from wobbling on the shaft. Use bearings where possible to cut down on friction. This is especially critical in the rear, or drive, axle since this point will take the most weight. Small, ¼- to 1-inch bearings can be ordered from Stock Drive Products (Appendix A).

The selection of wheels is important because traction is critical in some maneuvering situations. Captain Hook has a tendency to slip since his drive wheels are narrow. More experimenting might be needed here.

Testing the Platform Drive

Once the drive section of the platform base is complete, you simply hook the motor's leads to the battery, and observe the drive power and operation. Reverse the leads to see the unit driven in the other direction (figure 5.12). Hold up the platform in the front with your hand since the steering has not yet been attached. It might be necessary to adjust the gears for better alignment since out-of-line gears have a tendency to reduce power. Now stack several books on the platform, and try connecting the leads. If the platform does not still move quite readily, use bigger intermediate gears or a larger motor. (Always make sure your battery is fully charged before you decide you need a heavier motor or a larger gear.)

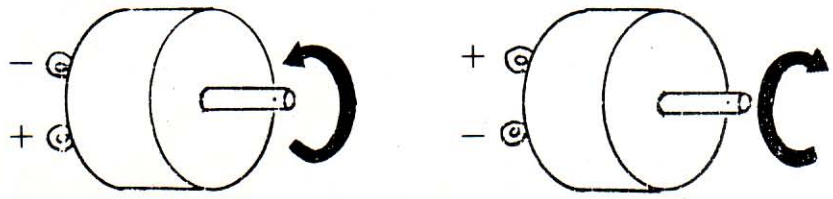
Designing the Steering

Robotic Steering Mechanics

Now that you have given your creature the power to move, you must give it a means to direct that motion; that is, it must be able to steer itself in the desired direction. There is nothing that looks more helpless than a computer-controlled robot blindly banging into a wall. It's pathetic. The robot needs to be able to guide its own way.

There are various arrangements that will provide some form of steering to our device, some of which are shown in figures 5.13, 5.14, and 5.15. You may find a simpler method than the ones shown here. I like the technique illustrated by figure 5.16. This gives the device a certain degree of proportional control in guidance (by degrees rather than all or nothing). There are several mechanical principles at work in this method. The lead screw provides the gearing to increase the torque of the DC motor. It also serves to change the rotary motion of the motor into a linear movement. This linear travel is further reduced by the control arm on the front wheel of the platform as shown by figure 5.6. All of these actions tend to slow down the process of steering. This makes the steering smooth and slow.

Figure 5.12
The rotation of a DC motor, depending on the connection of the leads to the terminal.



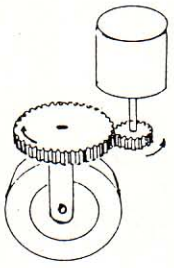


Figure 5.13
Spur-and-pinion steering.

Figure 5.14
Opposing-motor pivot steering.

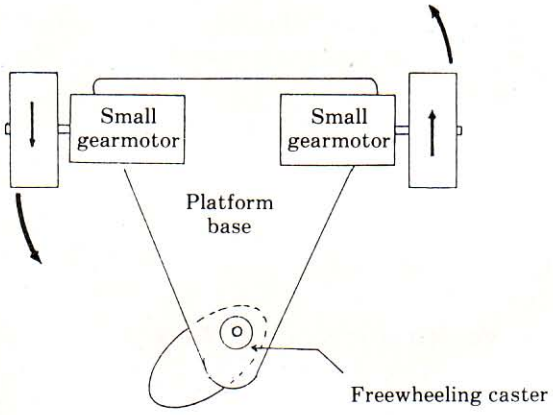
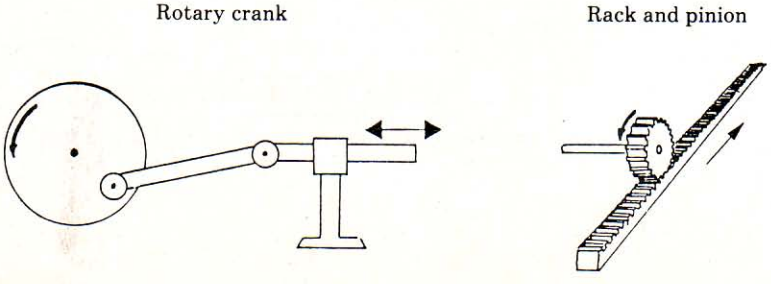


Figure 5.15
Two mechanisms that change rotary to linear motion.



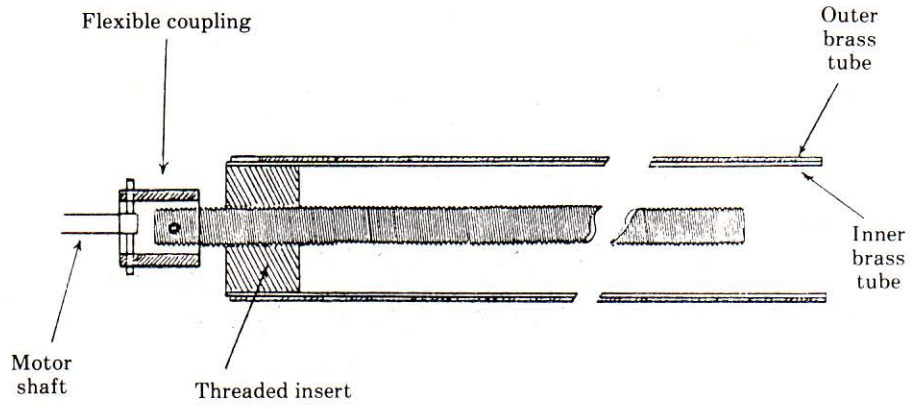


Figure 5.16
Lead-screw method of providing linear movement.

These are good mechanical characteristics of robot steering design. If speed were our concern, we would be building something more like gas-powered race cars, right?

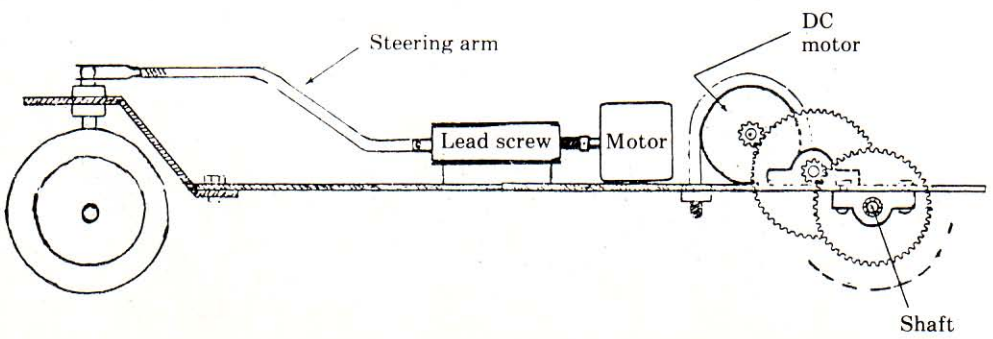
Figure 5.17 is a complete detail of a good proportional steering mechanism. I have built the one shown and was pleased with its performance.

The steering arm of Captain Hook's platform is shown in figure 5.18.

The Lead-Screw Actuator

The lead-screw actuator is one of the simpler devices for converting rotary movement into linear movement. (Linear motion is more adaptable than circular motion to robotic mechanisms because most robotic movement, such as extension, clamping, and elevating, is itself linear.)

Figure 5.17
A side view of the robotic platform showing the lead-screw method of steering.



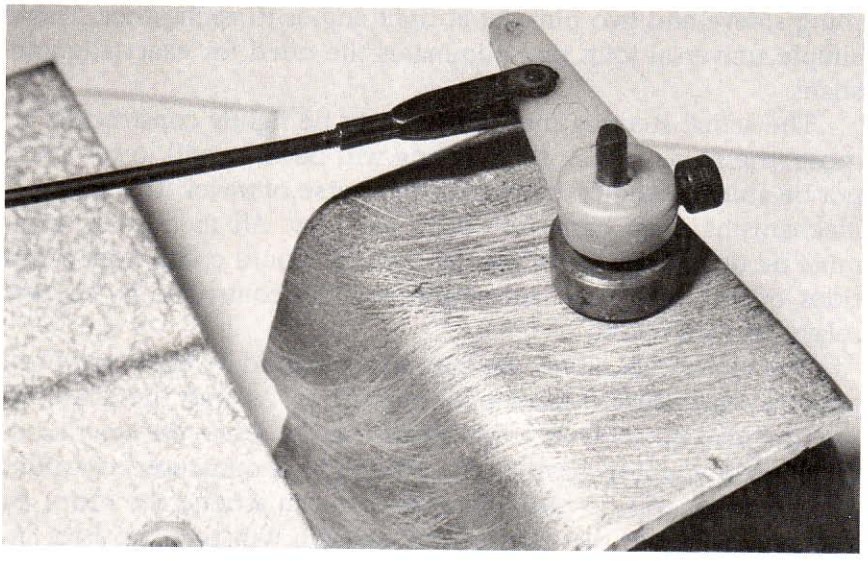


Figure 5.18
The steering arm of the platform. The steering rod can be linked to a lead-screw actuator.

The lead screw is a simple 3/32-inch threaded rod that can be obtained in 36-inch lengths from most hardware stores. This rod can be rotated directly by a small DC motor without expensive gearing. Gears can be added between the drive and the lead screw if higher torque is needed. Since the lead screw provides an increase in torque, this additional gearing is not usually necessary.

The mechanics of the lead-screw method of actuation are uncomplicated, and the materials required are inexpensive. Two tubes slide together to form an extender. The outside tube is stationary, and the inside tube extends. If these tubes are brass, and the outside diameter of the inner tube and the inside diameter of the outer tube are within close tolerances, the sliding is extremely smooth and frictionless. Brass tubing of this type is sold by many hobby shops and hardware stores. Although plastic and aluminum can be used for the extender, their sliding friction is usually much greater and will tend to decrease your available torque.

A brass threaded insert is fitted inside the inner tube's end. I find that if this insert has a *small* amount of play within the tube, operation is much smoother. The enlarged detail in figure 5.16 shows how the threaded insert may be anchored within the tube with a small amount of clearance. This play, if too large, will allow sloppy movement and cut down on the resolution of the extender. The threads of the brass insert should be slightly loose to prevent binding.

The lead screw is coupled to the motor's drive shaft with a small coupling sleeve and two pins set at right angles to each other. This creates a simple universal joint that eliminates the need for exact alignment of all shafts.

The actual steering mechanism must be tightly constructed. If it is too loose, the resolution of the steering will become still lower, and you will not be able to predict the platform's course of travel. The arm and motion mechanisms must have very little end-play. All mechanical connections must be tight. The power for the motors should come from a battery that rides on the platform. The motors will be controlled by transistors and relays detailed below.

The motor that drives the lead screw should be anchored firmly to the platform base since it takes a lot of twisting. As with the drive motor, this one can also be anchored with small U-bolts. Once the steering control is completed, you now have the mechanics for a computer-controlled microbot. This unit will do nothing more than run around the room, but it will demonstrate all of the movements that you will program later in the finished robot.

Don't be surprised, by the way, if even this small beginning unit takes on human characteristics. I've found myself thinking that the little three-wheeled monster knew what it was trying to do. With even simple program commands, the predictable movements seem to lend the little unit the air of having a mind of its own. Don't be concerned if you find yourself feeling emotions toward your creation. Love, hate, and frustration (perhaps especially these latter) are normal reactions.

Testing the Steering

The test for the steering is similar to that for the drive system. A simple hookup to the battery is all that is required at the moment. Try not to let the mechanism oversteer since, if it does oversteer, it will tend to tear itself apart due to the large amount of torque. We will discuss how to prevent this overrun condition when we look at feedback controls.

Electronic Motor Control

As we have seen, practically any device (a lamp or an LED, for instance) can be controlled very well with the proper electronics. The small DC motor is no exception. It is as suited to being turned on and off by the computer as were our lamps. You must use components to control a DC motor, however.

The most efficient control component is the transistor. The 2N2222 is a good high-power, high-current device for handling current. It will be the main electronic switch in our robot's circuitry. It is inexpensive and readily

available. There are even surplus ones available for pennies on the dollar. Try to find these if at all possible. If your motors are small (mine weren't) and if the weight of your robot is light, you could use the 2N2222 as the main control component. Figure 5.19 diagrams a DC motor driven directly from transistors. This circuit works extremely well with a small slot-car motor. If the motor is larger, then some form of relay control or silicon-controlled rectifier (an expensive component) will be required. I chose to use relay control since I had a few relays gathering dust.

Figure 5.20 is a diagram of how relays are inserted into the circuit to handle the larger current. The relays are driven by the 2N2222 transistors

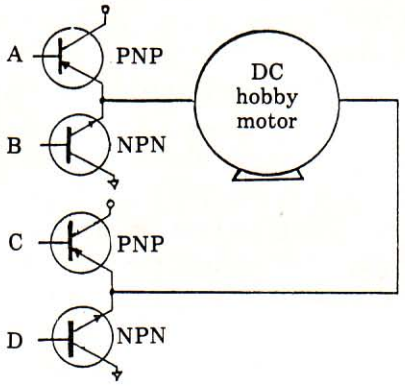
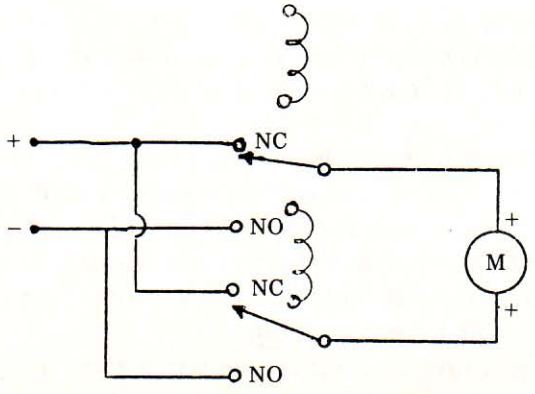


Figure 5.19
A small DC motor driven directly by transistors. The direction is changed by alternating between AD and BC pairs.

Figure 5.20
A typical relay section. Unless one relay is energized, the motor will receive only one polarity.



for more positive control. If your relays are sensitive (mine weren't), the 2N2222s are probably not needed since the relay could be readily closed by the current from the output port. If you find that your relays will not close and open consistently, then add the transistors for more relay drive. Notice that two relays are needed for each motor. In our unit so far, we have two motors, so we will need four relays. One relay turns the motor in one direction while the other handles the reverse current.

To manually test any of the motor control circuits, simply connect the control lead to the negative terminal of the supply to see if the relay or transistor will drive the motor. Later you can have the output port provide the on/off signal.

Figure 5.21 is the wiring diagram of the entire circuit that controls the two motors of the drive mechanics and steering mechanism. Other pairs of relays may be added for other functions. This circuit may be breadboarded and placed on the platform for test purposes. Once it is working, it should be permanently transferred to a wire-wrap board and placed on the platform in its own connector. Notice in figure 5.21 that two 4049 inverting buffers are placed between the 74LS154 and the 2N2222 or relay for each motor. These buffering sections keep electrical interference from getting back into the computer. (Small electrical spikes will not hurt the computer, but they can make it lose its place when a program is running.) I've found that these buffers are not necessary with all motors. If the program works well when your motors are running without buffers, you need not add buffers.

It is important to understand that the 74LS154 decoder shown in figure 5.21 is not the same circuit as the one used in our circuit to produce a device-select pulse.

Programming Test for the Platform

Listed below is a simple test that will program the unit to move forward and backward. A simple touch of the keyboard will make the platform turn right and left. Since you have not yet established feedback, the unit will try to turn right (or left) when you tell it to, even if it is already turned as far right (or left) as possible. So you must avoid instructing it to do something it can't.

The device-select pulse is taken from pin 1 of the 74LS154 decoder we built earlier. This makes the address of the device 192. This simple program is intended only for test purposes. The length of time a command is given to the control is determined by a small programmed loop. Later we will connect the platform to the input port so that the computer can get feedback about what it is doing.

The tethering, or connecting, wires should be about 6 feet long. This length will give the robot enough room to move on the floor for about 2

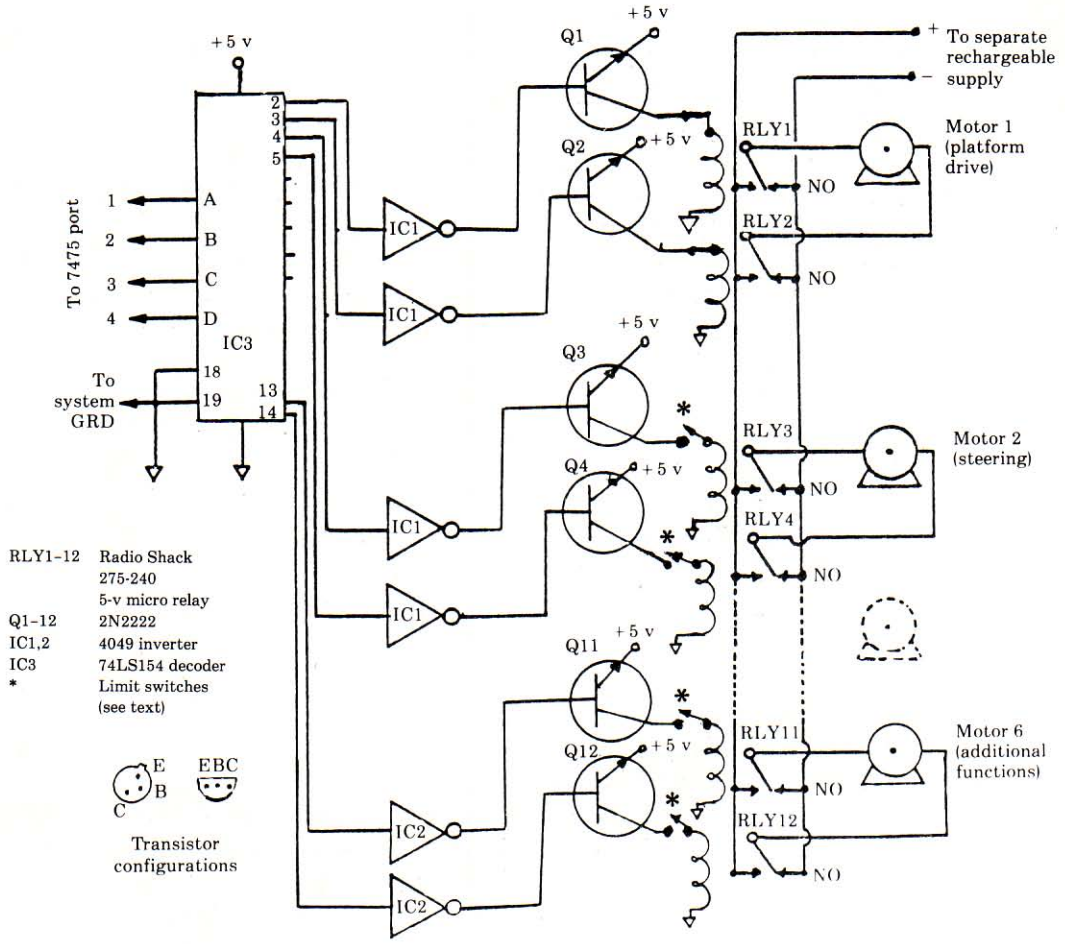


Figure 5.21
A complete diagram of the robotic motor controls. Motor controls 3 through 5 are identical.

feet. Be careful that the unit does not drag your computer onto the floor with it.

Type in the following short routine, and enter RUN. If the program seems to fail, remember that we are now working with both mechanics and software. Either one could be at fault.

Test Program

You have the motor devices mounted and working on the robot's base. If you have tested the motors as previously described and you have them

working, you are now ready to program the robot for movement. I have made most of the programmed movements simple ones in order to keep the tether from becoming entangled. Although the movements are short, they will still give you important information on how your robot is operating.

The tether will carry both movement data and the power for the relays and transistors. The motors should get their power from a separate small heavy-duty battery that is rechargeable. A small motorcycle battery is good if you have motors that match the voltage of the battery. A small 6-volt lantern battery will also work. The tether will need to have six wires—two for the power (from your small power supply or from your breadboard battery) and four for the control data.

The forward signal goes to pin 16 of the 7475 output port. This is the same pin that outputs bit 1 of your data word. Therefore, in order to get the relay to close (and drive the motor for forward movement), a data word of 01 is output. The reverse current is activated by 02 since the matching relay connects to pin 3 of the 74LS154 drive decoder, which is activated by a data word of 02.

A little study of the diagram will show that steering right will be activated by 03 and steering left by 04. Since each function requires a separate input, these are the only four movement commands that would be possible if you were using only the 7475 port. Of course, 00 stops all action; since your mechanical wonder might decide to try to run away from home all of a sudden, you may find this 00 command useful.

The short routine that follows shows how each command can be called with the keyboard. The routine first clears the port and then waits for a command to be entered. Entering an F will (we hope) cause the platform to move forward for about 2 seconds, at which time the platform will (we hope) stop and wait for another instruction. The command R will reverse the platform's direction and cause it to back up for around 2 seconds. Depressing your left or right arrow key will tell it to turn. The turn will continue until you release the key since we have not added limit switches yet, so don't hold the key down very long. Entering S should stop the robot if something goes wrong (this is the command that enters all 0s into the output port).

```
 5  OUT 192,0
10  IF INKEY$ = "" THEN GOTO 10
15  IF INKEY$ = "5" THEN GOTO 80
20  IF INKEY$ = "F" THEN GOTO 45
25  IF INKEY$ = "8" THEN GOTO 85
30  IF INKEY$ = "R" THEN GOTO 70
35  IF INKEY$ = "S" THEN GOTO 90
```

```
40  GOTO 5
45  OUT 192,1
50  FOR N = 1 TO 120: LET A = 100: NEXT N: OUT 192,0
60  GOTO 10
70  OUT 192,2: GOTO 50
80  OUT 192,3: GOTO 50
85  OUT 192,4: GOTO 50
90  OUT 192,0: STOP
```

Of course, the above program assumes that you have connected your 7475 output port to pin 1 of the 74LS154 port decoder to obtain your select pulse for the port at device 192. If you use another pin, you need to change the OUT command number accordingly.

Test Operation

Make certain that the secondary power supply to the decoder and relays is off. If this set of components is powered up first, the port might register an instruction word accidentally, and the poor robot (what there is of it so far) could be off and running, out of control. (If this does ever happen, simply cut the power to the decoder.)

First start the program running, which will clear out the port. Then power up the control components from your secondary supply or battery. The platform should remain motionless until a key is pressed. If any key other than those mentioned is entered, the program will simply keep placing 0s into the port. If only the 7475 port was used for relay control, the robot could possibly try to go forward and backward at the same time. Your robot might then suffer a frustrated convulsion, otherwise known as a burnout!

It is possible, though not too desirable, to have the platform turn and go forward at the same time. Since two separate motors would be called, the strain might be more than the battery could handle unless it is very heavy-duty. Generally, it is good practice to have the device perform only one action at a time.

Although the computer is telling the platform what to do, it is not actually executing a preprogrammed set of instructions. Since you are controlling the computer, most of the action is manually induced. True robotic programming is more than this. If a predetermined action is programmed and the device performs this action after the RUN key is pressed, then true robotic programming is demonstrated. For example, let's have the robotic platform turn itself around. We can easily write a short routine to do this very efficiently. It would consist of *forward, turn, backward, turn,* and *forward.* If the amounts of time allotted to the functions are equal, the

platform should end up where it started (more or less) except that it should have turned itself around. Try the following routine to see how close your unit can get to its original position. Remember to turn on the robot's controls only after starting the program. Then touch any key to start the action.

```
 1  REM PLATFORM TURN PROGRAM
 5  OUT 192,0: IF INKEY$ = "" THEN GOTO 5: OUT 192,1:
    GOSUB 60
10  OUT 192,3: GOSUB 70
20  OUT 192,2: GOSUB 60
30  OUT 192,4: GOSUB 70
40  OUT 192,1: GOSUB 60
50  OUT 192,0: STOP
60  FOR N = 1 TO 400: LET A = 100: NEXT N: RETURN
70  FOR N = 1 TO 200: LET A = 100: NEXT N: RETURN
```

You may think this looks too simple. But remember that the platform is only performing a simple turn. You probably will need to change the 400 and 200 in the FOR . . . NEXT loops to match your gearing. These numbers allow the function to operate for 2 seconds and 1 second, respectively. These lengths of time may be too great or too small for your platform, depending on your design. If the required actions were many, the various functions could be placed in a subroutine of their own, and the main program would be nothing more than a bunch of GOSUBs. You might want to experiment a little with some programming before things get more mechanically complicated in the next section.

The Robotic Arm and Hand

The Arm

Now that you have mastered the art of moving the platform around, you might feel brave enough to go on to something bigger and better: the construction of a working robotic arm. Although it is nice to have a programmable platform scuttling around, a robot needs a few tricks—retrieving an object from the opposite side of the room, for instance. There are so many forces working at the same time in arm mechanics, however, that it is quite difficult to get them all working correctly at the same time in a mechanical arm. So let's start slowly.

First, let's discuss a few concepts and some of the terms we will be using. A good mechanical arm should have several working sections, probably including a section that moves up and down, or *pitches*; a section that

moves out and in, or *extends*; a section that rotates over and back, or *rolls*; and a section that *pivots*. (Also, at the end of the arm should be a mechanism that *grasps*. This is called the *end effector*, or hand, and will be the subject of the next section.) There are various arrangements that allow each of the kinds of movement. Choose the best for your design and the best suited to your available parts.

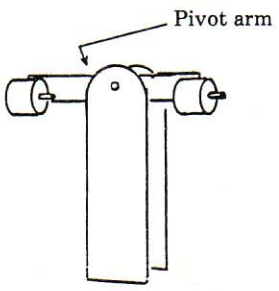
Pitch

Pitch is motion up or down. In robotic design, this movement is usually limited to the robot's upper and lower arm. Some designs allow the wrist to pitch also. Our designs will not get that intricate. If we can design a structure that will raise and lower itself, we will have done well.

There are several methods by which the arm can be made to pitch. Some are better than others. I feel that gearing, again, gives the greatest drive. Drive is important because the arm will have to take a great deal of strain. Most of this strain will be caused by the arm's lifting of its own weight. Therefore, the arm should be as light as possible. Very lightweight materials should be used in the construction of the arm, especially in the lower section. At the same time, the arm must be strong, despite the fact that lightness of weight often means slightness of strength. Aluminum tubing is the lightest and strongest material I've found. Use it whenever possible.

The arm, when extended, places a great amount of strain on the drive mechanics. I find that, for this reason, balancing the drive motors on the front and back of the arm works to decrease some of the weight that you would otherwise get (figure 5.22). The drive motors should be positioned close to the pivot point and not out by the wrist. Imagine the muscles that control your fingers as drive motors. Most of them aren't placed out on your hand. They are further up near the elbow, believe it or not. Captain

Figure 5.22
Actuating motors balanced to lessen the strain on the pivot drive.



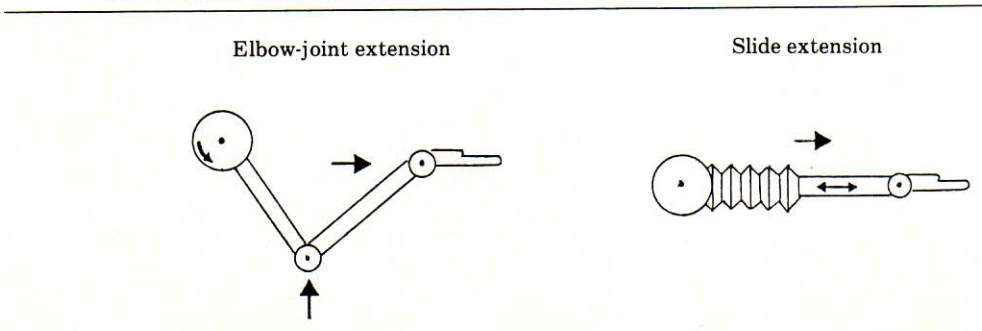
Hook's arm pivots only at the shoulder (for vertical movement). He moves his platform to swing the arm horizontally. This simple arrangement holds the mechanics to a minimum. The gearing is three sets of spur-and-pinion gears. If you use a larger motor, you might get enough torque with only two sets of gears, but the arm will move fast and have a tendency to return to its lower position when power is removed. (The lack of friction on the gears will allow the motor to turn backward because of the weight.) If I need my robot to swing its arm horizontially, I command the platform base to turn.

Extension

If the platform can move forward, arm extension is not absolutely necessary, but I included it on the Captain to demonstrate the extension principle. This one movement works better than all the rest. Figure 5.23 shows two different kinds of arm extension, but they both do basically the same task. Extension gives the arm the ability to reach. It is a very humanlike movement. My microbot's reaching gesture has the look of a gesture by a child. The ability to reach or extend the arm permits the mechanism to grasp objects higher or lower than the pivot of the arm would otherwise allow. Extension can especially help the robot reach objects on the floor (figure 5.24).

Figure 5.23 shows that extension can be provided by either a slide mechanism (my choice for the Captain) or the bending of the elbow and shoulder. The bendable elbow adds a lot of flexibility to the arm, but the mechanics are very complicated. The slide extender (figure 5.25) is actuated by a lag, or lead, screw on a threaded section (figure 5.26)—basically the same arrangement we used in the steering-arm actuator. I firmly believe the lead-screw method can provide complete arm and hand movements. You might want to try this approach since the mechanics are so simple and inexpensive. Threaded rods cost much less than gears!

Figure 5.23
Two methods of arm extension illustrated.



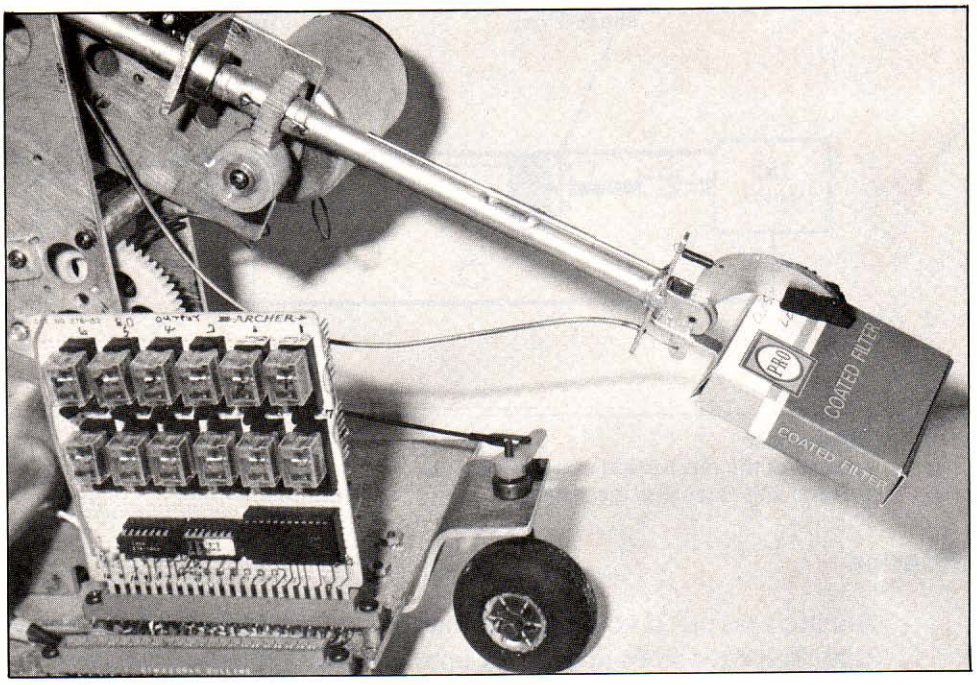
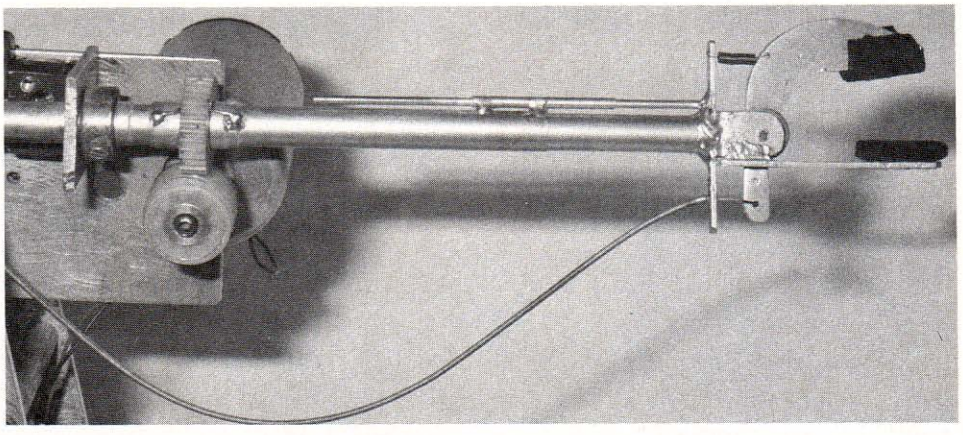


Figure 5.24
The downward extension of the robot's arm. By extending its arm, the robot can reach objects higher or lower than it could otherwise.

Figure 5.25
The extender mechanics. Notice that a small slide is necessary to keep the extender from turning.



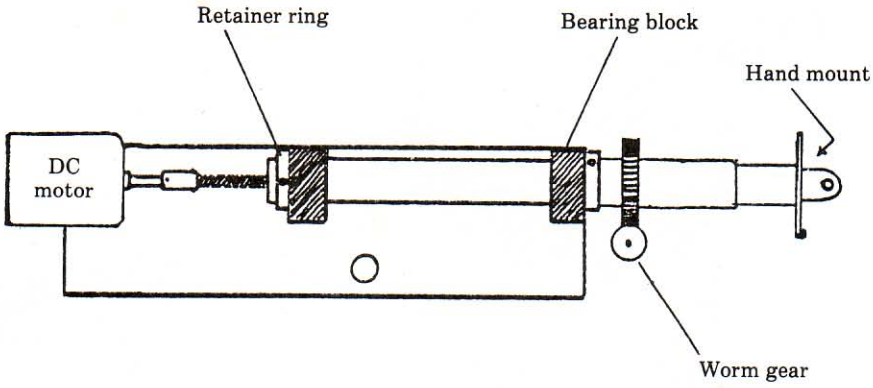
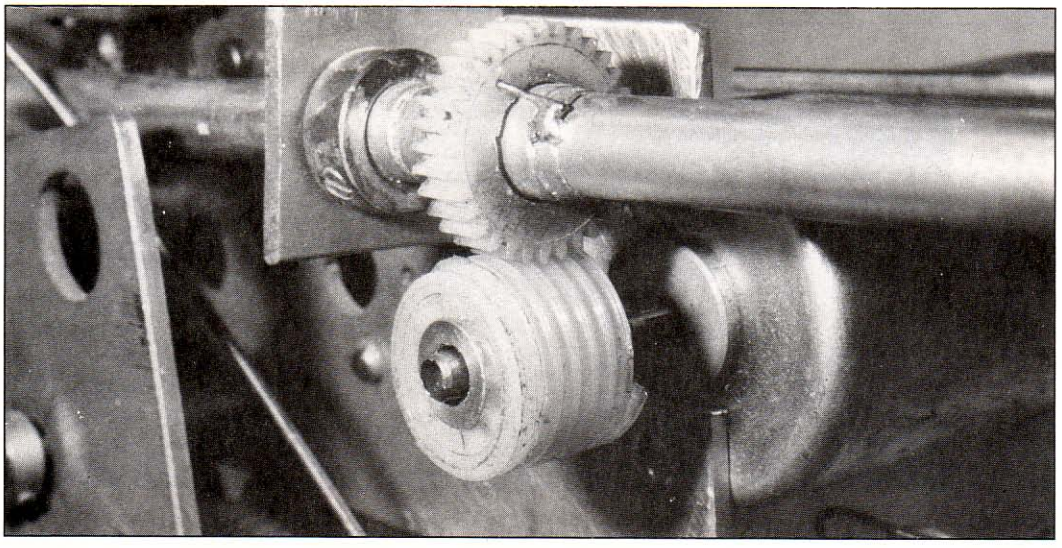


Figure 5.26
A detail of the mechanical arm showing the rotation and extension mechanics.

Rotation

Rotation is roll. If you roll your hand, the forearm actually causes the rotation. The wrist does not roll, at least not with respect to the bones to which it connects. Roll is one of the most difficult movements to design a robotic arm to accomplish. In figure 5.27, you can see the worm gear and spur that rotate Captain Hook's forearm. Forearm rotation allows the hand to rotate

Figure 5.27
A view of the worm gear that rotates the extender.



an object in its grasp. It also allows the hand to grasp an object that it could not grip if its hand stayed in one fixed plane.

Figure 5.28 shows the worm-gear mechanics that can be used for arm rotation. The simplicity of the design is apparent. As the DC motor turns, the arm will rotate 360 degrees and more if the proper limit switches have not been added. We'll talk more about these switches in the feedback section. Although I used a worm drive, you could use a simple spur-and-pinion gear just as easily. Worm gears are a little more compact than pinion gears, though. Figure 5.25 shows the complete arm, with the spur and worm gears fitted to the arm's primary tubing. Notice that some form of fixed slide is required to keep the forearm or secondary tubing from turning also. Several methods could be used to prevent this rotation of the forearm, but I used two small brass tubes as the fixed slide.

Pivoting

Figure 5.29 is a diagram of the gearing for the pivot control. This is not the simplest arrangement, but the arm has to lift a lot of weight. I felt that

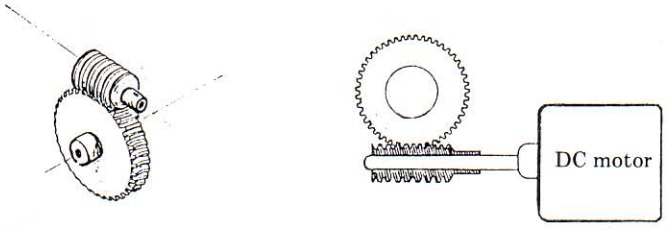


Figure 5.28
Two views of the worm-gear drive, one of them also showing the DC motor and motor shaft.

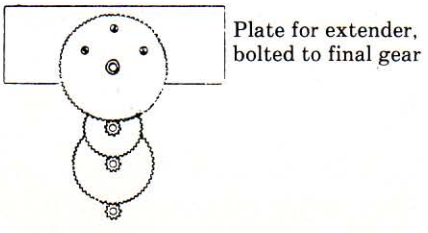


Figure 5.29
The gear train of the robot arm pivot. The three-stage gearing is needed because of the weight of the arm.

gearing would slow the rotary (motor-produced) motion down to a snail's pace and thus produce more torque. I was correct, but working out the mechanics took a long time.

The pivot gearing is a three-stage drive that was, frankly, a nightmare to align and construct. The gears had to be mounted with adjustable bearing blocks that allowed slight adjustments to be made to the gears for better tooth mesh. The final spur gear is bolted to the arm's frame so that the complete arm can be pivoted upward and downward. Again, I think that the lead-screw, or threaded-rod, method would have worked just as well, without all of the hassle.

All shafts are ¼-inch, and none have bearings inserted. The lack of bearings seems to cause no problem. Each shaft is retained by retaining rings with set screws. Each spur gear has a retaining screw to keep it in line with the other gears. Notice that there are three small pinion gears that drive the larger spurs. With this many stages, the arm can lift several pounds, as long as it has a fairly large motor.

I used solid aluminum sheet for the frame for the pivot mechanics, but the result was much heavier than I had hoped. If Captain Hook's pivot-gear housing and arm frame had been made of aluminum angle pieces, I believe I could have lessened their weight drastically. (As you can tell, most of the design and construction of Captain Hook was simply a large experiment.)

The Hand

The robot hand, or manipulator, is one part of the machine that you can design endlessly without exhausting all possibilities. There are many good designs, but none of them is completely satisfactory for every application. The human hand is undoubtedly more versatile than any grasping mechanism ever designed. But several designers have successfully designed robot hands that almost duplicate the movement of human fingers. One inventive designer has even designed a robot hand that plays the piano. (He is very secretive about how he did it, though!)

Most gripping mechanisms are task-oriented (that is, designed for a specific task and no other). A hand designed to lift radioactive rods would probably be useless for manipulating packing crates. The two hand designs would be completely different. Many designers are now creating gripping mechanisms that are removable. If the job changes, the hand changes.

General Hand Design

Though it is very difficult, we will design a mechanical hand that picks up most small objects. I wanted one that could wash my car, but this was too much to ask. I chose instead to build a much more simpleminded two-

fingered grasping device that could simply open and close on command. The final result is a *pincer* (figure 5.30).

Figure 5.31 shows four different types of hand design. Some robotic hands are complicated, and others are simple. The simple ones seem to work best, but they are not the most versatile when it comes to object

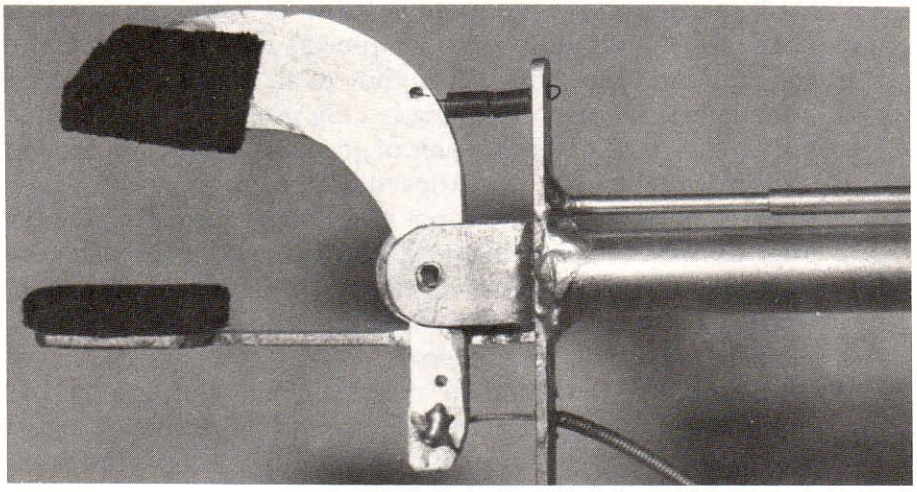
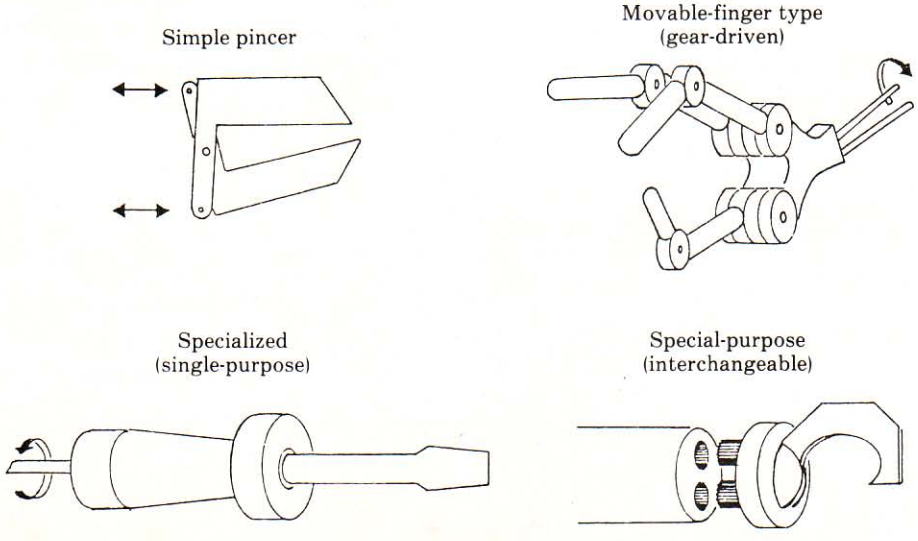


Figure 5.30
The pincer of the completed arm.

Figure 5.31
Four types of manipulator.



manipulation. Can you imagine trying to pick up a 2-inch steel ball with the pincer arrangement? Captain Hook is still trying to do that one successfully.

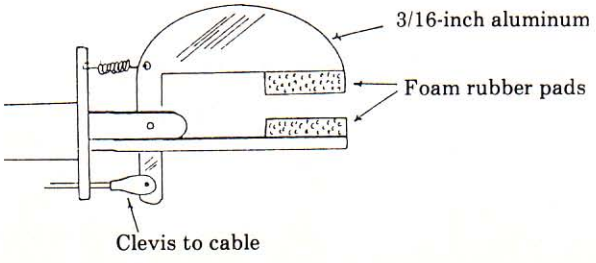
Finger Pads

Since the pincer arrangement will be our choice for now, we will need to make the pincers a little more versatile by adding finger pads. These are foam-rubber pads glued onto the pincers for better grip. The foam rubber will conform to the irregular outline of an object to keep it from slipping from the robot's grasp. Slipping is a big problem because most objects are not balanced. An object's center of gravity is usually not located at its physical center. When the robot tries to lift the object, gravity causes the object to twist. Since by then the robot is no longer increasing the pressure of its grip, the object generally slips. The result is usually the loud *cluck* of impact as the fallen object meets the floor. Captain Hook's first task was simply to pick up a screwdriver. He repeatedly dropped it until I added the pressure pads to his two fingers.

Figure 5.32 is a complete diagram of the grasping mechanism on my unit. Though not a perfect arrangement, the pincers do operate well, for the most part. The pincers are returned to the open position by a small spring, which works against (but does not seem to interfere seriously with) the gripping mechanics. Purists may object to this arrangement, but I was glad to find something that worked.

The actuating force is provided by a small servomotor with the electronics removed. You could as easily use the lead-screw method, but I wanted to control the pincer with a cable, and the servo was handy. The servo is placed near the arm's pivot point. Most designers insist on having the motor near the hand, for some reason. To my way of thinking, that would only increase the weight of the arm. It is even possible to have all of the arm's actuating motors located on the platform base. Radio Shack's

Figure 5.32
A construction detail of the pincer.



Armatron (figure 5.4) uses this method very successfully. Its arm is very light. The mechanics are somewhat more complicated, however.

You can now see why arm rotation is important. Any grasper has to be oriented properly before it can grasp an object. Pincers lifting a small upright bottle by its neck must be oriented horizontally. The pincers must be vertical, however, to grasp a screwdriver lying on a workbench.

The hand can be the most creative part of the entire robot. A realistic hand can make the robot very impressive. If, for example, the hand has five movable digits, simply covering the hand with a rubber glove makes the hand movements look almost human. Whatever the final design, the actuating force must be strong, but not strong enough to crush the object that has been grasped. Larger robots have a habit of mashing things. If your unit is large, some form of pressure-sensing feedback is really necessary to keep the applied pressure of the grip from being too great. Since my small servo could produce only very little pressure, I chose to simply let it apply its full force until it loaded down.

Captain Hook's hand is made from aluminum and brass. The fixed brass portion is soldered to the rod, or forearm. If a rigid cable is used, the return spring is not necessary. My cable was too flexible to push the pincer back to the open position, so I added the small spring. The slack in the actuating cable lets the forearm extend outward some 8 or 9 inches without interfering with the hand control.

Arm and Hand Electronics

The last main section dealt to some extent with the electronics necessary for computer control of our little metal marvel. The arm mechanics and associated electronics are simply extensions of those required for the platform.

In the beginning, there were only two motors (on the platform base) with which to work and which you could program. By now there are six motors, a larger number but not the largest number possible. The electronics required to control many motors at once, however, can quickly get unmanageable. If we control the motors as we did those of the base and steering, we will need six sets of controls.

You could have all of the motors running at once, but the battery would have to be rather strong, to say the least. I feel that, since the computer can think about only one thing at a time, the motors themselves can appropriately be run only one at a time, too. This keeps the strain off of the battery and makes it easier for you to program and avoid mistakes.

The 74LS154 Decoder

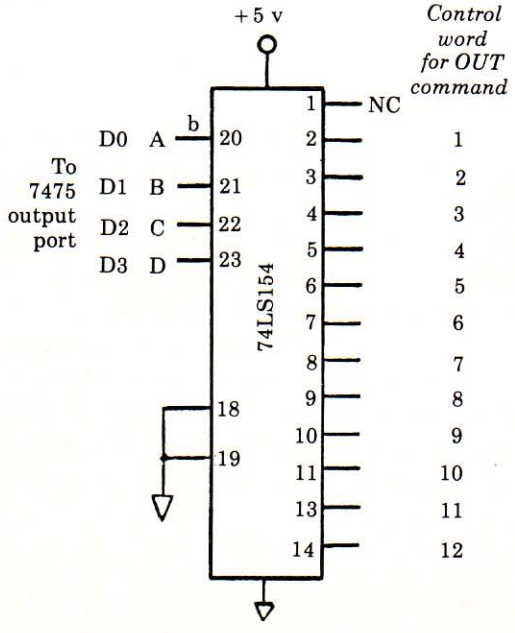
If we use only the 7475 output port for control, there can be only eight actions that the robot can perform without some means of decoding the

signals from the port. But if something can decode the data presented at the 7475 output port, the total number of possible robot actions could be as high as 255. The 74LS154 decoder, again, comes to the rescue!

We first used this decoder to generate a pulse to activate a device. We also used this component to generate a signal to each of the twelve different transistors or relay drivers. Figure 5.33 shows how each of the twelve relays is connected. The 74LS154 is wired to produce a drive signal depending on the binary data that is made available to it. For instance, 1 means *forward,* and 2 means *reverse.* The 74LS154 simply decodes the information and outputs it to the various transistors. Using the 74LS154, we can generate up to fifteen different control signals. Notice that each line is marked with the number that represents its activation data.

It is impossible to have the robot do more than one thing at a time with this component. You'll find that the programming will be easier and that commands to stop or reverse will be less likely to be lost if you ask the robot to do only one thing at a time. Moreover, when we add our feedback controls, you'll see that a feedback signal is easily lost if the computer is being called upon to do other chores while counting incoming data.

Figure 5.33
The function decoder for the complete robot. Note the data that is used for the various functions. The transistor relays connect to pins 2 through 11 and pins 13 and 14.

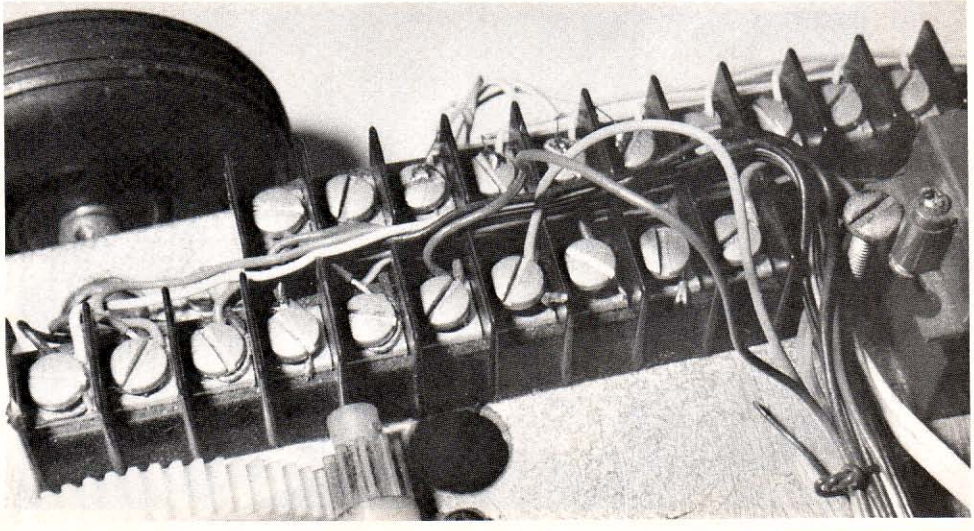


After constructing the arm mechanics, testing them, and adding them to the base, wire-wrap the circuit shown in figure 5.21, and add it to your mechanism by inserting it on the platform. The control wires running to the robot will attach to a connector block as shown in figure 5.34. Ribbon cable is used for the connector since there will be fourteen leads from the computer's port to the robot. This cable will be its tether. It should be about 10 feet in length. A longer cable is apt to give the robot a case of glitches. Try to keep the cable from entangling itself under the robot as the robot makes several turns. This ribbon is used mainly for demonstration purposes and would not allow the robot to wander about the house.

Notice the two separate power supplies in the circuit shown by figure 5.21. One 6-volt supply is used entirely for the power source for the three ICs on the wire-wrap board that will mount on the platform. This small supply can be four small C batteries connected in series. The three components will not eat too much current, and this supply should last indefinitely. Make sure that an on/off switch is included that will cut this power pack off. Notice that the ground connection of only the small supply goes back to the ground connection on the computer. This keeps the reference point of the voltage the same for both units.

The other supply is your heavy-duty 6-volt battery that powers the motors. If your motors are small, then the battery can be smaller. The reason for this second supply is to reduce interference from the motors. When a motor is running, it produces spikes in the power lines that drive

Figure 5.34
The connector block of the platform. The block keeps the many wires in order.

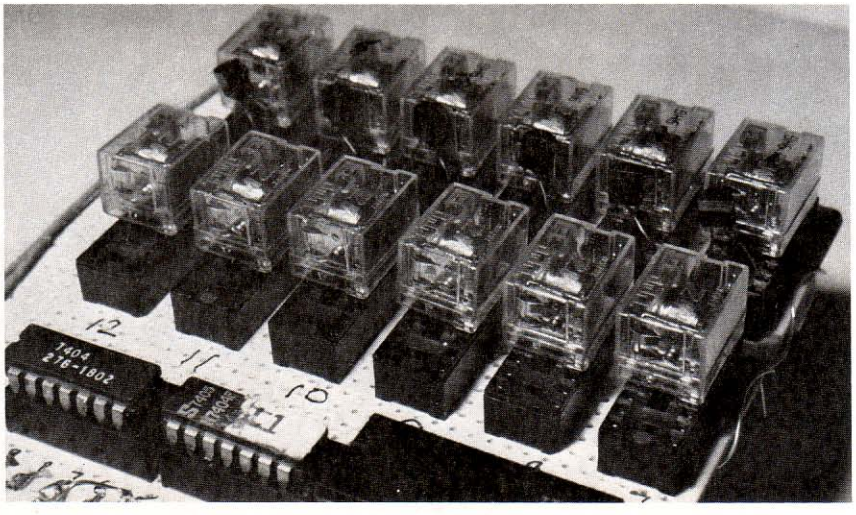


all of the digital ICs nuts. With the separate supply, this problem does not occur.

The relay board on the completed platform is shown in figure 5.35. The 2N2222 transistors are activated (to close the relays) by a positive voltage from your control circuit. Since we are using the 74LS154 as a function decoder, the voltage produced when a certain motor function is called is negative. In order to get the right polarity, use 4049 hex inverters. I suppose you could avoid having to use the inverters by using PNP transistors, but these do not dissipate enough heat unless you pay dearly for them. The 4049s are inexpensive, and the 2N2222s are plentiful. Use them if possible.

In the diagram (figure 5.21), the motors are connected to the relays' power connections only. The computer control is connected only to the relays' coils. If your motors are large, so should the relays be; that is, the relays should be able to handle the amount of current that the motors will pull. If you plan to make your drives out of motorcycle starters, for instance, the relays should be quite large, as should the battery and power wiring. If you do use large relays, the relay coils should have a despiking diode placed across them, as shown in figure 5.36. Whenever a coil is de-energized, it produces a spike of voltage that is fed back through the circuit. If you are using very small 5-volt relays, this may not be a problem. Otherwise, the spike may be large enough to do some damage.

Figure 5.35
The relay board on the completed platform. Notice the transistors that drive the relays.



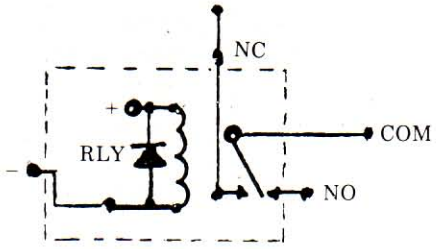


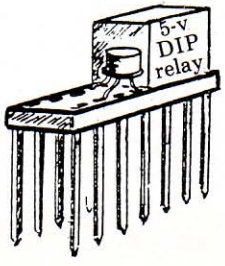
Figure 5.36
A despiking diode across coils on a heavy-current relay.

Figure 5.21 depicts only three of the required six motors, and their associated control circuits. The dots tell you to build the other three in the same way. Although the circuit looks complicated, it really is simple, except that there are six repeats of the same layout. I suggest that you build one motor circuit at a time, and test it, before wire-wrapping all six of the controls. This way, you might be able to catch a mistake before you have spent much time and trouble.

Component Carriers

I was lucky to find small 5-volt relays that were DIP-type (that is, of the sort that fits into an IC socket). I chose to insert the relay into a sixteen-pin wire-wrap socket and wire-wrap to the posts. Also, I found that I had several unused socket connections, and I used them to insert the transistor driver. Figure 5.37 shows these components in place in the socket. I found the circuit board much neater and easier to wire than it would otherwise have been. This carrier method can be used for wire-wrapping almost any

Figure 5.37
A relay and a driver transistor mounted on a wire-wrap socket. This arrangement eases component replacement.



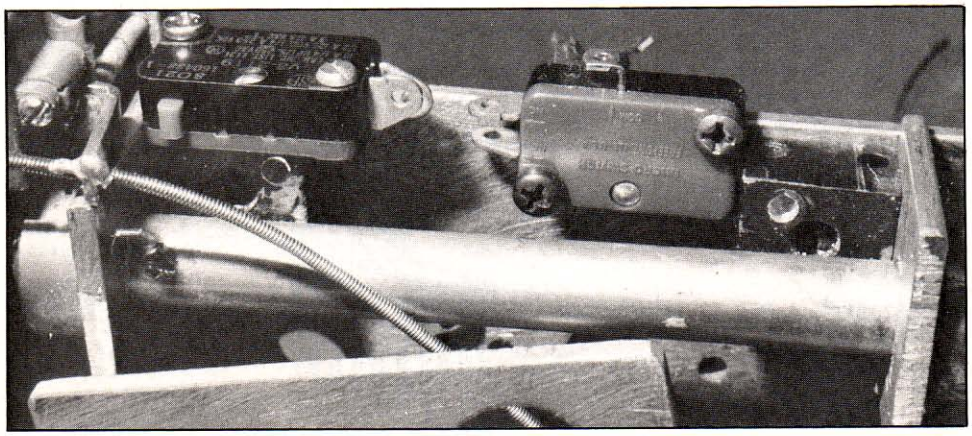
type of component that will fit in the socket. All components used in the finished circuit are available from Radio Shack, and many can be ordered from surplus dealers whose addresses are shown in Appendix A.

Limit Switches

Like most mechanical devices, this device needs some form of limit control to keep it from tearing itself up if something goes haywire. Note that on the relay coil lead there are two points marked with asterisks (figure 5.21). These are the places where limit switches can be inserted. A limit switch is a simple disconnect switch that will disconnect the motor from power if the mechanism moves past a certain point. If the robot were allowed to extend its arm without limit, for example, it would simply lose its forearm. Of course, too, if your motors are large, the mechanism can actually tear itself up or burn out one of the motors if a limit switch is not in the circuit. Figure 5.38 shows simple microswitches used to control the extender. If the arm moves to a specific point, the power is simply cut off from the relay that is commanding the arm to extend itself. Notice that only one relay is cut off at a time. This allows the computer to retract the extender and bring the mechanism back to center. Microswitches are also used in robotic feedback circuits, as detailed in the next main section.

Limit switches are critical little single-pole, single-throw cutoff switches that are activated by the movement of the mechanics of the robot. When open, these limiters deactivate the drive power, cutting off all movement caused by that drive power. Since each drive is opposed by a drive that moves the mechanism in the other direction, the cutting off of one drive by a limit switch still allows the reverse drive to operate. Thus, your unit does not get stuck in one position.

Figure 5.38
Limit switches that control the extender.



Limit switches are not feedback circuits. They are not hooked to the computer in any way. They simply cut power if the control is moved too far. You'll see that they are really safety devices since they rarely operate unless there is a miscalculation in the program. If a limit switch is open, the computer will not know it unless the first startup commands are commands that position the mechanisms against their limits in one certain direction. For instance, the extender should be brought all the way in as the first movement that the robot performs with the extender. This initial setup procedure is highly recommended since we will not cover the highly complicated mechanisms necessary to give the computer feedback as to the relative static positions of the robot's own parts.

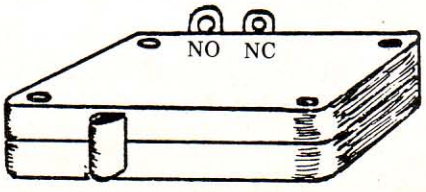
Most mechanisms have two limits and need two switches, one for each direction. But not all mechanisms need these safety switches. If the mechanism is rotary (such as is the platform drive), there is no limit, and a stop is not required. The devices without limiters will depend entirely on the feedback circuits for their correct positioning.

Figure 5.39 depicts the limit switch most often used. It is called a *microswitch* since it is small, but it can handle large amounts of current. It is expensive for its size, but I can't find a switch that doesn't cost a fortune. I've tried homemade switches, but they cause more trouble than they are worth. Several surplus supply houses sell microswitches for a lot less than you can get them for locally.

Adjusting the Limit Switches Adjusting limit switches is a real challenge. You must get them aligned correctly and position them properly on the mechanism. Even then, they may not be right because of a little lag in the mechanics. Once they have been adjusted, they must be extremely secure, or readjustment will constantly be necessary.

Since the most critical stop switch is the one on the rotator of the arm, let's talk about how it is adjusted. The photo in figure 5.38 shows the two switches and how they are activated. The microswitch has a very light touch, so almost any pressure will trigger it. The small flexible reeds soldered to the rotating section touch the microswitches mounted securely on

Figure 5.39
A typical microswitch.



the nonrotating portion. Each reed must be flexible so that it will bend slightly when the arm continues to rotate a bit even after the switch is opened. Remember, a microswitch should be stationary, and the actuator should be the part that moves. Adjustments are a lot easier if this is the arrangement.

Install the microswitches as shown (figures 5.21 and 5.38), and enter the following program to ensure that they will work correctly. The transistor must be controlled as shown in figure 5.33 in order for the program to work correctly.

Testing the Limit Switches Always test a section of the circuit before going on to the next section so that you know that you understand the details of the circuit's operation. This is a good rule of thumb even with respect to the uncomplicated limit switches.

Make certain that the relay driver is connected to the robot's 74LS154 function decoder as shown in figure 5.33. Then enter and run the following. This small program will rotate the arm for 5 seconds. This is more than enough to activate the small microswitch. The arm should rotate until the microswitch is opened, at which time the arm should stop. (If the switch has not been adjusted properly, and the reed misses the button, simply cut the power to the relays.) Once the arm has been stopped by the limit switch, it will pause slightly and then rotate in the other direction. This will continue until you press the BREAK key. The program to test the microswitch is as follows:

```
10  OUT 192,9
20  PAUSE 300
30  OUT 192,10
40  PAUSE 300
50  GOTO 10
```

Of course, limit-switches should be applied to the circuits of the rest of the mechanical functions that should be limited in their travel. Try never to let a mechanism hit a point that chokes the motor down while it is under power. If this occurs, several things could result, including chewed-up gear teeth, burned-out motors, and dead batteries. Most of your mechanisms will probably not have the power to tear up the gear teeth, but the strain on the motors is not good. Moreover, if the arm or whatever chokes itself down because of hitting its limit, it has a tendency to jam the gears so that opposite movement will be almost impossible to activate without a rather strong motor.

Robotic Feedback

Now that we have a robot that can go almost anywhere and do almost anything it is told, we almost have a complete working model. But we have to watch its every movement in order to tell the functions when to stop. It is not yet a full-fledged robot. True robots take orders and carry them out unattended, because true robots use feedback controls.

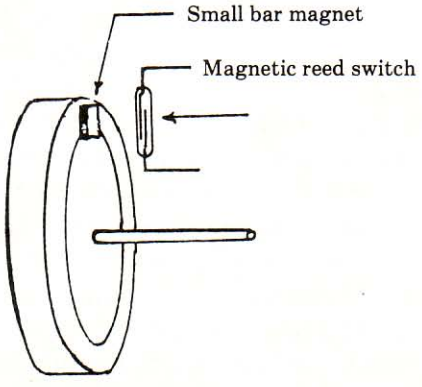
Feedback, as we have said, is the ability to sense something and respond appropriately to it. The small switch on the door that warns of intruders is a good example. Why not put several of these switches on our robot and have it respond to the signals that they produce? That way, the robot can tell if it has crashed into a wall, gone a certain distance, or pivoted its arm a specified number of degrees. We can even program the robot to scan the signal inputs on the 8212 port and make appropriate responses.

There are several means of providing feedback to a robotic mechanism, including switches, photocells, magnets (figure 5.40), reed switches, and even rotary potentiometers (figure 5.41). Let's explore the use of several of these devices and try one or two on our unit.

Using Microswitches

Figure 5.42 shows how a microswitch can also be used as an environmental sensor. By mounting four microswitches around your robot's perimeter,

Figure 5.40
A magnetic reed switch actuated by a small bar magnet. The magnet can be glued to a wheel or gear.



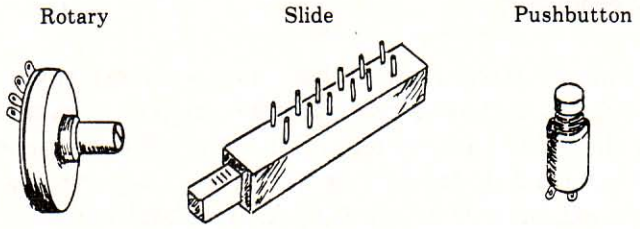
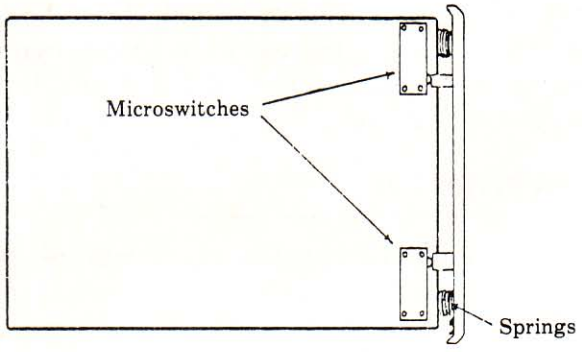


Figure 5.41
Three types of feedback switch. The rotary and slide switches can provide some positional feedback information.

Figure 5.42
A detail of a front bumper with microswitches.



you create sensitive bumpers. If the switch on the right front is closed, a program can turn the steering left 90 degrees, back the unit up for a certain distance, turn the unit straight, and give the *forward* command. Our robot can effectively navigate the entire room if these bumpers and a longer tether are added.

Microswitches can be used as sensors in other ways. For example, a microswitch can be mounted near an axle fitted with a small cam as shown in figure 5.43. Each time the cam passes the microswitch, it closes the switch and provides a signal to the sensing input port. A program then can tell the base to move for a certain number of closures rather than for so many seconds. This would be a tremendous improvement in telling the robot how much of a function to carry out.

A wiper switch can also let the robot know how long to perform a function. Figure 5.44 shows a round disk that can be etched from copper-clad board and mounted on a wheel or gear. A signal can be produced by two stationary wipers riding on the robot's base. Each time the wheel rolls

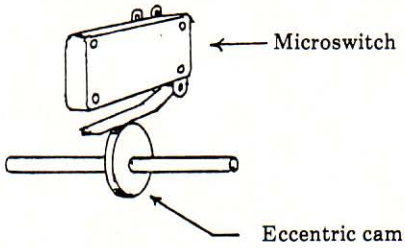
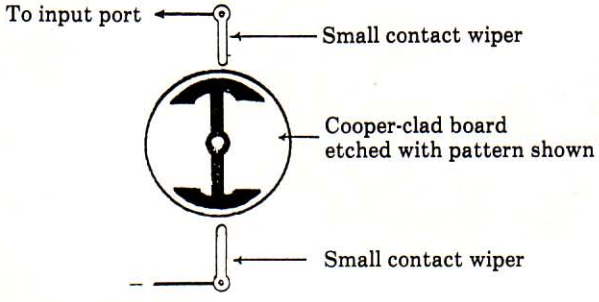


Figure 5.43
A small microswitch actuated by a cam fitted on an axle. The switch can count revolutions.

Figure 5.44
The wiper assembly that detects motion of the platform.



one-half of its circumference, a sensing signal is sent back to the port for interpretation.

Using Light for Sensing Distance

One of the most fascinating of all components is the phototransistor. This is a transistor that is activated by light. It acts much as does our 2N2222. But, rather than sending a control signal to its base, we shine a light on it. The light can come from a small bulb or a clear LED. When the light is received, the transistor conducts.

It is quite possible to build a small distance indicator with an LED and a phototransistor as shown in the schematic and diagram of figures 5.45 and 5.46.

Most gears have perforations in them, and you can use these holes to make the phototransistor work. Simply place an LED on one side of a perforated gear and a phototransistor on the other side (figure 5.47). Each time a hole passes by the LED, the transistor will indicate a signal. If none of your gears has perforations, cut a disk from a thin piece of aluminum,

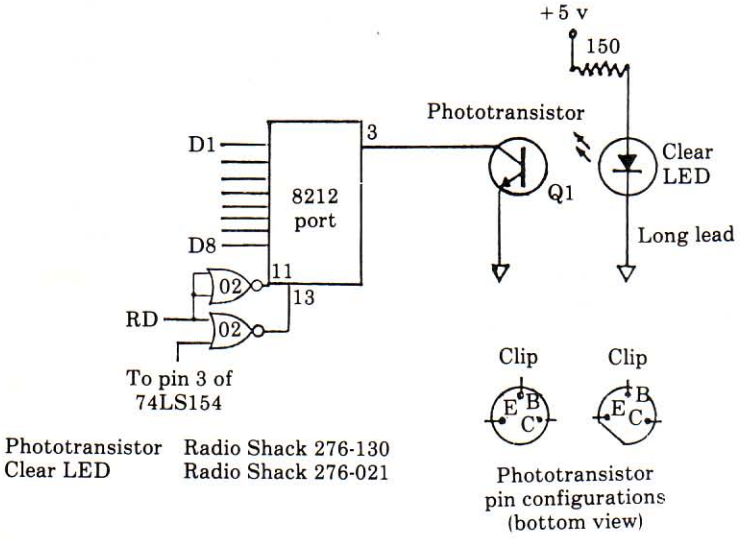
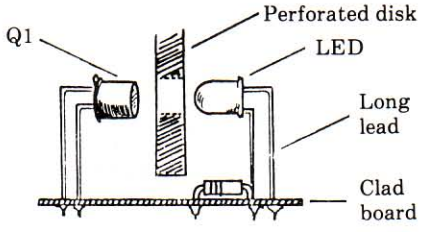


Figure 5.45
Diagram of a feedback circuit with a phototransistor and a clear LED. The transistor pinouts depict two types of transistors supplied by Radio Shack with the same part number. The input port is wired as usual.

Figure 5.46
A detail of a phototransistor sensing circuit. The perforations can be in a gear or a gear-driven disk. Place the two components as close together as possible for the best light reception.



and drill holes around the perimeter as shown in figure 5.48. This can then be attached to a shaft or gear.

Notice in the schematic (figure 5.45) that the 8212 input port is connected to pin 3 of the 74LS154 decoder. This lets the device-select number become 193. This way, the 7475 output port that is providing data to the function decoder can be called by 192 since it is connected to pin 1. You

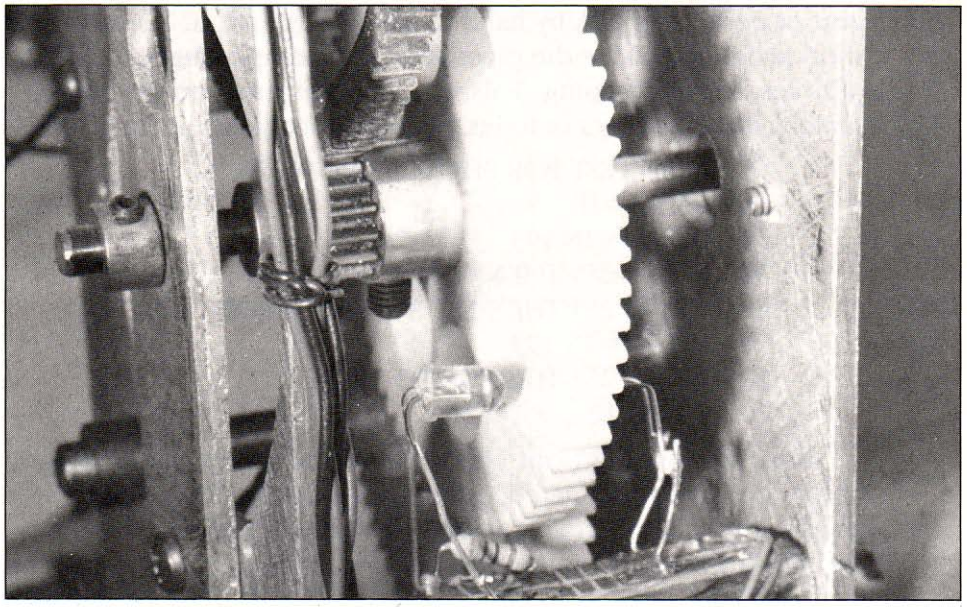


Figure 5.47
The phototransistor-LED assembly on the perforated gearing of the pivot mechanics.

Figure 5.48
A perforated disk cut from thin aluminum.

might want to breadboard the circuit shown and test it until you get it working to your liking before building the perforation sensor. The base lead on the phototransistor is to be clipped off since it is not needed.

Once the circuit is complete, the following small program will test the sensor to indicate proper operation. This simple program looks at the input port and assigns a value to the variable X. If the sensor shows X to be 254, then the phototransistor is reading a hole. If the value is 255, the sensor is between holes. This program will count the holes as they pass

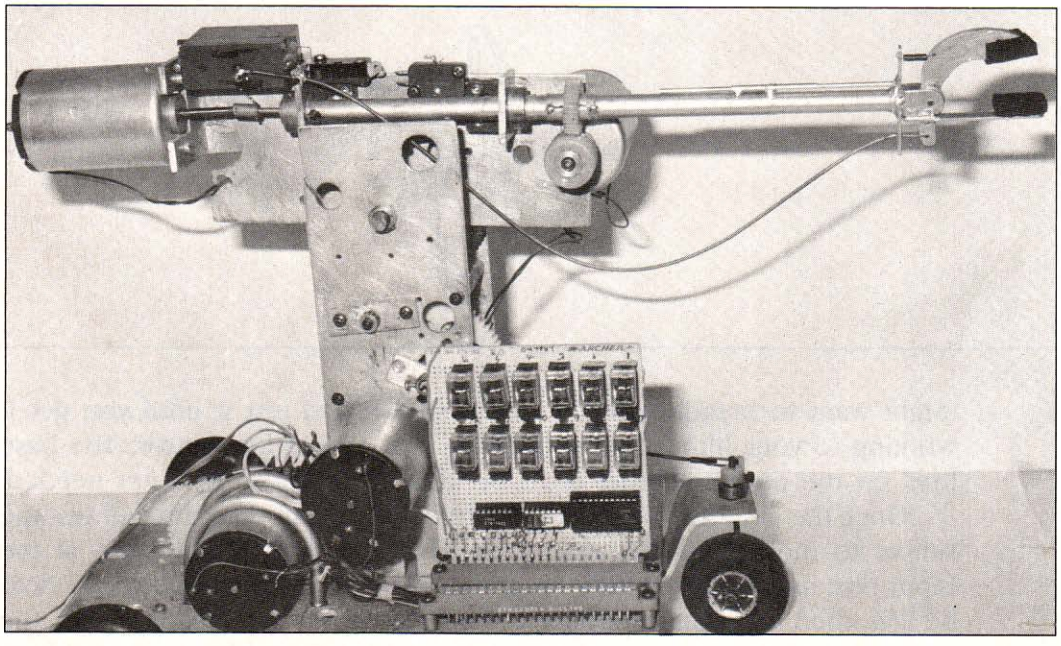
the transistor and print the number of holes that pass. You can turn the gear or perforated disk by hand to test the operation. Then you can write a short program to have the pivot gear, for example, move a certain number of holes before stopping. This way, you can program the arm to pivot up *some, a lot,* or *all.* Try not to let the pivot encounter the limit switch.

```
 5  REM "TEST FOR PHOTOTRANSISTOR"
10  LET A = 0
20  LET X = IN 193
30  IF X = 255 THEN GOTO 20
40  IF X = 254 THEN LET A = A + 1: PRINT A
50  LET X = IN 193
60  IF X = 254 THEN GOTO 50
70  GOTO 10
```

Since the program can check the status of the port very quickly, the holes could be miscounted (one hole could let light through for several checks). But the program contains two short loops that hold the program up until the number on the port changes. A simple program like this could also check the operation of the microswitches and magnetic reed switches. You can see how a short loop can be made to control the motion of the robot to a great extent.

Figure 5.49, by the way, shows my pride and joy, Captain Hook.

Figure 5.49
The completed robot.



6

Advanced Feedback Projects

Most interfacing projects involve feedback circuits. The circuits can feed information back to either you or the computer. The projects described in this chapter involve feedback to both of you about what is happening outside of the computer.

We will experiment with a small speech synthesizer that will provide direct information to you and with several smaller circuits that sample the environment. Each of these projects is presented in a way that I think you will find very easy to follow.

The Speech Synthesizer for the Robot

What good is a robot without its chirps, beeps, and whistles? If you added sound to your creation, you could get your robot to put on a really impressive show. The robot could not only obey your command but also inform you when a task is complete, call you for help, or even tell you the time. This may sound complicated; it isn't. With the inexpensive speech processor and speech ROM sold by Radio Shack and a simple circuit, your robot can do these things and more.

The circuit itself is really quite simple. If you don't choose to build the robot, the voice synthesizer can be hooked into one of the other sensing circuits and programmed to alert you to that circuit's condition. One circuit could be a talking alarm clock. Another could be a talking thermometer. The circuit could be added to one of the many BASIC games, added to your security system, or even programmed to instruct youngsters in counting and

arithmetic. Nothing seems to be completely out of the question with a few components, a small speech processor, and the Timex/Sinclair 2068.

The secret, of course, is in the programming. I'll not bore you with page after page of listings, but I will show you a couple of programs that will familiarize you with the basic techniques. The rest will be up to you.

The SPO256 Speech Processor

Radio Shack and other outlets are offering a complete speech-processor chip set that is both inexpensive and easy to assemble into a working speech module. Figure 6.1 shows the completed module built on a small piece of perf board by the wire-wrap technique. The performance of the assembled unit is very impressive, considering its size and small number of components. The components and layout are not critical. Most of the components can merely approximate the values shown on the diagram presented in figure 6.2

The SPO256 speech IC and its companion ROM, the SPR16, are sold together to ensure that the complete module will perform its intended function (to act as a talking clock). What is so great is that the vocabulary of the serial ROM is separated into thirty-seven different words and three separate melodies. These can be used in any combination for various so-called sentences. A sentence might be a string of numbers meaningful only to the operator, but the outcome is the same, intelligent speech. Besides being able to speak the words in the supplied ROM, the processor also has several noise-generating capabilities. Both the words and the noises are listed in figure 6.3. You can expand the module by an additional ROM with

Figure 6.1
The completed speech module.

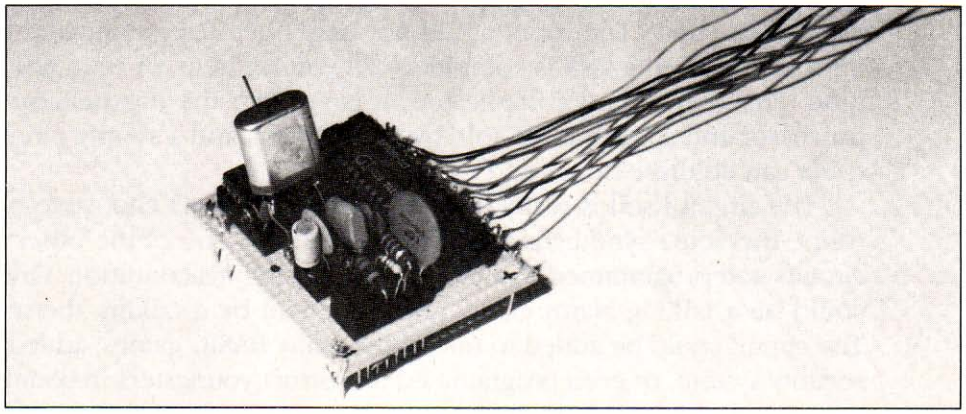
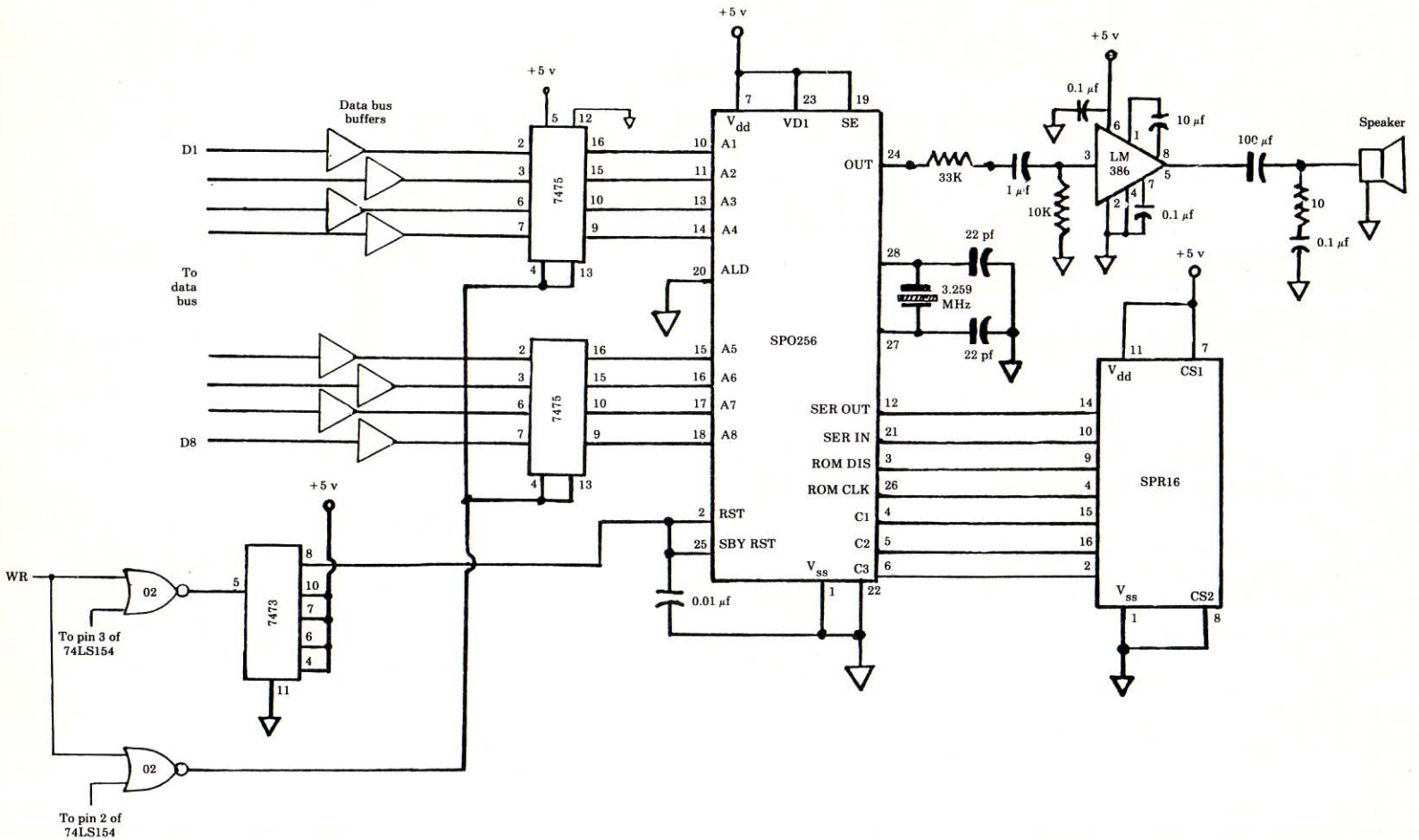


Figure 6.2
A complete diagram of the speech synthesizer circuit. The 7475 output port is connected as usual.



Data	*Sound*	*Data*	*Sound*
0	Oh	24	It is
1	One	25	A.M.
2	Two	26	P.M.
3	Three	27	Hour
4	Four	28	Minute
5	Five	29	Hundred hour
6	Six	30	Good morning
7	Seven	31	Attention please
8	Eight	32	Please hurry
9	Nine	33	[Melody A]
10	Ten	34	[Melody B]
11	Eleven	35	[Melody C]
12	Twelve	40	[Whine]
13	Thirteen	41	[Chirp A]
14	Fourteen	52	[Low whine]
15	Fifteen	64	[Bounce, crash]
16	Sixteen	71	[Zap]
17	Seventeen	76	[Chirp B]
18	Eighteen	77	[Quick crash]
19	Nineteen	86	[Hiss]
20	Twenty	136	[Laser sound]
21	Thirty	138	[Two chirps]
22	Forty	245	[Motor running]
23	Fifty		

Figure 6.3
Words and other sounds that can be produced by data stored in the speech ROM.

more words if you want the speech synthesizer to have a larger vocabulary. Radio Shack plans to offer an SPO256-AL2 ROM that will replace the SPO256. This new ROM will have sixty speech syllables with which to work.

Although the SPO256 was designed for use with another microprocessor, the Z80 CPU works extremely well with the SPO256 set. If the speech synthesizer is connected to the 7475 output port and programmed with BASIC OUT instructions, the results are amazing.

Constructing the Circuit

I first breadboarded and tested the circuit shown in figure 6.2 to make certain that all my junkbox parts would work with it. The circuit worked on almost the first try, although I did have to make several tries before it gave me something other than noise. The first time it said "Oh," I thought I had hurt it! What it meant was "zero."

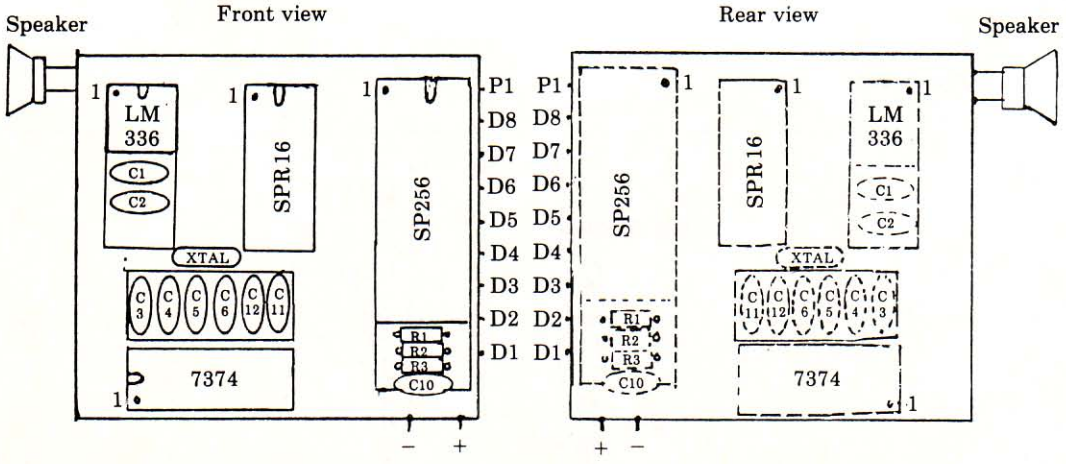
The only critical parts are the two capacitors that connect to pins 27 and 28 of the processor. Even these and the crystal can be just near the values indicated. Try to get the values close to those shown, however.

I later wire-wrapped the circuit using sixteen-pin IC sockets into which I inserted both the discrete components and the ICs. This did away with most of the soldering that otherwise would have been required. Figure 6.4 details the positions of the parts on the board, but this layout can be changed to whatever you feel works best. Notice that eight lines, marked D1 through D8, exit the side of the board (figure 6.4). These are the data lines that go to the output port. The data word that is presented on these lines determines what the spoken word will be. The line marked P1 (figure 6.4) is the reset line that clears all the internal registers of the processor and gets it ready for the next word. Of course, there are the ever-present power supply lines, which should go to the separate supply. Make sure that the 2068 and the module have a common ground.

Using the Just Wrap method described in chapter 2, I wire-wrapped the circuit shown in figure 6.2 in less than two hours. Since there are so many discrete components, breadboarding this circuit is a nightmare, but, if you want to experiment with the circuit, breadboarding is the only way to go.

The SPO256 is a twenty-eight-pin IC and should be placed in an IC wire-wrap socket. Since twenty-eight-pin sockets are difficult to find, I used

Figure 6.4
A component layout of the speech module showing the front and rear of the wire-wrap board.



a forty-pin socket and used the extra space on it for several of the remaining components. The only component that I chose to mount on wire-wrap pins was the crystal (XTAL), and even this could be inserted into the IC sockets if there is enough room. The specification sheet provided with the set calls for a 3.12-MHz crystal, but a 3.579 colorburst crystal will work just fine and costs a lot less. Try to keep the leads for the crystal and its two capacitors as short as possible.

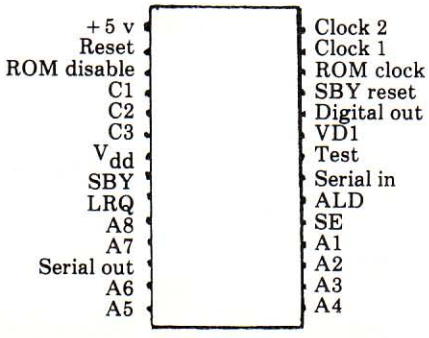
Circuit Operation

The SPO256 has all of the circuitry it needs to transfer data from the serial ROM into its twelve holding registers and produce speech through its digital filters. These filters emulate the human vocal tract and depend on the supplied data only for the correct address of the right pattern stored in the SPR16 serial ROM.

Figure 6.5 is a complete pinout of the SPO256 that makes the processor appear more complicated than it really is. You can ground several of the control pins without having to worry about proper operation. The only pin critical to the operation is the reset pin.

For the processor to operate (talk), the reset pin must be at logic 1, or high. Reset is grounded (brought low) to clear the internal registers and then brought high to send in a new data word, which is used by the processor to locate the correct address in its ROM. Therefore, for the SPO256 to begin talking, the reset line is held low, a data word is placed into the data latches of the 7475 output port, and the reset line is brought high. The processor (running on its own clock) shifts the speech pattern from the ROM into its registers and outputs the synthesized vocal pattern at pin 24.

Figure 6.5
A pinout of the SPO256.



The synthesized output is fed to a low-voltage audio amplifier that produces sound on a small 8-ohm speaker. The resulting sound is amazingly realistic if somewhat robotic. The low-wattage (400-mw) output is not too loud, and you could change the IC to a higher-power one if your application requires more decibels.

Programming the Speech Synthesizer

Like all of the rest of our circuits, the synthesizer is useless without the proper programming. With this project, the programming is the most fun. Ingenious programmers can quickly have the synthesizer saying almost anything, in its own limited way. Several examples are given below to illustrate how the OUT command is used to control the reset line. Notice that the port is called by device number 192. The device number can be any number that the device decoder can produce, however, depending on which pin you select. The reset line is controlled by the OUT command line, 194. This line number, too, could be another number. I chose the numbers I used only because they were convenient. Notice in the program that follows that it takes two OUT 194 commands to complete the speech cycle. With the first OUT command, the processor starts the word synthesis. The second OUT actually does the resetting.

Since many of my capacitors were not exact, I found that it took the reset line a few microseconds to reset. This required that I insert a small pause after the reset command. You may find that this second pause (line 55, PAUSE 20) is unnecessary with your circuit. The first pause (line 40, PAUSE 60) is inserted to give the processor time to finish its word before the unit is reset. The melodies will require a much longer pause here.

The operation of the SPO256 depends almost entirely on programming. Without programming, you might get a low hiss, but that's about all. The data presented to the processor consists of the addresses that the processor sends to the ROM. The patterns (thirty-four in all) are stored in the 256 locations called by the data word. The other locations contain hisses, whistles, and a lot of noise. Even these can be called by number.

Let's run a small program to check the operation of the processor. This simple program mainly shows how the reset line is handled with the OUT 194 command. Notice that it takes three OUT commands to complete a word. The resetting could be done with hardware, but this would require more components. Run this little program and enter a number from the keyboard (0 through 34). The speech processor will output each data word presented on the data bus.

```
10  INPUT A
20  OUT 192,A
```

```
30  OUT 194,0
40  PAUSE 60
50  OUT 194,0
55  PAUSE 20
60  GOTO 10
```

The simplicity of this small program shows what can be done with a data word and three OUT commands. Now try a little program that runs by itself. This one counts down from 10 to blast-off. I think you'll like the ending.

```
 5  LET X = 60
10  FOR N = 1 TO 11
20  READ A
30  GOSUB 200
40  NEXT N
50  DATA 10,9,8,7,6,5,4,3,2,1,64
60  LET A = 34
70  LET X = 200
80  GOSUB 200
90  STOP
200 OUT 192,A
210 OUT 194,0
220 PAUSE X
230 OUT 194,0
240 PAUSE 20
250 RETURN
```

The neat thing about this program is that the activate routine is placed in a subroutine that can be called at any time. Also, the value of the data word can be placed in a data list. This makes speech programming much simpler than with many sophisticated speech synthesizers.

But can this sound synthesizer do anything intelligent? The answer is yes. How about telling time? Try this little program to get a feel for the synthesizer's capabilities:

```
10  FOR N = 1 TO 9
20  READ A
30  GOSUB 200
40  NEXT N
50  DATA 30,24,6,0,1,25,32,40,40
200 OUT 192,A
210 OUT 194,0
220 PAUSE 50
230 OUT 194,0
```

```
240  PAUSE 20
250  RETURN
```

If you want to write a clock program using the list shown in figure 6.3, the values can be inserted with the proper programming.

Here is also a little program to show that the computer is smart enough to answer simple addition problems. Try this one:

```
 10  INPUT A
 20  LET X = A: GOSUB 200
 30  INPUT B
 40  LET X = B: GOSUB 200
 50  LET X = 24: GOSUB 200
 60  LET X = A + B: GOSUB 200
 70  GOTO 10
200  OUT 192,X
210  OUT 194,0
220  PAUSE 60
230  OUT 194,0
240  PAUSE 40
250  RETURN
```

Now enter any number between 1 and 20. The computer will repeat two numbers and give the addition answer verbally. It gets a little confused if the answer is above 20. With a little programming that places the number in a string and then evaluates that string for the digits, the answer could be read out perfectly every time.

I'm sure you can do much better than this with a little work. In this discussion, I have only scratched the surface of the capabilities of the speech processor. I think it is a wonderful project all by itself (without a robot). Now you can be ready for the neighbor who comes over to see your computer and says, "Yeah, but can it talk?"

The Analog-to-Digital Converter

Any discussion of interfacing must include a discussion of the analog-to-digital converter (ADC). Stimuli in the environment can be directly communicated as on and off signals only rarely. Our world sends us what can be called analog signals—signals that vary along a continuum. The seasons change from cold to hot by degrees. Night changes to day in increments. Temperatures vary within a narrow band that we can interpret easily with our sophisticated skin receptors. The computer does not have our capabilities. In order for it to sense these shades of difference, we must provide it

with digital data that represents the variations. The ADC0804 analog-to-digital converter is a perfect component for translating analog to digital signals for the computer.

The ADC is a device that senses varying voltages, converts them to digital words, and presents the words to the computer as digital data. The computer can then make a program decision as to what each data word represents. The device that converts the voltage is a small integrated circuit that can digitize an applied voltage and give it a binary representation.

We'll not cover the internal construction of the ADC except to say that it consists mainly of a circuit that compares a received voltage level with another, reference voltage level and approximates the received voltage in terms of that reference. Since the ADC0804 is an 8-bit device, there are eight successive voltage-approximating tests run on the input to convert the input level to binary data. The ADC0804 uses a reference of 2.5 volts DC in its approximating tests. If the input level is lower than a given test level, the ADC adds the reference voltage to the input voltage and compares again until the test level is overrun. With each test, the ADC places a digital 1 or 0 into its holding register. The reference voltage can be changed to other values that give the ADC the ability to increment the tests on either smaller or larger approximations. If a very small voltage is being measured (within the ADC0804's 0- to 5-volt range), the reference voltage might be changed to 1.5 for greater accuracy. Generally, a reference voltage of 2.5, or half of the operating range of the ADC, is sufficient. You'll see the difference in the scale of values when we construct our first test circuit incorporating the ADC.

Now that we have established that there is a circuit that can digitize a varying voltage, all we need to do is construct devices that take environmental analog (varying) signals and change them to a voltage level that varies with the analog signals' intensity.

Prime examples of environmental inputs are light, heat, pressure, sound, and humidity. Of course, there are other analog signals that can also be converted to a voltage level—acidity, roughness, position, concentration, and other such things that change and can be measured. We can, in fact, connect the computer to almost anything we want to measure. And, if necessary, we can program the computer to interpret the measured results in ways that might takes us hours with paper and pen. With this capability, the computer is a real tool for the measurement of our environment.

In the next main sections, we'll experiment with a few circuits that sense the outside world. We will process analog signals and send them to the computer in terms that it can understand. In this way, we can have the computer analyze the data and even make useful interpretations. We will build a computerized weather station as a project. I think you'll see that

the computer can handle almost any task that we can handle, but with much greater ease than we ever thought possible.

Interfacing the ADC

Let's run a few experiments with the ADC. These will familiarize you with its operation. The ADC0804 is by no means a simple component, but the ease with which it operates will amaze you. Having the ability to change an analog voltage to a binary representation is worth the bother of wiring this fascinating component into the 2068.

Many local supply outlets handle the ADC0804. It is a component that is sold by Jim Pak Inc., and most dealers are now selling Jim Pak products from wall or rotary displays. If this device cannot be found locally, it may be ordered directly from Jameco Electronics, whose address is shown in Appendix A.

Using the breadboard, wire the circuit as shown by figure 6.6. Notice that the read, or RD, pin is connected to a single-pole, double-throw switch. This can be a microswitch or one that has a common, a normally connected (NC), and a normally off (NO) connection. Since we want the chip to be active constantly, we will wire its chip-select (CS) to ground.

The ADC converts voltage by comparing and shifting it by means of the on-chip clock circuit. Notice that pins 4 and 19 have the resistor-capacitor circuit for clock frequency. The values of the components are not critical, but yours should be as close as possible to those shown. The write (WR) pin is left unconnected for this test.

Pin 9 of the ADC is the reference voltage ($V_{ref}2$). It is this voltage that the chip uses to compare the input voltage. The circuit shown uses two small penlight cells to obtain the 2.5 volts needed. Later we will try using a reference voltage obtained from a small op amp that will produce a stable 2.5 volts. It is important that the two AA cells have their ground connected to the system ground also.

The input voltage (the voltage that we want digitized) is fed to pins 6 and 7. Notice that pin 6 is marked plus and pin 7 minus. The minus is connected to ground. We will vary the positive voltage from 0 to 5 volts to see the operation of the ADC. The means to vary the voltage is the variable resistor shown in figure 6.6. The potentiometer can be any value around 1K ohms.

The output of the ADC is on pins 11 through 18. The ADC has active high output lines, and we will buffer the signals before applying them to small LEDs. The buffers add protection to the ADC and give added drive for the LEDs. You might need to add 220-ohm resistors to the LEDs to keep them from pulling too much current. I did not find this necessary, however.

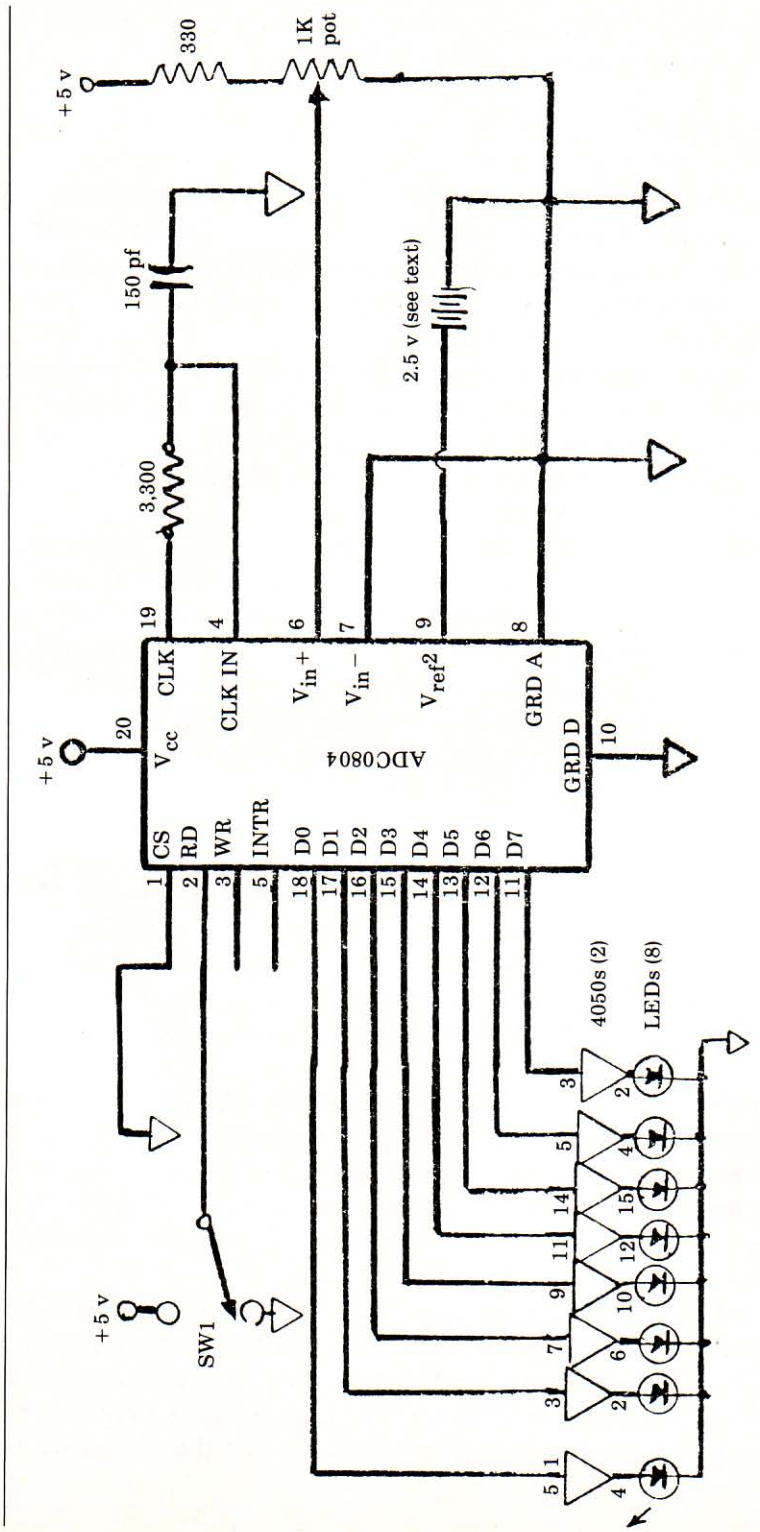


Figure 6.6
The analog-to-digital converter circuit. Clock pulses are provided by an RC circuit at pins 4 and 19.

Testing the ADC

With the circuit completely breadboarded, connect your VOM to ground and pin 6 of the ADC. This will give you the voltage that is being varied into the device. Now vary the potentiometer control. Notice that the voltage swings from 0 to 5 volts. Notice, too, that the ADC does not respond to these changes. The ADC must be told to convert this voltage to a data word. This is done by toggling the RD line. Toggle the microswitch to plus momentarily and then to ground. When the switch is at ground, the data word appears on the LEDs in the form of a binary word. As the voltage is increased, a binary word with a greater value is obtained. As the voltage is dropped, the binary representation is less. Make certain that a switch is used since holding the lead with your hand and touching the ground rail will not work (because your hand will provide enough capacitance to make the RD line misread).

Troubleshooting the ADC

If the procedure described above does not work, recheck all of your connections (especially power). Make sure that the cathodes of the LEDs are connected in common to ground. Almost every LED has a long and a short lead. The short lead is the one that is grounded.

If the ADC still fails to convert the voltage, drag out your logic probe and check pin 5. It is marked INTR, meaning *interrupt.* (Many ADCs call this line BUSY.) Pin 5 remains high until the conversion is complete, at which time it goes low to signal the computer that the conversion is finished and that data is ready for presentation. This line should toggle (change from high to low) each time the microswitch is toggled. If it does not show this transition, one or more connections are incorrect. The INTR line is not used since we are using BASIC. BASIC is not fast enough to need a ready signal. The ADC converts the voltage in about 2 milliseconds, which is plenty of time for our purposes.

Graphing the ADC Output

Using graph paper, draw axes similar to those shown in figure 6.7. The left side is for the decimal values of the binary output from 0 to 256. Along the bottom, note your input voltages stepped by ¼ volt.

Now that your graph setup is ready and your VOM is connected as discussed previously, set the input voltage on pin 6 to about ½ volt. You may need to use the 3-volt scale of your VOM here. Toggle the RD line with the microswitch, and read the value of the LEDs. Their value should be around 10 (LEDs 2 and 8 should be on). Since most VOMs are not exact

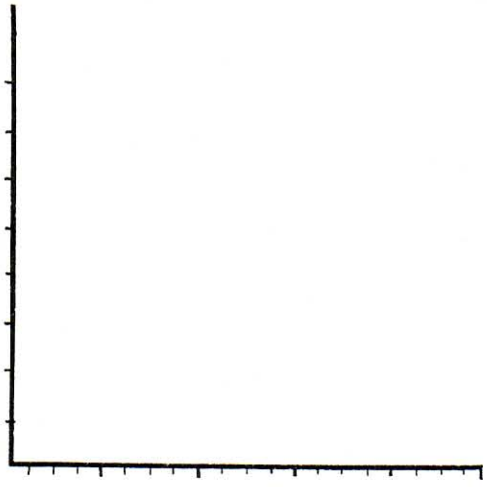


Figure 6.7
A graph setup for logging data provided by the ADC.

and usually read plus or minus 1 volt, you may read 9 or 11. ADCs are very susceptible to line noise, so LED 1 may be on one time and off the next, even with the same input.

If you got a reading around 10, mark it on the graph. Increase the voltage to 1 volt, and repeat the conversion. Turn the voltage up to 1.5 volts, and do it again. Each time, mark the graph with the number represented on the LEDs. Notice that you are getting a straight diagonal line through the center of the graph. Continue doing this until the voltage reaches 5 (the limit of the ADC0804). You might occasionally find that your reading plots some distance from the straight line, probably because the ADC read some noise and gave a misleading value. When this happens, simply disregard that value and redo the conversion.

Now change the reference voltage from 2.5 to 1.5 volts. To do this, simply connect pin 9 to the positive side of a single AA battery. Now, each time a conversion is made, the ADC will approximate it using 1.5-volt steps. Plot another line on the graph using this new reference voltage. Notice that you run out of graph well before you get to 5 volts. This is a good reference voltage to use if your input voltages are low: it gives you a much broader range of numbers to evaluate than would the higher reference voltages.

Repeat the above procedure using the full 5 volts as reference. This time, the graph is much lower on the scale. The range is much too low for use in voltage-level evaluation. Your complete conversion graph should resemble the one shown in figure 6.8.

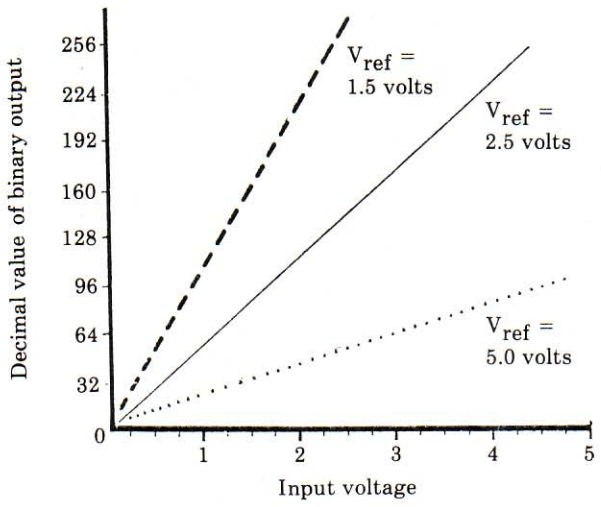


Figure 6.8
Plots of the readouts of the ADC for three reference voltages.

In this test we will let the computer do the toggling. We do this by connecting the read line of the ADC to our device-select pulse. Notice in figure 6.9 that the select pulse is connected through the NOR gate with WR as in previous experiments. The write line on the ADC must be connected to the WR line of the 2068 bus also. Figure 6.9 also shows how the LM366 is used to obtain a 2.5 reference voltage. You may leave the two AA cells connected if you prefer. The other connections on the ADC remain as they were in the previous test.

The data lines are not connected to the 2068 yet. This test simply determines if the 2068 will command the ADC to convert an input voltage. Enter and run the following short test routine, and observe the output of the LEDs as the potentiometer is varied through its voltage range:

```
10  OUT 192,1
20  GOTO 10
```

With the above two-line program running, you should observe an immediate change in the LEDs as the pot is varied. The value on the LED display should increase as the pot is increased, and vice versa. If your logic probe is placed on the INTR pin of the ADC, it should show an alternating signal as each conversion is completed.

With your ADC connected as in the previous test, replace the LEDs with data lines going to the 8212 input port as shown in previous examples. Connect the device-select line for the 8212 port to pin 2 on the 74LS154 decoder. This will give the input port a device number of 193. Therefore,

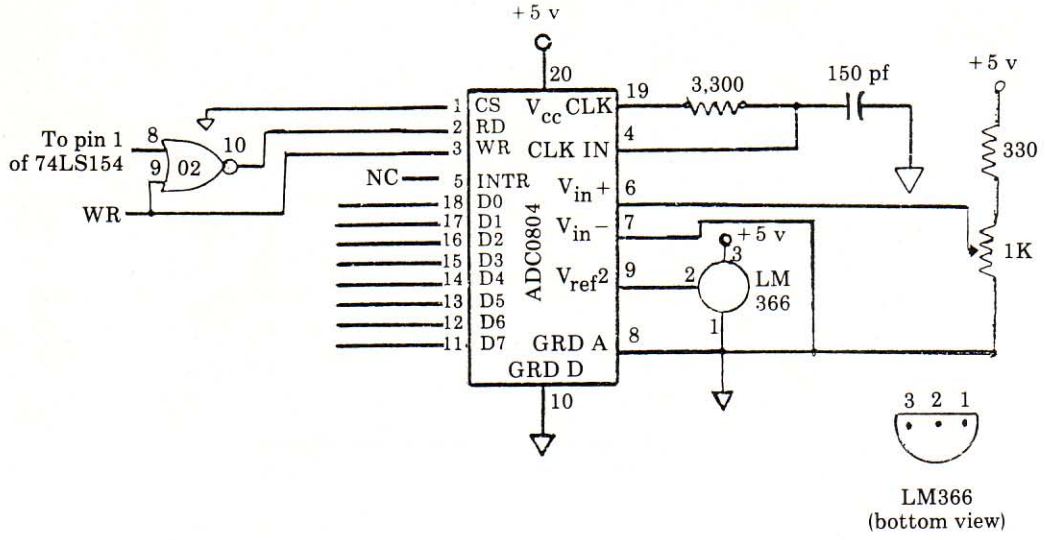


Figure 6.9
A diagram of the ADC showing how output is controlled. If the separate transformer power supply is used, the 2.5-volt battery should be replaced with the LM366 voltage regulator.

any time that we want to read data into the computer, we simply call this device number with the IN instruction. An example would be:

```
LET A = IN 193
```

This will transfer the data that has been converted by the ADC into the computer and assign the converted value to the variable A.

The RD line of the ADC is connected through a NOR gate to pin 1 of the 74LS154, so its device-select number is 192. So, in order to convert an analog value, simply enter the following:

```
OUT 192,1
```

This places the converted value on the inputs of our 8212 input port, and the previous command transfers it into the computer. I told you the ADC was simple. However, you are now using almost more ICs than your breadboards can take, and you may want to eliminate the 4050 CMOS buffers to get more room. You do lose a little protection for the ADC when this is done. Make sure that you took out the LEDs since they will make the power supply noisy and cause the ADC to misread values.

The Weather Station

Up to this point, we have only tested the operation of the ADC. Now that you know how it works and that your converter is working as it should, you need to construct a circuit that will put it to use.

Since the world sends analog signals almost exclusively, we should have no trouble finding such signals to convert. The problem is determining which kind we want. You may want to convert such things as light, pressure, acidity, concentration, or even sound level. There are hundreds of other candidates for conversion. I have chosen temperature, wind speed, and moisture sensors as three good circuits to show the use of the ADC. With these, it is possible to connect your computer to the weather, so to speak. Using the data that can be gathered by these circuits, your computer can possibly even make predictions about the weather.

The Moisture Detector

Now let's build the weather station. Actually, we will convert only three of the many kinds of information that reflect the weather, but the knowledge that you gain here can easily be applied to the many other weather factors. We will first build a moisture sensor. With this circuit, your computer can determine the moisture level in the ground or in the soil of house plants, and, if adjusted properly, it could easily register the humidity of the air.

Then we will construct a simple anemometer, or wind indicator. With additional work, you can easily wire this device to tell wind direction, calculate the wind-chill factor, and determine gust speed.

Of course, no self-respecting weather watcher would be complete without a circuit to determine the temperature. A simple heat sensor is detailed that is very fast and sensitive. It can be adjusted to measure the temperature of the house, the yard, the refrigerator, the heat pump, or even your wood stove.

Constructing the Moisture Detector

Your ADC should be wired up as in the last test. The only connections that will be changed for this circuit are the connections that go to the *voltage in* pins ($V_{in}+$ and $V_{in}-$). And, since your breadboards are full of components and these circuits are simple, you might want to connect these circuits together using your small, multicolored jumper leads with the alligator

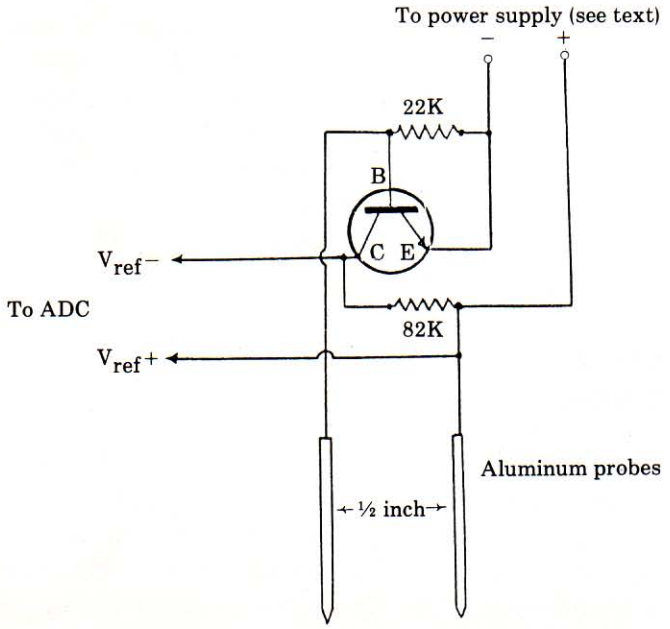
clips. Once the circuits are working, you can put them on a small piece of perf board and wire them up point to point.

The moisture sensor uses the familiar 2N2222 transistor. The sensor is basically a balanced circuit that is activated by a decrease in the resistance between its probes. Figure 6.10 shows the complete sensor circuit, and figure 6.11 shows the finished board. It may seem simple, but it works just great with the capabilities of your ADC.

The output of the 2N2222 is determined by the voltages on the collector and the base. Therefore, it is extremely important that the values shown for the resistors be exact. It is this balance that allows the circuit to operate. With a high resistance across the probes, the output to the ADC is very low. Any decrease in resistance will cause the transistor to start conducting, and the voltage will rise proportionally. It is this voltage that the ADC will measure and convert to a moisture-level reading.

The probes can be any metallic rods, but I've found that aluminum nails and copper work best. They should be around ½ inch apart and exactly parallel. Their length is not critical, but I've found that 6 to 8 inches seems to work best. Their diameter is not critical either, and I've chosen 3/16-inch rods since they don't bend when they are inserted into the ground.

Figure 6.10
A diagram of the moisture detector. The probes are made of aluminum nails, which are inserted into soil.



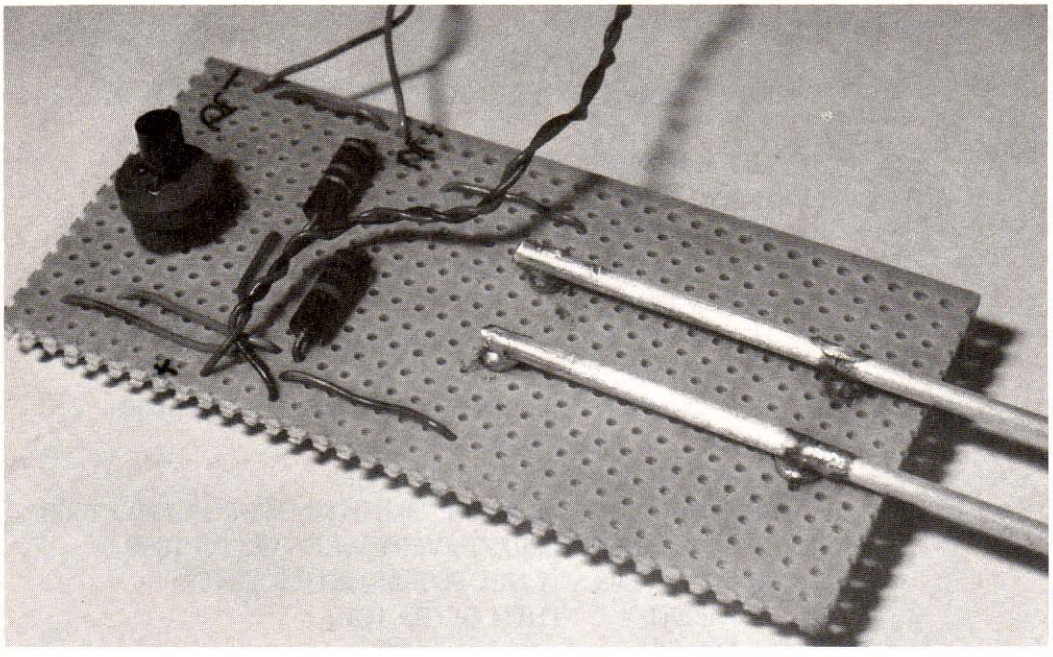


Figure 6.11
The completed moisture detector.

Once the circuit is wired as shown in figure 6.10, you can give it an initial test by connecting the positive V_{in} lead to the VOM positive test lead and the negative lead to the negative V_{in} input. With the VOM set to read on the 5-volt scale, moisten your thumb and first finger and place them across the probes. The voltage should rise immediately. You might have to press the probes considerably between your fingers to make the level increase. Dry your fingers and repeat the test. The level should now be much lower.

Calibrating the Moisture Detector

When the moisture sensor is connected through the ADC to the computer, the computer will not know that it is taking readings on moisture. Remember, all it is doing is measuring a voltage level and converting it to a decimal value between 0 and 255. You, the programmer, will need to calibrate the numbers to meaningful values that the computer can output.

For example, if the sensor indicates a 2.5-volt level, the computer outputs a value of 128. You must tell it what this value represents. If your calibrating tests show that 128 is a value obtained when the probes are in slightly damp soil, then that is what the program should output. A simple soil moisture-content test is shown below. I performed it with my probes

and circuit; your values may be somewhat different, but the procedure would be the same.

```
  5  REM "MOISTURE TEST ROUTINE"
 10  OUT 192,1
 20  LET A = IN 193
 30  IF A < 128 THEN GOTO 100
 40  IF A = 128 THEN PRINT "DAMP": GOTO 1000
100  IF A > 64 AND A < 128 THEN PRINT "SLIGHTLY DAMP":
      GOTO 1000
110  IF A < 64 AND A > 32 THEN PRINT "SLIGHTLY DRY":
      GOTO 1000
120  IF A <= 32 THEN PRINT "VERY DRY": GOTO 1000
130  IF A < 255 THEN GOTO 200
140  PRINT "SATURATED": GOTO 1000
200  IF A <= 192 THEN PRINT "SLIGHTLY WET": GOTO 1000
210  IF A > 224 THEN PRINT "VERY WET": GOTO 1000
1000 PRINT "PRESS ANY KEY FOR ANOTHER TEST"
1010 IF INKEY$ = "" THEN GOTO 1010
1020 GOTO 10
```

Notice that the values that the moisture sensor outputs are broken down into six calibrations. More calibrations are possible; I kept the number low so that the program would not get lengthy.

If you want, you can write a program that plots a graph of the values of the moisture content. Also, a program with long pauses could be written to measure moisture over long periods of time (to indicate drying out, or degree of drop in moisture level).

The Wind Indicator

A device for measuring wind speed, again, is called an *anemometer.* An anemometer takes the speed of a small propeller and converts the reading to indicate the horizontal velocity of the moving air. The National Weather Service uses anemometers to calculate the wind-chill factor and in estimating air movement at ground level.

Now that we have a small device for changing a voltage to a decimal value for the computer, this type of measurement should cause us no trouble. The problem, of course, is constructing a mechanical device that will generate a voltage that is proportional to the wind speed. And, really, even this is not a problem, with a little imagination and a few simple parts. Several hobby shops sell small wind indicators that could be converted for our purposes. I much prefer to build my own, being the tinkerer that I am.

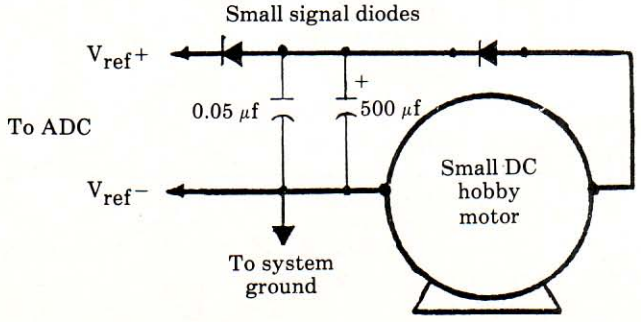
Constructing the Anemometer

I have always wanted to use a small DC motor to generate some type of current that I could use constructively. This is a great opportunity. If a small permanent-magnet-type DC hobby motor is spun, a small voltage is produced on its output leads. Any time a copper winding is moved through a magnetic field, this phenomenon occurs. If this small motor is connected to a propeller, the speed of the propeller, when the propeller is spun by the wind, is proportional to the voltage produced. With this in mind, let's build a small experimental wind-speed indicator that the computer can use to calculate wind speed.

The motor must be chosen by trial and error. That is, the motor needs to be small and easily turned. If the magnets are too strong, the propeller will hesitate in starting. Of course, the stronger the magnets, the greater the voltage, usually. Therefore, you should try three or four different motors to determine which is best. Use your VOM on the 3-volt scale, and spin several motors with your fingers. The needle should jump to about half scale. If the needle tries to go down, either reverse the leads or spin the motor shaft in the other direction. If a small slot-car motor is nearby, grab it; it will work fine. Almost any old battery-operated toy around the house has the type of motor I'm talking about.

Now look at the circuit shown in figure 6.12. Notice how a few simple components can smooth the voltage out a little and make reading of the voltage level easier for the ADC circuit we constructed earlier. The two diodes rectify the voltage, which has a tendency to be alternating current. These two diodes are small-signal types that are not critical and can be similar to the ones sold by Radio Shack as their part 276-1620. Then the two capacitors shown smooth the current and keep voltage spikes from

Figure 6.12
A detail of the connection of a small permanent-magnet hobby motor for the wind-speed indicator.



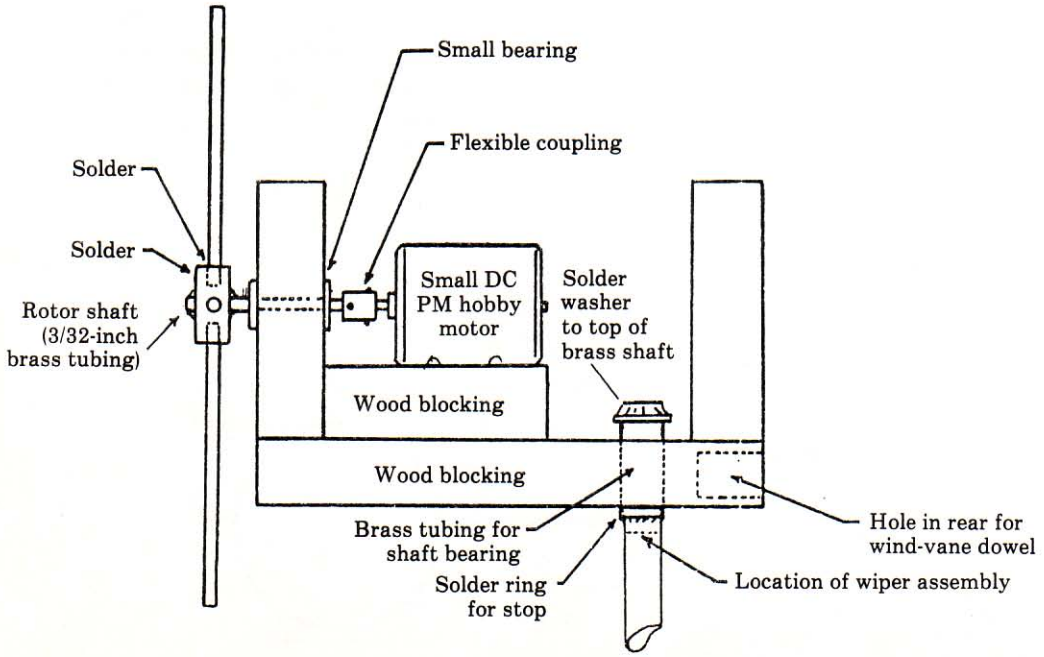
getting to the ADC. Even then, some variations get through, but they can be easily handled by software. We'll cover this later, once the circuit is operating.

Now things may get a little complicated, but I'll keep the mechanics of the indicator as simple as possible to save you expenses and headaches in obtaining parts.

The small anemometer that I built mainly from wood and brass tubing works great, and I feel that the construction can be handled by the average tinkerer (figures 6.13 and 6.14). The materials specified, such as the tubing and hardwood, can be bought at most hardware stores and hobby shops. The small bearings may be hard to find but can be ordered from Stock Drive Products Inc. (Appendix A). The inside diameter on these is 3/32 inch. You could use a close-tolerance brass tube for the bearings, but the friction would be a little greater with this arrangement.

The body, or housing, of the indicator is formed of simple wood blocks cut as shown and then covered with a thin aluminum sheet to protect the motor from rain and birds. The front and rear blocks are approximately 3 inches high and 2 inches wide. Their thickness should be at least ¾ inch or thicker. This will give the rotor shaft a greater area for bearing and increase

Figure 6.13
A construction detail of the anemometer. The wood blocking is covered with a thin aluminum sheet for weather protection.



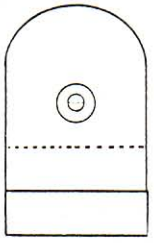


Figure 6.14
A detail of the anemometer front.

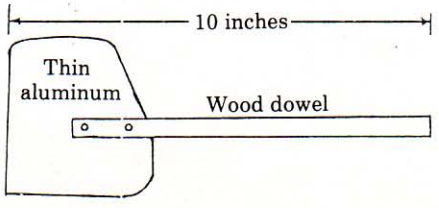
its strength to resist wobble. The blocking is held together by simple wood screws.

The length of the housing is not critical, but it should be around 4 inches. You may come up with a better design than this. An indicator made entirely of metal would be best, of course. Notice in figure 6.13 that the housing revolves on the vertical shaft using a short piece of brass tubing as a bearing. The housing is held at the top of the shaft by a small brass ring soldered on the vertical shaft, and the housing is kept in place by a retaining washer soldered to the top of the shaft. When the wind vane, or tail (figure 6.15), is attached as shown in figure 6.13, the housing will turn and track into the wind. The tail should be the length necessary to balance the indicator. The tail should be the last part attached so that you can determine the length of the wood dowel that will give proper balance.

Building the Rotor

One of the more critical parts is the rotor, especially its paddles, or blades. Figures 6.16 and 6.17 show how the rotor should look. Mine had only two blades (figure 6.18), and it had a hard time starting rotation until I increased the blade area. You might prefer building the two-blade type and making the paddles larger. I made the paddles out of hard balsa wood and

Figure 6.15
A detail of the wind vane on the anemometer.



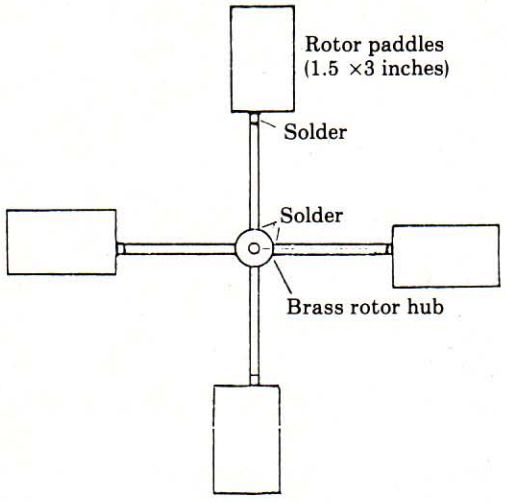
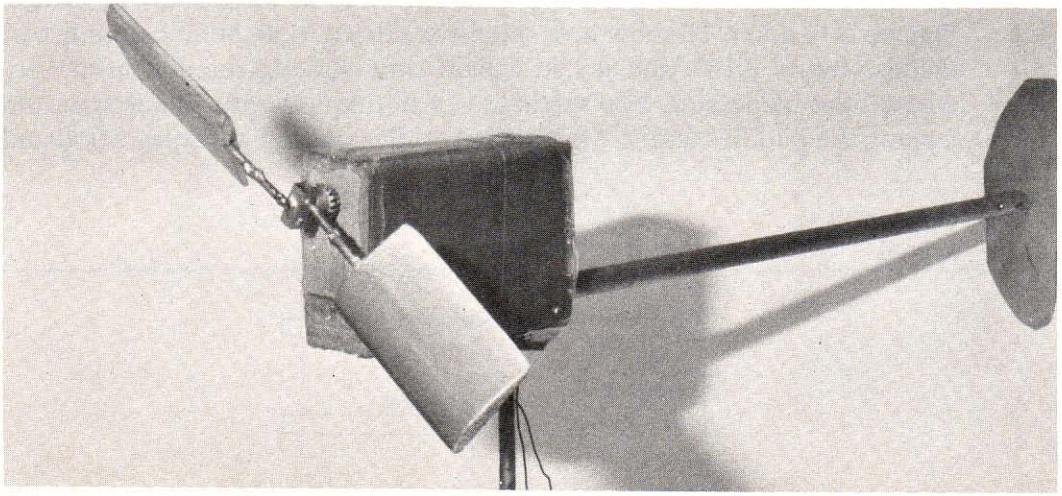


Figure 6.16
A detail of the rotor for the anemometer.

Figure 6.17
A cross section of the rotor paddle showing the embedded brass tubing.

Figure 6.18
The completed anemometer.



glued a brass tube in their center. I could have saved myself some trouble if I had used 1/16-inch aluminum sheet for the paddles and simply bolted them to the paddle shafts. I sprayed the balsa with clear lacquer to protect it from moisture. With aluminum, this wouldn't have been necessary.

The rotor shaft (the shaft that goes through the bearings) is 3/32-inch brass tubing. It is soldered to a round solid-brass hub. The paddle shafts are 7/64-inch brass tubing. The brass-tubing inserts in the paddles slide down onto the paddle shafts for balancing and pitch adjustment. Once the paddles are slid onto the paddle shafts, their pitch is set, and the blades are balanced. The paddle inserts are then soldered to the paddle shafts. The pitch (angle to the wind) should be about 30 to 40 degrees. This large angle makes it easier for the paddles to start rotating in a slight breeze but keeps them from spinning too fast in higher winds.

The rotor shaft should connect to the small motor through a small, flexible coupling. Because it is almost impossible to get exact alignment, some play is needed to keep things from binding. I simply soldered a small pin across the shaft of the motor and cut a slot into the rotor shaft into which the pin fitted. The motor is held in place on wood blocks with thin aluminum straps.

Since winds can get rather strong, all solder joints should be clean and solid. This is especially important on the paddles and rotor hub. I speak from experience. When the anemometer was first completed, I couldn't find a breeze anywhere. That night, we had gale-force winds. Needless to say, rotor blades and paddle shafts went everywhere. I increased the amount and quality of soldering, and now everything holds nicely. Make sure that the rotor is balanced. If it isn't, the vibrations will cause the mechanics to fail.

Testing the Rotor

Once the rotor is completed and connected to the small motor in the housing, test it for operation. Hold it in front of a fan that is operating on low. It should start rotating with no problem. If it sticks, check the alignment, and make sure that the motor is not too hard to turn. Remember, the ADC measures even small voltage, so change to an even smaller motor if everything else rotates freely.

Constructing the Wiper Assembly

Since the entire indicator can revolve as the wind changes direction, some method must be devised to transmit the voltage to the ADC and not snap the wires that carry the voltage. This method involves the use of a wiper assembly. Two metal wipers from an old large potentiometer are attached to the bottom of the housing. These wipers contact two tracks on a round,

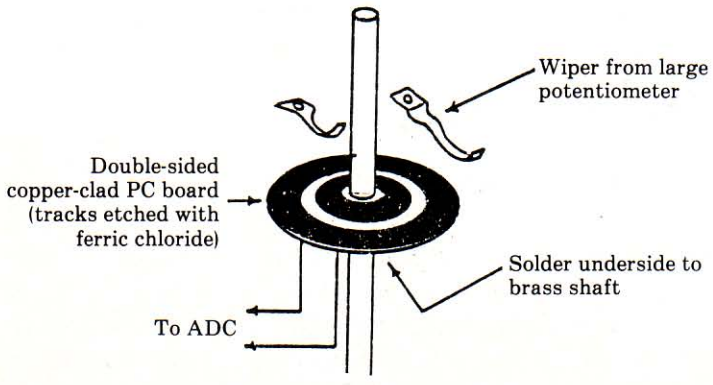
double-sided, copper-clad board. This board is drilled in the center to fit onto the vertical shaft. Two round tracks are covered with etch resist so that there is bare copper on each side of the circular tracks (figure 6.19). The copper on the underside of the board is sprayed with lacquer to keep it from being etched. The leads from the motor attach to the wipers, which contact the copper tracks. The leads that run to the ADC are inserted through the bottom of the clad board to the tracks on the top side. Figure 6.19 shows how this is done. The copper underside of the board is cleaned of the lacquer and soldered firmly to the vertical shaft. Before you place the indicator on top of the house (or wherever), you should coat the wiper assembly liberally with silicon grease to protect it from wear and corrosion.

Because the wiper assembly is a little difficult to build, I thought about constructing the indicator with a vertical rotor shaft that used small rounded cups to rotate. But I decided that I wanted my anemometer to track wind direction and elected to build the paddle design. You might want to try an indicator that doesn't need a wiper assembly. Figure 6.20 shows how the rotor would appear. Everything else except the vertical shaft and wiper assembly would be built the same.

Before final placement and after testing the indicator in a slight breeze, make sure that the anemometer is protected from rain. Coat all seams in the housing with silicon rubber compound. This will keep the rain out of the housing and protect the motor. Use very lightweight wire if you place the anemometer on the highest point of your house or chimney. In a thunderstorm, the wire could act as a lightning rod, so don't connect it to anything during a thunderstorm.

If you want to monitor wind direction, you can construct a rotary wiper assembly that fits on the inside of the housing much like the wiper assem-

Figure 6.19
A detail of the construction of the wiper assembly that allows the wind indicator to turn and still transmit voltage to the ADC.



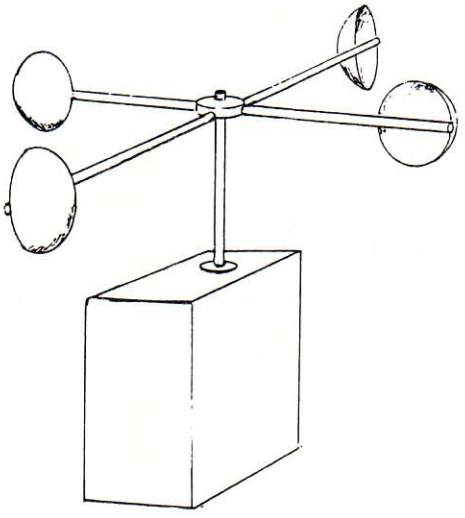
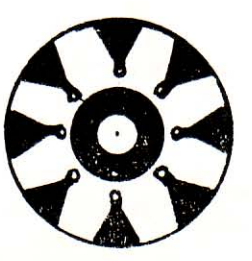


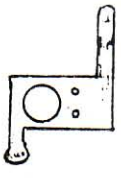
Figure 6.20
An alternate anemometer.

bly just constructed. On this board would be four to eight radial patterns that contact the wiper at different directional points. Such a wiper assembly is depicted in figure 6.21.

The eight wires could be run down the center of the hollow vertical shaft to a separate 8212 input port that could be called by the program to indicate a data word. That word would be translated into directions. I'll let you work on that project, however, since right now I have to go and get Captain Hook out of the pantry again. (I wish I had never installed that voice synthesizer. That "Please hurry" is driving me crazy.)

Figure 6.21
A diagram of the wiper assembly to transmit wind direction. The outer contacts connect to the data lines. The center track connects to ground.





Wiper contact

Now that the wind indicator is indicating correctly, we need to write a short program that will translate the readings into wind-speed measurements. This is not a simple task. The wind is constantly changing direction and speed at ground level, and this complicates the readout of the speed. Then there is the problem of gusts. The wind blows steadily in some places, but around here it is blowing 35 miles an hour one minute and 2 miles an hour the next. The ADC might read the wind speed at any time, even during a gust. The Weather Service gives the low average wind speed and tells you the speed of occasional gusts. In my example, the wind speed would be 2 mph with gusts to 35. I wondered if a program could handle this. It did.

The Anemometer Program

The ADC reads all of the many values that the DC motor puts out. The motor makes a lot of noise, so it reads these spikes, too. The result seems to be all garbage. However, by using your VOM in conjunction with the ADC as it reads the voltage, you notice that the numbers are higher on the average the faster the rotor spins. The important word is *average*. Since the anemometer is reading signals from gusts, no wind, and noise, we must average the values. We want to get only an average *value* at this point; we'll change it to an indication of speed later.

Try the following program on the 2068, and you'll get a screenful of numbers. Since your incoming voltage will be small, connect the reference voltage to the single AA cell for a reference of 1.5 volts. This makes the resolution of the ADC greater. Make sure your ADC is connected to pin 1 of the 74LS154 decoder and that your 8112 I/O port select line is connected to pin 2 of the 74LS154 decoder.

```
 5 REM WIND TEST PROGRAM
10 OUT 192,1
20 LET A = IN 193
30 PRINT A
40 PAUSE 30
50 GOTO 10
```

Notice that your numbers vary widely if the wind was blowing. If it wasn't, most of the numbers should be about the same, probably around 8 or 9. Remember this value since it represents no-wind conditions.

We could add all twenty-two numbers down the screen and divide them by 22 to get the average, but why not let the 2068 do these things for us? Try the program below to get the average value:

```
10  LET X = 0
20  FOR N = 1 TO 23
30  OUT 192,1
40  LET A = IN 193
45  PAUSE 30
50  LET X = X + A
60  LET X = X/22
70  PRINT X
```

You see how we sampled the voltage twenty-two times and then added up the values in the variable X. Then we got the average value by dividing X by 22. This eliminated the high and low values and gave us the middle value.

Now convert the average value to wind speed. To do this, you again must calibrate the values obtained to the known values. Let's say that the wind was not blowing, and you got an average value of 9. This figure would represent calm conditions, or a wind speed of 0. If the Weather Service is saying the wind is blowing at 5 miles an hour, and your value is 23, then 23 represents a 5-mph wind speed. You need several known wind speeds to exactly calibrate the readings.

Of course, your numbers will be different once you calibrate the anemometer for different wind speeds, but the following program shows how the average wind speed can be read out using the output of my ADC:

```
10  LET X = 0
20  LET B = 0
30  FOR N = 1 TO 100
40  OUT 192,1
50  LET A = IN 193
60  LET B = B + A
70  PAUSE 20
80  NEXT N
90  LET B = B/100
100 PRINT B
110 LET A$ = "WIND IS APPROX. "
115 LET B$ = " MPH"
120 IF B > 50 AND B < 100 THEN PRINT A$;35;B$
130 IF B > 30 AND B < 50 THEN PRINT A$;25;B$
140 IF B > 20 AND B < 30 THEN PRINT A$;15;B$
150 IF B > 15 AND B < 20 THEN PRINT A$;10;B$
160 IF B > 0 AND B < 15 THEN PRINT A$;"CALM"
170 LET X = X + 1
```

```
180  IF X = 22 THEN GOTO 200
190  GOTO 30
200  CLS: GOTO 10
```

The resolution could be even greater, but the program starts to get a little long. You can get fancy by including the wind-chill factor in the output. You'll be surprised how cold wind gusts can get. Use the wind-chill chart shown in figure 6.22. Have the computer make the necessary conversions. Only a few lines need be added to the above program.

The Temperature Sensor

A good temperature sensor is a must in today's world of high fuel bills and energy shortages. I feel that heat is one of our most important commodities. I have thermometers all over. They are next to worthless; I think I could freeze to death before they showed any change. They seem to have a response time of about an hour. The shop thermometer might be showing 75 degrees while my feet are numb. I need a fast-acting sensor if for nothing else than to prove I'm not dying.

Constructing the Temperature Sensor

Most of the commercial devices that read temperature rely on the LM3911 heat-sensing IC produced by National Semiconductor. This IC is proclaimed to have the best tracking of all ICs that are produced for sensing heat. It gives 10 millivolts for each degree change in temperature. The fact that the output is in Kelvin shouldn't be a problem. My only problem with this IC was that I couldn't get it to work. Maybe I missed something in the specification sheet?

Figure 6.22
A chart for calculating the wind chill with the anemometer project.

Wind speed (mph)	*Outside temperature*					
	30	*20*	*10*	*0*	*−10*	*−20*
0	30	20	10	0	−10	−20
5	27	16	6	−5	−15	−26
10	16	4	−10	−20	−35	−45
20	5	−10	−25	−40	−55	−70

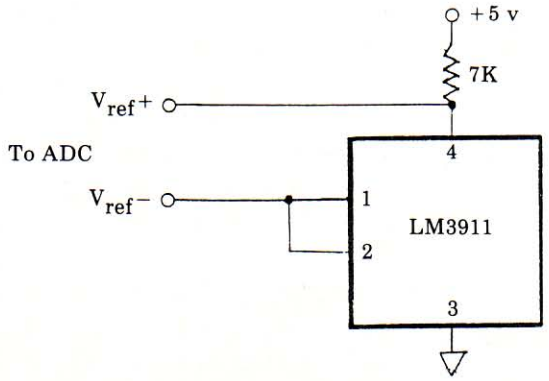
Figure 6.23 shows how I connected the LM3911. Still nothing! I had to have a device that would work, and fast. I was getting colder just thinking about it. Why not make it simple? There were several good thermistors lying around the shop. Could one of these little inexpensive devices be made to work? The answer was yes. The complete circuit is shown in figure 6.24.

We found that the 2N2222 transistor can be made to output a voltage by changing the bias of the base lead. The thermistor is a small, heat-sensitive, variable resistor that could be attached to the base to provide this changing signal. (I felt that I should be able to calibrate the temperature just using my feet as indicators.)

Although you may want to experiment with the LM3911, I found that the Workman thermistor worked fine. The part number is FR191, and the thermistor has a cold resistance of 79 ohms. Any with this specification would work. Most electronic supply outlets carry this part since it is a standard TV replacement part. It costs very little compared to some of the sophisticated heat-sensing ICs.

Again, this circuit is so simple that you can clip it together with your jumper leads. The circuit is balanced, so use the resistor values shown. The 560-ohm resistor is critical, for without it the circuit can be effectively shorted out if the pot is turned all the way up. The 1K potentiometer should be a good-quality, smooth, linear-taper pot since it is used to set the sensitivity of the sensor. This circuit can be adjusted to provide a fast or slow rise when the temperature varies. I wanted to see the changes in the shop temperature more or less as they happened, so I adjusted it especially fast and found that it tracked right along with my feet. I could not want anything better.

Figure 6.23
A detail of a temperature sensor using an LM3911 IC. This component supplies 10 millivolts for each degree Kelvin.



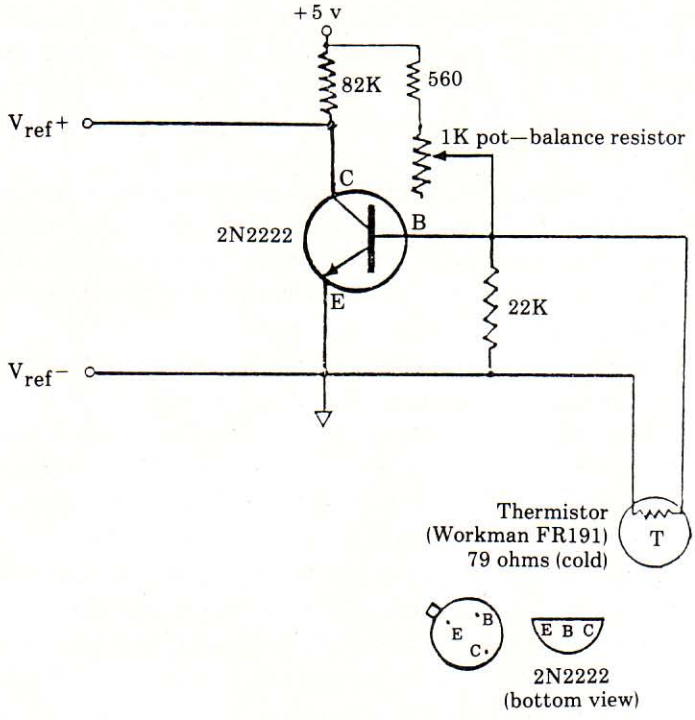


Figure 6.24
A temperature-sensing circuit. The potentiometer is balanced for different ranges of temperatures.

Calibration of the ADC values is difficult since the circuit can be adjusted variously. Therefore, it might be a good idea to place the potentiometer and sensor circuit in a small case so that the pot can have a sensitivity scale on the front. This will give you an idea where you are in the sensitivity range. I used my VOM to see the sensitivity when I tested the device. Simply connect the VOM to the inputs to the ADC, and watch the 3-volt scale as the pot is adjusted. Then place your fingers on the thermistor, and watch the rise time. To make the circuit sensitive, turn the pot slightly up to make it hold the reading longer when your fingers are removed. Now touch the thermistor, and the reading should rise rapidly. The thermistor is now very sensitive. I got it so sensitive once that it picked up my body heat as I stood next to it. This range could be used for reading values within a very narrow temperature band. The sensor can be placed on a longer lead and plugged into the case with plug-in connections. Thus prepared, the sensor can be placed in different locations.

The output of the circuit to the ADC is high, so 2.5 volts was used as the reference voltage to the ADC. If the thermistor is to be placed outside,

it should be sprayed with clear lacquer to protect it from moisture. Figure 6.25 is a sketch of the FR191. Figure 6.26 shows the completed heat sensor.

Calibrating the Temperature Sensor

You'll have to use trial and error for calibrating this circuit since the thermistor does not track in a linear fashion. Figure 6.27 shows the values I obtained in the extra-sensitive range as the heater in the shop came on and went off. Now do you see why I freeze? The calibration values given on the scale are estimates, but they are accurate enough for my purposes. As you can see, the shop almost certainly has too little insulation since the change

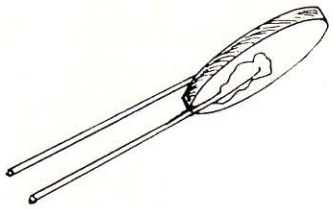
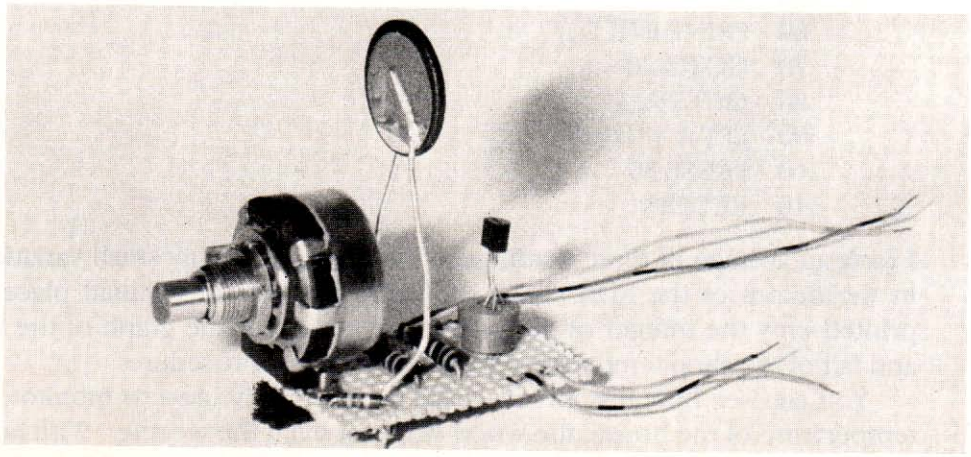


Figure 6.25
The thermistor.

Figure 6.26
The completed heat sensor. The thermistor could be attached with a cable for better placement.



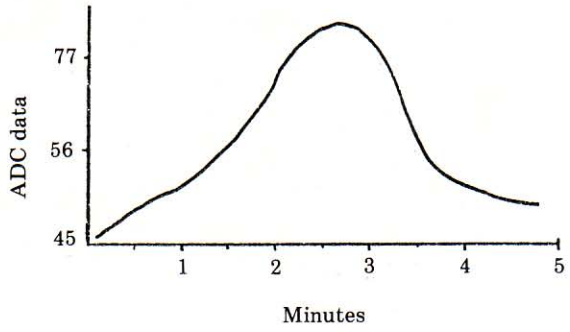


Figure 6.27
A diagram of the ADC output showing the rise and fall in temperature as the shop heater came on and went off.

in temperature is quite rapid. Of course, the weather outside was very cold at the time. A good program would take this factor into account when calculating temperature and heat-loss radiation effects.

This circuit is not as noisy as the wind indicator, so averaging is not as important. It may read 1 or 2 points off in either direction due to the estimating procedure used inside the ADC. I found that a simple logging program was all that I needed. Notice that the program is very similar to the one used earlier for the moisture detector.

```
  5  REM "TEMPERATURE PROGRAM"
 10  PRINT "THE TEMPERATURE VALUE"
 20  GOSUB 80
 30  LET B = A: GOSUB 80
 40  LET C = A: GOSUB 80
 50  LET D = (A + B + C)/3
 60  PRINT INT(D);" ";
 70  GOTO 20
 80  OUT 192,1
 90  LET A = IN 193
100  PAUSE 30
110  RETURN
```

I took an average of three readings to compensate for any small variations in the output of the ADC. Since the value had lots of decimal places, I printed only the integer of the value. The values for the graph of the rise and fall of the shop temperature were taken by this procedure.

You can see how this circuit could be effectively used to monitor the temperature of the house, the wood stove, or even the weather. With additional circuits like those built previously, many things are possible.

7

Going Further

By building the circuits that have been covered, you now have the ability to go that extra mile. Unless you do, the knowledge that you now have will vanish. That mile will be the hardest since there are no guideposts to follow. This is where you take the facts and concepts that you have learned and apply them to those ideas that you know can work but which you have never tried.

Most books of this nature just quit and let you stumble on. I will try to give you a few ideas that I know will work with the electronics that have been constructed. Most of these I have tried. Many have even worked.

The Timex/Sinclair 2068 is a beautiful piece of equipment. It has all you need to accomplish almost any task that you want. Many personal computers in this price range can't make that claim. Therefore, you are not limited by the computer, your knowledge, or the electronics. All that you need is a little imagination and a lot of hard work.

Imagine that you want your home completely computerized. First imagine everything that is electrical in your home as being connected to and controlled entirely by the computer. Then cross off only those things that could not possibly be good candidates. Here are a few home connections that might be on your final list:

1 *Garage door opener.* Many garage door openers rely on radio waves. Why not use light or sound? I put this first on the list since it is my next project. I have to get out of the car and open that blasted door each time I roll in. Most of the time, it seems to be raining cats and dogs. This design could easily be adapted to open gates, too. Radio Shack has a fine little solenoid that could be wired up.

2 *Automatic garden sprinkler.* I seem to spend most of my summers watering a few skinny plants that always wither up and die no matter how much attention they get. I should turn this job over to the 2068 and forget it. It's quite possible with the components we have covered. The electrical ice-cube-tray-filler relay on the refrigerator or the electrical inlet valve on the washer would make a great sprinkler control. Hmmm . . .

3 *Model railroad control.* I've always loved trains but never had the room for a complete layout. If I did, the 2068 would control it. After all, most of the track controls are small relays. This would be a great project. Having visions of Silver Streak?

4 *Radio-controlled robot.* We didn't cover the radio control of robots since it alone could be the subject of a whole book, but a robot that is controlled by the 2068 through a radio link wouldn't be all that difficult to build. Just think, you could send him or her out for coffee! The AY3-1015 UART shown in Appendix D is half of the needed circuit, and I saw a great little transmitter and receiver at Radio Shack just the other day. A radio link to the 2068 could be used to log data for the ADC from remote locations such as the office, yard, shop, or barn.

5 *Automatic telephone-answering service.* With a little more work and another programmable speech synthesizer, you could easily build an answering device and recorder control. With a little more work, the watering system or whatever could be activated by the phone from downtown, and you wouldn't ever have to come home. The possibilities here are endless.

6 *Programmable stereo system.* This is probably already commercially available, but I'll bet it costs a fortune. With an indexed collection, you could just simply press the desired selection, loudness, mood, or whatever, and the 2068 would do the rest. It would be complicated but very impressive.

You get the point. There is nothing (or, at least, very little) that the home personal computer can't handle. The Timex/Sinclair 2068 can be as versatile as you need. Many peripherals and exotic parts are coming on the market that will make your interfacing for the above projects even less complicated. Take advantage of them if at all possible. Don't be too quick to buy a circuit that you can easily build since you don't want to miss that satisfaction of having done it yourself.

One final note: like me, you may have started with the TS 1000 or 1500, and may still have it around. In case you want to experiment, I've included a circuit diagram for a device-selection pulse generator (Appendix H), along with software conversion instructions (Appendix I), that will work for the 1000 or 1500 for all of the projects covered in this book. You simply use PEEK and POKE rather than IN and OUT. I include it since you might want to try a circuit that controls a computer with a computer. Now that's real interfacing!

Appendixes

Appendix A: Mail Order Suppliers

Electronic Parts

JDR Microdevices
1224 South Bascom Avenue
San Jose, CA 95128
408 995-5430

Solid State Sales
P.O. Box 74D
Somerville, MA 02143
800 343-5230

Digi Key Corporation
Highway 32 South
Thief River Falls, MN 56701
800 346-5144

Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002
415 592-8097

Advanced Computer Products
P.O. Box 17329
Irvine, CA 92705
800 854-8230

Quest Electronics
P.O. Box 4430(S)
Santa Clara, CA 95054
408 988-1640

Electronic Surplus Supermarket
P.O. Box 988
Lynnfield, MA 01940
617 532-2323

BNF Surplus Enterprises
119 Foster Street, P.O. Box 3357
Peabody, MA 01960
617 531-5774

Robotic Parts

Amsi Corporation
P.O. Box 56
Bellport, NY 11713
516 541-5419

Small Parts Inc.
P.O. Box 381736
Miami, FL 33138
305 751-0856

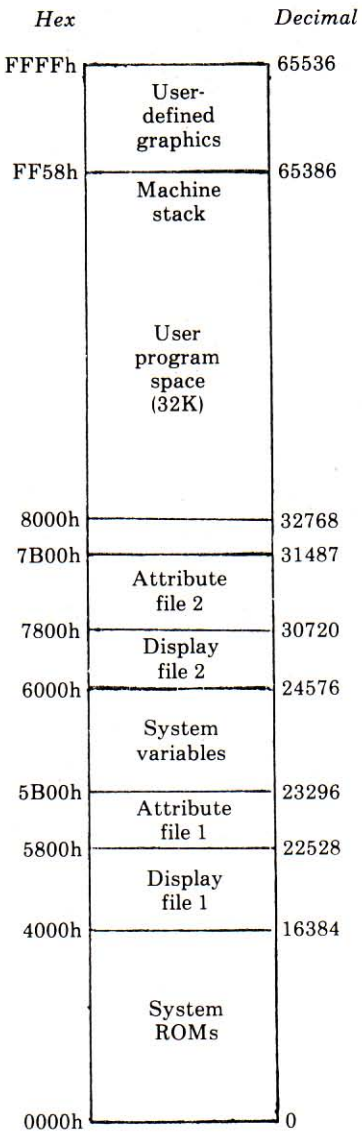
Tech-O
P.O. Box 176
Los Alamitos, CA 90720
213 596-3677

Vernitech
300 Marcus Boulevard
Deer Park, NY 11729
516 586-5100

Harvard Assoc.
260 Beacon Street
Somerville, MA 02143
617 492-0660

Stock Drive Products
55 South Denton Avenue
New Hyde Park, NY 11040
516 328-3330

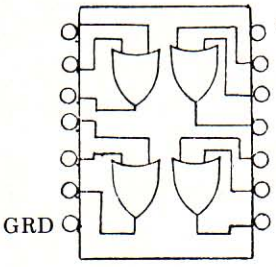
Appendix B: The 2068 Memory Map



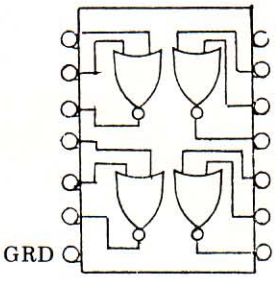
Appendix C: Frequently Used Z80 Instructions

Instruction	*Mnemonic*	*Hex*	*Decimal*	*Comment*
IN DEVICE N	IN A,N	DB	219	Take device N's value to A
OUT DEVICE N	OUT (N),A	D3	211	Register A's contents to location N
LOAD A,N	LD A,N	3E,N	62,N	Load register A with N
LOAD B,N	LD B,N	06,N	6,N	Load register B with N
LOAD C,N	LD C,N	0E,N	14,N	Load register C with N
LOAD (HL),A	LD (HL),A	77	119	Load address in HL with contents of A
LOAD (HL),B	LD (HL),B	70	112	Load address in HL with contents of B
LOAD HL,N	LD HL,N	21,N	33,N	Load register HL with N
LOAD A,(HL)	LD A,(HL)	7E	126	Load A with contents of HL address
LOAD C,B	LD C,B	48	72	Load contents of register B to register C
LOAD (NN),A	LD (NN),A	32	50	Store accumulator at memory NN
HALT	HALT	76	118	Stop CPU
NO OPERATION	NOP	00	0	Idle CPU for one cycle
JUMP IF NOT 0	JP NZ,NN	28,NN	194	Jump up or back if flag 1
JUMP	JP,NN	C3,NN	195	Automatic jump to location NN
JUMP REL IF 0	JR,Z	28,NN	40	Jump up or back if flag 0
RESTART AT 10H	RST,10	D7	215	Jump to location 10 hex
RETURN	RET	C9	201	Return to BASIC
COMPARE L-A	CP L	BD	189	Compare A with L and set flag
INCREMENT A	INC A	3C	60	Add 10 to accumulator
INCREMENT L	INC L	2C	44	Add 1 to register L
DECREMENT A	DEC A	3D	61	Subtract 1 from register A
DECREMENT C	DEC C	0D	13	Subtract 1 from register C
DECREMENT L	DEC L	2D	45	Subtract 1 from register L
ADD HL-BC	ADD HL,BC	09	9	Sum registers HL and BC
ADD A-L	ADD A,L	85	133	Sum registers A and L; sum in A

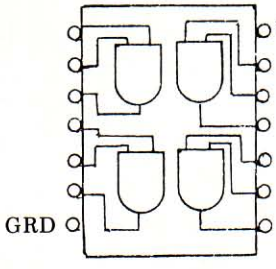
Appendix D: Pinouts of Interfacing Components



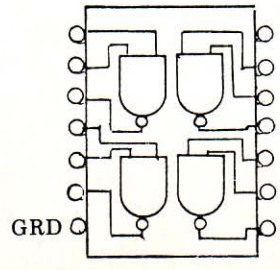
7432
quad
two-input
OR gate
(74C32 CMOS)



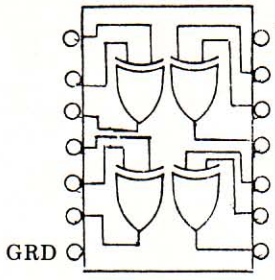
7402
quad
two-input
NOR gate
(4001 CMOS)



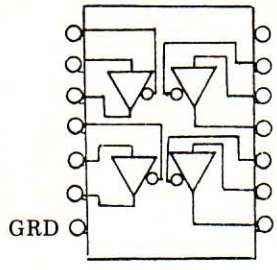
7408
quad
two-input
AND gate
(74C08 CMOS)



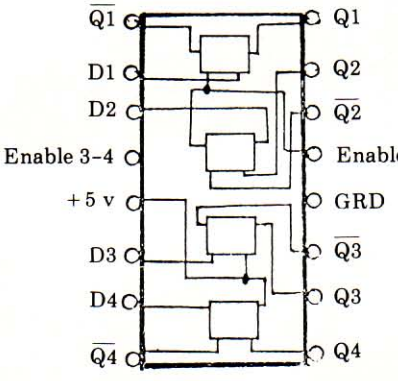
7400
quad
two-input
NAND gate
(4011 CMOS)



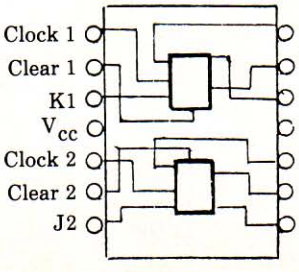
7486
quad
two-input
exclusive
OR gate
(74C86 CMOS)



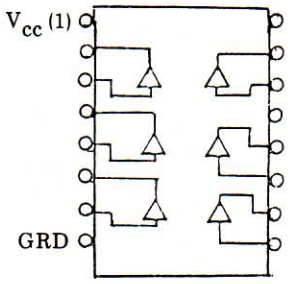
74125
quad
Tri-State
buffer/driver



7475
quad
latch
(4042 CMOS)



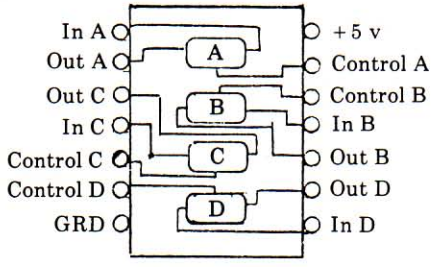
7473
Dual
JK flip-flop
(4027 CMOS)



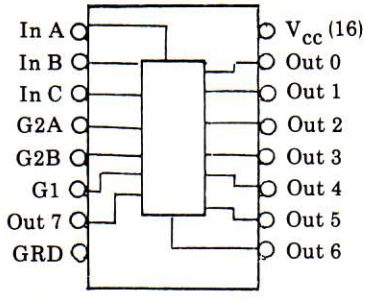
7407
hex
noninverting
buffer
(4050 CMOS)

and

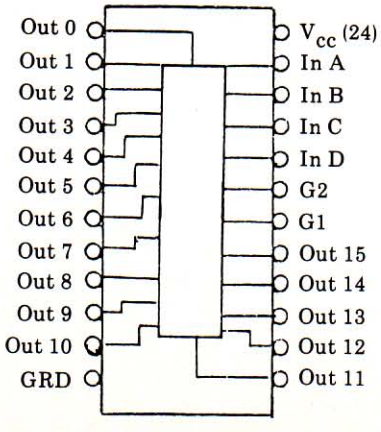
7404
hex
inverter
(4049 CMOS)



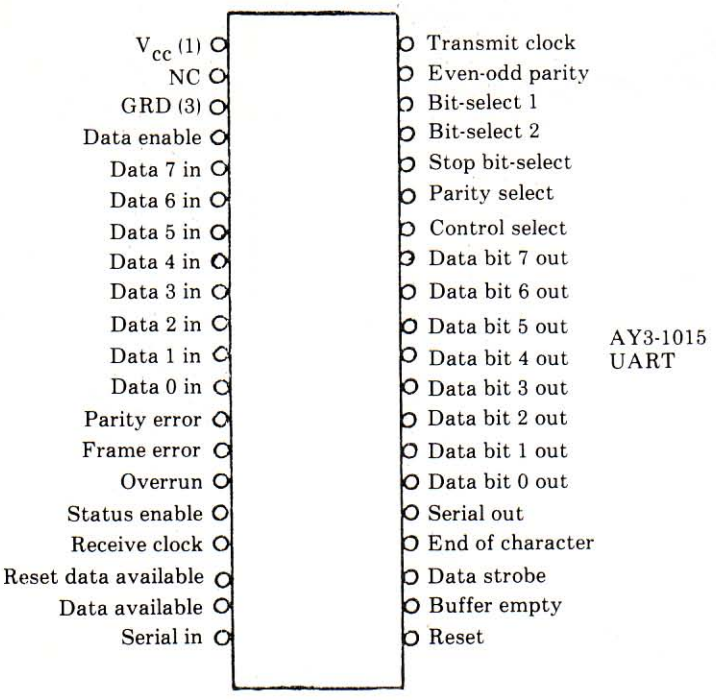
4016 or 4066
CMOS
quad
bilateral
switch

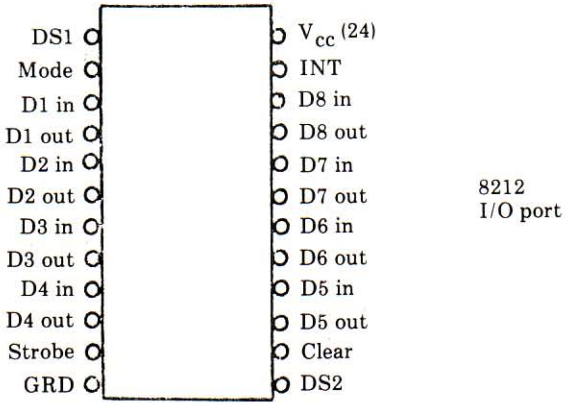


74LS138
one-of-eight
decoder-multiplexer

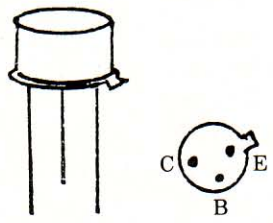


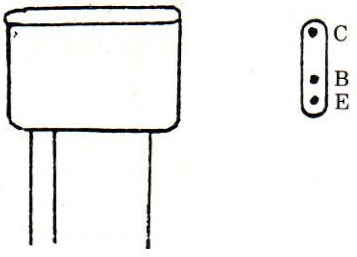
74LS154
four-to-sixteen-line
decoder-multiplexer

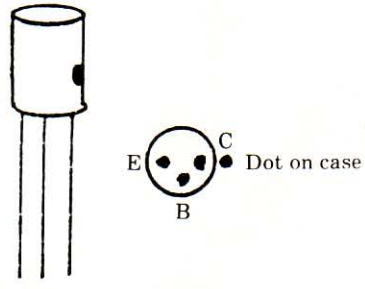

V_{cc} (1)
NC
GRD (3)
Data enable
Data 7 in
Data 6 in
Data 5 in
Data 4 in
Data 3 in
Data 2 in
Data 1 in
Data 0 in
Parity error
Frame error
Overrun
Status enable
Receive clock
Reset data available
Data available
Serial in
Transmit clock
Even-odd parity
Bit-select 1
Bit-select 2
Stop bit-select
Parity select
Control select
Data bit 7 out
Data bit 6 out
Data bit 5 out
Data bit 4 out
Data bit 3 out
Data bit 2 out
Data bit 1 out
Data bit 0 out
Serial out
End of character
Data strobe
Buffer empty
Reset
AY3-1015
UART

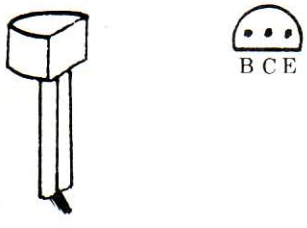

DS1
Mode
D1 in
D1 out
D2 in
D2 out
D3 in
D3 out
D4 in
D4 out
Strobe
GRD
V_{cc} (24)
INT
D8 in
D8 out
D7 in
D7 out
D6 in
D6 out
D5 in
D5 out
Clear
DS2
8212
I/O port

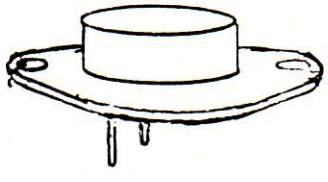
Appendix E: *Transistor Configurations*

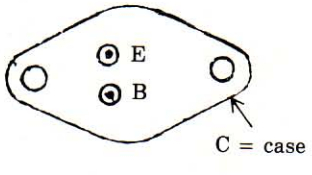




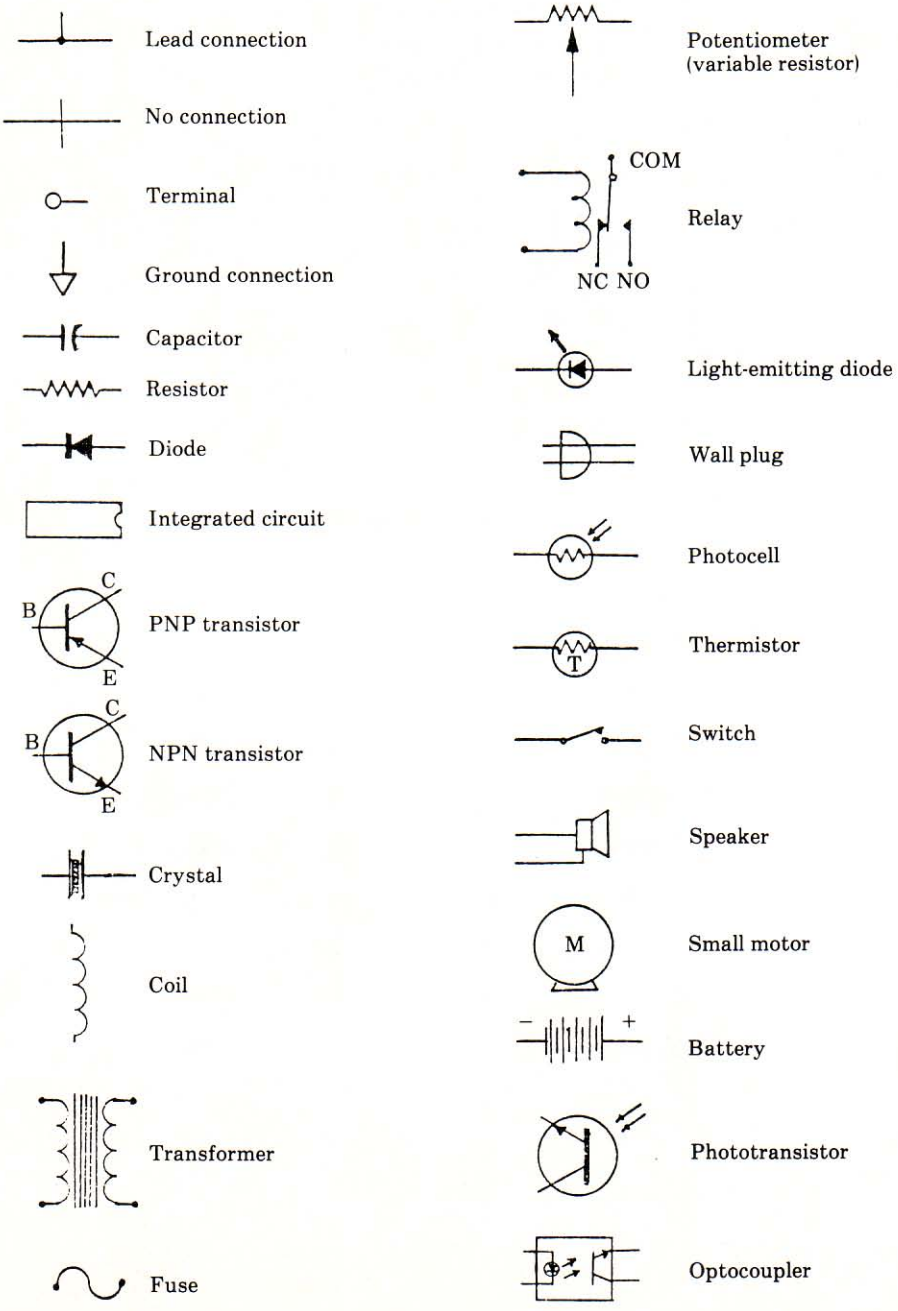




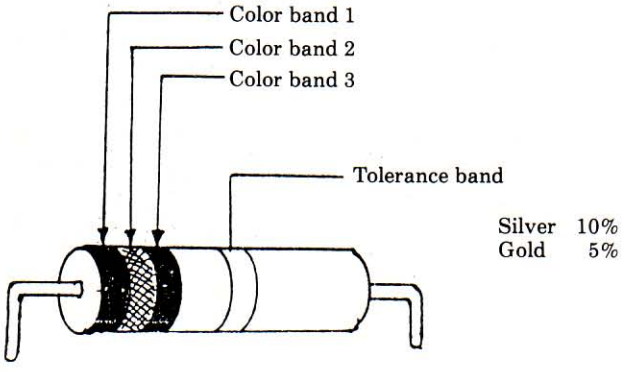




Appendix F: Common Schematic Symbols

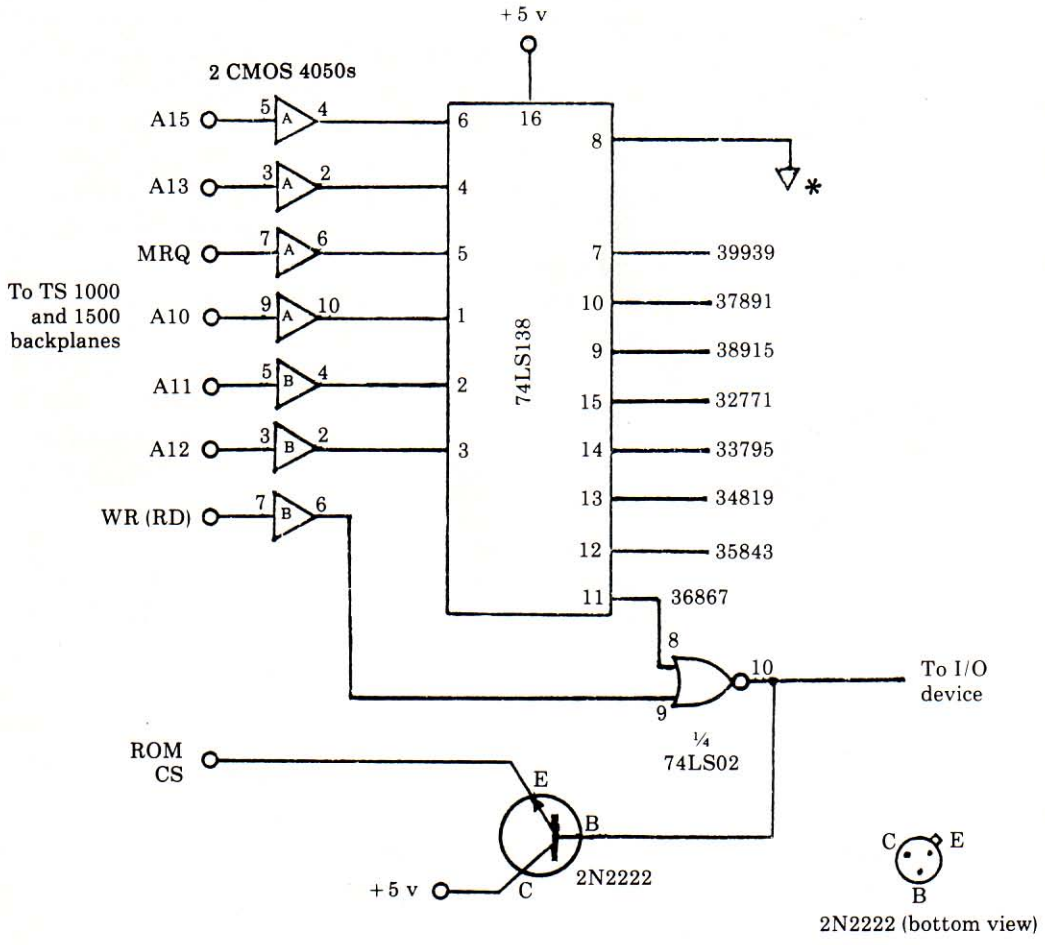


Appendix G: The Resistor Color Code



Color	*Band 1*	*Band 2*	*Multiplier band 3*
Black	0	0	1
Brown	1	1	10
Red	2	2	100
Orange	3	3	1,000
Yellow	4	4	10,000
Green	5	5	100,000
Blue	6	6	1,000,000
Purple	7	7	
Gray	8	8	0.10
White	9	9	0.01

Appendix H: A Device-Select Pulse Decoder for the TS 1000 and 1500



*Ground connectic must go to battery power supply and to computer ground.

Appendix I: Converting 2068 Software for the TS 1000 and 1500

All programs listed in this book will run on both the Timex/Sinclair 1000 and 1500 computers if proper changes are made in the 2068 driver software. The decoder diagram in Appendix H will replace the device-select decoder shown in figure 3.16. Again, the WR line is used to select output devices, and the RD line from the computer is used with the 74LS138 to select input devices. All backplane connections on the 1000 and 1500 computers are basically the same as those used on the 2068, and no modifications are needed to the backplane wiring connector. The user's manuals for the TS 1000 and TS 1500 contain complete diagrams of the layouts of the backplane signals for these two computers.

The 74LS138 decoder (Appendix H) will provide eight different select pulses on its output pins. The device number of each pin is shown in the diagram as a five-digit number. These numbers are the addresses of the device; that is, when a select pulse is required for a device connected to pin 11 of the 138, for example, the OUT command in the 2068 program is simply changed to a PEEK or POKE to address 36867 for reading or writing to the device.

If the 2068 software uses a BEEP or SOUND command, this section of the program is simply deleted or changed to a PRINT statement in the 1000 or 1500 adaptation. Let's look at an example to see how a 2068 program is converted.

A program shown in chapter 3 starts with these two lines:

```
10  LET A = 0
20  OUT 192,A
```

Since this is an output to an output port, we use the POKE command in the conversion. If line 20 had used IN, we would use PEEK. The above two lines would be changed to:

```
10  LET A = 0
20  POKE 36867,A
```

This assumes that the WR line from the computer is connected to the 7402 and that the other line of the 7402 gate is connected to pin 11 of the 74LS138. Up to eight NOR gates may be connected to the decoder for input or output. If the signal to the NOR gate had been connected to the RD line, we would have used the PEEK command to obtain the device-select pulse in the above example.

Glossary

Accumulator The A register, usually used to store, or *accumulate,* sums of numbers.

Adapter A device used to connect two items not originally designed to operate together.

ADC Analog-to-digital converter.

Address The location of a data word in memory.

Analog Varying in degrees, as with resistance and voltage.

ASCII American Standard Code for Information Interchange.

Backplane A peripheral connector for a computer, usually a multifingered edge connector on the computer's back.

Baud Bits per second. A measure of how quickly serial data is transmitted.

Binary A numbering system using only 1s and 0s.

Bit One of the eight binary digits in a data word.

Buffer A device or component to separate and enhance a signal to and from a computer. Also, an area in memory that holds data temporarily.

Bus A group of like signal lines that carry information to and from the CPU.

Byte An 8-bit data word.

Capacitor An electronic component that stores a small electrical charge.

Command A statement in BASIC that elicits some action on the part of the computer. Differs from an instruction.

Console A housing for the main components. In microcomputing, the keyboard housing.

CPU Central processing unit.

Crash A program wipeout due to a mistake in the program or in the incoming data.

Cross talk The radiation of signals into other signal paths, or a signal's hearing, or reading, of adjacent signals.

Decoder An electronic device that outputs a signal based on a combination of other signals.

Device-select A signal that is produced when only a certain I/O number is called by the program.

Digital Two-state numbering, usually in binary form.

Fan-out The number of additional gates that a single gate can drive reliably.

Feedback Data that is returned to the instruction device as a result of an instructed action.

Filter A device (such as a capacitor) that smooths a signal or eliminates noise and voltage spikes.

Firmware Computer programs stored permanently on ROM.

Gate An electronic switch controlling the output of the data and control signals.

Handshaking The synchronized control lines of the computer and input/output devices.

Hardware The electronic circuits that make up the computer system.

I/O Input/output.

Input Data that is fed into the computer system.

Interfacing The proper connection of a device to a computer to achieve a coordinated system.

Instruction In machine language, a data word that results in a controlled action by the CPU.

K 1,000.

LED Light-emitting diode.

Machine language The group of instructions of a CPU that perform various functions of the device.

Memory-mapped I/O Input/output devices that are called and addressed as if they were chunks of memory.

Mnemonic The abbreviation of a machine code instruction.

Output Data leaving the CPU through a peripheral.

Parallel transmission The sending of binary data words in simultaneous 8-bit chunks.

Peripheral Any device that connects to the computer for a specific purpose.

Pinion A small gear that drives larger spur gears.

Pivot The point at which a lever rotates. The pivot of the human lower arm is the elbow.

Port The input or output point at which the computer receives or sends data.

Prototyping Building a project for the first time; assembling an experimental circuit.

RAM Random-access memory.

ROM Read-only memory.

Serial transmission Sending binary data words as a single stream of 1s and 0s that requires only one channel.

Schematic A drawing of an electronic circuit.

Smoke test Initial power turn-on of a device.

System The entire computer, including the display console and its peripherals.

Terminal A remote input/output point for a computer. The keyboard is a terminal for the input of commands.

Thermistor An electronic resistor that changes resistance as temperature changes.

Toggle To switch from one state to another, as with outputs of a flip-flop.

TTL Transistor-transistor logic. Boolean logic produced by the use of transistors.

UART Universal asynchronous receiver-transmitter. Used to send and receive serial data.

Index

> $12.95

POWERFUL PROJECTS WITH YOUR TIMEX/SINCLAIR

"This book includes the nicest synopsis of Z80 assembly language I've ever seen. . . . Plenty of fine examples provide the reader with his money's worth."—Sharon Zardetto Aker, Computer Columnist

Designed for hobbyists and experimenters, novice or experienced, **Powerful Projects with Your Timex/Sinclair** shows you how to build creative electronic projects in your home.

Beginning with the basics, Jim Stephens clearly explains all the wiring techniques and components you'll need to control external devices with your computer. You'll learn how to

- construct simple yet powerful control circuits
- build an interface connector
- create a weather station
- build your own robot
- construct a speech synthesizer for your robot

and more!

Powerful Projects with Your Timex/Sinclair will help you develop a solid understanding of computer electronics while you're building these enjoyable projects. As an added bonus, this how-to book gives you detailed instructions for doing these projects with the TS 2068, 1500, **and** 1000 computers.

If you'd like to do more with your computer than just balance your checkbook, you need this book!

Jim Stephens is an educator and a devoted electronics hobbyist. A resident of Nashville, Tennessee, he has worked with the Tennessee Department of Education for the past ten years. Stephens constructed his own microcomputer in 1978 and has published numerous articles in such popular magazines as **Microcomputing**, **Sync**, **Timex Sinclair User**, **Radio-Electronics**, and **RUN Magazine**.

Scott, Foresman and Company

ISBN 0-673-18038-7